lintrans

by Dyson Dyson

Centre Name: The Duston School
Centre Number: 27220
Candidate Number: 9065

Contents

1   Analysis

One of the topics in the A Level Further Maths course is linear transformations, as represented by matrices. This is a topic all about how vectors move and get transformed in the plane. It's a topic that lends itself exceedingly well to visualization, but students often find it hard to visualize this themselves, and there is a considerable lack of good tools to provide visual intuition on the subject. There is the YouTube series *Essence of Linear Algebra* by 3blue1brown[21], which is excellent, but I couldn't find any good interactive visualizations.

My solution is to develop a desktop application that will allow the user to define 2×2 matrices and view these matrices and compositions thereof as linear transformations of a 2D plane. This will give students a way to get to grips with linear transformations in a more hands-on way, and will give teachers the ability to easily and visually show concepts like the determinant and invariant lines.

1.1   Computational Approach

This solution is particularly well suited to a computational approach since it is entirely focussed on visualizing transformations, which require complex mathematics to properly display. It will also have lots of settings to allow the user to configure aspects of the visualization. As previously mentioned, visualizing transformations in one's own head is difficult, so a piece of software to do it would be very valuable to teachers and learners, but current solutions are considerably lacking.

My solution will make use of abstraction by allowing the user to define a set of matrices which they can use in expressions. This allows them to use a matrix multiple times and they don't have to keep track of any of the numbers. All the actual processing and mathematics happens behind the scenes and the user never has to worry about it - they just compose their defined matrices into transformations. This abstraction allows the user to focus on exploring the transformations themselves without having to do any actual computations. This will make learning the subject much easier, as they will able to gain a visual intuition for linear transformations without worrying about computation until after they've built up that intuition.

I will also employ decomposition and modularization by breaking the project down into many smaller parts, such as one module to keep track of defined matrices, one module to validate and parse matrix expressions, one module for the main GUI, as well as sub-modules for the widgets and dialog boxes, etc. This decomposition allows for simpler project design, easier code maintenance (since module coupling is kept to a minimum, so bugs are isolated in their modules), inheritance of classes to reduce code repetition, and unit testing to inform development. I also intend this unit testing to be automated using GitHub Actions.

Selection will also be used widely in the application. The GUI will provide many settings for visualization, and these settings will need to be checked when rendering the transformation. For example, the user will have the option to render the determinant, so I will need to check this setting on every render cycle and only render the determinant parallelogram if the user has enabled that option. The app will have many options for visualization, which will be useful in learning, but if all these options were being rendered at the same time, then there would be too much information for the user to properly process, so I will let the user configure these display options to their liking and only render the things they want to be rendered.

Validation will also be prevalent because the matrix expressions will need to follow a strict format, which will be validated. The buttons to render and animate the matrix will only be clickable when the given expression is valid, so I will need to check this and update the buttons every time the text in the text box is changed. I will also need to parse matrix expressions so that I can evaluate them properly. All this validation ensures that crashes due to malformed input are practically impossible, and makes the user's life easier since they don't need to worry about if their input is in the right format - the app will tell them.

I will also make use of iteration, primarily in animation. I will have to re-calculate positions and

values to render everything for every frame of the animation and this will likely be done with a simple `for` loop. A `for` loop will allow me to just loop over every frame and use the counter variable as a way to measure how far through the animation we are on each frame. This is preferable to a `while` loop, since that would require me to keep track of which frame we're on with a separate variable.

Finally, the core of the application is visualization, so that will definitely be used a lot. I will have to calculate positions of points and lines based on given matrices, and when animating, I will also have to calculate these matrices based on the current frame. Then I will have to use the rendering capabilities of the GUI framework that I choose to render these calculated points and lines onto a widget, which will form the viewport of the main GUI. I may also have to convert between coordinate systems. I will have the origin in the middle with positive x going to the right and positive y going up, but I may need to convert that to standard computer graphics coordinates with the origin in the top left, positive x going to the right, and positive y going down. This visualization of linear transformations is the core component of the app and is the primary feature, so it is incredibly important.

1.2   Stakeholders

Stakeholders for my app include A Level Further Maths students and teachers, who learn and teach linear transformations respectively. They will be able to provide useful input as to what they would like to see in the app, and they can provide feedback on what they like and what I can add or improve. I already know from experience that linear transformations are tricky to visualize and a computer-based visualization would be useful. My stakeholders agreed with this. Multiple teachers said that a desktop app that could render and animate linear transformations would be useful in a classroom environment and students said that it would be helpful to have something that they could play around with at home and use to get to grips with matrices and linear transformations. They also said that an online version would probably be easier to use, but I have absolutely no experience in web development and I'm much more comfortable making a desktop app.

Some teachers also suggested that it would be useful to have an option to save and load sets of matrices. This would allow them to have a single save file containing some matrices, and then just load this file to use for demonstrations in the classroom. This would probably be quite easy to implement. I could just wrap all the relevant information into one object and use Python's `pickle` module to save the binary data to a file, and then load this data back into the app in a similar way.

My stakeholders agreed that being able to see incremental animation - where, for example, we apply matrix **A** to the current scene, pause, and then apply matrix **B** - would be beneficial. This would be a good demonstration of matrix multiplication being non-commutative. **AB** is not always equal to **BA**. Being able to see this in terms of animating linear transformations would be good for learning.

They also agreed that a tutorial on using the software would be useful, so I plan to implement this through an online written tutorial hosted with GitHub Pages, and perhaps a video tutorial as well. This would make the app much easier to use for people who have never seen it before. It wouldn't be a lesson on the maths itself, but just a guide on how to use the software.

1.3   Research on existing solutions

There are actually quite a few web apps designed to help visualize 2D linear transformations but many of them are hard to use and lacking many features.

1.3.1   MIT 'Matrix Vector' Mathlet

Arguably the best app that I found was an MIT 'Mathlet' - a simple web app designed to help visualize a maths concept. This one is called 'Matrix Vector'[22] and allows the user to drag an input vector

around the plane and see the corresponding output vector, transformed by a matrix that the user can define, although this definition is finicky since it involves sliders rather than keyboard input.

This app fails in two crucial ways in my opinion. It doesn't show the basis vectors or let the user drag them around, and the user can only define and therefore visualize a single matrix at once. This second problem was common among every solution I found, so I won't mention it again, but it is a big issue in my opinion and my app will allow for multiple matrices. I like the idea of having a drag-gable input vector and rendering its output, so I will probably have this feature in my app, but I also want the ability to define multiple matrices and be able to drag the basis vectors to visually define a matrix. Being able to drag the basis vectors will help build intuition, so I think this would greatly benefit the app.
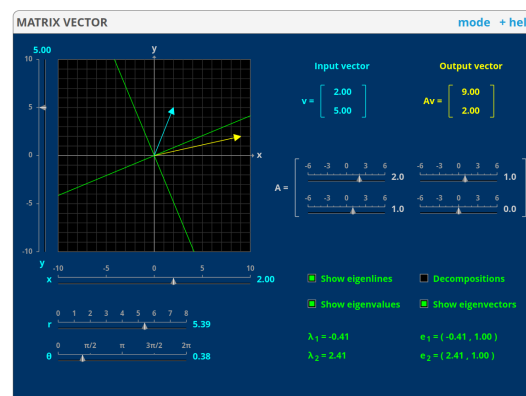
Figure 1.3.1: The MIT 'Matrix Vector' Mathlet

However, in the comments on this Mathlet, a user called 'David S. Bruce' suggested that the Mathlet should display the basis vectors, to which a user called 'hrm' (who I assume to be the 'H. Miller' to whom the copyright of the whole website is accredited) replied saying that this Mathlet is primarily focussed on eigenvectors, that it is perhaps badly named, and that displaying the basis vectors 'would make a good focus for a second Mathlet about 2×2 matrices'. This Mathlet does not exist. But I do like the idea of showing the eigenvectors and eigenlines, so I will definitely have that in my app. Showing the invariant lines or lack thereof will help with learning, since these are often hard to visualize.

1.3.2 Linear Transformation Visualizer
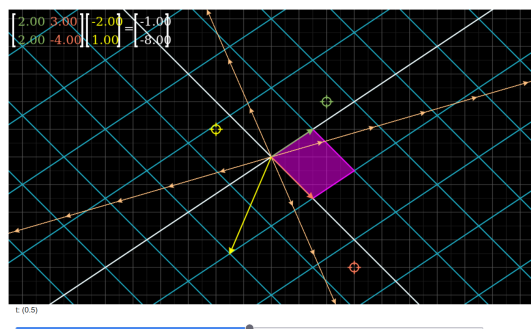
Figure 1.3.2: 'Linear Transformation Visual-izer' halfway through an animation

Another web app that I found was one sim-ply called 'Linear Transformation Visualizer' by Shad Sharma[53]. This one was similarly inspired by 3blue1brown's YouTube series. This app has the ability to render input and output vectors and eigenlines, but it can also render the determinant parallelogram; it allows the user to drag the basis vectors; and it has the option to snap vectors to the background grid, which is quite useful. It also implements a simple form of animation where the tips of the vectors move in straight lines from where they start to where they end, and the animation is controlled by dragging a slider labelled t. This isn't particularly intuitive.

I really like the vectors snapping to the grid, the input and output vectors, and rendering the deter-minant. This app also renders positive and negative determinants in different colours, which is really nice - I intend to use that idea in my own app, since it helps create understanding about negative de-terminants in terms of orientation changes. However, I think that the animation system here is flawed and not very easy to use. My animation will likely be a button, which just triggers an animation, rather than a slider. I also don't like the way vector dragging is handled. If you click anywhere on the grid, then the closest vector target (the final position of the target's associated vector) snaps to that location. I think it would be more intuitive to have to drag the vector from its current location to where you want it. This was also a problem with the MIT Mathlet.

1.3.3 Desmos app

One of the solutions I found was a Desmos app[7], which was quite hard to use and arguably over-complicated. Desmos is not designed for this kind of thing - it's designed to graph pure mathematical functions - and it shows here. However, this app brings some really interesting ideas to the table, mainly functions. This app allows you to define custom functions and view them before and after the transformation. This is achieved by treating the functions parametrically as the set of points $(t, f(t))$ and then transforming each coordinate by the given matrix to get a new coordinate.
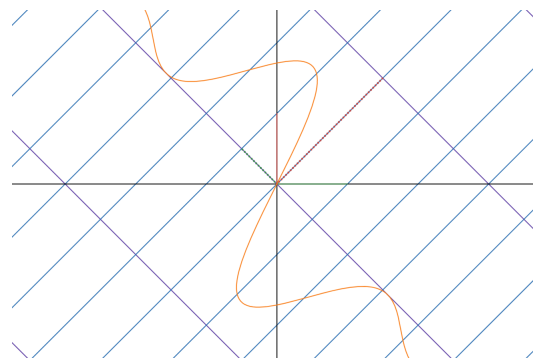
Figure 1.3.3: The Desmos app halfway through an animation, rendering $f(x) = \frac{\sin^2 x}{x}$ in orange

Desmos does this for every point and then renders the resulting transformed function parametrically. This is a really interesting technique and idea, but I'm not going to use it in my app. I don't think arbitrary functions fit with the linearity of the whole app, and I don't think it's necessary. It's just overcomplicating things, and rendering it on a widget would be tricky, because I'd have to render every point myself, possibly using something like OpenGL. It's just not worth implementing.

Additionally, this Desmos app makes things quite hard to see. It's hard to tell where any of the vectors are - they just get lost in the sea of grid lines. This image also hides some of the extra information. For instance, this image doesn't show the original function $f(x) = \frac{\sin^2 x}{x}$, only the transformed version. This app easily gets quite cluttered. I will give my vectors arrowheads to make them easily identifiable amongst the grid lines.

1.3.4 Visualizing Linear Transformations
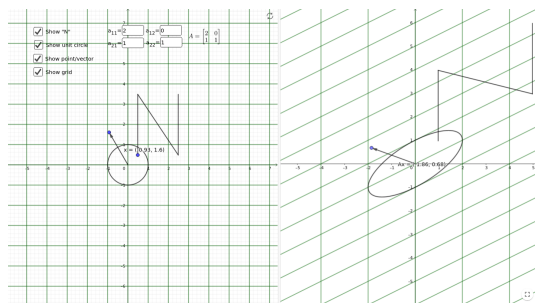
Figure 1.3.4: The GeoGebra applet rendering its default matrix

The last solution that I want to talk about is a GeoGebra applet simply titled 'Visualizing Linear Transformations'[24]. This applet has input and output vectors, original and transformed grid lines, a unit circle, and the letter N. It allows the user to define a matrix as 4 numbers and view the afore-mentioned N (which the user can translate to any-where on the grid), the unit circle, the input/out-put vectors, and the grid lines. It also has the input vector snapping to integer coordinates, but that's a standard part of GeoGebra.

I've already talked about most of these features but the thing I wanted to talk about here is the N. I don't particularly want the letter N to be a prominent part of my own app, but I really like the idea of being able to define a custom polygon and see how that polygon gets transformed by a given transformation. I think that would really help with building intuition and it shouldn't be too hard to implement.

1.4   Essential features

The primary aim of this application is to visualize linear transformations, so this will obviously be the centre of the app and an essential feature. I will have a widget which can render a background grid and a second version of the grid, transformed according to a user-defined matrix expression. This is necessary because it is the entire purpose of the app. It's designed to visualize linear transformations and would be completely useless without this visual component. I will give the user the ability to render a custom matrix expression containing matrices they have previously defined, as well as reset the canvas to the default identity matrix transformation. This will obviously require an input box to enter the expression, a render button, a reset button, and various dialog boxes to define matrices in different ways. I want the user to be able to define a matrix as a set of 4 numbers, and by dragging the basis vectors i and j. These dialogs will allow the user to define new matrices to be used in expressions, and having multiple ways to do it will make it easier, and will aid learning.

Another essential feature is animation. I want the user to be able to smoothly animate between matrices. I see two options for how this could work. If **C** is the matrix for the currently displayed transformation, and **T** is the matrix for the target transformation, then we could either animate from **C** to **T** or we could animate from **C** to **TC**. I would probably call these transitional and applicative animation respectively. Perhaps I'll give the user the option to choose which animation method they want to use. I might even have an option for sequential animation, where the user can define a sequence of matrices, perhaps separated with commas or semicolons, and the app will animate through the sequence, applying one at a time. Sequential animation would be nice, but is not crucial.

Either way, animation is used in most of the alternative solutions that I found, and it's a great way to build intuition, by allowing students to watch the transformation happen in real time. Compared to simply rendering the transformations, animating them would profoundly benefit learning, and since that's the main aim of the project, I think animation is a necessary part of the app.

Something that I thought was a big problem in every alternative solution I found was the fact that the user could only visualize a single matrix at once. I see this as a fatal flaw and I will allow the user to define 25 different matrices (all capital letters except **I** for the identity matrix) and use all of them in expressions. This will allow teachers to define multiple matrices and then just change the expression to demonstrate different concepts rather than redefine a new transformation every time. It will also make things easier for students as it will allow them to visualize compositions of different matrix transformations without having to do any computations themselves.

Additionally, being able to show information on the currently displayed matrix is an essential tool for learning. Rendering things like the determinant parallelogram and the invariant lines of the transformation will greatly assist with learning and building understanding, so I think that having the option to render these attributes of the currently displayed transformation is necessary for success.

1.5   Limitations

The main limitation in this app is likely to be drawing grid lines. Most transformations will be fine but in some cases, the app will be required to draw potentially thousands of grid lines on the canvas and this will probably cause noticeable lag, especially in the animations. I will have to artificially limit the number of grid lines that can be drawn on the screen. This won't look fantastic, because it means that the grid lines will only extend a certain distance from the origin, but it's an inherent limitation of computers. Perhaps if I was using a faster, compiled language like C++ rather than Python, this processing would happen faster and I could render more grid lines, but it's impossible to render all the grid lines and any implementation of this idea must limit them for performance.

An interesting limitation is that I don't think I'll implement panning. I suspect that I'll have to convert between coordinate systems and having the origin in the centre of the canvas will probably make the code much simpler. Also, linear transformations always leave the origin fixed, so always having it in the centre of the canvas seems thematically appropriate. Panning is certainly an option - the Desmos solution in §1.3.3 and GeoGebra solution in §1.3.4 both allow panning as a default part

of Desmos and GeoGebra respectively, for example - but I don't think I'll implement it myself. I just don't think it's worth it.

I'm also not going to do any work with 3D linear transformations. 3D transformations are often harder to visualize and thus it would make sense to target them in an app like this, designed to help with learning and intuition, but 3D transformations are also harder to code. I would have to use a full graphics package rather than a simple widget, and I think it would be too much work for this project and I wouldn't be able to do it in the time frame. It's definitely a good idea, but I'm currently incapable of creating an app like that.

There are other limitations inherent to matrices. For instance, it's impossible to take an inverse of a singular matrix. There's nothing I can do about that without rewriting most of mathematics. Matrices can also only represent linear transformations. There's definitely a market for an app that could render any arbitrary transformation from $\mathbb{R}^2 \to \mathbb{R}^2$ - I know I'd want an app like that - but matrices can only represent linear transformations, so those are the only kind of transformations that I'll be looking at with this project.

1.6   Hardware and software requirements

1.6.1   Hardware

Hardware requirements for the project are the same between the release and development environments and they're quite simple. I expect the app to require a processor with at least 1 GHz clock speed, roughly 100MB free disk space, and about 1 GB of available RAM. The processor and RAM requirements are needed by the Python runtime and mainly by Qt5 - the GUI library I'll be using. The 100MB of disk space is just for the executable binary that I'll compile for the public release. The code itself is less than 1 MB, but the compiled binary has to package all the dependencies and the entire CPython runtime to allow it to run on systems that don't have that, so the file size is much bigger.

I will also require that the user has a monitor that is at least 1920×1080 pixels in resolution. This isn't necessarily required, because the app will likely run in a smaller window, but a HD monitor is highly recommended. This allows the user to go fullscreen if they want to, and it gives them enough resolution to easily see everything in the app. A large, wall-mounted screen is also highly recommended for use in the classroom, although this is common among schools.

I will also require a keyboard with all standard Latin alphabet characters. This is because the matrices are defined as uppercase Latin letters. Any UK or US keyboard will suffice for this. The app will also require a mouse with at least one button. I don't intend to have right click do anything, so only the primary mouse button is required, although getting a single button mouse to actually work on modern computers is probably quite a challenge. A separate mouse is not strictly required - a laptop trackpad is equally sufficient.

1.6.2   Software

Software requirements differ slightly between release and development, although everything that the release environment requires is also required by the development environment. I will require a modern operating system - namely Windows 10 or later, macOS 10.9 'Mavericks'[1] or later, or any modern Linux distro[2]. Basically, it just requires an operating system that is compatible with Python 3.8 or higher as well as Qt5, since I'll be using these in the project. Of course, Qt5 will need to be installed on the user's computer, although it's standard pretty much everywhere these days.

---

[1]Python 3.8 or higher won't compile on any earlier versions of macOS[40]

[2]Specifying a Linux version is practically impossible. Python 3.8 or higher is available in many package repositories, but all modern Python versions will compile on any modern distro. Qt5 is available in many package repositories and can be compiled on any `x86` or `x86_64` generic Linux machine with `gcc` version 5 or later[43]

Python won't actually be required for the end user, because I will be compiling the app into a stand-alone binary executable for release, and this binary will contain the required Python runtime and dependencies. However, if the user wishes to download and run the source code themself, then they will need Python 3.8 or higher and the package dependencies: `numpy`, `nptyping`, and `pyqt5`. These can be automatically installed with the command `python -m pip install -r requirements.txt` from the root of the repository, although the whole project will be an installable Python package, so using `pip install -e .` will be preferred.

`numpy` is a maths library that allows for fast matrix maths; `nptyping` is used by `mypy` for type-checking and isn't actually a runtime dependency but the imports in the `typing` module fail if it's not installed at runtime[3]; and `pyqt5` is a library that just allows interop between Python and Qt5, which is originally a C++ library.

In the development environment, I use PyCharm for actually writing my code, and I use a virtual environment to isolate my project dependencies. There are also some development dependencies listed in the file `dev_requirements.txt`. They are: `mypy`, `pyqt5-stubs`, `flake8`, `pycodestyle`, `pydocstyle`, and `pytest`. `mypy` is a static type checker[4]; `pyqt5-stubs` is a collection of type annotations for the PyQt5 API for `mypy` to use; `flake8`, `pycodestyle`, and `pydocstyle` are all linters; and `pytest` is a unit testing framework. I use these libraries to make sure my code is good quality and actually working properly during development.

1.7   Success criteria

The main aim of the app is to help teach students about linear transformations. As such, the primary measure of success will be letting teachers get to grips with the app and then asking if they would use it in the classroom or recommend it to students to use at home.

Additionally, the app must fulfil some basic requirements:

1. It must allow the user to define multiple matrices in at least two different ways (numerically and visually)

2. It must be able to validate arbitrary matrix expressions

3. It must be able to parse any valid matrix expression

4. It must be able to render any valid matrix expression

5. It must be able to animate from $\mathbf{I}$ to any valid matrix expression

6. It must be able to apply a matrix expression to the current scene and animate this (animate from $\mathbf{C}$ to $\mathbf{TC}$, and perhaps do sequential animation)

7. It must be able to display information about the currently rendered transformation (determinant, eigenlines, etc.)

8. It must be able to save and load sessions (defined matrices, display settings, etc.)

9. It must allow the user to define and transform arbitrary polygons

Defining multiple matrices is a feature that I thought was lacking from every other solution I researched, and I think it would make the app much easier to use, so I think it's necessary for success. Validating matrix expressions is necessary because if the user tries to render an expression that doesn't make sense, has an undefined matrix, or contains the inverse of a singular matrix, then we have to disallow that or else the app will crash.

Visualizing matrix expressions as linear transformations is the core part of the app, so basic rendering of them is definitely a requirement for success. Animating these expressions is also a pretty crucial

---

[3]These `nptyping` imports are needed for type annotations all over the code base, so factoring them out is not feasible

[4]Python has weak, dynamic typing with optional type annotations but `mypy` enforces these static type annotations

part of the app, so I would consider this necessary for success. Displaying the information of a matrix transformation is also very useful for building understanding, so I would consider this needed to succeed.

Saving and loading isn't strictly necessary for success, but it is a standard part of many apps, so will likely be expected by users, and it will benefit the app by allowing teachers to plan lessons in advance and save the matrices they've defined for that lesson to be loaded later.

Transforming polygons is the lowest priority item on this list and will likely be implemented last, but it would definitely benefit learning. I wouldn't consider it necessary for success, but it would be very good to include, and it's certainly a feature that I want to have.

If the majority of teachers would use and/or recommend the app and it meets all of these points, then I will consider the app as a whole to be a success.

2   Design

2.1   Problem decomposition

I have decomposed the problem of visualization as follows:
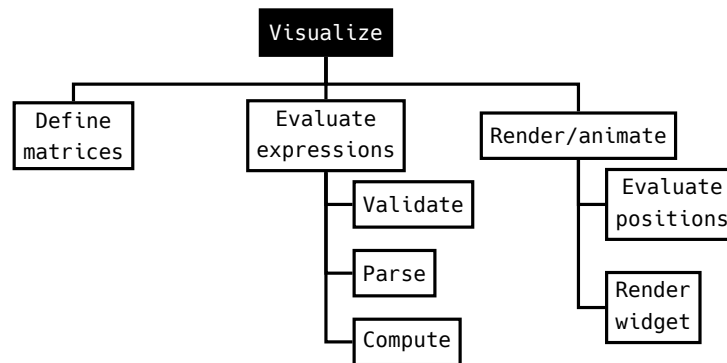
Defining matrices is key to visualization because we need to have matrices to actually visualize. This is a key part of the app, and the user will be able to define multiple separate matrices numerically and visually using the GUI.

Evaluating expressions is another key part of the app and can be further broken down into validating, parsing, and computing the value. Validating an expression simply consists of checking that it adheres to a set of syntax rules for matrix expressions, and that it only contains matrices which have already been defined. Parsing consists of breaking an expression down into tokens, which are then much easier to evaluate. Computing the expression with these tokens is then just a series of simple operations, which will produce a final matrix at the end.

Rendering and animating will likely be the largest part in reality, but I've only decomposed it into simple blocks here. Evaluating positions involves evaluating the matrix expression that the user has input and using the columns of the resultant matrix to find the new positions of the basis vectors, and then extrapolating this for the rest of the plane. Rendering onto the widget is likely to be quite complicated and framework-dependent, so I've abstracted away the details for brevity here. Rendering will involve using the previously calculated values to render grid lines and vectors. Animating will probably be a `for` loop which just renders slightly different matrices onto the widget and sleeps momentarily between frames.

I have deliberately broken this problem down into parts that can be easily translated into modules in my eventual coded solution. This is simply to ease the design and development process, since now I already know my basic project structure. This problem could've been broken down into the parts that the user will directly interact with, but that would be less useful to me when actually starting development, since I would then have to decompose the problem differently to write the actual code.

2.2   Structure of the solution

I have decomposed my solution like so:

```
lintrans
├── __main__
├── global_settings
├── gui
│   ├── main_window
│   ├── session
│   ├── settings
│   ├── dialogs
│   │   ├── define_new_matrix
│   │   ├── misc
│   │   └── settings
│   └── plots
│       ├── classes
│       └── widgets
└── matrices
    ├── parse
    ├── utility
    └── wrapper
```
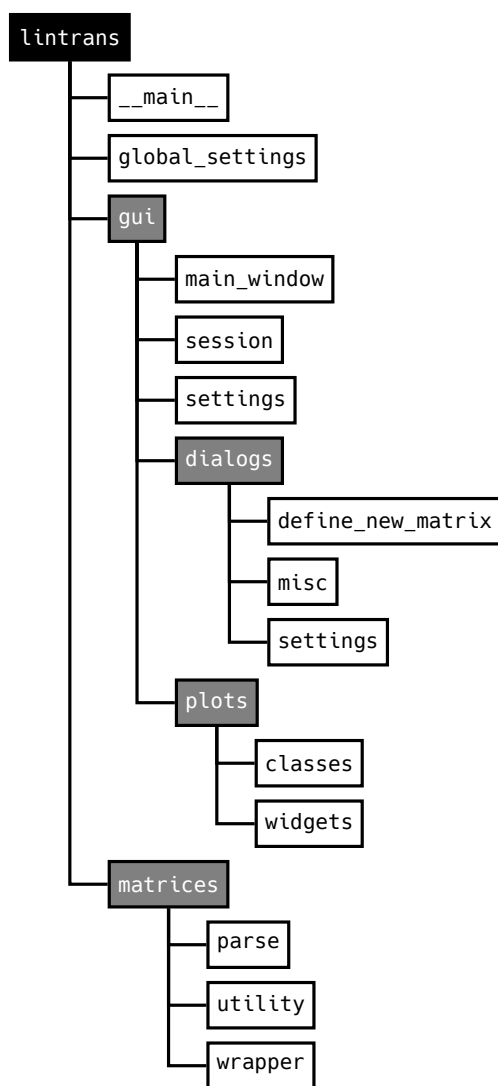
The `lintrans` node is simply the root of the whole project. `__main__` is the Python way to make the project executable as `python -m lintrans` on the command line. For release, I will package it into a standalone binary executable, using this module as the entry point.

The `global_settings` module will define a `GlobalSettings` singleton class. This class will manage global settings and variables - things like where to save sessions by default, etc. I'm not entirely sure what I want to put in here, but I expect that I'll want global settings in the future. Having this class will allow me to easily read and write these settings to a file to have them persist between sessions.

`matrices` is the package that will allow the user to define, validate, parse, evaluate, and use matrices. The `matrices.parse` module will contain functions to validate matrix expressions - likely using regular expressions - and functions to parse matrix expressions. It will not know which matrices are defined, so validation will be naïve and evaluation will be in the `matrices.wrapper` module. This `wrapper` module will contain a `MatrixWrapper` class, which will hold a dictionary of matrix names and values. It is this class which will have aware validation - making sure that all the matrices used in an expression are actually defined in the wrapper - as well the ability to evaluate matrix expressions, in addition to its basic behaviour of setting and getting matrices by name. There will also be a `matrices.utility` module, which will contain some simple functions for simple functionality. Functions like `create_rotation_matrix()`, which will generate a rotation matrix from an angle using the formula $\left(\begin{smallmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{smallmatrix} \right)$.

`gui` is the package that will contain all the frontend code for everything GUI-related. `gui.main_window` is the module that will define the `LintransMainWindow` class, which will act as the main window of the application and have an instance of `MatrixWrapper` to keep track of which matrices are defined and allow for evaluation of matrix expressions. It will also have methods for rendering and animating matrix expressions, which will be connected to buttons in the GUI. The most important part of the main window is the viewport, which will be discussed shortly. This module will also contain a simple `main()` function to instantiate and launch the application GUI.

The `gui.session` module will contain functions to save and load a session from a file. A session will consist of the `MatrixWrapper`, along with perhaps the display settings and maybe some other things. I know that saving the wrapper will be essential, but I'll see what else should be saved as the project evolves.

The `gui.settings` module will contain a `DisplaySettings` dataclass[5] that will represent the settings for visualizing transformations. The viewport class will have an instance of this class and check against it when rendering things. The user will be able to open a dialog to change these display settings, which will update the main window's instance of this class.

The `gui.dialogs` subpackage will contain modules with different dialog classes. It will have a `gui.dialogs.define_new_matrices` module, which will have a `DefineDialog` abstract superclass. It will then contain classes that inherit from this superclass and provide dialogs for defining new matrices visually,

---

[5]This is the Python equivalent of a `struct` or `record` in other languages

numerically, and as an expression in terms of other matrices. Additionally, it will contain a `gui.dialogs.settings` module, which will provide a `SettingsDialog` superclass and a `DisplaySettingsDialog` class, which will allow the user to configure the aforementioned display settings. It may also have a `GlobalSettingsDialog` class in the future, which would similarly allow the user to configure the app's global settings through a dialog. This will only be implemented once I've actually got global settings to configure.

The `gui.dialogs.misc` module will contain small miscellaneous dialog boxes - things like the about box which are very simple and don't need a dedicated module.

The `gui.plots` subpackage will have a `gui.plots.classes` module and a `gui.plots.widgets` module. The `classes` module will have the abstract superclasses `BackgroundPlot` and `VectorGridPlot`. The former will provide helper methods to convert between coordinate systems and draw the background grid, while the latter will provide helper methods to draw transformations and their components. It will have `point_i` and `point_j` attributes and will provide methods to draw the transformed version of the grid, the vectors and their arrowheads, the eigenlines of the transformation, etc. These methods can then be called from the Qt5 `paintEvent` handler which will be declared abstract and must therefore be implemented by all subclasses.

The `gui.plots.widgets` module will have the classes `VisualizeTransformationWidget` and `DefineVisuallyWidget`, which will both inherit from `VectorGridPlot`. They will both implement their own `paintEvent` handler to actually draw the respective widgets, and `DefineVisuallyWidget` will also implement handlers for mouse events, allowing the user to drag around the basis vectors.

I also want the user to be able to define arbitrary polygons and view their transformations. I imagine this polygon definition will happen in a separate dialog, but I don't know where that's going to fit just yet. I'll probably have the widget in `gui.plots.widgets`, but possibly elsewhere.

2.3 Algorithm design

The project will have many algorithms and a lot of them will be related to drawing transformations on the canvas itself, but almost all of the algorithms will evolve over time. In this section, I will present pseudocode for some of the most interesting parts of the project. The real implementations may be different from these initial plans.

The `lintrans.matrices.utility` module will look roughly like this:

```
1    import numpy as np // Python import
2
3    // Create a matrix representing a rotation (anticlockwise) by the given angle
4    function create_rotation_matrix(angle: float, degrees: bool = True) -> MatrixType
5        if (degrees) then
6            rad = np.deg2rad(angle MOD 360)
7        else
8            rad = angle MOD (2 * np.pi)
9        endif
10
11       return np.array([
12           [np.cos(rad), -1 * np.sin(rad)],
13           [np.sin(rad), np.cos(rad)]
14       ])
15   endfunction
```

And the `lintrans.matrices.wrapper` module will look like this:

```
1    import re
2    import numpy as np
3
4    // The `.utility` syntax means that the utility module is next to this one in the tree
5    from .utility import create_rotation_matrix
```

```
6
7    class MatrixWrapper
8        // This is a hashmap from string to matrices, but the matrices might be null
9        private matrices: Dict[string, Optional[MatrixType]]
10
11       public procedure new()
12           matrices = {
13               "A": null, "B": null, "C": null, "D": null,
14               "E": null, "F": null, "G": null, "H": null,
15               "I": np.eye(2),  // I is always defined as the identity matrix
16               "J": null, "K": null, "L": null, "M": null,
17               "N": null, "O": null, "P": null, "Q": null,
18               "R": null, "S": null, "T": null, "U": null,
19               "V": null, "W": null, "X": null, "Y": null,
20               "Z": null
21           }
22       endprocedure
23
24       // This is a Python "magic method", which enable syntax like `wrapper['A']` to get the matrix A
25       public function getitem(name: string) -> Optional[MatrixType]
26           // If it is a simple name, it will just be fetched from the dictionary. If the name is ``rot(x)``, with
27           // a given angle in degrees, then we return a new matrix representing a rotation by that angle
28
29           // Return a new rotation matrix
30           match = re.match(r"^rot\((-?\d*\.?\d*)\)$", name)
31           if (match != null) then
32               return create_rotation_matrix(float(match.group(1)))
33           endif
34
35           if (!matrices.contains(name)) then
36               raise NameError(f'Unrecognised matrix name "{name}"')
37           endif
38
39           return matrices[name]
40       endfunction
41
42       // Again, this is Python magic. This one allows assignments like `wrapper['A'] = my_matrix`
43       public procedure setitem(name: string, new_matrix: Optional[MatrixType])
44           // If new_matrix is null, then that effectively unsets the matrix name.
45
46           if (name == "I" OR !matrices.contains(name)) then
47               raise NameError("Matrix name is illegal")
48           endif
49
50           if (new_matrix == null) then
51               matrices[name] = null
52               return
53           endif
54
55           if (!is_matrix_type(new_matrix)) then
56               raise TypeError("Matrix must be a 2x2 NumPy array")
57           endif
58
59           // All matrices must have float entries
60           a = float(new_matrix[0][0])
61           b = float(new_matrix[0][1])
62           c = float(new_matrix[1][0])
63           d = float(new_matrix[1][1])
64
65           matrices[name] = np.array([[a, b], [c, d]])
66       endprocedure
67   endclass
```

The `lintrans.gui.plots.classes` module will contain the following utility functions:

```
1    from PyQt5.QtGui import QPainter
2    from PyQt5.QtWidgets import QWidget
3
4    const DEFAULT_GRID_SPACING: int = 85
5
6    // This class will act as a baseclass for the viewport and visual defintion dialog.
```

```
7      // They will both use different subclasses of this class to visualize things on a grid.
8      // Canvas coordinates are the coordinates that Qt5 uses internally. These are standard
9      // computer graphics coordinates, with (0, 0) in the top left, positive x to the right,
10     // and positive y going down.
11     class BackgroundPlot extends QWidget
12         private grid_spacing: int
13
14         public procedure new()
15             super.new()
16
17             grid_spacing = DEFAULT_GRID_SPACING
18         endprocedure
19
20         // Return the canvas coordinates of the grid origin (centre)
21         private function canvas_origin() -> (int, int):
22             // width() and height() come from QWidget. They get the total width or height of the widget
23             return (width() DIV 2, height() DIV 2)
24         endfunction
25
26         // Convert an x coordinate from grid coords to canvas coords so it can be drawn
27         private function canvas_x(self, x: float) -> int
28             return int(canvas_origin()[0] + x * grid_spacing)
29         endfunction
30
31         // Convert an x coordinate from grid coords to canvas coords so it can be drawn
32         private function canvas_y(self, y: float) -> int
33             return int(canvas_origin()[1] - y * grid_spacing)
34         endfunction
35
36         // Convert a coordinate from grid coords to canvas coords
37         private function canvas_coords(x: float, y: float) -> (int, int)
38             return (canvas_x(x), canvas_y(y))
39         endfunction
40
41         // Find the grid coordinates of the top right corner. We use the top right because
42         // both coordinates will be positive
43         private function grid_corner() -> (float, float)
44             // Again, width() and height() come from QWidget
45             return (width() / (2 * grid_spacing), height() / (2 * grid_spacing))
46         endfunction
47
48         // Convert a coordinate from canvas coords to grid coords
49         private function grid_coords(x: int, y: int) -> (float, float)
50             return (
51                 (x - canvas_origin[0]) / grid_spacing,
52                 (-y + canvas_origin[1]) / grid_spacing
53             )
54         endfunction
55
56         // Draw the background grid. This method is meant to be used by subclasses when
57         // they paint their whole scene
58         private function draw_background(painter: QPainter)
59             // Draw equally spaced vertical lines, starting in the middle and going out
60             // We loop uo to half the width because we draw one line on each side of the axis per iteration
61             for (int x = width() DIV 2 + grid_spacing; x < width(); x += grid_spacing)
62                 painter.drawLine((x, 0), (x, height()))
63                 painter.drawLine((width() - x, 0), (width() - x, height()))
64             next x
65
66             // Now do the same for horizontal lines
67             for (int y = height() DIV 2 + grid_spacing; y < height(); x += grid_spacing)
68                 painter.drawLine((0, y), (width(), y))
69                 painter.drawLine((0, height() - y), (width(), height() - y))
70             next y
71
72             // Now draw the axes
73             painter.drawLine((width() DIV 2, 0), (width() DIV 2, height()))
74             painter.drawLine((0, height() DIV 2), (width(), height() DIV 2))
75         endfunction
76     endclass
```

These modules handle the creation, storage, and use of matrices. Their implementations are deliber-

ately simple, since they don't have to do much. I will eventually extend the `MatrixWrapper` class to allow strings as matrices, so they can be defined as expressions, but this is unnecessary for now. It will simply be more conditions in `__getitem__` and `__setitem__` and a method to evaluate expressions.

Parsing matrix expressions will be quite tricky and I don't really know how I'm going to do it. I think it will be possible with regular expressions, since I won't support nested expressions at first. But adding support for nested expressions may require something more complicated. I will have a function to validate a matrix expression, which can definitely be done with regular expressions, and I'll have another public function to parse matrix expressions, although this one may use some private functions to implement it properly.

I'm not sure on any algorithms yet, but here's the full BNF specification for matrix expressions (including nested expressions):

```
1    expression        ::=  [ "-" ] matrices { ( "+" | "-" ) matrices };
2    matrices          ::=  matrix { matrix };
3    matrix            ::=  [ real_number ] matrix_identifier [ index ] | "(" expression ")";
4    matrix_identifier ::=  "A" .. "Z" | "rot(" [ "-" ] real_number ")";
5    index             ::=  "^{" index_content "}" | "^" index_content;
6    index_content     ::=  [ "-" ] integer_not_zero | "T";
7
8    digit_no_zero     ::=  "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
9    digit             ::=  "0" | digit_no_zero;
10   digits            ::=  digit | digits digit;
11   integer_not_zero  ::=  digit_no_zero [ digits ];
12   real_number       ::=  ( integer_not_zero [ "." digits ] | "0" "." digits );
```

Obviously the data structure returned by the parser is very important. I have decided to use `list[list[tuple[str, str, str]]]`. Each tuple represents a real number multiplier, a matrix identifier, and an index. The multiplier and index may be empty strings. These tuples are contained in lists which represent matrices to be multiplied together, in order. Those lists are contained in a top level list, which represents multiplication groups which should be summed.

This type makes the structure of the input clear, and makes it very easy for the `MatrixWrapper` to evaluate a parsed expression.

2.4   Usability features

My main concern in terms of usability is colour. In the 3blue1brown videos on linear algebra, red and green are used for the basis vectors, but these colours are often hard to distinguish in most common forms of colour blindness. The most common form is deuteranopia[58], which makes red and green look incredibly similar. I will use blue and red for my basis vectors. These colours are easy to distinguish for people with deuteranopia and protanopia - the two most common forms of colour blindness. Tritanopia makes it harder to distinguish blue and yellow, but my colour scheme is still be accessible for people with tritanopia, as red and blue are very distinct in this form of colour blindness.

I will probably use green for the eigenvectors and eigenlines, which will be hard to distinguish from the red basis vector for people with red-green colour blindness, but I think that the basis vectors and eigenvectors/eigenlines will look physically different enough from each other that the colour shouldn't be too much of a problem. Additionally, I will use a tool called Color Oracle[28] to make sure that my app is accessible to people with different forms of colour blindness[6].

Another solution would be to have one default colour scheme, and allow the user to change the colour scheme to something more accessible for colour blind people, but I don't see the point in this. I think it's easier for colour blind people to just have the main colour scheme be accessible, and it's not really an inconvenience to non-colour blind people, so I think this is the best option.

---

[6]I actually had to clone a fork of this project[1] to get it working on Ubuntu 20.04 and adapt it slightly to create a working jar file[2]

The layout of my app will be self-consistent and follow standard conventions. I will have a menu bar at the top of the main window for actions like saving and loading, as well as accessing the tutorial (which will also be accessible by pressing `F1` at any point) and documentation. The dialogs will always have the confirm button in the bottom right and the cancel button just to the left of that. They will also have the matrix name drop-down on the left. This consistency will make the app easier to learn and understand.

I will also have hotkeys for everything that can have hotkeys - buttons, checkboxes, etc. This makes my life easier, since I'm used to having hotkeys for everything, and thus makes the app faster to test because I don't need to click everything. This also makes things easier for other people like me, who prefer to stay at the keyboard and not use the mouse. Obviously a mouse will be required for things like dragging basis vectors and polygon vertices, but hotkeys will be available wherever possible to help people who don't like using the mouse or find it difficult.

2.5   Variables and validation

The most important variables in the project will be instance attributes on the `LintransMainWindow` class. It will have a `MatrixWrapper` instance, a `DisplaySettings` instance, and most importantly, a `VisualizeTransformationWidget` instance. These will handle the matrices and various settings respectively. Having these as instance attributes allows them to be referenced from any method in the class, and Qt5 uses lots of slots (basically callback methods) and handlers, so it's good to be able to access the attributes I need right there rather than having to pass them around from method to method.

The `MatrixWrapper` class will have a dictionary of names and matrices. The names will be single letters[7] and the matrices will be of type `MatrixType`. This will be a custom type alias representing a 2×2 `numpy` array of floats. When setting the values for these matrices, I will have to manually check the types. This is because Python has weak typing, and if we got, say, an integer in place of a matrix, then operations would fail when trying to evaluate a matrix expression, and the program would crash. To prevent this, we have to validate the type of every matrix when it's set. I have chosen to use a dictionary here because it makes accessing a matrix by its name easier. We don't have to check against a list of letters and another list of matrices, we just index into the dictionary.

The settings dataclasses will have instance attributes for each setting. Most of these will be booleans, since they will be simple binary options like *Show determinant*, which will be represented with checkboxes in the GUI. The `DisplaySettings` dataclass will also have an attribute of type `int` representing the time in milliseconds to pause during animations.

The `DefineDialog` superclass have a `MatrixWrapper` instance attribute, which will be a parameter in the constructor. When `LintransMainWindow` spawns a definition dialog (which subclasses `DefineDialog`), it will pass in a copy of its own `MatrixWrapper` and connect the `accepted` signal for the dialog. The slot (method) that this signal is connected to will get called when the dialog is closed with the *Confirm* button[8]. This allows the dialog to mutate its own `MatrixWrapper` object and then the main window can copy that mutated version back into its own instance attribute when the user confirms the change. This reduces coupling and makes everything easier to reason about and debug, as well as reducing the number of bugs, since the classes will be independent of each other. In another language, I could pass a pointer to the wrapper and let the dialog mutate it directly, but this is potentially dangerous, and Python doesn't have pointers anyway.

Validation will also play a very big role in the application. The user will be able to enter matrix expressions and these must be validated. I will define a BNF schema and either write my own RegEx or use that BNF to programmatically generate a RegEx. Every matrix expression input will be checked against it. This is to ensure that the matrix wrapper can actually evaluate the expression. If we didn't validate the expression, then the parsing would fail and the program could crash. I've chosen to use a RegEx here rather than any other option because it's the simplest. Creating a RegEx

---

[7] I would make these `char` but Python only has a `str` type for strings

[8] Actually when the dialog calls `.accept()`. The `Confirm` button is actually connected to a method which first takes the info and updates the instance `MatrixWrapper`, and then calls `.accept()`

can be difficult, especially for complicated patterns, but it's then easier to use it. Also, Python can compile a RegEx pattern, which makes it much faster to match against, so I will compile the pattern at initialization time and just compare expressions against that pre-compiled pattern, since we know it won't change at runtime.

Additionally, the buttons to render and animate the current matrix expression will only be enabled when the expression is valid. Textboxes in Qt5 emit a `textChanged` signal, which can be connected to a slot. This is just a method that gets called whenever the text in the textbox is changed, so I can use this method to validate the input and update the buttons accordingly. An empty string will count as invalid, so the buttons will be disabled when the box is empty.

I will also apply this matrix expression validation to the textbox in the dialog which allows the user to define a matrix as an expression involving other matrices, and I will validate the input in the numeric definition dialog to make sure that all the inputs are floats. Again, this is to prevent crashes, since a matrix with non-number values in it will likely crash the program.

2.6 Iterative test data

There is a Python library called `pytest`, which can be used to run unit tests[25]. Unit tests are automatic tests that I can run whenever I like[59]. I can also set these up to be run automatically by using GitHub Actions[17]. Doing this will make it easy to test my code, because I can write separate testing code which makes sure that all my backend functions work as intended. This testing code is easy to run, and can be run automatically after every change, so I know that my changes won't break previous behaviour. All the unit testing code is in appendix B.

Additionally, `pytest` supports doctests[26], which are unit tests written in the source code documentation[39]. These tests are typically small, simple examples used to demonstrate how to use a function to an end user of the API (in this case, me). It is important to tests these examples to verify that the API is always correctly documented. I will use a few doctests for important functions, but I will mainly focus on unit tests.

In unit tests, I will test the validation, parsing, and generation of rotation matrices from an angle. I will also unit test the utility functions for the GUI, such as `is_valid_float()`, which is needed to verify input when defining a matrix visually. I will not be testing the GUI in unit tests, since this is almost impossible to do automatically. Instead, I will have to regularly test the GUI manually; mostly defining matrices, thinking about what I expect them to look like, and then making sure they look like that. I don't see a way around this limitation. I will make my backend unit tests very thorough, but testing the GUI can only be done manually.

For the validation of matrix expressions, I will have data like the following:

Valid	Invalid
`"A"`	`""`
`"AB"`	`"A^"`
`"-3.4A"`	`"rot()"`
`"A^2"`	`"A^{2"`
`"A^T"`	`"^12"`
`"A^{-1}"`	`"A^3.2"`
`"rot(45)"`	`"A^B"`
`"3A^{12}"`	`".A"`
`"2B^2+A^TC^{-1}"`	`"--A"`
`"3.5A^{4}5.6rot(19.2)^T-B^{-1}4.1C^5"`	`"A--B"`

This list is not exhaustive, mostly to save space and time, but the full unit testing code is included in appendix B.

The invalid expressions presented here have been chosen to be almost valid, but not quite. They are

edge cases. I will also test blatantly invalid expressions like `This is a matrix expression` to make sure the validation works.

Here's an example of some test data for parsing:

Input	Expected
`"A"`	`[[("", "A", "")]]`
`"AB"`	`[[("", "A", ""), ("", "B", "")]]`
`"2A+B^2"`	`[[("2", "A", ""), ("", "B", "2")]]`
`"3A^T2.4B^{-1}-C"`	`[[("3", "A", "T"), ("2.4", "B", "-1")], [("-1", "C", "")]]`

The parsing output is pretty verbose and this table doesn't have enough space for most of the more complicated inputs, so here's a monster one:

```
"2.14A^{3} 4.5rot(14.5)^-1 + 8.5B^T 5.97C^14 - 3.14D^{-1} 6.7E^T"
```

which should parse to give:

```
[[("2.14", "A", "3"), ("4.5", "rot(14.5)", "-1")], [("8.5", "B", "T"), ("5.97", "C", "14")],
                  [("-3.14", "D", "-1"), ("6.7", "E", "T")]]
```

Any invalid expression will also raise a `MatrixParseError`, so I will check every invalid input previously mentioned and make sure it raises the appropriate error.

Again, this section is brief to save space and time. All unit tests are included in appendix B.

2.7 Post-development test data

I will test all parts of the program to evaluate how well I have met the success criteria outlined in §1.7. See §4 for the tests themselves.

3   Development

Please note, throughout this section, every code snippet will have two comments at the top. The first is the git commit hash that the snippet was taken from[9]. The second comment is the file name. The line numbers of the snippet reflect the line numbers of the file from where the snippet was taken. After a certain point, I introduced copyright comments at the top of every file. These are always omitted here.

3.1   Matrices backend

3.1.1   `MatrixWrapper` class

The first real part of development was creating the `MatrixWrapper` class. It needs a simple instance dictionary to be created in the constructor, and it needs a way of accessing the matrices. I decided to use Python's `__getitem__()` and `__setitem__()` special methods[38] to allow indexing into a `MatrixWrapper` object like `wrapper['M']`. This simplifies using the class.

```python
# 29ec1fedbf307e3b7ca731c4a381535fec899b0b
# src/lintrans/matrices/wrapper.py
"""A module containing a simple MatrixWrapper class to wrap matrices and context."""

import numpy as np

from lintrans.typing import MatrixType


class MatrixWrapper:
    """A simple wrapper class to hold all possible matrices and allow access to them."""

    def __init__(self):
        """Initialise a MatrixWrapper object with a matrices dict."""
        self._matrices: dict[str, MatrixType | None] = {
            'A': None, 'B': None, 'C': None, 'D': None,
            'E': None, 'F': None, 'G': None, 'H': None,
            'I': np.eye(2),  # I is always defined as the identity matrix
            'J': None, 'K': None, 'L': None, 'M': None,
            'N': None, 'O': None, 'P': None, 'Q': None,
            'R': None, 'S': None, 'T': None, 'U': None,
            'V': None, 'W': None, 'X': None, 'Y': None,
            'Z': None
        }

    def __getitem__(self, name: str) -> MatrixType | None:
        """Get the matrix with `name` from the dictionary.

        Raises:
            KeyError:
                If there is no matrix with the given name
        """
        return self._matrices[name]

    def __setitem__(self, name: str, new_matrix: MatrixType) -> None:
        """Set the value of matrix `name` with the new_matrix.

        Raises:
            ValueError:
                If `name` isn't a valid matrix name
        """
        name = name.upper()

        if name == 'I' or name not in self._matrices:
            raise NameError('Matrix name must be a capital letter and cannot be "I"')
```

---

[9]A history of all commits can be found in the GitHub repository[3]

```
44
45                  self._matrices[name] = new_matrix
```

This code is very simple. The constructor (`__init__()`) creates a dictionary of matrices which all start out as having no value, except the identity matrix **I**. The `__getitem__()` and `__setitem__()` methods allow the user to easily get and set matrices just like a dictionary, and `__setitem__()` will raise an error if the name is invalid. This is a very early prototype, so it doesn't validate the type of whatever the user is trying to assign it to yet. This validation will come later.

I could make this class subclass `dict`, since it's basically just a dictionary at this point, but I want to extend it with much more functionality later, so I chose to handle the dictionary stuff myself.

I then had to write unit tests for this class, and I chose to do all my unit tests using a framework called `pytest`.

```python
# 29ec1fedbf307e3b7ca731c4a381535fec899b0b
# tests/test_matrix_wrapper.py

1   """Test the MatrixWrapper class."""
2
3   import numpy as np
4   import pytest
5   from lintrans.matrices import MatrixWrapper
6
7   valid_matrix_names = 'ABCDEFGHJKLMNOPQRSTUVWXYZ'
8   test_matrix = np.array([[1, 2], [4, 3]])
9
10
11  @pytest.fixture
12  def wrapper() -> MatrixWrapper:
13      """Return a new MatrixWrapper object."""
14      return MatrixWrapper()
15
16
17  def test_get_matrix(wrapper) -> None:
18      """Test MatrixWrapper.__getitem__()."""
19      for name in valid_matrix_names:
20          assert wrapper[name] is None
21
22      assert (wrapper['I'] == np.array([[1, 0], [0, 1]])).all()
23
24
25  def test_get_name_error(wrapper) -> None:
26      """Test that MatrixWrapper.__getitem__() raises a KeyError if called with an invalid name."""
27      with pytest.raises(KeyError):
28          _ = wrapper['bad name']
29          _ = wrapper['123456']
30          _ = wrapper['Th15 Is an 1nV@l1D n@m3']
31          _ = wrapper['abc']
32
33
34  def test_set_matrix(wrapper) -> None:
35      """Test MatrixWrapper.__setitem__()."""
36      for name in valid_matrix_names:
37          wrapper[name] = test_matrix
38          assert (wrapper[name] == test_matrix).all()
39
40
41  def test_set_identity_error(wrapper) -> None:
42      """Test that MatrixWrapper.__setitem__() raises a NameError when trying to assign to I."""
43      with pytest.raises(NameError):
44          wrapper['I'] = test_matrix
45
46
47  def test_set_name_error(wrapper) -> None:
48      """Test that MatrixWrapper.__setitem__() raises a NameError when trying to assign to an invalid name."""
49      with pytest.raises(NameError):
50          wrapper['bad name'] = test_matrix
51          wrapper['123456'] = test_matrix
```

```
52              wrapper['Th15 Is an 1nV@l1D n@m3'] = test_matrix
53              wrapper['abc'] = test_matrix


(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *]
$ pytest -v
======================================= test session starts =======================================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml, testpaths: tests
collected 5 items

tests/test_matrix_wrapper.py::test_get_matrix PASSED                                      [ 20%]
tests/test_matrix_wrapper.py::test_get_name_error PASSED                                  [ 40%]
tests/test_matrix_wrapper.py::test_set_matrix PASSED                                      [ 60%]
tests/test_matrix_wrapper.py::test_set_identity_error PASSED                              [ 80%]
tests/test_matrix_wrapper.py::test_set_name_error PASSED                                  [100%]


======================================= 5 passed in 0.27s =======================================
```

Figure 3.1.1: Running `pytest` with the new tests

These tests are quite simple and just ensure that the expected behaviour works the way it should, and that the correct errors are raised when they should be. It verifies that matrices can be assigned, that every valid name works, and that the identity matrix **I** cannot be assigned to.

The function decorated with `@pytest.fixture` allows functions to use a parameter called `wrapper` and `pytest` will automatically call this function and pass it as that parameter. It just saves on code repetition.

3.1.2 Rudimentary parsing and evaluating

This first thing I did here was improve the `__setitem__()` and `__getitem__()` methods to validate input and easily get transposes and simple rotation matrices.

```python
# f89fc9fd8d5917d07557fc50df3331123b55ad6b
# src/lintrans/matrices/wrapper.py

11   class MatrixWrapper:
...
60       def __setitem__(self, name: str, new_matrix: MatrixType) -> None:
61           """Set the value of matrix `name` with the new_matrix.
62
63           :param str name: The name of the matrix to set the value of
64           :param MatrixType new_matrix: The value of the new matrix
65           :rtype: None
66
67           :raises NameError: If the name isn't a valid matrix name or is 'I'
68           """
69           if name not in self._matrices.keys():
70               raise NameError('Matrix name must be a single capital letter')
71
72           if name == 'I':
73               raise NameError('Matrix name cannot be "I"')
74
75           # All matrices must have float entries
76           a = float(new_matrix[0][0])
77           b = float(new_matrix[0][1])
78           c = float(new_matrix[1][0])
79           d = float(new_matrix[1][1])
80
81           self._matrices[name] = np.array([[a, b], [c, d]])
```

In this method, I'm now casting all the values to floats. This is very simple validation, since this cast will raise **ValueError** if it fails to cast the value to a float. I should've declared `:raises ValueError:` in the docstring, but this was an oversight at the time.

```python
# f89fc9fd8d5917d07557fc50df3331123b55ad6b
# src/lintrans/matrices/wrapper.py

11      class MatrixWrapper:
...
27          def __getitem__(self, name: str) -> Optional[MatrixType]:
28              """Get the matrix with the given name.
29
30              If it is a simple name, it will just be fetched from the dictionary.
31              If the name is followed with a 't', then we will return the transpose of the named matrix.
32              If the name is 'rot()', with a given angle in degrees, then we return a new rotation matrix with that angle.
33
34              :param str name: The name of the matrix to get
35              :returns: The value of the matrix (may be none)
36              :rtype: Optional[MatrixType]
37
38              :raises NameError: If there is no matrix with the given name
39              """
40              # Return a new rotation matrix
41              match = re.match(r'rot\((\d+)\)', name)
42              if match is not None:
43                  return create_rotation_matrix(float(match.group(1)))
44
45              # Return the transpose of this matrix
46              match = re.match(r'([A-Z])t', name)
47              if match is not None:
48                  matrix = self[match.group(1)]
49
50                  if matrix is not None:
51                      return matrix.T
52                  else:
53                      return None
54
55              if name not in self._matrices:
56                  raise NameError(f'Unrecognised matrix name "{name}"')
57
58              return self._matrices[name]
59
```

This `__getitem__()` method now allows for easily accessing transposes and rotation matrices by checking input with regular expressions. This makes getting matrices easier and thus makes evaluating full expressions simpler.

The `create_rotation_matrix()` method is also defined in this file and just uses the $\left(\begin{smallmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{smallmatrix}\right)$ formula from before:

```python
# f89fc9fd8d5917d07557fc50df3331123b55ad6b
# src/lintrans/matrices/wrapper.py

158     def create_rotation_matrix(angle: float) -> MatrixType:
159         """Create a matrix representing a rotation by the given number of degrees anticlockwise.
160
161         :param float angle: The number of degrees to rotate by
162         :returns MatrixType: The resultant rotation matrix
163         """
164         rad = np.deg2rad(angle)
165         return np.array([
166             [np.cos(rad), -1 * np.sin(rad)],
167             [np.sin(rad), np.cos(rad)]
168         ])
```

At this stage, I also implemented a simple parser and evaluator using regular expressions. It's not great and it's not very flexible, but it can evaluate simple expressions.

```python
# f89fc9fd8d5917d07557fc50df3331123b55ad6b
# src/lintrans/matrices/wrapper.py
```

```
11      class MatrixWrapper:
...
83          def parse_expression(self, expression: str) -> MatrixType:
84              """Parse a given expression and return the matrix for that expression.
85
86              Expressions are written with standard LaTeX notation for exponents. All whitespace is ignored.
87
88              Here is documentation on syntax:
89                  A single matrix is written as 'A'.
90                  Matrix A multiplied by matrix B is written as 'AB'
91                  Matrix A plus matrix B is written as 'A+B'
92                  Matrix A minus matrix B is written as 'A-B'
93                  Matrix A squared is written as 'A^2'
94                  Matrix A to the power of 10 is written as 'A^10' or 'A^{10}'
95                  The inverse of matrix A is written as 'A^-1' or 'A^{-1}'
96                  The transpose of matrix A is written as 'A^T' or 'At'
97
98              :param str expression: The expression to be parsed
99              :returns MatrixType: The matrix result of the expression
100
101             :raises ValueError: If the expression is invalid, such as an empty string
102             """
103             if expression == '':
104                 raise ValueError('The expression cannot be an empty string')
105
106             match = re.search(r'[^-+A-Z^{}rot()\d.]', expression)
107             if match is not None:
108                 raise ValueError(f'Invalid character "{match.group(0)}"')
109
110             # Remove all whitespace in the expression
111             expression = re.sub(r'\s', '', expression)
112
113             # Wrap all exponents and transposition powers with {}
114             expression = re.sub(r'(?<=\^)(-?\d+|T)(?=[^}]|$)', r'{\g<0>}', expression)
115
116             # Replace all subtractions with additions, multiplied by -1
117             expression = re.sub(r'(?<=.)-(?=[A-Z])', '+-1', expression)
118
119             # Replace a possible leading minus sign with -1
120             expression = re.sub(r'^-(?=[A-Z])', '-1', expression)
121
122             # Change all transposition exponents into lowercase
123             expression = expression.replace('^{T}', 't')
124
125             # Split the expression into groups to be multiplied, and then we add those groups at the end
126             # We also have to filter out the empty strings to reduce errors
127             multiplication_groups = [x for x in expression.split('+') if x != '']
128
129             # Start with the 0 matrix and add each group on
130             matrix_sum: MatrixType = np.array([[0., 0.], [0., 0.]])
131
132             for group in multiplication_groups:
133                 # Generate a list of tuples, each representing a matrix
134                 # These tuples are (the multiplier, the matrix (with optional
135                 # 't' at the end to indicate a transpose), the exponent)
136                 string_matrices: list[tuple[str, str, str]]
137
138                 # The generate tuple is (multiplier, matrix, full exponent, stripped exponent)
139                 # The full exponent contains ^{}, so we ignore it
140                 # The multiplier and exponent might be '', so we have to set them to '1'
141                 string_matrices = [(t[0] if t[0] != '' else '1', t[1], t[3] if t[3] != '' else '1')
142                                    for t in re.findall(r'(-?\d*\.?\d*)([A-Z]t?|rot\(\d+\))(\^{(-?\d+|T)})?', group)]
143
144                 # This list is a list of tuple, where each tuple is (a float multiplier,
145                 # the matrix (gotten from the wrapper's __getitem__()), the integer power)
146                 matrices: list[tuple[float, MatrixType, int]]
147                 matrices = [(float(t[0]), self[t[1]], int(t[2])) for t in string_matrices]
148
149                 # Process the matrices and make actual MatrixType objects
150                 processed_matrices: list[MatrixType] = [t[0] * np.linalg.matrix_power(t[1], t[2]) for t in matrices]
151
152                 # Add this matrix product to the sum total
153                 matrix_sum += reduce(lambda m, n: m @ n, processed_matrices)
```

```
154
155                return matrix_sum
```

I think the comments in the code speak for themselves, but we basically split the expression up into groups to be added, and then for each group, we multiply every matrix in that group to get its value, and then add all these values together at the end.

This code is objectively bad. At the time of writing, it's now quite old, so I can say that. This code has no real error handling, and line 127 introduces the glaring error that `'A++B'` is now a valid expression because we disregard empty strings. Not to mention the fact that the method is called `parse_expression()` but actually evaluates an expression. All these issues will be fixed in the future, but this was the first implementation of matrix evaluation, and it does the job decently well.

I then implemented several tests for this parsing.

```python
# 60e0c713b244e097bab8ee0f71142b709fde1a8b
# tests/test_matrix_wrapper_parse_expression.py

1    """Test the MatrixWrapper parse_expression() method."""
2
3    import numpy as np
4    from numpy import linalg as la
5    import pytest
6    from lintrans.matrices import MatrixWrapper
7
8
9    @pytest.fixture
10   def wrapper() -> MatrixWrapper:
11       """Return a new MatrixWrapper object with some preset values."""
12       wrapper = MatrixWrapper()
13
14       root_two_over_two = np.sqrt(2) / 2
15
16       wrapper['A'] = np.array([[1, 2], [3, 4]])
17       wrapper['B'] = np.array([[6, 4], [12, 9]])
18       wrapper['C'] = np.array([[-1, -3], [4, -12]])
19       wrapper['D'] = np.array([[13.2, 9.4], [-3.4, -1.8]])
20       wrapper['E'] = np.array([
21           [root_two_over_two, -1 * root_two_over_two],
22           [root_two_over_two, root_two_over_two]
23       ])
24       wrapper['F'] = np.array([[-1, 0], [0, 1]])
25       wrapper['G'] = np.array([[np.pi, np.e], [1729, 743.631]])
26
27       return wrapper
28
29
30   def test_simple_matrix_addition(wrapper: MatrixWrapper) -> None:
31       """Test simple addition and subtraction of two matrices."""
32
33       # NOTE: We assert that all of these values are not None just to stop mypy complaining
34       # These values will never actually be None because they're set in the wrapper() fixture
35       # There's probably a better way  do this, because this method is a bit of a bdoge, but this works for now
36       assert wrapper['A'] is not None and wrapper['B'] is not None and wrapper['C'] is not None and \
37           wrapper['D'] is not None and wrapper['E'] is not None and wrapper['F'] is not None and \
38           wrapper['G'] is not None
39
40       assert (wrapper.parse_expression('A+B') == wrapper['A'] + wrapper['B']).all()
41       assert (wrapper.parse_expression('E+F') == wrapper['E'] + wrapper['F']).all()
42       assert (wrapper.parse_expression('G+D') == wrapper['G'] + wrapper['D']).all()
43       assert (wrapper.parse_expression('C+C') == wrapper['C'] + wrapper['C']).all()
44       assert (wrapper.parse_expression('D+A') == wrapper['D'] + wrapper['A']).all()
45       assert (wrapper.parse_expression('B+C') == wrapper['B'] + wrapper['C']).all()
46
47
48   def test_simple_two_matrix_multiplication(wrapper: MatrixWrapper) -> None:
49       """Test simple multiplication of two matrices."""
50       assert wrapper['A'] is not None and wrapper['B'] is not None and wrapper['C'] is not None and \
51           wrapper['D'] is not None and wrapper['E'] is not None and wrapper['F'] is not None and \
```

```python
52                    wrapper['G'] is not None
53
54        assert (wrapper.parse_expression('AB') == wrapper['A'] @ wrapper['B']).all()
55        assert (wrapper.parse_expression('BA') == wrapper['B'] @ wrapper['A']).all()
56        assert (wrapper.parse_expression('AC') == wrapper['A'] @ wrapper['C']).all()
57        assert (wrapper.parse_expression('DA') == wrapper['D'] @ wrapper['A']).all()
58        assert (wrapper.parse_expression('ED') == wrapper['E'] @ wrapper['D']).all()
59        assert (wrapper.parse_expression('FD') == wrapper['F'] @ wrapper['D']).all()
60        assert (wrapper.parse_expression('GA') == wrapper['G'] @ wrapper['A']).all()
61        assert (wrapper.parse_expression('CF') == wrapper['C'] @ wrapper['F']).all()
62        assert (wrapper.parse_expression('AG') == wrapper['A'] @ wrapper['G']).all()
63
64
65    def test_identity_multiplication(wrapper: MatrixWrapper) -> None:
66        """Test that multiplying by the identity doesn't change the value of a matrix."""
67        assert wrapper['A'] is not None and wrapper['B'] is not None and wrapper['C'] is not None and \
68                wrapper['D'] is not None and wrapper['E'] is not None and wrapper['F'] is not None and \
69                wrapper['G'] is not None
70
71        assert (wrapper.parse_expression('I') == wrapper['I']).all()
72        assert (wrapper.parse_expression('AI') == wrapper['A']).all()
73        assert (wrapper.parse_expression('IA') == wrapper['A']).all()
74        assert (wrapper.parse_expression('GI') == wrapper['G']).all()
75        assert (wrapper.parse_expression('IG') == wrapper['G']).all()
76
77        assert (wrapper.parse_expression('EID') == wrapper['E'] @ wrapper['D']).all()
78        assert (wrapper.parse_expression('IED') == wrapper['E'] @ wrapper['D']).all()
79        assert (wrapper.parse_expression('EDI') == wrapper['E'] @ wrapper['D']).all()
80        assert (wrapper.parse_expression('IEIDI') == wrapper['E'] @ wrapper['D']).all()
81        assert (wrapper.parse_expression('EI^3D') == wrapper['E'] @ wrapper['D']).all()
82
83
84    def test_simple_three_matrix_multiplication(wrapper: MatrixWrapper) -> None:
85        """Test simple multiplication of two matrices."""
86        assert wrapper['A'] is not None and wrapper['B'] is not None and wrapper['C'] is not None and \
87                wrapper['D'] is not None and wrapper['E'] is not None and wrapper['F'] is not None and \
88                wrapper['G'] is not None
89
90        assert (wrapper.parse_expression('ABC') == wrapper['A'] @ wrapper['B'] @ wrapper['C']).all()
91        assert (wrapper.parse_expression('ACB') == wrapper['A'] @ wrapper['C'] @ wrapper['B']).all()
92        assert (wrapper.parse_expression('BAC') == wrapper['B'] @ wrapper['A'] @ wrapper['C']).all()
93        assert (wrapper.parse_expression('EFG') == wrapper['E'] @ wrapper['F'] @ wrapper['G']).all()
94        assert (wrapper.parse_expression('DAC') == wrapper['D'] @ wrapper['A'] @ wrapper['C']).all()
95        assert (wrapper.parse_expression('GAE') == wrapper['G'] @ wrapper['A'] @ wrapper['E']).all()
96        assert (wrapper.parse_expression('FAG') == wrapper['F'] @ wrapper['A'] @ wrapper['G']).all()
97        assert (wrapper.parse_expression('GAF') == wrapper['G'] @ wrapper['A'] @ wrapper['F']).all()
98
99
100   def test_matrix_inverses(wrapper: MatrixWrapper) -> None:
101       """Test the inverses of single matrices."""
102       assert wrapper['A'] is not None and wrapper['B'] is not None and wrapper['C'] is not None and \
103               wrapper['D'] is not None and wrapper['E'] is not None and wrapper['F'] is not None and \
104               wrapper['G'] is not None
105
106       assert (wrapper.parse_expression('A^{-1}') == la.inv(wrapper['A'])).all()
107       assert (wrapper.parse_expression('B^{-1}') == la.inv(wrapper['B'])).all()
108       assert (wrapper.parse_expression('C^{-1}') == la.inv(wrapper['C'])).all()
109       assert (wrapper.parse_expression('D^{-1}') == la.inv(wrapper['D'])).all()
110       assert (wrapper.parse_expression('E^{-1}') == la.inv(wrapper['E'])).all()
111       assert (wrapper.parse_expression('F^{-1}') == la.inv(wrapper['F'])).all()
112       assert (wrapper.parse_expression('G^{-1}') == la.inv(wrapper['G'])).all()
113
114       assert (wrapper.parse_expression('A^-1') == la.inv(wrapper['A'])).all()
115       assert (wrapper.parse_expression('B^-1') == la.inv(wrapper['B'])).all()
116       assert (wrapper.parse_expression('C^-1') == la.inv(wrapper['C'])).all()
117       assert (wrapper.parse_expression('D^-1') == la.inv(wrapper['D'])).all()
118       assert (wrapper.parse_expression('E^-1') == la.inv(wrapper['E'])).all()
119       assert (wrapper.parse_expression('F^-1') == la.inv(wrapper['F'])).all()
120       assert (wrapper.parse_expression('G^-1') == la.inv(wrapper['G'])).all()
121
122
123   def test_matrix_powers(wrapper: MatrixWrapper) -> None:
124       """Test that matrices can be raised to integer powers."""
```

```
125        assert wrapper['A'] is not None and wrapper['B'] is not None and wrapper['C'] is not None and \
126            wrapper['D'] is not None and wrapper['E'] is not None and wrapper['F'] is not None and \
127            wrapper['G'] is not None
128
129        assert (wrapper.parse_expression('A^2') == la.matrix_power(wrapper['A'], 2)).all()
130        assert (wrapper.parse_expression('B^4') == la.matrix_power(wrapper['B'], 4)).all()
131        assert (wrapper.parse_expression('C^{12}') == la.matrix_power(wrapper['C'], 12)).all()
132        assert (wrapper.parse_expression('D^12') == la.matrix_power(wrapper['D'], 12)).all()
133        assert (wrapper.parse_expression('E^8') == la.matrix_power(wrapper['E'], 8)).all()
134        assert (wrapper.parse_expression('F^{-6}') == la.matrix_power(wrapper['F'], -6)).all()
135        assert (wrapper.parse_expression('G^-2') == la.matrix_power(wrapper['G'], -2)).all()
```

```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *]
$ pytest -v
======================================= test session starts =======================================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml, testpaths: tests
collected 12 items

tests/test_matrix_wrapper_parse_expression.py::test_simple_matrix_addition PASSED            [  8%]
tests/test_matrix_wrapper_parse_expression.py::test_simple_two_matrix_multiplication PASSED  [ 16%]
tests/test_matrix_wrapper_parse_expression.py::test_identity_multiplication PASSED            [ 25%]
tests/test_matrix_wrapper_parse_expression.py::test_simple_three_matrix_multiplication PASSED [ 33%]
tests/test_matrix_wrapper_parse_expression.py::test_matrix_inverses PASSED                    [ 41%]
tests/test_matrix_wrapper_parse_expression.py::test_matrix_powers PASSED                      [ 50%]
tests/test_matrix_wrapper_setitem_and_getitem.py::test_get_matrix PASSED                      [ 58%]
tests/test_matrix_wrapper_setitem_and_getitem.py::test_get_name_error PASSED                  [ 66%]
tests/test_matrix_wrapper_setitem_and_getitem.py::test_set_matrix PASSED                      [ 75%]
tests/test_matrix_wrapper_setitem_and_getitem.py::test_set_identity_error PASSED              [ 83%]
tests/test_matrix_wrapper_setitem_and_getitem.py::test_set_name_error PASSED                  [ 91%]
tests/test_matrix_wrapper_setitem_and_getitem.py::test_set_type_error PASSED                  [100%]

======================================= 12 passed in 0.35s ========================================
```

Figure 3.1.2: Running `pytest` with the new tests

These test lots of simple expressions, but don't test any more complicated expressions, nor do they test any validation, mostly because validation doesn't really exist at this point. `'A++B'` is still a valid expression and is equivalent to `'A+B'`.

3.1.3 Simple matrix expression validation

My next major step was to implement proper parsing, but I procrastinated for a while and first implemented proper validation.

```python
# 39b918651f60bc72bc19d2018075b24a6fc3af17
# src/lintrans/_parse/matrices.py

9   def compile_valid_expression_pattern() -> Pattern[str]:
10      """Compile the single regular expression that will match a valid matrix expression."""
11      digit_no_zero = '[123456789]'
12      digits = '\\d+'
13      integer_no_zero = '-?' + digit_no_zero + '(' + digits + ')?'
14      real_number = f'({integer_no_zero}(\\.{digits})?|-?0?\\.{digits})'
15
16      index_content = f'({integer_no_zero}|T)'
17      index = f'(\\^\\{{{index_content}\\}}|\\^{index_content}|t)'
18      matrix_identifier = f'([A-Z]|rot\\({real_number}\\))'
19      matrix = '(' + real_number + '?' + matrix_identifier + index + '?)'
20      expression = f'{matrix}+((\\+|-){matrix}+)*'
21
22      return re.compile(expression)
23
24
25   # This is an expensive pattern to compile, so we compile it when this module is initialized
26   valid_expression_pattern = compile_valid_expression_pattern()
```

```
27
28
29    def validate_matrix_expression(expression: str) -> bool:
30        """Validate the given matrix expression.
31
32        This function simply checks the expression against a BNF schema. It is not
33        aware of which matrices are actually defined in a wrapper. For an aware
34        version of this function, use the MatrixWrapper().is_valid_expression() method.
35
36        Here is the schema for a valid expression given in a version of BNF:
37
38            expression       ::=  matrices { ( "+" | "-" ) matrices };
39            matrices         ::=  matrix { matrix };
40            matrix           ::=  [ real_number ] matrix_identifier [ index ];
41            matrix_identifier ::= "A" .. "Z" | "rot(" real_number ")";
42            index            ::=  "^{" index_content "}" | "^" index_content | "t";
43            index_content    ::=  integer_not_zero | "T";
44
45            digit_no_zero    ::=  "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
46            digit            ::=  "0" | digit_no_zero;
47            digits           ::=  digit | digits digit;
48            integer_not_zero ::=  [ "-" ] digit_no_zero [ digits ];
49            real_number      ::=  ( integer_not_zero [ "." digits ] | [ "-" ] [ "0" ] "." digits );
50
51        :param str expression: The expression to be validated
52        :returns bool: Whether the expression is valid according to the schema
53        """
54        match = valid_expression_pattern.match(expression)
55        return expression == match.group(0) if match is not None else False
```

Here, I'm using a BNF schema to programmatically generate a regular expression. I use a function to generate this pattern and assign it to a variable when the module is initialized. This is because the pattern compilation is expensive and it's more efficient to compile the pattern once and then just use it in the `validate_matrix_expression()` function.

I also created a method `is_valid_expression()` in `MatrixWrapper`, which just validates a given expression. It uses the aforementioned `validate_matrix_expression()` and also checks that every matrix referenced in the expression is defined in the wrapper.

```
      # 39b918651f60bc72bc19d2018075b24a6fc3af17
      # src/lintrans/matrices/wrapper.py

12    class MatrixWrapper:
...
99        def is_valid_expression(self, expression: str) -> bool:
100           """Check if the given expression is valid, using the context of the wrapper.
101
102           This method calls _parse.validate_matrix_expression(), but also ensures
103           that all the matrices in the expression are defined in the wrapper.
104
105           :param str expression: The expression to validate
106           :returns bool: Whether the expression is valid according the schema
107           """
108           # Get rid of the transposes to check all capital letters
109           expression = re.sub(r'\^T', 't', expression)
110           expression = re.sub(r'\^{T}', 't', expression)
111
112           # Make sure all the referenced matrices are defined
113           for matrix in {x for x in expression if re.match('[A-Z]', x)}:
114               if self[matrix] is None:
115                   return False
116
117           return _parse.validate_matrix_expression(expression)
```

I then implemented some simple tests to make sure the function works with valid and invalid expressions.

```
      # a0fb029f7da995803c24ee36e7e8078e5621f676
```

```
     # tests/_parse/test_parse_and_validate_expression.py

1    """Test the _parse.matrices module validation and parsing."""
2
3    import pytest
4    from lintrans._parse import validate_matrix_expression
5
6    valid_inputs: list[str] = [
7        'A', 'AB', '3A', '1.2A', '-3.4A', 'A^2', 'A^-1', 'A^{-1}',
8        'A^12', 'A^T', 'A^{5}', 'A^{T}', '4.3A^7', '9.2A^{18}',
9
10       'rot(45)', 'rot(12.5)', '3rot(90)',
11       'rot(135)^3', 'rot(51)^T', 'rot(-34)^-1',
12
13       'A+B', 'A+2B', '4.3A+9B', 'A^2+B^T', '3A^7+0.8B^{16}',
14       'A-B', '3A-4B', '3.2A^3-16.79B^T', '4.752A^{17}-3.32B^{36}',
15       'A--1B', '-A', '--1A'
16
17       '3A4B', 'A^TB', 'A^{T}B', '4A^6B^3',
18       '2A^{3}4B^5', '4rot(90)^3', 'rot(45)rot(13)',
19       'Arot(90)', 'AB^2', 'A^2B^2', '8.36A^T3.4B^12',
20
21       '3.5A^{4}5.6rot(19.2)^T-B^{-1}4.1C^5',
22   ]
23
24   invalid_inputs: list[str] = [
25       '', 'rot()', 'A^', 'A^1.2', 'A^{3.4}', '1,2A', 'ro(12)', '5', '12^2',
26       '^T', '^{12}', 'A{13}', 'A^3}', 'A^A', '^2', 'A--B', '--A'
27
28       'This is 100% a valid matrix expression, I swear'
29   ]
30
31
32   @pytest.mark.parametrize('inputs,output', [(valid_inputs, True), (invalid_inputs, False)])
33   def test_validate_matrix_expression(inputs: list[str], output: bool) -> None:
34       """Test the validate_matrix_expression() function."""
35       for inp in inputs:
36           assert validate_matrix_expression(inp) == output
```

```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *]
$ pytest -v
========================================= test session starts =========================================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml, testpaths: tests
collected 18 items

tests/_parse/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs0-True] PASSED    [  5%]
tests/_parse/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs1-False] PASSED   [ 11%]
tests/matrices/test_rotation_matrices.py::test_create_rotation_matrix PASSED                                [ 16%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_matrix_addition PASSED               [ 22%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_two_matrix_multiplication PASSED     [ 27%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_identity_multiplication PASSED              [ 33%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_three_matrix_multiplication PASSED   [ 38%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_inverses PASSED                      [ 44%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_powers PASSED                        [ 50%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_transpose PASSED                     [ 55%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_rotation_matrices PASSED                    [ 61%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_value_errors PASSED                         [ 66%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_get_matrix PASSED                           [ 72%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_get_name_error PASSED                       [ 77%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_matrix PASSED                           [ 83%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_identity_error PASSED                   [ 88%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_name_error PASSED                       [ 94%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_type_error PASSED                       [100%]

========================================= 18 passed in 0.31s =========================================
```

Figure 3.1.3: Running `pytest` with the new tests

Here, we test some valid data, some definitely invalid data, and some edge cases. At this stage, 'A--1B' was considered a valid expression. This was a quirk of the validator at the time, but I fixed it later. This should obviously be an invalid expression, especially since 'A--B' is considered invalid, but 'A--1B' is valid.

The `@pytest.mark.parametrize` decorator on line 32 means that `pytest` will run one test for valid

inputs, and then another test for invalid inputs, and these will count as different tests. This makes it easier to see which tests failed and then debug the app.

While I was adding tests, I added new doctests for the `MatrixWrapper` class and `create_rotation` `_matrix()` function.

```python
# 6db1a1a1b3deaa30b1a8d23bb66ff738a3f212d2
# src/lintrans/matrices/wrapper.py

class MatrixWrapper:
    """A simple wrapper class to hold all possible matrices and allow access to them.

    The contained matrices can be accessed with square bracket notation.

    :Example:

    >>> wrapper = MatrixWrapper()
    >>> wrapper['I']
    array([[1., 0.],
           [0., 1.]])
    >>> wrapper['M']  # Returns None
    >>> wrapper['M'] = np.array([[1, 2], [3, 4]])
    >>> wrapper['M']
    array([[1., 2.],
           [3., 4.]])

    Methods:
        is_valid_expression(expression: str) -> bool:
            Check if the given expression is valid, using the context of the wrapper.

        evaluate_expression(expression: str) -> MatrixType:
            Evaluate a given expression and return the matrix for that expression.
    """
```

```python
# e79664861434059eb2b27a38310a140e00c6432b
# src/lintrans/matrices/wrapper.py

def create_rotation_matrix(angle: float, degrees: bool = True) -> MatrixType:
    """Create a matrix representing a rotation by the given angle (anticlockwise).

    :Example:

    >>> create_rotation_matrix(30)
    array([[ 0.8660254, -0.5      ],
           [ 0.5      ,  0.8660254]])
    >>> create_rotation_matrix(45)
    array([[ 0.70710678, -0.70710678],
           [ 0.70710678,  0.70710678]])
    >>> create_rotation_matrix(np.pi / 3, degrees=False)
    array([[ 0.5      , -0.8660254],
           [ 0.8660254,  0.5      ]])

    :param float angle: The angle to rotate anticlockwise by
    :param bool degrees: Whether to interpret the angle as degrees (True) or radians (False)
    :returns MatrixType: The resultant rotation matrix
    """
```

```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *=]
$ pytest --doctest-modules src/ -v
========================================= test session starts =========================================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml
collected 2 items

src/lintrans/matrices/wrapper.py::lintrans.matrices.wrapper.MatrixWrapper PASSED                 [ 50%]
src/lintrans/matrices/wrapper.py::lintrans.matrices.wrapper.create_rotation_matrix PASSED        [100%]

========================================== 2 passed in 0.28s ==========================================
```

Figure 3.1.4: Running `pytest` with the new doctests

3.1.4 Parsing matrix expressions

Parsing is quite an interesting problem and something I didn't feel able to tackle head-on, so I wrote the unit tests first. I had a basic idea of what I wanted the parser to return, but no real idea of how to implement that. My unit tests looked like this:

```python
# e9f7a81892278fe70684562052f330fb3a02bf9b
# tests/_parse/test_parse_and_validate_expression.py

expressions_and_parsed_expressions: list[tuple[str, MatrixParseList]] = [
    # Simple expressions
    ('A', [[('', 'A', '')]]),
    ('A^2', [[('', 'A', '2')]]),
    ('A^{2}', [[('', 'A', '2')]]),
    ('3A', [[('3', 'A', '')]]),
    ('1.4A^3', [[('1.4', 'A', '3')]]),

    # Multiplications
    ('4A^{3} 6B^2', [[('4', 'A', '3'), ('6', 'B', '2')]]),
    ('4.2A^{T} 6.1B^-1', [[('4.2', 'A', 'T'), ('6.1', 'B', '-1')]]),
    ('-1.2A^2 rot(45)^2', [[('-1.2', 'A', '2'), ('', 'rot(45)', '2')]]),
    ('3.2A^T 4.5B^{5} 9.6rot(121.3)', [[('3.2', 'A', 'T'), ('4.5', 'B', '5'), ('9.6', 'rot(121.3)', '')]]),
    ('-1.18A^{-2} 0.1B^{2} 9rot(34.6)^-1', [[('-1.18', 'A', '-2'), ('0.1', 'B', '2'), ('9', 'rot(34.6)', '-1')]]),

    # Additions
    ('A + B', [[('', 'A', '')], [('', 'B', '')]]),
    ('A + B - C', [[('', 'A', '')], [('', 'B', '')], [('-1', 'C', '')]]),
    ('2A^3 + 8B^T - 3C^-1', [[('2', 'A', '3')], [('8', 'B', 'T')], [('-3', 'C', '-1')]]),

    # Additions with multiplication
    ('2.14A^{3} 4.5rot(14.5)^-1 + 8B^T - 3C^-1', [[('2.14', 'A', '3'), ('4.5', 'rot(14.5)', '-1')],
                                                  [('8', 'B', 'T')], [('-3', 'C', '-1')]]),
    ('2.14A^{3} 4.5rot(14.5)^-1 + 8.5B^T 5.97C^4 - 3.14D^{-1} 6.7E^T',
     [[('2.14', 'A', '3'), ('4.5', 'rot(14.5)', '-1')], [('8.5', 'B', 'T'), ('5.97', 'C', '4')],
      [('-3.14', 'D', '-1'), ('6.7', 'E', 'T')]]),
]


@pytest.mark.skip(reason='parse_matrix_expression() not implemented')
def test_parse_matrix_expression() -> None:
    """Test the parse_matrix_expression() function."""
    for expression, parsed_expression in expressions_and_parsed_expressions:
        # Test it with and without whitespace
        assert parse_matrix_expression(expression) == parsed_expression
        assert parse_matrix_expression(expression.replace(' ', '')) == parsed_expression
```

```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *]
$ pytest -v
============================================================ test session starts ============================================================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml, testpaths: tests
collected 19 items

tests/_parse/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs0-True] PASSED         [  5%]
tests/_parse/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs1-False] PASSED        [ 10%]
tests/_parse/test_parse_and_validate_expression.py::test_parse_matrix_expression SKIPPED (parse_matrix_expression() not implemented)  [ 15%]
tests/matrices/test_rotation_matrices.py::test_create_rotation_matrix PASSED                                     [ 21%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_matrix_addition PASSED                    [ 26%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_two_matrix_multiplication PASSED          [ 31%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_identity_multiplication PASSED                   [ 36%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_three_matrix_multiplication PASSED        [ 42%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_inverses PASSED                           [ 47%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_powers PASSED                             [ 52%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_transpose PASSED                          [ 57%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_rotation_matrices PASSED                         [ 63%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_value_errors PASSED                              [ 68%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_get_matrix PASSED                                [ 73%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_get_name_error PASSED                            [ 78%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_matrix PASSED                                [ 84%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_identity_error PASSED                        [ 89%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_name_error PASSED                            [ 94%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_type_error PASSED                            [100%]

============================================================ 18 passed, 1 skipped in 0.33s ============================================================
```

Figure 3.1.5: Running `pytest` with the new tests

I just had example inputs and what I expected as output. I also wanted the parser to ignore whitespace.

The decorator on line 69 just skips the test because the parser wasn't implemented yet.

When implementing the parser, I first had to tighten up validation to remove anomalies like `'A--1B'` being valid. I did this by factoring out the optional minus signs from being part of a number, to being optionally in front of a number. This eliminated this kind of repetition and made `'A--1B'` invalid, as it should be.

```
# fd80d8d3b0e975e92dcc7c10f1f0f1276879f408
# src/lintrans/_parse/matrices.py

32    def compile_valid_expression_pattern() -> Pattern[str]:
33        """Compile the single regular expression that will match a valid matrix expression."""
34        digit_no_zero = '[123456789]'
35        digits = '\\d+'
36        integer_no_zero = digit_no_zero + '(' + digits + ')?'
37        real_number = f'({integer_no_zero}(\\.{digits})?|0?\\.{digits})'
38
39        index_content = f'(-?{integer_no_zero}|T)'
40        index = f'(\\^\\{{{index_content}\\}}|\\^{index_content}|t)'
41        matrix_identifier = f'([A-Z]|rot\\(-?{real_number}\\))'
42        matrix = '(' + real_number + '?' + matrix_identifier + index + '?)'
43        expression = f'-?{matrix}+((\\+|-){matrix}+)*'
44
45        return re.compile(expression)
```

The code can be a bit hard to read with all the RegEx stuff, but the BNF illustrates these changes nicely.

Compare the old version:

```
# 39b918651f60bc72bc19d2018075b24a6fc3af17
# src/lintrans/_parse/matrices.py

29    def validate_matrix_expression(expression: str) -> bool:
...
36        Here is the schema for a valid expression given in a version of BNF:
...
38            expression        ::=  matrices { ( "+" | "-" ) matrices };
39            matrices          ::=  matrix { matrix };
40            matrix            ::=  [ real_number ] matrix_identifier [ index ];
41            matrix_identifier ::=  "A" .. "Z" | "rot(" real_number ")";
42            index             ::=  "^{" index_content "}" | "^" index_content | "t";
43            index_content     ::=  integer_not_zero | "T";
44
45            digit_no_zero     ::=  "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
46            digit             ::=  "0" | digit_no_zero;
47            digits            ::=  digit | digits digit;
48            integer_not_zero  ::=  [ "-" ] digit_no_zero [ digits ];
49            real_number       ::=  ( integer_not_zero [ "." digits ] | [ "-" ] [ "0" ] "." digits );
```

to the new version:

```
# fd80d8d3b0e975e92dcc7c10f1f0f1276879f408
# src/lintrans/_parse/matrices.py

52    def validate_matrix_expression(expression: str) -> bool:
...
59        Here is the schema for a valid expression given in a version of BNF:
...
61            expression        ::=  [ "-" ] matrices { ( "+" | "-" ) matrices };
62            matrices          ::=  matrix { matrix };
63            matrix            ::=  [ real_number ] matrix_identifier [ index ];
64            matrix_identifier ::=  "A" .. "Z" | "rot(" [ "-" ] real_number ")";
65            index             ::=  "^{" index_content "}" | "^" index_content | "t";
66            index_content     ::=  [ "-" ] integer_not_zero | "T";
67
```

```
68          digit_no_zero     ::=  "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
69          digit             ::=  "0" | digit_no_zero;
70          digits            ::=  digit | digits digit;
71          integer_not_zero  ::=  digit_no_zero [ digits ];
72          real_number       ::=  ( integer_not_zero [ "." digits ] | [ "0" ] "." digits );
```

Then once I'd fixed the validation, I could implement the parser itself.

```
# fd80d8d3b0e975e92dcc7c10f1f0f1276879f408
# src/lintrans/_parse/matrices.py

86    def parse_matrix_expression(expression: str) -> MatrixParseList:
87        """Parse the matrix expression and return a list of results.
88
89        The return value is a list of results. This results list contains lists of tuples.
90        The top list is the expressions that should be added together, and each sublist
91        is expressions that should be multiplied together. These expressions to be
92        multiplied are tuples, where each tuple is (multiplier, matrix identifier, index).
93        The multiplier can be any real number, the matrix identifier is either a named
94        matrix or a new rotation matrix declared with 'rot()', and the index is an
95        integer or 'T' for transpose.
96
97        :param str expression: The expression to be parsed
98        :returns MatrixParseTuple: A list of results
99        """
100       # Remove all whitespace
101       expression = re.sub(r'\s', '', expression)
102
103       # Check if it's valid
104       if not validate_matrix_expression(expression):
105           raise MatrixParseError('Invalid expression')
106
107       # Wrap all exponents and transposition powers with {}
108       expression = re.sub(r'(?<=\^)(-?\d+|T)(?=[^}]|$)', r'{\g<0>}', expression)
109
110       # Remove any standalone minuses
111       expression = re.sub(r'-(?=[A-Z])', '-1', expression)
112
113       # Replace subtractions with additions
114       expression = re.sub(r'-(?=\d+\.?\d*([A-Z]|rot))', '+-', expression)
115
116       # Get rid of a potential leading + introduced by the last step
117       expression = re.sub(r'^\+', '', expression)
118
119       return [
120           [
121               # The tuple returned by re.findall is (multiplier, matrix identifier, full index, stripped index),
122               # so we have to remove the full index, which contains the {}
123               (t[0], t[1], t[3])
124               for t in re.findall(r'(-?\d+\.?\d*)?([A-Z]|rot\(-?\d+\.?\d*\))(\^{(-?\d+|T)})?', group)
125           ]
126           # We just split the expression by '+' to have separate groups
127           for group in expression.split('+')
128       ]
```

It works similarly to the old `MatrixWrapper.parse_expression()` method in §3.1.2 but with a powerful list comprehension at the end. It splits the expression up into groups and then uses some RegEx magic to find all the matrices in these groups as a tuple.

This method passes all the unit tests, as expected.

My next step was then to rewrite the evaluation to use this new parser, like so:

```
# a453774bcdf824676461f9b9b441d7b94969ea55
# src/lintrans/matrices/wrapper.py

22    class MatrixWrapper:
```

```
...
147         def evaluate_expression(self, expression: str) -> MatrixType:
148             """Evaluate a given expression and return the matrix for that expression.
149
150             Expressions are written with standard LaTeX notation for exponents. All whitespace is ignored.
151
152             Here is documentation on syntax:
153                 A single matrix is written as a capital letter like 'A', or as 'rot(x)', where x is some angle in
                    ↪ degrees.
154                 Matrix A multiplied by matrix B is written as 'AB'
155                 Matrix A plus matrix B is written as 'A+B'
156                 Matrix A minus matrix B is written as 'A-B'
157                 Matrix A squared is written as 'A^2'
158                 Matrix A to the power of 10 is written as 'A^10' or 'A^{10}'
159                 The inverse of matrix A is written as 'A^-1' or 'A^{-1}'
160                 The transpose of matrix A is written as 'A^T' or 'At'
161                 Any matrix may be multiplied by a real constant, like '3A', or '1.2B'
162
163             :param str expression: The expression to be parsed
164             :returns MatrixType: The matrix result of the expression
165
166             :raises ValueError: If the expression is invalid
167             """
168             if not self.is_valid_expression(expression):
169                 raise ValueError('The expression is invalid')
170
171             parsed_result = _parse.parse_matrix_expression(expression)
172             final_groups: list[list[MatrixType]] = []
173
174             for group in parsed_result:
175                 f_group: list[MatrixType] = []
176
177                 for matrix in group:
178                     if matrix[2] == 'T':
179                         m = self[matrix[1]]
180                         assert m is not None
181                         matrix_value = m.T
182                     else:
183                         matrix_value = np.linalg.matrix_power(self[matrix[1]],
184                                                               1 if (index := matrix[2]) == '' else int(index))
185
186                     matrix_value *= 1 if (multiplier := matrix[0]) == '' else float(multiplier)
187                     f_group.append(matrix_value)
188
189                 final_groups.append(f_group)
190
191             return reduce(add, [reduce(matmul, group) for group in final_groups])
```

Here, we go through the list of tuples and evaluate the matrix represented by each tuple, putting this together in a list as we go. Then at the end, we simply reduce the sublists and then reduce these new matrices using a list comprehension in the reduce() call using add and matmul from the operator library. It's written in a functional programming style, and it passes all the previous tests.

3.2 Initial GUI

3.2.1 First basic GUI

The discrepancy in all the GUI code between `snake_case` and `camelCase` is because Qt5 was originally a C++ framework that was adapted into PyQt5 for Python. All the Qt API is in `camelCase`, but my Python code is in `snake_case`.

```python
# 93ce763f7b993439fc0da89fad39456d8cc4b52c
# src/lintrans/gui/main_window.py

"""The module to provide the main window as a QMainWindow object."""

import sys

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QApplication, QHBoxLayout, QMainWindow, QVBoxLayout

from lintrans.matrices import MatrixWrapper


class LintransMainWindow(QMainWindow):
    """The class for the main window in the lintrans GUI."""

    def __init__(self):
        """Create the main window object, creating every widget in it."""
        super().__init__()

        self.matrix_wrapper = MatrixWrapper()

        self.setWindowTitle('Linear Transformations')
        self.setMinimumWidth(750)

        # === Create widgets

        # Left layout: the plot and input box

        # NOTE: This QGraphicsView is only temporary
        self.plot = QtWidgets.QGraphicsView(self)

        self.text_input_expression = QtWidgets.QLineEdit(self)
        self.text_input_expression.setPlaceholderText('Input matrix expression...')
        self.text_input_expression.textChanged.connect(self.update_render_buttons)

        # Right layout: all the buttons

        # Misc buttons

        self.button_create_polygon = QtWidgets.QPushButton(self)
        self.button_create_polygon.setText('Create polygon')
        # TODO: Implement create_polygon()
        # self.button_create_polygon.clicked.connect(self.create_polygon)
        self.button_create_polygon.setToolTip('Define a new polygon to view the transformation of')

        self.button_change_display_settings = QtWidgets.QPushButton(self)
        self.button_change_display_settings.setText('Change\ndisplay settings')
        # TODO: Implement change_display_settings()
        # self.button_change_display_settings.clicked.connect(self.change_display_settings)
        self.button_change_display_settings.setToolTip('Change which things are rendered on the plot')

        # Define new matrix buttons

        self.label_define_new_matrix = QtWidgets.QLabel(self)
        self.label_define_new_matrix.setText('Define a\nnew matrix')
        self.label_define_new_matrix.setAlignment(QtCore.Qt.AlignCenter)

        # TODO: Implement defining a new matrix visually, numerically, as a rotation, and as an expression

        self.button_define_visually = QtWidgets.QPushButton(self)
        self.button_define_visually.setText('Visually')
```

```
60              self.button_define_visually.setToolTip('Drag the basis vectors')
61
62              self.button_define_numerically = QtWidgets.QPushButton(self)
63              self.button_define_numerically.setText('Numerically')
64              self.button_define_numerically.setToolTip('Define a matrix just with numbers')
65
66              self.button_define_as_rotation = QtWidgets.QPushButton(self)
67              self.button_define_as_rotation.setText('As a rotation')
68              self.button_define_as_rotation.setToolTip('Define an angle to rotate by')
69
70              self.button_define_as_expression = QtWidgets.QPushButton(self)
71              self.button_define_as_expression.setText('As an expression')
72              self.button_define_as_expression.setToolTip('Define a matrix in terms of other matrices')
73
74              # Render buttons
75
76              self.button_render = QtWidgets.QPushButton(self)
77              self.button_render.setText('Render')
78              self.button_render.setEnabled(False)
79              self.button_render.clicked.connect(self.render_expression)
80              self.button_render.setToolTip('Render the expression<br><b>(Ctrl + Enter)</b>')
81
82              self.button_render_shortcut = QtWidgets.QShortcut(QtGui.QKeySequence('Ctrl+Return'), self)
83              self.button_render_shortcut.activated.connect(self.button_render.click)
84
85              self.button_animate = QtWidgets.QPushButton(self)
86              self.button_animate.setText('Animate')
87              self.button_animate.setEnabled(False)
88              self.button_animate.clicked.connect(self.animate_expression)
89              self.button_animate.setToolTip('Animate the expression<br><b>(Ctrl + Shift + Enter)</b>')
90
91              self.button_animate_shortcut = QtWidgets.QShortcut(QtGui.QKeySequence('Ctrl+Shift+Return'), self)
92              self.button_animate_shortcut.activated.connect(self.button_animate.click)
93
94              # === Arrange widgets
95
96              self.setContentsMargins(10, 10, 10, 10)
97
98              self.vlay_left = QVBoxLayout()
99              self.vlay_left.addWidget(self.plot)
100             self.vlay_left.addWidget(self.text_input_expression)
101
102             self.vlay_misc_buttons = QVBoxLayout()
103             self.vlay_misc_buttons.setSpacing(20)
104             self.vlay_misc_buttons.addWidget(self.button_create_polygon)
105             self.vlay_misc_buttons.addWidget(self.button_change_display_settings)
106
107             self.vlay_define_new_matrix = QVBoxLayout()
108             self.vlay_define_new_matrix.setSpacing(20)
109             self.vlay_define_new_matrix.addWidget(self.label_define_new_matrix)
110             self.vlay_define_new_matrix.addWidget(self.button_define_visually)
111             self.vlay_define_new_matrix.addWidget(self.button_define_numerically)
112             self.vlay_define_new_matrix.addWidget(self.button_define_as_rotation)
113             self.vlay_define_new_matrix.addWidget(self.button_define_as_expression)
114
115             self.vlay_render = QVBoxLayout()
116             self.vlay_render.setSpacing(20)
117             self.vlay_render.addWidget(self.button_animate)
118             self.vlay_render.addWidget(self.button_render)
119
120             self.vlay_right = QVBoxLayout()
121             self.vlay_right.setSpacing(50)
122             self.vlay_right.addLayout(self.vlay_misc_buttons)
123             self.vlay_right.addLayout(self.vlay_define_new_matrix)
124             self.vlay_right.addLayout(self.vlay_render)
125
126             self.hlay_all = QHBoxLayout()
127             self.hlay_all.setSpacing(15)
128             self.hlay_all.addLayout(self.vlay_left)
129             self.hlay_all.addLayout(self.vlay_right)
130
131             self.central_widget = QtWidgets.QWidget()
132             self.central_widget.setLayout(self.hlay_all)
```

```
133                self.setCentralWidget(self.central_widget)
134
135        def update_render_buttons(self) -> None:
136            """Enable or disable the render and animate buttons according to the validity of the matrix expression."""
137            valid = self.matrix_wrapper.is_valid_expression(self.text_input_expression.text())
138            self.button_render.setEnabled(valid)
139            self.button_animate.setEnabled(valid)
140
141        def render_expression(self) -> None:
142            """Render the expression in the input box, and then clear the box."""
143            # TODO: Render the expression
144            self.text_input_expression.setText('')
145
146        def animate_expression(self) -> None:
147            """Animate the expression in the input box, and then clear the box."""
148            # TODO: Animate the expression
149            self.text_input_expression.setText('')
150
151
152    def main() -> None:
153        """Run the GUI."""
154        app = QApplication(sys.argv)
155        window = LintransMainWindow()
156        window.show()
157        sys.exit(app.exec_())
158
159
160    if __name__ == '__main__':
161        main()
```
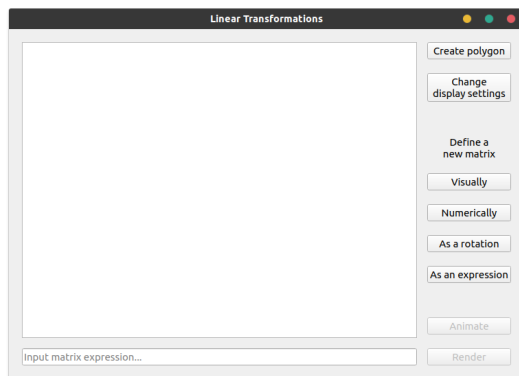

Figure 3.2.1: The first version of the GUI

A lot of the methods here don't have implementations yet, but they will. This version is just a very early prototype to get a rough draft of the GUI.

I create the widgets and layouts in the constructor as well as configuring all of them. The most important non-constructor method is `update_render_buttons()`. It gets called whenever the text in `text_input_expression` is changed. This happens because we connect it to the `textChanged` signal on line 32.

The big white box here will eventually be replaced with an actual viewport. This is just a prototype.

3.2.2 Numerical definition dialog

My next major addition was a dialog that would allow the user to define a matrix numerically.

```
# cedbd3ed126a1183f197c27adf6dabb4e5d301c7
# src/lintrans/gui/dialogs/define_new_matrix.py

1     """The module to provide dialogs for defining new matrices."""
2
3     from numpy import array
4     from PyQt5 import QtGui, QtWidgets
5     from PyQt5.QtWidgets import QDialog, QGridLayout, QHBoxLayout, QVBoxLayout
6
7     from lintrans.matrices import MatrixWrapper
8
9     ALPHABET_NO_I = 'ABCDEFGHJKLMNOPQRSTUVWXYZ'
10
11
12    def is_float(string: str) -> bool:
13        """Check if a string is a float."""
```

```python
14          try:
15              float(string)
16              return True
17          except ValueError:
18              return False
19
20
21  class DefineNumericallyDialog(QDialog):
22      """The dialog class that allows the user to define a new matrix numerically."""
23
24      def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
25          """Create the dialog, but don't run it yet.
26
27          :param matrix_wrapper: The MatrixWrapper that this dialog will mutate
28          :type matrix_wrapper: MatrixWrapper
29          """
30          super().__init__(*args, **kwargs)
31
32          self.matrix_wrapper = matrix_wrapper
33          self.setWindowTitle('Define a matrix')
34
35          # === Create the widgets
36
37          self.button_confirm = QtWidgets.QPushButton(self)
38          self.button_confirm.setText('Confirm')
39          self.button_confirm.setEnabled(False)
40          self.button_confirm.clicked.connect(self.confirm_matrix)
41          self.button_confirm.setToolTip('Confirm this as the new matrix<br><b>(Ctrl + Enter)</b>')
42
43          QtWidgets.QShortcut(QtGui.QKeySequence('Ctrl+Return'), self).activated.connect(self.button_confirm.click)
44
45          self.button_cancel = QtWidgets.QPushButton(self)
46          self.button_cancel.setText('Cancel')
47          self.button_cancel.clicked.connect(self.close)
48          self.button_cancel.setToolTip('Cancel this definition<br><b>(Ctrl + Q)</b>')
49
50          QtWidgets.QShortcut(QtGui.QKeySequence('Ctrl+Q'), self).activated.connect(self.button_cancel.click)
51
52          self.element_tl = QtWidgets.QLineEdit(self)
53          self.element_tl.textChanged.connect(self.update_confirm_button)
54
55          self.element_tr = QtWidgets.QLineEdit(self)
56          self.element_tr.textChanged.connect(self.update_confirm_button)
57
58          self.element_bl = QtWidgets.QLineEdit(self)
59          self.element_bl.textChanged.connect(self.update_confirm_button)
60
61          self.element_br = QtWidgets.QLineEdit(self)
62          self.element_br.textChanged.connect(self.update_confirm_button)
63
64          self.matrix_elements = (self.element_tl, self.element_tr, self.element_bl, self.element_br)
65
66          self.letter_combo_box = QtWidgets.QComboBox(self)
67
68          # Everything except I, because that's the identity
69          for letter in ALPHABET_NO_I:
70              self.letter_combo_box.addItem(letter)
71
72          self.letter_combo_box.activated.connect(self.load_matrix)
73
74          # === Arrange the widgets
75
76          self.setContentsMargins(10, 10, 10, 10)
77
78          self.grid_matrix = QGridLayout()
79          self.grid_matrix.setSpacing(20)
80          self.grid_matrix.addWidget(self.element_tl, 0, 0)
81          self.grid_matrix.addWidget(self.element_tr, 0, 1)
82          self.grid_matrix.addWidget(self.element_bl, 1, 0)
83          self.grid_matrix.addWidget(self.element_br, 1, 1)
84
85          self.hlay_buttons = QHBoxLayout()
86          self.hlay_buttons.setSpacing(20)
```

```
87              self.hlay_buttons.addWidget(self.button_cancel)
88              self.hlay_buttons.addWidget(self.button_confirm)
89
90              self.vlay_right = QVBoxLayout()
91              self.vlay_right.setSpacing(20)
92              self.vlay_right.addLayout(self.grid_matrix)
93              self.vlay_right.addLayout(self.hlay_buttons)
94
95              self.hlay_all = QHBoxLayout()
96              self.hlay_all.setSpacing(20)
97              self.hlay_all.addWidget(self.letter_combo_box)
98              self.hlay_all.addLayout(self.vlay_right)
99
100             self.setLayout(self.hlay_all)
101
102             # Finally, we load the default matrix A into the boxes
103             self.load_matrix(0)
104
105         def update_confirm_button(self) -> None:
106             """Enable the confirm button if there are numbers in every box."""
107             for elem in self.matrix_elements:
108                 if elem.text() == '' or not is_float(elem.text()):
109                     # If they're not all numbers, then we can't confirm it
110                     self.button_confirm.setEnabled(False)
111                     return
112
113             # If we didn't find anything invalid
114             self.button_confirm.setEnabled(True)
115
116         def load_matrix(self, index: int) -> None:
117             """If the selected matrix is defined, load it into the boxes."""
118             matrix = self.matrix_wrapper[ALPHABET_NO_I[index]]
119
120             if matrix is None:
121                 for elem in self.matrix_elements:
122                     elem.setText('')
123
124             else:
125                 self.element_tl.setText(str(matrix[0][0]))
126                 self.element_tr.setText(str(matrix[0][1]))
127                 self.element_bl.setText(str(matrix[1][0]))
128                 self.element_br.setText(str(matrix[1][1]))
129
130             self.update_confirm_button()
131
132         def confirm_matrix(self) -> None:
133             """Confirm the inputted matrix and assign it to the name."""
134             letter = self.letter_combo_box.currentText()
135             matrix = array([
136                 [float(self.element_tl.text()), float(self.element_tr.text())],
137                 [float(self.element_bl.text()), float(self.element_br.text())]
138             ])
139
140             self.matrix_wrapper[letter] = matrix
141             self.close()
```

When I add more definition dialogs, I will factor out a superclass, but this is just a prototype to make sure it all works as intended.

Hopefully the methods are relatively self explanatory, but they're just utility methods to update the GUI when things are changed. We connect the QLineEdit widgets to the update_confirm_button() slot to make sure the confirm button is always up to date.

Figure 3.2.2: The first version of the numerical definition dialog

The confirm_matrix() method just updates the instance's matrix wrapper with the new matrix. We

pass a reference to the `LintransMainWindow` instance's matrix wrapper when we open the dialog, so we're just updating the referenced object directly.

In the `LintransMainWindow` class, we're just connecting a lambda slot to the button so that it opens the dialog, as seen here:

```
# cedbd3ed126a1183f197c27adf6dabb4e5d301c7
# src/lintrans/gui/main_window.py

12   class LintransMainWindow(QMainWindow):
...
15       def __init__(self):
...
66           self.button_define_numerically.clicked.connect(
67               lambda: DefineNumericallyDialog(self.matrix_wrapper, self).exec()
68           )
```

3.2.3 More definition dialogs

I then factored out the constructor into a `DefineDialog` superclass so that I could easily create other definition dialogs.

```
# 5d04fb7233a03d0cd8fa0768f6387c6678da9df3
# src/lintrans/gui/dialogs/define_new_matrix.py

22   class DefineDialog(QDialog):
23       """A superclass for definitions dialogs."""
24
25       def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
26           """Create the dialog, but don't run it yet.
27
28           :param matrix_wrapper: The MatrixWrapper that this dialog will mutate
29           :type matrix_wrapper: MatrixWrapper
30           """
31           super().__init__(*args, **kwargs)
32
33           self.matrix_wrapper = matrix_wrapper
34           self.setWindowTitle('Define a matrix')
35
36           # === Create the widgets
37
38           self.button_confirm = QtWidgets.QPushButton(self)
39           self.button_confirm.setText('Confirm')
40           self.button_confirm.setEnabled(False)
41           self.button_confirm.clicked.connect(self.confirm_matrix)
42           self.button_confirm.setToolTip('Confirm this as the new matrix<br><b>(Ctrl + Enter)</b>')
43           QShortcut(QKeySequence('Ctrl+Return'), self).activated.connect(self.button_confirm.click)
44
45           self.button_cancel = QtWidgets.QPushButton(self)
46           self.button_cancel.setText('Cancel')
47           self.button_cancel.clicked.connect(self.close)
48           self.button_cancel.setToolTip('Cancel this definition<br><b>(Ctrl + Q)</b>')
49           QShortcut(QKeySequence('Ctrl+Q'), self).activated.connect(self.button_cancel.click)
50
51           self.label_equals = QtWidgets.QLabel()
52           self.label_equals.setText('=')
53
54           self.letter_combo_box = QtWidgets.QComboBox(self)
55
56           # Everything except I, because that's the identity
57           for letter in ALPHABET_NO_I:
58               self.letter_combo_box.addItem(letter)
59
60           self.letter_combo_box.activated.connect(self.load_matrix)
```

This superclass just has a constructor that subclasses can use. When I added the `DefineAsARotationDialog`

class, I also moved the cancel and confirm buttons into the constructor and added abstract methods that all dialog subclasses must implement.

```python
# 0d534c35c6a4451e317d41a0d2b3ecb17827b45f
# src/lintrans/gui/dialogs/define_new_matrix.py

24    class DefineDialog(QDialog):
...
27        def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
...
61            # === Arrange the widgets
62
63            self.setContentsMargins(10, 10, 10, 10)
64
65            self.horizontal_spacer = QSpacerItem(50, 5, hPolicy=QSizePolicy.Expanding, vPolicy=QSizePolicy.Minimum)
66
67            self.hlay_buttons = QHBoxLayout()
68            self.hlay_buttons.setSpacing(20)
69            self.hlay_buttons.addItem(self.horizontal_spacer)
70            self.hlay_buttons.addWidget(self.button_cancel)
71            self.hlay_buttons.addWidget(self.button_confirm)
72
73        @property
74        def selected_letter(self) -> str:
75            """The letter currently selected in the combo box."""
76            return self.letter_combo_box.currentText()
77
78        @abc.abstractmethod
79        def update_confirm_button(self) -> None:
80            """Enable the confirm button if it should be enabled."""
81            ...
82
83        @abc.abstractmethod
84        def confirm_matrix(self) -> None:
85            """Confirm the inputted matrix and assign it.
86
87            This should mutate self.matrix_wrapper and then call self.accept().
88            """
89            ...
```

I then added the class for the rotation definition dialog.

```python
# 0d534c35c6a4451e317d41a0d2b3ecb17827b45f
# src/lintrans/gui/dialogs/define_new_matrix.py

182   class DefineAsARotationDialog(DefineDialog):
183       """The dialog that allows the user to define a new matrix as a rotation."""
184
185       def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
186           """Create the dialog, but don't run it yet."""
187           super().__init__(matrix_wrapper, *args, **kwargs)
188
189           # === Create the widgets
190
191           self.label_equals.setText('= rot(')
192
193           self.text_angle = QtWidgets.QLineEdit(self)
194           self.text_angle.setPlaceholderText('angle')
195           self.text_angle.textChanged.connect(self.update_confirm_button)
196
197           self.label_close_paren = QtWidgets.QLabel(self)
198           self.label_close_paren.setText(')')
199
200           self.checkbox_radians = QtWidgets.QCheckBox(self)
201           self.checkbox_radians.setText('Radians')
202
203           # === Arrange the widgets
204
205           self.hlay_checkbox_and_buttons = QHBoxLayout()
206           self.hlay_checkbox_and_buttons.setSpacing(20)
```

```
207            self.hlay_checkbox_and_buttons.addWidget(self.checkbox_radians)
208            self.hlay_checkbox_and_buttons.addItem(self.horizontal_spacer)
209            self.hlay_checkbox_and_buttons.addLayout(self.hlay_buttons)
210
211            self.hlay_definition = QHBoxLayout()
212            self.hlay_definition.addWidget(self.letter_combo_box)
213            self.hlay_definition.addWidget(self.label_equals)
214            self.hlay_definition.addWidget(self.text_angle)
215            self.hlay_definition.addWidget(self.label_close_paren)
216
217            self.vlay_all = QVBoxLayout()
218            self.vlay_all.setSpacing(20)
219            self.vlay_all.addLayout(self.hlay_definition)
220            self.vlay_all.addLayout(self.hlay_checkbox_and_buttons)
221
222            self.setLayout(self.vlay_all)
223
224        def update_confirm_button(self) -> None:
225            """Enable the confirm button if there is a valid float in the angle box."""
226            self.button_confirm.setEnabled(is_float(self.text_angle.text()))
227
228        def confirm_matrix(self) -> None:
229            """Confirm the inputted matrix and assign it."""
230            self.matrix_wrapper[self.selected_letter] = create_rotation_matrix(
231                float(self.text_angle.text()),
232                degrees=not self.checkbox_radians.isChecked()
233            )
234            self.accept()
```

This dialog class just overrides the abstract methods of the superclass with its own implementations. This will be the pattern that all of the definition dialogs will follow.

It has a checkbox for radians, since this is supported in `create_rotation_matrix()`, but the textbox only supports numbers, so the user would have to calculate some multiple of π and paste in several decimal places. I expect people to only use degrees, because these are easier to use.
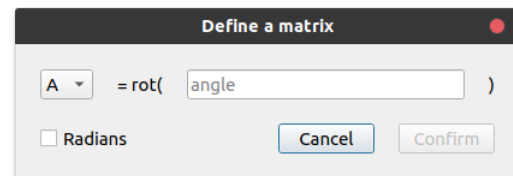
Figure 3.2.3: The first version of the rotation definition dialog

Additionally, I created a helper method in `LintransMainWindow`. Rather than connecting the `clicked` signal of the buttons to lambdas that instantiate an instance of the `DefineDialog` subclass and call `.exec()` on it, I now connect the `clicked` signal of the buttons to lambdas that call `self.dialog_define_matrix()` with the specific subclass.

```
     # 6269e04d453df7be2d2f9c7ee176e83406ccc139
     # src/lintrans/gui/main_window.py

17   class LintransMainWindow(QMainWindow):
...
170      def dialog_define_matrix(self, dialog_class: Type[DefineDialog]) -> None:
171          """Open a generic definition dialog to define a new matrix.
172
173          The class for the desired dialog is passed as an argument. We create an
174          instance of this class and the dialog is opened asynchronously and modally
175          (meaning it blocks interaction with the main window) with the proper method
176          connected to the ``dialog.finished`` slot.
177
178          .. note::
179              ``dialog_class`` must subclass :class:`lintrans.gui.dialogs.define_new_matrix.DefineDialog`.
180
181          :param dialog_class: The dialog class to instantiate
182          :type dialog_class: Type[lintrans.gui.dialogs.define_new_matrix.DefineDialog]
183          """
184          # We create a dialog with a deepcopy of the current matrix_wrapper
185          # This avoids the dialog mutating this one
```

```
186              dialog = dialog_class(deepcopy(self.matrix_wrapper), self)
187
188              # .open() is asynchronous and doesn't spawn a new event loop, but the dialog is still modal (blocking)
189              dialog.open()
190
191              # So we have to use the finished slot to call a method when the user accepts the dialog
192              # If the user rejects the dialog, this matrix_wrapper will be the same as the current one, because we copied
        ↪  it
193              # So we don't care, we just assign the wrapper anyway
194              dialog.finished.connect(lambda: self._assign_matrix_wrapper(dialog.matrix_wrapper))
195
196          def _assign_matrix_wrapper(self, matrix_wrapper: MatrixWrapper) -> None:
197              """Assign a new value to self.matrix_wrapper.
198
199              This is a little utility function that only exists because a lambda
200              callback can't directly assign a value to a class attribute.
201
202              :param matrix_wrapper: The new value of the matrix wrapper to assign
203              :type matrix_wrapper: MatrixWrapper
204              """
205              self.matrix_wrapper = matrix_wrapper
```

I also then implemented a simple `DefineAsAnExpressionDialog`, which evaluates a given expression in the current `MatrixWrapper` context and assigns the result to the given matrix name.

```
        # d5f930e15c3c8798d4990486532da46e926a6cb9
        # src/lintrans/gui/dialogs/define_new_matrix.py

241  class DefineAsAnExpressionDialog(DefineDialog):
242      """The dialog that allows the user to define a matrix as an expression."""
243
244      def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
245          """Create the dialog, but don't run it yet."""
246          super().__init__(matrix_wrapper, *args, **kwargs)
247
248          self.setMinimumWidth(450)
249
250          # === Create the widgets
251
252          self.text_box_expression = QtWidgets.QLineEdit(self)
253          self.text_box_expression.setPlaceholderText('Enter matrix expression...')
254          self.text_box_expression.textChanged.connect(self.update_confirm_button)
255
256          # === Arrange the widgets
257
258          self.hlay_definition.addWidget(self.text_box_expression)
259
260          self.vlay_all = QVBoxLayout()
261          self.vlay_all.setSpacing(20)
262          self.vlay_all.addLayout(self.hlay_definition)
263          self.vlay_all.addLayout(self.hlay_buttons)
264
265          self.setLayout(self.vlay_all)
266
267      def update_confirm_button(self) -> None:
268          """Enable the confirm button if the expression is valid."""
269          self.button_confirm.setEnabled(
270              self.matrix_wrapper.is_valid_expression(self.text_box_expression.text())
271          )
272
273      def confirm_matrix(self) -> None:
274          """Evaluate the matrix expression and assign its value to the chosen matrix."""
275          self.matrix_wrapper[self.selected_letter] = \
276              self.matrix_wrapper.evaluate_expression(self.text_box_expression.text())
277          self.accept()
```

My next dialog that I wanted to implement was a visual definition dialog, which would allow the user to drag around the basis vectors to define a transformation. However, I would first need to create the `lintrans.gui.plots` package to allow for actually visualizing matrices and transformations.

3.3   Visualizing matrices

3.3.1   Asking strangers on the internet for help

After creating most of the GUI skeleton, I wanted to build the viewport. Unfortunately, I had no idea what I was doing.

While looking through the PyQt5 docs, I found a pretty comprehensive explanation of the Qt5 'Graphics View Framework'[37], which seemed pretty good, but not really what I was looking for. I wanted a way to easily draw lots of straight, parallel lines. This framework seemed more focussed on manipulating objects on a canvas, almost like sprites. I knew of a different Python library called `matplotlib`, which has various backends available. I learned that it could be embedded in a standard PyQt5 GUI, so I started doing some research.

I didn't get very far with `matplotlib`. I hadn't used it much before and it's designed for visualizing data. It can draw manually defined straight lines on a canvas, but that's not what it's designed for and it's not very good at it. Thankfully, my horrific `matplotlib` code has been lost to time. I used the `Qt5Agg` backend from `matplotlib` to create a custom PyQt5 widget for the GUI and I could graph randomly generated data with it after following a tutorial[33].

I realised that I wasn't going to get very far with `matplotlib`, but I didn't know what else to do. I couldn't find any relevant examples on the internet, so I decided to post a question on a forum myself. I'd had experience with StackOverflow and its unfriendly community before, so I decided to ask the `r/learnpython` subreddit[4].

I only got one response, but it was incredibly helpful. The person told me that if I couldn't find an easy way to do what I wanted, I could write a custom PyQt5 widget. I knew this was possible with a class that just inherited from `QWidget`, but had no idea how to actually make something useful. Thankfully, this person provided a link to a GitLab repository of theirs, where they had multiple examples of custom widgets with PyQt5[5].

When looking through this repo, I found out how to draw on a widget like a simple canvas. All I have to do is override the `paintEvent()` method and use a `QPainter` object to draw on the widget. I used this knowledge to start creating the actual viewport for the GUI, starting with the background axes.

3.3.2   Creating the plots package

Initially, the `lintrans.gui.plots` package just has some classes for widgets. `TransformationPlotWidget` acts as a base class and then `ViewTransformationWidget` acts as a wrapper. I will expand this class in the future.

```
# 4af63072b383dc9cef9adbb8900323aa007e7f26
# src/lintrans/gui/plots/plot_widget.py

1    """This module provides the basic classes for plotting transformations."""
2
3    from __future__ import annotations
4
5    from PyQt5.QtCore import Qt
6    from PyQt5.QtGui import QColor, QPainter, QPaintEvent, QPen
7    from PyQt5.QtWidgets import QWidget
8
9
10   class TransformationPlotWidget(QWidget):
11       """An abstract superclass for plot widgets.
12
13       This class provides a background (untransformed) plane, and all the backend
14       details for a Qt application, but does not provide useful functionality. To
15       be useful, this class must be subclassed and behaviour must be implemented
16       by the subclass.
```

```python
17
18              .. warning:: This class should never be directly instantiated, only subclassed.
19
20              .. note::
21                 I would make this class have ``metaclass=abc.ABCMeta``, but I can't because it subclasses ``QWidget``,
22                 and a every superclass of a class must have the same metaclass, and ``QWidget`` is not an abstract class.
23              """
24
25          def __init__(self, *args, **kwargs):
26              """Create the widget, passing ``*args`` and ``**kwargs`` to the superclass constructor (``QWidget``)."""
27              super().__init__(*args, **kwargs)
28
29              self.setAutoFillBackground(True)
30
31              # Set the background to white
32              palette = self.palette()
33              palette.setColor(self.backgroundRole(), Qt.white)
34              self.setPalette(palette)
35
36              # Set the gird colour to grey and the axes colour to black
37              self.grid_colour = QColor(128, 128, 128)
38              self.axes_colour = QColor(0, 0, 0)
39
40              self.grid_spacing: int = 50
41              self.line_width: float = 0.4
42
43          @property
44          def w(self) -> int:
45              """Return the width of the widget."""
46              return self.size().width()
47
48          @property
49          def h(self) -> int:
50              """Return the height of the widget."""
51              return self.size().height()
52
53          def paintEvent(self, e: QPaintEvent):
54              """Handle a ``QPaintEvent`` by drawing the widget."""
55              qp = QPainter()
56              qp.begin(self)
57              self.draw_widget(qp)
58              qp.end()
59
60          def draw_widget(self, qp: QPainter):
61              """Draw the grid and axes in the widget."""
62              qp.setRenderHint(QPainter.Antialiasing)
63              qp.setBrush(Qt.NoBrush)
64
65              # Draw the grid
66              qp.setPen(QPen(self.grid_colour, self.line_width))
67
68              # We draw the background grid, centered in the middle
69              # We deliberately exclude the axes - these are drawn separately
70              for x in range(self.w // 2 + self.grid_spacing, self.w, self.grid_spacing):
71                  qp.drawLine(x, 0, x, self.h)
72                  qp.drawLine(self.w - x, 0, self.w - x, self.h)
73
74              for y in range(self.h // 2 + self.grid_spacing, self.h, self.grid_spacing):
75                  qp.drawLine(0, y, self.w, y)
76                  qp.drawLine(0, self.h - y, self.w, self.h - y)
77
78              # Now draw the axes
79              qp.setPen(QPen(self.axes_colour, self.line_width))
80              qp.drawLine(self.w // 2, 0, self.w // 2, self.h)
81              qp.drawLine(0, self.h // 2, self.w, self.h // 2)
82
83
84  class ViewTransformationWidget(TransformationPlotWidget):
85      """This class is used to visualise matrices as transformations."""
86
87          def __init__(self, *args, **kwargs):
88              """Create the widget, passing ``*args`` and ``**kwargs`` to the superclass constructor."""
89              super().__init__(*args, **kwargs)
```
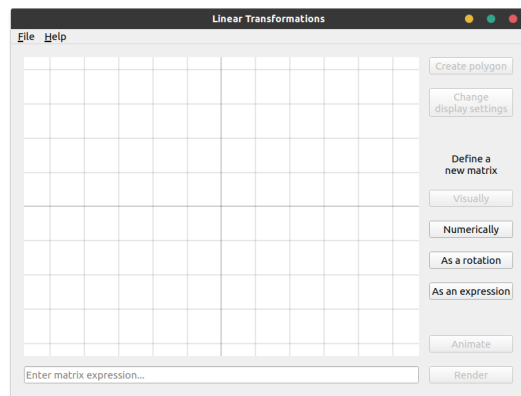
Figure 3.3.1: The GUI with background axes

The meat of this class is the `draw_widget()` method. Right now, this method only draws the background axes. My next step is to implement basis vector attributes and draw them in `draw_widget()`. After changing the the `plot` attribute in `LintransMainWindow` to an instance of `ViewTransformationWidget`, the plot was visible in the GUI.

I then refactored the code slightly to rename `draw_widget()` to `draw_background()` and then call it from the `paintEvent()` method in `ViewTransformationWidget`.

3.3.3   Implementing basis vectors

My first step in implementing basis vectors was to add some utility methods to convert between coordinate systems. The matrices are using Cartesian coordinates with $(0, 0)$ in the middle, positive x going to the right, and positive y going up. However, Qt5 is using standard computer graphics coordinates, with $(0, 0)$ in the top left, positive x going to the right, and positive y going down. I needed a way to convert Cartesian 'grid' coordinates to Qt5 'canvas' coordinates, so I wrote some little utility methods.

```python
# 1fa7e1c61d61cb6aeff773b9698541f82fee39ea
# src/lintrans/gui/plots/plot_widget.py

12  class TransformationPlotWidget(QWidget):
...
45      @property
46      def origin(self) -> tuple[int, int]:
47          """Return the canvas coords of the origin."""
48          return self.width() // 2, self.height() // 2
49
50      def trans_x(self, x: float) -> int:
51          """Transform an x coordinate from grid coords to canvas coords."""
52          return int(self.origin[0] + x * self.grid_spacing)
53
54      def trans_y(self, y: float) -> int:
55          """Transform a y coordinate from grid coords to canvas coords."""
56          return int(self.origin[1] - y * self.grid_spacing)
57
58      def trans_coords(self, x: float, y: float) -> tuple[int, int]:
59          """Transform a coordinate in grid coords to canvas coords."""
60          return self.trans_x(x), self.trans_y(y)
```

Once I had a way to convert coordinates, I could add the basis vectors themselves. I did this by creating attributes for the points in the constructor and creating a `transform_by_matrix()` method to change these point attributes accordingly.

```python
# 37e7c208a33d7cbbc8e0bb6c94cd889e2918c605
# src/lintrans/gui/plots/plot_widget.py

92  class ViewTransformationWidget(TransformationPlotWidget):
93      """This class is used to visualise matrices as transformations."""
94
95      def __init__(self, *args, **kwargs):
96          """Create the widget, passing ``*args`` and ``**kwargs`` to the superclass constructor."""
97          super().__init__(*args, **kwargs)
98
99          self.point_i: tuple[float, float] = (1., 0.)
100         self.point_j: tuple[float, float] = (0., 1.)
```

```
101
102            self.colour_i = QColor(37, 244, 15)
103            self.colour_j = QColor(8, 8, 216)
104
105            self.width_vector_line = 1
106            self.width_transformed_grid = 0.6
107
108        def transform_by_matrix(self, matrix: MatrixType) -> None:
109            """Transform the plane by the given matrix."""
110            self.point_i = (matrix[0][0], matrix[1][0])
111            self.point_j = (matrix[0][1], matrix[1][1])
112            self.update()
```

I also created a `draw_transformed_grid()` method which gets called in `paintEvent()`.

```
# 37e7c208a33d7cbbc8e0bb6c94cd889e2918c605
# src/lintrans/gui/plots/plot_widget.py

92    class ViewTransformationWidget(TransformationPlotWidget):
...
122        def draw_transformed_grid(self, painter: QPainter) -> None:
123            """Draw the transformed version of the grid, given by the unit vectors."""
124            # Draw the unit vectors
125            painter.setPen(QPen(self.colour_i, self.width_vector_line))
126            painter.drawLine(*self.origin, *self.trans_coords(*self.point_i))
127            painter.setPen(QPen(self.colour_j, self.width_vector_line))
128            painter.drawLine(*self.origin, *self.trans_coords(*self.point_j))
```

I then changed the `render_expression()` method in `LintransMainWindow` to call this new `transform_by_matrix()` method.

```
# 37e7c208a33d7cbbc8e0bb6c94cd889e2918c605
# src/lintrans/gui/main_window.py

19    class LintransMainWindow(QMainWindow):
...
229        def render_expression(self) -> None:
230            """Render the expression in the input box, and then clear the box."""
231            self.plot.transform_by_matrix(
232                self.matrix_wrapper.evaluate_expression(
233                    self.lineedit_expression_box.text()
234                )
235            )
```

Testing this new code shows that it works well.
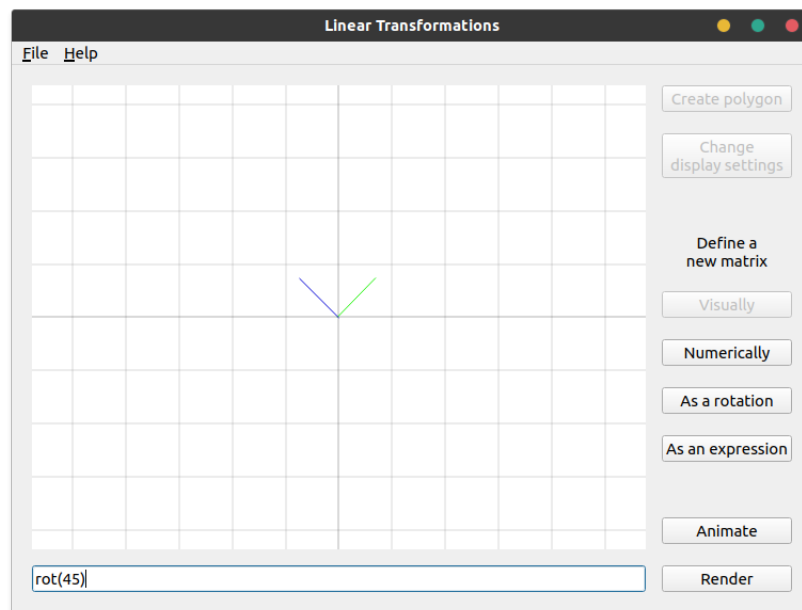
Figure 3.3.2: Basis vectors drawn for a 45° rotation

3.3.4 Drawing the transformed grid

After drawing the basis vectors, I wanted to draw the transformed version of the grid. I first created a `grid_corner()` utility method to return the grid coordinates of the top right corner of the canvas. This allows me to find the bounding box in which to draw the grid lines.

```
# 2ade98ac28d1c3f6691e4afa819142a3ab8e9fd9
# src/lintrans/gui/plots/plot_widget.py

14    class TransformationPlotWidget(QWidget):
...
64        def grid_corner(self) -> tuple[float, float]:
65            """Return the grid coords of the top right corner."""
66            return self.width() / (2 * self.grid_spacing), self.height() / (2 * self.grid_spacing)
```

I then created a `draw_parallel_lines()` method that would fill the bounding box with a set of lines parallel to a given vector with spacing defined by the intersection with a given point.

```
# 2ade98ac28d1c3f6691e4afa819142a3ab8e9fd9
# src/lintrans/gui/plots/plot_widget.py

96     class ViewTransformationWidget(TransformationPlotWidget):
...
126        def draw_parallel_lines(self, painter: QPainter, vector: tuple[float, float], point: tuple[float, float]) ->
     ↪      None:
127            """Draw a set of grid lines parallel to ``vector`` intersecting ``point``."""
128            max_x, max_y = self.grid_corner()
129            vector_x, vector_y = vector
130            point_x, point_y = point
131
132            if vector_x == 0:
133                painter.drawLine(self.trans_x(0), 0, self.trans_x(0), self.height())
134
135                for i in range(int(max_x / point_x)):
136                    painter.drawLine(
137                        self.trans_x((i + 1) * point_x),
138                        0,
139                        self.trans_x((i + 1) * point_x),
```

```
140                        self.height()
141                    )
142                painter.drawLine(
143                    self.trans_x(-1 * (i + 1) * point_x),
144                    0,
145                    self.trans_x(-1 * (i + 1) * point_x),
146                    self.height()
147                )
148
149        elif vector_y == 0:
150            painter.drawLine(0, self.trans_y(0), self.width(), self.trans_y(0))
151
152            for i in range(int(max_y / point_y)):
153                painter.drawLine(
154                    0,
155                    self.trans_y((i + 1) * point_y),
156                    self.width(),
157                    self.trans_y((i + 1) * point_y)
158                )
159                painter.drawLine(
160                    0,
161                    self.trans_y(-1 * (i + 1) * point_y),
162                    self.width(),
163                    self.trans_y(-1 * (i + 1) * point_y)
164                )
```

I then called this method from `draw_transformed_grid()`.

```
# 2ade98ac28d1c3f6691e4afa819142a3ab8e9fd9
# src/lintrans/gui/plots/plot_widget.py

96   class ViewTransformationWidget(TransformationPlotWidget):
...
166      def draw_transformed_grid(self, painter: QPainter) -> None:
167          """Draw the transformed version of the grid, given by the unit vectors."""
168          # Draw the unit vectors
169          painter.setPen(QPen(self.colour_i, self.width_vector_line))
170          painter.drawLine(*self.origin, *self.trans_coords(*self.point_i))
171          painter.setPen(QPen(self.colour_j, self.width_vector_line))
172          painter.drawLine(*self.origin, *self.trans_coords(*self.point_j))
173
174          # Draw all the parallel lines
175          painter.setPen(QPen(self.colour_i, self.width_transformed_grid))
176          self.draw_parallel_lines(painter, self.point_i, self.point_j)
177          painter.setPen(QPen(self.colour_j, self.width_transformed_grid))
178          self.draw_parallel_lines(painter, self.point_j, self.point_i)
```

This worked quite well when the matrix involved no rotation, as seen on the right, but this didn't work with rotation. When trying `'rot(45)'` for example, it looked the same as in Figure 3.3.2.

Also, the vectors aren't particularly clear. They'd be much better with arrowheads on their tips, but this is just a prototype. The arrowheads will come later.

My next step was to make the transformed grid lines work with rotations.
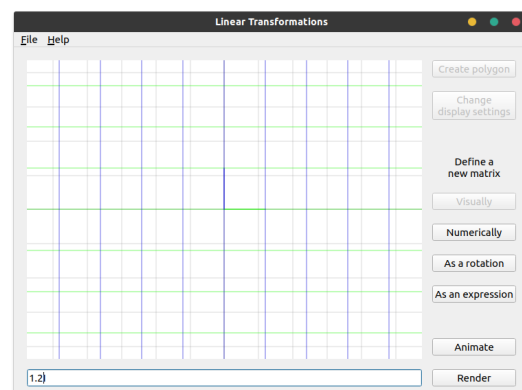
Figure 3.3.3: Parallel lines being drawn for matrix 1.2**I**

```
# 7dfe1e24729562501e2fd88a839dca6b653a3375
# src/lintrans/gui/plots/plot_widget.py
```

```
 96      class ViewTransformationWidget(TransformationPlotWidget):
...
126          def draw_parallel_lines(self, painter: QPainter, vector: tuple[float, float], point: tuple[float, float]) ->
      ↪      None:
127              """Draw a set of grid lines parallel to ``vector`` intersecting ``point``."""
128              max_x, max_y = self.grid_corner()
129              vector_x, vector_y = vector
130              point_x, point_y = point
131
132              print(max_x, max_y, vector_x, vector_y, point_x, point_y)
133
134              # We want to use y = mx + c but m = y / x and if either of those are 0, then this
135              # equation is harder to work with, so we deal with these edge cases first
136              if abs(vector_x) < 1e-12 and abs(vector_y) < 1e-12:
137                  # If both components of the vector are practically 0, then we can't render any grid lines
138                  return
139
140              elif abs(vector_x) < 1e-12:
141                  painter.drawLine(self.trans_x(0), 0, self.trans_x(0), self.height())
142
143                  for i in range(abs(int(max_x / point_x))):
144                      painter.drawLine(
145                          self.trans_x((i + 1) * point_x),
146                          0,
147                          self.trans_x((i + 1) * point_x),
148                          self.height()
149                      )
150                      painter.drawLine(
151                          self.trans_x(-1 * (i + 1) * point_x),
152                          0,
153                          self.trans_x(-1 * (i + 1) * point_x),
154                          self.height()
155                      )
156
157              elif abs(vector_y) < 1e-12:
158                  painter.drawLine(0, self.trans_y(0), self.width(), self.trans_y(0))
159
160                  for i in range(abs(int(max_y / point_y))):
161                      painter.drawLine(
162                          0,
163                          self.trans_y((i + 1) * point_y),
164                          self.width(),
165                          self.trans_y((i + 1) * point_y)
166                      )
167                      painter.drawLine(
168                          0,
169                          self.trans_y(-1 * (i + 1) * point_y),
170                          self.width(),
171                          self.trans_y(-1 * (i + 1) * point_y)
172                      )
173
174              else:  # If the line is not horizontal or vertical, then we can use y = mx + c
175                  m = vector_y / vector_x
176                  c = point_y - m * point_x
177
178                  # For c = 0
179                  painter.drawLine(
180                      *self.trans_coords(
181                          -1 * max_x,
182                          m * -1 * max_x
183                      ),
184                      *self.trans_coords(
185                          max_x,
186                          m * max_x
187                      )
188                  )
189
190                  # Count up how many multiples of c we can have without wasting time rendering lines off screen
191                  multiples_of_c: int = 0
192                  ii: int = 1
193                  while True:
194                      y1 = m * max_x + ii * c
```

```
195                    y2 = -1 * m * max_x + ii * c
196
197                    if y1 < max_y or y2 < max_y:
198                        multiples_of_c += 1
199                        ii += 1
200
201                    else:
202                        break
203
204                # Once we know how many lines we can draw, we just draw them all
205                for i in range(1, multiples_of_c + 1):
206                    painter.drawLine(
207                        *self.trans_coords(
208                            -1 * max_x,
209                            m * -1 * max_x + i * c
210                        ),
211                        *self.trans_coords(
212                            max_x,
213                            m * max_x + i * c
214                        )
215                    )
216                    painter.drawLine(
217                        *self.trans_coords(
218                            -1 * max_x,
219                            m * -1 * max_x - i * c
220                        ),
221                        *self.trans_coords(
222                            max_x,
223                            m * max_x - i * c
224                        )
225                    )
```

This code checks if x or y is zero[10] and if they're not, then we have to use the standard straight line equation $y = mx + c$ to create parallel lines. We find our value of m and then iterate through all the values of c that keep the line within the bounding box.
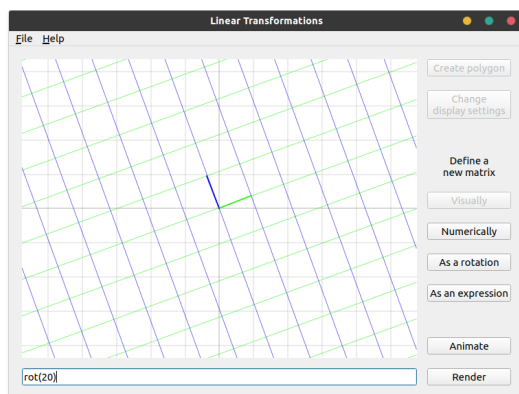
Figure 3.3.4: An example of a 20° rotation

There are some serious logical errors in this code. It works fine for things like `'3rot(45)'` or `'0.5rot(⌐20)'`, but something like `'rot(115)'` will leave the program hanging indefinitely.

In fact, this code only works for rotations between 0° and 90°, and will hang forever when given a matrix like $\left(\begin{smallmatrix} 12 & 4 \\ -2 & 3 \end{smallmatrix} \right)$, because it's just not very good.

I will fix these issues in the future, but it works somewhat decently, so I decided to do animation next, because that sounded more fun.

3.3.5   Implementing animation

Now that I had a very crude renderer, I could create a method to animate a matrix. Eventually I want to be able to apply a given matrix to the currently rendered scene and animate between them. However, I wanted to start simple by animating from the identity to the given matrix.

```
# 829a130af5aee9819bf0269c03ecfb20bec1a108
# src/lintrans/gui/main_window.py

20   class LintransMainWindow(QMainWindow):
```

---
[10]We actually check if they're less than 10^{-12} to allow for floating point errors

```
...
238        def animate_expression(self) -> None:
239            """Animate the expression in the input box, and then clear the box."""
240            self.button_render.setEnabled(False)
241            self.button_animate.setEnabled(False)
242
243            matrix = self.matrix_wrapper.evaluate_expression(self.lineedit_expression_box.text())
244            matrix_move = matrix - self.matrix_wrapper['I']
245            steps: int = 100
246
247            for i in range(0, steps + 1):
248                self.plot.visualize_matrix_transformation(
249                    self.matrix_wrapper['I'] + (i / steps) * matrix_move
250                )
251
252                self.update()
253                self.repaint()
254
255                time.sleep(0.01)
256
257            self.button_render.setEnabled(False)
258            self.button_animate.setEnabled(False)
```

This code creates the `matrix_move` variable and adds scaled versions of it to the identity matrix and renders that each frame. It's simple, but it works well for this simple use case. Unfortunately, it's very hard to show off an animation in a PDF, since all these images are static. The git commit hashes are included in the code snippets if you want to clone the repo[3], checkout this commit, and run it yourself if you want.

3.3.6 Preserving determinants

Ignoring the obvious flaw with not being able to render transformations with a more than 90° rotation, the animations don't respect determinants. When rotating 90°, the determinant changes during the animation, even though we're going from a determinant 1 matrix (the identity) to another determinant 1 matrix. This is because we're just moving each vector to its new position in a straight line. I want to animate in a way that smoothly transitions the determinant.
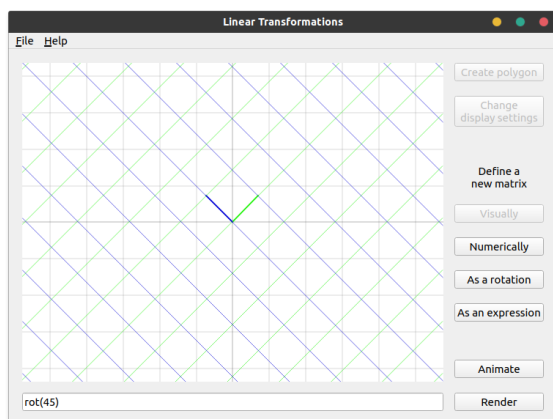
Figure 3.3.5: What we would expect halfway through a 90° rotation
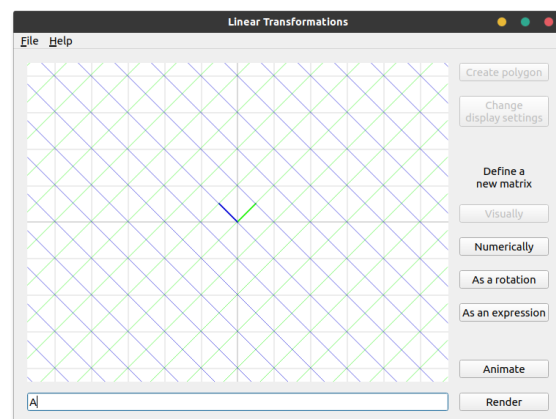
Figure 3.3.6: What we actually get halfway through a 90° rotation

In order to smoothly animate the determinant, I had to do some maths. I first defined the matrix $\mathbf{A}$ to be equivalent to the `matrix_move` variable from before - the target matrix minus the identity, scaled by the proportion. I then wanted to normalize $\mathbf{A}$ so that it had a determinant of 1 so that I could scale it up with the `proportion` variable through the animation.

I think I first tried just multiplying $\mathbf{A}$ by $\dfrac{1}{\det(\mathbf{A})}$ but that didn't work, so I googled it. I found a

post[29] on ResearchGate about the topic, and thanks to a very helpful comment from Jeffrey L Stuart, I learned that for a 2×2 matrix $\mathbf{A}$ and a scalar c, $\det(c\mathbf{A}) = c^2 \det(\mathbf{A})$.

I wanted a c such that $\det(c\mathbf{A}) = 1$. Therefore $c = \dfrac{1}{\sqrt{|\det(\mathbf{A})|}}$. I then defined matrix $\mathbf{B}$ to be $c\mathbf{A}$.

Then I wanted to scale this normalized matrix $\mathbf{B}$ to have the same determinant as the target matrix $\mathbf{T}$ using some scalar d. We know that $\det(d\mathbf{B}) = d^2 \det(\mathbf{B}) = \det(\mathbf{T})$. We can just rearrange to find d and get $d = \sqrt{\left|\dfrac{\det(\mathbf{T})}{\det(\mathbf{B})}\right|}$. But $\mathbf{B}$ is defined so that $\det(\mathbf{B}) = 1$, so we can get $d = \sqrt{|\det(\mathbf{T})|}$.

However, we want to scale this over time with our `proportion` variable p, so our final scalar $s = 1 + p\left(\sqrt{|\det(\mathbf{T})|} - 1\right)$. We define a matrix $\mathbf{C} = s\mathbf{B}$ and render $\mathbf{C}$ each frame. When in code form, this is the following:

```python
# 6ff49450d8438ea2b2e7d2a97125dc518e648bc5
# src/lintrans/gui/main_window.py

22      class LintransMainWindow(QMainWindow):
...
240         def animate_expression(self) -> None:
...
245             # Get the target matrix and it's determinant
246             matrix_target = self.matrix_wrapper.evaluate_expression(self.lineedit_expression_box.text())
247             det_target = linalg.det(matrix_target)
248
249             identity = self.matrix_wrapper['I']
250             steps: int = 100
251
252             for i in range(0, steps + 1):
253                 # This proportion is how far we are through the loop
254                 proportion = i / steps
255
256                 # matrix_a is the identity plus some part of the target, scaled by the proportion
257                 # If we just used matrix_a, then things would animate, but the determinants would be weird
258                 matrix_a = identity + proportion * (matrix_target - identity)
259
260                 # So to fix the determinant problem, we get the determinant of matrix_a and use it to normalise
261                 det_a = linalg.det(matrix_a)
262
263                 # For a 2x2 matrix A and a scalar c, we know that det(cA) = c^2 det(A)
264                 # We want B = cA such that det(B) = 1, so then we can scale it with the animation
265                 # So we get c^2 det(A) = 1 => c = sqrt(1 / abs(det(A)))
266                 # Then we scale A down to get a determinant of 1, and call that matrix_b
267                 if det_a == 0:
268                     c = 0
269                 else:
270                     c = np.sqrt(1 / abs(det_a))
271
272                 matrix_b = c * matrix_a
273
274                 # matrix_c is the final matrix that we transform by
275                 # It's B, but we scale it up over time to have the target determinant
276
277                 # We want some C = dB such that det(C) is some target determinant T
278                 # det(dB) = d^2 det(B) = T => d = sqrt(abs(T / det(B)))
279                 # But we defined B to have det 1, so we can ignore it there
280
281                 # We're also subtracting 1 and multiplying by the proportion and then adding one
282                 # This just scales the determinant along with the animation
283                 scalar = 1 + proportion * (np.sqrt(abs(det_target)) - 1)
284
285                 matrix_c = scalar * matrix_b
286
287                 self.plot.visualize_matrix_transformation(matrix_c)
288
289                 self.repaint()
290                 time.sleep(0.01)
```

Unfortunately, the system I use to render matrices is still quite bad at its job. This makes it hard to test properly. But, transformations like `'2rot(90)'` work exactly as expected, which is very good.

3.4   Improving the GUI

3.4.1   Fixing rendering

Now that I had the basics of matrix visualization sorted, I wanted to make the GUI and UX better. My first step was overhauling the rendering code to make it actually work with rotations of more than 90°.

I narrowed down the issue with PyCharm's debugger and found that the loop in `VectorGridPlot.draw_parallel_lines()` was looping forever if it tried to doing anything outside of the top right quadrant. To fix this, I decided to instead delegate this task of drawing a set of oblique lines to a separate method, and work on that instead.

```
     # cf05e09e5ebb6ea7a96db8660d0d8de6b946490a
     # src/lintrans/gui/plots/classes.py

118  class VectorGridPlot(BackgroundPlot):
...
150      def draw_parallel_lines(self, painter: QPainter, vector: tuple[float, float], point: tuple[float, float]) ->
     ↪   None:
...
203          else:  # If the line is not horizontal or vertical, then we can use y = mx + c
204              m = vector_y / vector_x
205              c = point_y - m * point_x
206
207              # For c = 0
208              painter.drawLine(
209                  *self.trans_coords(
210                      -1 * max_x,
211                      m * -1 * max_x
212                  ),
213                  *self.trans_coords(
214                      max_x,
215                      m * max_x
216                  )
217              )
218
219              # We keep looping and increasing the multiple of c until we stop drawing lines on the canvas
220              multiple_of_c = 1
221              while self.draw_pair_of_oblique_lines(painter, m, multiple_of_c * c):
222                  multiple_of_c += 1
```

This separation of functionality made designing and debugging this part of the solution much easier. The `draw_pair_of_oblique_lines()` method looked like this:

```
     # cf05e09e5ebb6ea7a96db8660d0d8de6b946490a
     # src/lintrans/gui/plots/classes.py

118  class VectorGridPlot(BackgroundPlot):
...
224      def draw_pair_of_oblique_lines(self, painter: QPainter, m: float, c: float) -> bool:
225          """Draw a pair of oblique lines, using the equation y = mx + c.
226
227          This method just calls :meth:`draw_oblique_line` with ``c`` and ``-c``,
228          and returns True if either call returned True.
229
230          :param QPainter painter: The ``QPainter`` object to use for drawing the vectors and grid lines
231          :param float m: The gradient of the lines to draw
232          :param float c: The y-intercept of the lines to draw. We use the positive and negative versions
233          :returns bool: Whether we were able to draw any lines on the canvas
234          """
235          return any([
236              self.draw_oblique_line(painter, m, c),
237              self.draw_oblique_line(painter, m, -c)
238          ])
239
```

```python
240        def draw_oblique_line(self, painter: QPainter, m: float, c: float) -> bool:
241            """Draw an oblique line, using the equation y = mx + c.
242
243            We only draw the part of the line that fits within the canvas, returning True if
244            we were able to draw a line within the boundaries, and False if we couldn't draw a line
245
246            :param QPainter painter: The ``QPainter`` object to use for drawing the vectors and grid lines
247            :param float m: The gradient of the line to draw
248            :param float c: The y-intercept of the line to draw
249            :returns bool: Whether we were able to draw a line on the canvas
250            """
251            max_x, max_y = self.grid_corner()
252
253            # These variable names are shortened for convenience
254            # myi is max_y_intersection, mmyi is minus_max_y_intersection, etc.
255            myi = (max_y - c) / m
256            mmyi = (-max_y - c) / m
257            mxi = max_x * m + c
258            mmxi = -max_x * m + c
259
260            # The inner list here is a list of coords, or None
261            # If an intersection fits within the bounds, then we keep its coord,
262            # else it is None, and then gets discarded from the points list
263            # By the end, points is a list of two coords, or an empty list
264            points: list[tuple[float, float]] = [
265                x for x in [
266                    (myi, max_y) if -max_x < myi < max_x else None,
267                    (mmyi, -max_y) if -max_x < mmyi < max_x else None,
268                    (max_x, mxi) if -max_y < mxi < max_y else None,
269                    (-max_x, mmxi) if -max_y < mmxi < max_y else None
270                ] if x is not None
271            ]
272
273            # If no intersections fit on the canvas
274            if len(points) < 2:
275                return False
276
277            # If we can, then draw the line
278            else:
279                painter.drawLine(
280                    *self.trans_coords(*points[0]),
281                    *self.trans_coords(*points[1])
282                )
283                return True
```

To illustrate what this code is doing, I'll use a diagram.
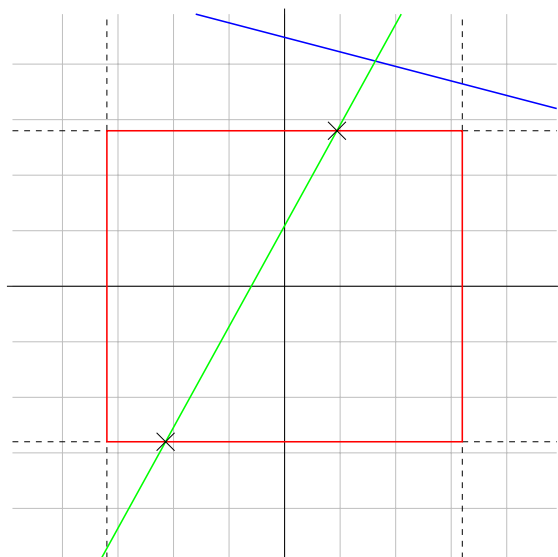
Figure 3.4.1: Two example lines and the viewport box
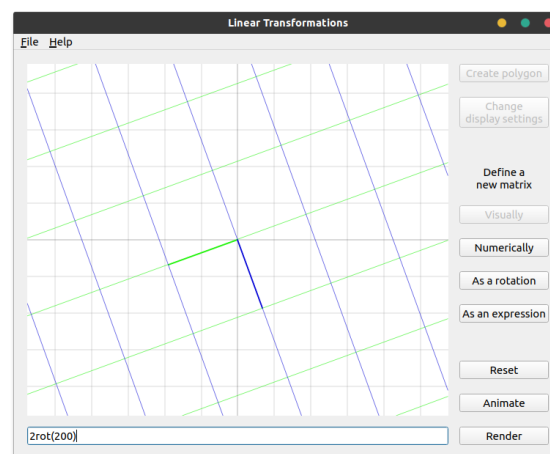
Figure 3.4.2: A demonstration of the new oblique lines system.

The red box represents the viewport of the GUI. The dashed lines represent the extensions of the red box. For a given line we want to draw, we first want to find where it intersects these orthogonal lines. Any oblique line will intersect each of these lines exactly once. This is what the `myi`, `mmyi`, `mxi`, and `mmxi` variables represent. The value of `myi` is the x value where the line intersects the maximum y line, for example.

In the case of the blue line, all 4 intersection points are outside the bounds of the box, whereas the green line intersects with the box, as shown with the crosses. We use a list comprehension over a list of ternaries to get the `points` list. This list contains 0 or 2 coordinates, and we may or may not draw a line accordingly.

That's how the `draw_oblique_line()` method works, and the `draw_pair_of_oblique_lines()` method just calls it with positive and negative values of c.

3.4.2 Adding vector arrowheads
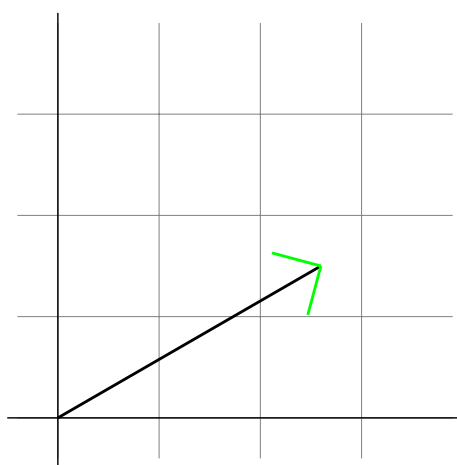
Figure 3.4.3: An example of a vector with the arrowheads highlighted in green

Now that I had a good renderer, I wanted to add arrowheads to the vectors to make them easier to see. They were already thicker than the gridlines, but adding arrowheads like in the 3blue1brown series would make them much easier to see. Unfortunately, I couldn't work out how to do this.

I wanted a function that would take a coordinate, treat it as a unit vector, and draw lines at 45° angles at the tip. This wasn't how I was conceptualising the problem at the time and because of that, I couldn't work out how to solve this problem. I could create this 45° lines in the top right quadrant, but none of my possible solutions worked for any arbitrary point.

So I started googling and found a very nice algorithm on `csharphelper.com`[56], which I adapted for Python.

```python
# 5373b1ad8040f6726147cccea523c0570251cf67
# src/lintrans/gui/plots/widgets.py

12    class VisualizeTransformationWidget(VectorGridPlot):
...
52        def draw_arrowhead_away_from_origin(self, painter: QPainter, point: tuple[float, float]) -> None:
53            """Draw an arrowhead at ``point``, pointing away from the origin.
54
55            :param QPainter painter: The ``QPainter`` object to use to draw the arrowheads with
56            :param point: The point to draw the arrowhead at, given in grid coords
57            :type point: tuple[float, float]
58            """
59            # This algorithm was adapted from a C# algorithm found at
60            # http://csharphelper.com/blog/2014/12/draw-lines-with-arrowheads-in-c/
61
62            # Get the x and y coords of the point, and then normalize them
63            # We have to normalize them, or else the size of the arrowhead will
64            # scale with the distance of the point from the origin
65            x, y = point
66            nx = x / np.sqrt(x * x + y * y)
67            ny = y / np.sqrt(x * x + y * y)
68
69            # We choose a length and do some magic to find the steps in the x and y directions
70            length = 0.15
71            dx = length * (-nx - ny)
72            dy = length * (nx - ny)
73
74            # Then we just plot those lines
75            painter.drawLine(*self.trans_coords(x, y), *self.trans_coords(x + dx, y + dy))
76            painter.drawLine(*self.trans_coords(x, y), *self.trans_coords(x - dy, y + dx))
77
78        def draw_vector_arrowheads(self, painter: QPainter) -> None:
79            """Draw arrowheads at the tips of the basis vectors.
80
81            :param QPainter painter: The ``QPainter`` object to use to draw the arrowheads with
82            """
83            painter.setPen(QPen(self.colour_i, self.width_vector_line))
84            self.draw_arrowhead_away_from_origin(painter, self.point_i)
85            painter.setPen(QPen(self.colour_j, self.width_vector_line))
86            self.draw_arrowhead_away_from_origin(painter, self.point_j)
```

As the comments suggest, we get the x and y components of the normalised vector, and then do some magic with a chosen length and get some distance values, and then draw those lines. I don't really understand how this code works, but I'm happy that it does. All we have to do is call `draw_vector_arrowheads()` from `paintEvent()`.
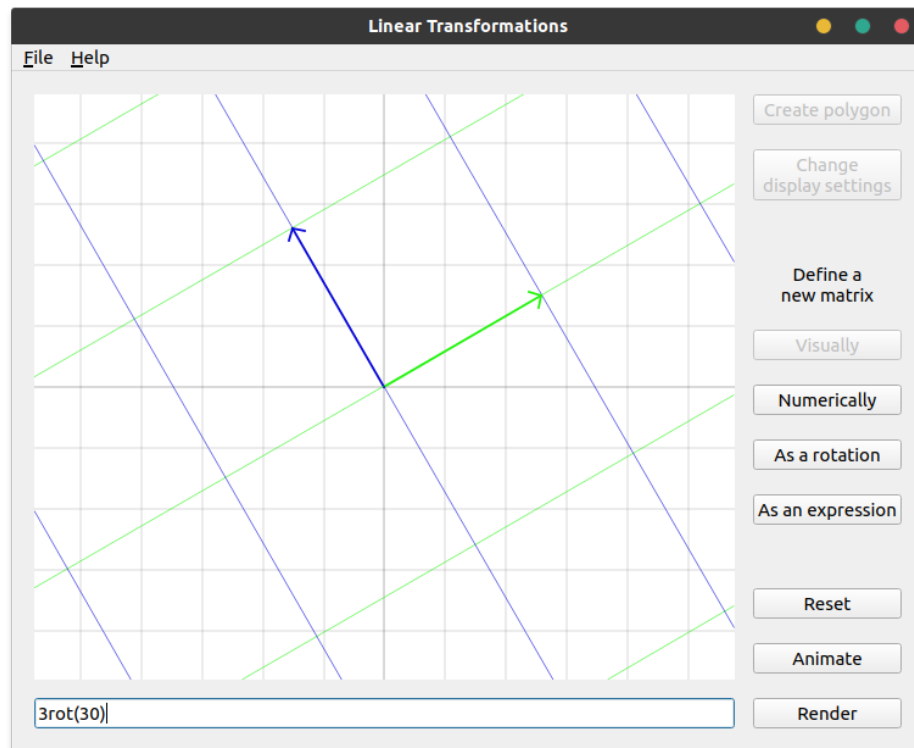
Figure 3.4.4: An example of the i and j vectors with arrowheads

3.4.3   Implementing zoom

The next thing I wanted to do was add the ability to zoom in and out of the viewport, and I wanted a button to reset the zoom level as well. I added a `default_grid_spacing` class attribute in `BackgroundPlot` and used that as the `grid_spacing` instance attribute in `__init__()`.

```
# d944e86e1d0fdc2c4be4d63479bc6bc3a31568ef
# src/lintrans/gui/plots/classes.py

12    class BackgroundPlot(QWidget):
...
27        default_grid_spacing: int = 50

28
29        def __init__(self, *args, **kwargs):
30            """Create the widget and setup backend stuff for rendering.
31
32            .. note:: ``*args`` and ``**kwargs`` are passed the superclass constructor (``QWidget``).
33            """
34            super().__init__(*args, **kwargs)
35
36            self.setAutoFillBackground(True)
37
38            # Set the background to white
39            palette = self.palette()
40            palette.setColor(self.backgroundRole(), Qt.white)
41            self.setPalette(palette)
42
43            # Set the gird colour to grey and the axes colour to black
44            self.colour_background_grid = QColor(128, 128, 128)
45            self.colour_background_axes = QColor(0, 0, 0)
46
47            self.grid_spacing = BackgroundPlot.default_grid_spacing
```

The reset button in `LintransMainWindow` simply sets `plot.grid_spacing` to the default.

To actually allow for zooming, I had to implement the `wheelEvent()` method in `BackgroundPlot` to listen for mouse wheel events. After reading through the docs for the `QWheelEvent` class[45], I learned how to handle this event.

```
# d944e86e1d0fdc2c4be4d63479bc6bc3a31568ef
# src/lintrans/gui/plots/classes.py

12   class BackgroundPlot(QWidget):
...
119      def wheelEvent(self, event: QWheelEvent) -> None:
120          """Handle a ``QWheelEvent`` by zooming in or our of the grid."""
121          # angleDelta() returns a number of units equal to 8 times the number of degrees rotated
122          degrees = event.angleDelta() / 8
123
124          if degrees is not None:
125              self.grid_spacing = max(1, self.grid_spacing + degrees.y())
126
127          event.accept()
128          self.update()
```

All we do is get the amount that the user scrolled and add that to the current spacing, taking the max with 1, which acts as a minimum grid spacing. We need to use `degrees.y()` on line 125 because Qt5 allows for mice that can scroll in the x and y directions, and we only want the y component. Line 127 marks the event as accepted so that the parent widget doesn't try to act on it.
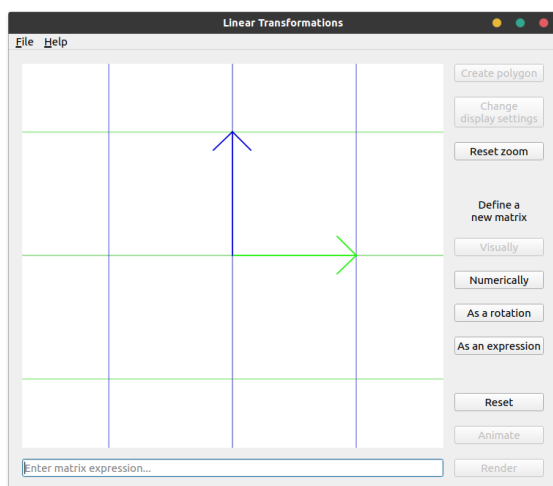
Figure 3.4.5: The GUI zoomed in a bit
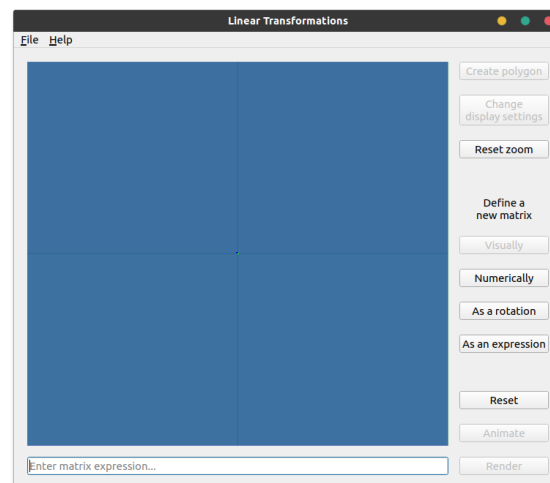
Figure 3.4.6: The GUI zoomed out as far as possible

There are two things I don't like here. Firstly, the minimum grid spacing is too small. The user can zoom out too far. Secondly, the arrowheads are too big in figure 3.4.5.

The first problem is minor and won't be fixed for quite a while, but I fixed the second problem quite quickly.

We want the arrowhead length to not just be 0.15, but to scale with the zoom level (the ratio between default grid spacing and current spacing).

This creates a slight issue when zoomed out all the way, because the arrowheads are then far larger than the vectors themselves, so we take the minimum of the scaled length and the vector length.

I factored out the default arrowhead length into the `arrowhead_length` instance attribute and initialize it in `__init__()`.

```
# 3d19a003368ae992ebb60049685bb04fde0836b5
# src/lintrans/gui/plots/widgets.py
```

```
12    class VisualizeTransformationWidget(VectorGridPlot):
...
54        def draw_arrowhead_away_from_origin(self, painter: QPainter, point: tuple[float, float]) -> None:
...
68            vector_length = np.sqrt(x * x + y * y)
69            nx = x / vector_length
70            ny = y / vector_length
71
72            # We choose a length and find the steps in the x and y directions
73            length = min(
74                self.arrowhead_length * self.default_grid_spacing / self.grid_spacing,
75                vector_length
76            )
```

This code results in arrowheads that stay the same length unless the user is zoomed out basically as far as possible.
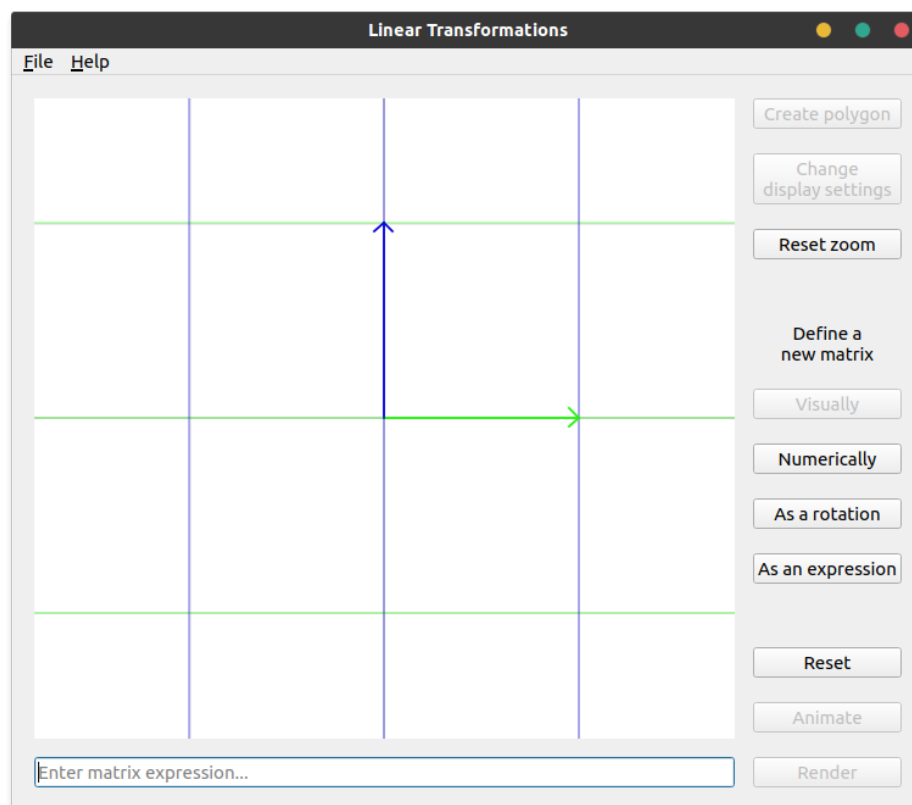
Figure 3.4.7: The arrowheads adjusted for zoom level

3.4.4 Animation blocks zooming

The biggest problem with this new zoom feature is that when animating between matrices, the user is unable to zoom. This is because when `LintransMainWindow.animate_expression()` is called, it uses Python's standard library `time.sleep()` function to delay each frame, which prevents Qt from handling user interaction while we're animating. This was a problem.

I did some googling and found a helpful post on StackOverflow[23] that gave me a nice solution. The user `ekhumoro` used the functions `QApplication.processEvents()` and `QThread.msleep()` to solve the problem, and I used these functions in my own app, with much success.

After reading 'The Event System' in the Qt5 documentation[57], I learned that Qt5 uses an event loop,

a lot like JavaScript. This means that events are scheduled to be executed on the next pass of the event loop. I also read the documentation for the `repaint()` and `update()` methods on the `QWidget` class[48, 50] and decided that it would be better to just queue a repaint by calling `update()` on the plot rather than immediately repaint with `repaint()`, and then call `QApplication.processEvents()` to process the pending events on the main thread. This is a nicer way of repainting, which reduces potential flickering issues, and using `QThread.msleep()` allows for asynchronous processing and therefore non-blocking animation.

3.4.5   Rank 1 transformations

The rank of a matrix is the dimension of its column space. This is the dimension of the span of its columns, which is to say the dimension of the output space. The rank of a matrix must be less than or equal to the dimension of the matrix, so we only need to worry about ranks 0, 1, and 2. There is only one rank 0 matrix, which is the **0** matrix itself. I've already covered this case by just not drawing any transformed grid lines.

Rank 2 matrices encompass most 2D matrices, and I've already covered this case in §3.3.4 and §3.4.1. A rank 1 matrix collapses all of 2D space onto a single line, so for this type of matrix, we should just draw this line.

This code is in `VectorGridPlot.draw_parallel_lines()`. We assemble the matrix $\begin{pmatrix} \text{vector\_x} & \text{point\_x} \\ \text{vector\_y} & \text{point\_y} \end{pmatrix}$ (which is actually the matrix used to create the transformation we're trying to render lines for) and use this matrix to check determinant and rank.

```
# 677b38c87bb6722b16aaf35058cf3cef66e43c21
# src/lintrans/gui/plots/classes.py
132    class VectorGridPlot(BackgroundPlot):
...
164        def draw_parallel_lines(self, painter: QPainter, vector: tuple[float, float], point: tuple[float, float]) ->
       ↪    None:
...
177            # If the determinant is 0
178            if abs(vector_x * point_y - vector_y * point_x) < 1e-12:
179                rank = np.linalg.matrix_rank(
180                    np.array([
181                        [vector_x, point_x],
182                        [vector_y, point_y]
183                    ])
184                )
185
186                # If the matrix is rank 1, then we can draw the column space line
187                if rank == 1:
188                    self.draw_oblique_line(painter, vector_y / vector_x, 0)
189
190                # If the rank is 0, then we don't draw any lines
191                else:
192                    return
```

Additionally, there was a bug with animating these determinant 0 matrices, since we try to scale the determinant through the animation, as documented in §3.3.6, but when the determinant is 0, this causes issues. To fix this, we just check the `det_target` variable in `LintransMainWindow.animate_expression` and if it's 0, we use the non-scaled version of the matrix.

```
# b889b686d997c2b64124bee786bccba3fc4f6b08
# src/lintrans/gui/main_window.py
22     class LintransMainWindow(QMainWindow):
...
262        def animate_expression(self) -> None:
...
```

```
274              for i in range(0, steps + 1):
...
307                      # If we're animating towards a det 0 matrix, then we don't want to scale the
308                      # determinant with the animation, because this makes the process not work
309                      # I'm doing this here rather than wrapping the whole animation logic in an
310                      # if block mainly because this looks nicer than an extra level of indentation
311                      # The extra processing cost is negligible thanks to NumPy's optimizations
312                      if det_target == 0:
313                          matrix_c = matrix_a
314                      else:
315                          matrix_c = scalar * matrix_b
```

3.4.6 Matrices that are too big

One of my friends was playing around with the prototype and she discovered a bug. When trying to render really big matrices, we can get errors like '`OverflowError: argument 3 overflowed: value must be in the range -2147483648 to 2147483647`' because PyQt5 is a wrapper over Qt5, which is a C++ library that uses the C++ `int` type for the `painter.drawLine()` call. This type is a 32-bit integer. Python can store integers of arbitrary precision, but when PyQt5 calls the underlying C++ library code, this gets cast to a C++ `int` and we can get an `OverflowError`.

This isn't a problem with the gridlines, because we only draw them inside the viewport, as discussed in §3.4.1, and these calculations all happen in Python, so integer precision is not a concern. However, when drawing the basis vectors, we just draw them directly, so we'll have to check that they're within the limit.

I'd previously created a `LintransMainWindow.show_error_message()` method for telling the user when they try to take the inverse of a singular matrix[11].

```
# 0f699dd95b6431e95b2311dcb03e7af49c19613f
# src/lintrans/gui/main_window.py

23   class LintransMainWindow(QMainWindow):
...
378      def show_error_message(self, title: str, text: str, info: str | None = None) -> None:
379          """Show an error message in a dialog box.
380
381          :param str title: The window title of the dialog box
382          :param str text: The simple error message
383          :param info: The more informative error message
384          :type info: Optional[str]
385          """
386          dialog = QMessageBox(self)
387          dialog.setIcon(QMessageBox.Critical)
388          dialog.setWindowTitle(title)
389          dialog.setText(text)
390
391          if info is not None:
392              dialog.setInformativeText(info)
393
394          dialog.open()
395
396          dialog.finished.connect(self.update_render_buttons)
```

I then created the `is_matrix_too_big()` method to just check that the elements of the matrix are within the desired bounds. If it returns `True` when we try to render or animate, then we call `show_error_message()`.

```
# 4682a7b225747cfd77aca0fe3abcdd1397b7c5dd
# src/lintrans/gui/main_window.py
```

---

[11]This commit didn't get a standalone section in this write-up because it was so small

```
 24    class LintransMainWindow(QMainWindow):
...
407        def is_matrix_too_big(self, matrix: MatrixType) -> bool:
408            """Check if the given matrix will actually fit onto the canvas.
409
410            Convert the elements of the matrix to canvas coords and make sure they fit within Qt's 32-bit integer limit.
411
412            :param MatrixType matrix: The matrix to check
413            :returns bool: Whether the matrix fits on the canvas
414            """
415            coords: list[tuple[int, int]] = [self.plot.trans_coords(*vector) for vector in matrix.T]
416
417            for x, y in coords:
418                if not (-2147483648 <= x <= 2147483647 and -2147483648 <= y <= 2147483647):
419                    return True
420
421            return False
```

3.4.7   Creating the `DefineVisuallyDialog`

Next, I wanted to allow the user to define a matrix visually by dragging the basis vectors. To do this, I obviously needed a new `DefineDialog` subclass for it.

```
       # 16ca0229aab73b3f4a8fe752dee3608f3ed6ead5
       # src/lintrans/gui/dialogs/define_new_matrix.py

135    class DefineVisuallyDialog(DefineDialog):
136        """The dialog class that allows the user to define a matrix visually."""
137
138        def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
139            """Create the widgets and layout of the dialog.
140
141            :param MatrixWrapper matrix_wrapper: The MatrixWrapper that this dialog will mutate
142            """
143            super().__init__(matrix_wrapper, *args, **kwargs)
144
145            self.setMinimumSize(500, 450)
146
147            # === Create the widgets
148
149            self.combobox_letter.activated.connect(self.show_matrix)
150
151            self.plot = DefineVisuallyWidget(self)
152
153            # === Arrange the widgets
154
155            self.hlay_definition.addWidget(self.plot)
156            self.hlay_definition.setStretchFactor(self.plot, 1)
157
158            self.vlay_all = QVBoxLayout()
159            self.vlay_all.setSpacing(20)
160            self.vlay_all.addLayout(self.hlay_definition)
161            self.vlay_all.addLayout(self.hlay_buttons)
162
163            self.setLayout(self.vlay_all)
164
165            # We load the default matrix A into the plot
166            self.show_matrix(0)
167
168            # We also enable the confirm button, because any visually defined matrix is valid
169            self.button_confirm.setEnabled(True)
170
171        def update_confirm_button(self) -> None:
172            """Enable the confirm button.
173
174            .. note::
175               The confirm button is always enabled in this dialog and this method is never actually used,
176               so it's got an empty body. It's only here because we need to implement the abstract method.
177            """
```

```
178
179        def show_matrix(self, index: int) -> None:
180            """Show the selected matrix on the plot. If the matrix is None, show the identity."""
181            matrix = self.matrix_wrapper[ALPHABET_NO_I[index]]
182
183            if matrix is None:
184                matrix = self.matrix_wrapper['I']
185
186            self.plot.visualize_matrix_transformation(matrix)
187            self.plot.update()
188
189        def confirm_matrix(self) -> None:
```

This `DefineVisuallyDialog` class just implements the normal methods needed for a `DefineDialog` and has a `plot` attribute to handle drawing graphics and handling mouse movement. After creating the `DefineVisuallyWidget` as a skeleton and doing some more research in the Qt5 docs[47], I renamed the `trans_coords()` methods to `canvas_coords()` to make the intent more clear, and created a `grid_coords()` method.

```
           # 417aea6555029b049c470faff18df29f064f6101
           # src/lintrans/gui/plots/classes.py

13         class BackgroundPlot(QWidget):
...
85             def grid_coords(self, x: int, y: int) -> tuple[float, float]:
86                 """Convert a coordinate from canvas coords to grid coords.
87
88                 :param int x: The x component of the canvas coordinate
89                 :param int y: The y component of the canvas coordinate
90                 :returns: The resultant grid coordinates
91                 :rtype: tuple[float, float]
92                 """
93                 # We get the maximum grid coords and convert them into canvas coords
94                 return (x - self.canvas_origin[0]) / self.grid_spacing, (-y + self.canvas_origin[1]) / self.grid_spacing
```

I then needed to implement the methods to handle mouse movement in the `DefineVisuallyWidget` class. Thankfully, Ross Wilson, the person who helped me learn about the `QWidget.paintEvent()` method in §3.3.1, also wrote an example of draggable points[6]. In my post, I had explained that I needed draggable points on my canvas, and Ross was helpful enough to create an example in their own time. I probably could've worked it out myself eventually, but this example allowed me to learn a lot quicker.

```
           # 417aea6555029b049c470faff18df29f064f6101
           # src/lintrans/gui/plots/widgets.py

56         class DefineVisuallyWidget(VisualizeTransformationWidget):
57             """This class is the widget that allows the user to visually define a matrix.
58
59             This is just the widget itself. If you want the dialog, use
60             :class:`lintrans.gui.dialogs.define_new_matrix.DefineVisuallyDialog`.
61             """
62
63             def __init__(self, *args, **kwargs):
64                 """Create the widget and enable mouse tracking. ``*args`` and ``**kwargs`` are passed to ``super()``."""
65                 super().__init__(*args, **kwargs)
66
67                 # self.setMouseTracking(True)
68                 self.dragged_point: tuple[float, float] | None = None
69
70                 # This is the distance that the cursor needs to be from the point to drag it
71                 self.epsilon: int = 5
72
73             def mousePressEvent(self, event: QMouseEvent) -> None:
74                 """Handle a QMouseEvent when the user pressed a button."""
75                 mx = event.x()
76                 my = event.y()
```

```
77                    button = event.button()
78
79                    if button != Qt.LeftButton:
80                        event.ignore()
81                        return
82
83                    for point in (self.point_i, self.point_j):
84                        px, py = self.canvas_coords(*point)
85                        if abs(px - mx) <= self.epsilon and abs(py - my) <= self.epsilon:
86                            self.dragged_point = point[0], point[1]
87
88                    event.accept()
89
90            def mouseReleaseEvent(self, event: QMouseEvent) -> None:
91                """Handle a QMouseEvent when the user release a button."""
92                if event.button() == Qt.LeftButton:
93                    self.dragged_point = None
94                    event.accept()
95                else:
96                    event.ignore()
97
98            def mouseMoveEvent(self, event: QMouseEvent) -> None:
99                """Handle the mouse moving on the canvas."""
100               mx = event.x()
101               my = event.y()
102
103               if self.dragged_point is not None:
104                   x, y = self.grid_coords(mx, my)
105
106                   if self.dragged_point == self.point_i:
107                       self.point_i = x, y
108
109                   elif self.dragged_point == self.point_j:
110                       self.point_j = x, y
111
112                   self.dragged_point = x, y
113
114                   self.update()
115
116                   print(self.dragged_point)
117                   print(self.point_i, self.point_j)
118
119                   event.accept()
120
121               event.ignore()
```

This snippet has the line '`self.setMouseTracking(True)`' commented out. This line was in the example, but it turns out that I don't want it. Mouse tracking means that a widget will receive a `QMouseEvent` every time the mouse moves. But if it's disabled (the default), then the widget will only receive a `QMouseEvent` for mouse movement when a button is held down at the same time.

I've also left in some print statements on lines 116 and 117. These small oversights are there because I just forgot to remove them before I committed these changes. They were removed 3 commits later.

3.4.8   Fixing a division by zero bug

When drawing the rank line for a determinant 0, rank 1 matrix, we can encounter a division by zero error. I'm sure this originally manifested in a crash with a `ZeroDivisionError` at runtime, but now I can only get a `RuntimeWarning` when running the old code from commit `16ca0229aab73b3f4a8fe752dee3608f3ed6ead5`.

Whether it crashes or just warns the user, there is a division by zero bug when trying to render $\begin{pmatrix} k & 0 \\ 0 & 0 \end{pmatrix}$

or $\begin{pmatrix} 0 & 0 \\ 0 & k \end{pmatrix}$. To fix this, I just handled those cases separately in `VectorGridPlot.draw_parallel_lines(`$\lrcorner$
`)`.

```
       # 40bee6461d477a5c767ed132359cd511c0051e3b
       # src/lintrans/gui/plots/classes.py

140    class VectorGridPlot(BackgroundPlot):
...
174        def draw_parallel_lines(self, painter: QPainter, vector: tuple[float, float], point: tuple[float, float]) ->
       ↪   None:
...
188            if abs(vector_x * point_y - vector_y * point_x) < 1e-12:
...
196                # If the matrix is rank 1, then we can draw the column space line
197                if rank == 1:
198                    if abs(vector_x) < 1e-12:
199                        painter.drawLine(self.width() // 2, 0, self.width() // 2, self.height())
200                    elif abs(vector_y) < 1e-12:
201                        painter.drawLine(0, self.height() // 2, self.width(), self.height() // 2)
202                    else:
203                        self.draw_oblique_line(painter, vector_y / vector_x, 0)
204
205                # If the rank is 0, then we don't draw any lines
206                else:
207                    return
```

3.4.9   Implementing transitional animation

Currently, all animation animates from **I** to the target matrix **T**. This means it resets the plot at the start. I eventually want an applicative animation system, where the matrix in the box is applied to the current scene. But I also want an option for a transitional animation, where the program animates from the start matrix **S** to the target matrix **T**, and this seems easier to implement, so I'll do it first.

In `LintransMainWindow`, I created a new method called `animate_between_matrices()` and I call it from `animate_expression()`. The maths for smoothening determinants in §3.3.6 assumed the starting matrix had a determinant of 1, but when using transitional animation, this may not always be true.

If we let **S** be the starting matrix, and **A** be the matrix from the first stage of calculation as specified in §3.3.6, then we want a c such that $\det(c\mathbf{A}) = \det(\mathbf{S})$, so we get $c = \sqrt{\left|\dfrac{\det(\mathbf{S})}{\det(\mathbf{A})}\right|}$ by the identity $\det(c\mathbf{A}) = c^2 \det(\mathbf{A})$.

Following the same logic as in §3.3.6, we can let $\mathbf{B} = c\mathbf{A}$ and then scale it by d to get the same determinant as the target matrix **T** and find that $d = \sqrt{\left|\dfrac{\det(\mathbf{T})}{\det(\mathbf{B})}\right|}$. Unlike previously, $\det(\mathbf{B})$ could be any scalar, so we can't simplify our expression for d.

We then scale this with our `proportion` variable p to get a scalar $s = 1 + p\left(\sqrt{\left|\dfrac{\det(\mathbf{T})}{\det(\mathbf{B})}\right|} - 1\right)$ and render $\mathbf{C} = s\mathbf{B}$ on each frame.

In code, that looks like this:

```
       # 4017b84fbce67d8e041bc9ce84cefcb0b6e65e1f
       # src/lintrans/gui/main_window.py

25     class LintransMainWindow(QMainWindow):
...
275        def animate_expression(self) -> None:
276            """Animate from the current matrix to the matrix in the expression box."""
277            self.button_render.setEnabled(False)
278            self.button_animate.setEnabled(False)
279
280            # Get the target matrix and it's determinant
281            try:
```

```
282                    matrix_target = self.matrix_wrapper.evaluate_expression(self.lineedit_expression_box.text())
283
284            except linalg.LinAlgError:
285                self.show_error_message('Singular matrix', 'Cannot take inverse of singular matrix')
286                return
287
288            matrix_start: MatrixType = np.array([
289                [self.plot.point_i[0], self.plot.point_j[0]],
290                [self.plot.point_i[1], self.plot.point_j[1]]
291            ])
292
293            self.animate_between_matrices(matrix_start, matrix_target)
294
295            self.button_render.setEnabled(True)
296            self.button_animate.setEnabled(True)
297
298        def animate_between_matrices(self, matrix_start: MatrixType, matrix_target: MatrixType, steps: int = 100) ->
    ↪  None:
299            """Animate from the start matrix to the target matrix."""
300            det_target = linalg.det(matrix_target)
301            det_start = linalg.det(matrix_start)
302
303            for i in range(0, steps + 1):
304                # This proportion is how far we are through the loop
305                proportion = i / steps
306
307                # matrix_a is the start matrix plus some part of the target, scaled by the proportion
308                # If we just used matrix_a, then things would animate, but the determinants would be weird
309                matrix_a = matrix_start + proportion * (matrix_target - matrix_start)
310
311                # So to fix the determinant problem, we get the determinant of matrix_a and use it to normalise
312                det_a = linalg.det(matrix_a)
313
314                # For a 2x2 matrix A and a scalar c, we know that det(cA) = c^2 det(A)
315                # We want B = cA such that det(B) = det(S), where S is the start matrix,
316                # so then we can scale it with the animation, so we get
317                # det(cA) = c^2 det(A) = det(S) => c = sqrt(abs(det(S) / det(A)))
318                # Then we scale A to get the determinant we want, and call that matrix_b
319                if det_a == 0:
320                    c = 0
321                else:
322                    c = np.sqrt(abs(det_start / det_a))
323
324                matrix_b = c * matrix_a
325                det_b = linalg.det(matrix_b)
326
327                # matrix_c is the final matrix that we then render for this frame
328                # It's B, but we scale it over time to have the target determinant
329
330                # We want some C = dB such that det(C) is some target determinant T
331                # det(dB) = d^2 det(B) = T => d = sqrt(abs(T / det(B))
332
333                # We're also subtracting 1 and multiplying by the proportion and then adding one
334                # This just scales the determinant along with the animation
335                scalar = 1 + proportion * (np.sqrt(abs(det_target / det_b)) - 1)
336
337                # If we're animating towards a det 0 matrix, then we don't want to scale the
338                # determinant with the animation, because this makes the process not work
339                # I'm doing this here rather than wrapping the whole animation logic in an
340                # if block mainly because this looks nicer than an extra level of indentation
341                # The extra processing cost is negligible thanks to NumPy's optimizations
342                if det_target == 0:
343                    matrix_c = matrix_a
344                else:
345                    matrix_c = scalar * matrix_b
346
347                if self.is_matrix_too_big(matrix_c):
348                    self.show_error_message('Matrix too big', "This matrix doesn't fit on the canvas")
349                    return
350
351                self.plot.visualize_matrix_transformation(matrix_c)
352
353                # We schedule the plot to be updated, tell the event loop to
```

```
354                    # process events, and asynchronously sleep for 10ms
355                    # This allows for other events to be processed while animating, like zooming in and out
356                    self.plot.update()
```

This change results in an animation system that will transition from the current matrix to whatever the user types into the input box.

3.4.10   Allowing for sequential animation with commas

Applicative animation has two main forms. There's the version where a standard matrix expression gets applied to the current scene, and the kind where the user defines a sequence of matrices and we animate through the sequence, applying one at a time. Both of these are referenced in success criterion 6.

I want the user to be able to decide if they want applicative animation or transitional animation, so I'll need to create some form of display settings. However, transitional animation doesn't make much sense for sequential animation[12], so I can implement this now.

Applicative animation is just animating from the matrix **C** representing the current scene to the composition **TC** with the target matrix **T**.

We use **TC** instead of **CT** because matrix multiplication can be thought of as applying successive transformations from right to left. **TC** is the same as starting with the identity **I**, applying **C** (to get to the current scene), and then applying **T**.

Doing this in code is very simple. We just split the expression on commas, and then apply each sub-expression to the current scene one by one, pausing on each comma.

```
      # 60584d2559cacbf23479a1bebbb986a800a32331
      # src/lintrans/gui/main_window.py

25    class LintransMainWindow(QMainWindow):
...
284       def animate_expression(self) -> None:
285           """Animate from the current matrix to the matrix in the expression box."""
286           self.button_render.setEnabled(False)
287           self.button_animate.setEnabled(False)
288
289           matrix_start: MatrixType = np.array([
290               [self.plot.point_i[0], self.plot.point_j[0]],
291               [self.plot.point_i[1], self.plot.point_j[1]]
292           ])
293
294           text = self.lineedit_expression_box.text()
295
296           # If there's commas in the expression, then we want to animate each part at a time
297           if ',' in text:
298               current_matrix = matrix_start
299
300               # For each expression in the list, right multiply it by the current matrix,
301               # and animate from the current matrix to that new matrix
302               for expr in text.split(',')[::-1]:
303                   new_matrix = self.matrix_wrapper.evaluate_expression(expr) @ current_matrix
304
305                   self.animate_between_matrices(current_matrix, new_matrix)
306                   current_matrix = new_matrix
307
308                   # Here we just redraw and allow for other events to be handled while we pause
309                   self.plot.update()
310                   QApplication.processEvents()
311                   QThread.msleep(500)
```

---

[12] I have since changed my thoughts on this, and I allowed sequential transitional animation much later, in commit 41907b81661f3878e435b794d9d719491ef14237

```
312
313            # If there's no commas, then just animate directly from the start to the target
314            else:
315                # Get the target matrix and it's determinant
316                try:
317                    matrix_target = self.matrix_wrapper.evaluate_expression(text)
318
319                except linalg.LinAlgError:
320                    self.show_error_message('Singular matrix', 'Cannot take inverse of singular matrix')
321                    return
322
323                self.animate_between_matrices(matrix_start, matrix_target)
324
325        self.update_render_buttons()
```

We're deliberately not checking if the sub-expressions are valid here. We would normally validate the expression in `LintransMainWindow.update_render_buttons()` and only allow the user to render or animate an expression if it's valid. Now we have to check all the sub-expressions if the expression contains commas. Additionally, we can only animate these expressions with commas in them, so rendering should be disabled when the expression contains commas.

Compare the old code to the new code:

```
# 4017b84fbce67d8e041bc9ce84cefcb0b6e65e1f
# src/lintrans/gui/main_window.py

25    class LintransMainWindow(QMainWindow):
...
243        def update_render_buttons(self) -> None:
244            """Enable or disable the render and animate buttons according to whether the matrix expression is valid."""
245            valid = self.matrix_wrapper.is_valid_expression(self.lineedit_expression_box.text())
246            self.button_render.setEnabled(valid)
247            self.button_animate.setEnabled(valid)
```

```
# 60584d2559cacbf23479a1bebbb986a800a32331
# src/lintrans/gui/main_window.py

25    class LintransMainWindow(QMainWindow):
...
243        def update_render_buttons(self) -> None:
244            """Enable or disable the render and animate buttons according to whether the matrix expression is valid."""
245            text = self.lineedit_expression_box.text()
246
247            if ',' in text:
248                self.button_render.setEnabled(False)
249
250                valid = all(self.matrix_wrapper.is_valid_expression(x) for x in text.split(','))
251                self.button_animate.setEnabled(valid)
252
253            else:
254                valid = self.matrix_wrapper.is_valid_expression(text)
255                self.button_render.setEnabled(valid)
256                self.button_animate.setEnabled(valid)
```

3.5   Adding display settings

3.5.1   Creating the dataclass (and implementing applicative animation)

The first step of adding display settings is creating a dataclass to hold all of the settings. This dataclass will hold attributes to manage how a matrix transformation is displayed. Things like whether to show eigenlines or the determinant parallelogram. It will also hold information for animation. We can factor out the code used to smoothen the determinant, as written in §3.3.6, and make it dependant on a `bool` attribute of the `DisplaySettings` dataclass.

This is a standard class rather than some form of singleton to allow different plots to have different display settings. For example, the user might want different settings for the main view and the visual definition dialog. Allowing each instance of a subclass of `VectorGridPlot` to have its own `DisplaySettings` attribute allows for separate settings for separate plots.

However, this class initially just contained attributes relevant to animation, so it was only an attribute on `LintransMainWindow`.

```
# 2041c7a24d963d8d142d6f0f20ec3828ba8257c6
# src/lintrans/gui/settings.py

1   """This module contains the :class:`DisplaySettings` class, which holds configuration for display."""
2
3   from dataclasses import dataclass
4
5
6   @dataclass
7   class DisplaySettings:
8       """This class simply holds some attributes to configure display."""
9
10      animate_determinant: bool = True
11      """This controls whether we want the determinant to change smoothly during the animation."""
12
13      applicative_animation: bool = True
14      """There are two types of simple animation, transitional and applicative.
15
16      Let ``C`` be the matrix representing the currently displayed transformation, and let ``T`` be the target matrix.
17      Transitional animation means that we animate directly from ``C`` from ``T``,
18      and applicative animation means that we animate from ``C`` to ``TC``, so we apply ``T`` to ``C``.
19      """
20
21      animation_pause_length: int = 400
22      """This is the number of milliseconds that we wait between animations when using comma syntax."""
```

Once I had the dataclass, I just had to add '`from .settings import DisplaySettings`' to the top of the file, and '`self.display_settings = DisplaySettings()`' to the constructor of `LintransMainWindow`. I could then use the attributes of this dataclass in `animate_expression()`.

```
# 2041c7a24d963d8d142d6f0f20ec3828ba8257c6
# src/lintrans/gui/main_window.py

26  class LintransMainWindow(QMainWindow):
...
286     def animate_expression(self) -> None:
287         """Animate from the current matrix to the matrix in the expression box."""
288         self.button_render.setEnabled(False)
289         self.button_animate.setEnabled(False)
290
291         matrix_start: MatrixType = np.array([
292             [self.plot.point_i[0], self.plot.point_j[0]],
293             [self.plot.point_i[1], self.plot.point_j[1]]
294         ])
295
296         text = self.lineedit_expression_box.text()
297
```

```
298                 # If there's commas in the expression, then we want to animate each part at a time
299             if ',' in text:
300                 current_matrix = matrix_start
301
302                 # For each expression in the list, right multiply it by the current matrix,
303                 # and animate from the current matrix to that new matrix
304                 for expr in text.split(',')[::-1]:
305                     new_matrix = self.matrix_wrapper.evaluate_expression(expr) @ current_matrix
306
307                     self.animate_between_matrices(current_matrix, new_matrix)
308                     current_matrix = new_matrix
309
310                     # Here we just redraw and allow for other events to be handled while we pause
311                     self.plot.update()
312                     QApplication.processEvents()
313                     QThread.msleep(self.display_settings.animation_pause_length)
314
315             # If there's no commas, then just animate directly from the start to the target
316             else:
317                 # Get the target matrix and it's determinant
318                 try:
319                     matrix_target = self.matrix_wrapper.evaluate_expression(text)
320
321                 except linalg.LinAlgError:
322                     self.show_error_message('Singular matrix', 'Cannot take inverse of singular matrix')
323                     return
324
325                 # The concept of applicative animation is explained in /gui/settings.py
326                 if self.display_settings.applicative_animation:
327                     matrix_target = matrix_target @ matrix_start
328
329                 self.animate_between_matrices(matrix_start, matrix_target)
330
331             self.update_render_buttons()
```

Lines 327 are very important here. I included applicative animation as an option in the display settings because once I'd implemented animating from one matrix to another, it was very easy to implement applicative animation.

The user will input whatever matrix they wanted to apply to the current scene. Let's call that target matrix **T**. The matrix representing the starting state of the viewport is **S**. Animating from **S** to **T** is a transitional animation, but an applicative animation is simply animating from **S** to **TS**, so we can just say `matrix_target = matrix_target @ matrix_start` on line 327 (where `@` is the matrix multiplication operator), and continue as normal.

I also wrapped the main logic of `animate_between_matrices()` in an **if** block to check if the user wants the determinant to be smoothed.

```
# 03e154e1326dc256ffc1a539e97d8ef5ec89f6fd
# src/lintrans/gui/main_window.py

26   class LintransMainWindow(QMainWindow):
...
333      def animate_between_matrices(self, matrix_start: MatrixType, matrix_target: MatrixType, steps: int = 100) ->
     ↪   None:
334          """Animate from the start matrix to the target matrix."""
335          det_target = linalg.det(matrix_target)
336          det_start = linalg.det(matrix_start)
337
338          for i in range(0, steps + 1):
339              # This proportion is how far we are through the loop
340              proportion = i / steps
341
342              # matrix_a is the start matrix plus some part of the target, scaled by the proportion
343              # If we just used matrix_a, then things would animate, but the determinants would be weird
344              matrix_a = matrix_start + proportion * (matrix_target - matrix_start)
345
346              if self.display_settings.animate_determinant and det_target != 0:
347                  # To fix the determinant problem, we get the determinant of matrix_a and use it to normalise
```

```
348                        det_a = linalg.det(matrix_a)
349
350                        # For a 2x2 matrix A and a scalar c, we know that det(cA) = c^2 det(A)
351                        # We want B = cA such that det(B) = det(S), where S is the start matrix,
352                        # so then we can scale it with the animation, so we get
353                        # det(cA) = c^2 det(A) = det(S) => c = sqrt(abs(det(S) / det(A)))
354                        # Then we scale A to get the determinant we want, and call that matrix_b
355                        if det_a == 0:
356                            c = 0
357                        else:
358                            c = np.sqrt(abs(det_start / det_a))
359
360                        matrix_b = c * matrix_a
361                        det_b = linalg.det(matrix_b)
362
363                        # matrix_to_render is the final matrix that we then render for this frame
364                        # It's B, but we scale it over time to have the target determinant
365
366                        # We want some C = dB such that det(C) is some target determinant T
367                        # det(dB) = d^2 det(B) = T => d = sqrt(abs(T / det(B)))
368
369                        # We're also subtracting 1 and multiplying by the proportion and then adding one
370                        # This just scales the determinant along with the animation
371                        scalar = 1 + proportion * (np.sqrt(abs(det_target / det_b)) - 1)
372                        matrix_to_render = scalar * matrix_b
373
374                    else:
375                        matrix_to_render = matrix_a
376
377                    if self.is_matrix_too_big(matrix_to_render):
378                        self.show_error_message('Matrix too big', "This matrix doesn't fit on the canvas")
379                        return
380
381                    self.plot.visualize_matrix_transformation(matrix_to_render)
382
383                    # We schedule the plot to be updated, tell the event loop to
384                    # process events, and asynchronously sleep for 10ms
385                    # This allows for other events to be processed while animating, like zooming in and out
386                    self.plot.update()
387                    QApplication.processEvents()
388                    QThread.msleep(1000 // steps)
```

3.5.2   Creating the settings dialog

Display settings are good, but useless on their own. My next step was to add a settings dialog that would allow the user to edit these settings.

I first had to create the dialog class itself, so I created the `SettingsDialog` superclass first, so that I could use it for global settings in the future, as well as the specific `DisplaySettingsDialog` subclass now.

As far as I know, a dialog in Qt can't really return a value when it's closed[13], so the dialog keeps a public instance attribute for the `DisplaySettings` class itself, and then the main window can copy that instance attribute when the dialog is closed.

```
# b1ba4adc3c7723c95b490e831e651a7781af7d99
# src/lintrans/gui/dialogs/settings.py

1    """This module provides dialogs to edit settings within the app."""
2
3    from __future__ import annotations
4
5    import abc
```

---

[13]This is because Qt uses a system of event loops, so the main window continues executing its main loop while the dialog is doing the same. That means that the main window can't wait around for the dialog to close, so nothing can be returned from it.

```
6      import copy
7
8      from PyQt5 import QtWidgets
9      from PyQt5.QtCore import Qt
10     from PyQt5.QtGui import QIntValidator, QKeySequence
11     from PyQt5.QtWidgets import QCheckBox, QDialog, QHBoxLayout, QShortcut, QSizePolicy, QSpacerItem, QVBoxLayout
12
13     from lintrans.gui.settings import DisplaySettings
14
15
16     class SettingsDialog(QDialog):
17         """An abstract superclass for other simple dialogs."""
18
19         def __init__(self, *args, **kwargs):
20             """Create the widgets and layout of the dialog, passing ``*args`` and ``**kwargs`` to super."""
21             super().__init__(*args, **kwargs)
22
23             # === Create the widgets
24
25             self.button_confirm = QtWidgets.QPushButton(self)
26             self.button_confirm.setText('Confirm')
27             self.button_confirm.clicked.connect(self.confirm_settings)
28             self.button_confirm.setToolTip('Confirm these new settings<br><b>(Ctrl + Enter)</b>')
29             QShortcut(QKeySequence('Ctrl+Return'), self).activated.connect(self.button_confirm.click)
30
31             self.button_cancel = QtWidgets.QPushButton(self)
32             self.button_cancel.setText('Cancel')
33             self.button_cancel.clicked.connect(self.reject)
34             self.button_cancel.setToolTip('Revert these settings<br><b>(Escape)</b>')
35
36             # === Arrange the widgets
37
38             self.setContentsMargins(10, 10, 10, 10)
39
40             self.hlay_buttons = QHBoxLayout()
41             self.hlay_buttons.setSpacing(20)
42             self.hlay_buttons.addItem(QSpacerItem(50, 5, hPolicy=QSizePolicy.Expanding, vPolicy=QSizePolicy.Minimum))
43             self.hlay_buttons.addWidget(self.button_cancel)
44             self.hlay_buttons.addWidget(self.button_confirm)
45
46             self.vlay_options = QVBoxLayout()
47             self.vlay_options.setSpacing(20)
48
49             self.vlay_all = QVBoxLayout()
50             self.vlay_all.setSpacing(20)
51             self.vlay_all.addLayout(self.vlay_options)
52             self.vlay_all.addLayout(self.hlay_buttons)
53
54             self.setLayout(self.vlay_all)
55
56         @abc.abstractmethod
57         def load_settings(self) -> None:
58             """Load the current settings into the widgets."""
59
60         @abc.abstractmethod
61         def confirm_settings(self) -> None:
62             """Confirm the settings chosen in the dialog."""
63
64
65     class DisplaySettingsDialog(SettingsDialog):
66         """The dialog to allow the user to edit the display settings."""
67
68         def __init__(self, display_settings: DisplaySettings, *args, **kwargs):
69             """Create the widgets and layout of the dialog.
70
71             :param DisplaySettings display_settings: The :class:`lintrans.gui.settings.DisplaySettings` object to mutate
72             """
73             super().__init__(*args, **kwargs)
74
75             self.display_settings = display_settings
76             self.setWindowTitle('Change display settings')
77
78             # === Create the widgets
```

```
 79
 80                font_label = self.font()
 81                font_label.setUnderline(True)
 82                font_label.setPointSize(int(font_label.pointSize() * 1.2))
 83
 84                self.label_animations = QtWidgets.QLabel(self)
 85                self.label_animations.setText('Animations')
 86                self.label_animations.setAlignment(Qt.AlignCenter)
 87                self.label_animations.setFont(font_label)
 88
 89                self.checkbox_animate_determinant = QCheckBox(self)
 90                self.checkbox_animate_determinant.setText('Animate determinant')
 91                self.checkbox_animate_determinant.setToolTip('Smoothly animate the determinant during animation')
 92
 93                self.checkbox_applicative_animation = QCheckBox(self)
 94                self.checkbox_applicative_animation.setText('Applicative animation')
 95                self.checkbox_applicative_animation.setToolTip(
 96                    'Animate the new transformation applied to the current one,\n'
 97                    'rather than just that transformation on its own'
 98                )
 99
100                self.label_animation_pause_length = QtWidgets.QLabel(self)
101                self.label_animation_pause_length.setText('Animation pause length (ms)')
102                self.label_animation_pause_length.setToolTip(
103                    'How many milliseconds to pause for in comma-separated animations'
104                )
105
106                self.lineedit_animation_pause_length = QtWidgets.QLineEdit(self)
107                self.lineedit_animation_pause_length.setValidator(QIntValidator(1, 999, self))
108
109                # === Arrange the widgets
110
111                self.hlay_animation_pause_length = QHBoxLayout()
112                self.hlay_animation_pause_length.addWidget(self.label_animation_pause_length)
113                self.hlay_animation_pause_length.addWidget(self.lineedit_animation_pause_length)
114
115                self.vlay_options.addWidget(self.label_animations)
116                self.vlay_options.addWidget(self.checkbox_animate_determinant)
117                self.vlay_options.addWidget(self.checkbox_applicative_animation)
118                self.vlay_options.addLayout(self.hlay_animation_pause_length)
119
120                # Finally, we load the current settings
121                self.load_settings()
122
123            def load_settings(self) -> None:
124                """Load the current display settings into the widgets."""
125                self.checkbox_animate_determinant.setChecked(self.display_settings.animate_determinant)
126                self.checkbox_applicative_animation.setChecked(self.display_settings.applicative_animation)
127                self.lineedit_animation_pause_length.setText(str(self.display_settings.animation_pause_length))
128
129            def confirm_settings(self) -> None:
130                """Build a :class:`lintrans.gui.settings.DisplaySettings` object and assign it."""
131                self.display_settings.animate_determinant = self.checkbox_animate_determinant.isChecked()
132                self.display_settings.applicative_animation = self.checkbox_applicative_animation.isChecked()
133                self.display_settings.animation_pause_length = int(self.lineedit_animation_pause_length.text())
134
135                self.accept()
```

I then just had to enable the button in the main GUI and implement the method to open the new dialog. I have to use a lambda to capture the local `dialog` variable, but a separate method to actually assign its display settings, since Python doesn't allow assignments in lambda expressions.

```
     # b1ba4adc3c7723c95b490e831e651a7781af7d99
     # src/lintrans/gui/main_window.py

 27  class LintransMainWindow(QMainWindow):
...
436      def dialog_change_display_settings(self) -> None:
437          """Open the dialog to change the display settings."""
438          dialog = DisplaySettingsDialog(self.display_settings, self)
```

```
439                dialog.open()
440                dialog.finished.connect(lambda: self._assign_display_settings(dialog.display_settings))
441
442         def _assign_display_settings(self, display_settings: DisplaySettings) -> None:
443             """Assign a new value to ``self.display_settings``."""
444             self.display_settings = display_settings
```

The `dialog.finished` signal on line 440 should really be `dialog.accepted`. Currently, we re-assign the display settings whenever the dialog is closed in any way. Really, we should only re-assign them when the user hits the confirm button, but trying to cancel the changes will currently save them. This was a silly mistake and I fixed it along with some similar signal-related bugs a few weeks later. See §3.9.1.

Figure 3.5.1: The display settings dialog

3.5.3 Fixing a bug with transitional animation

While playing around with these new display settings, I encountered a bug with transitional animation. When you animate an expression with transitional animation and then animate the same thing again, nothing happens. This is because the app tries to transition from the starting position to the target position, but they are the same position, so nothing moves.

To fix this, I had to check if the start and target matrices were the same (within floating point error), and then reset the viewport to the identity first, before animating to the target as requested.

```
     # fa4a65540749e84b750ddea8abfd36a86c224b47
     # src/lintrans/gui/main_window.py

27   class LintransMainWindow(QMainWindow):
...
285      def animate_expression(self) -> None:
...
315          else:
...
328              # If we want a transitional animation and we're animating the same matrix, then restart the animation
329              # We use this check rather than equality because of small floating point errors
330              elif (matrix_start - matrix_target < 1e-12).all():
331                  matrix_start = self.matrix_wrapper['I']
332
333                  # We pause here for 200 ms to make the animation look a bit nicer
334                  self.plot.visualize_matrix_transformation(matrix_start)
335                  self.plot.update()
336                  QApplication.processEvents()
337                  QThread.msleep(200)
```

I later found a bug on line 330. If we subtract the start and target matrices and get a matrix of all negative numbers (rather than all zeroes, which is what I wanted to check for), then the if condition will still be true. That means that some completely different matrices can be considered the same, and the viewport will reset before animating them. To fix this, I can simply take the absolute value.

```
     # 3c490c48a0f4017ab8ee9cf471a65c251817b00e
     # src/lintrans/gui/main_window.py

333              elif (abs(matrix_start - matrix_target) < 1e-12).all():
```

3.5.4 Adding the determinant parallelogram

The determinant can be represented as the area of the parallelogram formed by the basis vectors. This would be good to visualize in the app.

To do that, I had to add a setting to the display settings, create a function to actually draw it in `VectorGridPlot`, and call that function from `paintEvent()`.

```
      # e9e76c1d4f28452efc6ae18afb936616006fd04a
      # src/lintrans/gui/settings.py

  9   class DisplaySettings:
...
 26       draw_determinant_parallelogram: bool = False
 27       """This controls whether or not we should shade the parallelogram representing the determinant of the matrix."""
```

```
      # e9e76c1d4f28452efc6ae18afb936616006fd04a
      # src/lintrans/gui/plots/classes.py

140   class VectorGridPlot(BackgroundPlot):
...
385       def draw_determinant_parallelogram(self, painter: QPainter) -> None:
386           """Draw the parallelogram of the determinant of the matrix."""
387           path = QPainterPath()
388           path.moveTo(*self.canvas_origin)
389           path.lineTo(*self.canvas_coords(*self.point_i))
390           path.lineTo(*self.canvas_coords(self.point_i[0] + self.point_j[0], self.point_i[1] + self.point_j[1]))
391           path.lineTo(*self.canvas_coords(*self.point_j))
392
393           brush = QBrush(QColor(16, 235, 253, alpha=128), Qt.SolidPattern)
394           painter.fillPath(path, brush)
```

```
      # e9e76c1d4f28452efc6ae18afb936616006fd04a
      # src/lintrans/gui/plots/widgets.py

 13   class VisualizeTransformationWidget(VectorGridPlot):
...
 42       def paintEvent(self, event: QPaintEvent) -> None:
 43           """Handle a ``QPaintEvent`` by drawing the background grid and the transformed grid.
 44
 45           The transformed grid is defined by the basis vectors i and j, which can
 46           be controlled with the :meth:`visualize_matrix_transformation` method.
 47           """
 48           painter = QPainter()
 49           painter.begin(self)
 50
 51           painter.setRenderHint(QPainter.Antialiasing)
 52           painter.setBrush(Qt.NoBrush)
 53
 54           self.draw_background(painter)
 55           self.draw_transformed_grid(painter)
 56           self.draw_vector_arrowheads(painter)
 57
 58           if self.display_settings.draw_determinant_parallelogram:
 59               self.draw_determinant_parallelogram(painter)
 60
 61           painter.end()
 62           event.accept()
```

I then wanted to change the determinant parallelogram to be blue when it's positive and red when it's negative. I did this by just checking the sign of the determinant and changing the colour accordingly.

```
      # cc75c7dc85e941540f7e98fe027d0657ad5462b8
      # src/lintrans/gui/plots/classes.py

140   class VectorGridPlot(BackgroundPlot):
```

```
...
385         def draw_determinant_parallelogram(self, painter: QPainter) -> None:
386             """Draw the parallelogram of the determinant of the matrix."""
387             det = np.linalg.det(np.array([
388                 [self.point_i[0], self.point_j[0]],
389                 [self.point_i[1], self.point_j[1]]
390             ]))
391
392             if det == 0:
393                 return
394
395             path = QPainterPath()
396             path.moveTo(*self.canvas_origin)
397             path.lineTo(*self.canvas_coords(*self.point_i))
398             path.lineTo(*self.canvas_coords(self.point_i[0] + self.point_j[0], self.point_i[1] + self.point_j[1]))
399             path.lineTo(*self.canvas_coords(*self.point_j))
400
401             color = (16, 235, 253) if det > 0 else (253, 34, 16)
402             brush = QBrush(QColor(*color, alpha=128), Qt.SolidPattern)
403
404             painter.fillPath(path, brush)
```

I then had the determinant parallelogram for positive and negative determinants.
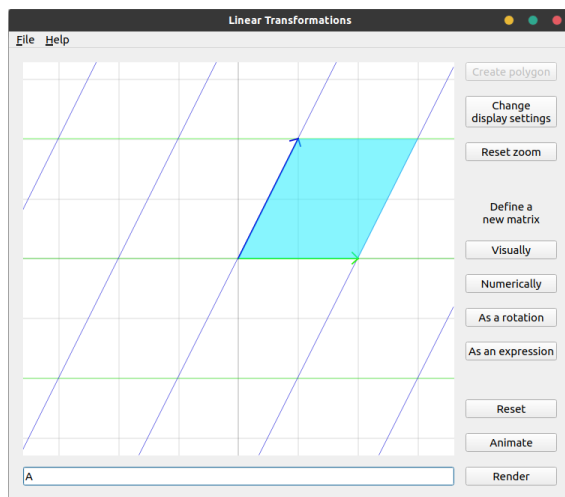
Figure 3.5.2: The blue parallelogram
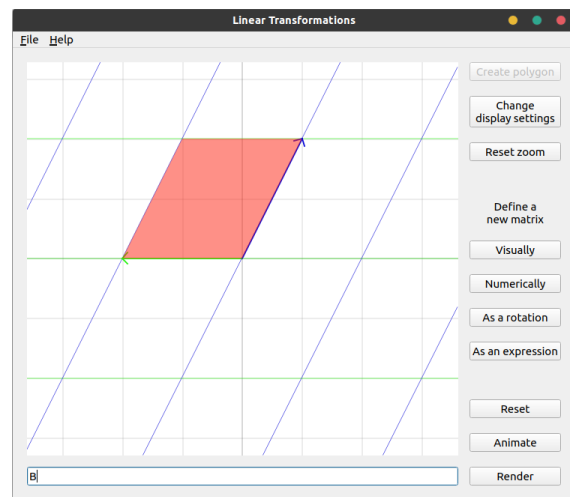
Figure 3.5.3: The red parallelogram

3.5.5   Adding the determinant text

Seeing the determinant as a shape is one thing, but knowing its exact value is also often very useful. To do this, I had to add a variable in the `DisplaySettings` for it, add a checkbox in the `DisplaySettingsDialog`, and create a method to actually draw the text in the right place, which I can call from `paintEvent()`.

```
# e344e50eccfd87c0834cfbdf459f0dd1d555fcd6
# src/lintrans/gui/settings.py

  9     class DisplaySettings:
...
 35         draw_determinant_text: bool = True
 36         """This controls whether we should write the text value of the determinant inside the parallelogram.
 37
 38         The text only gets draw if :attr:`draw_determinant_parallelogram` is also True.
 39         """


# e344e50eccfd87c0834cfbdf459f0dd1d555fcd6
# src/lintrans/gui/dialogs/settings.py
```

```
 63        class DisplaySettingsDialog(SettingsDialog):
...
 66            def __init__(self, display_settings: DisplaySettings, *args, **kwargs):
...
108                self.checkbox_draw_determinant_text = QCheckBox(self)
109                self.checkbox_draw_determinant_text.setText('Draw determinant text')
110                self.checkbox_draw_determinant_text.setToolTip(
111                    'Write the text value of the determinant inside the parallelogram'
112                )


           # e344e50eccfd87c0834cfbdf459f0dd1d555fcd6
           # src/lintrans/gui/plots/classes.py

142        class VectorGridPlot(BackgroundPlot):
...
416            def draw_determinant_text(self, painter: QPainter) -> None:
417                """Write the string value of the determinant in the middle of the parallelogram."""
418                painter.setPen(QPen(QColor(0, 0, 0), self.width_vector_line))
419                painter.drawText(
420                    *self.canvas_coords(
421                        (self.point_i[0] + self.point_j[0]) / 2,
422                        (self.point_i[1] + self.point_j[1]) / 2
423                    ),
424                    f'{self.det:.2f}'
425                )
```

It doesn't make much sense to show the text without also showing the parallelogram, so we should only show the text when the parallelogram is also being show, and the checkbox for the text should only be clickable when the parallelogram is enabled.

To do this, I created an `update_gui()` method which gets called when the parallelogram checkbox is clicked. This method will enable or disable the text checkbox appropriately.

```
           # e344e50eccfd87c0834cfbdf459f0dd1d555fcd6
           # src/lintrans/gui/plots/widgets.py

 13        class VisualizeTransformationWidget(VectorGridPlot):
...
 42            def paintEvent(self, event: QPaintEvent) -> None:
...
 58                if self.display_settings.draw_determinant_parallelogram:
 59                    self.draw_determinant_parallelogram(painter)
 60
 61                    if self.display_settings.draw_determinant_text:
 62                        self.draw_determinant_text(painter)


           # 517773e1ace0dc4485c425134cd36ba482ba65df
           # src/lintrans/gui/dialogs/settings.py

 63        class DisplaySettingsDialog(SettingsDialog):
...
 66            def __init__(self, display_settings: DisplaySettings, *args, **kwargs):
...
107                self.checkbox_draw_determinant_parallelogram.clicked.connect(self.update_gui)
...
173            def update_gui(self) -> None:
174                """Update the GUI according to other widgets in the GUI.
175
176                For example, this method updates which checkboxes are enabled based on the values of other checkboxes.
177                """
178                self.checkbox_draw_determinant_text.setEnabled(self.checkbox_draw_determinant_parallelogram.isChecked())
```

3.6   Fixing bugs and adding polish

3.6.1   Fixing an animation crash

The scaling logic in 3.3.6 creates a matrix $\mathbf{A}$ which is the start matrix plus some proportion of the difference between the target and start matrices. It then defines matrix $\mathbf{B}$ to be the matrix A normalised to have a determinant of 1. We then divide by $\det(\mathbf{B})$ to get matrix $\mathbf{C}$, which we then render.

This works very well for most matrices, but if we're animating from $\mathbf{I}$ to $-\mathbf{I}$ for example, then we can get the following problem:

When we're halfway through the animation, $p = \frac{1}{2}$.

$$\mathbf{A} = \mathbf{S} + p(\mathbf{T} - \mathbf{S})$$
$$= \mathbf{I} + \frac{1}{2}(-\mathbf{I} - \mathbf{I})$$
$$= \mathbf{I} + \frac{-1}{\cancel{2}}\cancel{2}\mathbf{I}$$
$$= \mathbf{I} - \mathbf{I} = \mathbf{0}$$

I'm using $\mathbf{I}$ as an example here, but this can happen with the right p for many matrix pairs. Since $\mathbf{A} = \mathbf{0}$, $\det(\mathbf{A}) = 0$. We check for this case already when we find c:

```
# f7a91cdc35695f8fb9269b17bc103e42578072bd
# src/lintrans/gui/main_window.py
```

```
367                    if det_a == 0:
368                        c = 0
369                    else:
370                        c = np.sqrt(abs(det_start / det_a))
```

But if $\det(\mathbf{A}) = 0$, then $c = 0$ and $\det(\mathbf{B}) = 0$, so we also need to check that before we divide by it.

Old:

```
# f7a91cdc35695f8fb9269b17bc103e42578072bd
# src/lintrans/gui/main_window.py
```

```
383                    scalar = 1 + proportion * (np.sqrt(abs(det_target / det_b)) - 1)
384                    matrix_to_render = scalar * matrix_b
```

New:

```
# 4383808a4cc29d192c55aca56161d8affda8c9a7
# src/lintrans/gui/main_window.py
```

```
384                    # That is all of course, if we can do that
385                    # We'll crash if we try to do this with det(B) == 0
386                    if det_b != 0:
387                        scalar = 1 + proportion * (np.sqrt(abs(det_target / det_b)) - 1)
388                        matrix_to_render = scalar * matrix_b
389
390                    else:
391                        matrix_to_render = matrix_a
```

This change fixes a division by zero bug, which eliminates a possible crash here.

3.6.2 Limiting parallel lines

If you try to render a matrix like `0.01Irot(45)`, then the app ends up drawing as many parallel lines as it can physically fit in the viewport. This leads to a lot of lag, especially when zoomed out far. To fix this, I just introduced a maximum number of parallel lines. I chose 150 as a number that was big enough to have enough parallel lines for matrices that need a lot, while also causing virtually no lag.

```
# bd9aaa2e3037214f65d0fc1d12d67db35af0e5ec
# src/lintrans/gui/plots/classes.py
142     class VectorGridPlot(BackgroundPlot):
...
151         def __init__(self, *args, **kwargs):
...
169             self.max_parallel_lines = 150
```

```
# bd9aaa2e3037214f65d0fc1d12d67db35af0e5ec
# src/lintrans/gui/plots/classes.py
142     class VectorGridPlot(BackgroundPlot):
...
191         def draw_parallel_lines(self, painter: QPainter, vector: tuple[float, float], point: tuple[float, float]) ->
        ↪   None:
...
230             # Draw vertical lines
231             elif abs(vector_x) < 1e-12:
232                 painter.drawLine(self.canvas_x(0), 0, self.canvas_x(0), self.height())
233
234                 for i in range(max(abs(int(max_x / point_x)), self.max_parallel_lines)):
235                     painter.drawLine(
236                         self.canvas_x((i + 1) * point_x),
237                         0,
238                         self.canvas_x((i + 1) * point_x),
239                         self.height()
240                     )
241                     painter.drawLine(
242                         self.canvas_x(-1 * (i + 1) * point_x),
243                         0,
244                         self.canvas_x(-1 * (i + 1) * point_x),
245                         self.height()
246                     )
247
248             # Draw horizontal lines
249             elif abs(vector_y) < 1e-12:
250                 painter.drawLine(0, self.canvas_y(0), self.width(), self.canvas_y(0))
251
252                 for i in range(max(abs(int(max_y / point_y)), self.max_parallel_lines)):
253                     painter.drawLine(
254                         0,
255                         self.canvas_y((i + 1) * point_y),
256                         self.width(),
257                         self.canvas_y((i + 1) * point_y)
258                     )
259                     painter.drawLine(
260                         0,
261                         self.canvas_y(-1 * (i + 1) * point_y),
262                         self.width(),
263                         self.canvas_y(-1 * (i + 1) * point_y)
264                     )
265
266             # If the line is oblique, then we can use y = mx + c
267             else:
268                 m = vector_y / vector_x
269                 c = point_y - m * point_x
270
271                 self.draw_oblique_line(painter, m, 0)
272
273                 # We don't want to overshoot the max number of parallel lines,
274                 # but we should also stop looping as soon as we can't draw any more lines
```

```
275                  for i in range(1, self.max_parallel_lines + 1):
276                      if not self.draw_pair_of_oblique_lines(painter, m, i * c):
277                          break
```

The idea behind this code is just to limit the maximum number of parallel lines that get drawn. It works perfectly for oblique lines, but there's a small bug for orthogonal lines that I never noticed. I just forgot to test it.

On lines 234 and 252, I call the built-in `max()` function with the maximum number of parallel lines and the total number of lines that could fit in the viewport. This should be a call to `min()` instead. I fixed this before releasing it for my end users, but it took an embarrassingly long time to notice something this simple.
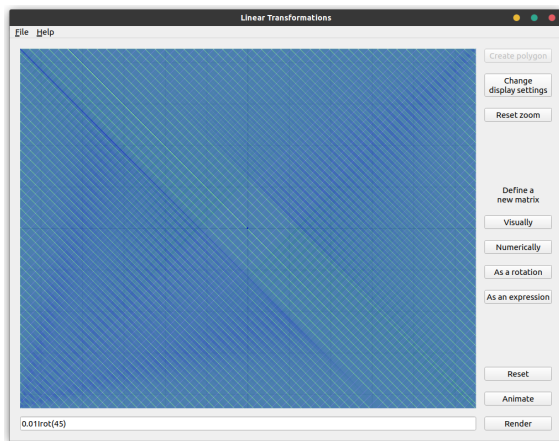
Figure 3.6.1: The old version with too many parallel lines.
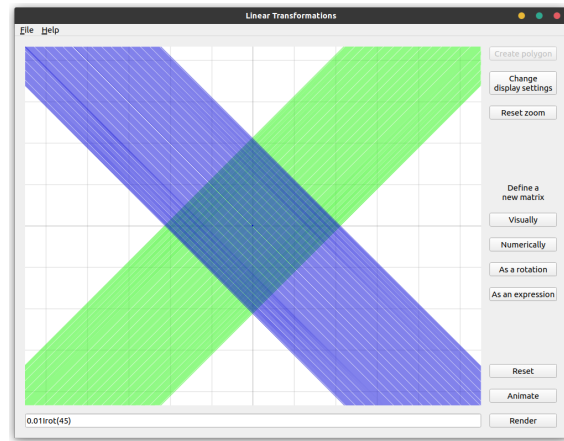
Figure 3.6.2: The fixed version with a maximum number of parallel lines.

3.6.3 Giving focus to the expression box

It would be quite nice to be able to just start typing an expression after defining a matrix or changing display settings. To do this, we can simply set the app's focus on the expression box after either of these actions.

Additionally, it would be nice to update the render buttons at the same time. That would allow the user to use a matrix in an expression, then define it, and be able to render the expression as soon as they close the dialog (assuming the expression is valid).

```
     # bd7f8ba18266a8a095549d815dcfe6f24de514b6
     # src/lintrans/gui/main_window.py

27   class LintransMainWindow(QMainWindow):
...
438      def assign_matrix_wrapper(self, matrix_wrapper: MatrixWrapper) -> None:
439          """Assign a new value to ``self.matrix_wrapper`` and give the expression box focus.
440
441          :param matrix_wrapper: The new value of the matrix wrapper to assign
442          :type matrix_wrapper: MatrixWrapper
443          """
444          self.matrix_wrapper = matrix_wrapper
445          self.lineedit_expression_box.setFocus()
446          self.update_render_buttons()
...
454      def assign_display_settings(self, display_settings: DisplaySettings) -> None:
455          """Assign a new value to ``self.plot.display_settings`` and give the expression box focus."""
456          self.plot.display_settings = display_settings
457          self.plot.update()
```

```
458                self.lineedit_expression_box.setFocus()
459                self.update_render_buttons()
```

3.6.4 Fixing a crash when animating singular matrices in sequence

If we have a matrix **A** defined as $\begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}$, then when we try to render `A^-1`, we get a pop-up dialog box saying that we can't take the inverse of a singular matrix. This is good, since if NumPy just took the inverse blindly, it would crash. When we try to animate `A^-1`, we get the same pop-up box. When we try to use it in an animation sequence, however, like `rot(45),A^-1`, we don't check if each element of the sequence for singularity, so NumPy takes the inverse blindly and the whole app crashes. This is bad.

To fix this, we can simply catch the error when trying to evaluate the element in the sequence.

```
     # 8db0df1d9d6a1be1f15a6f705e779d982db9ee29
     # src/lintrans/gui/main_window.py

 27  class LintransMainWindow(QMainWindow):
...
287      def animate_expression(self) -> None:
...
300          if ',' in text:
301              current_matrix = matrix_start
302
303              # For each expression in the list, right multiply it by the current matrix,
304              # and animate from the current matrix to that new matrix
305              for expr in text.split(',')[::-1]:
306                  try:
307                      new_matrix = self.matrix_wrapper.evaluate_expression(expr) @ current_matrix
308                  except linalg.LinAlgError:
309                      self.show_error_message('Singular matrix', 'Cannot take inverse of singular matrix')
310                      return
```

3.6.5 Allowing animations to be cancelled

Currently, if you try to reset the viewport partway through an animation, it just resets the basis vectors for a tick, but then they start moving again, because the animation loop is still running. To fix this, we can track whether we should be animating using an instance variable, set it to false when the user hits reset, and break out of the animation loop when it's false.

```
     # b665bc59ec99664ed7b2c17f94e76ae49c6eb331
     # src/lintrans/gui/main_window.py

 27  class LintransMainWindow(QMainWindow):
...
 33      def __init__(self):
...
 45          self.animating: bool = False
 46          self.animating_sequence: bool = False
...
269      def reset_transformation(self) -> None:
270          """Reset the visualized transformation back to the identity."""
271          self.plot.visualize_matrix_transformation(self.matrix_wrapper['I'])
272          self.animating = False
273          self.animating_sequence = False
274          self.plot.update()
...
292      def animate_expression(self) -> None:
...
304          # If there's commas in the expression, then we want to animate each part at a time
305          if ',' in text:
306              current_matrix = matrix_start
```

```
307                    self.animating_sequence = True
308
309                    # For each expression in the list, right multiply it by the current matrix,
310                    # and animate from the current matrix to that new matrix
311                    for expr in text.split(',')[::-1]:
312                        try:
313                            new_matrix = self.matrix_wrapper.evaluate_expression(expr) @ current_matrix
314                        except linalg.LinAlgError:
315                            self.show_error_message('Singular matrix', 'Cannot take inverse of singular matrix')
316                            return
317
318                        if not self.animating_sequence:
319                            break
320
321                        self.animate_between_matrices(current_matrix, new_matrix)
322                        current_matrix = new_matrix
323
324                        # Here we just redraw and allow for other events to be handled while we pause
325                        self.plot.update()
326                        QApplication.processEvents()
327                        QThread.msleep(self.plot.display_settings.animation_pause_length)
328
329                    self.animating_sequence = False
...
360        def animate_between_matrices(self, matrix_start: MatrixType, matrix_target: MatrixType, steps: int = 100) ->
    ↪    None:
...
365            self.animating = True
366
367            for i in range(0, steps + 1):
368                if not self.animating:
369                    break
370
...
429            self.animating = False
```

Here, `self.animating_sequence` is whether a sequence is being animated, and `self.animating` is whether an individual matrix is currently being animated. An individual matrix means a matrix on its own, or a single element in a sequence. That means that `self.animating` can be set and unset multiple times in a single sequence.

3.6.6   Validating expression input

The user can only render or animate an expression if it's actually valid, as discussed in §3.1.3, and the render and animate buttons will be greyed out if the expression is invalid. But they can still type anything into the box.

It was at this point that I learned about the `QValidator` class[44]. This class allows me to control what the user can actually type. Using the implementation below, they can only enter characters that are allowed in valid matrix expressions.

```
# f73575c017548d754e4171449344a52cb44b7ef4
# src/lintrans/gui/main_window.py

28    class LintransMainWindow(QMainWindow):
...
34        def __init__(self):
...
125            self.lineedit_expression_box.setValidator(MatrixExpressionValidator(self))


# f73575c017548d754e4171449344a52cb44b7ef4
# src/lintrans/gui/validate.py

1     """This simple module provides a :class:`MatrixExpressionValidator` class to validate matrix expression input."""
2
```

```
3      from __future__ import annotations
4
5      import re
6
7      from PyQt5.QtGui import QValidator
8
9      from lintrans.matrices import parse
10
11
12     class MatrixExpressionValidator(QValidator):
13         """This class validates matrix expressions in an Qt input box."""
14
15         def validate(self, text: str, pos: int) -> tuple[QValidator.State, str, int]:
16             """Validate the given text according to the rules defined in the :mod:`lintrans.matrices` module."""
17             clean_text = re.sub(r'[\sA-Z\d.rot()^{},+-]', '', text)
18
19             if clean_text == '':
20                 if parse.validate_matrix_expression(clean_text):
21                     return QValidator.Acceptable, text, pos
22                 else:
23                     return QValidator.Intermediate, text, pos
24
25             return QValidator.Invalid, text, pos
26
```

I also then added validators to the definition dialogs, to make sure that users can only enter valid input. Qt5 provides some basic validators already, for things like integers and floating point numbers (called **double** in C++, equivalent to `float` in Python).

```
       # a2fd14b99fa752a18b42352a01142ffbc2600570
       # src/lintrans/gui/dialogs/define_new_matrix.py

213    class DefineNumericallyDialog(DefineDialog):
...
225            # tl = top left, br = bottom right, etc.
226            self.element_tl = QtWidgets.QLineEdit(self)
227            self.element_tl.textChanged.connect(self.update_confirm_button)
228            self.element_tl.setValidator(QDoubleValidator())
229
230            self.element_tr = QtWidgets.QLineEdit(self)
231            self.element_tr.textChanged.connect(self.update_confirm_button)
232            self.element_tr.setValidator(QDoubleValidator())
233
234            self.element_bl = QtWidgets.QLineEdit(self)
235            self.element_bl.textChanged.connect(self.update_confirm_button)
236            self.element_bl.setValidator(QDoubleValidator())
237
238            self.element_br = QtWidgets.QLineEdit(self)
239            self.element_br.textChanged.connect(self.update_confirm_button)
240            self.element_br.setValidator(QDoubleValidator())
...
299    class DefineAsARotationDialog(DefineDialog):
...
314            self.lineedit_angle = QtWidgets.QLineEdit(self)
315            self.lineedit_angle.setPlaceholderText('angle')
316            self.lineedit_angle.textChanged.connect(self.update_confirm_button)
317            self.lineedit_angle.setValidator(QDoubleValidator())
...
358    class DefineAsAnExpressionDialog(DefineDialog):
...
372            self.lineedit_expression_box = QtWidgets.QLineEdit(self)
373            self.lineedit_expression_box.setPlaceholderText('Enter matrix expression...')
374            self.lineedit_expression_box.textChanged.connect(self.update_confirm_button)
375            self.lineedit_expression_box.setValidator(MatrixExpressionValidator())
```

3.6.7 Adding keyboard shortcuts

Keyboard shortcuts are often very useful and can make the process of using software much more efficient if you get good at using the shortcuts. On this note, I decided to add keyboard shortcuts to the display settings dialog.

Qt5 lets you use a & character in the text of a widget to act on the letter following it. This letter becomes underlined in the text, and the user can hold Alt and press this letter to activate the widget. I also want to be able to toggle the checkboxes by just pressing the letter without holding Alt, so I had to implement this myself with a dictionary and custom override of keyPressEvent().

```python
# 67d43a364ee2605b95b8caca9f1e4eb714cbb7c6
# src/lintrans/gui/dialogs/settings.py

63    class DisplaySettingsDialog(SettingsDialog):
64        """The dialog to allow the user to edit the display settings."""
65
66        def __init__(self, display_settings: DisplaySettings, *args, **kwargs):
67            """Create the widgets and layout of the dialog.
68
69            :param DisplaySettings display_settings: The :class:`lintrans.gui.settings.DisplaySettings` object to mutate
70            """
71            super().__init__(*args, **kwargs)
72
73            self.display_settings = display_settings
74            self.setWindowTitle('Change display settings')
75
76            self.dict_checkboxes: dict[str, QCheckBox] = dict()
77
78            # === Create the widgets
79
80            # Animations
81
82            self.checkbox_smoothen_determinant = QCheckBox(self)
83            self.checkbox_smoothen_determinant.setText('&Smoothen determinant')
84            self.checkbox_smoothen_determinant.setToolTip(
85                'Smoothly animate the determinant transition during animation (if possible)'
86            )
87            self.dict_checkboxes['s'] = self.checkbox_smoothen_determinant
88
89            self.checkbox_applicative_animation = QCheckBox(self)
90            self.checkbox_applicative_animation.setText('&Applicative animation')
91            self.checkbox_applicative_animation.setToolTip(
92                'Animate the new transformation applied to the current one,\n'
93                'rather than just that transformation on its own'
94            )
95            self.dict_checkboxes['a'] = self.checkbox_applicative_animation
96
97            self.label_animation_pause_length = QtWidgets.QLabel(self)
98            self.label_animation_pause_length.setText('Animation pause length (ms)')
99            self.label_animation_pause_length.setToolTip(
100               'How many milliseconds to pause for in comma-separated animations'
101           )
102
103           self.lineedit_animation_pause_length = QtWidgets.QLineEdit(self)
104           self.lineedit_animation_pause_length.setValidator(QIntValidator(1, 999, self))
105
106           # Matrix info
107
108           self.checkbox_draw_determinant_parallelogram = QCheckBox(self)
109           self.checkbox_draw_determinant_parallelogram.setText('Draw &determinant parallelogram')
110           self.checkbox_draw_determinant_parallelogram.setToolTip(
111               'Shade the parallelogram representing the determinant of the matrix'
112           )
113           self.checkbox_draw_determinant_parallelogram.clicked.connect(self.update_gui)
114           self.dict_checkboxes['d'] = self.checkbox_draw_determinant_parallelogram
115
116           self.checkbox_draw_determinant_text = QCheckBox(self)
117           self.checkbox_draw_determinant_text.setText('Draw determinant &text')
118           self.checkbox_draw_determinant_text.setToolTip(
```

```
119                         'Write the text value of the determinant inside the parallelogram'
120                 )
121             self.dict_checkboxes['t'] = self.checkbox_draw_determinant_text
122
123             # === Arrange the widgets in QGroupBoxes
124
125             # Animations
126
127             self.hlay_animation_pause_length = QHBoxLayout()
128             self.hlay_animation_pause_length.addWidget(self.label_animation_pause_length)
129             self.hlay_animation_pause_length.addWidget(self.lineedit_animation_pause_length)
130
131             self.vlay_groupbox_animations = QVBoxLayout()
132             self.vlay_groupbox_animations.setSpacing(20)
133             self.vlay_groupbox_animations.addWidget(self.checkbox_smoothen_determinant)
134             self.vlay_groupbox_animations.addWidget(self.checkbox_applicative_animation)
135             self.vlay_groupbox_animations.addLayout(self.hlay_animation_pause_length)
136
137             self.groupbox_animations = QGroupBox('Animations', self)
138             self.groupbox_animations.setLayout(self.vlay_groupbox_animations)
139
140             # Matrix info
141
142             self.vlay_groupbox_matrix_info = QVBoxLayout()
143             self.vlay_groupbox_matrix_info.setSpacing(20)
144             self.vlay_groupbox_matrix_info.addWidget(self.checkbox_draw_determinant_parallelogram)
145             self.vlay_groupbox_matrix_info.addWidget(self.checkbox_draw_determinant_text)
146
147             self.groupbox_matrix_info = QGroupBox('Matrix info', self)
148             self.groupbox_matrix_info.setLayout(self.vlay_groupbox_matrix_info)
149
150             self.vlay_options.addWidget(self.groupbox_animations)
151             self.vlay_options.addWidget(self.groupbox_matrix_info)
152
153             # Finally, we load the current settings and update the GUI
154             self.load_settings()
155             self.update_gui()
...
188     def keyPressEvent(self, event: QKeyEvent) -> None:
189         """Handle a ``QKeyEvent`` by manually activating toggling checkboxes.
190
191         Qt handles these shortcuts automatically and allows the user to do ``Alt + Key``
192         to activate a simple shortcut defined with ``&``. However, I like to be able to
193         just hit ``Key`` and have the shortcut activate.
194         """
195         letter = event.text().lower()
196         key = event.key()
197
198         if letter in self.dict_checkboxes:
199             self.dict_checkboxes[letter].animateClick()
200
201         # Return or keypad enter
202         elif key == 0x01000004 or key == 0x01000005:
203             self.button_confirm.click()
204
205         # Escape
206         elif key == 0x01000000:
207             self.button_cancel.click()
208
209         else:
210             event.ignore()
```

3.6.8 Centering text in the determinant parallelogram

The text in the determinant parallelogram is the numerical value of the determinant. Currently, it's not centered. It's drawn by just writing the text at a point, chosen to be the centre of the parallelogram. The `QPainter` class uses this point as the start of the baseline of the text, so it's effectively the bottom left corner.

```
# 67d43a364ee2605b95b8caca9f1e4eb714cbb7c6
# src/lintrans/gui/plots/classes.py

142  class VectorGridPlot(BackgroundPlot):
...
419      def draw_determinant_text(self, painter: QPainter) -> None:
420          """Write the string value of the determinant in the middle of the parallelogram."""
421          painter.setPen(QPen(QColor(0, 0, 0), self.width_vector_line))
422          painter.drawText(
423              *self.canvas_coords(
424                  (self.point_i[0] + self.point_j[0]) / 2,
425                  (self.point_i[1] + self.point_j[1]) / 2
426              ),
427              f'{self.det:.2f}'
428          )
```

Obviously, this text will look better if it's centered. To do this, we can create a bounding rectangle around the parallelogram and get the painter to draw the text in the centre of that rectangle.

We build the rectangle by getting the coordinates of each vertex of the parallelogram. Then the top left corner is the minimum x coordinate with the maximum y coordinate, and the bottom right corner is the maximum x with the minimum y.

```
# 9550416c0b273b16c90eb8d6319f5e17493ef9a8
# src/lintrans/gui/plots/classes.py

142  class VectorGridPlot(BackgroundPlot):
...
419      def draw_determinant_text(self, painter: QPainter) -> None:
420          """Write the string value of the determinant in the middle of the parallelogram."""
421          painter.setPen(QPen(QColor(0, 0, 0), self.width_vector_line))
422
423          # We're building a QRect that encloses the determinant parallelogram
424          # Then we can center the text in this QRect
425          coords: list[tuple[float, float]] = [
426              (0, 0),
427              self.point_i,
428              self.point_j,
429              (
430                  self.point_i[0] + self.point_j[0],
431                  self.point_i[1] + self.point_j[1]
432              )
433          ]
434
435          xs = [t[0] for t in coords]
436          ys = [t[1] for t in coords]
437
438          top_left = QPoint(*self.canvas_coords(min(xs), max(ys)))
439          bottom_right = QPoint(*self.canvas_coords(max(xs), min(ys)))
440
441          rect = QRectF(top_left, bottom_right)
442
443          painter.drawText(
444              rect,
445              Qt.AlignHCenter | Qt.AlignVCenter,
446              f'{self.det:.2f}'
447          )
```
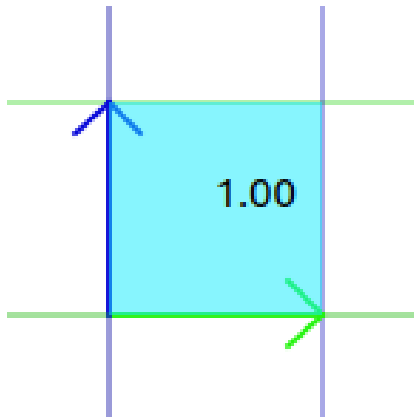
Figure 3.6.3: Text not centered.
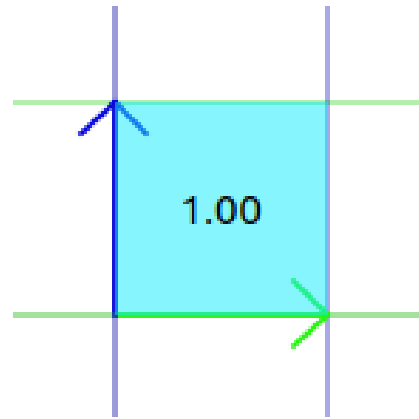
Figure 3.6.4: Text centered.

3.6.9   Defining matrices as expressions

Currently, you can "define" a matrix in terms of an expression, but it doesn't really define the matrix like that. Instead, it evaluates the expression immediately, and assigns that numerical result to the name you specified. It would be much better if the matrix could be actually defined as the expression, and then evaluated only when it needs to be used. Then, the user could have a matrix **M** defined as something like `0.5A^-1rot(45)B`, and it would always have that value, even if the user has changed the definition of **A** or **B** since defining **M**.

To do this, I'll have to completely change how matrices are stored and retrieved. The `MatrixWrapper` class contains a dictionary `self._matrices`, which currently maps `str` to `Optional[MatrixType]`, meaning that a matrix could be a 2×2 NumPy array, or nothing. I'm going to change this type to `Optional[Union[MatrixType, str]]`. This means that if a matrix exists, then it's either a 2×2 NumPy array or a string. We then check which one it is when we retrieve the matrix, and act accordingly. If it's an expression, then we evaluate and return the numerical result.

Here's the relevant parts of the old `MatrixWrapper` class:

```
# 9550416c0b273b16c90eb8d6319f5e17493ef9a8
# src/lintrans/matrices/wrapper.py

17    class MatrixWrapper:
...
35        def __init__(self):
36            """Initialise a :class:`MatrixWrapper` object with a dictionary of matrices which can be accessed."""
37            self._matrices: dict[str, Optional[MatrixType]] = {
38                'A': None, 'B': None, 'C': None, 'D': None,
39                'E': None, 'F': None, 'G': None, 'H': None,
40                'I': np.eye(2),  # I is always defined as the identity matrix
41                'J': None, 'K': None, 'L': None, 'M': None,
42                'N': None, 'O': None, 'P': None, 'Q': None,
43                'R': None, 'S': None, 'T': None, 'U': None,
44                'V': None, 'W': None, 'X': None, 'Y': None,
45                'Z': None
46            }
...
91        def __getitem__(self, name: str) -> Optional[MatrixType]:
92            """Get the matrix with the given name.
93
94            If it is a simple name, it will just be fetched from the dictionary. If the name is ``rot(x)``, with
95            a given angle in degrees, then we return a new matrix representing a rotation by that angle.
96
97            :param str name: The name of the matrix to get
98            :returns: The value of the matrix (may be None)
99            :rtype: Optional[MatrixType]
100
```

```
101                     :raises NameError: If there is no matrix with the given name
102                     """
103                     # Return a new rotation matrix
104                     if (match := re.match(r'rot\((-?\d*\.?\d*)\)', name)) is not None:
105                         return create_rotation_matrix(float(match.group(1)))
106
107                     if name not in self._matrices:
108                         raise NameError(f'Unrecognised matrix name "{name}"')
109
110                     # We copy the matrix before we return it so the user can't accidentally mutate the matrix
111                     return copy(self._matrices[name])
112
113         def __setitem__(self, name: str, new_matrix: Optional[MatrixType]) -> None:
114             """Set the value of matrix ``name`` with the new_matrix.
115
116             :param str name: The name of the matrix to set the value of
117             :param Optional[MatrixType] new_matrix: The value of the new matrix (may be None)
118
119             :raises NameError: If the name isn't a valid matrix name or is 'I'
120             :raises TypeError: If the matrix isn't a valid 2x2 NumPy array
121             """
122             if name not in self._matrices:
123                 raise NameError('Matrix name must be a single capital letter')
124
125             if name == 'I':
126                 raise NameError('Matrix name cannot be "I"')
127
128             if new_matrix is None:
129                 self._matrices[name] = None
130                 return
131
132             if not is_matrix_type(new_matrix):
133                 raise TypeError('Matrix must be a 2x2 NumPy array')
134
135             # All matrices must have float entries
136             a = float(new_matrix[0][0])
137             b = float(new_matrix[0][1])
138             c = float(new_matrix[1][0])
139             d = float(new_matrix[1][1])
140
141             self._matrices[name] = np.array([[a, b], [c, d]])
142
143         def is_valid_expression(self, expression: str) -> bool:
144             """Check if the given expression is valid, using the context of the wrapper.
145
146             This method calls :func:`lintrans.matrices.parse.validate_matrix_expression`, but also
147             ensures that all the matrices in the expression are defined in the wrapper.
148
149             :param str expression: The expression to validate
150             :returns: Whether the expression is valid in this wrapper
151             :rtype: bool
152             """
153             # Get rid of the transposes to check all capital letters
154             new_expression = expression.replace('^T', '').replace('^{T}', '')
155
156             # Make sure all the referenced matrices are defined
157             for matrix in {x for x in new_expression if re.match('[A-Z]', x)}:
158                 if self[matrix] is None:
159                     return False
160
161             return validate_matrix_expression(expression)
```

And here's the new version, which supports matrices defined as expressions:

```
# 01e866a74cf0f02ecba6438763d43e6eb90fe218
# src/lintrans/matrices/wrapper.py

17      class MatrixWrapper:
...
38          def __init__(self):
39              """Initialise a :class:`MatrixWrapper` object with a dictionary of matrices which can be accessed."""
```

```
40              self._matrices: dict[str, Optional[Union[MatrixType, str]]] = {
41                  'A': None, 'B': None, 'C': None, 'D': None,
42                  'E': None, 'F': None, 'G': None, 'H': None,
43                  'I': np.eye(2),  # I is always defined as the identity matrix
44                  'J': None, 'K': None, 'L': None, 'M': None,
45                  'N': None, 'O': None, 'P': None, 'Q': None,
46                  'R': None, 'S': None, 'T': None, 'U': None,
47                  'V': None, 'W': None, 'X': None, 'Y': None,
48                  'Z': None
49              }
...
94          def __getitem__(self, name: str) -> Optional[MatrixType]:
95              """Get the matrix with the given name.
96
97              If it is a simple name, it will just be fetched from the dictionary. If the name is ``rot(x)``, with
98              a given angle in degrees, then we return a new matrix representing a rotation by that angle.
99
100             :param str name: The name of the matrix to get
101             :returns: The value of the matrix (may be None)
102             :rtype: Optional[MatrixType]
103
104             :raises NameError: If there is no matrix with the given name
105             """
106             # Return a new rotation matrix
107             if (match := re.match(r'rot\((-?\d*\.?\d*)\)', name)) is not None:
108                 return create_rotation_matrix(float(match.group(1)))
109
110             if name not in self._matrices:
111                 raise NameError(f'Unrecognised matrix name "{name}"')
112
113             # We copy the matrix before we return it so the user can't accidentally mutate the matrix
114             matrix = copy(self._matrices[name])
115
116             if isinstance(matrix, str):
117                 return self.evaluate_expression(matrix)
118
119             return matrix
120
121         def __setitem__(self, name: str, new_matrix: Optional[Union[MatrixType, str]]) -> None:
122             """Set the value of matrix ``name`` with the new_matrix.
123
124             :param str name: The name of the matrix to set the value of
125             :param Optional[Union[MatrixType, str]] new_matrix: The value of the new matrix (may be None)
126
127             :raises NameError: If the name isn't a legal matrix name
128             :raises TypeError: If the matrix isn't a valid 2x2 NumPy array
129             """
130             if not (name in self._matrices and name != 'I'):
131                 raise NameError('Matrix name is illegal')
132
133             if new_matrix is None:
134                 self._matrices[name] = None
135                 return
136
137             if isinstance(new_matrix, str):
138                 if self.is_valid_expression(new_matrix):
139                     self._matrices[name] = new_matrix
140                     return
141
142             if not is_matrix_type(new_matrix):
143                 raise TypeError('Matrix must be a 2x2 NumPy array')
144
145             # All matrices must have float entries
146             a = float(new_matrix[0][0])
147             b = float(new_matrix[0][1])
148             c = float(new_matrix[1][0])
149             d = float(new_matrix[1][1])
150
151             self._matrices[name] = np.array([[a, b], [c, d]])
152
153         def get_expression(self, name: str) -> Optional[str]:
154             """If the named matrix is defined as an expression, return that expression, else return None.
155
```

```
156                :param str name: The name of the matrix
157                :returns: The expression that the matrix is defined as, or None
158                :rtype: Optional[str]
159
160                :raises NameError: If the name is invalid
161                """
162                if name not in self._matrices:
163                    raise NameError('Matrix must have a legal name')
164
165                matrix = self._matrices[name]
166                if isinstance(matrix, str):
167                    return matrix
168
169                return None
170
171        def is_valid_expression(self, expression: str) -> bool:
172            """Check if the given expression is valid, using the context of the wrapper.
173
174            This method calls :func:`lintrans.matrices.parse.validate_matrix_expression`, but also
175            ensures that all the matrices in the expression are defined in the wrapper.
176
177            :param str expression: The expression to validate
178            :returns: Whether the expression is valid in this wrapper
179            :rtype: bool
180            """
181            # Get rid of the transposes to check all capital letters
182            new_expression = expression.replace('^T', '').replace('^{T}', '')
183
184            # Make sure all the referenced matrices are defined
185            for matrix in {x for x in new_expression if re.match('[A-Z]', x)}:
186                if self[matrix] is None:
187                    return False
188
189                if (expr := self.get_expression(matrix)) is not None:
190                    if not self.is_valid_expression(expr):
191                        return False
192
193            return validate_matrix_expression(expression)
```

One of the more subtle things added here is on lines 189-191. When checking if an expression is valid
in the context of the wrapper, we have to make sure all the referenced matrices are actually defined,
but if any of those matrices are defined as an expression, then obviously that expression has to be
valid as well. This recursion means that all references to matrices must be valid, even traversing down
through matrices that are defined as expressions.

I also added some unit tests to automatically test this new feature.

```
     # 239bcbfd1dde3f7623318d03e8544dd67dc02e3d
     # tests/matrices/matrix_wrapper/test_setitem_and_getitem.py

42   def test_set_expression(test_wrapper: MatrixWrapper) -> None:
43       """Test that MatrixWrapper.__setitem__() can accept a valid expression."""
44       test_wrapper['N'] = 'A^2'
45       test_wrapper['O'] = 'BA+2C'
46       test_wrapper['P'] = 'E^T'
47       test_wrapper['Q'] = 'C^-1B'
48       test_wrapper['R'] = 'A^{2}3B'
49       test_wrapper['S'] = 'N^-1'
50       test_wrapper['T'] = 'PQP^-1'
51
52       with pytest.raises(TypeError):
53           test_wrapper['U'] = 'A+1'
54           test_wrapper['V'] = 'K'
55           test_wrapper['W'] = 'L^2'
56           test_wrapper['X'] = 'M^-1'
57
58
59   def test_simple_dynamic_evaluation(test_wrapper: MatrixWrapper) -> None:
60       """Test that expression-defined matrices are evaluated dynamically."""
61       test_wrapper['N'] = 'A^2'
```

```python
 62         test_wrapper['O'] = '4B'
 63         test_wrapper['P'] = 'A+C'
 64
 65         assert (test_wrapper['N'] == test_wrapper.evaluate_expression('A^2')).all()
 66         assert (test_wrapper['O'] == test_wrapper.evaluate_expression('4B')).all()
 67         assert (test_wrapper['P'] == test_wrapper.evaluate_expression('A+C')).all()
 68
 69         assert (test_wrapper.evaluate_expression('N^2 + 30') ==
 70                 la.matrix_power(test_wrapper.evaluate_expression('A^2'), 2) +
 71                 3 * test_wrapper.evaluate_expression('4B')
 72                 ).all()
 73         assert (test_wrapper.evaluate_expression('P^-1 - 3N0^2') ==
 74                 la.inv(test_wrapper.evaluate_expression('A+C')) -
 75                 (3 * test_wrapper.evaluate_expression('A^2')) @
 76                 la.matrix_power(test_wrapper.evaluate_expression('4B'), 2)
 77                 ).all()
 78
 79         test_wrapper['A'] = np.array([
 80             [19, -21.5],
 81             [84, 96.572]
 82         ])
 83         test_wrapper['B'] = np.array([
 84             [-0.993, 2.52],
 85             [1e10, 0]
 86         ])
 87         test_wrapper['C'] = np.array([
 88             [0, 19512],
 89             [1.414, 19]
 90         ])
 91
 92         assert (test_wrapper['N'] == test_wrapper.evaluate_expression('A^2')).all()
 93         assert (test_wrapper['O'] == test_wrapper.evaluate_expression('4B')).all()
 94         assert (test_wrapper['P'] == test_wrapper.evaluate_expression('A+C')).all()
 95
 96         assert (test_wrapper.evaluate_expression('N^2 + 30') ==
 97                 la.matrix_power(test_wrapper.evaluate_expression('A^2'), 2) +
 98                 3 * test_wrapper.evaluate_expression('4B')
 99                 ).all()
100         assert (test_wrapper.evaluate_expression('P^-1 - 3N0^2') ==
101                 la.inv(test_wrapper.evaluate_expression('A+C')) -
102                 (3 * test_wrapper.evaluate_expression('A^2')) @
103                 la.matrix_power(test_wrapper.evaluate_expression('4B'), 2)
104                 ).all()
105
106
107     def test_recursive_dynamic_evaluation(test_wrapper: MatrixWrapper) -> None:
108         """Test that dynamic evaluation works recursively."""
109         test_wrapper['N'] = 'A^2'
110         test_wrapper['O'] = '4B'
111         test_wrapper['P'] = 'A+C'
112
113         test_wrapper['Q'] = 'N^-1'
114         test_wrapper['R'] = 'P-40'
115         test_wrapper['S'] = 'NOP'
116
117         assert test_wrapper['Q'] == pytest.approx(test_wrapper.evaluate_expression('A^-2'))
118         assert test_wrapper['R'] == pytest.approx(test_wrapper.evaluate_expression('A + C - 16B'))
119         assert test_wrapper['S'] == pytest.approx(test_wrapper.evaluate_expression('A^{2}4BA + A^{2}4BC'))
120
121
122     def test_set_identity_error(new_wrapper: MatrixWrapper) -> None:
123         """Test that MatrixWrapper().__setitem__() raises a NameError when trying to assign to I."""
124         with pytest.raises(NameError):
125             new_wrapper['I'] = test_matrix
126
127
128     def test_set_name_error(new_wrapper: MatrixWrapper) -> None:
129         """Test that MatrixWrapper().__setitem__() raises a NameError when trying to assign to an invalid name."""
130         with pytest.raises(NameError):
131             new_wrapper['bad name'] = test_matrix
132             new_wrapper['123456'] = test_matrix
133             new_wrapper['Th15 Is an 1nV@l1D n@m3'] = test_matrix
134             new_wrapper['abc'] = test_matrix
```

```python
135                new_wrapper['a'] = test_matrix
136
137
138    def test_set_type_error(new_wrapper: MatrixWrapper) -> None:
139        """Test that MatrixWrapper().__setitem__() raises a TypeError when trying to set a non-matrix."""
140        with pytest.raises(TypeError):
141            new_wrapper['M'] = 12
142            new_wrapper['M'] = [1, 2, 3, 4, 5]
143            new_wrapper['M'] = [[1, 2], [3, 4]]
144            new_wrapper['M'] = True
145            new_wrapper['M'] = 24.3222
146            new_wrapper['M'] = 'This is totally a matrix, I swear'
147            new_wrapper['M'] = MatrixWrapper
148            new_wrapper['M'] = MatrixWrapper()
149            new_wrapper['M'] = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
150            new_wrapper['M'] = np.eye(100)


       # ea00703f19c13af86c39ae30170569819937fa31
       # tests/matrices/matrix_wrapper/test_misc.py

1      """Test the miscellaneous methods of the MatrixWrapper class."""
2
3      from lintrans.matrices import MatrixWrapper
4
5
6      def test_get_expression(test_wrapper: MatrixWrapper) -> None:
7          """Test the get_expression method of the MatrixWrapper class."""
8          test_wrapper['N'] = 'A^2'
9          test_wrapper['O'] = '4B'
10         test_wrapper['P'] = 'A+C'
11
12         test_wrapper['Q'] = 'N^-1'
13         test_wrapper['R'] = 'P-40'
14         test_wrapper['S'] = 'NOP'
15
16         assert test_wrapper.get_expression('A') is None
17         assert test_wrapper.get_expression('B') is None
18         assert test_wrapper.get_expression('C') is None
19         assert test_wrapper.get_expression('D') is None
20         assert test_wrapper.get_expression('E') is None
21         assert test_wrapper.get_expression('F') is None
22         assert test_wrapper.get_expression('G') is None
23
24         assert test_wrapper.get_expression('N') == 'A^2'
25         assert test_wrapper.get_expression('O') == '4B'
26         assert test_wrapper.get_expression('P') == 'A+C'
27
28         assert test_wrapper.get_expression('Q') == 'N^-1'
29         assert test_wrapper.get_expression('R') == 'P-40'
30         assert test_wrapper.get_expression('S') == 'NOP'
```

```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *]
$ pytest -v
============================================================== test session starts ==============================================================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml, testpaths: tests
collected 29 items

tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs0-True] PASSED                                                      [  3%]
tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs1-False] PASSED                                                     [  6%]
tests/gui/test_dialog_utility_functions.py::test_round_float PASSED                                                                       [ 10%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs0-True] PASSED                                [ 13%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs1-False] PASSED                               [ 17%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_matrix_expression PASSED                                                 [ 20%]
tests/matrices/test_rotation_matrices.py::test_create_rotation_matrix PASSED                                                              [ 24%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_matrix_addition PASSED                                             [ 27%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_two_matrix_multiplication PASSED                                   [ 31%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_identity_multiplication PASSED                                            [ 34%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_three_matrix_multiplication PASSED                                 [ 37%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_inverses PASSED                                                    [ 41%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_powers PASSED                                                      [ 44%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_transpose PASSED                                                   [ 48%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_rotation_matrices PASSED                                                  [ 51%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_multiplication_and_addition PASSED                                        [ 55%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_complicated_expressions PASSED                                            [ 58%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_value_errors PASSED                                                       [ 62%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_linalgerror PASSED                                                        [ 65%]
tests/matrices/matrix_wrapper/test_misc.py::test_get_expression PASSED                                                                    [ 68%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_get_matrix PASSED                                                   [ 72%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_get_name_error PASSED                                                     [ 75%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_set_matrix PASSED                                                   [ 79%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_expression PASSED                                                     [ 82%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_simple_dynamic_evaluation PASSED                                          [ 86%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_recursive_dynamic_evaluation PASSED                                       [ 89%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_identity_error PASSED                                                 [ 93%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_name_error PASSED                                                     [ 96%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_type_error PASSED                                                     [100%]

=============================================================== 29 passed in 0.36s ===============================================================
```

Figure 3.6.5: Running `pytest` with the new tests

I then had to fix a small bug where the `DefineAsAnExpressionDialog` would evaluate the expression before assigning it, so I had to change that to just assign the test instead.

```
# 54e10dbfd3a1f3a962955c7fa3908848f5bd95b0
# src/lintrans/gui/dialogs/define_new_matrix.py

343    class DefineAsAnExpressionDialog(DefineDialog):
...
388        def confirm_matrix(self) -> None:
389            """Evaluate the matrix expression and assign its value to the name in the combo box."""
390            self.matrix_wrapper[self.selected_letter] = self.lineedit_expression_box.text()
391            self.accept()
```

I also created a virtual method in the `DefineDialog` superclass, which standardised how dialogs load a matrix when it's selected in the drop-down. The numerical and visual definition dialogs already did this, but it was inconsistent, so I made it the same across all subclasses, and added it to the expression dialog.

```
# d1b60b20666ab9297cdbf675b6226587fd2e417f
# src/lintrans/gui/dialogs/define_new_matrix.py

59     class DefineDialog(QDialog):
...
69         def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
...
98             self.combobox_letter = QtWidgets.QComboBox(self)
99
100            for letter in ALPHABET_NO_I:
101                self.combobox_letter.addItem(letter)
102
103            self.combobox_letter.activated.connect(self.load_matrix)
...
134        def load_matrix(self, index: int) -> None:
135            """Load the selected matrix into the dialog.
136
137            This method is optionally able to be overridden. If it is not overridden,
138            then no matrix is loaded when selecting a name.
139
140            We have this method in the superclass so that we can define it as the slot
141            for the combobox.changed signal in this constructor, rather than having to
```

```
142                    define that in the constructor of every subclass.
143                    """
...
352        class DefineAsAnExpressionDialog(DefineDialog):
...
386            def load_matrix(self, index: int) -> None:
387                """If the selected matrix is defined an expression, load that expression into the box."""
388                name = ALPHABET_NO_I[index]
389
390                if (expr := self.matrix_wrapper.get_expression(name)) is not None:
391                    self.lineedit_expression_box.setText(expr)
392                else:
393                    self.lineedit_expression_box.setText('')
```

Unfortunately, my initial implementation of this had a few bugs, and I noticed a few hours later that if you first define **A** as anything concrete, then you can define **A** to be the expression A. Then, when you put it in the expression box, the app just crashes. This is because it recurs forever, since it doesn't realise that the definition of **A** is self-referential[14].

To fix this, I can check that the expression is valid and that it doesn't contain itself before assigning the expression to the matrix name.

```
# 742e0955e344deab2c9302ba9a6c7298ec4583d4
# src/lintrans/gui/dialogs/define_new_matrix.py

362        class DefineAsAnExpressionDialog(DefineDialog):
...
393            def update_confirm_button(self) -> None:
...
395                text = self.lineedit_expression_box.text()
396                valid_expression = self.matrix_wrapper.is_valid_expression(text)
397
398                self.button_confirm.setEnabled(valid_expression and self.selected_letter not in text)
```

I also added this logic directly to the wrapper, so that there was no risk of me creating this kind of bug elsewhere.

```
# e56a5a90034f8335b046dd1bf76321eb48892050
# src/lintrans/matrices/wrapper.py

 17        class MatrixWrapper:
...
125            def __setitem__(self, name: str, new_matrix: Optional[Union[MatrixType, str]]) -> None:
...
145                if isinstance(new_matrix, str):
146                    if self.is_valid_expression(new_matrix):
147                        if name not in new_matrix:
148                            self._matrices[name] = new_matrix
149                            return
150                        else:
151                            raise ValueError('Cannot define a matrix recursively')
```

While I was working with expressions so much, I realised that defining a matrix as a rotation was a bit redundant when you can just use an expression like `rot(45)`. I spoke to the teacher that's going to use `lintrans` when it's finished, and she said that radians aren't really needed. The radians checkbox was the only unique part of the `DefineAsARotationDialog` class. Since it's not important, I decided to remove the whole dialog.

---

[14]Obviously it doesn't actually recur forever, but Python stops recursion after 1000 levels and crashes the program.

3.7   Implementing eigenstuffs

It's not universal, but the word 'eigenstuffs' is common enough in mathematics that I'm comfortable using it here to mean eigenvalues, eigenvectors, and eigenlines, where an eigenline is the span of an eigenvector.

3.7.1   Drawing eigenvectors

An eigenvector $\mathbf{v}$ of a matrix $\mathbf{M}$ is a vector that satisfies the equation $\mathbf{Mv} = \lambda\mathbf{v}$ for some scalar λ. Thankfully, I don't have to worry about actually computing $\mathbf{v}$ or λ, since NumPy has the `numpy.linalg.eig()` function[30].

This function takes a square matrix and returns an array of eigenvalues (λ), and a matrix of their associated eigenvectors ($\mathbf{v}$). Some matrices don't have any real eigenvalues, but all 2×2 matrices will have 2 (possibly complex) eigenvalues, as a direct consequence of the Fundamental Theorem of Algebra[15]. We don't want to try to render an eigenvector if its eigenvalue is complex, so we have to check and only render the real ones. Python doesn't distinguish really between `float` and `complex` types, so we can just check the `.imag` property no matter what. If it's 0, then we keep the eigenvalue. NumPy normalizes the eigenvectors to have a length of 1, but I'd much prefer them to have a length equal to their associated eigenvalue. To do that, we can just multiply the eigenvectors by their eigenvalues.

We then just draw a vector, consisting of a line and an arrowhead, from the origin to the extended eigenvector.

```
# b8614334de5cba4b1a6d92508b08fa8bd2fe77c0
# src/lintrans/gui/plots/classes.py

142    class VectorGridPlot(BackgroundPlot):
...
151        def __init__(self, *args, **kwargs):
...
163            self.colour_eigen = QColor('#fff900')
...
450        def draw_eigenvectors(self, painter: QPainter) -> None:
451            """Draw the eigenvectors of the displayed matrix transformation."""
452            painter.setPen(QPen(self.colour_eigen, self.width_vector_line))
453
454            values, vectors = np.linalg.eig(self.matrix)
455            vectors = vectors.T
456
457            for value, vector in zip(values, vectors):
458                x = value * vector[0]
459                y = value * vector[1]
460
461                if x.imag != 0 or y.imag != 0:
462                    continue
463
464                painter.drawLine(*self.canvas_origin, *self.canvas_coords(x, y))
465                self.draw_arrowhead_away_from_origin(painter, (x, y))
```

```
# b8614334de5cba4b1a6d92508b08fa8bd2fe77c0
# src/lintrans/gui/plots/widgets.py

13    class VisualizeTransformationWidget(VectorGridPlot):
...
42        def paintEvent(self, event: QPaintEvent) -> None:
...
58            self.draw_eigenvectors(painter)
```

---

[15]$\mathbf{Mv} = \lambda\mathbf{v} \implies \mathbf{Mv} = \lambda\mathbf{Iv} \implies (\mathbf{M} - \lambda\mathbf{I})\mathbf{v} = \mathbf{0} \implies \det(\mathbf{M} - \lambda\mathbf{I}) = 0$ (since we only want non-zero vectors)
$\implies \begin{vmatrix} a - \lambda & b \\ c & d - \lambda \end{vmatrix} = 0 \implies (a - \lambda)(d - \lambda) - bc = 0 \implies \lambda^2 - (a + d)\lambda + (ad - bc) = 0$
$\implies \lambda$ has 2 solutions in $\mathbb{C}$ by the Fundamental Theorem of Algebra[15]

At this point in development, I didn't particularly care about the colours of various elements. It was more important to get things working first, so I ended up choosing this horrible yellow for the eigenvectors. It's clearly an awful choice for text, and it's not very good for the eigenvectors either, since it makes them hard to see against the white background. I wasn't really considering the usability features discussed in §2.4, but since I was the only user, and changing a few colours later on wouldn't be much work, I wasn't worried about it.
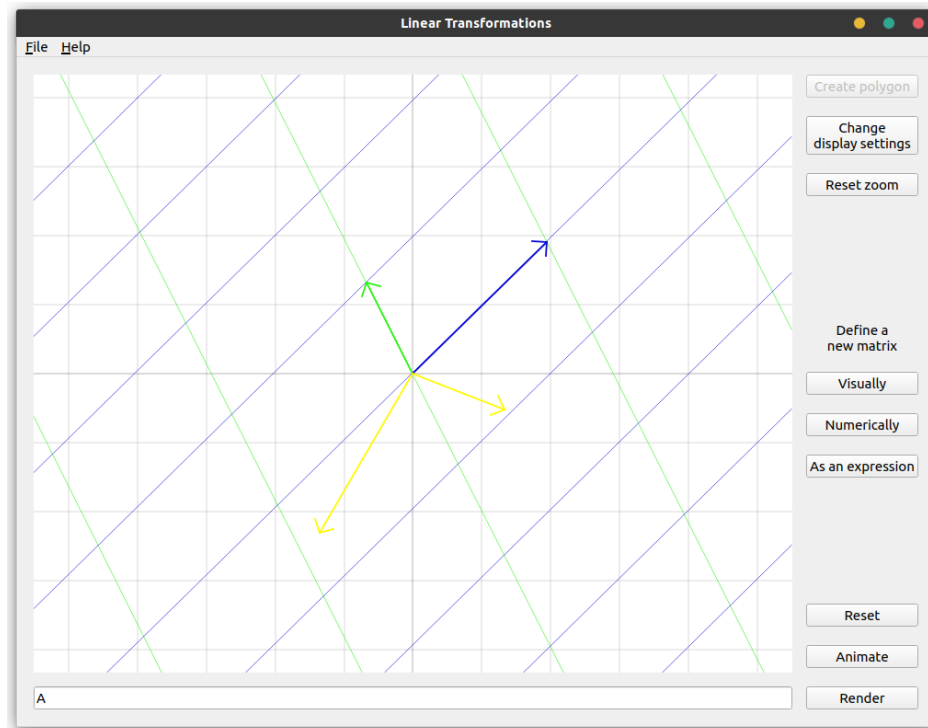
Figure 3.7.1: The eigenvectors being displayed for an arbitrary matrix

3.7.2 Adding display settings for eigenvectors

Once I'd got the eigenvectors working and being drawn, I wanted to be able to turn them on and off from the display settings. This was a simple case of just adding it to the `DisplaySettings` dataclass, adding the checkbox to the dialog, and only drawing the eigenvectors if this display setting is true.

```
# 3ebb5997a7a887e751b5f4f717aa5161ed013e62
# src/lintrans/gui/settings.py

9    class DisplaySettings:
...
41       draw_eigenvectors: bool = False
42       """This controls whether we should draw the eigenvectors of the transformation."""


# 3ebb5997a7a887e751b5f4f717aa5161ed013e62
# src/lintrans/gui/dialogs/settings.py

63   class DisplaySettingsDialog(SettingsDialog):
...
66       def __init__(self, display_settings: DisplaySettings, *args, **kwargs):
...
123          self.checkbox_draw_eigenvectors = QCheckBox(self)
124          self.checkbox_draw_eigenvectors.setText('Draw &eigenvectors')
125          self.checkbox_draw_eigenvectors.setToolTip('Draw the eigenvectors of the transformations')
126          self.dict_checkboxes['e'] = self.checkbox_draw_eigenvectors
```

```
     # 3ebb5997a7a887e751b5f4f717aa5161ed013e62
     # src/lintrans/gui/plots/widgets.py

13   class VisualizeTransformationWidget(VectorGridPlot):
...
42       def paintEvent(self, event: QPaintEvent) -> None:
...
58           if self.display_settings.draw_eigenvectors:
59               self.draw_eigenvectors(painter)
```

3.7.3   Refactoring drawing vectors

Since I've now got several drawing methods that involve drawing vectors, I thought I should factor
out this functionality into a separate utility method which I could call from all these places.

```
     # 754eba0318a682b068a8a5d5ca451decbaa204ce
     # src/lintrans/gui/plots/classes.py

142  class VectorGridPlot(BackgroundPlot):
...
388      def draw_position_vector(self, painter: QPainter, point: tuple[float, float], colour: QColor) -> None:
389          """Draw a vector from the origin to the given point.
390
391          :param QPainter painter: The ``QPainter`` object to use to draw the arrowheads with
392          :param point: The tip of the position vector in grid coords
393          :type point: tuple[float, float]
394          :param QColor colour: The colour to draw the position vector in
395          """
396          painter.setPen(QPen(colour, self.width_vector_line))
397          painter.drawLine(*self.canvas_origin, *self.canvas_coords(*point))
398          self.draw_arrowhead_away_from_origin(painter, point)
399
400      def draw_basis_vectors(self, painter: QPainter) -> None:
401          """Draw arrowheads at the tips of the basis vectors.
402
403          :param QPainter painter: The ``QPainter`` object to use to draw the arrowheads with
404          """
405          self.draw_position_vector(painter, self.point_i, self.colour_i)
406          self.draw_position_vector(painter, self.point_j, self.colour_j)
...
454      def draw_eigenvectors(self, painter: QPainter) -> None:
455          """Draw the eigenvectors of the displayed matrix transformation."""
456          values, vectors = np.linalg.eig(self.matrix)
457          vectors = vectors.T
458
459          for value, vector in zip(values, vectors):
460              x = value * vector[0]
461              y = value * vector[1]
462
463              if x.imag != 0 or y.imag != 0:
464                  continue
465
466              self.draw_position_vector(painter, (x, y), self.colour_eigen)
```

When I was testing this refactor, I realised that the draw_transformed_grid() method originally drew
the bodies of the basis vectors and the caller had to draw the arrowheads separately. This is silly and
was never planned that way; it was an unfortunate consequence of implementing the lines and arrow-
heads at different times. But now after this refactor, the caller has to call draw_basis_vectors() to
draw the whole basis vectors. So I had to add this call to VisualizeTransformationWidget.paintEvent()
and DefineVisuallyWidget.paintEvent() in src/lintrans/gui/plots/widgets.py and remove the calls
to draw_vector_arrowheads().

3.7.4 Adding eigenlines

Drawing some eigenvectors that point in a general direction is fine, but drawing the whole span of the vector would be much more useful. These spans are called eigenlines, and are just lines in the direction of the vector. To implement these, I knew I would have to get the eigenvalues and eigenvectors, zip them, and iterate over them like I did in §3.7.1. To make this simpler, I decided to factor out this zipping into a separate `self.eigs` property[16].

```python
# e1606f1e45ba93102dddb74b45ab22649a63fa53
# src/lintrans/gui/plots/classes.py

144    class VectorGridPlot(BackgroundPlot):
...
187        @property
188        def eigs(self) -> Iterable[tuple[float, NDArray[(1, 2), Float]]]:
189            """Return the eigenvalues and eigenvectors zipped together to be iterated over.
190
191            :rtype: Iterable[tuple[float, NDArray[(1, 2), Float]]]
192            """
193            values, vectors = np.linalg.eig(self.matrix)
194            return zip(values, vectors.T)
...
465        def draw_eigenvectors(self, painter: QPainter) -> None:
466            """Draw the eigenvectors of the displayed matrix transformation."""
467            for value, vector in self.eigs:
468                x = value * vector[0]
469                y = value * vector[1]
470
471                if x.imag != 0 or y.imag != 0:
472                    continue
473
474                self.draw_position_vector(painter, (x, y), self.colour_eigen)
```

I could then create a new method to find the gradient of the vector line and draw an orthogonal or oblique line respectively. Like before, we only want to render the eigenlines for real-valued eigenvectors, so we have to check their imaginary part.

```python
# 8d4d41fc4780cc037be39a0e574158e6cd34e997
# src/lintrans/gui/plots/classes.py

144    class VectorGridPlot(BackgroundPlot):
...
476        def draw_eigenlines(self, painter: QPainter) -> None:
477            """Draw the eigenlines (invariant lines).
478
479            :param QPainter painter: The painter to draw the lines with
480            """
481            painter.setPen(QPen(self.colour_eigen, self.width_transformed_grid))
482
483            for value, vector in self.eigs:
484                if value.imag != 0:
485                    continue
486
487                x, y = vector
488
489                if x == 0:
490                    x_mid = int(self.width() / 2)
491                    painter.drawLine(x_mid, 0, x_mid, self.height())
492
493                elif y == 0:
494                    y_mid = int(self.height() / 2)
495                    painter.drawLine(0, y_mid, self.width(), y_mid)
496
497                else:
498                    self.draw_oblique_line(painter, y / x, 0)
```

---

[16]A `@property` in Python is a value on a class which is dynamically evaluated only when it's needed. It functions just like a method with no arguments, but has more concise syntax for the caller.

I then just had to put these new eigenlines behind a display setting.

```
# 12cfabde606ebd3d48b2c3efaad0412f6100c3c5
# src/lintrans/gui/settings.py

9    class DisplaySettings:
...
44       draw_eigenlines: bool = False
45       """This controls whether we should draw the eigenlines of the transformation."""
```

```
# 12cfabde606ebd3d48b2c3efaad0412f6100c3c5
# src/lintrans/gui/dialogs/settings.py

63   class DisplaySettingsDialog(SettingsDialog):
...
66       def __init__(self, display_settings: DisplaySettings, *args, **kwargs):
...
128          self.checkbox_draw_eigenlines = QCheckBox(self)
129          self.checkbox_draw_eigenlines.setText('Draw eigen&lines')
130          self.checkbox_draw_eigenlines.setToolTip('Draw the eigenlines (invariant lines) of the transformations')
131          self.dict_checkboxes['l'] = self.checkbox_draw_eigenlines
```

```
# 12cfabde606ebd3d48b2c3efaad0412f6100c3c5
# src/lintrans/gui/plots/widgets.py

13   class VisualizeTransformationWidget(VectorGridPlot):
...
42       def paintEvent(self, event: QPaintEvent) -> None:
...
58          if self.display_settings.draw_eigenlines:
59              self.draw_eigenlines(painter)
```
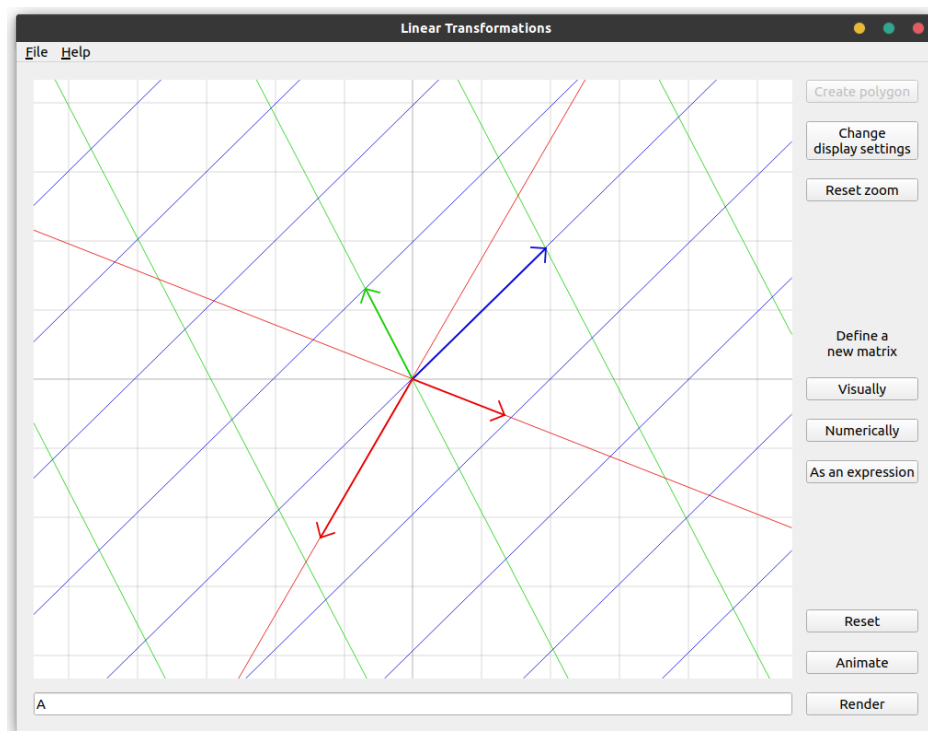

Figure 3.7.2: The eigenvectors being displayed for a similar matrix as in Figure 3.7.1

3.7.5 Adding eigenvalues as text

The next thing I wanted to do was tell the user what the actual eigenvalue numbers were. I decided the best way to do this would be to have the eigenvectors labelled with their associated eigenvalues.

To align the text properly, I decided to do something similar to what I did for determinants in §3.6.8. There, I enclosed the parallelogram in a rectangle and drew the text in the centre of it. Here, I decided to create a rectangle based on which quadrant the eigenvector was in. The rectangle would have corners of the tip of the eigenvector, and the appropriate corner of the viewport. I also had to choose a suitable alignment for each quadrant, so that the text would appear in the correct corner of the rectangle, next to the tip of the eigenvector.

```python
# a7e9a17ecdb2a585e9f766b13029116d7ba29bdc
# src/lintrans/gui/plots/classes.py

144    class VectorGridPlot(BackgroundPlot):
...
466        def draw_eigenvectors(self, painter: QPainter) -> None:
467            """Draw the eigenvectors of the displayed matrix transformation."""
468            for value, vector in self.eigs:
469                x = value * vector[0]
470                y = value * vector[1]
471
472                if x.imag != 0 or y.imag != 0:
473                    continue
474
475                self.draw_position_vector(painter, (x, y), self.colour_eigen)
476
477                # Now we need to draw the eigenvalue at the tip of the eigenvector
478
479                offset = 3
480                top_left: QPoint
481                bottom_right: QPoint
482                alignment_flags: int
483
484                if x >= 0 and y >= 0:  # Q1
485                    top_left = QPoint(self.canvas_x(x) + offset, 0)
486                    bottom_right = QPoint(self.width(), self.canvas_y(y) - offset)
487                    alignment_flags = Qt.AlignLeft | Qt.AlignBottom
488
489                elif x < 0 and y >= 0:  # Q2
490                    top_left = QPoint(0, 0)
491                    bottom_right = QPoint(self.canvas_x(x) - offset, self.canvas_y(y) - offset)
492                    alignment_flags = Qt.AlignRight | Qt.AlignBottom
493
494                elif x < 0 and y < 0:  # Q3
495                    top_left = QPoint(0, self.canvas_y(y) + offset)
496                    bottom_right = QPoint(self.canvas_x(x) - offset, self.height())
497                    alignment_flags = Qt.AlignRight | Qt.AlignTop
498
499                else:  # Q4
500                    top_left = QPoint(self.canvas_x(x) + offset, self.canvas_y(y) + offset)
501                    bottom_right = QPoint(self.width(), self.height())
502                    alignment_flags = Qt.AlignLeft | Qt.AlignTop
503
504                painter.setPen(QPen(self.colour_text, self.width_vector_line))
505                painter.drawText(QRectF(top_left, bottom_right), alignment_flags, f'{value:.2f}')
```

3.7.6 A tiny UI change
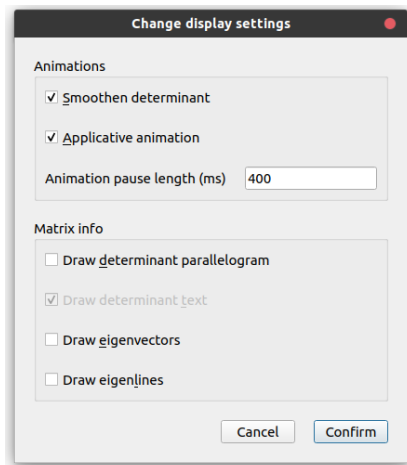
This bit isn't really related to eigenvectors, but it's a tiny change that doesn't really have a good place anywhere else, and it fits roughly here chronologically.

I really liked the groupboxes used in the display settings (left) and I'd quite like to enclose the matrix definition buttons in their own groupbox to separate them from the rest of the UI and better associate them with the label above them. This was a trivial addition.

Figure 3.7.3: The display settings

```
# 90425137edd4596219ab564ccbeccd65b5754008
# src/lintrans/gui/main_window.py

27    class LintransMainWindow(QMainWindow):
...
33        def __init__(self):
...
173           self.vlay_define_new_matrix = QVBoxLayout()
174           self.vlay_define_new_matrix.setSpacing(20)
175           self.vlay_define_new_matrix.addWidget(self.button_define_visually)
176           self.vlay_define_new_matrix.addWidget(self.button_define_numerically)
177           self.vlay_define_new_matrix.addWidget(self.button_define_as_expression)
178
179           self.groupbox_define_new_matrix = QtWidgets.QGroupBox('Define a new matrix', self)
180           self.groupbox_define_new_matrix.setLayout(self.vlay_define_new_matrix)
...
226           self.vlay_right.addWidget(self.groupbox_define_new_matrix)
```
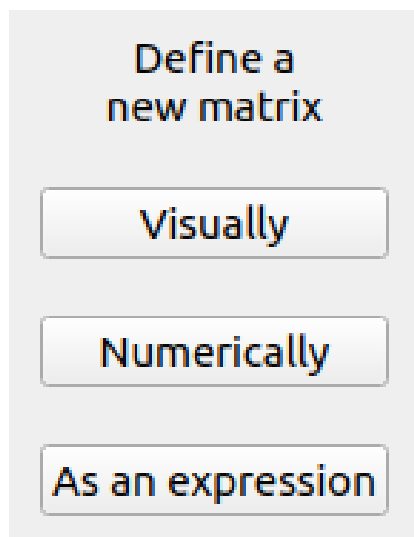

Figure 3.7.4: The old matrix definition buttons
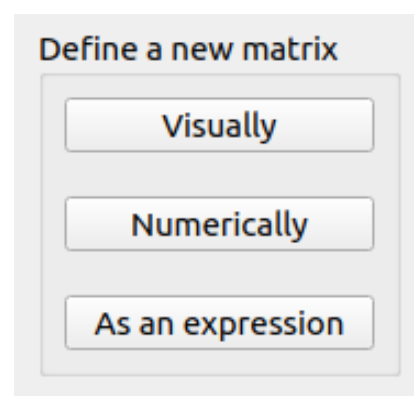
Figure 3.7.5: The new matrix definition buttons

3.8   Fumbling with SemVer

3.8.1   The first version numbers

At this point, I've been developing lintrans for quite a while, but I don't have any kind of version numbers yet. I wanted to fix this, to keep track of versions through history. SemVer[52] is a system of numbering versions of software that ensures compatibility across versions and gives semantic meaning to the version numbers.

My first foray into using SemVer was to declare `lintrans` at this point to be `0.1.0-alpha`. In retrospect, these early versions don't make much sense. I didn't release anything for anyone else to use until version `0.3.0`, so everything before then doesn't make much sense and should probably just be ignored.

I then made a few adjustments to syntax and documentation before declaring version `0.1.1-alpha` just over 24 hours later. This one was completely unnecessary in hindsight and I think I was just excited about being able to tag git commits. Additionally, there is a `__version__` variable in the root of the package, and the value of this variable is still `"0.1.0-alpha"` at the `v0.1.1-alpha` tag because I forgot to update it before I tagged the commit, and I didn't know how to fix it at the time.

3.8.2   Licensing

Using version numbers reminded me of licensing. I was already using the GNU GPLv3[19] for `lintrans` and most of my other projects, so I just wanted to add the copyright notice to all the source files of the project, as per the license. I already had the `COPYING` file in the project root with the full license, as required.

The notice looks like this[17]:

```
1    # lintrans - The linear transformation visualizer
2    # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4    # This program is licensed under GNU GPLv3, available here:
5    # <https://www.gnu.org/licenses/gpl-3.0.html>
```

3.8.3   Making the package executable

So far, I'd been running the program by running a separate script called `run_gui.py`, but Python lets you make a package executable by creating a file called `__main__.py` in the project root. You can then run it by using the command `python -m package_name`. This will make the whole program more cohesive by integrating the execution into the same directory as the source library code. I just moved the script here and changed its docstring a little bit.

```
     # 847cdf61f64b30e81edf0821b7a8d2eb060fb825
     # src/lintrans/__main__.py

9    """This module very simply runs the app by calling :func:`lintrans.gui.main_window.main`.
10
11   This allows the user to run the app like ``python -m lintrans`` from the command line.
12   """
13
14   import sys
15
16   from lintrans.gui import main_window
```

---

[17]I'm calling myself D. Dyson here because my name is Dyson, and I was in the process of getting it legally changed to Dyson Dyson when I made this change. I can't attribute this copyright to Dyson, since that's already a registered trademark of Dyson Limited. Calling myself Dyson Dyson felt strange, especially since it wasn't even my legal name at the time. So I chose to use D. Dyson instead.

```
17
18    if __name__ == '__main__':
19        main_window.main(sys.argv)
```

Somehow this caused a conflict between my `lintrans.typing` package, which contained custom types like `MatrixType`, and the standard library `typing` package, so I had to rename mine to `lintrans.typing_`.

3.8.4　Adding a graph of internal imports

Throughout the whole project, I've been using GitHub Actions[17] to automatically do things like test and lint my code, as well as compile the documentation whenever I push my changes to GitHub. I recently learned about `pylint`'s ability to generate a graph of all the internal imports in a package[36]. This graph shows how modules in the project depend on each other internally. It doesn't include external imports like `numpy`. I decided to generate this graph automatically and add it to my documentation.

```
      # 39a3727fca69ea65571a15c55741578abce1e763
      # .github/workflows/compile-docs.yaml

1     name: Compile docs for gh-pages
2
3     on:
4       push:
5         branches: [ main ]
6
7     jobs:
8       compile-docs:
9         runs-on: ubuntu-latest
10
11        concurrency:
12          group: ${{ github.workflow }}-${{ github.ref }}
13
14        steps:
15        - uses: actions/checkout@v2
16
17        - name: Set up Python 3.10
18          uses: actions/setup-python@v2
19          with:
20            python-version: '3.10'
21
22        - name: Install dependencies
23          run: |
24            pip install --upgrade pip
25            pip install -r requirements.txt -r docs/docs_requirements.txt
26            pip install -e .
27            pip install pylint
28            sudo apt-get install -y graphviz
29
30        - name: Create pylint import graphs
31          run: |
32            shopt -s globstar
33            pylint --rcfile=/dev/null --exit-zero --reports=y --disable=all --enable=imports,RP0402
              ↪  --int-import-graph=docs/source/int-imports.png src/lintrans/**/*.py
34
35        - name: Build docs
36          run: cd docs/ && make html && cd ..
37
38        - name: Deploy
39          uses: peaceiris/actions-gh-pages@v3.8.0
40          if: ${{ github.ref == 'refs/heads/main' }}
41          with:
42            github_token: ${{ secrets.GITHUB_TOKEN }}
43            publish_dir: ./docs/build/html/
44            keep_files: true
45            destination_dir: docs
46            user_name: 'github-actions[bot]'
```

```
47            user_email: 'github-actions[bot]@users.noreply.github.com'
48            commit_message: 'compile docs:'
```
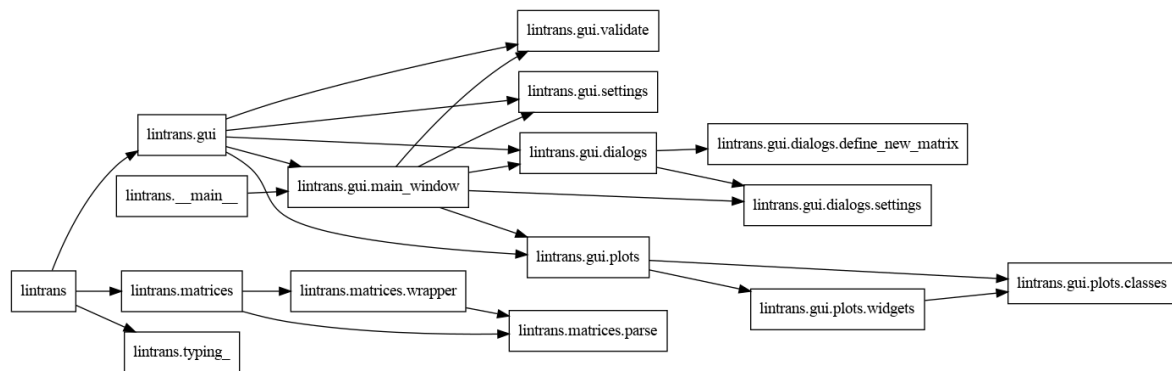

Figure 3.8.1: The internal imports graph

3.8.5   Fixing the colours

Now that I was using version numbers, I thought it was finally time to fix the colours in accordance with
§2.4. I decided to use red and blue for the basis vectors, since these are the most easily distinguished
colours for colour blind users. I decided to keep the green colour and use it for the eigenvectors and
eigenlines. These are very visually distinct from the basis vectors, so I don't think I need to further
distinguish them with colours. Red-green colour blind users should be able to tell them apart with
ease.
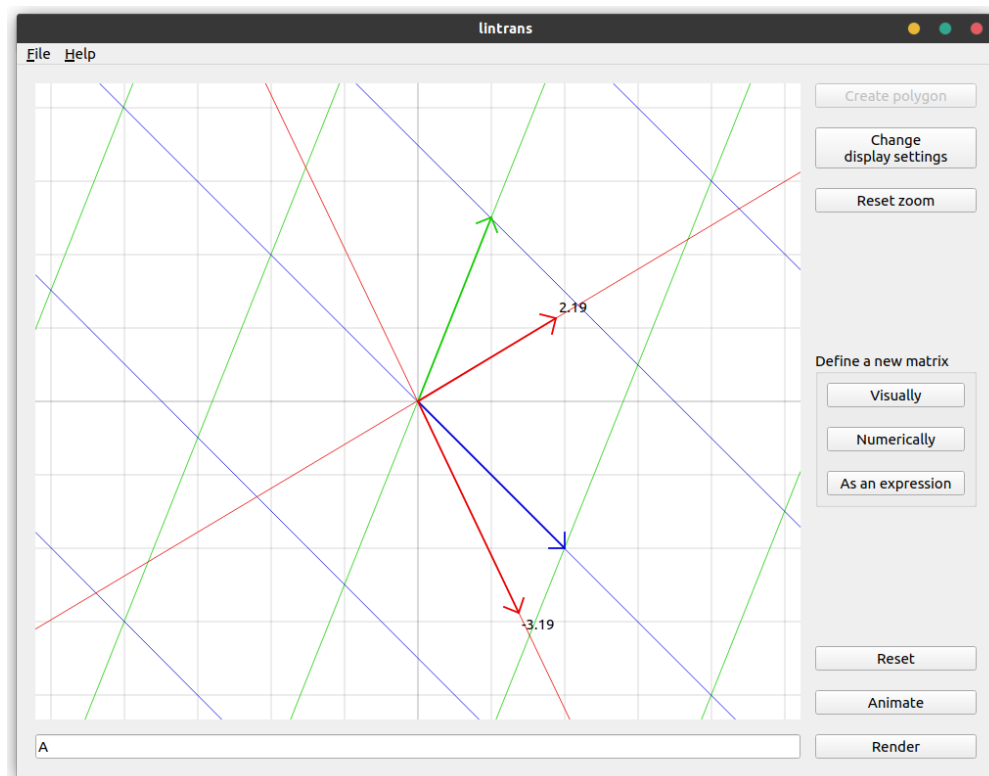
Figure 3.8.2: The old colours
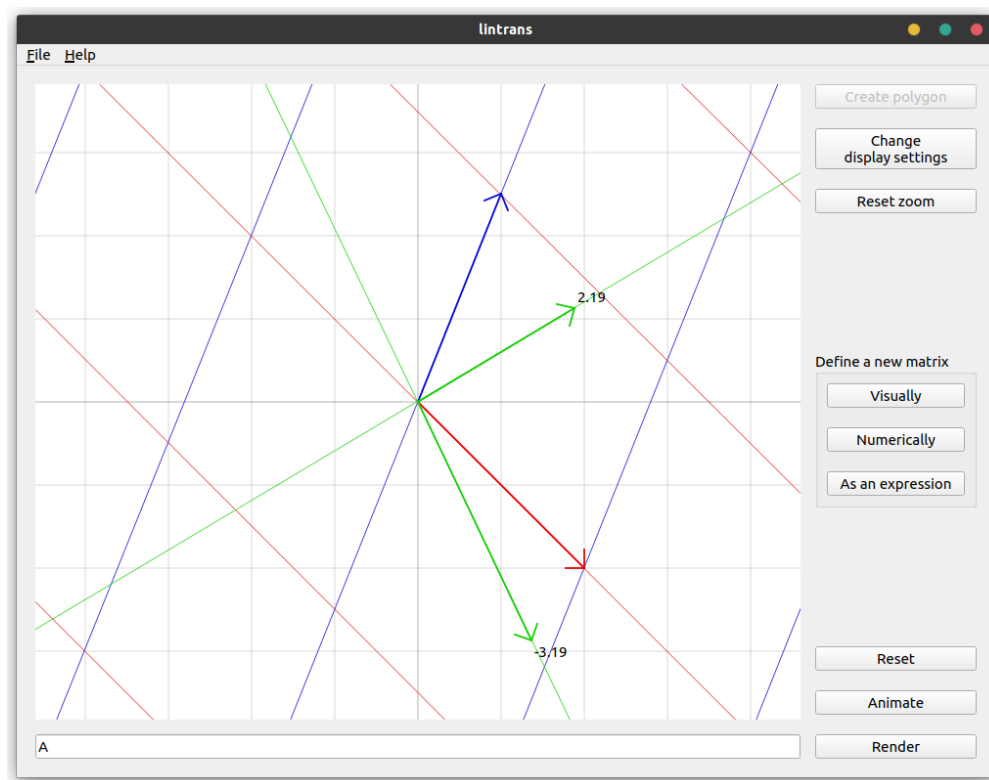
Figure 3.8.3: The new colours

3.8.6   Compiling for release

At this point, I felt ready to show `lintrans` to some teachers. Obviously I didn't expect them to have
a modern version of Python to run the program, especially not on a school laptop, so I had to compile
the program. I had a look at things like Nuitka[31], but I had a few issues with it and it took ages to
compile. Eventually I chose PyInstaller[34], which I've used to compile Python programs before. It's
fast and effective and works very well.

I use Ubuntu as my daily driver, so I don't really want to spin up a Windows virtual machine and
following some exact steps every time I want to compile `lintrans` for release, so I automated the
process with GitHub Actions.

```
# 9d0cbe69c596c04a07809a288d72b5e624f485d7
# .github/workflows/compile-release.yaml

1   name: Compile and release
2
3   on:
4     push:
5       tags:
6         - 'v*.*.*'
7
8   jobs:
9     check:
10      runs-on: ubuntu-latest
11      steps:
12        - uses: actions/checkout@v2
13
14        - name: Set up Python ${{ matrix.python-version }}
15          uses: actions/setup-python@v2
16          with:
17            python-version: '3.10'
18
```

```
19            - name: Install dependencies
20              run: |
21                pip install --upgrade pip
22                pip install -r requirements.txt -r dev_requirements.txt
23                pip install -e .
24
25            - name: Check
26              run: |
27                mypy src/ tests/
28                flake8 src/ tests/
29                pycodestyle src/ tests/
30                pydocstyle src/ tests/
31
32      test:
33        needs: check
34        runs-on: ${{ matrix.os }}
35
36        strategy:
37          matrix:
38            os: [ubuntu-latest, macos-latest, windows-latest]
39
40        steps:
41          - uses: actions/checkout@v2
42
43          - name: Set up Python 3.10
44            uses: actions/setup-python@v2
45            with:
46              python-version: '3.10'
47
48          - name: Install dependencies
49            run: |
50              pip install --upgrade pip
51              pip install -r requirements.txt -r dev_requirements.txt
52              pip install -e .
53
54          - name: Test
55            run: |
56              pytest
57              pytest --doctest-modules src/
58
59      compile:
60        needs: test
61        runs-on: ${{ matrix.os }}
62
63        strategy:
64          matrix:
65            os: [ubuntu-latest, macos-latest, windows-latest]
66
67        steps:
68          - uses: actions/checkout@v2
69
70          - name: Set up Python 3.10
71            uses: actions/setup-python@v2
72            with:
73              python-version: '3.10'
74
75          - name: Install dependencies
76            run: |
77              pip install --upgrade pip
78              pip install -r requirements.txt -r dev_requirements.txt
79              pip install -e .
80              pip install pyinstaller
81
82          - name: Compile
83            run: pyinstaller --onefile --windowed --distpath=./dist --name lintrans-${{ runner.os }}
                 ↪  src/lintrans/__main__.py
84
85          - name: Upload artifact
86            uses: actions/upload-artifact@v3
87            with:
88              name: ${{ matrix.os }}-binary
89              path: dist/lintrans*
90
```

```yaml
 91      publish:
 92        needs: compile
 93        runs-on: ubuntu-latest
 94
 95        steps:
 96          - uses: actions/checkout@v2
 97
 98          - name: Download Windows binary
 99            uses: actions/download-artifact@v3
100            with:
101              name: windows-latest-binary
102              path: dist/
103
104          - name: Download macOS binary
105            uses: actions/download-artifact@v3
106            with:
107              name: macos-latest-binary
108              path: dist/
109
110          - name: Download Linux binary
111            uses: actions/download-artifact@v3
112            with:
113              name: ubuntu-latest-binary
114              path: dist/
115
116          - name: Check for alpha
117            id: checkalpha
118            run: |
119              isalpha=$(if [ -n "$(echo $GITHUB_REF | grep -o -- 'alpha')" ]; then echo 1; else echo 0; fi)
120              echo "::set-output name=isalpha::$isalpha"
121
122          - name: Upload binaries (normal release)
123            if: steps.checkalpha.outputs.isalpha == 0
124            uses: softprops/action-gh-release@v1
125            with:
126              fail_on_unmatched_files: true
127              prerelease: false
128              draft: true  # I'll manually add the title and description when the draft is online
129              files: |
130                dist/lintrans-Windows.exe
131                dist/lintrans-macOS
132                dist/lintrans-Linux
133
134          - name: Upload binaries (pre-release)
135            if: steps.checkalpha.outputs.isalpha == 1
136            uses: softprops/action-gh-release@v1
137            with:
138              fail_on_unmatched_files: true
139              prerelease: true
140              draft: true  # I'll manually add the title and description when the draft is online
141              files: |
142                dist/lintrans-Windows.exe
143                dist/lintrans-macOS
144                dist/lintrans-Linux
```

This workflow only runs when I push a tag containing a version number. It firstly installs Python[18] and then lints and tests everything. If that all passes, then it compiles lintrans on Ubuntu, macOS, and Windows. Then it publishes the compiled binaries to a GitHub draft release. I then go in when it's finished to add a description to the release and publish it.

Now that I had this in place, I released version v0.2.0-alpha.

After some minor adjustments, such as including the tag name in the binary name, and installing UPX[60] for the Linux compilation[19], I released v0.2.0 just 6 hours later.

---

[18]There's a mistake on line 14. There is no strategy matrix, so '${{ matrix.python-version }}' is treated as plain text rather than an actual version number. This was caused by copying a previous workflow. The actual version that it installs is fine, though. It's only the name of the step that doesn't look right.

[19]UPX is designed to compress executable files, but I later learned that PyInstaller doesn't even use UPX on Linux; it only uses it on Windows.

I made a few minor adjustments to the workflows, but soon moved on from this version number fiasco and started doing more actual work.

3.9   Preparing for `v0.2.1`

3.9.1   Fixing slots and signals

I was perusing the Qt5 documentation when I learned about the difference between the `dialog.finished` signal and the `dialog.accepted` signal. I decided to rework some old code to make better use these signals.

When defining a new matrix or dialog settings, we only want to save the new data if the user actually accepted the dialog by clicking the confirm button. We don't want to save it if they clicked cancel. However, in the case of the error message dialog, we always want to update the render buttons when it's closed, no matter how the user closed the dialog.

```python
# 66242465222a153a5f37c4a1a3c2bd50bfd90933
# src/lintrans/gui/main_window.py

35    class LintransMainWindow(QMainWindow):
...
447       @pyqtSlot(DefineDialog)
448       def dialog_define_matrix(self, dialog_class: Type[DefineDialog]) -> None:
...
461           # We create a dialog with a deepcopy of the current matrix_wrapper
462           # This avoids the dialog mutating this one
463           dialog = dialog_class(deepcopy(self.matrix_wrapper), self)
464
465           # .open() is asynchronous and doesn't spawn a new event loop, but the dialog is still modal (blocking)
466           dialog.open()
467
468           # So we have to use the accepted signal to call a method when the user accepts the dialog
469           dialog.accepted.connect(self.assign_matrix_wrapper)
...
478       @pyqtSlot()
479       def dialog_change_display_settings(self) -> None:
480           """Open the dialog to change the display settings."""
481           dialog = DisplaySettingsDialog(self.plot.display_settings, self)
482           dialog.open()
483           dialog.accepted.connect(lambda: self.assign_display_settings(dialog.display_settings))
...
493       def show_error_message(self, title: str, text: str, info: str | None = None) -> None:
...
511           # This is `finished` rather than `accepted` because we want to update the buttons no matter what
512           dialog.finished.connect(self.update_render_buttons)
```

I also added the `@pyqtSlot()` decorator to all the relevant methods in the matrix definition dialogs. The types in the brackets indicate the signature of the method.

A slot in Qt5 is just a method that is expected to be connected to a signal, so it gets called from the event loop. Using the decorator makes it clear that a method is a slot, and also allows slightly better performance.

```python
# 9beff9cf25d3af655e134205572a5668279f42cc
# src/lintrans/gui/dialogs/define_new_matrix.py

67    class DefineDialog(QDialog):
...
138       @abc.abstractmethod
139       @pyqtSlot()
140       def update_confirm_button(self) -> None:
...
143       @pyqtSlot(int)
144       def load_matrix(self, index: int) -> None:
...
155       @abc.abstractmethod
156       @pyqtSlot()
157       def confirm_matrix(self) -> None:
```

```
...
164        class DefineVisuallyDialog(DefineDialog):
...
194            @pyqtSlot()
195            def update_confirm_button(self) -> None:
...
203            @pyqtSlot(int)
204            def load_matrix(self, index: int) -> None:
...
214            @pyqtSlot()
215            def confirm_matrix(self) -> None:
...
226        class DefineNumericallyDialog(DefineDialog):
...
276            @pyqtSlot()
277            def update_confirm_button(self) -> None:
...
288            @pyqtSlot(int)
289            def load_matrix(self, index: int) -> None:
...
305            @pyqtSlot()
306            def confirm_matrix(self) -> None:
...
317        class DefineAsAnExpressionDialog(DefineDialog):
...
348            @pyqtSlot()
349            def update_confirm_button(self) -> None:
...
356            @pyqtSlot(int)
357            def load_matrix(self, index: int) -> None:
...
364            @pyqtSlot()
365            def confirm_matrix(self) -> None:
```

3.9.2   Linking in documentation

I've been using Sphinx[55] for my documentation this whole time, and I've been using the Sphinx extension intersphinx to link to the Python standard library documentation. It uses a system of binary inventory files which define a reference map between names to use in the documentation, and where to link those names to. I recently learned of the sphobjinv Python package, which allows you to easily create your own local inventory files to reference some external source of documentation, such as the Qt5 documentation. I read through the sphobjinv documentation[54] and designed a small script to read a custom text file, and create the binary inventory file needed by Sphinx.

```
    # 5455265a51666e29ab976152c1a758a422e1004a
    # docs/create_objects_inv.py

9   """A simple script to convert my manually curated text file to an invetory file that intersphinx can use.
10
11  .. note:: The URIs in the text file must not have .html suffices
12  """
13
14  import re
15  from glob import glob
16
17  import sphobjinv as soi
18
19
20  pattern = re.compile(r'^(\S+)\s+([^:\s]+):([^:\s]+)\s+(\d+)\s+(\S+)\s+(\S+)$')
21
22
23  def generate_objects_inv(prefix: str) -> None:
24      """Generate the ``objects.inv`` file for PyQt5.
25
26      We read from ``prefix-objects.txt`` and write to ``prefix-objects.inv``,
27      so if you want to use ``pyqt5-objects.txt``, then the prefix should be ``pyqt5``.
28
29      :param str prefix: The prefix for the object files
```

```
30              """
31              inv = soi.Inventory()
32              inv.project = 'PyQt5'
33              inv.version = '5.15'
34
35              with open(prefix + '-objects.txt', 'r', encoding='utf-8') as f:
36                  text = f.read().splitlines()
37
38              for line in text:
39                  if line == '' or line.lstrip().startswith('#'):
40                      continue
41
42                  if (match := re.match(pattern, line)) is None:
43                      raise ValueError(f'Every line in {prefix}-objects.txt must match the pattern')
44
45                  name, domain, role, priority, uri, disp_name = match.groups()
46
47                  inv.objects.append(soi.DataObjStr(
48                      name=name, domain=domain, role=role, priority=priority, uri=uri, dispname=disp_name
49                  ))
50
51              compressed_text = soi.compress(inv.data_file(contract=True))
52              soi.writebytes(f'source/{prefix}-objects.inv', compressed_text)
53
54
55      def main() -> None:
56          """Call :func:`generate_objects_inv` for every file matching the glob pattern '*-objects.txt'."""
57          for filename in glob('*-objects.txt'):
58              prefix = filename[:-12]
59              print(f'Generating {prefix}-objects.inv')
60              generate_objects_inv(prefix)
61
62
63      if __name__ == '__main__':
64          main()
```

Line 11 should say 'must have .html suffices'.

```
        # aab8e88b0e2cdae8038c9935031c74bcdae0ad5c
        # docs/pyqt5-objects.txt

1       # This format is:
2       # <reference name> <domain>:<role> <priority> <URI> <display name>
3       # Sphinx handles the prefix for the URI
4       # If the display name is '-', then it's the same as the reference name
5
6       # === Classes
7
8       QApplication py:class 1 qapplication.html -
9       QDialog      py:class 1 qdialog.html      -
10      QKeyEvent    py:class 1 qkeyevent.html    -
11      QPainter     py:class 1 qpainter.html     -
12      QPaintEvent  py:class 1 qpaintevent.html  -
13      QWheelEvent  py:class 1 qwheelevent.html  -
14      QWidget      py:class 1 qwidget.html      -
15
16      # === Methods
17
18      QPainter.drawLine:iiii py:method 1 qpainter.html#drawLine-2 QPainter.drawLine()
19
20      # === Signals
21
22      QComboBox.activated py:method 1 qcombobox.html#activated -
23      QDialog.accepted    py:method 1 qdialog.html#accepted    -
24
25      # These are in full form so that autodoc can reference base classes and param types
26
27      PyQt5.QtGui.QKeyEvent   py:class 1 qkeyevent.html   -
28      PyQt5.QtGui.QPainter    py:class 1 qpainter.html    -
29      PyQt5.QtGui.QPaintEvent py:class 1 qpaintevent.html -
30      PyQt5.QtGui.QWheelEvent py:class 1 qwheelevent.html -
```

```
31    PyQt5.QtWidgets.QDialog py:class 1 qdialog.html    -
32    PyQt5.QtWidgets.QWidget py:class 1 qwidget.html    -
```

I then just had to change all the references to Qt5 things in the documentation and then Sphinx would automatically link all the Qt5 references to their appropriate links, as defined in this file.

3.9.3 Improving tests

I made some small improvements to the unit tests by making sure they handled greedy index parsing, which means that something like `A^2 3B` will get parsed as `A^{23}B` because whitespace is ignored, as well asserting that all invalid expressions raise `MatrixParseError`. I also added the copyright comment to the test files.

This was a tiny change, but worth noting.

```
      # c07d97024e1fe00ab110f43e5c7e6737c955d680
      # tests/matrices/test_parse_and_validate_expression.py

41    expressions_and_parsed_expressions: list[tuple[str, MatrixParseList]] = [
...
50        ('A^12', [[('', 'A', '12')]]),
51        ('A^234', [[('', 'A', '234')]]),
...
55        ('A^2 3B', [[('', 'A', '23'), ('', 'B', '')]]),
...
72        ('2.14A^{3} 4.5rot(14.5)^-1 + 8.5B^T 5.97C^14 - 3.14D^{-1} 6.7E^T',
73         [[('2.14', 'A', '3'), ('4.5', 'rot(14.5)', '-1')], [('8.5', 'B', 'T'), ('5.97', 'C', '14')],
...
75    ]
...
78    def test_parse_matrix_expression() -> None:
79        """Test the parse_matrix_expression() function."""
80        for expression, parsed_expression in expressions_and_parsed_expressions:
81            # Test it with and without whitespace
82            assert parse_matrix_expression(expression) == parsed_expression
83            assert parse_matrix_expression(expression.replace(' ', '')) == parsed_expression


      # 70e1a7271a61f3009cc4d342f46743b248498a1c
      # tests/matrices/test_parse_and_validate_expression.py

26    invalid_inputs: list[str] = [
27        '', 'rot()', 'A^', 'A^1.2', 'A^{3.4}', '1,2A', 'ro(12)', '5', '12^2', '^T', '^{12}',
28        'A^{13', 'A^3}', 'A^A', '^2', 'A--B', '--A', '+A', '--1A', 'A--B', 'A--1B', '.A', '1.A'
29
30        'This is 100% a valid matrix expression, I swear'
31    ]
...
86    def test_parse_error() -> None:
87        """Test that parse_matrix_expression() raises a MatrixParseError."""
88        for expression in invalid_inputs:
89            with pytest.raises(MatrixParseError):
90                parse_matrix_expression(expression)
```

```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *]
$ pytest -v
=================================== test session starts ===================================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml, testpaths: tests
collected 30 items

tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs0-True] PASSED              [  3%]
tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs1-False] PASSED             [  6%]
tests/gui/test_dialog_utility_functions.py::test_round_float PASSED                               [ 10%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs0-True] PASSED   [ 13%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs1-False] PASSED  [ 16%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_matrix_expression PASSED         [ 20%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_error PASSED                     [ 23%]
tests/matrices/test_rotation_matrices.py::test_create_rotation_matrix PASSED                      [ 26%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_matrix_addition PASSED     [ 30%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_two_matrix_multiplication PASSED  [ 33%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_identity_multiplication PASSED    [ 36%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_three_matrix_multiplication PASSED  [ 40%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_inverses PASSED            [ 43%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_powers PASSED              [ 46%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_transpose PASSED           [ 50%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_rotation_matrices PASSED          [ 53%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_multiplication_and_addition PASSED [ 56%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_complicated_expressions PASSED    [ 60%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_value_errors PASSED               [ 63%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_linalgerror PASSED                [ 66%]
tests/matrices/matrix_wrapper/test_misc.py::test_get_expression PASSED                            [ 70%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_get_matrix PASSED           [ 73%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_get_name_error PASSED             [ 76%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_set_matrix PASSED           [ 80%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_expression PASSED             [ 83%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_simple_dynamic_evaluation PASSED  [ 86%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_recursive_dynamic_evaluation PASSED [ 90%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_identity_error PASSED         [ 93%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_name_error PASSED             [ 96%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_type_error PASSED             [100%]

=================================== 30 passed in 0.36s ===================================
```

Figure 3.9.1: Running `pytest` with the new tests

3.9.4   The Windows version file

Windows stores metadata for `.exe` files inside the `.exe` files themselves[61][62]. PyInstaller lets you embed that metadata when compiling on Windows, using a version file[35]. Obviously, I wanted to include this metadata in my compiled `.exe` in the release.

I created a simple `precompile.py` script that would create different pre-compilation artefacts. I could then use these in the compilation workflow for GitHub Actions. I started with just Windows, but obviously I'm going to expand into Linux and macOS as well later.

```
    # 2126959cb6f836b1bc6c92dad859b43cbd86e1ab
    # precompile.py

 9  """A simple pre-compile script for the automated GitHub page compilation action."""
10
11  import re
12  import sys
13
14
15  def precompile_macos() -> None:
16      """Run the pre-compile steps for macOS."""
17      print('Pre-compile for macOS not implemented yet')
18
19
20  def precompile_windows(args: list[str]) -> None:
21      """Run the pre-compile steps for Windows."""
22      print('Pre-compiling for Windows')
23
24      if len(args) < 1:
25          raise ValueError('Windows pre-compile needs tag name argument')
26
27      tag_name = args[0]
28
29      if (m := re.match(r'v(\d+)\.(\d+)\.(\d+)(-alpha)?', tag_name)) is not None:
30          major, minor, patch, alpha = m.groups()
31
```

```python
32          else:
33              raise ValueError('Tag name must match format')
34
35          if alpha is not None:
36              flags = '0x2'
37          else:
38              flags = '0x0'
39
40          version_tuple = f'{major}, {minor}, {patch}, 0'
41
42          version_info = f'''VSVersionInfo(
43      ffi=FixedFileInfo(
44        filevers=({version_tuple}),
45        prodvers=({version_tuple}),
46        mask=0x3f,
47        flags={flags},
48        OS=0x40004,
49        fileType=0x1,
50        subtype=0x0,
51        date=(0, 0)
52      ),
53      kids=[
54        StringFileInfo(
55          [
56            StringTable(
57              '040904B0',
58              kids=[
59                StringStruct('CompanyName', 'D. Dyson (DoctorDalek1963)'),
60                StringStruct('FileDescription', 'Linear transformation visualizer'),
61                StringStruct('FileVersion', '{tag_name}'),
62                StringStruct('InternalName', 'lintrans'),
63                StringStruct('LegalCopyright', '(C) D. Dyson (DoctorDalek1963) under GPLv3'),
64                StringStruct('OriginalFilename', 'lintrans-Windows-{tag_name}.exe'),
65                StringStruct('ProductName', 'lintrans'),
66                StringStruct('ProductVersion', '{tag_name}')
67              ]
68            )
69          ]
70        ),
71        VarFileInfo([VarStruct('Translation', [2057, 1200])])
72      ]
73    )
74    '''
75
76          with open('version_info.txt', 'w', encoding='utf-8') as f:
77              f.write(version_info)
78
79          print('Version file written to version_info.txt')
80
81
82      def main(args: list[str]) -> None:
83          """Evaluate the arguments and pre-compile accordingly."""
84          if len(args) < 1:
85              raise ValueError('Script must be supplied with the name of an operating system.')
86
87          os_name = args[0].lower()
88
89          if os_name == 'linux':
90              print("Linux doesn't need any pre-compilation")
91
92          elif os_name == 'macos':
93              precompile_macos()
94
95          elif os_name == 'windows':
96              precompile_windows(args[1:])
97
98          else:
99              raise ValueError(f'Unsupported operating system "{os_name}"')
100
101
102     if __name__ == '__main__':
103         main(sys.argv[1:])
```

```yaml
     # 2126959cb6f836b1bc6c92dad859b43cbd86e1ab
     # .github/workflows/compile-release.yaml

8    jobs:
...
60     compile:
...
68       steps:
...
87         - name: Pre-compile
88           run: python precompile.py ${{ runner.os }} $GITHUB_REF_NAME
89           shell: bash
90
91         - name: Compile (Linux)
92           if: runner.os == 'Linux'
93           run: pyinstaller --onefile --windowed --distpath=./dist --name lintrans-${{ runner.os }}-$GITHUB_REF_NAME
             ↪ src/lintrans/__main__.py
94           shell: bash
95
96         - name: Compile (macOS)
97           if: runner.os == 'macOS'
98           run: pyinstaller --onefile --windowed --distpath=./dist --name lintrans-${{ runner.os }}-$GITHUB_REF_NAME
             ↪ src/lintrans/__main__.py
99           shell: bash
100
101        - name: Compile (Windows)
102          if: runner.os == 'Windows'
103          run: pyinstaller --onefile --windowed --distpath=./dist --version-file version_info.txt --name
             ↪ lintrans-${{ runner.os }}-$GITHUB_REF_NAME src/lintrans/__main__.py
104          shell: bash
```

I quickly realised that this design would make the compilation process more complicated, since the process would be in separate parts. Instead, I decided to create a unified compilation script, which runs a pre-compile step dependent on the operating system, and then compiles the program with the PyInstaller.__main__.run() function.

```python
     # ca674c7f7d61e8eed3410d456787cfe5b2bc28e5
     # compile.py

9    """A simple pre-compile script for the automated GitHub page compilation action."""
10
11   import argparse
12   import os
13   import re
14   import shutil
15   import sys
16   from textwrap import dedent
17
18   from PyInstaller.__main__ import run as run_pyi
19
20   import lintrans
21
22
23   class Compiler:
24       """A simple class to encapsulate compilation logic."""
25
26       def __init__(
27               self, *,
28               platform: str | None = None,
29               version_name: str | None = None,
30               filename: str | None = None
31       ):
32           """Create a Compiler object."""
33           self.platform = platform if platform else sys.platform
34           self.version_name = version_name if version_name else lintrans.__version__
35           self.filename = filename if filename else 'lintrans'
36
37       def _precompile_windows(self) -> None:
38           """Pre-compile for Windows."""
39           if (m := re.match(r'v?(\d+)\.(\d+)\.(\d+)(-alpha)?', self.version_name)) is not None:
```

```
40                    major, minor, patch, alpha = m.groups()
41
42                else:
43                    raise ValueError('Tag name must match format')
44
45                if alpha is not None:
46                    flags = '0x2'
47                else:
48                    flags = '0x0'
49
50                version_tuple = f'{major}, {minor}, {patch}, 0'
51
52                version_info = dedent(f'''
53                VSVersionInfo(
54                  ffi=FixedFileInfo(
55                    filevers=({version_tuple}),
56                    prodvers=({version_tuple}),
57                    mask=0x3f,
58                    flags={flags},
59                    OS=0x40004,
60                    fileType=0x1,
61                    subtype=0x0,
62                    date=(0, 0)
63                  ),
64                  kids=[
65                    StringFileInfo(
66                      [
67                        StringTable(
68                          '040904B0',
69                          kids=[
70                            StringStruct('CompanyName', 'D. Dyson (DoctorDalek1963)'),
71                            StringStruct('FileDescription', 'Linear transformation visualizer'),
72                            StringStruct('FileVersion', '{self.version_name}'),
73                            StringStruct('InternalName', 'lintrans'),
74                            StringStruct('LegalCopyright', '(C) D. Dyson (DoctorDalek1963) under GPLv3'),
75                            StringStruct('OriginalFilename', '{self.filename}.exe'),
76                            StringStruct('ProductName', 'lintrans'),
77                            StringStruct('ProductVersion', '{self.version_name}')
78                          ]
79                        )
80                      ]
81                    ),
82                    VarFileInfo([VarStruct('Translation', [2057, 1200])])
83                  ]
84                )
85                ''')
86
87                with open('version_info.txt', 'w', encoding='utf-8') as f:
88                    f.write(version_info)
89
90                print('Version file written to version_info.txt')
91
92        def precompile(self) -> None:
93            """Pre-compile for the appropriate operating system."""
94            if self.platform == 'linux':
95                print("Linux doesn't need any pre-compilation")
96
97            elif self.platform == 'darwin':
98                print("macOS doesn't need any pre-compilation")
99
100           elif self.platform == 'win32':
101               print('Pre-compiling for Windows')
102               self._precompile_windows()
103
104           else:
105               raise ValueError(f'Unsupported operating system "{self.platform}"')
106
107       def _get_pyi_args(self) -> list[str]:
108           """Return the common args for PyInstaller."""
109           return [
110               'src/lintrans/__main__.py',
111               '--onefile',
112               '--windowed',
```

```
113                    '--distpath=./dist',
114                    '--workpath=./build',
115                    '--noconfirm',
116                    '--clean',
117                    f'--name={self.filename}'
118                ]
119
120        def _compile_macos(self) -> None:
121            """Compile for macOS."""
122            run_pyi(self._get_pyi_args())
123
124            os.rename(os.path.join('dist', self.filename + '.app'), self.filename + '.app')
125
126        def _compile_linux(self) -> None:
127            """Compile for Linux."""
128            run_pyi(self._get_pyi_args())
129
130            os.rename(os.path.join('dist', self.filename), self.filename)
131
132        def _compile_windows(self) -> None:
133            """Compile for Windows."""
134            if not os.path.isfile('version_info.txt'):
135                raise ValueError('Windows compilation requires version_info.txt from pre-compilation')
136
137            run_pyi([
138                *self._get_pyi_args(),
139                '--version-file',
140                'version_info.txt'
141            ])
142
143            os.remove('version_info.txt')
144
145            os.rename(os.path.join('dist', self.filename + '.exe'), self.filename + '.exe')
146
147        def compile(self) -> None:
148            """Compile for the appropriate operating system."""
149            if self.platform == 'darwin':
150                self._compile_macos()
151
152            elif self.platform == 'linux':
153                self._compile_linux()
154
155            elif self.platform == 'win32':
156                self._compile_windows()
157
158            else:
159                raise ValueError(f'Unsupported operating system "{self.platform}"')
160
161            shutil.rmtree('dist')
162            shutil.rmtree('build')
163            os.remove(self.filename + '.spec')
164
165
166    def main() -> None:
167        """Run any pre-compilation, and then compile."""
168        parser = argparse.ArgumentParser(description='Compile this version of lintrans for your operating system',
        ↪    add_help=True)
169        parser.add_argument('-f', '--filename', type=str, required=False, default=None, help='the filename (without
        ↪    extension)')
170        parser.add_argument('-v', '--version', type=str, required=False, default=None, help='the version name in the
        ↪    format v1.2.3')
171
172        args = parser.parse_args()
173
174        compiler = Compiler(filename=args.filename, version_name=args.version)
175        compiler.precompile()
176        compiler.compile()
177
178
179    if __name__ == '__main__':
180        main()
```

This new compilation script captures the whole process. On Linux or macOS, it just compiles the program with PyInstaller. On Windows, it has to generate the version file, then compile the program with an additional argument to include the version file. I then updated the GitHub Actions workflow to use this new compilation script.

```yaml
# e47fe732954bf018128bdcb5ee9c354910517f36
# .github/workflows/compile-release.yaml

8    jobs:
...
60     compile:
...
68       steps:
...
87         - name: Compile
88           run: python compile.py -f lintrans-${{ runner.os }}-${{ env.GITHUB_REF_NAME }} -v $GITHUB_REF_NAME
```

3.9.5   Compiling for macOS

Compiling for macOS is considerably more difficult. I run Ubuntu as my primary operating system. I used to use Windows and I have a Windows 10 virtual machine, so compiling for Windows was never an issue. But I've never used a Mac, and Apple don't like people installing macOS on non-Apple hardware. Getting a virtual machine set up running macOS Monterey was quite difficult, but I eventually managed to get it working somewhat properly.

Once I had `lintrans` compiling on macOS, I wanted to do something similar to what I did with Windows, where I added extra metadata to the executable. In macOS, executables are bundled as `.app` files, which are just directory structures, containing the necessary files and metadata[13]. All the metadata for an application bundle is contained in the `Info.plist` file in the bundle[10][9]. After reading the documentation for these files, I created one for `lintrans`.

```python
# e716000521c92259ed4a1b33ab37e3860a7b7875
# compile.py

23   class Compiler:
...
92       def _macos_replace_info_plist(self) -> None:
93           """Replace the Info.plist file in the macOS app."""
94           short_version_name = self.version_name
95
96           if short_version_name.startswith('v'):
97               short_version_name = short_version_name[1:]
98
99           if short_version_name.endswith('-alpha'):
100              short_version_name = short_version_name[:-6]
101
102          new_info_plist = dedent(f'''
103          <?xml version="1.0" encoding="UTF-8"?>
104          <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd"
105          <plist version="1.0">
106          <dict>
107              <key>CFBundleDisplayName</key>
108              <string>lintrans</string>
109              <key>CFBundleExecutable</key>
110              <string>lintrans</string>
111              <key>CFBundleIconFile</key>
112              <string>icon-windowed.icns</string>
113              <key>CFBundleIdentifier</key>
114              <string>lintrans</string>
115              <key>CFBundleInfoDictionaryVersion</key>
116              <string>6.0</string>
117              <key>CFBundleName</key>
118              <string>lintrans</string>
119              <key>CFBundleType</key>
120              <string>APPL</string>
```

```
121              <key>CFBundleVersion</key>
122              <string>{self.version_name}</string>
123              <key>CFBundleShortVersionString</key>
124              <string>{short_version_name}</string>
125              <key>NSHighResolutionCapable</key>
126              <true/>
127              <key>NSHumanReadableCopyright</key>
128              <string>(C) D. Dyson (DoctorDalek1963) under GPLv3</string>
129          </dict>
130          </plist>
131          '''[1:])
132
133          with open(os.path.join(self.filename + '.app','Contents', 'Info.plist'), 'w', encoding='utf-8') as f:
134              f.write(new_info_plist)
...
149      def _compile_macos(self) -> None:
150          """Compile for macOS."""
151          run_pyi(self._get_pyi_args())
152
153          os.rename(os.path.join('dist', self.filename + '.app'), self.filename + '.app')
154
155          self._macos_replace_info_plist()
```

This could would automatically generate the `Info.plist` file and write it to the correct place. However, I couldn't distribute this bundled app, since Apple requires an app to be signed by a trusted author before other people can download and run that code on their own Apple devices[8]. And unfortunately, Apple charges \$99 per year for membership to their 'Apple Developer Program', which is required to become a trusted developer[11][20].

Microsoft also wants its developers to sign their code for security and trust, and also charges for this privilege. But on Windows, you can run a foreign, unsigned app so long as you dismiss the potential virus warning. On macOS, it's completely forbidden.

I don't particularly want to pay \$99 per year to compile `lintrans` for macOS, when my audience on that platform is realistically almost zero, so I just removed macOS from the GitHub Actions publish workflow. For anyone that wants to run `lintrans` on macOS, I will provide instructions to compile it from source, since I can't distribute a binary file that I built myself. I will however continue to run the unit tests and compile the program on macOS in GitHub Actions, to ensure that everything works fine on that platform.

3.9.6 Supporting flags

Most Linux apps support command line arguments and/or flags[21], and so I wanted to support these with `lintrans`. I wanted one flag to display help for the program, and one flag to display the version number of the program. Implementing this was very simple and I didn't bother using a library like `argparse`; I just used `sys.argv`.

```
    # ffc603f1bf049811cb927f879ce7989456f6a537
    # src/lintrans/__main__.py

9   """This module provides a :func:`main` function to interpret command line arguments and run the program."""
10
11  import sys
12  from textwrap import dedent
13
14  from lintrans import __version__
15  from lintrans.gui import main_window
16
17
18  def main(prog_name: str, args: list[str]) -> None:
19      """Interpret program-specific command line arguments and run the main window in most cases.
```

---

[20]The price is listed in the small print at the bottom of the page.

[21]For explanation and examples, see https://betterdev.blog/command-line-arguments-anatomy-explained/

```
20
21          If the user supplies --help or --version, then we simply respond to that and then return.
22          If they don't supply either of these, then we run :func:`lintrans.gui.main_window.main`.
23
24          ``prog_name`` is ``sys.argv[0]`` when this script is run with ``python -m lintrans``.
25
26          :param str prog_name: The name of the program
27          :param list[str] args: The other arguments to the program
28          """
29          if '-h' in args or '--help' in args:
30              print(dedent(f'''
31              Usage: {prog_name} [option]
32
33              Options:
34                  -h, --help      Display this help text and exit
35                  -V, --version   Display the version information and exit
36
37              Any other options will get passed to the QApplication constructor.
38              If you don't know what that means, then don't provide any arguments and just the run the program.'''[1:]))
39
40          elif '-V' in args or '--version' in args:
41              print(dedent(f'''
42              lintrans (version {__version__})
43              The linear transformation visualizer
44
45              Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
46
47              This program is licensed under GNU GPLv3, available here:
48              <https://www.gnu.org/licenses/gpl-3.0.html>'''[1:]))
49
50          else:
51              main_window.main(args)
52
53
54      if __name__ == '__main__':
55          main(sys.argv[0], sys.argv[1:])
```

Here is the expected output when using these flags at a shell prompt:

```
$ python -m lintrans --help
Usage: /home/dyson/repos/lintrans/src/lintrans/__main__.py [option]

Options:
    -h, --help      Display this help text and exit
    -V, --version   Display the version information and exit

Any other options will get passed to the QApplication constructor.
If you don't know what that means, then don't provide any arguments and just the run the program.
$ python -m lintrans --version
lintrans (version 0.2.0)
The linear transformation visualizer

Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)

This program is licensed under GNU GPLv3, available here:
<https://www.gnu.org/licenses/gpl-3.0.html>
```

3.9.7 Adding the about dialog

In addition to the help and version information available from the command line, most apps with a GUI provide a graphical way to access information about the app, typically in the form of an about box. I wanted to do this for lintrans. I had intended to have this from the start, but it now felt like the right time to implement it, since I was preparing to release version 0.2.1.

To do this, I create a new file to house the new dialog class, and then added the functionality to the original button, so that it would now open the new dialog.

```python
# 7423fff72f09b5f5a3253c734d42c4e7c3182efe
# src/lintrans/gui/dialogs/misc.py

"""This module provides miscellaneous dialog classes like :class:`AboutDialog`."""

from __future__ import annotations

import platform

from PyQt5 import QtWidgets
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QDialog, QVBoxLayout

import lintrans


class AboutDialog(QDialog):
    """A simple dialog class to display information about the app to the user.

    It only has an :meth:`__init__` method because it only has label widgets, so no other methods are necessary
    here.
    """

    def __init__(self, *args, **kwargs):
        """Create an :class:`AboutDialog` object with all the label widgets."""
        super().__init__(*args, **kwargs)

        self.setWindowTitle('About lintrans')

        # === Create the widgets

        label_title = QtWidgets.QLabel(self)
        label_title.setText(f'lintrans (version {lintrans.__version__})')
        label_title.setAlignment(Qt.AlignCenter)

        font_title = label_title.font()
        font_title.setPointSize(font_title.pointSize() * 2)
        label_title.setFont(font_title)

        label_version_info = QtWidgets.QLabel(self)
        label_version_info.setText(
            f'With Python version {platform.python_version()}\n'
            f'Running on {platform.platform()}'
        )
        label_version_info.setAlignment(Qt.AlignCenter)

        label_info = QtWidgets.QLabel(self)
        label_info.setText(
            'lintrans is a program designed to help visualise<br>'
            '2D linear transformations represented with matrices.<br><br>'
            "It's designed for teachers and students and any feedback<br>"
            'is greatly appreciated at <a href="https://github.com/DoctorDalek1963/lintrans" '
            'style="color: black;">my GitHub page</a><br>or via email '
            '(<a href="mailto:dyson.dyson@icloud.com" style="color: black;">dyson.dyson@icloud.com</a>).'
        )
        label_info.setAlignment(Qt.AlignCenter)
        label_info.setTextFormat(Qt.RichText)
        label_info.setOpenExternalLinks(True)

        label_copyright = QtWidgets.QLabel(self)
        label_copyright.setText(
            'This program is free software.<br>Copyright 2021-2022 D. Dyson (DoctorDalek1963).<br>'
            'This program is licensed under GPLv3, which can be found '
            '<a href="https://www.gnu.org/licenses/gpl-3.0.html" style="color: black;">here</a>.'
        )
        label_copyright.setAlignment(Qt.AlignCenter)
        label_copyright.setTextFormat(Qt.RichText)
        label_copyright.setOpenExternalLinks(True)

        # === Arrange the widgets

        self.setContentsMargins(10, 10, 10, 10)
```

```
76              vlay = QVBoxLayout()
77              vlay.setSpacing(20)
78              vlay.addWidget(label_title)
79              vlay.addWidget(label_version_info)
80              vlay.addWidget(label_info)
81              vlay.addWidget(label_copyright)
82
83              self.setLayout(vlay)
84
85              self.setFixedSize(self.baseSize())


        # 7423fff72f09b5f5a3253c734d42c4e7c3182efe
        # src/lintrans/gui/main_window.py

36      class LintransMainWindow(QMainWindow):
...
42          def __init__(self):
...
64              self.menu_help = QtWidgets.QMenu(self.menubar)
65              self.menu_help.setTitle('&Help')
...
97              self.action_about = QtWidgets.QAction(self)
98              self.action_about.setText('&About')
99              self.action_about.triggered.connect(lambda: dialogs.AboutDialog(self).open())
...
117             self.menu_help.addAction(self.action_about)
```

This feature works just as expected.
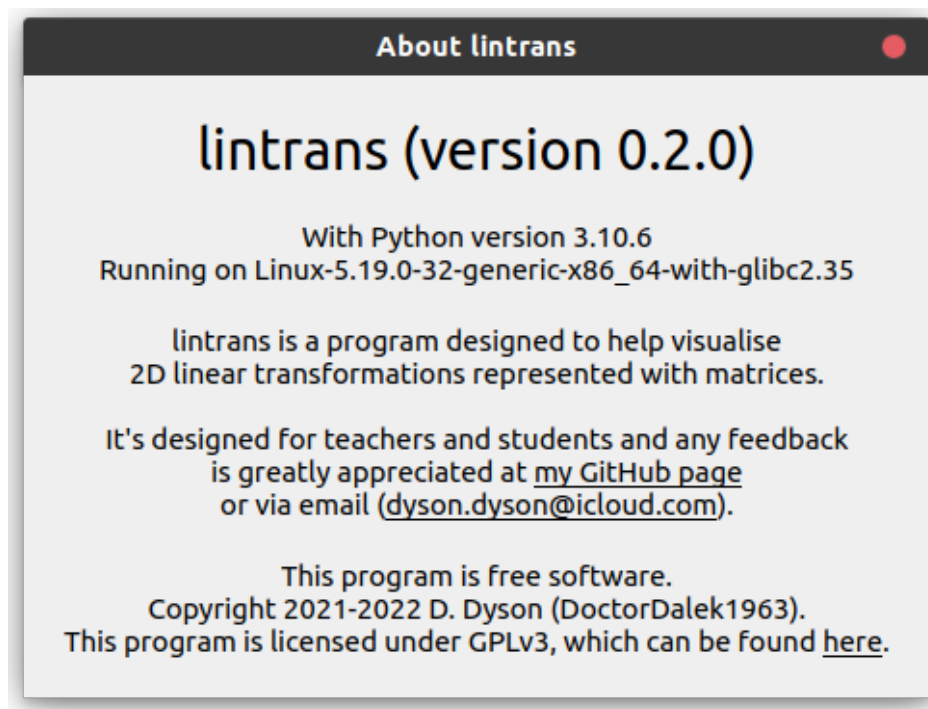
Figure 3.9.2: The about dialog

3.9.8   Creating the `FixedSizeDialog` class

Currently, several dialogs are a fixed size, meaning they can't be resized. Qt5 doesn't have the option to make a dialog non-resizeable - that's only available for whole windows - so to achieve this effect, I had to add the line `self.setFixedSize(self.baseSize())` to the end of the constructor for every fixed

size dialog[22]. This was easy to forget and it didn't make it immediately clear which dialogs were fixed size and which weren't.

To fix this, I decided to use a different technique, by overriding the `open()` method of the dialog. This method gets called when the dialog is first opened. If I called the superclass' implementation first to do all the actual opening logic, then I could use that same previous technique to set the fixed size to whatever Qt5 had calculated it to be. By overriding a method, I could put that override into a simple superclass called `FixedSizeDialog` and then make all fixed size dialogs inherit from this class instead of just `QDialog`.

```python
# a59ea87fb37fc593cf0bae3066bdbd24be656798
# src/lintrans/gui/dialogs/misc.py

20    class FixedSizeDialog(QDialog):
21        """A simple superclass to create modal dialog boxes with fixed size.
22
23        We override the :meth:`open` method to set the fixed size as soon as the dialog is opened modally.
24        """
25
26        def open(self) -> None:
27            """Override :meth:`QDialog.open` to set the dialog to a fixed size."""
28            super().open()
29            self.setFixedSize(self.size())
...
32    class AboutDialog(FixedSizeDialog):


# a59ea87fb37fc593cf0bae3066bdbd24be656798
# src/lintrans/gui/dialogs/define_new_matrix.py

68    class DefineDialog(FixedSizeDialog):


# a59ea87fb37fc593cf0bae3066bdbd24be656798
# src/lintrans/gui/dialogs/settings.py

21    class SettingsDialog(FixedSizeDialog):
```

I used `self.size()` here instead of `self.baseSize()` like before. There doesn't seem to be much difference, but the docs seem to suggest that `self.size()` is more appropriate for this use case[46][49].

3.9.9 Increasing minimum grid spacing

When you zoom out, the grid lines get closer and closer together. I previously limited this to 1 pixel, since the program would hang or crash if it was any lower. However, having the grid lines that close together made it very hard to see what was happening. I could easily fix this by limiting the minimum grid spacing to say, 5 pixels.

---

[22]This works because Qt5 needs to arrange all the widgets and allocate space for all of them first, and then we can use that calculated size as the fixed size.
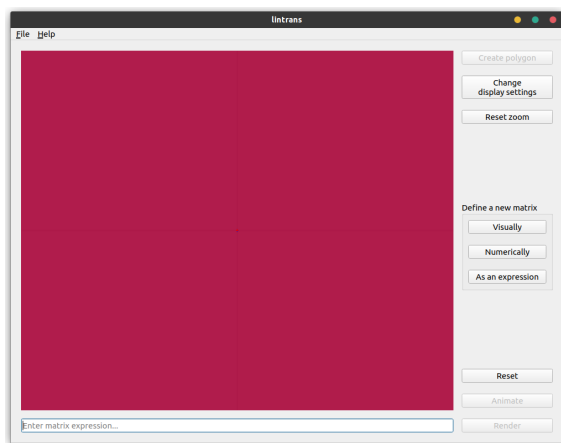
Figure 3.9.3: Fully zoomed out before
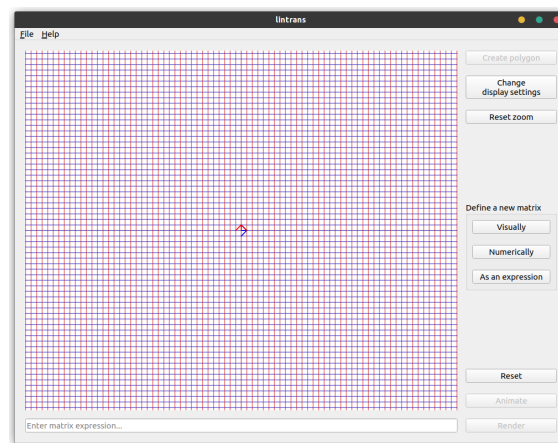
Figure 3.9.4: Fully zoomed out after

```
# 51d35018c81deaaf7e32646b1f99d6e46643bf24
# src/lintrans/gui/plots/classes.py

23    class BackgroundPlot(QWidget):
...
39        minimum_grid_spacing: int = 5
...
154       def wheelEvent(self, event: QWheelEvent) -> None:
155           """Handle a :class:`QWheelEvent` by zooming in or our of the grid."""
156           # angleDelta() returns a number of units equal to 8 times the number of degrees rotated
157           degrees = event.angleDelta() / 8
158
159           if degrees is not None:
160               new_spacing = max(1, self.grid_spacing + degrees.y())
161
162               if new_spacing >= self.minimum_grid_spacing:
163                   self.grid_spacing = new_spacing
164
165           event.accept()
166           self.update()
```

3.9.10   Creating a changelog

I've know about changelogs for a long time. They're used to keep track of changes to a project over time. I recently learned about the project 'keep a changelog', which tries to create a standard for changelog formats[27]. I want to have a proper changelog for lintrans, so I decided to implement one.

```
<!-- 47c68c7f4780d0e2c374cf12b9b54c031277af6d -->
<!-- CHANGELOG.md -->

1    # Changelog
2
3    All notable changes to this project will be documented in this file.
4
5    The format is based on [Keep a Changelog](https://keepachangelog.com/en/1.0.0/),
6    and this project adheres to [Semantic Versioning](https://semver.org/spec/v2.0.0.html).
7
8    ## [Unreleased]
9
10   ### Added
11
12   - Explicit `@pyqtSlot` decorators
13   - Link to Qt5 docs in project docs with intersphinx
14   - Copyright comment in tests and `setup.py`
15   - Create version file for Windows compilation
16   - Create full compile.py script
17   - Add `Info.plist` file for macOS compilation
```

```
18    - Support --help and --version flags in `__main__.py`
19    - Create about dialog in help menu
20    - Implement minimum grid spacing
21
22    ### Fixed
23
24    - Fix problems with compile script
25    - Fix small bugs and docstrings
26
27    ## [0.2.0] - 2022-03-11
28
29    There were alpha tags before this, but I wasn't properly adhering to semantic versioning, so I'll start the
   ↪    changelog here.
30
31    If I'd been using semantic versioning from the start, there would much more changelog here, but instead, I'll just
   ↪    summarise the features.
32
33    ### Added
34
35    - Matrix context with the `MatrixWrapper` class
36    - Parsing and evaluating matrix expressions
37    - A simple GUI with a viewport to render linear transformations
38    - Simple dialogs to create matrices and assign them to names
39    - Ability to render and animate linear transformations parsed from defined matrices
40    - Ability to zoom in and out of the viewport
41    - Add dialog to change display settings
42
43    [Unreleased]: https://github.com/DoctorDalek1963/lintrans/compare/v0.2.0...HEAD
44    [0.2.0]: https://github.com/DoctorDalek1963/lintrans/compare/13600cc6ff6299dc4a8101a367bc52fe08607554...v0.2.0
```

3.9.11 Releasing `v0.2.1`

Now all I had to do for the release was update the version number in `__init__.py` and update the changelog to include the changes under the correct heading, and add a new, empty 'Unreleased' heading.

```
# d47f63eb0bcdc89cff5ed19afe2fc63899edf1d9
# src/lintrans/__init__.py

13    __version__ = '0.2.1'
```

```
<!-- d47f63eb0bcdc89cff5ed19afe2fc63899edf1d9 -->
<!-- CHANGELOG.md -->

1    # Changelog
2
3    All notable changes to this project will be documented in this file.
4
5    The format is based on [Keep a Changelog](https://keepachangelog.com/en/1.0.0/),
6    and this project adheres to [Semantic Versioning](https://semver.org/spec/v2.0.0.html).
7
8    ## [Unreleased]
9
10    Nothing here yet...
11
12    ## [0.2.1] - 2022-03-22
13
14    ### Added
15
16    - Explicit `@pyqtSlot` decorators
17    - Link to Qt5 docs in project docs with intersphinx
18    - Copyright comment in tests and `setup.py`
19    - Create version file for Windows compilation
20    - Create full compile.py script
21    - Add `Info.plist` file for macOS compilation
22    - Support --help and --version flags in `__main__.py`
23    - Create about dialog in help menu
24    - Implement minimum grid spacing
```

```
25
26    ### Fixed
27
28    - Fix problems with compile script
29    - Fix small bugs and docstrings
30
31    ## [0.2.0] - 2022-03-11
32
33    There were alpha tags before this, but I wasn't properly adhering to semantic versioning, so I'll start the
  ↪    changelog here.
34
35    If I'd been using semantic versioning from the start, there would much more changelog here, but instead, I'll just
  ↪    summarise the features.
36
37    ### Added
38
39    - Matrix context with the `MatrixWrapper` class
40    - Parsing and evaluating matrix expressions
41    - A simple GUI with a viewport to render linear transformations
42    - Simple dialogs to create matrices and assign them to names
43    - Ability to render and animate linear transformations parsed from defined matrices
44    - Ability to zoom in and out of the viewport
45    - Add dialog to change display settings
46
47    [Unreleased]: https://github.com/DoctorDalek1963/lintrans/compare/v0.2.1...HEAD
48    [0.2.1]: https://github.com/DoctorDalek1963/lintrans/compare/v0.2.0...v0.2.1
49    [0.2.0]: https://github.com/DoctorDalek1963/lintrans/compare/13600cc6ff6299dc4a8101a367bc52fe08607554...v0.2.0
```


Figure 3.9.5: The release of `v0.2.1` on GitHub

3.9.12 Automating release note generation

I had to copy the changelog over and polish up the release notes myself for this release. It would be quite convenient to have a script that does this automatically, so I made one.

```python
# 99a88575f9beb8fed2dcc41dacbb020b31bc8176
# generate_release_notes.py

"""A very simple script to generate release notes."""

import re
import sys

TEXT = '''DESCRIPTION

---

The Linux binary should work fine, but if you use the Windows `.exe` file, you will get a warning that the program
↪ may be unsafe. This is expected and you can just ignore it. There's no binary for macOS due to Apple code
↪ signing issues.

If you're running macOS, then you will need to compile the program from source. This is also an option on Linux and
↪ Windows. Instructions can be found [here](https://doctordalek1963.github.io/lintrans/tutorial/compile/).

---

CHANGELOG
'''

# This RegEx is complicated because of the newlines
# It requires the current tag to have a header like
# ## [0.2.1] - 2022-03-22
# And all other tags to have similar headers
# It also won't work on the first tag, but that's fine
RE_PATTERN = r'''(?<=## \[TAG_NAME\] - \d{4}-\d{2}-\d{2}

).*?(?=

## \[\d+\.\d+\.\d+(-[\S]+)?\] - \d{4}-\d{2}-\d{2})'''


def main(args: list[str]) -> None:
    """Generate the release notes for this release and write them to `release_notes.md`."""
    if len(args) < 1:
        raise ValueError('Tag name is required to generate release notes')

    tag_name = args[0]

    print(f'Generating release notes for tag {tag_name}')

    with open('CHANGELOG.md', 'r', encoding='utf-8') as f:
        changelog_text = f.read()

    if (m := re.search(
        RE_PATTERN.replace('TAG_NAME', re.escape(tag_name[1:])),
        changelog_text,
        flags=re.S
    )) is not None:
        text = TEXT.replace('CHANGELOG', m.group(0))

    else:
        raise ValueError('Error in searching for changelog notes. Bad format')

    with open('release_notes.md', 'w', encoding='utf-8') as f:
        f.write(text)


if __name__ == '__main__':
    main(sys.argv[1:])
```

This script just parses the changelog and generates a file called `release_notes.md`, which I can then automatically use as the body for the GitHub release by changing the workflow.

```yaml
# 99a88575f9beb8fed2dcc41dacbb020b31bc8176
# .github/workflows/compile-release.yaml

  8    jobs:
...
 97      publish:
...
101        steps:
...
124          - name: Generate release notes
125            run: python generate_release_notes.py $GITHUB_REF_NAME
126
127          # This is practically the same step twice just to allow for pre-releases
128          - name: Upload binaries (normal release)
129            if: steps.checkprerelease.outputs.isprerelease == 0
130            uses: softprops/action-gh-release@v1
131            with:
132              fail_on_unmatched_files: true
133              prerelease: false
134              draft: true
135              body_path: release_notes.md
136              files: dist/lintrans*
137
138          - name: Upload binaries (pre-release)
139            if: steps.checkprerelease.outputs.isprerelease == 1
140            uses: softprops/action-gh-release@v1
141            with:
142              fail_on_unmatched_files: true
143              prerelease: true
144              draft: true
145              body_path: release_notes.md
146              files: dist/lintrans*
```

3.10   Making `v0.2.2`

3.10.1   Hiding the background and transformed grids

I spoke to my main stakeholder, who is the teacher that will be using `lintrans` when it's finished, and she said that the background grid and transformed grid can get a little bit in the way of the core action and make it harder to understand what's happening. Taking this feedback on board, I decided to add a display setting to toggle the background grid, and one to toggle the transformed grid.

I did the background grid first and then repeated everything for the transformed version of the grid as well. I am combining them here for brevity. The first step was of course to add a display setting for each of them. Then I had to add checkboxes for them in the display settings dialog, and then incorporate the settings into the actual drawing of the canvas.

```
     # d045057d568ac133b621ee9ca9daed361d570d7a
     # src/lintrans/gui/settings.py

14   @dataclass
15   class DisplaySettings:
16       """This class simply holds some attributes to configure display."""
17
18       # === Basic stuff
19
20       draw_background_grid: bool = True
21       """This controls whether we want to draw the background grid.
22
23       The background axes will always be drawn. This makes it easy to identify the center of the space.
24       """
25
26       draw_transformed_grid: bool = True
27       """This controls whether we want to draw the transformed grid. Vectors are handled separately."""
```

```
     # d045057d568ac133b621ee9ca9daed361d570d7a
     # src/lintrans/gui/dialogs/settings.py

70   class DisplaySettingsDialog(SettingsDialog):
...
73       def __init__(self, display_settings: DisplaySettings, *args, **kwargs):
...
89           self.checkbox_draw_background_grid = QCheckBox(self)
90           self.checkbox_draw_background_grid.setText('Draw &background grid')
91           self.checkbox_draw_background_grid.setToolTip(
92               'Draw the background grid (axes are always drawn)'
93           )
94           self.dict_checkboxes['b'] = self.checkbox_draw_background_grid
95
96           self.checkbox_draw_transformed_grid = QCheckBox(self)
97           self.checkbox_draw_transformed_grid.setText('Draw t&ransformed grid')
98           self.checkbox_draw_transformed_grid.setToolTip(
99               'Draw the transformed grid (vectors are handled separately)'
100          )
101          self.dict_checkboxes['r'] = self.checkbox_draw_transformed_grid
...
204      def load_settings(self) -> None:
...
207          self.checkbox_draw_background_grid.setChecked(self.display_settings.draw_background_grid)
208          self.checkbox_draw_transformed_grid.setChecked(self.display_settings.draw_transformed_grid)
...
221      def confirm_settings(self) -> None:
...
224          self.display_settings.draw_background_grid = self.checkbox_draw_background_grid.isChecked()
225          self.display_settings.draw_transformed_grid = self.checkbox_draw_transformed_grid.isChecked()
```

```
     # d045057d568ac133b621ee9ca9daed361d570d7a
     # src/lintrans/gui/plots/widgets.py
```

```
19     class VisualizeTransformationWidget(VectorGridPlot):
...
48         def paintEvent(self, event: QPaintEvent) -> None:
...
60             self.draw_background(painter, self.display_settings.draw_background_grid)
61
62             if self.display_settings.draw_transformed_grid:
63                 self.draw_transformed_grid(painter)
```

```
# d045057d568ac133b621ee9ca9daed361d570d7a
# src/lintrans/gui/plots/classes.py
```

```
23     class BackgroundPlot(QWidget):
...
129        def draw_background(self, painter: QPainter, draw_grid: bool) -> None:
130            """Draw the background grid.
131
132            .. note:: This method is just a utility method for subclasses to use to render the background grid.
133
134            :param QPainter painter: The painter to draw the background with
135            :param bool draw_grid: Whether to draw the grid lines
136            """
137            if draw_grid:
138                painter.setPen(QPen(self.colour_background_grid, self.width_background_grid))
139
140                # Draw equally spaced vertical lines, starting in the middle and going out
141                # We loop up to half of the width. This is because we draw a line on each side in each iteration
142                for x in range(self.width() // 2 + self.grid_spacing, self.width(), self.grid_spacing):
143                    painter.drawLine(x, 0, x, self.height())
144                    painter.drawLine(self.width() - x, 0, self.width() - x, self.height())
145
146                # Same with the horizontal lines
147                for y in range(self.height() // 2 + self.grid_spacing, self.height(), self.grid_spacing):
148                    painter.drawLine(0, y, self.width(), y)
149                    painter.drawLine(0, self.height() - y, self.width(), self.height() - y)
150
151            # Now draw the axes
152            painter.setPen(QPen(self.colour_background_axes, self.width_background_grid))
153            painter.drawLine(self.width() // 2, 0, self.width() // 2, self.height())
154            painter.drawLine(0, self.height() // 2, self.width(), self.height() // 2)
```

Then I added this change to the changelog.

```
<!-- d045057d568ac133b621ee9ca9daed361d570d7a -->
<!-- CHANGELOG.md -->
```

```
12     ### Added
13
14     - Add options to hide background grid and transformed grid
```

3.10.2 Hiding the basis vectors

While I was implementing new display settings, I decided to implement hiding basis vectors. This will give users the option of just seeing the grid get transformed. The process was exactly the same as before. Add the setting, add it to the dialog, use it when drawing.

```
# 11ffbaf71f9fe29e1832a62f2b127aa3939e520d
# src/lintrans/gui/settings.py
```

```
15     class DisplaySettings:
...
29         draw_basis_vectors: bool = True
30         """This controls whether we want to draw the transformed basis vectors."""
```

```
# 11ffbaf71f9fe29e1832a62f2b127aa3939e520d
# src/lintrans/gui/dialogs/settings.py
```

```
# d045057d568ac133b621ee9ca9daed361d570d7a
```

```
70      class DisplaySettingsDialog(SettingsDialog):
...
73          def __init__(self, display_settings: DisplaySettings, *args, **kwargs):
...
103             self.checkbox_draw_basis_vectors = QCheckBox(self)
104             self.checkbox_draw_basis_vectors.setText('Draw basis &vectors')
105             self.checkbox_draw_basis_vectors.setToolTip(
106                 'Draw the transformed basis vectors'
107             )
108             self.dict_checkboxes['v'] = self.checkbox_draw_basis_vectors
...
212         def load_settings(self) -> None:
...
217             self.checkbox_draw_basis_vectors.setChecked(self.display_settings.draw_basis_vectors)
...
230         def confirm_settings(self) -> None:
...
235             self.display_settings.draw_basis_vectors = self.checkbox_draw_basis_vectors.isChecked()
```

```
# 11ffbaf71f9fe29e1832a62f2b127aa3939e520d
# src/lintrans/gui/plots/widgets.py

19      class VisualizeTransformationWidget(VectorGridPlot):
...
48          def paintEvent(self, event: QPaintEvent) -> None:
...
65              if self.display_settings.draw_basis_vectors:
66                  self.draw_basis_vectors(painter)
```

And then of course add it to the changelog.

```
<!-- 11ffbaf71f9fe29e1832a62f2b127aa3939e520d -->
<!-- CHANGELOG.md -->

12      ### Added
13
14      - Add options to hide background grid, transformed grid, and basis vectors
```

3.10.3 Improving argument parsing

Qt5 accepts arguments to its main method. I don't really know what these arguments can do, but it would be nice to be able to use them. I also want to be able to save sessions as files in the future, and it would be quite useful to open a session file by passing it as a command line argument. To make both of these easier, I decided to refactor my argument parsing.

Python has a built-in library called argparse, which allows for more sophisticated argument parsing. One of the things argparse can do is parse only some of the command line arguments with a method called parse_known_args()[41]. I can then pass the unconsumed arguments on to Qt5. __main__.py now looks like this:

```
# a688a14839caba2ee14f8551764b771ae803d935
# src/lintrans/__main__.py

9       """This module provides a :func:`main` function to interpret command line arguments and run the program."""
10
11      from argparse import ArgumentParser
12      import sys
13      from textwrap import dedent
14
15      from lintrans import __version__
16      from lintrans.gui import main_window
17
18
```

```python
19    def main(args: list[str]) -> None:
20        """Interpret program-specific command line arguments and run the main window in most cases.
21
22        If the user supplies --help or --version, then we simply respond to that and then return.
23        If they don't supply either of these, then we run :func:`lintrans.gui.main_window.main`.
24
25        :param list[str] args: The full argument list (including program name)
26        """
27        parser = ArgumentParser(add_help=False)
28
29        parser.add_argument(
30            '-h',
31            '--help',
32            default=False,
33            action='store_true'
34        )
35
36        parser.add_argument(
37            '-V',
38            '--version',
39            default=False,
40            action='store_true'
41        )
42
43        parsed_args, unparsed_args = parser.parse_known_args()
44
45        if parsed_args.help:
46            print(dedent('''
47            Usage: lintrans [option]
48
49            Options:
50                -h, --help       Display this help text and exit
51                -V, --version    Display the version information and exit
52
53            Any other options will get passed to the QApplication constructor.
54            If you don't know what that means, then don't provide any arguments and just the run the program.'''[1:]))
55            return
56
57        if parsed_args.version:
58            print(dedent(f'''
59            lintrans (version {__version__})
60            The linear transformation visualizer
61
62            Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
63
64            This program is licensed under GNU GPLv3, available here:
65            <https://www.gnu.org/licenses/gpl-3.0.html>'''[1:]))
66            return
67
68        for arg in unparsed_args:
69            print(f'Passing "{arg}" to QApplication. See --help for recognised args')
70
71        main_window.main(args[:1] + unparsed_args)
72
73
74    if __name__ == '__main__':
75        main(sys.argv)
```

The '`args[:1]` + `unparsed_args`' on line 71 means that we pass the name of the program first, and the rest of the unconsumed arguments after it.

And of course, I added it to the changelog, this time as a fix rather than an addition:

```
<!-- a688a14839caba2ee14f8551764b771ae803d935 -->
<!-- CHANGELOG.md -->

16    ### Fixed
17
18    - Improve command line argument handling
```

3.10.4 Respecting display settings in the visual definition dialog

DefineVisuallyWidget is a subclass of VisualizeTransformationWidget. If it had its own instance attribute of type DisplaySettings, then it could use its superclass's paintEvent() method, and that would respect the display settings in the visual definition dialog.

```python
# 5850aa916b685992f31e58680267916927ed590d
# src/lintrans/gui/plots/widgets.py

84  class DefineVisuallyWidget(VisualizeTransformationWidget):
...
91      def __init__(self, *args, display_settings: DisplaySettings, **kwargs):
92          """Create the widget and enable mouse tracking. ``*args`` and ``**kwargs`` are passed to ``super()``."""
93          super().__init__(*args, display_settings=display_settings, **kwargs)
```

```python
# 5850aa916b685992f31e58680267916927ed590d
# src/lintrans/gui/dialogs/define_new_matrix.py

166  class DefineVisuallyDialog(DefineDialog):
...
169      def __init__(self, *args, matrix_wrapper: MatrixWrapper, display_settings: DisplaySettings, **kwargs):
170          """Create the widgets and layout of the dialog.
171
172          :param MatrixWrapper matrix_wrapper: The MatrixWrapper that this dialog will mutate
173          """
174          super().__init__(*args, matrix_wrapper=matrix_wrapper, **kwargs)
175
176          self.setMinimumSize(700, 550)
177
178          # === Create the widgets
179
180          self.plot = DefineVisuallyWidget(self, display_settings=display_settings)
```

Since the DefineVisuallyDialog now accepts display settings but the other definition dialogs don't, we need change the LintransMainWindow.dialog_define_matrix() method to treat the visual definition dialog differently.

```python
# 5850aa916b685992f31e58680267916927ed590d
# src/lintrans/gui/main_window.py

36  class LintransMainWindow(QMainWindow):
...
447      @pyqtSlot(DefineDialog)
448      def dialog_define_matrix(self, dialog_class: Type[DefineDialog]) -> None:
449          """Open a generic definition dialog to define a new matrix.
450
451          The class for the desired dialog is passed as an argument. We create an
452          instance of this class and the dialog is opened asynchronously and modally
453          (meaning it blocks interaction with the main window) with the proper method
454          connected to the :meth:`QDialog.accepted` signal.
455
456          .. note:: ``dialog_class`` must subclass :class:`lintrans.gui.dialogs.define_new_matrix.DefineDialog`.
457
458          :param dialog_class: The dialog class to instantiate
459          :type dialog_class: Type[lintrans.gui.dialogs.define_new_matrix.DefineDialog]
460          """
461          # We create a dialog with a deepcopy of the current matrix_wrapper
462          # This avoids the dialog mutating this one
463          if dialog_class == DefineVisuallyDialog:
464              dialog = DefineVisuallyDialog(
465                  self,
466                  matrix_wrapper=deepcopy(self.matrix_wrapper),
467                  display_settings=self.plot.display_settings
468              )
469          else:
470              dialog = dialog_class(self, matrix_wrapper=deepcopy(self.matrix_wrapper))
471
```

```
472          # .open() is asynchronous and doesn't spawn a new event loop, but the dialog is still modal (blocking)
473          dialog.open()
474
475          # So we have to use the accepted signal to call a method when the user accepts the dialog
476          dialog.accepted.connect(self.assign_matrix_wrapper)
```

And, of course, I updated the changelog:

```
<!-- 5850aa916b685992f31e58680267916927ed590d -->
<!-- CHANGELOG.md -->

12   ### Added
...
15   - Fully respect display settings in visual definition widget
```

3.10.5 Changing the order in which things are drawn

Currently, `VisualizeTransformationWidget` draws the background, then the transformed grid, then the basis vectors, then the eigenlines and eigenvectors, then the determinant parallelogram and text. This means that the determinant parallelogram gets drawn on top of the basis vectors, which doesn't look very good. To fix this, we can simply re-order the drawing of different things. If we instead draw the transformed grid and basis vectors last, then they will appear on top of everything else, which should look significantly better.

I also renamed the `draw_determinant_text` display setting attribute to `show_determinant_value`.

Before:

```
# e9da6737cbdb68e800c245bcc34e1c5c3824458a
# src/lintrans/gui/plots/widgets.py

19   class VisualizeTransformationWidget(VectorGridPlot):
...
48       def paintEvent(self, event: QPaintEvent) -> None:
...
60           self.draw_background(painter, self.display_settings.draw_background_grid)
61
62           if self.display_settings.draw_transformed_grid:
63               self.draw_transformed_grid(painter)
64
65           if self.display_settings.draw_basis_vectors:
66               self.draw_basis_vectors(painter)
67
68           if self.display_settings.draw_eigenlines:
69               self.draw_eigenlines(painter)
70
71           if self.display_settings.draw_eigenvectors:
72               self.draw_eigenvectors(painter)
73
74           if self.display_settings.draw_determinant_parallelogram:
75               self.draw_determinant_parallelogram(painter)
76
77               if self.display_settings.draw_determinant_text:
78                   self.draw_determinant_text(painter)
```

After:

```
# acdf206a69d346dce67f74f2c54ff4c512c96229
# src/lintrans/gui/plots/widgets.py

19   class VisualizeTransformationWidget(VectorGridPlot):
...
48       def paintEvent(self, event: QPaintEvent) -> None:
```

```
...
60              self.draw_background(painter, self.display_settings.draw_background_grid)
61
62              if self.display_settings.draw_eigenlines:
63                  self.draw_eigenlines(painter)
64
65              if self.display_settings.draw_eigenvectors:
66                  self.draw_eigenvectors(painter)
67
68              if self.display_settings.draw_determinant_parallelogram:
69                  self.draw_determinant_parallelogram(painter)
70
71                  if self.display_settings.show_determinant_value:
72                      self.draw_determinant_text(painter)
73
74              if self.display_settings.draw_transformed_grid:
75                  self.draw_transformed_grid(painter)
76
77              if self.display_settings.draw_basis_vectors:
78                  self.draw_basis_vectors(painter)
```
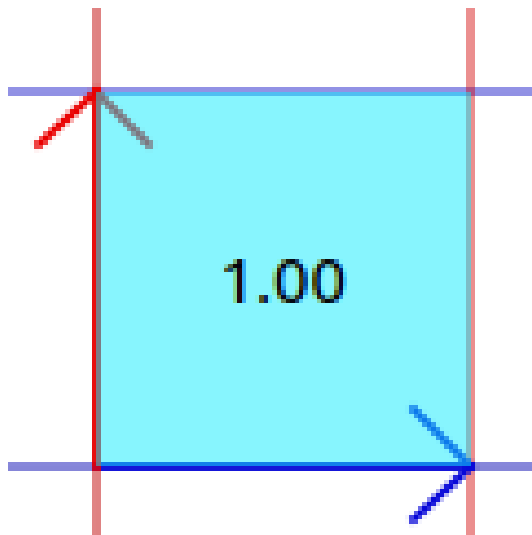

Figure 3.10.1: Before re-ordering drawing commands
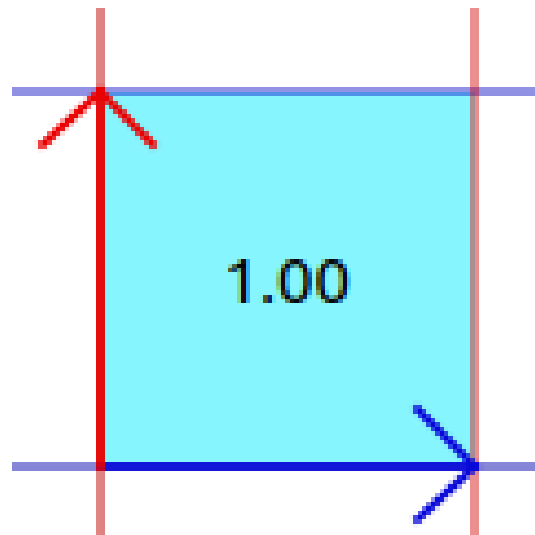
Figure 3.10.2: After re-ordering drawing commands

3.10.6 Improving online documentation with *Read the Docs*

So far, I've been building my documentation automatically using a tool called Sphinx[55]. I've been using it to scan through my source code, extract all the docstrings[20], and compile a HTML version of the source code documentation. I then publish this documentation to GitHub Pages[18], and all that happens automatically along with the unit tests whenever I push my changes to GitHub.

However, this system can't track changes in documentation over time. There's a popular website for hosting open source documentation, especially for Python packages, called *Read the Docs*[51]. It would be nice to host my documentation on there, since it would provide a standard build for the docs, and would allow me to keep track of documentation for different versions over time. I read through their tutorial and followed along with it, and then I could use Read the Docs for lintrans.

```
# a7a8aa09148f4acf8591547d6d8b9cd8b8d52905
# .readthedocs.yaml

1   version: 2
2
```

```
 3    build:
 4      os: ubuntu-20.04
 5      tools:
 6        python: "3.10"
 7      apt_packages:
 8        - graphviz
 9
10      jobs:
11        pre_build:
12          - cd docs/ && $(pwd | sed "s/checkouts\(\/[^/]\+\)\/docs\$/envs\1/")/bin/python create_objects_inv.py
13          - $(pwd | sed "s/checkouts\(\/[^/]\+\)\$/envs\1/")/bin/python -m pylint --rcfile=/dev/null --exit-zero
              ↪  --reports=y --disable=all --enable=imports,RP0402 --int-import-graph=docs/source/int-imports.png $(find
              ↪  ./src/lintrans/ -name "*.py" | tr "\n" " ")
14          - mkdir -p docs/source/_static
15          - $(pwd | sed "s/checkouts\(\/[^/]\+\)\$/envs\1/")/bin/python -m pip install -e .
16          - $(pwd | sed "s/checkouts\(\/[^/]\+\)\$/envs\1/")/bin/python -c "import lintrans" && echo success || echo
              ↪  fail
17
18    sphinx:
19      builder: html
20      configuration: docs/source/conf.py
21      fail_on_warning: true
22
23    python:
24      install:
25        - requirements: requirements.txt
26        - requirements: docs/docs_requirements.txt
27
28      system_packages: false
```

This file governs the build pipeline for the documentation. It defines how the docs will be built on the remote machine. It basically just installs all the dependencies and `lintrans` itself in a virtual environment, generates the object inventory for `intersphinx` (see §3.9.2), builds the internal import graph (see §3.8.4), and makes sure that lintrans can be installed successfully in the virtual environment. The string '`$(pwd | sed "s/checkouts\(\/[^/]\+\)\/docs\$/envs\1/")/bin/python`' appears frequently. This string finds the version of Python that will be used for building the actual documentation at the end of the pipeline. `lintrans` has some issues with circular imports, which means that it can only be installed in editable mode[23]. This means that I have to bodge the build system to install `lintrans` properly.

Additionally, having a build system like this means that builds should be repeatable, which means I should pin the exact versions of all my dependencies, to avoid any breaking changes in the future.

```
# 152f9e59b5b4e22607cf2b687c85c70708054579
# requirements.txt

1    nptyping==1.4.4
2    numpy==1.21.0
3    pyqt5==5.15.6
```

```
# 152f9e59b5b4e22607cf2b687c85c70708054579
# dev_requirements.txt

1    flake8==4.0.1
2    mypy==0.942
3    pycodestyle==2.8.0
4    pydocstyle==6.1.1
5    pytest==6.2.5
6    pyqt5-stubs==5.15.2.0
7    toml==0.10.2
```

---

[23]Editable mode is an option that you can choose when installing a package with `pip`[32]. Instead of copying the source code to the installation directory like normal, it creates a link to the source code from the installation directory - basically a symlink in Linux. This means that if you change the source code, the installed version gets updated instantly, which is obviously very useful for fast, iterative development. It also has some strange effects with import order and resolution. Importing packages and modules is very complicated in Python[12], and I repeatedly failed to get `lintrans` to be installable without editable mode, so I had to resort to just using this bodge on Read the Docs instead.

```
# 152f9e59b5b4e22607cf2b687c85c70708054579
# docs/docs_requirements.txt

1    Sphinx==4.3.2
2    sphinx-rtd-theme==1.0.0
3    sphobjinv==2.2
```

Now that I was using Read the Docs, I could remove the old GitHub Actions workflow to compile the documentation.

3.10.7 Parsing parentheses

`git` has a feature called branches[16], which allow you to work on different things simultaneously. A branch is basically a sandbox where you can make changes and focus on a particular feature while not having to worry about the rest of the project. It's particularly good for teams where different people can work on different features at the same time, but it's also useful for individual developers. The first feature branch I merged for `lintrans` was called `dev/parse-parens`. I created it a week or two ago, and I used it to focus on improving the parser to understand parenthesised expressions like `"A(B+C)^2"`.

3.10.7.1 Extending validation

The first thing I needed to do was allow parenthesised expressions as valid. If the validator rejects them, then they can never be parsed. Any expression can be evaluated to a matrix, and a pair of balanced parentheses can contain any valid expression. As such, a parenthesised expression can be considered as a type of matrix.

```
# 1e49c1479939b8de04751ab1a798afb2145b1550
# docs/source/bnf.txt

1    expression       ::=  [ "-" ] matrices { ( "+" | "-" ) matrices };
2    matrices         ::=  matrix { matrix };
3    matrix           ::=  [ real_number ] matrix_identifier [ index ] | "(" expression ")";
4    matrix_identifier ::= "A" .. "Z" | "rot(" [ "-" ] real_number ")";
5    index            ::=  "^{" index_content "}" | "^" index_content;
6    index_content    ::=  [ "-" ] integer_not_zero | "T";
7
8    digit_no_zero    ::=  "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
9    digit            ::=  "0" | digit_no_zero;
10   digits           ::=  digit | digits digit;
11   integer_not_zero ::=  digit_no_zero [ digits ];
12   real_number      ::=  ( integer_not_zero [ "." digits ] | [ "0" ] "." digits );
```

```
# 1e49c1479939b8de04751ab1a798afb2145b1550
# src/lintrans/matrices/parse.py

16   NAIVE_CHARACTER_CLASS = r'[-+\sA-Z0-9.rot()^{}]'
...
23   def compile_naive_expression_pattern() -> Pattern[str]:
24       """Compile the single RegEx pattern that will match a valid matrix expression."""
25       digit_no_zero = '[123456789]'
26       digits = '\\d+'
27       integer_no_zero = digit_no_zero + '(' + digits + ')?'
28       real_number = f'({integer_no_zero}(\\.{digits})?|0?\\.{digits})'
29
30       index_content = f'(-?{integer_no_zero}|T)'
31       index = f'(\\^{{{index_content}}}|\\^{index_content})'
32       matrix_identifier = f'([A-Z]|rot\\(-?{real_number}\\)|\\({NAIVE_CHARACTER_CLASS}+\\))'
33       matrix = '(' + real_number + '?' + matrix_identifier + index + '?)'
34       expression = f'^-?{matrix}+((\\+|-){matrix}+)*$'
35
36       return re.compile(expression)
```

```
37
38
39      # This is an expensive pattern to compile, so we compile it when this module is initialized
40      naive_expression_pattern = compile_naive_expression_pattern()
41
42
43      def find_sub_expressions(expression: str) -> list[str]:
44          """Find all the sub-expressions in the given expression.
45
46          This function only goes one level deep, so may return strings like ``'A(BC)D'``.
47
48          :raises MatrixParseError: If there are unbalanced parentheses
49          """
50          sub_expressions: list[str] = []
51          string = ''
52          paren_depth = 0
53          pointer = 0
54
55          while True:
56              char = expression[pointer]
57
58              if char == '(' and expression[pointer - 3:pointer] != 'rot':
59                  paren_depth += 1
60
61                  # This is a bit of a manual bodge, but it eliminates extraneous parens
62                  if paren_depth == 1:
63                      pointer += 1
64                      continue
65
66              elif char == ')' and re.match(f'{NAIVE_CHARACTER_CLASS}*?rot\\([-\\d.]+$', expression[:pointer]) is None:
67                  paren_depth -= 1
68
69              if paren_depth > 0:
70                  string += char
71
72              if paren_depth == 0 and string:
73                  sub_expressions.append(string)
74                  string = ''
75
76              pointer += 1
77
78              if pointer >= len(expression):
79                  break
80
81          if paren_depth != 0:
82              raise MatrixParseError('Unbalanced parentheses in expression')
83
84          return sub_expressions
85
86
87      def validate_matrix_expression(expression: str) -> bool:
88          """Validate the given matrix expression.
89
90          This function simply checks the expression against the BNF schema documented in
91          :ref:`expression-syntax-docs`. It is not aware of which matrices are actually defined
92          in a wrapper. For an aware version of this function, use the
93          :meth:`lintrans.matrices.wrapper.MatrixWrapper.is_valid_expression` method.
94
95          :param str expression: The expression to be validated
96          :returns bool: Whether the expression is valid according to the schema
97          """
98          # Remove all whitespace
99          expression = re.sub(r'\s', '', expression)
100
101         match = naive_expression_pattern.match(expression)
102
103         if match is None:
104             return False
105
106         # Check that the whole expression was matched against
107         if expression != match.group(0):
108             return False
109
```

```
110        try:
111            sub_expressions = find_sub_expressions(expression)
112        except MatrixParseError:
113            return False
114
115        if not sub_expressions:
116            return True
117
118        return all(validate_matrix_expression(m) for m in sub_expressions)
```

We create a naïve character class that just contains all characters that could possibly be in a valid expression. Then when we compile the naïve expression pattern, we say that a matrix identifier can be one of three things: it can be a capital letter, the `rot()` command with a real number angle, or a balanced pair of parentheses around some characters. The characters in the parentheses are not validated at this stage (hence why this pattern is naïve), but we know that those characters are only allowed to be from the naïve character class.

Then we have a function that finds sub-expressions in a given expression. It simply scans through the given string with a pointer and keeps track of the current sub-expression. It uses the `paren_depth` variable to keep track of how deep into a sub-expression it currently is. This function deliberately avoids counting the parentheses from `rot()` commands towards the `paren_depth`. This is to avoid scanning an expression like `"3rot(45)^2"` and finding `"45"` as a sub-expression.

Then we have to actually use this function when validating. All we do is recursively check that each sub-expression is a valid expression on its own. The `find_sub_expressions()` function only goes one level deep, so in an expression like `"A(B+C(D^2-E))"`, it will only find `"B+C(D^2-E)"` as a sub-expression, so recursion is needed to make the validation work properly.

There's actually a small mistake in the BNF that I didn't notice until much much later. The validation code is correct - it treats a parenthesised expression as a matrix identifier, so it can accept a multiplier on the left and an index on the right - but the BNF treats a parenthesised expression as a full matrix, which means it can't accept a multiplier or index. This doesn't affect the code, but does make the documentation slightly confusing.

And of course, I then added to the automatic unit tests to make sure this new validation worked.

```
   # 1e49c1479939b8de04751ab1a798afb2145b1550
   # tests/matrices/test_parse_and_validate_expression.py

16  expected_sub_expressions: list[tuple[str, list[str]]] = [
17      ('2(AB)^-1', ['AB']),
18      ('-3(A+B)^2-C(B^TA)^-1', ['A+B', 'B^TA']),
19      ('rot(45)', []),
20      ('()', []),
21      ('(())', ['()']),
22      ('2.3A^-1(AB)^-1+(BC)^2', ['AB', 'BC']),
23      ('(2.3A^-1(AB)^-1+(BC)^2)', ['2.3A^-1(AB)^-1+(BC)^2']),
24  ]
25
26
27  def test_find_sub_expressions() -> None:
28      """Test the :func:`lintrans.matrices.parse.find_sub_expressions` function."""
29      for inp, output in expected_sub_expressions:
30          assert find_sub_expressions(inp) == output
...
33  valid_inputs: list[str] = [
...
50      '(A)', '(AB)^-1', '2.3(3B^TA)^2', '-3.4(9D^{2}3F^-1)^T+C', '(AB)(C)',
51      '3(rot(34)^-7A)^-1+B', '3A^2B+4A(B+C)^-1D^T-A(C(D+E)B)'
52  ]
...
54  invalid_inputs: list[str] = [
...
57      '2.3AB)^T', '(AB+)', '-4.6(9A', '-2(3.4A^{-1}-C^)^2', '9.2)', '3A^2B+4A(B+C)^-1D^T-A(C(D+EB)',
...
60  ]
```

```
...
 70       expressions_and_parsed_expressions: list[tuple[str, MatrixParseList]] = [
...
105           # Parenthesized expressions
106           ('(AB)^-1', [[('', 'AB', '-1')]]),
107           ('-3(A+B)^2-C(B^TA)^-1', [[('-3', 'A+B', '2')], [('-1', 'C', ''), ('', 'B^TA', '-1')]]),
108           ('2.3(3B^TA)^2', [[('2.3', '3B^TA', '2')]]),
109           ('-3.4(9D^{2}3F^-1)^T+C', [[('-3.4', '9D^{2}3F^-1', 'T')], [('', 'C', '')]]),
110           ('2.39(3.1A^{-1}2.3B(CD)^-1)^T + (AB^T)^-1', [[('2.39', '3.1A^{-1}2.3B(CD)^-1', 'T')], [('', 'AB^T', '-1')]])
111       ]
```

I also added `@pytest.mark.xfail` to `test_parse_matrix_expression()`, since parsing should fail with the parenthesised expressions because I haven't implemented it yet.

```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *]
$ pytest -v
================================================= test session starts =================================================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml, testpaths: tests
collected 31 items

tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs0-True] PASSED                          [  3%]
tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs1-False] PASSED                         [  6%]
tests/gui/test_dialog_utility_functions.py::test_round_float PASSED                                           [  9%]
tests/matrices/test_parse_and_validate_expression.py::test_find_sub_expressions PASSED                        [ 12%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs0-True] PASSED    [ 16%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs1-False] PASSED   [ 19%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_matrix_expression XFAIL                      [ 22%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_error PASSED                                 [ 25%]
tests/matrices/test_rotation_matrices.py::test_create_rotation_matrix PASSED                                  [ 29%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_matrix_addition PASSED                 [ 32%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_two_matrix_multiplication PASSED       [ 35%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_identity_multiplication PASSED                [ 38%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_three_matrix_multiplication PASSED     [ 41%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_inverses PASSED                        [ 45%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_powers PASSED                          [ 48%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_transpose PASSED                       [ 51%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_rotation_matrices PASSED                      [ 54%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_multiplication_and_addition PASSED            [ 58%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_complicated_expressions PASSED                [ 61%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_value_errors PASSED                           [ 64%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_linalgerror PASSED                            [ 67%]
tests/matrices/matrix_wrapper/test_misc.py::test_get_expression PASSED                                        [ 70%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_get_matrix PASSED                       [ 74%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_get_name_error PASSED                         [ 77%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_set_matrix PASSED                       [ 80%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_expression PASSED                         [ 83%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_simple_dynamic_evaluation PASSED              [ 87%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_recursive_dynamic_evaluation PASSED           [ 90%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_identity_error PASSED                     [ 93%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_name_error PASSED                         [ 96%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_type_error PASSED                         [100%]

======================================== 30 passed, 1 xfailed in 0.38s ========================================
```

Figure 3.10.3: Running `pytest` with the new tests

3.10.7.2   Creating the parser class

I previously used regular expressions to parse matrix expressions. The algorithm would replace various parts of the expression to coerce it into the correct format, and then use a big RegEx at the end to find all the parts necessary for parsing. This was hard to understand and hard to maintain. It's now time to overhaul this parsing system and use a parser class to scan through the input and parse character-by-character, keeping track of the state as it goes.

```
# 736809714e0ad7579c419db03c4b5d8b8fd0c1a1
# src/lintrans/matrices/parse.py

122   @dataclass
123   class MatrixToken:
124       """A simple dataclass to hold information about a matrix token being parsed."""
125
126       multiplier: str = ''
127       identifier: str = ''
```

```
128              exponent: str = ''
129
130          @property
131          def tuple(self) -> tuple[str, str, str]:
132              """Create a tuple of the token for parsing."""
133              return self.multiplier, self.identifier, self.exponent
134
135
136      class ExpressionParser:
137          """A class to hold state during parsing."""
138
139          def __init__(self, expression: str):
140              """Create an instance of the parser with the given expression."""
141              # Remove all whitespace
142              expression = re.sub(r'\s', '', expression)
143
144              # Check if it's valid
145              if not validate_matrix_expression(expression):
146                  raise MatrixParseError('Invalid expression')
147
148              # Wrap all exponents and transposition powers with {}
149              expression = re.sub(r'(?<=\^)(-?\d+|T)(?=[^}]|$)', r'{\g<0>}', expression)
150
151              # Remove any standalone minuses
152              expression = re.sub(r'-(?=[A-Z])', '-1', expression)
153
154              # Replace subtractions with additions
155              expression = re.sub(r'-(?=\d+\.?\d*([A-Z]|rot))', '+-', expression)
156
157              # Get rid of a potential leading + introduced by the last step
158              expression = re.sub(r'^\+', '', expression)
159
160              self.expression = expression
161              self.pointer: int = 0
162
163              self.current_token: MatrixToken = MatrixToken()
164              self.current_group: list[tuple[str, str, str]] = []
165
166              self.final_list: MatrixParseList = []
167
168          def __repr__(self) -> str:
169              """Return a simple repr."""
170              return f'{self.__class__.__module__}.{self.__class__.__name__}("{self.expression}")'
171
172          @property
173          def char(self) -> str:
174              """Return the char pointed to by the pointer."""
175              return self.expression[self.pointer]
176
177          def parse(self) -> MatrixParseList:
178              """Parse the instance's matrix expression and return the MatrixParseList.
179
180              :returns MatrixParseList: The parsed expression
181              """
182              self._parse_multiplication_group()
183
184              while self.pointer < len(self.expression):
185                  if self.expression[self.pointer] != '+':
186                      raise MatrixParseError('Expected "+" between multiplication groups')
187
188                  self.pointer += 1
189                  self._parse_multiplication_group()
190
191              return self.final_list
192
193          def _parse_multiplication_group(self) -> None:
194              """Parse a group of matrices to be multiplied.
195
196              :returns bool: Success or failure
197              """
198              while self._parse_matrix():
199                  if self.pointer >= len(self.expression) or self.char == '+':
200                      self.final_list.append(self.current_group)
```

```python
201                    self.current_group = []
202                    self.pointer += 1
203
204        def _parse_matrix(self) -> bool:
205            """Parse a full matrix using :meth:`_parse_matrix_part`.
206
207            :returns bool: Success or failure
208            """
209            self.current_token = MatrixToken()
210
211            while self._parse_matrix_part():
212                pass  # The actual execution is taken care of in the loop condition
213
214            if self.current_token.identifier == '':
215                return False
216
217            self.current_group.append(self.current_token.tuple)
218            return True
219
220        def _parse_matrix_part(self) -> bool:
221            """Parse part of a matrix (multiplier, identifier, or exponent) from the expression and pointer.
222
223            .. note:: This method mutates ``self.current_token``.
224
225            :returns bool: Success or failure
226            :raises MatrixParseError: If we fail to parse this part of the token
227            """
228            if self.pointer >= len(self.expression):
229                return False
230
231            if self.char.isdigit() or self.char == '-':
232                if self.current_token.multiplier != '':
233                    return False
234
235                self._parse_multiplier()
236
237            elif self.char.isalpha() and self.char.isupper():
238                if self.current_token.identifier != '':
239                    return False
240
241                self.current_token.identifier = self.char
242                self.pointer += 1
243
244            elif self.char == 'r':
245                if self.current_token.identifier != '':
246                    return False
247
248                self._parse_rot_identifier()
249
250            elif self.char == '(':
251                if self.current_token.identifier != '':
252                    return False
253
254                self._parse_sub_expression()
255
256            elif self.char == '^':
257                if self.current_token.exponent != '':
258                    return False
259
260                self._parse_exponent()
261
262            elif self.char == '+':
263                return False
264
265            else:
266                raise MatrixParseError(f'Unrecognised character "{self.char}" in matrix expression')
267
268            return True
269
270        def _parse_multiplier(self) -> None:
271            """Parse a multiplier from the expression and pointer.
272
273            .. note:: This method mutates ``self.current_token.multiplier``.
```

```
274
275                 :raises MatrixParseError: If we fail to parse this part of the token
276                 """
277             multiplier = ''
278
279             while self.char.isdigit() or self.char in ('.', '-'):
280                 multiplier += self.char
281                 self.pointer += 1
282
283             # There can only be one dot in the multiplier
284             if len(multiplier.split('.')) > 2:
285                 raise MatrixParseError(f'Multiplier "{multiplier}" has more than one dot')
286
287             if '-' in multiplier and '-' in multiplier[1:]:
288                 raise MatrixParseError('Character "-" can only occur at the start of a multiplier')
289
290             self.current_token.multiplier = multiplier
291
292         def _parse_rot_identifier(self) -> None:
293             """Parse a ``rot()``-style identifier from the expression and pointer.
294
295             .. note:: this method mutates ``self.current_token.identifier``.
296
297             :raises MatrixParseError: If we fail to parse this part of the token
298             """
299             if match := re.match(r'rot\([^()]+\)', self.expression[self.pointer:]):
300                 self.current_token.identifier = match.group(0)
301                 self.pointer += len(match.group(0))
302             else:
303                 raise MatrixParseError(f'Invalid rot-identifier "{self.expression[self.pointer:self.pointer + 15]}..."')
304
305         def _parse_sub_expression(self) -> None:
306             """Parse a parenthesized sub-expression as the identifier, from the expression and pointer.
307
308             .. note:: this method mutates ``self.current_token.identifier``.
309
310             :raises MatrixParseError: If we fail to parse this part of the token
311             """
312             # TODO
313             raise MatrixParseError('Sub-expressions are currently not supported as identifiers')
314
315         def _parse_exponent(self) -> None:
316             """Parse a matrix exponent from the expression and pointer.
317
318             .. note:: this method mutates ``self.current_token.exponent``.
319
320             :raises MatrixParseError: If we fail to parse this part of the token
321             """
322             if match := re.match(r'\^\{(-?\d+|T)\}', self.expression[self.pointer:]):
323                 self.current_token.exponent = match.group(1)
324                 self.pointer += len(match.group(0))
325             else:
326                 raise MatrixParseError(f'Invalid exponent "{self.expression[self.pointer:self.pointer + 10]}..."')
327
328
329     def parse_matrix_expression(expression: str) -> MatrixParseList:
330         """Parse the matrix expression and return a :data:`lintrans.typing_.MatrixParseList`.
331
332         :Example:
333
334         >>> parse_matrix_expression('A')
335         [[('', 'A', '')]]
336         >>> parse_matrix_expression('-3M^2')
337         [[('-3', 'M', '2')]]
338         >>> parse_matrix_expression('1.2rot(12)^{3}2B^T')
339         [[('1.2', 'rot(12)', '3'), ('2', 'B', 'T')]]
340         >>> parse_matrix_expression('A^2 + 3B')
341         [[('', 'A', '2')], [('3', 'B', '')]]
342         >>> parse_matrix_expression('-3A^{-1}3B^T - 45M^2')
343         [[('-3', 'A', '-1'), ('3', 'B', 'T')], [('-45', 'M', '2')]]
344         >>> parse_matrix_expression('5.3A^{4} 2.6B^{-2} + 4.6D^T 8.9E^{-1}')
345         [[('5.3', 'A', '4'), ('2.6', 'B', '-2')], [('4.6', 'D', 'T'), ('8.9', 'E', '-1')]]
346
```

```
347            :param str expression: The expression to be parsed
348            :returns: A list of parsed components
349            :rtype: :data:`lintrans.typing_.MatrixParseList`
350            """
351            return ExpressionParser(expression).parse()
```

The `MatrixToken` class just holds data about a matrix multiplier, identifier, and exponent. It's just a simple way to package up this data into a single dataclass to keep track of which matrices we've parsed.

The `ExpressionParser` class works like so: when you create a new instance, you give it an expression to parse and it cleans up the expression by removing whitespace, making sure it's valid, wrapping exponents with `{}`, etc. Most of these replacement steps are taken from the previous RegEx-based parser. It then sets up some internal state to keep track of the expression, the pointer, the token that it's currently parsing, the group that it's currently parsing, and its progress so far.

When you've created a new instance, you should call the public `parse()` method, which will return the parsed expression at the end, in the same format that the old parser used. The `parse_matrix⌋ _expression()` function is a wrapper that just creates an `ExpressionParser` and calls `parse()`.

The `parse()` method itself just delegates to `_parse_multiplication_group()`. A multiplication group is a group of matrices that will be multiplied together in a particular order. An expression should start with a multiplication group, and may have extra ones separated with plus signs. When all the multiplication groups have been parsed, it returns the final parse list.

`_parse_multiplication_group()` tries to parse matrices until it reaches the end of the expression or it finds a plus sign, at which point it appends the current group to the final list and returns.

`_parse_matrix()` returns a boolean to indicate success. It just attempts to parse a 'matrix part' (multiplier, identifier, or exponent) until that fails, and then ensures that the resultant matrix token has a valid identifier. It then appends the matrix token to the current group.

`_parse_matrix_part()` is where most of the selection happens. This method has to determine if it's looking at a multiplier, which would start with a digit or a minus sign; a normal identifier, which would start with a capital letter; a rotation identifier, which would start with a lowercase letter r; a sub-expression, which would start with an open bracket; or an exponent, with will start with a caret. If it encounters a plus sign, then that's a soft error, so it can return **False**, but if it encounters something it doesn't recognise, then that's a hard error.

`_parse_multiplier()` just parses a real number into `self.current_token.multiplier`.

`_parse_rot_identifier()` parses a `rot()` command with a real number angle[24].

`_parse_sub_expression()` hasn't been implemented yet, but it will eventually parse sub-expressions, hence the name.

`_parse_exponent()` parses a caret, and then a balanced pair of braces, containing an integer or a capital letter T.

And of course, this new parser should still pass all the unit tests for the previous parser, which it does.

---

[24]It doesn't need to parse the real number itself to make sure it's valid, since the expression validator already ensured that it was a valid real number.

```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *]
$ pytest -v
=============================== test session starts ===============================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml, testpaths: tests
collected 31 items

tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs0-True] PASSED         [  3%]
tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs1-False] PASSED        [  6%]
tests/gui/test_dialog_utility_functions.py::test_round_float PASSED                          [  9%]
tests/matrices/test_parse_and_validate_expression.py::test_find_sub_expressions PASSED       [ 12%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs0-True] PASSED  [ 16%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs1-False] PASSED [ 19%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_matrix_expression XFAIL     [ 22%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_error PASSED                [ 25%]
tests/matrices/test_rotation_matrices.py::test_create_rotation_matrix PASSED                 [ 29%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_matrix_addition PASSED [ 32%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_two_matrix_multiplication PASSED [ 35%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_identity_multiplication PASSED [ 38%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_three_matrix_multiplication PASSED [ 41%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_inverses PASSED       [ 45%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_powers PASSED         [ 48%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_transpose PASSED      [ 51%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_rotation_matrices PASSED     [ 54%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_multiplication_and_addition PASSED [ 58%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_complicated_expressions PASSED [ 61%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_value_errors PASSED          [ 64%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_linalgerror PASSED           [ 67%]
tests/matrices/matrix_wrapper/test_misc.py::test_get_expression PASSED                       [ 70%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_get_matrix PASSED      [ 74%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_get_name_error PASSED        [ 77%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_set_matrix PASSED      [ 80%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_expression PASSED        [ 83%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_simple_dynamic_evaluation PASSED [ 87%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_recursive_dynamic_evaluation PASSED [ 90%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_identity_error PASSED    [ 93%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_name_error PASSED        [ 96%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_type_error PASSED        [100%]

=============================== 30 passed, 1 xfailed in 0.38s ===============================
```

Figure 3.10.4: Running `pytest` with the new tests

3.10.7.3   Implementing sub-expression parsing

Of course, the next step was to actually implement sub-expression parsing. To do this, I used a very similar method to finding the sub-expressions, where I kept track of the depth of the parentheses. Except with this method, we only wanted the first sub-expression starting at the pointer. We can then assign this whole sub-expression to `self.current_token.identifier` at the end.

Much like when finding sub-expressions, this parser method only goes one level deep, but the `MatrixWrapper` will use recursion when evaluating the parsed expressions, so it will handle nested sub-expressions with.

```python
# 493f8cf3fb658408466f099f978d276fc2262243
# src/lintrans/matrices/parse.py

136    class ExpressionParser:
...
305        def _parse_sub_expression(self) -> None:
306            """Parse a parenthesized sub-expression as the identifier, from the expression and pointer.
307
308            .. note:: this method mutates ``self.current_token.identifier``.
309
310            :raises MatrixParseError: If we fail to parse this part of the token
311            """
312            if self.char != '(':
313                raise MatrixParseError('Sub-expression must start with "("')
314
315            self.pointer += 1
316            paren_depth = 1
317            identifier = ''
318
319            while paren_depth > 0:
320                if self.char == '(':
321                    paren_depth += 1
322                elif self.char == ')':
```

```
323                        paren_depth -= 1
324
325                    if paren_depth == 0:
326                        self.pointer += 1
327                        break
328
329                    identifier += self.char
330                    self.pointer += 1
331
332            self.current_token.identifier = identifier
```

3.10.7.4   Fixing little bugs

Since I've now implemented sub-expression parsing, I should be able to just remove the `@pytest.mark.xfail`
line from before and all the tests should pass with no problem.

```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *=]
$ pytest -v
============================================================ test session starts ============================================================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml, testpaths: tests
collected 31 items

tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs0-True] PASSED                                                [  3%]
tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs1-False] PASSED                                               [  6%]
tests/gui/test_dialog_utility_functions.py::test_round_float PASSED                                                                 [  9%]
tests/matrices/test_parse_and_validate_expression.py::test_find_sub_expressions PASSED                                              [ 12%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs0-True] PASSED                          [ 16%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs1-False] PASSED                         [ 19%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_matrix_expression FAILED                                          [ 22%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_error PASSED                                                       [ 25%]
tests/matrices/test_rotation_matrices.py::test_create_rotation_matrix PASSED                                                        [ 29%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_matrix_addition PASSED                                       [ 32%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_two_matrix_multiplication PASSED                             [ 35%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_identity_multiplication PASSED                                      [ 38%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_three_matrix_multiplication PASSED                           [ 41%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_inverses PASSED                                             [ 45%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_powers PASSED                                                [ 48%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_transpose PASSED                                             [ 51%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_rotation_matrices PASSED                                            [ 54%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_multiplication_and_addition PASSED                                  [ 58%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_complicated_expressions PASSED                                      [ 61%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_value_errors PASSED                                                 [ 64%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_linalgerror PASSED                                                  [ 67%]
tests/matrices/matrix_wrapper/test_misc.py::test_get_expression PASSED                                                              [ 70%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_get_matrix PASSED                                             [ 74%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_get_name_error PASSED                                               [ 77%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_set_matrix PASSED                                             [ 80%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_expression PASSED                                               [ 83%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_simple_dynamic_evaluation PASSED                                    [ 87%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_recursive_dynamic_evaluation PASSED                                 [ 90%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_identity_error PASSED                                           [ 93%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_name_error PASSED                                               [ 96%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_type_error PASSED                                               [100%]

================================================================= FAILURES =================================================================
_____ test_parse_matrix_expression _____

    def test_parse_matrix_expression() -> None:
        """Test the parse_matrix_expression() function."""
        for expression, parsed_expression in expressions_and_parsed_expressions:
            # Test it with and without whitespace
>           assert parse_matrix_expression(expression) == parsed_expression

tests/matrices/test_parse_and_validate_expression.py:119:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
src/lintrans/matrices/parse.py:370: in parse_matrix_expression
    return ExpressionParser(expression).parse()
src/lintrans/matrices/parse.py:182: in parse
    self._parse_multiplication_group()
src/lintrans/matrices/parse.py:198: in _parse_multiplication_group
    while self._parse_matrix():
src/lintrans/matrices/parse.py:211: in _parse_matrix
    while self._parse_matrix_part():
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = lintrans.matrices.parse.ExpressionParser(".1A")

    def _parse_matrix_part(self) -> bool:
        """Parse part of a matrix (multiplier, identifier, or exponent) from the expression and pointer.

        .. note:: This method mutates ``self.current_token``.

        :returns bool: Success or failure
        :raises MatrixParseError: If we fail to parse this part of the token
        """
        if self.pointer >= len(self.expression):
            return False

        if self.char.isdigit() or self.char == '-':
            if self.current_token.multiplier != '':
                return False

            self._parse_multiplier()

        elif self.char.isalpha() and self.char.isupper():
            if self.current_token.identifier != '':
                return False

            self.current_token.identifier = self.char
            self.pointer += 1

        elif self.char == 'r':
            if self.current_token.identifier != '':
                return False

            self._parse_rot_identifier()

        elif self.char == '(':
            if self.current_token.identifier != '':
                return False

            self._parse_sub_expression()

        elif self.char == '^':
            if self.current_token.exponent != '':
                return False

            self._parse_exponent()

        elif self.char == '+':
            return False

        else:
>           raise MatrixParseError(f'Unrecognised character "{self.char}" in matrix expression')
E           lintrans.matrices.parse.MatrixParseError: Unrecognised character "." in matrix expression

src/lintrans/matrices/parse.py:266: MatrixParseError
========================================================= short test summary info =========================================================
FAILED tests/matrices/test_parse_and_validate_expression.py::test_parse_matrix_expression - lintrans.matrices.parse.MatrixParseError: Unrecognised character "." in matrix expression
===================================================== 1 failed, 30 passed in 0.41s =====================================================
```

Figure 3.10.5: One of the tests failing

Ah. That's not good. So what went wrong? Well, we can see from the output that `test_parse_matrix_expression()` is the one that failed. This test iterates over a whole list, so which element broke it? Well, we can see from the line '`self = lintrans.matrices.parse.ExpressionParser(".1A")`' that it was trying to parse the expression `".1A"`. This doesn't work, not because `ExpressionParser._parse_multiplier()` doesn't understand this type of multiplier, but because `ExpressionParser._parse_matrix_part()` doesn't recognise a dot as the start of a multiplier, so never calls `_parse_multiplier()`.

This would be a very easy fix, but this error got me thinking about these multipliers and I decided to just forbid them. They can create confusion; `"0.1A"` is easier to understand than `".1A"`. Implementing this change just meant removing this type of multiplier from the tests.

Once I'd done that, the parser should work fine, right?

Figure 3.10.6: Another test failing

Well, that's strange. It was trying to parse the expression `"A 0.1B"` but it thought the `0.1` was part of the `A`. Why did that happen? Well, it's because of line 232 in this snippet:

```
# 9b1c69926c225574161d32dcbecd86055edb4065
# src/lintrans/matrices/parse.py

136    class ExpressionParser:
...
220        def _parse_matrix_part(self) -> bool:
...
231            if self.char.isdigit() or self.char == '-':
232                if self.current_token.multiplier != '':
233                    return False
234
235            self._parse_multiplier()
```

The parser parsed `"A"` as the matrix identifier and then encountered `"0"`. Since the current token didn't have a multiplier, it started parsing a multiplier for the matrix **A**. To fix this, we can just fail to parse a matrix part if we encounter the start of a multiplier when the current token already has an identifier. Failing to parse a matrix part here means that the parser will start parsing a new matrix.

```
# 8d7143fc33ea7bd4199e0f01b6a5308dfcf03ff9
# src/lintrans/matrices/parse.py

136    class ExpressionParser:
...
```

```
220        def _parse_matrix_part(self) -> bool:
...
231            if self.char.isdigit() or self.char == '-':
232                if self.current_token.multiplier != '' \
233                        or (self.current_token.multiplier == '' and self.current_token.identifier != ''):
234                    return False
235
236            self._parse_multiplier()
```

That should fix the tests, right?

Figure 3.10.7: *Another* test failing

I'm getting tired of this now. What's broken this time? Well, the parser processes its input to remove whitespace and add braces around exponents before any parsing takes place. This includes processing the contents of sub-expressions. That was never a problem before, since parsing wasn't affected by whitespace or braces, but now those changes propagate into the sub-expressions. I wrote the sub-expression parsing tests in §3.10.7.1 by using my literal expression input in the parsed sub-expressions. So to fix this error, I just had to account for the syntax transformations in the test input.

Now the tests should all pass, right?

```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *=]
$ pytest -v
============================================================= test session starts =============================================================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml, testpaths: tests
collected 31 items

tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs0-True] PASSED                                                    [  3%]
tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs1-False] PASSED                                                   [  6%]
tests/gui/test_dialog_utility_functions.py::test_round_float PASSED                                                                     [  9%]
tests/matrices/test_parse_and_validate_expression.py::test_find_sub_expressions PASSED                                                  [ 12%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs0-True] PASSED                              [ 16%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs1-False] PASSED                             [ 19%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_matrix_expression PASSED                                               [ 22%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_error PASSED                                                           [ 25%]
tests/matrices/test_create_rotation_matrices.py::test_create_rotation_matrix PASSED                                                     [ 29%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_matrix_addition PASSED                                           [ 32%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_two_matrix_multiplication PASSED                                 [ 35%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_identity_multiplication PASSED                                          [ 38%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_three_matrix_multiplication PASSED                               [ 41%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_inverses PASSED                                                  [ 45%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_powers PASSED                                                    [ 48%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_transpose PASSED                                                 [ 51%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_rotation_matrices PASSED                                                [ 54%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_multiplication_and_addition PASSED                                      [ 58%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_complicated_expressions PASSED                                          [ 61%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_value_errors PASSED                                                     [ 64%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_linalgerror PASSED                                                      [ 67%]
tests/matrices/matrix_wrapper/test_misc.py::test_get_expression PASSED                                                                  [ 70%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_get_matrix PASSED                                                 [ 74%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_get_name_error PASSED                                                   [ 77%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_set_matrix PASSED                                                 [ 80%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_expression PASSED                                                   [ 83%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_simple_dynamic_evaluation PASSED                                        [ 87%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_recursive_dynamic_evaluation PASSED                                     [ 90%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_identity_error PASSED                                               [ 93%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_name_error PASSED                                                   [ 96%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_type_error PASSED                                                   [100%]

============================================================= 31 passed in 0.34s =============================================================
```

Figure 3.10.8: All the tests finally passing

Thank god.

3.10.7.5   Making recursive evaluation work

When `MatrixWrapper.__getitem__()` tries to retrieve a matrix which is defined in terms of an expression, it has to call `MatrixWrapper.evaluate_expression()`. This method in turn calls `__getitem__()` again to evaluate the matrices used in the expression. Now that sub-expressions exist, `__getitem__()` might get called with a sub-expression identifier, so it needs to be able to evaluate these expressions. To do this, we can simply add another call to `evaluate_expression()`, as seen on lines 120-121.

```python
# ba1ee72fcafd5129aed76992f743e5966f5c199d
# src/lintrans/matrices/wrapper.py

23    class MatrixWrapper:
...
100        def __getitem__(self, name: str) -> Optional[MatrixType]:
101            """Get the matrix with the given name.
102
103            If it is a simple name, it will just be fetched from the dictionary. If the name is ``rot(x)``, with
104            a given angle in degrees, then we return a new matrix representing a rotation by that angle.
105
106            .. note::
107               If the named matrix is defined as an expression, then this method will return its evaluation.
108               If you want the expression itself, use :meth:`get_expression`.
109
110            :param str name: The name of the matrix to get
111            :returns Optional[MatrixType]: The value of the matrix (could be None)
112
113            :raises NameError: If there is no matrix with the given name
114            """
115            # Return a new rotation matrix
116            if (match := re.match(r'rot\((-?\d*\.?\d*)\)', name)) is not None:
117                return create_rotation_matrix(float(match.group(1)))
118
119            if name not in self._matrices:
120                if validate_matrix_expression(name):
121                    return self.evaluate_expression(name)
122
123                raise NameError(f'Unrecognised matrix name "{name}"')
124
```

```
125                # We copy the matrix before we return it so the user can't accidentally mutate the matrix
126                matrix = copy(self._matrices[name])
127
128                if isinstance(matrix, str):
129                    return self.evaluate_expression(matrix)
130
131                return matrix
...
214        def evaluate_expression(self, expression: str) -> MatrixType:
215            """Evaluate a given expression and return the matrix evaluation.
216
217            :param str expression: The expression to be parsed
218            :returns MatrixType: The matrix result of the expression
219
220            :raises ValueError: If the expression is invalid
221            """
222            if not self.is_valid_expression(expression):
223                raise ValueError('The expression is invalid')
224
225            parsed_result = parse_matrix_expression(expression)
226            final_groups: list[list[MatrixType]] = []
227
228            for group in parsed_result:
229                f_group: list[MatrixType] = []
230
231                for multiplier, identifier, index in group:
232                    if index == 'T':
233                        m = self[identifier]
234
235                        # This assertion is just so mypy doesn't complain
236                        # We know this won't be None, because we know that this matrix is defined in this wrapper
237                        assert m is not None
238                        matrix_value = m.T
239
240                    else:
241                        matrix_value = np.linalg.matrix_power(self[identifier], 1 if index == '' else int(index))
242
243                    matrix_value *= 1 if multiplier == '' else float(multiplier)
244                    f_group.append(matrix_value)
245
246                final_groups.append(f_group)
247
248            return reduce(add, [reduce(matmul, group) for group in final_groups])
```

I then added a new unit test function to make sure parenthesised expressions get evaluated properly.

```
    # ba1ee72fcafd5129aed76992f743e5966f5c199d
    # tests/matrices/matrix_wrapper/test_evaluate_expression.py

229   def test_parenthesized_expressions(test_wrapper: MatrixWrapper) -> None:
230       """Test evaluation of parenthesized expressions."""
231       assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
232               test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
233               test_wrapper['G'] is not None
234
235       assert (test_wrapper.evaluate_expression('(A^T)^2') == la.matrix_power(test_wrapper['A'].T, 2)).all()
236       assert (test_wrapper.evaluate_expression('(B^T)^3') == la.matrix_power(test_wrapper['B'].T, 3)).all()
237       assert (test_wrapper.evaluate_expression('(C^T)^4') == la.matrix_power(test_wrapper['C'].T, 4)).all()
238       assert (test_wrapper.evaluate_expression('(D^T)^5') == la.matrix_power(test_wrapper['D'].T, 5)).all()
239       assert (test_wrapper.evaluate_expression('(E^T)^6') == la.matrix_power(test_wrapper['E'].T, 6)).all()
240       assert (test_wrapper.evaluate_expression('(F^T)^7') == la.matrix_power(test_wrapper['F'].T, 7)).all()
241       assert (test_wrapper.evaluate_expression('(G^T)^8') == la.matrix_power(test_wrapper['G'].T, 8)).all()
242
243       assert (test_wrapper.evaluate_expression('D^3(A+6.2F-0.397G^TE)^-2+A') ==
244               la.matrix_power(test_wrapper['D'], 3) @ la.matrix_power(
245                   test_wrapper['A'] + 6.2 * test_wrapper['F'] - 0.397 * test_wrapper['G'].T @ test_wrapper['E'],
246                   -2
247               ) + test_wrapper['A']).all()
248
249       assert (test_wrapper.evaluate_expression('-1.2F^{3}4.9D^T(A^2(B+3E^TF)^-1)^2') ==
250               -1.2 * la.matrix_power(test_wrapper['F'], 3) @ (4.9 * test_wrapper['D'].T) @
```

```
251                     la.matrix_power(
252                         la.matrix_power(test_wrapper['A'], 2) @ la.matrix_power(
253                             test_wrapper['B'] + 3 * test_wrapper['E'].T @ test_wrapper['F'],
254                             -1
255                         ),
256                         2
257                     )).all()
```

```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *=]
$ pytest -v
============================================================= test session starts =============================================================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml, testpaths: tests
collected 32 items

tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs0-True] PASSED                                                    [  3%]
tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs1-False] PASSED                                                   [  6%]
tests/gui/test_dialog_utility_functions.py::test_round_float PASSED                                                                     [  9%]
tests/matrices/test_parse_and_validate_expression.py::test_find_sub_expressions PASSED                                                  [ 12%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs0-True] PASSED                              [ 15%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs1-False] PASSED                             [ 18%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_matrix_expression PASSED                                               [ 21%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_error PASSED                                                           [ 25%]
tests/matrices/test_rotation_matrices.py::test_create_rotation_matrix PASSED                                                            [ 28%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_matrix_addition PASSED                                           [ 31%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_two_matrix_multiplication PASSED                                 [ 34%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_identity_multiplication PASSED                                          [ 37%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_three_matrix_multiplication PASSED                               [ 40%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_inverses PASSED                                                  [ 43%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_powers PASSED                                                    [ 46%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_transpose PASSED                                                 [ 50%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_rotation_matrices PASSED                                                [ 53%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_multiplication_and_addition PASSED                                      [ 56%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_complicated_expressions PASSED                                          [ 59%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_parenthesized_expressions PASSED                                        [ 62%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_value_errors PASSED                                                     [ 65%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_linalgerror PASSED                                                      [ 68%]
tests/matrices/matrix_wrapper/test_misc.py::test_get_expression PASSED                                                                  [ 71%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_get_matrix PASSED                                                 [ 75%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_get_name_error PASSED                                                   [ 78%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_set_matrix PASSED                                                 [ 81%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_expression PASSED                                                   [ 84%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_simple_dynamic_evaluation PASSED                                        [ 87%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_recursive_dynamic_evaluation PASSED                                     [ 90%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_identity_error PASSED                                               [ 93%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_name_error PASSED                                                   [ 96%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_type_error PASSED                                                   [100%]

============================================================== 32 passed in 0.36s ==============================================================
```

Figure 3.10.9: Running `pytest` with the new tests

I also improved the doctests for `parse_matrix_expression()` by adding a case for these new parenthesised expressions, and I added a doctest for the expression parser.

```
# e3d633133f293fc1308d6fc0c60a1febaf67dda0
# src/lintrans/matrices/parse.py

349    def parse_matrix_expression(expression: str) -> MatrixParseList:
350        """Parse the matrix expression and return a :data:`lintrans.typing_.MatrixParseList`.
351
352        :Example:
353
354        >>> parse_matrix_expression('A')
355        [[('', 'A', '')]]
356        >>> parse_matrix_expression('-3M^2')
357        [[('-3', 'M', '2')]]
358        >>> parse_matrix_expression('1.2rot(12)^{3}2B^T')
359        [[('1.2', 'rot(12)', '3'), ('2', 'B', 'T')]]
360        >>> parse_matrix_expression('A^2 + 3B')
361        [[('', 'A', '2')], [('3', 'B', '')]]
362        >>> parse_matrix_expression('-3A^{-1}3B^T - 45M^2')
363        [[('-3', 'A', '-1'), ('3', 'B', 'T')], [('-45', 'M', '2')]]
364        >>> parse_matrix_expression('5.3A^{4} 2.6B^{-2} + 4.6D^T 8.9E^{-1}')
365        [[('5.3', 'A', '4'), ('2.6', 'B', '-2')], [('4.6', 'D', 'T'), ('8.9', 'E', '-1')]]
366        >>> parse_matrix_expression('2(A+B^TC)^2D')
367        [[('2', 'A+B^{T}C', '2'), ('', 'D', '')]]
368
369        :param str expression: The expression to be parsed
370        :returns: A list of parsed components
371        :rtype: :data:`lintrans.typing_.MatrixParseList`
372        """
373        return ExpressionParser(expression).parse()


# bf1f5a1eaac60d23964a44c9a6b8235c5e0fcab2
# src/lintrans/matrices/parse.py
```

```
136   class ExpressionParser:
137       """A class to hold state during parsing.
138
139       Most of the methods in this class are class-internal and should not be used from outside.
140
141       This class should be used like this:
142
143       >>> ExpressionParser('3A^-1B').parse()
144       [[('3', 'A', '-1'), ('', 'B', '')]]
145       >>> ExpressionParser('4(M^TA^2)^-2').parse()
146       [[('4', 'M^{T}A^{2}', '-2')]]
147       """
```

```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *=]
$ pytest --doctest-modules src/ -v
==================================== test session starts =====================================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/py
thon
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml
collected 4 items

src/lintrans/matrices/parse.py::lintrans.matrices.parse.ExpressionParser PASSED          [ 25%]
src/lintrans/matrices/parse.py::lintrans.matrices.parse.parse_matrix_expression PASSED   [ 50%]
src/lintrans/matrices/wrapper.py::lintrans.matrices.wrapper.MatrixWrapper PASSED          [ 75%]
src/lintrans/matrices/wrapper.py::lintrans.matrices.wrapper.create_rotation_matrix PASSED    [100%]

===================================== 4 passed in 0.33s ======================================
```

Figure 3.10.10: Running `pytest` with the new and improved doctests

And of course, I updated the changelog:

```
<!-- 9e26079475e33f3ab2e02e76ea59a66217d18f76 -->
<!-- CHANGELOG.md -->

12    ### Added
...
16    - Support parenthesized sub-expressions as matrix identifiers
```

3.10.7.6   Ensuring numerical formats

The validation checks the format of the numbers in multipliers, exponents and `rot()` commands, but it would be quite good to also validate the format of these numbers when parsing, just to make absolutely sure that they will be able to be correctly evaluated. To do this, we can simply perform the conversion in a try/except block. It would also be good to validate the sub-expressions.

```
# 761c6ed255ed65c3396c6a551acfa60bb7485a17
# src/lintrans/matrices/parse.py

136   class ExpressionParser:
...
282       def _parse_multiplier(self) -> None:
283           """Parse a multiplier from the expression and pointer.
284
285           .. note:: This method mutates ``self.current_token.multiplier``.
286
287           :raises MatrixParseError: If we fail to parse this part of the token
288           """
289           multiplier = ''
290
291           while self.char.isdigit() or self.char in ('.', '-'):
292               multiplier += self.char
293               self.pointer += 1
```

```
294
295            try:
296                float(multiplier)
297            except ValueError as e:
298                raise MatrixParseError(f'Invalid multiplier "{multiplier}"') from e
299
300            self.current_token.multiplier = multiplier
301
302        def _parse_rot_identifier(self) -> None:
303            """Parse a ``rot()``-style identifier from the expression and pointer.
304
305            .. note:: This method mutates ``self.current_token.identifier``.
306
307            :raises MatrixParseError: If we fail to parse this part of the token
308            """
309            if match := re.match(r'rot\(([\d.-]+)\)', self.expression[self.pointer:]):
310                # Ensure that the number in brackets is a valid float
311                try:
312                    float(match.group(1))
313                except ValueError as e:
314                    raise MatrixParseError(f'Invalid angle number "{match.group(1)}" in rot-identifier') from e
315
316                self.current_token.identifier = match.group(0)
317                self.pointer += len(match.group(0))
318            else:
319                raise MatrixParseError(f'Invalid rot-identifier "{self.expression[self.pointer:self.pointer + 15]}..."')
320
321        def _parse_sub_expression(self) -> None:
322            """Parse a parenthesized sub-expression as the identifier, from the expression and pointer.
323
324            .. note:: This method mutates ``self.current_token.identifier``.
325
326            :raises MatrixParseError: If we fail to parse this part of the token
327            """
328            if self.char != '(':
329                raise MatrixParseError('Sub-expression must start with "("')
330
331            self.pointer += 1
332            paren_depth = 1
333            identifier = ''
334
335            while paren_depth > 0:
336                if self.char == '(':
337                    paren_depth += 1
338                elif self.char == ')':
339                    paren_depth -= 1
340
341                    if paren_depth == 0:
342                        self.pointer += 1
343                        break
344
345                identifier += self.char
346                self.pointer += 1
347
348            if not validate_matrix_expression(identifier):
349                raise MatrixParseError(f'Invalid sub-expression identifier "{identifier}"')
350
351            self.current_token.identifier = identifier
352
353        def _parse_exponent(self) -> None:
354            """Parse a matrix exponent from the expression and pointer.
355
356            .. note:: This method mutates ``self.current_token.exponent``.
357
358            :raises MatrixParseError: If we fail to parse this part of the token
359            """
360            if match := re.match(r'\^\{(-?\d+|T)\}', self.expression[self.pointer:]):
361                exponent = match.group(1)
362
363                try:
364                    if exponent != 'T':
365                        int(exponent)
366                except ValueError as e:
```

```
367                         raise MatrixParseError(f'Invalid exponent "{match.group(1)}"') from e
368
369                 self.current_token.exponent = exponent
370                 self.pointer += len(match.group(0))
371             else:
372                 raise MatrixParseError(f'Invalid exponent "{self.expression[self.pointer:self.pointer + 10]}..."')
```

3.10.8   Fixing premature `rot()` evaluation

Now that parenthesised expressions are done, I can focus on other improvements.

When playing around with this new feature, I discovered a bug where an expression like `"rot(45)^2"` would be evaluated in the same way as `"rot(45)"`. After lots of confusion and debugging, I discovered that this bug was in `MatrixWrapper.__getitem__()` when it tries to evaluate `rot()` commands. It does this by eagerly searching for the `rot()` RegEx in the string. This was never an issue before, but now that this method might get called with a whole expression, this RegEx can trigger too early. If it's given an expression which contains a `rot()` command anywhere in it, then it will return a rotation matrix immediately. To fix this, we just have to ensure that the `rot()` command takes up the whole string before returning a rotation matrix. We can do this by anchoring the RegEx with `^` and `$`.

```
      # c23d540aa3b8df91b9478fb113cb42df4c7d5e42
      # src/lintrans/matrices/wrapper.py

 23   class MatrixWrapper:
...
100       def __getitem__(self, name: str) -> Optional[MatrixType]:
...
115           # Return a new rotation matrix
116           if (match := re.match(r'^rot\((-?\d*\.?\d*)\)$', name)) is not None:
117               return create_rotation_matrix(float(match.group(1)))
118
119           if name not in self._matrices:
120               if validate_matrix_expression(name):
121                   return self.evaluate_expression(name)
122
123               raise NameError(f'Unrecognised matrix name "{name}"')
124
125           # We copy the matrix before we return it so the user can't accidentally mutate the matrix
126           matrix = copy(self._matrices[name])
127
128           if isinstance(matrix, str):
129               return self.evaluate_expression(matrix)
130
131           return matrix
```

I also added cases to the parenthesised expressions test to cover this change.

```
      # c23d540aa3b8df91b9478fb113cb42df4c7d5e42
      # tests/matrices/matrix_wrapper/test_evaluate_expression.py

229   def test_parenthesized_expressions(test_wrapper: MatrixWrapper) -> None:
...
243       assert (test_wrapper.evaluate_expression('(rot(45)^1)^T') == create_rotation_matrix(45).T).all()
244       assert (test_wrapper.evaluate_expression('(rot(45)^2)^T') == la.matrix_power(create_rotation_matrix(45),
          ↪  2).T).all()
245       assert (test_wrapper.evaluate_expression('(rot(45)^3)^T') == la.matrix_power(create_rotation_matrix(45),
          ↪  3).T).all()
246       assert (test_wrapper.evaluate_expression('(rot(45)^4)^T') == la.matrix_power(create_rotation_matrix(45),
          ↪  4).T).all()
247       assert (test_wrapper.evaluate_expression('(rot(45)^5)^T') == la.matrix_power(create_rotation_matrix(45),
          ↪  5).T).all()
```

And then added it to the changelog.

```
<!-- c23d540aa3b8df91b9478fb113cb42df4c7d5e42 -->
<!-- CHANGELOG.md -->
```

```
18    ### Fixed
...
21    - Fixed bug with premature rot evaluation in sub-expressions
```

3.10.9   Animating rotations

When animating a rotation like `"rot(170)"`, it animates a rotation of 170°, but it doesn't look very good. I fixed an old rotation issue in §3.3.6, but the speed is still a problem.

The speed of rotation is significantly higher during the middle of the rotation, which makes it look unnatural. I want to fix this to make rotations look more natural. Ideally, the program would detect a rotation and animate it in a different way, by moving the tips of the basis vectors along the edge of a circle with the radius equal to the length of the basis vectors. That shouldn't be too hard. However, I'd also like to animate scaled rotations like `"2rot(170)"`. Animating in a circle won't work for this type of transformation, so I'll need to use a spiral.

This is a big feature, so I split it into a `dev/anim-rot` branch.

3.10.9.1   Factoring out animation frames

The `LintransMainWindow.animate_expression()` method obviously handles animating expressions. It does this by calling a separate method called `LintransMainWindow.animate_between_matrices()` after parsing and evaluating the expression. The first step of improving my rotational animation code is to factor out the generation of individual frames when animating between matrices. This separate method will make it easier to work on the new feature.

Here's the old code, including the `animate_expression()` method, which wasn't changed:

```
      # 751f185a2b64d31fe7ddd53aafab110008cff50b
      # src/lintrans/gui/main_window.py

36    class LintransMainWindow(QMainWindow):
...
376       def animate_between_matrices(self, matrix_start: MatrixType, matrix_target: MatrixType, steps: int = 100) ->
          ↪     None:
377           """Animate from the start matrix to the target matrix."""
378           det_target = linalg.det(matrix_target)
379           det_start = linalg.det(matrix_start)
380
381           self.animating = True
382
383           for i in range(0, steps + 1):
384               if not self.animating:
385                   break
386
387               # This proportion is how far we are through the loop
388               proportion = i / steps
389
390               # matrix_a is the start matrix plus some part of the target, scaled by the proportion
391               # If we just used matrix_a, then things would animate, but the determinants would be weird
392               matrix_a = matrix_start + proportion * (matrix_target - matrix_start)
393
394               if self.plot.display_settings.smoothen_determinant and det_start * det_target > 0:
395                   # To fix the determinant problem, we get the determinant of matrix_a and use it to normalize
396                   det_a = linalg.det(matrix_a)
397
398                   # For a 2x2 matrix A and a scalar c, we know that det(cA) = c^2 det(A)
399                   # We want B = cA such that det(B) = det(S), where S is the start matrix,
400                   # so then we can scale it with the animation, so we get
401                   # det(cA) = c^2 det(A) = det(S) => c = sqrt(abs(det(S) / det(A)))
```

```
402                          # Then we scale A to get the determinant we want, and call that matrix_b
403                          if det_a == 0:
404                              c = 0
405                          else:
406                              c = np.sqrt(abs(det_start / det_a))
407
408                          matrix_b = c * matrix_a
409                          det_b = linalg.det(matrix_b)
410
411                          # matrix_to_render is the final matrix that we then render for this frame
412                          # It's B, but we scale it over time to have the target determinant
413
414                          # We want some C = dB such that det(C) is some target determinant T
415                          # det(dB) = d^2 det(B) = T => d = sqrt(abs(T / det(B))
416
417                          # We're also subtracting 1 and multiplying by the proportion and then adding one
418                          # This just scales the determinant along with the animation
419
420                          # That is all of course, if we can do that
421                          # We'll crash if we try to do this with det(B) == 0
422                          if det_b != 0:
423                              scalar = 1 + proportion * (np.sqrt(abs(det_target / det_b)) - 1)
424                              matrix_to_render = scalar * matrix_b
425
426                          else:
427                              matrix_to_render = matrix_a
428
429                      else:
430                          matrix_to_render = matrix_a
431
432                  if self.is_matrix_too_big(matrix_to_render):
433                      self.show_error_message('Matrix too big', "This matrix doesn't fit on the canvas")
434                      return
435
436                  self.plot.visualize_matrix_transformation(matrix_to_render)
437
438                  # We schedule the plot to be updated, tell the event loop to
439                  # process events, and asynchronously sleep for 10ms
440                  # This allows for other events to be processed while animating, like zooming in and out
441                  self.plot.update()
442                  QApplication.processEvents()
443                  QThread.msleep(1000 // steps)
444
445          self.animating = False
```

And here's the new code, not including the unchanged `animate_expression()` method:

```
     # d47bb0165304d496caf7bbe3e09e4bbd12e45453
     # src/lintrans/gui/main_window.py

 36  class LintransMainWindow(QMainWindow):
...
376      def _get_animation_frame(self, start: MatrixType, target: MatrixType, proportion: float) -> MatrixType:
377          """Get the matrix to render for this frame of the animation.
378
379          This method will smoothen the determinant if that setting in enabled and if the determinant is positive.
380          It also animates rotation-like matrices using a logarithmic spiral to rotate around and scale continuously.
381          Essentially, it just makes things look good when animating.
382
383          :param MatrixType start: The starting matrix
384          :param MatrixType start: The target matrix
385          :param float proportion: How far we are through the loop
386          """
387          det_target = linalg.det(target)
388          det_start = linalg.det(start)
389
390          # This is the matrix that we're applying to get from start to target
391          # We want to check if it's rotation-like
392          matrix_application = target @ linalg.inv(start)
393
394          if linalg.det(matrix_application) > 0 and abs(np.dot(matrix_application.T[0], matrix_application.T[1])) <
             ↪  0.1:
```

```
395                    # TODO: Use logarithmic spiral here and return
396                    pass
397
398            # matrix_a is the start matrix plus some part of the target, scaled by the proportion
399            # If we just used matrix_a, then things would animate, but the determinants would be weird
400            matrix_a = start + proportion * (target - start)
401
402            if not self.plot.display_settings.smoothen_determinant or det_start * det_target <= 0:
403                return matrix_a
404
405            # To fix the determinant problem, we get the determinant of matrix_a and use it to normalize
406            det_a = linalg.det(matrix_a)
407
408            # For a 2x2 matrix A and a scalar c, we know that det(cA) = c^2 det(A)
409            # We want B = cA such that det(B) = det(S), where S is the start matrix,
410            # so then we can scale it with the animation, so we get
411            # det(cA) = c^2 det(A) = det(S) => c = sqrt(abs(det(S) / det(A)))
412            # Then we scale A to get the determinant we want, and call that matrix_b
413            if det_a == 0:
414                c = 0
415            else:
416                c = np.sqrt(abs(det_start / det_a))
417
418            matrix_b = c * matrix_a
419            det_b = linalg.det(matrix_b)
420
421            # matrix_to_render is the final matrix that we then render for this frame
422            # It's B, but we scale it over time to have the target determinant
423
424            # We want some C = dB such that det(C) is some target determinant T
425            # det(dB) = d^2 det(B) = T => d = sqrt(abs(T / det(B)))
426
427            # We're also subtracting 1 and multiplying by the proportion and then adding one
428            # This just scales the determinant along with the animation
429
430            # That is all of course, if we can do that
431            # We'll crash if we try to do this with det(B) == 0
432            if det_b == 0:
433                return matrix_a
434
435            scalar = 1 + proportion * (np.sqrt(abs(det_target / det_b)) - 1)
436            return scalar * matrix_b
437
438        def animate_between_matrices(self, matrix_start: MatrixType, matrix_target: MatrixType, steps: int = 100) ->
      ↪     None:
439            """Animate from the start matrix to the target matrix."""
440            self.animating = True
441
442            for i in range(0, steps + 1):
443                if not self.animating:
444                    break
445
446                matrix_to_render = self._get_animation_frame(matrix_start, matrix_target, i / steps)
447
448                if self.is_matrix_too_big(matrix_to_render):
449                    self.show_error_message('Matrix too big', "This matrix doesn't fit on the canvas")
450                    self.animating = False
451                    return
452
453                self.plot.visualize_matrix_transformation(matrix_to_render)
454
455                # We schedule the plot to be updated, tell the event loop to
456                # process events, and asynchronously sleep for 10ms
457                # This allows for other events to be processed while animating, like zooming in and out
458                self.plot.update()
459                QApplication.processEvents()
460                QThread.msleep(1000 // steps)
461
462            self.animating = False
```

Notice the `if` statement on line 394. This checks if the determinant is positive and the dot product of the basis vectors is close to 0. If these are both true, then the application matrix is a rotation

matrix[25].

3.10.9.2   Utility functions and logarithmic spirals

I moved the `create_rotation_matrix()` function from `lintrans.matrices.wrapper` to a new `lintrans.matrices.utility` module, which will contain utility functions to help with animating rotations. And for the sake of consistency, I decided to only care about positive angles, so I adapted the function to make all angles positive by taking them modulo 360° or 2π radians.

I also added some other utility functions - `polar_coords()` and `rect_coords()` - which will convert rectilinear (Cartesian) coordinates to polar coordinates and back again, respectively.

```python
# 183dd02b4194c93aad51a21b7ad4147418c387c5
# src/lintrans/matrices/utility.py

"""This module provides simple utility methods for matrix and vector manipulation."""

from __future__ import annotations

import math

import numpy as np

from lintrans.typing_ import MatrixType


def polar_coords(x: float, y: float, *, degrees: bool = False) -> tuple[float, float]:
    """Return the polar coordinates of a given (x, y) Cartesian coordinate.

    .. note:: We're returning the angle in the range [0, 2pi)
    """
    radius = math.hypot(x, y)

    # PyCharm complains about np.angle taking a complex argument even though that's what it's designed for
    # noinspection PyTypeChecker
    angle = float(np.angle(x + y * 1j, degrees))

    if angle < 0:
        angle += 2 * np.pi

    return radius, angle


def rect_coords(radius: float, angle: float, *, degrees: bool = False) -> tuple[float, float]:
    """Return the rectilinear coordinates of a given polar coordinate."""
    if degrees:
        angle = np.radians(angle)

    return radius * np.cos(angle), radius * np.sin(angle)


def create_rotation_matrix(angle: float, *, degrees: bool = True) -> MatrixType:
    """Create a matrix representing a rotation (anticlockwise) by the given angle.

    :Example:

    >>> create_rotation_matrix(30)
    array([[ 0.8660254, -0.5      ],
           [ 0.5      ,  0.8660254]])
    >>> create_rotation_matrix(45)
    array([[ 0.70710678, -0.70710678],
           [ 0.70710678,  0.70710678]])
    >>> create_rotation_matrix(np.pi / 3, degrees=False)
    array([[ 0.5      , -0.8660254],
           [ 0.8660254,  0.5      ]])
```

---

[25]This isn't actually true. An enlargement matrix would also satisfy these conditions, but I didn't realise that at the time.

```
57
58         :param float angle: The angle to rotate anticlockwise by
59         :param bool degrees: Whether to interpret the angle as degrees (True) or radians (False)
60         :returns MatrixType: The resultant matrix
61         """
62         rad = np.deg2rad(angle % 360) if degrees else angle % (2 * np.pi)
63         return np.array([
64             [np.cos(rad), -1 * np.sin(rad)],
65             [np.sin(rad), np.cos(rad)]
66         ])
```

I then added unit tests for these functions and to assert that `create_rotation_matrix()` always makes its angle positive.

```
   # 7b0c374d586e487e547dad886cc24239f768b554
   # tests/matrices/utility/test_coord_conversion.py

7      """Test conversion between polar and rectilinear coordinates in :mod:`lintrans.matrices.utility`."""
8
9      from numpy import pi, sqrt
10     from pytest import approx
11
12     from lintrans.matrices.utility import polar_coords, rect_coords
13
14     expected_coords: list[tuple[tuple[float, float], tuple[float, float]]] = [
15         ((0, 0), (0, 0)),
16         ((1, 1), (sqrt(2), pi / 4)),
17         ((0, 1), (1, pi / 2)),
18         ((1, 0), (1, 0)),
19         ((sqrt(2), sqrt(2)), (2, pi / 4)),
20         ((-3, 4), (5, 2.214297436)),
21         ((4, -3), (5, 5.639684198)),
22         ((5, -0.2), (sqrt(626) / 5, 6.24320662)),
23         ((-1.3, -10), (10.08414597, 4.583113976)),
24         ((23.4, 0), (23.4, 0)),
25         ((pi, -pi), (4.442882938, 1.75 * pi))
26     ]
27
28
29     def test_polar_coords() -> None:
30         """Test that :func:`lintrans.matrices.utility.polar_coords` works as expected."""
31         for rect, polar in expected_coords:
32             assert polar_coords(*rect) == approx(polar)
33
34
35     def test_rect_coords() -> None:
36         """Test that :func:`lintrans.matrices.utility.rect_coords` works as expected."""
37         for rect, polar in expected_coords:
38             assert rect_coords(*polar) == approx(rect)
39
40         assert rect_coords(1, 0) == approx((1, 0))
41         assert rect_coords(1, pi) == approx((-1, 0))
42         assert rect_coords(1, 2 * pi) == approx((1, 0))
43         assert rect_coords(1, 3 * pi) == approx((-1, 0))
44         assert rect_coords(1, 4 * pi) == approx((1, 0))
45         assert rect_coords(1, 5 * pi) == approx((-1, 0))
46         assert rect_coords(1, 6 * pi) == approx((1, 0))
47         assert rect_coords(20, 100) == approx(rect_coords(20, 100 % (2 * pi)))


   # 7b0c374d586e487e547dad886cc24239f768b554
   # tests/matrices/utility/test_rotation_matrices.py

74     def test_create_rotation_matrix() -> None:
75         """Test that create_rotation_matrix() works with given angles and expected matrices."""
76         for degrees, radians, matrix in angles_and_matrices:
77             assert create_rotation_matrix(degrees, degrees=True) == pytest.approx(matrix)
78             assert create_rotation_matrix(radians, degrees=False) == pytest.approx(matrix)
79
80             assert create_rotation_matrix(-1 * degrees, degrees=True) == pytest.approx(np.linalg.inv(matrix))
81             assert create_rotation_matrix(-1 * radians, degrees=False) == pytest.approx(np.linalg.inv(matrix))
```

```
82
83        assert (create_rotation_matrix(-90, degrees=True) ==
84                create_rotation_matrix(270, degrees=True)).all()
85        assert (create_rotation_matrix(-0.5 * np.pi, degrees=False) ==
86                create_rotation_matrix(1.5 * np.pi, degrees=False)).all()
```

```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main *=]
$ pytest -v
============================= test session starts =============================
platform linux -- Python 3.11.0, pytest-7.2.1, pluggy-1.0.0 -- /home/dyson/temp/lintrans/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/dyson/temp/lintrans, configfile: pyproject.toml, testpaths: tests
collected 32 items

tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs0-True] PASSED        [  3%]
tests/gui/test_dialog_utility_functions.py::test_is_valid_float[inputs1-False] PASSED       [  6%]
tests/gui/test_dialog_utility_functions.py::test_round_float PASSED                         [  9%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs0-True] PASSED   [ 12%]
tests/matrices/test_parse_and_validate_expression.py::test_validate_matrix_expression[inputs1-False] PASSED  [ 15%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_matrix_expression PASSED   [ 18%]
tests/matrices/test_parse_and_validate_expression.py::test_parse_error PASSED               [ 21%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_matrix_addition PASSED   [ 25%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_two_matrix_multiplication PASSED   [ 28%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_identity_multiplication PASSED   [ 31%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_simple_three_matrix_multiplication PASSED   [ 34%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_inverses PASSED      [ 37%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_powers PASSED        [ 40%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_matrix_transpose PASSED     [ 43%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_rotation_matrices PASSED    [ 46%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_multiplication_and_addition PASSED   [ 50%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_complicated_expressions PASSED   [ 53%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_value_errors PASSED         [ 56%]
tests/matrices/matrix_wrapper/test_evaluate_expression.py::test_linalgerror PASSED          [ 59%]
tests/matrices/matrix_wrapper/test_misc.py::test_get_expression PASSED                      [ 62%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_get_matrix PASSED     [ 65%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_get_name_error PASSED        [ 68%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_basic_set_matrix PASSED     [ 71%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_expression PASSED       [ 75%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_simple_dynamic_evaluation PASSED   [ 78%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_recursive_dynamic_evaluation PASSED   [ 81%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_identity_error PASSED   [ 84%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_name_error PASSED       [ 87%]
tests/matrices/matrix_wrapper/test_setitem_and_getitem.py::test_set_type_error PASSED       [ 90%]
tests/matrices/utility/test_coord_conversion.py::test_polar_coords PASSED                   [ 93%]
tests/matrices/utility/test_coord_conversion.py::test_rect_coords PASSED                     [ 96%]
tests/matrices/utility/test_rotation_matrices.py::test_create_rotation_matrix PASSED         [100%]

============================= 32 passed in 0.36s =============================
```

Figure 3.10.11: Running `pytest` with the new tests

Then I had to actually implement the logarithmic spiral. I spent several hours playing around with Desmos and various formulae on paper, and eventually devised a way to connect two points with a logarithmic spiral. Sometimes.

The formula I came up with was designed to give a polar coordinate (r, θ) which was some proportion p between 0 and 1 along the logarithmic spiral connecting the polar coordinates (r_1, θ_1) and (r_2, θ_2). A generic logarithmic spiral has the form $r = b^\theta$, where b is called the base. I came up with the following equations to define my connecting curve:

$$r = r_1 \times b^{(\theta - \theta_1)} \qquad \text{where } b = \left(\frac{r_2}{r_1}\right)^{\frac{-1}{\theta_1 - \theta_2}}$$

To get an arbitrary point with a proportion p along the curve, you can find θ by $\theta = \theta_1 + p(\theta_2 - \theta_1)$ and then plug that back in to find r.

Then I just had to implement this formula in code to move each basis vector accordingly.

```
# 2c86b3a7f40deb760e852fe6d0213cba15afcf9e
# src/lintrans/gui/main_window.py

37    class LintransMainWindow(QMainWindow):
...
377        def _get_animation_frame(self, start: MatrixType, target: MatrixType, proportion: float) -> MatrixType:
```

```
...
388              det_target = linalg.det(target)
389              det_start = linalg.det(start)
390
391              # This is the matrix that we're applying to get from start to target
392              # We want to check if it's rotation-like
393              matrix_application = target @ linalg.inv(start)
394
395              if linalg.det(matrix_application) > 0 and abs(np.dot(matrix_application.T[0], matrix_application.T[1])) <
     ↪    0.1:
396                  # Get the columns of the matrices
397                  # We're going to move i and then move j
398                  i_vectors = (start.T[0], target.T[0])
399                  j_vectors = (start.T[1], target.T[1])
400
401                  matrix_list: list[tuple[float, float]] = []
402
403                  for start_vector, end_vector in [i_vectors, j_vectors]:
404                      # We want the points in polar coordinates
405                      s_length, s_angle = polar_coords(start_vector[0], start_vector[1])
406                      e_length, e_angle = polar_coords(end_vector[0], end_vector[1])
407
408                      # We're using the standard formula for a logarithmic spiral,
409                      # but we want to connect two specific points
410                      # This base is just the base that we raise the angle to in the formula
411                      base = (e_length / s_length) ** (-1 / (s_angle - e_angle))
412
413                      angle = s_angle + proportion * (e_angle - s_angle)
414
415                      # Logarithmic spiral equation
416                      radius = s_length * base ** (angle - s_angle)
417
418                      matrix_list.append(rect_coords(radius, angle))
419
420                  return np.array(
421                      [
422                          [matrix_list[0][0], matrix_list[1][0]],
423                          [matrix_list[0][1], matrix_list[1][1]]
424                      ]
425                  )
```

It worked really well for simple rotations like `"rot(170)"` and even worked for `"2rot(180)"`, which I was very happy with. However, if you start from $\mathbf{I}$, which is the default starting position, and try to rotate any more than 270°, the i vector goes anticlockwise, and the j vector goes clockwise. If you start at $\mathbf{I}$ and try to animate `"rot(180)"` more than once, then any time after the first, the basis vectors will rotate in opposite directions and go through each other. Even though after applying `"rot(180)"` twice, you should be back to $\mathbf{I}$, the floating point error adds up and that tiny deviation breaks it.

This approach was a good start, but I'm going to need something better.

3.10.9.3   Checking for enlargement matrices

I found a bug where any enlargement matrix would immediately crash the program when you tried to animate. Suffice it to say, that's not very good. Recall back to footnote 25, where I said that the check would count enlargement matrices as rotation matrices. The logarithmic spiral formula can't connect two points with the same angle, since it would end up dividing by zero to find the base. This, of course, results in a crash.

To fix this bug, I simply changed the check to ignore any matrix that looks like an enlargement. If you divide an enlargement matrix by its determinant, then it will look like $\mathbf{I}$, so we can just check that. Additionally, I only want to animate rotations smoothly like this if the smoothen determinant setting is on.

```
# 31100f35e8d46fcddaad18ee801e28904dc3011d
# src/lintrans/gui/main_window.py
```

```
37    class LintransMainWindow(QMainWindow):
...
377       def _get_animation_frame(self, start: MatrixType, target: MatrixType, proportion: float) -> MatrixType:
...
395           if self.plot.display_settings.smoothen_determinant \
396                   and linalg.det(matrix_application) > 0 \
397                   and abs(np.dot(matrix_application.T[0], matrix_application.T[1])) < 0.1 \
398                   and not (matrix_application / linalg.det(matrix_application) - np.eye(2) < 1e-5).all():
```

3.10.9.4   Using incremental circles instead

The logarithmic spiral was a nice idea, but the instability made it unsuitable as a long term solution. Instead, I decided to use incremental circles instead. I would scale a radius according the start and end radii, as well as the proportion of the way through the animation. Then I would find an angle in the same way and combine them into one polar coordinate.

Also, if the starting matrix is singular, then NumPy can't invert it so it crashes. This is obviously also bad, so I just had to check for it.

```
# f97f6542a051c05896711a01eb48a04dba693f64
# src/lintrans/gui/main_window.py

37    class LintransMainWindow(QMainWindow):
...
377       def _get_animation_frame(self, start: MatrixType, target: MatrixType, proportion: float) -> MatrixType:
...
388           det_target = linalg.det(target)
389           det_start = linalg.det(start)
390
391           # This is the matrix that we're applying to get from start to target
392           # We want to check if it's rotation-like
393           if linalg.det(start) == 0:
394               matrix_application = None
395           else:
396               matrix_application = target @ linalg.inv(start)
397
398           if matrix_application is not None \
399                   and self.plot.display_settings.smoothen_determinant \
400                   and linalg.det(matrix_application) > 0 \
401                   and abs(np.dot(matrix_application.T[0], matrix_application.T[1])) < 0.1:
402               # Get the columns of the matrices
403               # We're going to move i and then move j
404               i_vectors = (start.T[0], target.T[0])
405               j_vectors = (start.T[1], target.T[1])
406
407               matrix_list: list[tuple[float, float]] = []
408               TWO_PI = 2 * np.pi
409
410               for start_vector, end_vector in [i_vectors, j_vectors]:
411                   # We want the points in polar coordinates
412                   s_length, s_angle = polar_coords(start_vector[0], start_vector[1])
413                   e_length, e_angle = polar_coords(end_vector[0], end_vector[1])
414
415                   angle_difference = e_angle - s_angle % TWO_PI
416
417                   if angle_difference > np.pi + 0.01:  # Extra 0.01 accounts for floating point error
418                       angle = s_angle + proportion * (TWO_PI - angle_difference)
419                   else:
420                       angle = s_angle + proportion * angle_difference
421
422                   radius = s_length + proportion * (e_length - s_length)
423                   # angle = s_angle % TWO_PI + proportion * angle_difference
424
425                   matrix_list.append(rect_coords(radius, angle))
426
427               return np.array(
428                   [
```

```
429                    [matrix_list[0][0], matrix_list[1][0]],
430                    [matrix_list[0][1], matrix_list[1][1]]
431                ]
432          )
```

Does this solve the instability problem with the basis vectors rotating in opposite directions? No. Does it make this algorithm simpler and easier to improve? Yes.

3.10.9.5   Treating the rotation as one thing
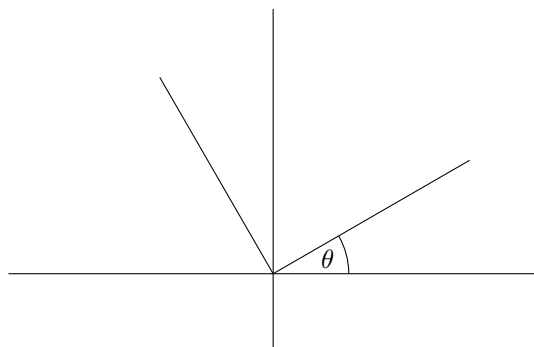
After much thought and deliberation, I realised that my fatal error was rotating the basis vectors independently. This allowed small floating point errors to cause the basis vectors to be rotated in opposite directions. What I need to do instead was to find the angle from one basis vector, and use that to rotate both.

I can find the angle θ by using the first column of the application matrix, and then rotate both basis vectors by this same angle. By finding the angle just once and using a dedicated `rotate_coord()` utility function to avoid converting back and forth, it ensures that both the basis vectors get rotated by the same angle in the same direction.

I first added a type alias for a vector, mainly for convenience.

```
# c228904ab61cb33a6581fe28bbd6330b5c2ece97
# src/lintrans/typing_/__init__.py

27    VectorType = NDArray[(2,), Float]
28    """This type represents a 2D vector as a NumPy array, for use with :attr:`MatrixType`."""
```

I then created a utility function to rotate a rectilinear coordinate by a given angle.

```
# 425f3d653770724a01fd5f94314c7d076dadfef2
# src/lintrans/matrices/utility.py

43    def rotate_coord(x: float, y: float, angle: float, *, degrees: bool = False) -> tuple[float, float]:
44        """Rotate a rectilinear coordinate by the given angle."""
45        if degrees:
46            angle = np.radians(angle)
47
48        r, theta = polar_coords(x, y, degrees=degrees)
49        theta = (theta + angle) % (2 * np.pi)
50
51        return rect_coords(r, theta, degrees=degrees)
```

Then when generating a frame, I normalise the angle θ to be $-\pi \leq \theta < \pi$ (radians) and then rotate the coordinates of each basis vector by a proportion of the angle for each frame in the animation.

```
# 425f3d653770724a01fd5f94314c7d076dadfef2
# src/lintrans/gui/main_window.py

37    class LintransMainWindow(QMainWindow):
...
377        def _get_animation_frame(self, start: MatrixType, target: MatrixType, proportion: float) -> MatrixType:
```

```
378            """Get the matrix to render for this frame of the animation.
379
380            This method will smoothen the determinant if that setting in enabled and if the determinant is positive.
381            It also animates rotation-like matrices using a logarithmic spiral to rotate around and scale continuously.
382            Essentially, it just makes things look good when animating.
383
384            :param MatrixType start: The starting matrix
385            :param MatrixType start: The target matrix
386            :param float proportion: How far we are through the loop
387            """
388            det_target = linalg.det(target)
389            det_start = linalg.det(start)
390
391            # This is the matrix that we're applying to get from start to target
392            # We want to check if it's rotation-like
393            if linalg.det(start) == 0:
394                matrix_application = None
395            else:
396                matrix_application = target @ linalg.inv(start)
397
398            if matrix_application is not None \
399                    and self.plot.display_settings.smoothen_determinant \
400                    and linalg.det(matrix_application) > 0 \
401                    and abs(np.dot(matrix_application.T[0], matrix_application.T[1])) < 0.1:
402                rotation_vector: VectorType = matrix_application.T[0]  # Take the i column
403                radius, angle = polar_coords(*rotation_vector)
404
405                # We want the angle to be in [-pi, pi), so we have to subtract 2pi from it if it's too big
406                if angle > np.pi:
407                    angle -= 2 * np.pi
408
409                i: VectorType = start.T[0]
410                j: VectorType = start.T[1]
411
412                # Scale the coords with a list comprehension
413                # It's a bit janky, but rotate_coords() will always return a 2-tuple,
414                # so new_i and new_j will always be lists of length 2
415                scale = (radius - 1) * proportion + 1
416                new_i = [scale * c for c in rotate_coord(i[0], i[1], angle * proportion)]
417                new_j = [scale * c for c in rotate_coord(j[0], j[1], angle * proportion)]
418
419                return np.array(
420                    [
421                        [new_i[0], new_j[0]],
422                        [new_i[1], new_j[1]]
423                    ]
424                )
```

And of course, I added this to the changelog.

```
<!-- 425f3d653770724a01fd5f94314c7d076dadfef2 -->
<!-- CHANGELOG.md -->

12    ### Added
...
16    - Add proper rotation animation that rotates at constant speed
```

Now rotations finally work properly!

3.10.10   Adding a setting for animation time

Once I'd merged the `dev/anim-rot` branch, I was basically done with `v0.2.2`, but I still wanted to add one last thing. I already had a display setting to control the delay between animating consecutive matrices in a sequence, but I also wanted a display setting to control how long the actual animation lasts. This would allow a teacher to animate something slowly to more precisely show an animation. Implementing it was very simple.

I just had to add the setting to the `DisplaySettings` dataclass, add code to the display settings dialog
to allow changing it, and then use it in animations.

```python
# 86343c4b0dd4ed9052184fd4ed924062a1cd9264
# src/lintrans/gui/settings.py

15  class DisplaySettings:
...
51      animation_time: int = 1200
52      """This is the number of milliseconds that an animation takes."""
```

```python
# 86343c4b0dd4ed9052184fd4ed924062a1cd9264
# src/lintrans/gui/dialogs/settings.py

21  class SettingsDialog(FixedSizeDialog):
...
24      def __init__(self, *args, **kwargs):
...
127         self.label_animation_time = QtWidgets.QLabel(self)
128         self.label_animation_time.setText('Total animation length (ms)')
129         self.label_animation_time.setToolTip(
130             'How long it takes for an animation to complete'
131         )
132
133         self.lineedit_animation_time = QtWidgets.QLineEdit(self)
134         self.lineedit_animation_time.setValidator(QIntValidator(1, 9999, self))
...
226     def load_settings(self) -> None:
...
236         self.lineedit_animation_time.setText(str(self.display_settings.animation_time))
...
245     def confirm_settings(self) -> None:
...
255         self.display_settings.animation_time = int(self.lineedit_animation_time.text())
```

```python
# 86343c4b0dd4ed9052184fd4ed924062a1cd9264
# src/lintrans/gui/main_window.py

37  class LintransMainWindow(QMainWindow):
...
465     def animate_between_matrices(self, matrix_start: MatrixType, matrix_target: MatrixType) -> None:
466         """Animate from the start matrix to the target matrix."""
467         self.animating = True
468
469         # Making steps depend on animation_time ensures a smooth animation without
470         # massive overheads for small animation times
471         steps = self.plot.display_settings.animation_time // 10
472
473         for i in range(0, steps + 1):
474             if not self.animating:
475                 break
476
477             matrix_to_render = self._get_animation_frame(matrix_start, matrix_target, i / steps)
478
479             if self.is_matrix_too_big(matrix_to_render):
480                 self.show_error_message('Matrix too big', "This matrix doesn't fit on the canvas")
481                 self.animating = False
482                 return
483
484             self.plot.visualize_matrix_transformation(matrix_to_render)
485
486             # We schedule the plot to be updated, tell the event loop to
487             # process events, and asynchronously sleep for 10ms
488             # This allows for other events to be processed while animating, like zooming in and out
489             self.plot.update()
490             QApplication.processEvents()
491             QThread.msleep(self.plot.display_settings.animation_time // steps)
492
493         self.animating = False
```

And then finally, I added it to the changelog.

```
<!-- 86343c4b0dd4ed9052184fd4ed924062a1cd9264 -->
<!-- CHANGELOG.md -->

10      ### Added
...
16      - Allow animation time to be varied
```

3.10.11 Releasing `v0.2.2`

Now I can just update the version number, tag the commit, and let GitHub Actions do the rest.

```
# 1139f94366be359aa5606c750c67b72c2777c961
# src/lintrans/__init__.py

11      __version__ = '0.2.2'
```


Figure 3.10.12: The release of `v0.2.2` on GitHub

3.11   Teacher suggestions

Now that I've got `v0.2.2` working and released, it's time to talk to some teachers. Ms Arnold is going to be the one that's actually using it, so obviously I needed to talk to her about it, but she wasn't available for a few days. I was talking to another maths teacher, Mr Dunkley, about my personal statement and took the opportunity to show him `lintrans` as well.

He was able to download and run it on a native Windows 10 installation, which was quite a relief. I'd been unable to test it on a real bare-metal Windows installation so far; I only had virtual machines.

He suggested two main improvements:

1. In the visual definition dialog, the basis vectors should snap to integer coordinates when they get close enough

2. There should be a dialog box which displays matrices that you've already defined

3.11.1   Fixing a bug with animating stretches

While Mr Dunkley was playing around with `lintrans`, I noticed a bug where animating uniform stretches, where every direction gets stretched equally work, just fine, but a matrix like $\begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$ would be animated as if it was $\begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}$ and the plane would stretch equally in all directions.

After looking at the code later, I realised this was a problem with my new animation code detecting this type of matrix as a rotation. To fix this, I could just add a check to the `if` statement before the rotation animation code. This new check would make sure that the basis vectors of the application matrix were approximately the same length. By checking this as well, I fixed the bug.

```python
# eb118f02db9fb9c76ce15a976f81a037415d2a4e
# src/lintrans/gui/main_window.py

 39    class LintransMainWindow(QMainWindow):
...
389        def _get_animation_frame(self, start: MatrixType, target: MatrixType, proportion: float) -> MatrixType:
...
410            # For a matrix to represent a rotation, it must have a positive determinant,
411            # its vectors must be perpendicular, and its vectors must be the same length
412            # The checks for 'abs(value) < 1e-10' are to account for floating point error
413            if matrix_application is not None \
414                    and self.plot.display_settings.smoothen_determinant \
415                    and linalg.det(matrix_application) > 0 \
416                    and abs(np.dot(matrix_application.T[0], matrix_application.T[1])) < 1e-10 \
417                    and abs(np.hypot(*matrix_application.T[0]) - np.hypot(*matrix_application.T[1])) < 1e-10:
```

3.11.2   Fixing a hang after closing during an animation

This bug was tiny and only really affected me, but when I run `lintrans` from the terminal with `python -m lintrans` and close it during an animation, I have to wait a moment to get my shell prompt back. Clearly, the program is still running even after the main window is closed.

I'm not entirely sure what was happening here, but it was something to do with Qt5's event loop and threading model, and I could fix the bug by just overriding `LintransMainWindow.closeEvent()` to set `self.animating = False` and that would ensure that animations are stopped before the main window closes.

```python
# d60c1878058799be544f5cf0847b478dccd3a021
# src/lintrans/gui/main_window.py
```

```
 39    class LintransMainWindow(QMainWindow):
...
262        def closeEvent(self, event: QCloseEvent) -> None:
263            """Handle a :class:`QCloseEvent` by cancelling animation first."""
264            self.animating = False
265            event.accept()
```

3.11.3   Adding snapping in the visual definition dialog

I was now ready to implement snapping in the visual definition dialog.

The `DefineVisuallyWidget` class has a `self.dragged_point` attribute, which is the coordinates of the point being dragged. In `mouseMoveEvent()`, we get the coordinates of the current cursor position and set the dragged point accordingly. That's all that happens in the old code.

```
# d60c1878058799be544f5cf0847b478dccd3a021
# src/lintrans/gui/plots/widgets.py

 86    class DefineVisuallyWidget(VisualizeTransformationWidget):
...
127        def mouseMoveEvent(self, event: QMouseEvent) -> None:
128            """Handle the mouse moving on the canvas."""
129            mx = event.x()
130            my = event.y()
131
132            if self.dragged_point is not None:
133                x, y = self.grid_coords(mx, my)
134
135                if self.dragged_point == self.point_i:
136                    self.point_i = x, y
137
138                elif self.dragged_point == self.point_j:
139                    self.point_j = x, y
140
141                self.dragged_point = x, y
142
143                self.update()
144
145                event.accept()
146
147            event.ignore()
```

To add snapping to this, we can just get the 4 integer coordinates around the dragged point, check each of their distances to the dragged point, and if it's sufficiently close to one of them, snap it to that point.

```
# f2de39fec299bb8ed155ee649d34c9063787e71a
# src/lintrans/gui/plots/widgets.py

 87    class DefineVisuallyWidget(VisualizeTransformationWidget):
...
128        def mouseMoveEvent(self, event: QMouseEvent) -> None:
129            """Handle the mouse moving on the canvas."""
130            mx = event.x()
131            my = event.y()
132
133            if self.dragged_point is None:
134                event.ignore()
135                return
136
137            x, y = self.grid_coords(mx, my)
138
139            possible_snaps: List[Tuple[int, int]] = [
140                (floor(x), floor(y)),
141                (floor(x), ceil(y)),
```

```
142                    (ceil(x), floor(y)),
143                    (ceil(x), ceil(y))
144                ]
145
146            snap_distances: List[Tuple[float, Tuple[int, int]]] = [
147                (dist((x, y), coord), coord)
148                for coord in possible_snaps
149            ]
150
151            for snap_dist, coord in snap_distances:
152                if snap_dist < 0.1:
153                    x, y = coord
154
155            if self.dragged_point == self.point_i:
156                self.point_i = x, y
157
158            elif self.dragged_point == self.point_j:
159                self.point_j = x, y
160
161            self.dragged_point = x, y
162
163            self.update()
164
165            event.accept()
```

This all worked very well.

3.11.4   Respecting transitional animation in animation sequences

It's good to animate a sequence of matrices by applying them one after another, but it can also be beneficial to animate a sequence by just moving between them in a transitional animation style. This is already possible for individual animations by disabling the *Applicative animation* display setting, but it would be nice to respect this setting for animation sequences.

To do this, I just have to add a few lines to the part where we do animation sequences.

```
# 41907b81661f3878e435b794d9d719491ef14237
# src/lintrans/gui/main_window.py

39   class LintransMainWindow(QMainWindow):
...
327      def animate_expression(self) -> None:
...
339          # If there's commas in the expression, then we want to animate each part at a time
340          if ',' in text:
341              current_matrix = matrix_start
342              self.animating_sequence = True
343
344              # For each expression in the list, right multiply it by the current matrix,
345              # and animate from the current matrix to that new matrix
346              for expr in text.split(',')[::-1]:
347                  try:
348                      new_matrix = self.matrix_wrapper.evaluate_expression(expr)
349
350                      if self.plot.display_settings.applicative_animation:
351                          new_matrix = new_matrix @ current_matrix
352                  except LinAlgError:
353                      self.show_error_message('Singular matrix', 'Cannot take inverse of singular matrix')
354                      return
355
356                  if not self.animating_sequence:
357                      break
358
359                  self.animate_between_matrices(current_matrix, new_matrix)
360                  current_matrix = new_matrix
361
362                  # Here we just redraw and allow for other events to be handled while we pause
```

```
363                    self.plot.update()
364                    QApplication.processEvents()
365                    QThread.msleep(self.plot.display_settings.animation_pause_length)
366
367                self.animating_sequence = False
```

3.11.5   Adding the info panel

One of things that Mr Dunkley wanted to see was a panel or dialog box that would display all the matrices that you've already defined.

3.11.5.1   Creating the initial dialog

The first thing I did was create a new git branch, and then I created a new dialog class, added a button to display the new info panel, and created a simple utility method for a `MatrixWrapper` to return a list of all the matrices it's got defined.

```
# 6469f9f3678a89b3b7f388a6c5132335d9a6a947
# src/lintrans/matrices/wrapper.py

24    class MatrixWrapper:
...
251        def get_defined_matrices(self) -> List[Tuple[str, Union[MatrixType, str]]]:
252            """Return a list of tuples containing the name and value of all defined matrices in the wrapper.
253
254            :returns: A list of tuples where the first element is the name, and the second element is the value
255            :rtype: List[Tuple[str, Union[MatrixType, str]]]
256            """
257            matrices = []
258
259            for name, value in self._matrices.items():
260                if value is not None:
261                    matrices.append((name, value))
262
263            return matrices
```

```
# 6469f9f3678a89b3b7f388a6c5132335d9a6a947
# src/lintrans/gui/dialogs/misc.py

100   class InfoPanelDialog(FixedSizeDialog):
101       """A simple dialog class to display an info panel that shows all currently defined matrices."""
102
103       def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
104           super().__init__(*args, **kwargs)
105           self.wrapper = matrix_wrapper
106
107           self.setWindowTitle('Defined matrices')
108
109           for name, value in self.wrapper.get_defined_matrices():
110               print(name, value)
```

```
# 6469f9f3678a89b3b7f388a6c5132335d9a6a947
# src/lintrans/gui/main_window.py

39    class LintransMainWindow(QMainWindow):
...
45        def __init__(self):
...
203           # Info panel button
204
205           self.button_info_panel = QtWidgets.QPushButton(self)
206           self.button_info_panel.setText('Show defined matrices')
207           self.button_info_panel.setToolTip('Open an info panel with all matrices that have been defined in this
              ↪  session')
208           self.button_info_panel.clicked.connect(lambda: InfoPanelDialog(self.matrix_wrapper, self).open())
```

Trying to open the new info panel results in a tiny, useless dialog box appearing but the defined matrices get printed out in the terminal, which is exactly what I expected. It's a good proof of concept, but obviously needs a bit of visual polish.
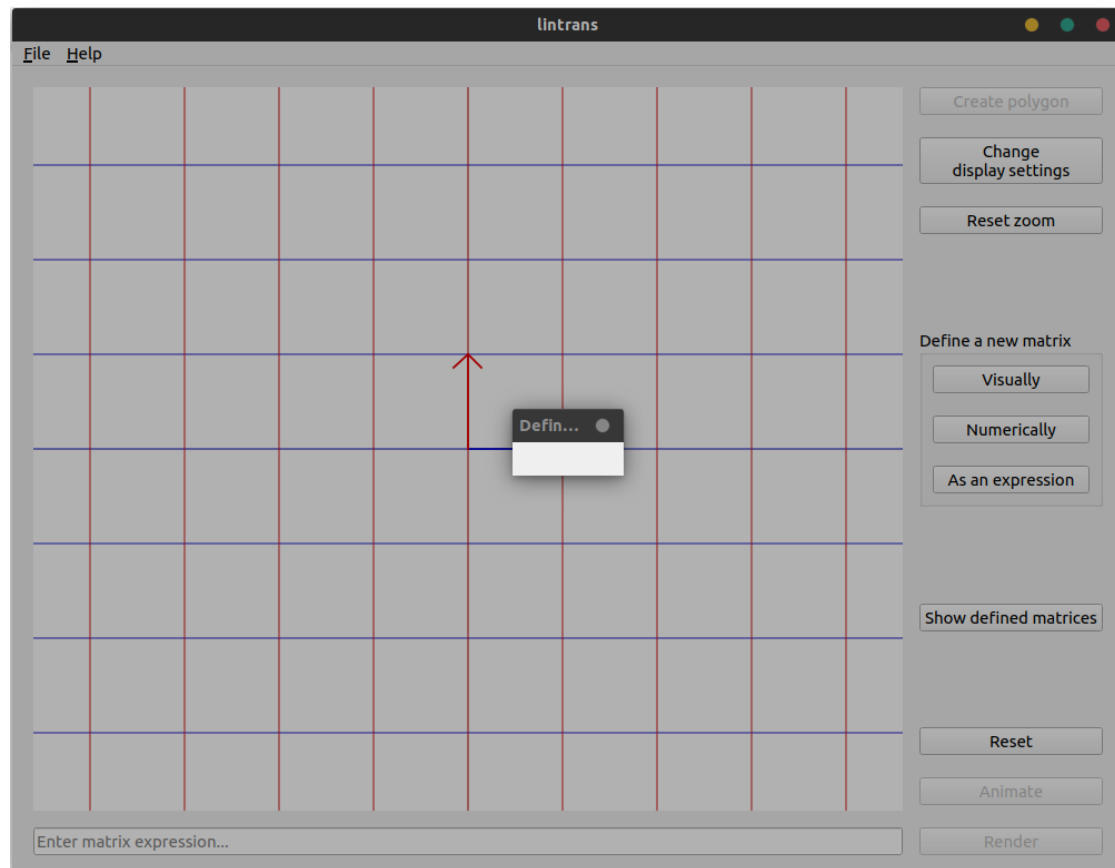
Figure 3.11.1: The GUI with the tiny dialog box and new button visible in the background

3.11.5.2 Adding the matrices to the panel

The next step was obviously to add the widgets that actually represent these matrices in the dialog box. To do this, I created a layout and iterated over the matrices in the wrapper to add some widgets for each. These widgets consist of a name label, a label for '=', and a container widget to display the content of the matrix. I created a separate method to generate a container widget for the matrix content.

```
# addc22a8984a9d8e1d100b5800836a2af3f42760
# src/lintrans/gui/dialogs/misc.py

102  class InfoPanelDialog(FixedSizeDialog):
103      """A simple dialog class to display an info panel that shows all currently defined matrices."""
104
105      def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
106          """Create the dialog box with all the widgets needed to show the information."""
107          super().__init__(*args, **kwargs)
108          self.wrapper = matrix_wrapper
109
110          self.setWindowTitle('Defined matrices')
111
112          vlay_main = QVBoxLayout()
113          vlay_main.setSpacing(20)
114
115          for name, value in self.wrapper.get_defined_matrices():
```

```
116                    label_name = QLabel(self)
117                    label_name.setText(name)
118                    label_name.setAlignment(Qt.AlignLeft)
119
120                    label_equals = QLabel(self)
121                    label_equals.setText('=')
122                    label_equals.setAlignment(Qt.AlignLeft)
123
124                    widget_matrix = self._get_matrix_widget(value)
125
126                    hlay_matrix = QHBoxLayout()
127                    hlay_matrix.setSpacing(20)
128                    hlay_matrix.addWidget(label_name)
129                    hlay_matrix.addWidget(label_equals)
130                    hlay_matrix.addWidget(widget_matrix)
131
132                    vlay_main.addLayout(hlay_matrix)
133
134            self.setContentsMargins(10, 10, 10, 10)
135            self.setLayout(vlay_main)
136
137        def _get_matrix_widget(self, matrix: Union[MatrixType, str]) -> QWidget:
138            """Return a :class:`QWidget` containing the value of the matrix.
139
140            If the matrix is defined as an expression, it will be a simple :class:`QLabel`.
141            If the matrix is defined as a matrix, it will be a :class:`QWidget` container
142            with multiple :class:`QLabel` objects in it.
143            """
144            if isinstance(matrix, str):
145                label = QLabel(self)
146                label.setText(matrix)
147                return label
148
149            elif is_matrix_type(matrix):
150                container = QWidget(self)
151                grid_layout = QGridLayout()
152
153                # tl = top left, br = bottom right, etc.
154                label_tl = QLabel(self)
155                label_tl.setText(round_float(matrix[0][0]))
156
157                label_tr = QLabel(self)
158                label_tr.setText(round_float(matrix[0][1]))
159
160                label_bl = QLabel(self)
161                label_bl.setText(round_float(matrix[1][0]))
162
163                label_br = QLabel(self)
164                label_br.setText(round_float(matrix[1][1]))
165
166                label_paren_left = QLabel(self)
167                label_paren_left.setText('(')
168
169                label_paren_right = QLabel(self)
170                label_paren_right.setText(')')
171
172                grid_layout.addWidget(label_paren_left, 0, 0, 2, 0)
173                grid_layout.addWidget(label_tl, 0, 1)
174                grid_layout.addWidget(label_tr, 0, 2)
175                grid_layout.addWidget(label_bl, 1, 1)
176                grid_layout.addWidget(label_br, 1, 2)
177                grid_layout.addWidget(label_paren_right, 0, 3, 2, 0)
178
179                container.setLayout(grid_layout)
180
181                return container
182
183            raise ValueError('Matrix was not MatrixType or str')
```

This container widget uses a grid layout to have the numbers arranged correctly, as well as stretching the parentheses either side to properly cover the numbers.

However, the parentheses don't stretch properly. I also get the message 'QGridLayout: Multi-cell fromCol greater than toCol' in the terminal whenever I open the info panel. One for each matrix defined numerically. Suffice to say, this is not what I was trying to do.
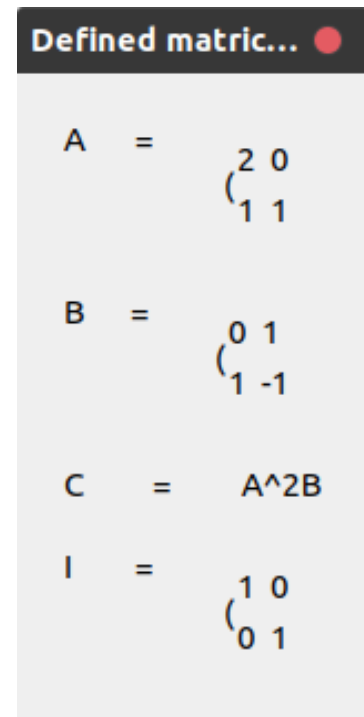
Figure 3.11.2: The info panel with some defined matrices, and the parentheses not working correctly

3.11.5.3   Fixing alignments and stretches

After checking the documentation for the `QGridLayout::addWidget()` function[42] and experimenting with some different numbers, I fixed the row and column numbers for the parentheses. I also created a temporary font for the parentheses to make them bigger. I had to make them thinner as well to avoid them looking weird. I also made the matrix names bold.

```
# 63279a27ac6fa86061c4ba10152244b2bb4ced87
# src/lintrans/gui/dialogs/misc.py
102   class InfoPanelDialog(FixedSizeDialog):
103       """A simple dialog class to display an info panel that shows all currently defined matrices."""
104
105       def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
106           """Create the dialog box with all the widgets needed to show the information."""
107           super().__init__(*args, **kwargs)
108           self.wrapper = matrix_wrapper
109
110           self.setWindowTitle('Defined matrices')
111
112           grid_layout = QGridLayout()
113           grid_layout.setSpacing(20)
114
115           bold_font = self.font()
116           bold_font.setBold(True)
117
118           name_value_pair: tuple[str, Union[MatrixType, str]]
119
120           for i, name_value_pair in enumerate(self.wrapper.get_defined_matrices()):
121               name, value = name_value_pair
122
123               label_name = QLabel(self)
124               label_name.setText(name)
125               label_name.setFont(bold_font)
```

```
126
127                label_equals = QLabel(self)
128                label_equals.setText('=')
129
130                widget_matrix = self._get_matrix_widget(value)
131
132                grid_layout.addWidget(label_name, i, 0, Qt.AlignCenter | Qt.AlignVCenter)
133                grid_layout.addWidget(label_equals, i, 1, Qt.AlignCenter | Qt.AlignVCenter)
134                grid_layout.addWidget(widget_matrix, i, 2, Qt.AlignCenter | Qt.AlignVCenter)
135
136            self.setContentsMargins(10, 10, 10, 10)
137            self.setLayout(grid_layout)
138
139        def _get_matrix_widget(self, matrix: Union[MatrixType, str]) -> QWidget:
140            """Return a :class:`QWidget` containing the value of the matrix.
141
142            If the matrix is defined as an expression, it will be a simple :class:`QLabel`.
143            If the matrix is defined as a matrix, it will be a :class:`QWidget` container
144            with multiple :class:`QLabel` objects in it.
145            """
146            if isinstance(matrix, str):
147                label = QLabel(self)
148                label.setText(matrix)
149                return label
150
151            elif is_matrix_type(matrix):
152                # tl = top left, br = bottom right, etc.
153                label_tl = QLabel(self)
154                label_tl.setText(round_float(matrix[0][0]))
155
156                label_tr = QLabel(self)
157                label_tr.setText(round_float(matrix[0][1]))
158
159                label_bl = QLabel(self)
160                label_bl.setText(round_float(matrix[1][0]))
161
162                label_br = QLabel(self)
163                label_br.setText(round_float(matrix[1][1]))
164
165                font_parens = self.font()
166                font_parens.setPointSize(int(font_parens.pointSize() * 2.5))
167                font_parens.setWeight(int(font_parens.weight() / 2.5))
168
169                label_paren_left = QLabel(self)
170                label_paren_left.setText('(')
171                label_paren_left.setFont(font_parens)
172
173                label_paren_right = QLabel(self)
174                label_paren_right.setText(')')
175                label_paren_right.setFont(font_parens)
176
177                container = QWidget(self)
178                grid_layout = QGridLayout()
179
180                grid_layout.addWidget(label_paren_left, 0, 0, -1, 1)
181                grid_layout.addWidget(label_tl, 0, 1)
182                grid_layout.addWidget(label_tr, 0, 2)
183                grid_layout.addWidget(label_bl, 1, 1)
184                grid_layout.addWidget(label_br, 1, 2)
185                grid_layout.addWidget(label_paren_right, 0, 3, -1, 1)
186
187                container.setLayout(grid_layout)
188
189                return container
190
191        raise ValueError('Matrix was not MatrixType or str')
```
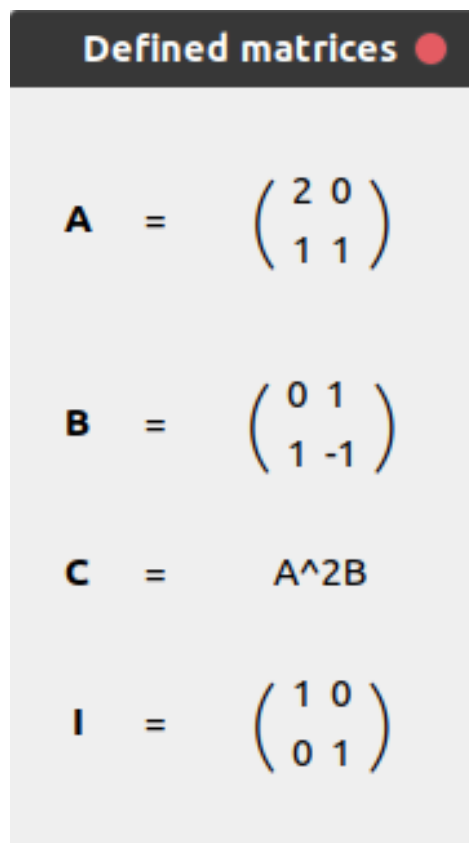
Figure 3.11.3: The new and improved info panel with correct alignment and stretching

3.11.5.4   Arranging matrices in multiple columns

The only remaining problem with the info panel is that it just extends into one long vertical column as more matrices are added. Obviously there are 26 letters in the alphabet, so 26 possible matrices that can be defined. However, 26 is a semiprime, which means if I want to arrange the matrices in a rectangle, I have to do 2×13 or 13×2. This is not particularly useful. Instead, I think it would be better to have an arrangement closer to a square, which a few matrices extra at the end. I'll go with 4 rows of columns with 6 matrices each and an extra column at the end for the last 2 matrices if necessary.

To do this, I just use the index of the current matrix to set the column for that matrix. Qt5's grid layout handles spacing and creating extra columns for me, so I just have to work out which column to put each matrix in.

```
# 8e0f2dfe27fb9629b0f8cd33a25a61f85b5443e8
# src/lintrans/gui/dialogs/misc.py

102     class InfoPanelDialog(FixedSizeDialog):
...
105         def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
...
120             for i, name_value_pair in enumerate(self.wrapper.get_defined_matrices()):
121                 name, value = name_value_pair
122
123                 label_name = QLabel(self)
124                 label_name.setText(name)
125                 label_name.setFont(bold_font)
126
127                 label_equals = QLabel(self)
128                 label_equals.setText('=')
```

```
129
130                widget_matrix = self._get_matrix_widget(value)
131
132                column = 3 * (i // 6)
133
134                grid_layout.addWidget(
135                    label_name,
136                    i - 2 * column,
137                    column,
138                    Qt.AlignCenter
139                )
140                grid_layout.addWidget(
141                    label_equals,
142                    i - 2 * column,
143                    column + 1,
144                    Qt.AlignCenter
145                )
146                grid_layout.addWidget(
147                    widget_matrix,
148                    i - 2 * column,
149                    column + 2,
150                    Qt.AlignCenter
151                )
```
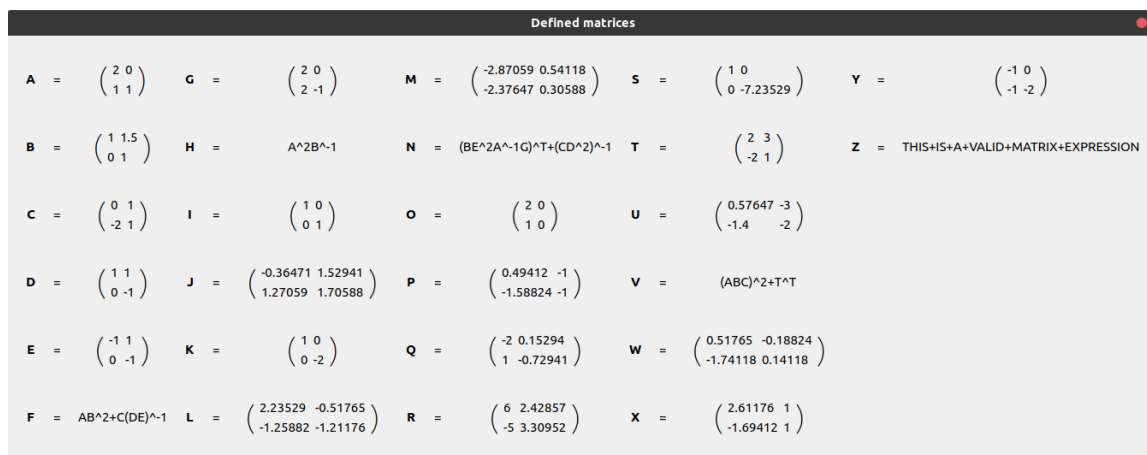

Figure 3.11.4: The info panel with all matrices defined, arranged in multiple columns

3.11.6 Fixing a bug with matrices with a 0 column

Currently, if you set one basis vector to **0**, then it will continue to draw the line for the zeroed basis vector. This doesn't make much sense, since the non-zero basis vector is the only one that has an impact on the matrix. To fix this, I can just check for a zero column and not draw the vector lines in that case.

```
       # 37abc3dbb5d99c9ec812e6c887fc6c021f9c91c6
       # src/lintrans/gui/plots/classes.py

171    class VectorGridPlot(BackgroundPlot):
...
231        def draw_parallel_lines(self, painter: QPainter, vector: Tuple[float, float], point: Tuple[float, float]) ->
           ↪    None:
...
244            # If the determinant is 0
245            if abs(vector_x * point_y - vector_y * point_x) < 1e-12:
246                rank = np.linalg.matrix_rank(
247                    np.array([
248                        [vector_x, point_x],
249                        [vector_y, point_y]
250                    ])
```

```
251                    )
252
253                # If the matrix is rank 1, then we can draw the column space line
254                if rank == 1:
255                    # If the vector does not have a 0 x or y component, then we can just draw the line
256                    if abs(vector_x) > 1e-12 and abs(vector_y) > 1e-12:
257                        self.draw_oblique_line(painter, vector_y / vector_x, 0)
258
259                    # Otherwise, we have to draw lines along the axes
260                    elif abs(vector_x) > 1e-12 and abs(vector_y) < 1e-12:
261                        painter.drawLine(0, self.height() // 2, self.width(), self.height() // 2)
262
263                    elif abs(vector_x) < 1e-12 and abs(vector_y) > 1e-12:
264                        painter.drawLine(self.width() // 2, 0, self.width() // 2, self.height())
265
266                    # If the vector is (0, 0), then don't draw a line for it
267                    else:
268                        return
269
270                # If the rank is 0, then we don't draw any lines
271                else:
272                    return
```


Figure 3.11.5: Before the bug fix
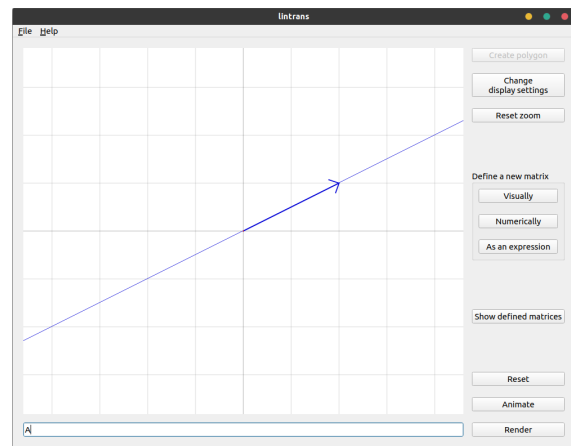
Figure 3.11.6: After the bug fix

4   Post-development Testing

Test ID	Description	Test data	Expected outcome	Actual outcome	Evidence	Comments
T1	The program should run in the Python interpreter and compile with PyInstaller.	N/A	The program runs and compiles with no problem.	As expected.	See Figure 4.1.	None.
T2	The user should be able to define matrices visually.	Approximately $\begin{pmatrix} 2 & 0.5 \\ 1 & 2 \end{pmatrix}$	The matrix should be successfully defined by the visual definition dialog.	The definition works fine with no errors and the defined matrix appears in the info panel.	See Figures 4.2 and 4.4	It's hard to correctly line up the j vector since it's $\begin{pmatrix} 0.5 \\ 2 \end{pmatrix}$ and the dialog only snaps to integer coordinates, but defining i was easy because of this.
T3	The user should be able to define matrices numerically.	$\begin{pmatrix} 2.2 & 18 \\ -0.4529 & 0 \end{pmatrix}$	The matrix should be successfully defined by the numerical definition dialog.	The definition works fine with no errors and the defined matrix appears in the info panel.	See Figures 4.3 and 4.4	Inputting the numbers directly makes it easy to get exactly what you want (such as precise decimals and large numbers that can be inconvenient to zoom out to find in the visual definition dialog) but there's no support for mathematical expressions or fractions, so the user is limited to decimal numbers.
T4	The program should be able to render matrices that the user has defined.	Rendering the previously defined **A** and then **B**.	The program renders the matrices correctly.	As expected.	See Figures 4.5 and 4.6.	Both the button and hotkey work as expected, as does the reset button.
T5	The program should be able to animate matrices that the user has defined.	Resetting, then animating **A**, then resetting and animating **B**>	The program animates smoothly from I to the matrices without any issues.	As expected.	See `pdtest/animate_defined.mp4` in the videos folder.	This test was performed with the default display settings.

T6	The program should be able to render and animate arbitrary rotations created with the `rot()` command.	Trying to render and animate `rot(135)`.	The program renders and animates the rotations correctly.	As expected.	See `pdtest/` `render_and_` `animate_` `rotations.mp4` in the videos folder.	None.
T7	The program should be able to render and animate expressions containing previously defined matrices.	Trying to render and animate **AB** and $\mathbf{A}^{-1}\mathbf{B}^2$.	The program renders and animates the expressions correctly.	As expected.	See `pdtest/` `render_and_` `animate_` `expressions.` `mp4` in the videos folder.	None.
T8	The program should be able to animate one matrix and then animate a transition to another.	Trying to render and animate **A** and then **B**.	The program animates them correctly, transitioning between them.	As expected.	See `pdtest/` `animate_` `multiple_` `transitional.` `mp4` in the videos folder.	None.
T9	The program should be able to animate one matrix and then apply another matrix to it.	I will start at **I**, then animate **B**, then apply **A**. This should be equivalent to **AB**, so I will apply $(\mathbf{AB})^{-1}$ to undo it.	The program animates everything correctly.	As expected.	See `pdtest/` `animate_` `multiple_` `applicative.` `mp4` in the videos folder.	None.
T10	The program should be able to animate a sequence of expressions using transitional animation.	I will animate the expression `"(AB)^-1,A,B"` and it will transition between each expression, right-to-left.	The program animates each expression in turn, transitioning between them.	As expected.	See `pdtest/` `animate_` `sequence_` `transitional.` `mp4` in the videos folder.	None.

T11	The program should be able to animate a sequence of expressions using applicative animation.	I will animate the expression `"(AB)^-1,A,B"` and it will apply each matrix in turn. The `"(AB)^-1"` at the end will undo the others and return it to the start.	The program animates each expression in turn, applying them from right to left.	As expected.	See `pdtest/ animate_ sequence_ applicative. mp4` in the videos folder.	None.
T12	The program should allow the user to define a matrix as an expression in terms of other matrices.	$\mathbf{C} = 0.2\mathbf{A}^2\mathbf{B}^{-1}$	The matrix $\mathbf{C}$ will be defined as this expression and render properly, as well as showing up in the info panel.	As expected.	See Figures 4.7, 4.8, 4.9 and `pdtest/ expression.mp4` in the videos folder.	None.
T13	When a matrix is defined as an expression, it should be re-calculated every time it's used. That means that if the user redefines one of the dependencies of an expression matrix, the expression matrix should update to reflect that the next time it's used.	$\mathbf{C} = 0.2\mathbf{A}^2\mathbf{B}^{-1}$ and then I will change $\mathbf{A}$ and re-render $\mathbf{C}$.	$\mathbf{C}$ will be rendered correctly both times, and both will be different.	As expected.	See `pdtest/ change_ expression_ dependencies. mp4` in the videos folder.	None.

| T14 | A key part of the display settings is the determinant parallelogram. It should work correctly. | I will enable the determinant parallelogram and try a variety of inputs, including compositions of matrices, expression-based matrices, and a rotation. | The matrix inputs should all be animated correctly, the determinant parallelogram will be visible for all of them, smoothed out during animations, blue for positive determinants, and red for negative determinants. | As expected. | See pdtest/ determinant_ parallelogram. mp4 in the videos folder. | None. |
| T15 | A key part of the display settings is displaying eigenvectors and eigenlines. This feature should work correctly. | I will enable eigenvectors and eigenlines and then animate each matrix in turn, resetting between each of them. | Each matrix should be animated correctly and the eigenvectors and eigenlines should be rendered properly at every stage. | As expected. | See pdtest/ eigenlines.mp4 in the videos folder. | Eigenvectors are discontinuous over animation. This means that the eigenvectors start aligned to the axes, and then snap to where they should be on the first frame of animation. This doesn't look great, but there's not much I can do to get around it. |
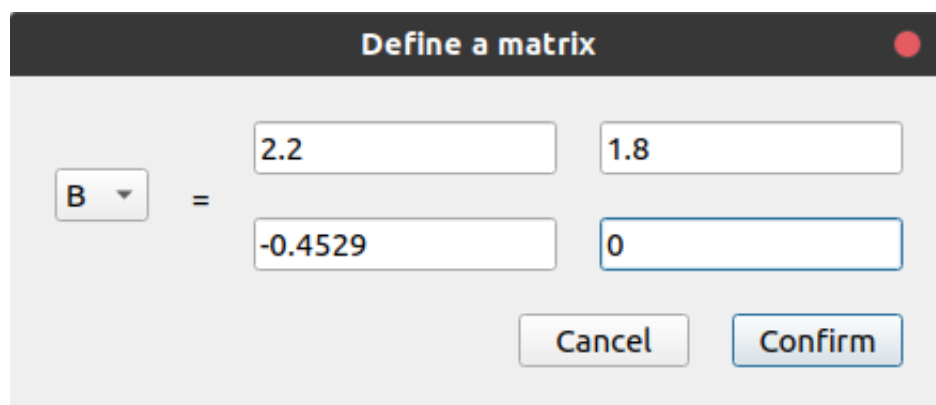
```
(lintrans) dyson@Harold-Ubuntu:~/repos/lintrans [main =]
$ ./compile.py
Created Compiler(filename=lintrans, version_name=0.3.0-alpha, platform=linux)
Compiling for platform=linux
503 INFO: PyInstaller: 5.5
503 INFO: Python: 3.11.0
504 INFO: Platform: Linux-5.19.0-35-generic-x86_64-with-glibc2.35
505 INFO: wrote /home/dyson/repos/lintrans/lintrans.spec
507 INFO: UPX is available.
507 INFO: Removing temporary files and cleaning cache in /home/dyson/.cache/pyinstaller
548 INFO: Extending PYTHONPATH with paths
['/home/dyson/repos/lintrans/src']
821 INFO: checking Analysis
821 INFO: Building Analysis because Analysis-00.toc is non existent
821 INFO: Initializing module dependency graph...
822 INFO: Caching module graph hooks...
826 WARNING: Several hooks defined for module 'numpy'. Please take care they do not conflict.
827 INFO: Analyzing base_library.zip ...
1798 INFO: Loading module hook 'hook-heapq.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks'...
2095 INFO: Loading module hook 'hook-encodings.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks'...
3566 INFO: Loading module hook 'hook-pickle.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks'...
5139 INFO: Caching module dependency graph...
5221 INFO: running Analysis Analysis-00.toc
5233 INFO: Analyzing /home/dyson/repos/lintrans/src/lintrans/__main__.py
5284 INFO: Loading module hook 'hook-numpy.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/numpy/_pyinstaller'...
5650 INFO: Loading module hook 'hook-numpy._pytesttester.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks'...
5923 INFO: Loading module hook 'hook-difflib.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks'...
6130 INFO: Loading module hook 'hook-multiprocessing.util.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks'...
6288 INFO: Loading module hook 'hook-xml.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks'...
6971 INFO: Loading module hook 'hook-platform.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks'...
7262 INFO: Loading module hook 'hook-sysconfig.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks'...
8588 INFO: Processing module hooks...
8638 INFO: Loading module hook 'hook-PyQt5.QtWidgets.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks'...
8848 INFO: Loading module hook 'hook-PyQt5.QtCore.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks'...
8905 INFO: Loading module hook 'hook-PyQt5.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks'...
8905 WARNING: Hidden import "sip" not found!
8936 INFO: Loading module hook 'hook-PyQt5.QtGui.py' from '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks'...
9041 INFO: Looking for ctypes DLLs
9081 INFO: Analyzing run-time hooks ...
9085 INFO: Including run-time hook '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks/rthooks/pyi_rth_inspect.py'
9087 INFO: Including run-time hook '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks/rthooks/pyi_rth_subprocess.py'
9088 INFO: Including run-time hook '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks/rthooks/pyi_rth_pyqt5.py'
9089 INFO: Including run-time hook '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks/rthooks/pyi_rth_pkgutil.py'
9091 INFO: Including run-time hook '/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/hooks/rthooks/pyi_rth_multiprocessing.py'
9102 INFO: Looking for dynamic libraries
/home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/building/build_main.py:173: UserWarning: The numpy.array_api submodule is still experimental. See NEP 47.
  __import__(package)
2120 WARNING: Cannot find libicuuc.so.56 (needed by /home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyQt5/Qt5/plugins/wayland-shell-integration/../../lib/libicui18n.so.56)
2120 WARNING: Cannot find libicudata.so.56 (needed by /home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyQt5/Qt5/plugins/wayland-shell-integration/../../lib/libicui18n.so.56)
2314 WARNING: Cannot find libicudata.so.56 (needed by /home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyQt5/Qt5/plugins/wayland-shell-integration/../../lib/libicuuc.so.56)
3346 WARNING: Cannot find libquadmath-96973f99.so.0.0.0 (needed by /home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/numpy/core/../../numpy.libs/libgfortran-040039e1.so.5.0.0)
3358 WARNING: Cannot find libgfortran-040039e1.so.5.0.0 (needed by /home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/numpy/core/../../numpy.libs/libopenblas64_p-r0-742d56dc.3.20.so)
12731 INFO: Looking for eggs
12731 INFO: Using Python library /home/dyson/.local/lib/libpython3.11.so.1.0
12736 INFO: Warnings written to /home/dyson/repos/lintrans/build/lintrans/warn-lintrans.txt
12770 INFO: Graph cross-reference written to /home/dyson/repos/lintrans/build/lintrans/xref-lintrans.html
12798 INFO: checking PYZ
12798 INFO: Building PYZ because PYZ-00.toc is non existent
12798 INFO: Building PYZ (ZlibArchive) /home/dyson/repos/lintrans/build/lintrans/PYZ-00.pyz
13312 INFO: Building PYZ (ZlibArchive) /home/dyson/repos/lintrans/build/lintrans/PYZ-00.pyz completed successfully.
13318 INFO: checking PKG
13318 INFO: Building PKG because PKG-00.toc is non existent
13318 INFO: Building PKG (CArchive) lintrans.pkg
63256 INFO: Building PKG (CArchive) lintrans.pkg completed successfully.
63260 INFO: Bootloader /home/dyson/repos/lintrans/venv/lib/python3.11/site-packages/PyInstaller/bootloader/Linux-64bit-intel/run
63260 INFO: checking EXE
63261 INFO: Building EXE because EXE-00.toc is non existent
63261 INFO: Building EXE from EXE-00.toc
63261 INFO: Copying bootloader EXE to /home/dyson/repos/lintrans/dist/lintrans
63261 INFO: Appending PKG archive to custom ELF section in EXE
64111 INFO: Building EXE from EXE-00.toc completed successfully.
Compilation finished
Auxiliary files cleaned up
```

Figure 4.1: The successful output of compilation
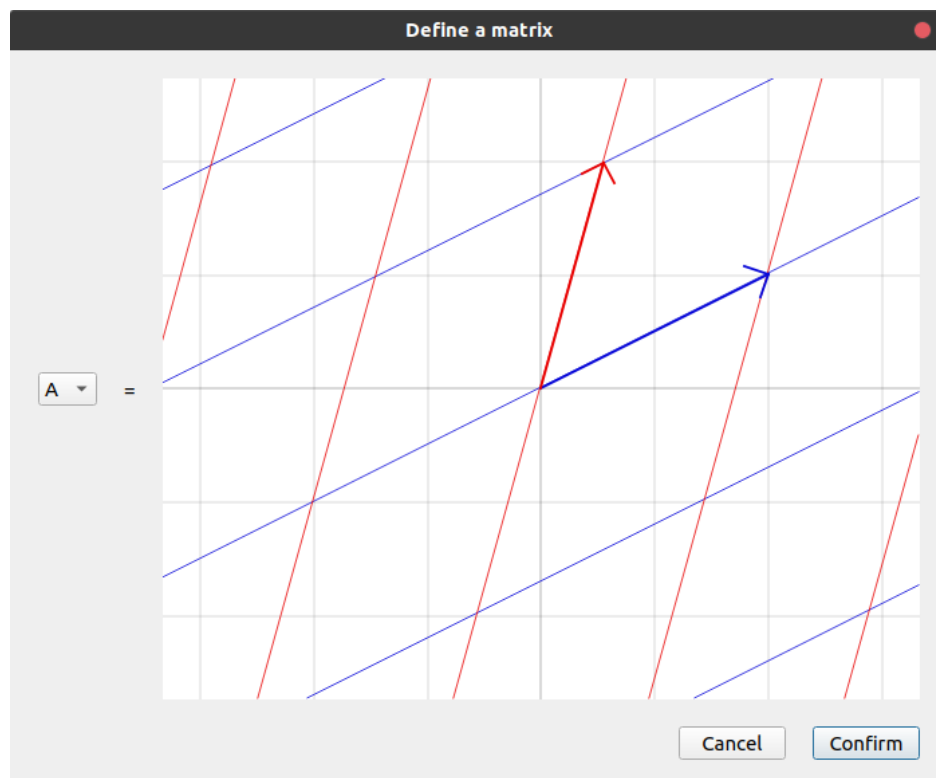
Figure 4.2: The matrix **A** being defined visually

Figure 4.3: The matrix **B** being defined numerically

Figure 4.4: The info panel after defining matrices **A** and **B**
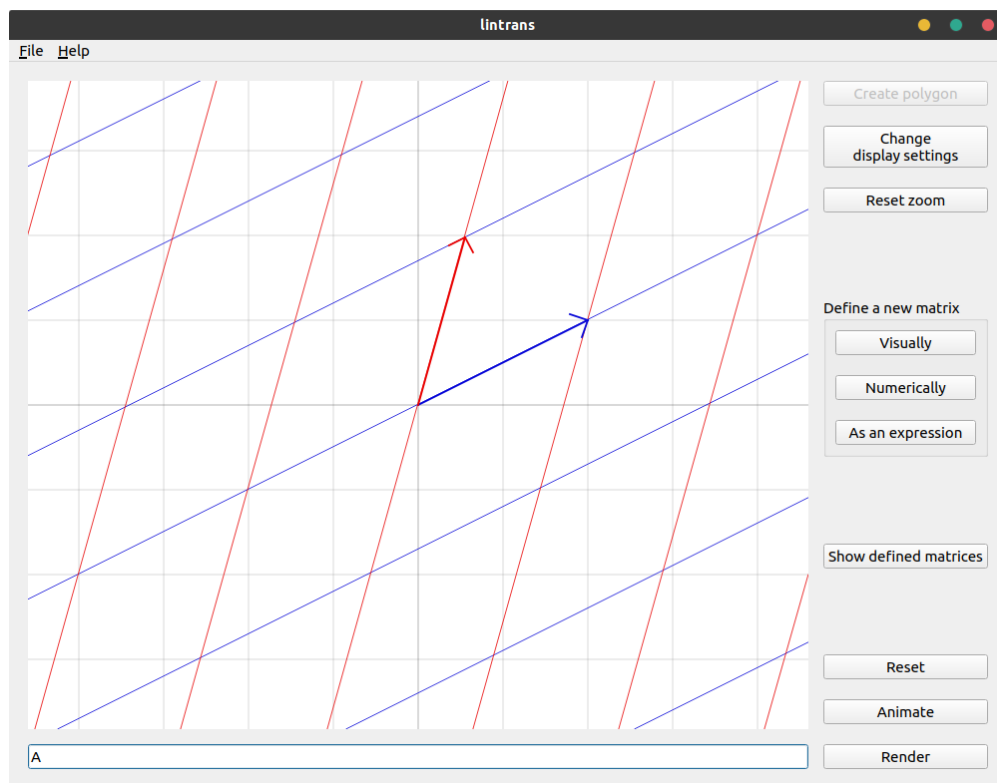
Figure 4.5: The matrix **A** being rendered
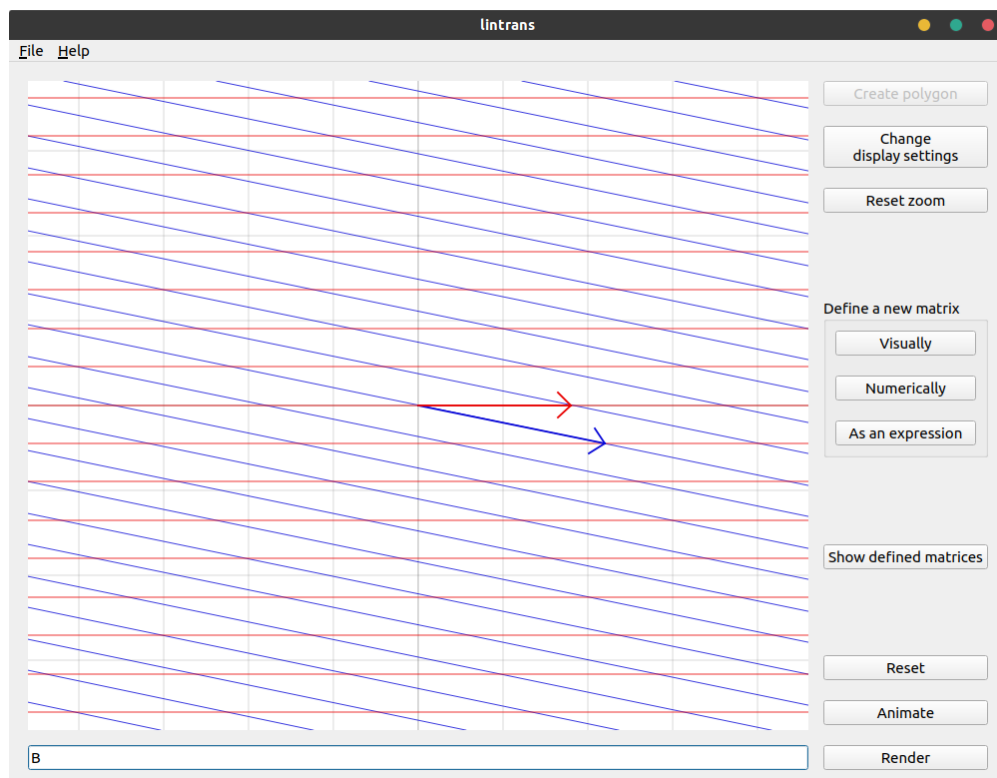
Figure 4.6: The matrix **B** being rendered

Define a matrix

C ▾   =   0.2A^2B^-1|

Cancel   Confirm
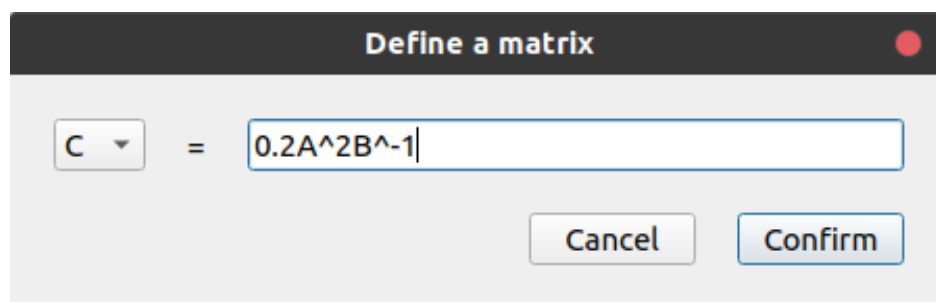
Figure 4.7: The expression matrix **C** being defined

Defined matrices
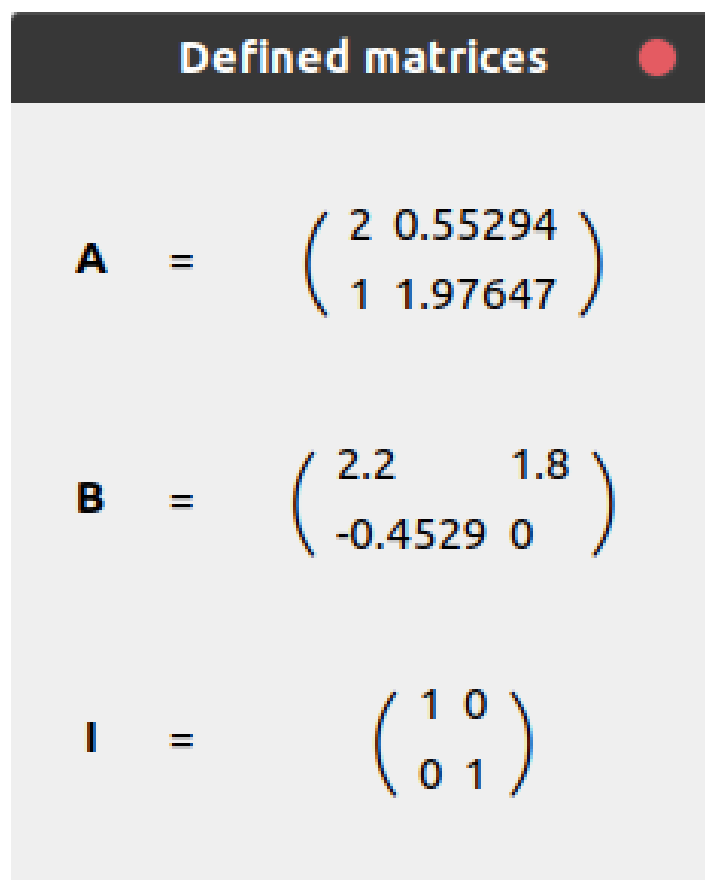
$$A = \begin{pmatrix} 2 & 0.55294 \\ 1 & 1.97647 \end{pmatrix}$$

$$B = \begin{pmatrix} 2.2 & 1.8 \\ -0.4529 & 0 \end{pmatrix}$$

$$C = 0.2A^2B^{-1}$$

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$
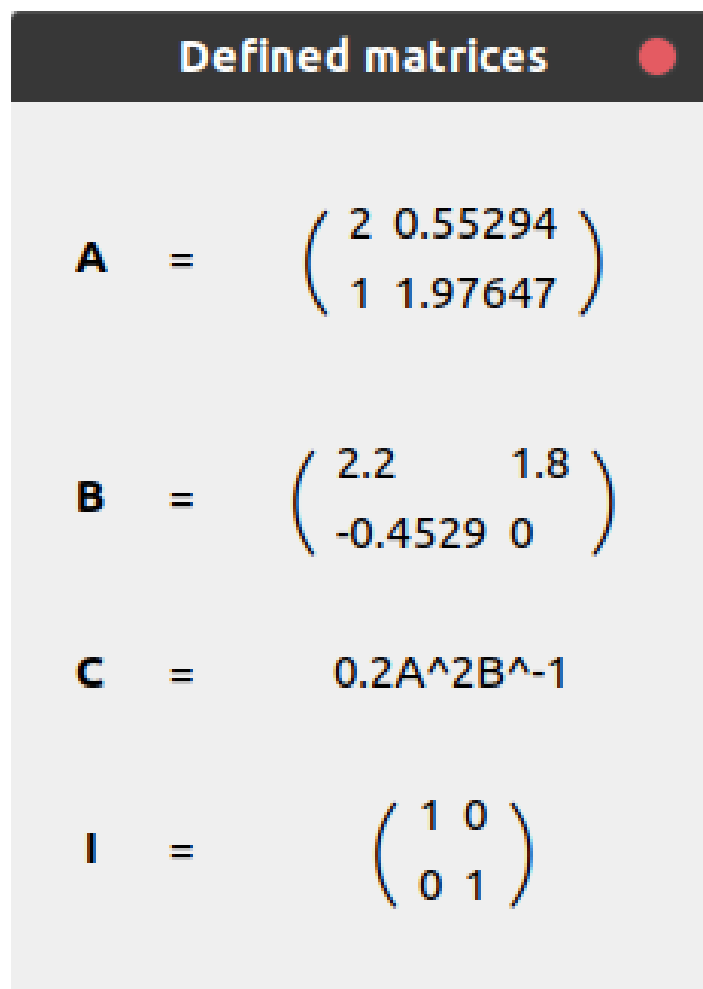
Figure 4.8: The info panel after defining the new matrix **C**
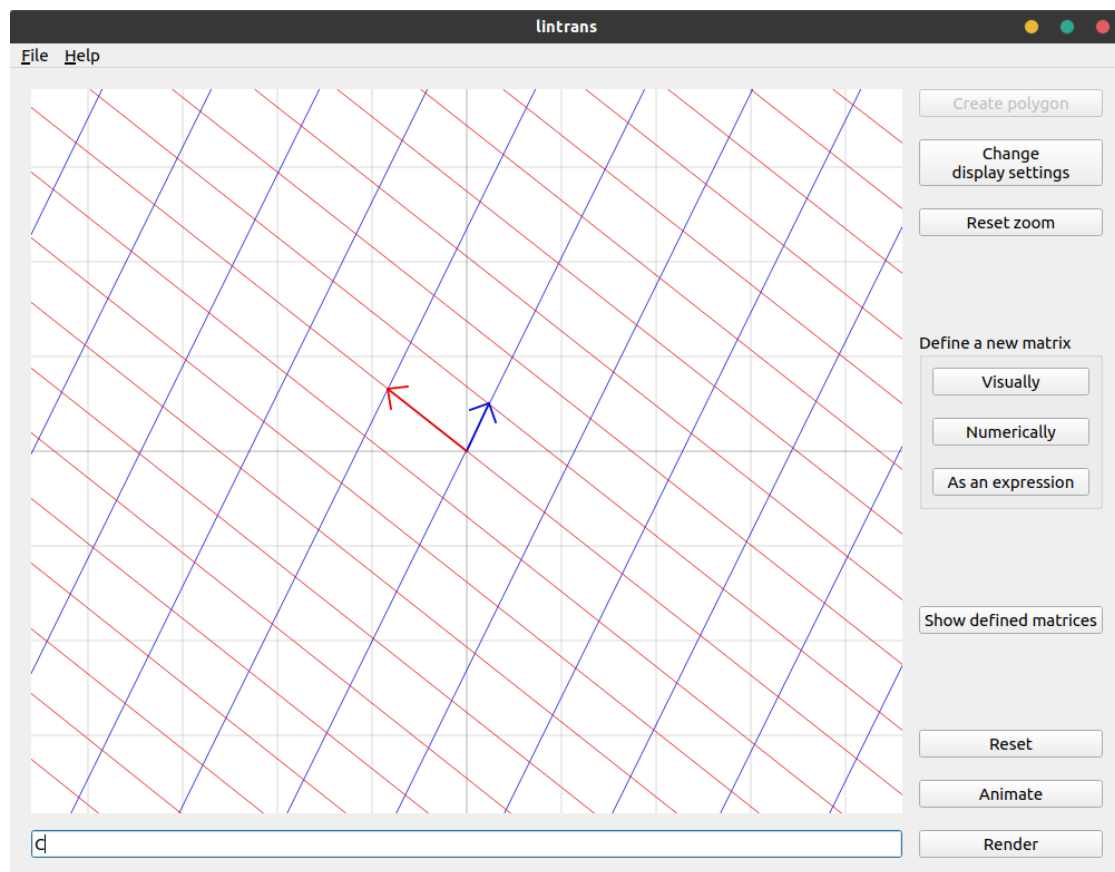
Figure 4.9: The matrix **C** being rendered

5   Evaluation

5.1   Evaluating success criteria

In §1.7, I laid out the success criteria for the project. It is now time to evaluate how well I met these criteria.

5.1.1   Defining multiple matrices

Criterion 1 says that the user must be able to define matrices in at least two different ways. I have been using these dialogs during development and they're quite useful. I would say I have completely fulfilled this criterion.
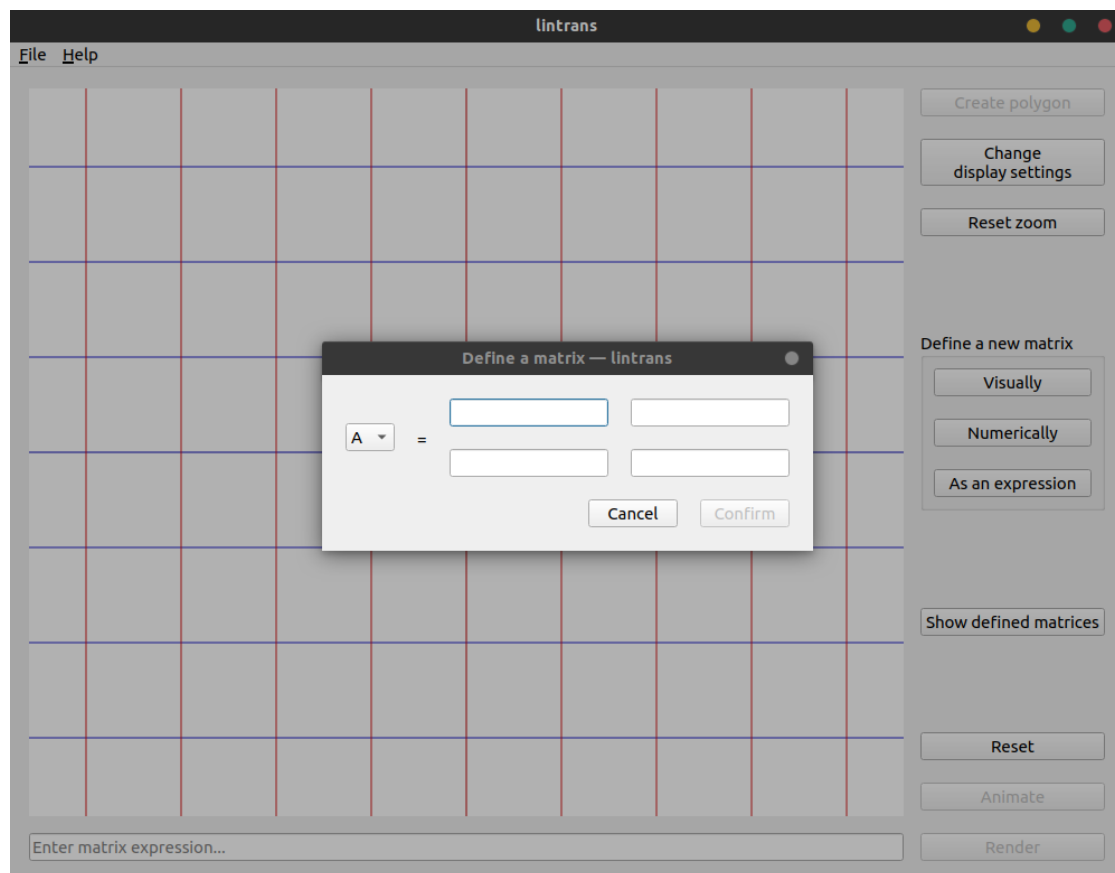
Figure 5.1.1: The numerical definition dialog
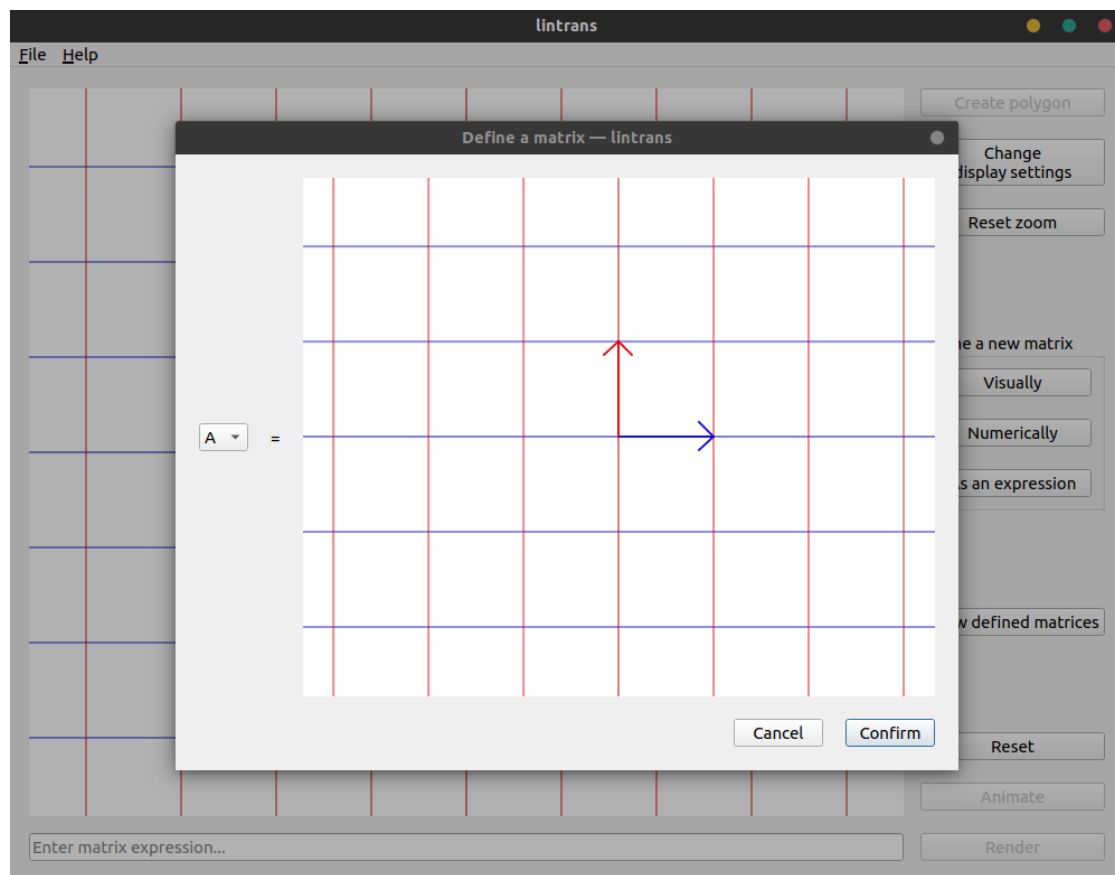
Figure 5.1.2: The visual definition dialog

5.1.2   Validating and parsing matrix expressions

Criterion 2 says that the program must be able to validate arbitrary matrix expressions and criterion 3 says that it must be able to parse any valid expression. As shown in §3.1.2, §3.1.4, and §3.10.7, the parser and validator are both complete and robust. I have fulfilled these success criteria.

5.1.3   Rendering any valid expression

Criterion 4 says that the program must be able to render any valid matrix expression. Rendering is quite simple and I got it sorted early on in §3.3.3 and §3.3.4. I have successfully fulfilled this criterion.

5.1.4   Animating any valid expression

Criterion 5 says that the program must be able to animate from **I** to any valid expression. I expected animation to be difficult but it was surprisingly easy. I just had to draw multiple frames using a `for` loop. I have covered animation in §3.3.5, §3.3.6, and §3.10.9. I have fulfilled this criterion.

5.1.5   Applicative animation

Criterion 6 says that the program must be able to apply a matrix transformation to the current scene. This is essentially the crux of the app. I was initially inspired to create `lintrans` because I wanted a way to demonstrate the non-commutativity of matrix multiplication ($\mathbf{AB} \neq \mathbf{BA}$). For me, the best

demonstration of this fact would be to show that applying two different matrices in different orders produces different results. Once I had the basis of an animation system, making it applicative wasn't as difficult as I expected.

I laid the groundwork for applicative animation in §3.4.9 and §3.4.10 and then I implemented applicative animation by just allowing the user to change the display settings in §3.5.1. I have fulfilled this criterion.

5.1.6   Display matrix info

Criterion 7 says that the program should be able to display information about the currently displayed matrix. I added display settings in §3.5 and these included things like determinants. I then polished determinant text in §3.6.8 and added eigenvectors and eigenlines in §3.7. I also added settings for animation timings in §3.5.1 and §3.10.10.

I'm quite happy with these display settings. I think being able to change colours and label the basis vectors i and j would have been nice, but I got the important parts done. I have fulfilled this criterion.
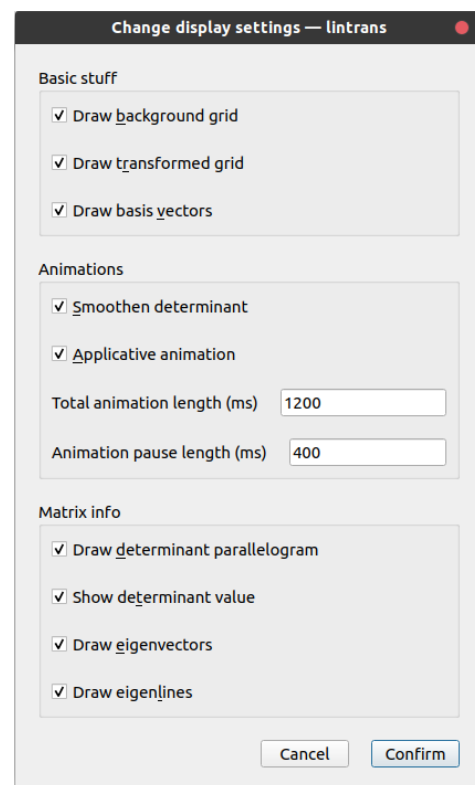
Figure 5.1.3: The display settings dialog
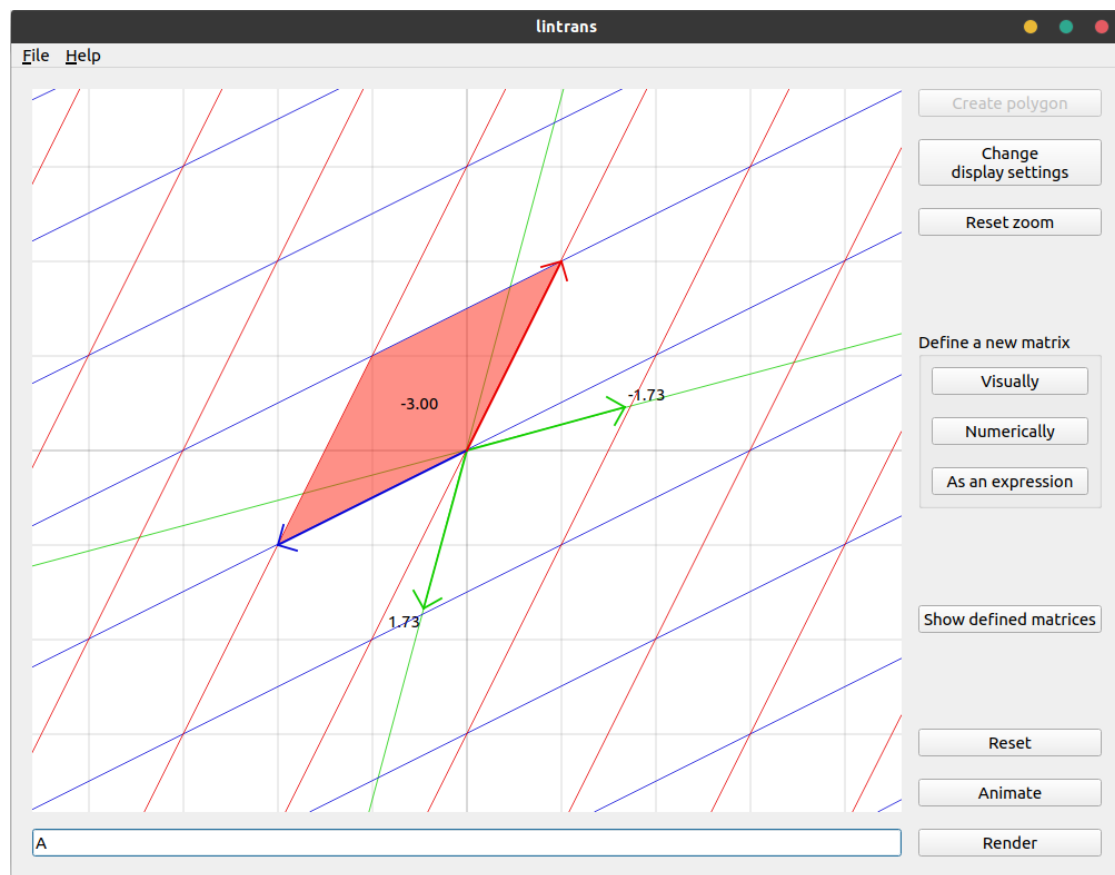
Figure 5.1.4: A matrix being rendered with all the display settings on

5.1.7   Saving and loading sessions

Criterion 8 says that the program should be able to save and load sessions. This was mostly a time restriction, but it shouldn't be particularly difficult to actually implement. I would create a dataclass to contain the data that we want to save, and use Python's `pickle` module to save it to a binary format for saving in a file. I failed to fulfil this criterion, mostly due to time.

5.1.8   Transforming polygons

Criterion 9 says that the program should allow the user to define and transform an arbitrary polygon. I was unable to complete this criterion. I'm not entirely sure how I would've done it. Obviously I'd have a dialog to define a polygon, and that would probably use a list of coordinates, which it would return to the main window when the user confirmed the dialog. Then I would just transform each coordinate one-by-one and draw the before and after versions. I failed to complete this criterion, again mostly due to running out of time.

Overall, I completed 7 out of my 9 criteria. I wish I'd had time to get the polygons done, and maybe I should've focussed on that instead of fixing bugs. But I'm happy with `lintrans` as it is now. I will probably add these missing features in a future release.

5.2   Assessing usability

The main usability features that I discussed in §2.4 were about colours, as well as layout and hotkeys.

The colours of the basis vectors are red and blue, as I said they should be. This allows users with common types of colour-blindness like deuteranopia to still distinguish parts of the UI. Refer to the images below to see that the colour scheme was a success[26]:
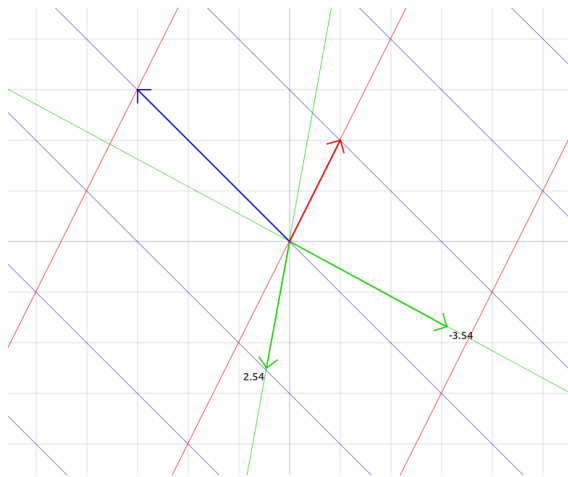
Figure 5.2.1: Normal vision
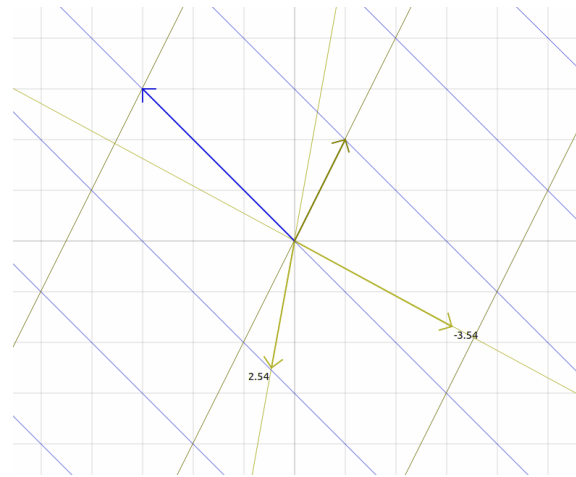
Figure 5.2.2: Deuteranopia (common)
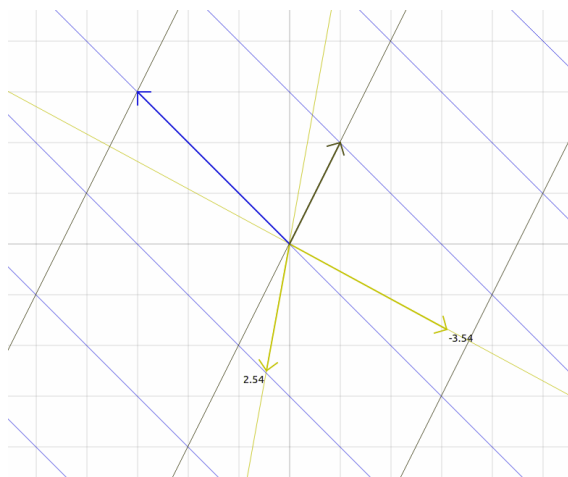
Figure 5.2.3: Protanopia (rare)
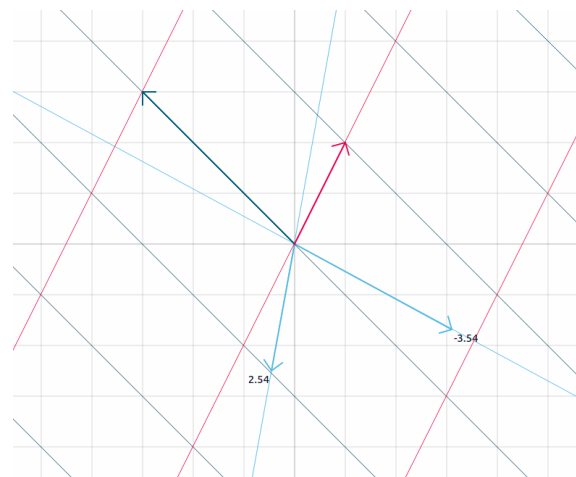
Figure 5.2.4: Tritanopia (very rare)

My other usability features were consistent UI layouts and hotkeys. All dialogs follow the same pattern of having the confirm button in the bottom right corner with the cancel button to the left of it, which is consistent with common menu design. I consider this a success.

Most of the functionality in the app has associated hotkeys. Rendering, animating, resetting, opening dialogs, and toggling display settings all have dedicated hotkeys. I forgot to add a shortcut to open the info panel, but that change is tiny, so I can just do it right now:

```
# 63f569328a1906976d725fa72da02e98e5d73afb
# src/lintrans/gui/main_window.py

39      class LintransMainWindow(QMainWindow):
...
45          def __init__(self):
...
208             self.button_info_panel.clicked.connect(
```

---

[26]These images were generated using my fork of Color Oracle[2].

```
209              # We have to use a lambda instead of 'InfoPanelDialog(self.matrix_wrapper, self).open' here
210              # because that would create an unnamed instance of InfoPanelDialog when LintransMainWindow is
211              # constructed, but we need to create a new instance every time to keep self.matrix_wrapper up to date
212              lambda: InfoPanelDialog(self.matrix_wrapper, self).open()
213          )
214          self.button_info_panel.setToolTip(
215              'Open an info panel with all matrices that have been defined in this session<br><b>(Ctrl + M)</b>'
216          )
217          QShortcut(QKeySequence('Ctrl+M'), self).activated.connect(self.button_info_panel.click)
```

I would consider my use of hotkeys to be a success.

5.3   Limitations

5.3.1   Feature completeness

As discussed in §5.1.7 and §5.1.8, `lintrans` is not feature complete according to my original success criteria. I attribute my failure to implement these features to me running out of time. If I focussed on implementing them instead of fixing bugs, then I definitely could've done them in time, but I wanted a more robust and bug-free app.

I think a more secure base allows for easier and more efficient expansion in the future. I can easily add these features in new versions, but I want to squash bugs early on so that I don't have to fix them later when the code base is larger. Overall, I wish I had more time to implement these missing features, but the core of the app is there, so I'll just add transforming arbitrary polygons and saving and loading sessions in the future.

5.3.2   Distribution

`lintrans` was surprisingly hard to distribute. I spend most of my time in the terminal, which makes it very easy to set up complete development environments and a virtual environment to manage the Python packages and interpreter needed to run `lintrans`. My stakeholders are teachers who don't even know what a shell is, let alone how to set up a virtual environment for Python. So I had to compile `lintrans` into a single Windows executable. That wasn't massively difficult, but it was certainly a headache. If I did this project again, I'd use a compiled language like Rust, Java, or C++, since those languages are much easier to distribute executables for.

Additionally, Rust can compile to WebAssembly[14], so I could make a web version of `lintrans`, which would avoid anyone having to download anything and would allow students to try it out very easily, perhaps even on their phones.

5.4   Maintenance

5.4.1   Class structure

As you can see from the following internal import graph and class diagram, the class structure isn't fantastic.
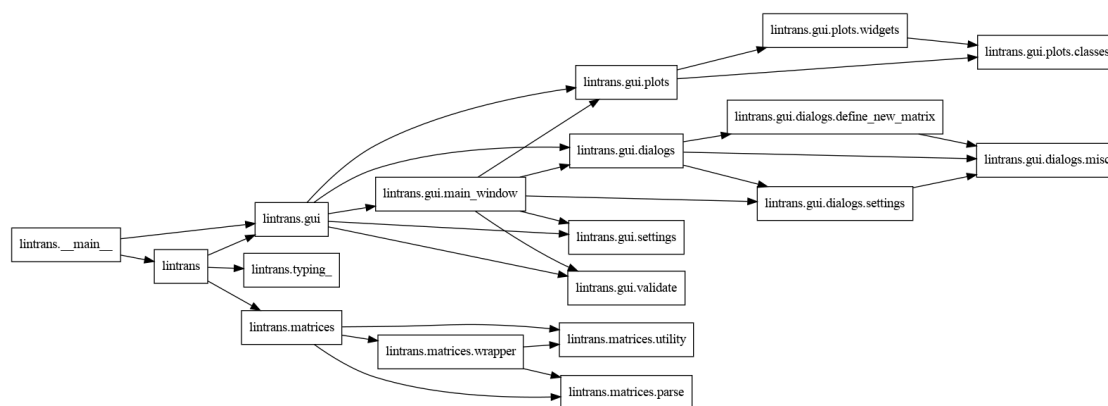
Figure 5.4.1: The final internal import graph
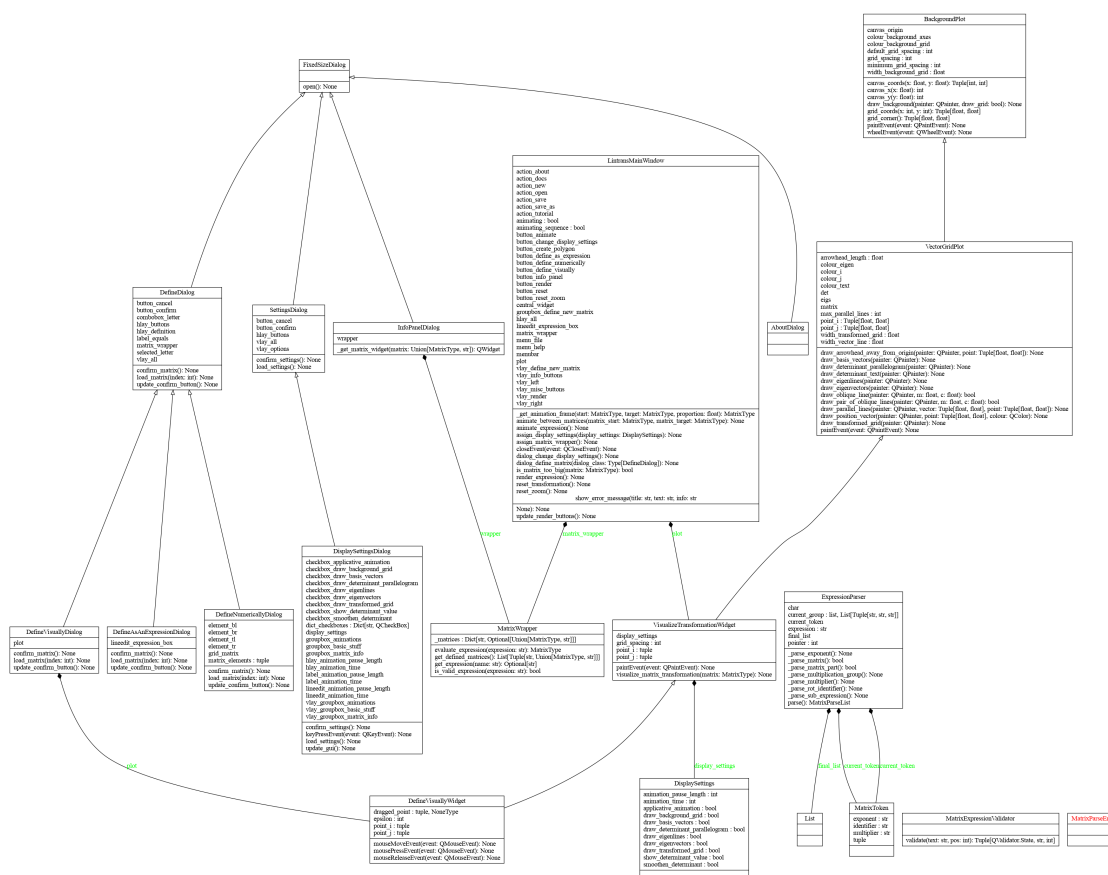
Figure 5.4.2: The final class diagram

However, I believe that the classes are well named and in files which are equally well named. This makes maintaining them easier than you might suspect by looking at these diagrams. Additionally, all my variables and functions have sensible names which identify their purpose. Each class, method, and function is small and individual. The only 'large' class is the main window, which makes sense because it has the most logic and is hard to split up.

Everything else is split into small enough logical chunks to make it pretty easy to maintain. I also have docstrings for every single class, method, and function, as well as comments to explain the complicated parts of algorithms. This annotation makes it much easier to remember what each thing does and how it does it.

Overall, there are a lot of classes and modules, but it's a complex program, so I think this is well justified. The logic is separate and modular.

5.4.2 Type safety

By far, my biggest maintainability problem is Python's lack of type safety. I use `mypy` to statically check the types of variables *before* runtime, but the Python interpreter makes no real guarantees about what the types of variables will be *at* runtime. This makes it much harder to debug things and introduces whole categories of bugs that would be impossible in a compiled, statically and strongly typed language like Rust, Java, or C++.

When I started `lintrans`, I didn't really care about types in a programming language and I was only really familiar with Python, so that's what I used. Since then, I've learned Rust and it's taught me the benefits of a static, strong type system. Rust doesn't even have `null` or exceptions. Python's `None` and hidden exception throwing have caused several bugs already and will undoubtedly cause many more in the future. With Rust, these would be impossible, but in Python, they are seemingly hiding in every dark corner, just waiting to crash the program in some weird edge case.

I will need to stay on top of these types of bugs when maintaining `lintrans`, unless I decide to rewrite the whole thing in a compiled language.

References

[1] Alan O'Callaghan (Alanocallaghan). *Color Oracle (dorkbox fork)*. Version 1.3. URL: `https://github.com/Alanocallaghan/color-oracle-java`.

[2] Dyson Dyson (DoctorDalek1963). *Color Oracle (my personal fork)*. Version 1.3. URL: `https://github.com/DoctorDalek1963/color-oracle-java`.

[3] Dyson Dyson (DoctorDalek1963). *lintrans*. URL: `https://github.com/DoctorDalek1963/lintrans`.

[4] Dyson Dyson (DoctorDalek1963). *Which framework should I use for creating draggable points and connecting lines on a 2D grid?* 26th Jan. 2022. URL: `https://www.reddit.com/r/learnpython/comments/sd2lbr`.

[5] Ross Wilson (rzzzwilson). *Python-Etudes/PyQtCustomWidget*. URL: `https://gitlab.com/rzzzwilson/python-etudes/-/tree/master/PyQtCustomWidget`.

[6] Ross Wilson (rzzzwilson). *Python-Etudes/PyQtCustomWidget - ijvectors.py*. 26th Jan. 2022. URL: `https://gitlab.com/rzzzwilson/python-etudes/-/blob/2b43f5d3c95aa4410db5bed77195bf242318a304/PyQtCustomWidget/ijvectors.py`.

[7] *2D linear transformation*. URL: `https://www.desmos.com/calculator/upooihuy4s`.

[8] *About Code Signing*. Apple Inc. URL: `https://developer.apple.com/library/archive/documentation/Security/Conceptual/CodeSigningGuide/Introduction/Introduction.html`.

[9] *About `Info.plist` Keys and Values*. Apple Inc. URL: `https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Introduction/Introduction.html#//apple_ref/doc/uid/TP40009248-SW1`.

[10] *About Information Property List Files*. Apple Inc. URL: `https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/AboutInformationPropertyListFiles.html#//apple_ref/doc/uid/TP40009254-SW1`.

[11] *Apple Developer Program. What You Need To Enroll*. Apple Inc. URL: `https://developer.apple.com/programs/enroll/`.

[12] David Beazley. *Modules and Packages: Live and Let Die!* PyCon 2015. 10th Apr. 2015. URL: `https://www.youtube.com/watch?v=0oTh1CXRaQ0`.

[13] *Bundle Structures*. Apple Inc. URL: `https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFBundles/BundleTypes/BundleTypes.html`.

[14] *Compiling from Rust to WebAssembly*. Mozilla. URL: `https://developer.mozilla.org/en-US/docs/WebAssembly/Rust_to_wasm`.

[15] *Fundamental theorem of algebra*. Wikipedia. URL: `https://en.wikipedia.org/wiki/Fundamental_theorem_of_algebra`.

[16] *git branch documentation*. Version 2.38.0. Software Freedom Conservancy. URL: `https://git-scm.com/docs/git-branch/2.38.0`.

[17] *GitHub Actions Documentation*. GitHub, Inc. URL: `https://docs.github.com/en/actions`.

[18] *GitHub Pages Documentation*. GitHub, Inc. URL: `https://docs.github.com/en/pages`.

[19] *GNU General Public License*. Version 3. Free Software Foundation. 29th June 2007. URL: `https://www.gnu.org/licenses/gpl-3.0.en.html`.

[20] David Goodger and Guido van Rossum. *PEP 257. Docstring Conventions*. 29th May 2001. URL: `https://peps.python.org/pep-0257/`.

[21] Grant Sanderson (3blue1brown). *Essence of Linear Algebra*. 6th Aug. 2016. URL: `https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab`.

[22] H. Hohn et al. *Matrix Vector*. MIT. 2001. URL: `https://mathlets.org/mathlets/matrix-vector/`.

[23] Jacek Wodecki and ekhumoro. *How to update window in PyQt5?* URL: `https://stackoverflow.com/questions/42045676/how-to-update-window-in-pyqt5`.

[24] je1324. *Visualizing Linear Transformations*. 15th Mar. 2018. URL: `https://www.geogebra.org/m/YCZa8TAH`.

[25] Holger Krekel and pytest-dev team. *pytest. helps you write better programs*. Version 7.0.x. URL: `https://docs.pytest.org/en/7.0.x/`.

[26] Holger Krekel and pytest-dev team. *pytest - How to run doctests*. Version 7.0.x. URL: `https://docs.pytest.org/en/7.0.x/how-to/doctest.html`.

[27] Olivier Lacan. *keep a changelog*. Version 1.0.0. URL: `https://keepachangelog.com/en/1.0.0/`.

[28] Nathaniel Vaughn Kelso and Bernie Jenny. *Color Oracle (original)*. Version 1.3. URL: `https://colororacle.org/`.

[29] *Normalize a matrix such that the determinat = 1*. ResearchGate. 26th June 2017. URL: `https://www.researchgate.net/post/normalize_a_matrix_such_that_the_determinat_1`.

[30] *NumPy `eig()` function*. Version 1.23. NumPy Developers. URL: `https://numpy.org/doc/1.23/reference/generated/numpy.linalg.eig.html`.

[31] *Nutika User Manual*. URL: `https://nuitka.net/doc/user-manual.html`.

[32] `pip install` *documentation. Editable option*. The pip developers. URL: `https://pip.pypa.io/en/stable/cli/pip_install/#cmdoption-e`.

[33] *Plotting with Matplotlib. Create PyQt5 plots with the popular Python plotting library*. URL: `https://www.pythonguis.com/tutorials/plotting-matplotlib/`.

[34] *PyInstaller Manual*. Version 4.10. URL: `https://pyinstaller.org/en/v4.10/`.

[35] *PyInstaller Manual. Capturing Windows Version Data*. Version 4.10. URL: `https://pyinstaller.org/en/v4.10/usage.html#capturing-windows-version-data`.

[36] *Pylint features (import graphs)*. Version 2.12.2. PyCQA. URL: `https://pylint.readthedocs.io/en/v2.12.2/technical_reference/features.html?highlight=graph#imports-checker-options`.

[37] *PyQt5 Graphics View Framework*. The Qt Company. URL: `https://doc.qt.io/qtforpython-5/overviews/graphicsview.html`.

[38] *Python 3 Data model - special methods*. Python Software Foundation. URL: `https://docs.python.org/3/reference/datamodel.html#special-method-names`.

[39] *Python 3 doctest module*. Python Software Foundation. URL: `https://docs.python.org/3/library/doctest.html#module-doctest`.

[40] *Python 3.10 Downloads*. Version 3.10. Python Software Foundation. URL: `https://www.python.org/downloads/release/python-3100/`.

[41] *Python `argparse` docs (`parse_known_args()` method)*. Version 3.10. Python Software Foundation. URL: `https://docs.python.org/3.10/library/argparse.html#argparse.ArgumentParser.parse_known_args`.

[42] *QGridLayout class*. The Qt Company. URL: `https://doc.qt.io/qt-5/qgridlayout.html#addWidget-2`.

[43] *Qt5 for Linux/X11*. The Qt Company. URL: `https://doc.qt.io/qt-5/linux.html`.

[44] *QValidator class*. The Qt Company. URL: `https://doc.qt.io/qt-5/qvalidator.html`.

[45] *QWheelEvent class*. The Qt Company. URL: `https://doc.qt.io/qt-5/qwheelevent.html`.

[46] *QWidget Class (`baseSize` property)*. The Qt Company. URL: `https://doc.qt.io/qt-5/qwidget.html#baseSize-prop`.

[47] *QWidget Class (`mouseMoveEvent()` method*. The Qt Company. URL: `https://doc.qt.io/qt-5/qwidget.html#mouseMoveEvent`.

[48] *QWidget Class (`repaint()` method)*. The Qt Company. URL: `https://doc.qt.io/qt-5/qwidget.html#repaint`.

[49] *QWidget Class (`size` property)*. The Qt Company. URL: `https://doc.qt.io/qt-5/qwidget.html#size-prop`.

[50] *QWidget Class (`update()` method)*. The Qt Company. URL: `https://doc.qt.io/qt-5/qwidget.html#update`.

[51] *Read the Docs*. Read the Docs, Inc. URL: `https://readthedocs.org/`.

[52] *Semantic Versioning*. Version 2.0.0. URL: `https://semver.org/spec/v2.0.0.html`.

[53] Shad Sharma. *Linear Transformation Visualizer*. 4th May 2017. URL: `https://shad.io/MatVis/`.

[54] Brian Skinn. *sphobjinv documentation*. Version 2.2. URL: `https://sphobjinv.readthedocs.io/en/v2.2/`.

[55] *Sphinx. Python Documentation Generator*. The Sphinx developers. URL: `https://www.sphinx-doc.org/en/master/`.

[56] Rod Stephens. *Draw lines with arrowheads in C#*. 5th Dec. 2014. URL: `http://csharphelper.com/howtos/howto_draw_arrows.html`.

[57] *The Event System*. The Qt Company. URL: `https://doc.qt.io/qt-5/eventsandfilters.html`.

[58] *Types of Color Blindness*. National Eye Institute. URL: `https://www.nei.nih.gov/learn-about-eye-health/eye-conditions-and-diseases/color-blindness/types-color-blindness`.

[59] *Unit testing*. Wikipedia. URL: `https://en.wikipedia.org/wiki/Unit_testing`.

[60] *UPX. The Ultimate Packer for eXecutables*. URL: `https://github.com/upx/upx`.

[61] *Version Information Structures*. Microsoft. URL: `https://learn.microsoft.com/en-gb/windows/win32/menurc/version-information-structures`.

[62] *VERSIONINFO Resource*. Microsoft. URL: `https://learn.microsoft.com/en-gb/windows/win32/menurc/versioninfo-resource`.

A   Project code

A.1   `__init__.py`

```
 1    # lintrans - The linear transformation visualizer
 2    # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
 3
 4    # This program is licensed under GNU GPLv3, available here:
 5    # <https://www.gnu.org/licenses/gpl-3.0.html>
 6
 7    """This is the top-level ``lintrans`` package, which contains all the subpackages of the project."""
 8
 9    from . import gui, matrices, typing_
10
11    __version__ = '0.3.0-alpha'
12
13    __all__ = ['gui', 'matrices', 'typing_', '__version__']
```

A.2   `__main__.py`

```
 1    #!/usr/bin/env python
 2
 3    # lintrans - The linear transformation visualizer
 4    # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
 5
 6    # This program is licensed under GNU GPLv3, available here:
 7    # <https://www.gnu.org/licenses/gpl-3.0.html>
 8
 9    """This module provides a :func:`main` function to interpret command line arguments and run the program."""
10
11    import sys
12    from argparse import ArgumentParser
13    from textwrap import dedent
14    from typing import List
15
16    from lintrans import __version__, gui
17
18
19    def main(args: List[str]) -> None:
20        """Interpret program-specific command line arguments and run the main window in most cases.
21
22        If the user supplies --help or --version, then we simply respond to that and then return.
23        If they don't supply either of these, then we run :func:`lintrans.gui.main_window.main`.
24
25        :param List[str] args: The full argument list (including program name)
26        """
27        parser = ArgumentParser(add_help=False)
28
29        parser.add_argument(
30            '-h',
31            '--help',
32            default=False,
33            action='store_true'
34        )
35
36        parser.add_argument(
37            '-V',
38            '--version',
39            default=False,
40            action='store_true'
41        )
42
43        parsed_args, unparsed_args = parser.parse_known_args()
44
45        if parsed_args.help:
46            print(dedent('''
47            Usage: lintrans [option]
48
```

```
49                 Options:
50                     -h, --help        Display this help text and exit
51                     -V, --version     Display the version information and exit
52
53                 Any other options will get passed to the QApplication constructor.
54                 If you don't know what that means, then don't provide any arguments and just the run the program.'''[1:]))
55                 return
56
57         if parsed_args.version:
58             print(dedent(f'''
59             lintrans (version {__version__})
60             The linear transformation visualizer
61
62             Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
63
64             This program is licensed under GNU GPLv3, available here:
65             <https://www.gnu.org/licenses/gpl-3.0.html>'''[1:]))
66             return
67
68         for arg in unparsed_args:
69             print(f'Passing "{arg}" to QApplication. See --help for recognised args')
70
71         gui.main(args[:1] + unparsed_args)
72
73
74     if __name__ == '__main__':
75         main(sys.argv)
```

A.3 `gui/__init__.py`

```
1     # lintrans - The linear transformation visualizer
2     # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4     # This program is licensed under GNU GPLv3, available here:
5     # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7     """This package supplies the main GUI and associated dialogs for visualization."""
8
9     from . import dialogs, plots, settings, validate
10    from .main_window import main
11
12    __all__ = ['dialogs', 'main', 'plots', 'settings', 'validate']
```

A.4 `gui/main_window.py`

```
1     # lintrans - The linear transformation visualizer
2     # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4     # This program is licensed under GNU GPLv3, available here:
5     # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7     """This module provides the :class:`LintransMainWindow` class, which provides the main window for the GUI."""
8
9     from __future__ import annotations
10
11    import re
12    import sys
13    import webbrowser
14    from copy import deepcopy
15    from typing import List, Tuple, Type
16
17    import numpy as np
18    from numpy import linalg
19    from numpy.linalg import LinAlgError
20    from PyQt5 import QtWidgets
21    from PyQt5.QtCore import pyqtSlot, QCoreApplication, QThread
22    from PyQt5.QtGui import QCloseEvent, QKeySequence
23    from PyQt5.QtWidgets import (QApplication, QHBoxLayout, QMainWindow, QMessageBox,
```

```
24                                   QShortcut, QSizePolicy, QSpacerItem, QStyleFactory, QVBoxLayout)
25
26      import lintrans
27      from lintrans.matrices import MatrixWrapper
28      from lintrans.matrices.parse import validate_matrix_expression
29      from lintrans.matrices.utility import polar_coords, rotate_coord
30      from lintrans.typing_ import MatrixType, VectorType
31      from .dialogs import (AboutDialog, DefineAsAnExpressionDialog, DefineDialog,
32                            DefineNumericallyDialog, DefineVisuallyDialog, InfoPanelDialog)
33      from .dialogs.settings import DisplaySettingsDialog
34      from .plots import VisualizeTransformationWidget
35      from .settings import DisplaySettings
36      from .validate import MatrixExpressionValidator
37
38
39      class LintransMainWindow(QMainWindow):
40          """This class provides a main window for the GUI using the Qt framework.
41
42          This class should not be used directly, instead call :func:`lintrans.gui.main_window.main` to create the GUI.
43          """
44
45          def __init__(self):
46              """Create the main window object, and create and arrange every widget in it.
47
48              This doesn't show the window, it just constructs it.
49              Use :func:`lintrans.gui.main_window.main` to show the GUI.
50              """
51              super().__init__()
52
53              self.matrix_wrapper = MatrixWrapper()
54
55              self.setWindowTitle('lintrans')
56              self.setMinimumSize(1000, 750)
57
58              self.animating: bool = False
59              self.animating_sequence: bool = False
60
61              # === Create menubar
62
63              self.menubar = QtWidgets.QMenuBar(self)
64
65              self.menu_file = QtWidgets.QMenu(self.menubar)
66              self.menu_file.setTitle('&File')
67
68              self.menu_help = QtWidgets.QMenu(self.menubar)
69              self.menu_help.setTitle('&Help')
70
71              self.action_new = QtWidgets.QAction(self)
72              self.action_new.setText('&New')
73              self.action_new.setShortcut('Ctrl+N')
74              self.action_new.triggered.connect(lambda: print('new'))
75
76              self.action_open = QtWidgets.QAction(self)
77              self.action_open.setText('&Open')
78              self.action_open.setShortcut('Ctrl+O')
79              self.action_open.triggered.connect(lambda: print('open'))
80
81              self.action_save = QtWidgets.QAction(self)
82              self.action_save.setText('&Save')
83              self.action_save.setShortcut('Ctrl+S')
84              self.action_save.triggered.connect(lambda: print('save'))
85
86              self.action_save_as = QtWidgets.QAction(self)
87              self.action_save_as.setText('Save as...')
88              self.action_save_as.triggered.connect(lambda: print('save as'))
89
90              self.action_tutorial = QtWidgets.QAction(self)
91              self.action_tutorial.setText('&Tutorial')
92              self.action_tutorial.setShortcut('F1')
93              self.action_tutorial.triggered.connect(lambda: print('tutorial'))
94
95              self.action_docs = QtWidgets.QAction(self)
96              self.action_docs.setText('&Docs')
```

```
97
98              # If this is an old release, use the docs for this release. Else, use the latest docs
99              # We use the latest because most use cases for non-stable releases will be in development and testing
100             docs_link = 'https://lintrans.readthedocs.io/en/'
101
102             if re.match(r'^\d+\.\d+\.\d+$', lintrans.__version__):
103                 docs_link += 'v' + lintrans.__version__
104             else:
105                 docs_link += 'latest'
106
107             self.action_docs.triggered.connect(
108                 lambda: webbrowser.open_new_tab(docs_link)
109             )
110
111             self.action_about = QtWidgets.QAction(self)
112             self.action_about.setText('&About')
113             self.action_about.triggered.connect(lambda: AboutDialog(self).open())
114
115             # TODO: Implement these actions and enable them
116             self.action_new.setEnabled(False)
117             self.action_open.setEnabled(False)
118             self.action_save.setEnabled(False)
119             self.action_save_as.setEnabled(False)
120             self.action_tutorial.setEnabled(False)
121
122             self.menu_file.addAction(self.action_new)
123             self.menu_file.addAction(self.action_open)
124             self.menu_file.addSeparator()
125             self.menu_file.addAction(self.action_save)
126             self.menu_file.addAction(self.action_save_as)
127
128             self.menu_help.addAction(self.action_tutorial)
129             self.menu_help.addAction(self.action_docs)
130             self.menu_help.addSeparator()
131             self.menu_help.addAction(self.action_about)
132
133             self.menubar.addAction(self.menu_file.menuAction())
134             self.menubar.addAction(self.menu_help.menuAction())
135
136             self.setMenuBar(self.menubar)
137
138             # === Create widgets
139
140             # Left layout: the plot and input box
141
142             self.plot = VisualizeTransformationWidget(self, display_settings=DisplaySettings())
143
144             self.lineedit_expression_box = QtWidgets.QLineEdit(self)
145             self.lineedit_expression_box.setPlaceholderText('Enter matrix expression...')
146             self.lineedit_expression_box.setValidator(MatrixExpressionValidator(self))
147             self.lineedit_expression_box.textChanged.connect(self.update_render_buttons)
148
149             # Right layout: all the buttons
150
151             # Misc buttons
152
153             self.button_create_polygon = QtWidgets.QPushButton(self)
154             self.button_create_polygon.setText('Create polygon')
155             # self.button_create_polygon.clicked.connect(self.create_polygon)
156             self.button_create_polygon.setToolTip('Define a new polygon to view the transformation of')
157
158             # TODO: Implement this and enable button
159             self.button_create_polygon.setEnabled(False)
160
161             self.button_change_display_settings = QtWidgets.QPushButton(self)
162             self.button_change_display_settings.setText('Change\ndisplay settings')
163             self.button_change_display_settings.clicked.connect(self.dialog_change_display_settings)
164             self.button_change_display_settings.setToolTip(
165                 "Change which things are rendered and how they're rendered<br><b>(Ctrl + D)</b>"
166             )
167             QShortcut(QKeySequence('Ctrl+D'), self).activated.connect(self.button_change_display_settings.click)
168
169             self.button_reset_zoom = QtWidgets.QPushButton(self)
```

```
170              self.button_reset_zoom.setText('Reset zoom')
171              self.button_reset_zoom.clicked.connect(self.reset_zoom)
172              self.button_reset_zoom.setToolTip('Reset the zoom level back to normal<br><b>(Ctrl + Shift + R)</b>')
173              QShortcut(QKeySequence('Ctrl+Shift+R'), self).activated.connect(self.button_reset_zoom.click)
174
175              # Define new matrix buttons and their groupbox
176
177              self.button_define_visually = QtWidgets.QPushButton(self)
178              self.button_define_visually.setText('Visually')
179              self.button_define_visually.setToolTip('Drag the basis vectors<br><b>(Alt + 1)</b>')
180              self.button_define_visually.clicked.connect(lambda: self.dialog_define_matrix(DefineVisuallyDialog))
181              QShortcut(QKeySequence('Alt+1'), self).activated.connect(self.button_define_visually.click)
182
183              self.button_define_numerically = QtWidgets.QPushButton(self)
184              self.button_define_numerically.setText('Numerically')
185              self.button_define_numerically.setToolTip('Define a matrix just with numbers<br><b>(Alt + 2)</b>')
186              self.button_define_numerically.clicked.connect(lambda: self.dialog_define_matrix(DefineNumericallyDialog))
187              QShortcut(QKeySequence('Alt+2'), self).activated.connect(self.button_define_numerically.click)
188
189              self.button_define_as_expression = QtWidgets.QPushButton(self)
190              self.button_define_as_expression.setText('As an expression')
191              self.button_define_as_expression.setToolTip('Define a matrix in terms of other matrices<br><b>(Alt +
     ↪    3)</b>')
192              self.button_define_as_expression.clicked.connect(lambda:
     ↪    self.dialog_define_matrix(DefineAsAnExpressionDialog))
193              QShortcut(QKeySequence('Alt+3'), self).activated.connect(self.button_define_as_expression.click)
194
195              self.vlay_define_new_matrix = QVBoxLayout()
196              self.vlay_define_new_matrix.setSpacing(20)
197              self.vlay_define_new_matrix.addWidget(self.button_define_visually)
198              self.vlay_define_new_matrix.addWidget(self.button_define_numerically)
199              self.vlay_define_new_matrix.addWidget(self.button_define_as_expression)
200
201              self.groupbox_define_new_matrix = QtWidgets.QGroupBox('Define a new matrix', self)
202              self.groupbox_define_new_matrix.setLayout(self.vlay_define_new_matrix)
203
204              # Info panel button
205
206              self.button_info_panel = QtWidgets.QPushButton(self)
207              self.button_info_panel.setText('Show defined matrices')
208              self.button_info_panel.clicked.connect(
209                  # We have to use a lambda instead of 'InfoPanelDialog(self.matrix_wrapper, self).open' here
210                  # because that would create an unnamed instance of InfoPanelDialog when LintransMainWindow is
211                  # constructed, but we need to create a new instance every time to keep self.matrix_wrapper up to date
212                  lambda: InfoPanelDialog(self.matrix_wrapper, self).open()
213              )
214              self.button_info_panel.setToolTip(
215                  'Open an info panel with all matrices that have been defined in this session<br><b>(Ctrl + M)</b>'
216              )
217              QShortcut(QKeySequence('Ctrl+M'), self).activated.connect(self.button_info_panel.click)
218
219              # Render buttons
220
221              self.button_reset = QtWidgets.QPushButton(self)
222              self.button_reset.setText('Reset')
223              self.button_reset.clicked.connect(self.reset_transformation)
224              self.button_reset.setToolTip('Reset the visualized transformation back to the identity<br><b>(Ctrl +
     ↪    R)</b>')
225              QShortcut(QKeySequence('Ctrl+R'), self).activated.connect(self.button_reset.click)
226
227              self.button_render = QtWidgets.QPushButton(self)
228              self.button_render.setText('Render')
229              self.button_render.setEnabled(False)
230              self.button_render.clicked.connect(self.render_expression)
231              self.button_render.setToolTip('Render the expression<br><b>(Ctrl + Enter)</b>')
232              QShortcut(QKeySequence('Ctrl+Return'), self).activated.connect(self.button_render.click)
233
234              self.button_animate = QtWidgets.QPushButton(self)
235              self.button_animate.setText('Animate')
236              self.button_animate.setEnabled(False)
237              self.button_animate.clicked.connect(self.animate_expression)
238              self.button_animate.setToolTip('Animate the expression<br><b>(Ctrl + Shift + Enter)</b>')
239              QShortcut(QKeySequence('Ctrl+Shift+Return'), self).activated.connect(self.button_animate.click)
```

```python
240
241              # === Arrange widgets
242
243          self.vlay_left = QVBoxLayout()
244          self.vlay_left.addWidget(self.plot)
245          self.vlay_left.addWidget(self.lineedit_expression_box)
246
247          self.vlay_misc_buttons = QVBoxLayout()
248          self.vlay_misc_buttons.setSpacing(20)
249          self.vlay_misc_buttons.addWidget(self.button_create_polygon)
250          self.vlay_misc_buttons.addWidget(self.button_change_display_settings)
251          self.vlay_misc_buttons.addWidget(self.button_reset_zoom)
252
253          self.vlay_info_buttons = QVBoxLayout()
254          self.vlay_info_buttons.setSpacing(20)
255          self.vlay_info_buttons.addWidget(self.button_info_panel)
256
257          self.vlay_render = QVBoxLayout()
258          self.vlay_render.setSpacing(20)
259          self.vlay_render.addWidget(self.button_reset)
260          self.vlay_render.addWidget(self.button_animate)
261          self.vlay_render.addWidget(self.button_render)
262
263          self.vlay_right = QVBoxLayout()
264          self.vlay_right.setSpacing(50)
265          self.vlay_right.addLayout(self.vlay_misc_buttons)
266          self.vlay_right.addItem(QSpacerItem(100, 2, hPolicy=QSizePolicy.Minimum, vPolicy=QSizePolicy.Expanding))
267          self.vlay_right.addWidget(self.groupbox_define_new_matrix)
268          self.vlay_right.addItem(QSpacerItem(100, 2, hPolicy=QSizePolicy.Minimum, vPolicy=QSizePolicy.Expanding))
269          self.vlay_right.addLayout(self.vlay_info_buttons)
270          self.vlay_right.addItem(QSpacerItem(100, 2, hPolicy=QSizePolicy.Minimum, vPolicy=QSizePolicy.Expanding))
271          self.vlay_right.addLayout(self.vlay_render)
272
273          self.hlay_all = QHBoxLayout()
274          self.hlay_all.setSpacing(15)
275          self.hlay_all.addLayout(self.vlay_left)
276          self.hlay_all.addLayout(self.vlay_right)
277
278          self.central_widget = QtWidgets.QWidget()
279          self.central_widget.setLayout(self.hlay_all)
280          self.central_widget.setContentsMargins(10, 10, 10, 10)
281
282          self.setCentralWidget(self.central_widget)
283
284      def closeEvent(self, event: QCloseEvent) -> None:
285          """Handle a :class:`QCloseEvent` by cancelling animation first."""
286          self.animating = False
287          event.accept()
288
289      def update_render_buttons(self) -> None:
290          """Enable or disable the render and animate buttons according to whether the matrix expression is valid."""
291          text = self.lineedit_expression_box.text()
292
293          # Let's say that the user defines a non-singular matrix A, then defines B as A^-1
294          # If they then redefine A and make it singular, then we get a LinAlgError when
295          # trying to evaluate an expression with B in it
296          # To fix this, we just do naive validation rather than aware validation
297          if ',' in text:
298              self.button_render.setEnabled(False)
299
300              try:
301                  valid = all(self.matrix_wrapper.is_valid_expression(x) for x in text.split(','))
302              except LinAlgError:
303                  valid = all(validate_matrix_expression(x) for x in text.split(','))
304
305              self.button_animate.setEnabled(valid)
306
307          else:
308              try:
309                  valid = self.matrix_wrapper.is_valid_expression(text)
310              except LinAlgError:
311                  valid = validate_matrix_expression(text)
312
```

```
313                        self.button_render.setEnabled(valid)
314                        self.button_animate.setEnabled(valid)
315
316            @pyqtSlot()
317            def reset_zoom(self) -> None:
318                """Reset the zoom level back to normal."""
319                self.plot.grid_spacing = self.plot.default_grid_spacing
320                self.plot.update()
321
322            @pyqtSlot()
323            def reset_transformation(self) -> None:
324                """Reset the visualized transformation back to the identity."""
325                self.plot.visualize_matrix_transformation(self.matrix_wrapper['I'])
326                self.animating = False
327                self.animating_sequence = False
328                self.plot.update()
329
330            @pyqtSlot()
331            def render_expression(self) -> None:
332                """Render the transformation given by the expression in the input box."""
333                try:
334                    matrix = self.matrix_wrapper.evaluate_expression(self.lineedit_expression_box.text())
335
336                except LinAlgError:
337                    self.show_error_message('Singular matrix', 'Cannot take inverse of singular matrix')
338                    return
339
340                if self.is_matrix_too_big(matrix):
341                    self.show_error_message('Matrix too big', "This matrix doesn't fit on the canvas")
342                    return
343
344                self.plot.visualize_matrix_transformation(matrix)
345                self.plot.update()
346
347            @pyqtSlot()
348            def animate_expression(self) -> None:
349                """Animate from the current matrix to the matrix in the expression box."""
350                self.button_render.setEnabled(False)
351                self.button_animate.setEnabled(False)
352
353                matrix_start: MatrixType = np.array([
354                    [self.plot.point_i[0], self.plot.point_j[0]],
355                    [self.plot.point_i[1], self.plot.point_j[1]]
356                ])
357
358                text = self.lineedit_expression_box.text()
359
360                # If there's commas in the expression, then we want to animate each part at a time
361                if ',' in text:
362                    current_matrix = matrix_start
363                    self.animating_sequence = True
364
365                    # For each expression in the list, right multiply it by the current matrix,
366                    # and animate from the current matrix to that new matrix
367                    for expr in text.split(',')[::-1]:
368                        try:
369                            new_matrix = self.matrix_wrapper.evaluate_expression(expr)
370
371                            if self.plot.display_settings.applicative_animation:
372                                new_matrix = new_matrix @ current_matrix
373                        except LinAlgError:
374                            self.show_error_message('Singular matrix', 'Cannot take inverse of singular matrix')
375                            return
376
377                        if not self.animating_sequence:
378                            break
379
380                        self.animate_between_matrices(current_matrix, new_matrix)
381                        current_matrix = new_matrix
382
383                        # Here we just redraw and allow for other events to be handled while we pause
384                        self.plot.update()
385                        QApplication.processEvents()
```

205

```
386                    QThread.msleep(self.plot.display_settings.animation_pause_length)
387
388                self.animating_sequence = False
389
390        # If there's no commas, then just animate directly from the start to the target
391        else:
392            # Get the target matrix and it's determinant
393            try:
394                matrix_target = self.matrix_wrapper.evaluate_expression(text)
395
396            except LinAlgError:
397                self.show_error_message('Singular matrix', 'Cannot take inverse of singular matrix')
398                return
399
400            # The concept of applicative animation is explained in /gui/settings.py
401            if self.plot.display_settings.applicative_animation:
402                matrix_target = matrix_target @ matrix_start
403
404            # If we want a transitional animation and we're animating the same matrix, then restart the animation
405            # We use this check rather than equality because of small floating point errors
406            elif (abs(matrix_start - matrix_target) < 1e-12).all():
407                matrix_start = self.matrix_wrapper['I']
408
409                # We pause here for 200 ms to make the animation look a bit nicer
410                self.plot.visualize_matrix_transformation(matrix_start)
411                self.plot.update()
412                QApplication.processEvents()
413                QThread.msleep(200)
414
415            self.animate_between_matrices(matrix_start, matrix_target)
416
417        self.update_render_buttons()
418
419    def _get_animation_frame(self, start: MatrixType, target: MatrixType, proportion: float) -> MatrixType:
420        """Get the matrix to render for this frame of the animation.
421
422        This method will smoothen the determinant if that setting in enabled and if the determinant is positive.
423        It also animates rotation-like matrices using a logarithmic spiral to rotate around and scale continuously.
424        Essentially, it just makes things look good when animating.
425
426        :param MatrixType start: The starting matrix
427        :param MatrixType start: The target matrix
428        :param float proportion: How far we are through the loop
429        """
430        det_target = linalg.det(target)
431        det_start = linalg.det(start)
432
433        # This is the matrix that we're applying to get from start to target
434        # We want to check if it's rotation-like
435        if linalg.det(start) == 0:
436            matrix_application = None
437        else:
438            matrix_application = target @ linalg.inv(start)
439
440        # For a matrix to represent a rotation, it must have a positive determinant,
441        # its vectors must be perpendicular, and its vectors must be the same length
442        # The checks for 'abs(value) < 1e-10' are to account for floating point error
443        if matrix_application is not None \
444                and self.plot.display_settings.smoothen_determinant \
445                and linalg.det(matrix_application) > 0 \
446                and abs(np.dot(matrix_application.T[0], matrix_application.T[1])) < 1e-10 \
447                and abs(np.hypot(*matrix_application.T[0]) - np.hypot(*matrix_application.T[1])) < 1e-10:
448            rotation_vector: VectorType = matrix_application.T[0]  # Take the i column
449            radius, angle = polar_coords(*rotation_vector)
450
451            # We want the angle to be in [-pi, pi), so we have to subtract 2pi from it if it's too big
452            if angle > np.pi:
453                angle -= 2 * np.pi
454
455            i: VectorType = start.T[0]
456            j: VectorType = start.T[1]
457
458            # Scale the coords with a list comprehension
```

```
459                    # It's a bit janky, but rotate_coords() will always return a 2-tuple,
460                    # so new_i and new_j will always be lists of length 2
461                    scale = (radius - 1) * proportion + 1
462                    new_i = [scale * c for c in rotate_coord(i[0], i[1], angle * proportion)]
463                    new_j = [scale * c for c in rotate_coord(j[0], j[1], angle * proportion)]
464
465                    return np.array(
466                        [
467                            [new_i[0], new_j[0]],
468                            [new_i[1], new_j[1]]
469                        ]
470                    )
471
472                # matrix_a is the start matrix plus some part of the target, scaled by the proportion
473                # If we just used matrix_a, then things would animate, but the determinants would be weird
474                matrix_a = start + proportion * (target - start)
475
476                if not self.plot.display_settings.smoothen_determinant or det_start * det_target <= 0:
477                    return matrix_a
478
479                # To fix the determinant problem, we get the determinant of matrix_a and use it to normalize
480                det_a = linalg.det(matrix_a)
481
482                # For a 2x2 matrix A and a scalar c, we know that det(cA) = c^2 det(A)
483                # We want B = cA such that det(B) = det(S), where S is the start matrix,
484                # so then we can scale it with the animation, so we get
485                # det(cA) = c^2 det(A) = det(S) => c = sqrt(abs(det(S) / det(A)))
486                # Then we scale A to get the determinant we want, and call that matrix_b
487                if det_a == 0:
488                    c = 0
489                else:
490                    c = np.sqrt(abs(det_start / det_a))
491
492                matrix_b = c * matrix_a
493                det_b = linalg.det(matrix_b)
494
495                # We want to return B, but we have to scale it over time to have the target determinant
496
497                # We want some C = dB such that det(C) is some target determinant T
498                # det(dB) = d^2 det(B) = T => d = sqrt(abs(T / det(B)))
499
500                # We're also subtracting 1 and multiplying by the proportion and then adding one
501                # This just scales the determinant along with the animation
502
503                # That is all of course, if we can do that
504                # We'll crash if we try to do this with det(B) == 0
505                if det_b == 0:
506                    return matrix_a
507
508                scalar = 1 + proportion * (np.sqrt(abs(det_target / det_b)) - 1)
509                return scalar * matrix_b

510
511        def animate_between_matrices(self, matrix_start: MatrixType, matrix_target: MatrixType) -> None:
512            """Animate from the start matrix to the target matrix."""
513            self.animating = True
514
515            # Making steps depend on animation_time ensures a smooth animation without
516            # massive overheads for small animation times
517            steps = self.plot.display_settings.animation_time // 10
518
519            for i in range(0, steps + 1):
520                if not self.animating:
521                    break
522
523                matrix_to_render = self._get_animation_frame(matrix_start, matrix_target, i / steps)
524
525                if self.is_matrix_too_big(matrix_to_render):
526                    self.show_error_message('Matrix too big', "This matrix doesn't fit on the canvas")
527                    self.animating = False
528                    return
529
530                self.plot.visualize_matrix_transformation(matrix_to_render)
531
```

```
532                    # We schedule the plot to be updated, tell the event loop to
533                    # process events, and asynchronously sleep for 10ms
534                    # This allows for other events to be processed while animating, like zooming in and out
535                    self.plot.update()
536                    QApplication.processEvents()
537                    QThread.msleep(self.plot.display_settings.animation_time // steps)
538
539                self.animating = False
540
541        @pyqtSlot(DefineDialog)
542        def dialog_define_matrix(self, dialog_class: Type[DefineDialog]) -> None:
543            """Open a generic definition dialog to define a new matrix.
544
545            The class for the desired dialog is passed as an argument. We create an
546            instance of this class and the dialog is opened asynchronously and modally
547            (meaning it blocks interaction with the main window) with the proper method
548            connected to the :meth:`QDialog.accepted` signal.
549
550            .. note:: ``dialog_class`` must subclass :class:`lintrans.gui.dialogs.define_new_matrix.DefineDialog`.
551
552            :param dialog_class: The dialog class to instantiate
553            :type dialog_class: Type[lintrans.gui.dialogs.define_new_matrix.DefineDialog]
554            """
555            # We create a dialog with a deepcopy of the current matrix_wrapper
556            # This avoids the dialog mutating this one
557            dialog: DefineDialog
558
559            if dialog_class == DefineVisuallyDialog:
560                dialog = DefineVisuallyDialog(
561                    self,
562                    matrix_wrapper=deepcopy(self.matrix_wrapper),
563                    display_settings=self.plot.display_settings
564                )
565            else:
566                dialog = dialog_class(self, matrix_wrapper=deepcopy(self.matrix_wrapper))
567
568            # .open() is asynchronous and doesn't spawn a new event loop, but the dialog is still modal (blocking)
569            dialog.open()
570
571            # So we have to use the accepted signal to call a method when the user accepts the dialog
572            dialog.accepted.connect(self.assign_matrix_wrapper)
573
574        @pyqtSlot()
575        def assign_matrix_wrapper(self) -> None:
576            """Assign a new value to ``self.matrix_wrapper`` and give the expression box focus."""
577            self.matrix_wrapper = self.sender().matrix_wrapper
578            self.lineedit_expression_box.setFocus()
579            self.update_render_buttons()
580
581        @pyqtSlot()
582        def dialog_change_display_settings(self) -> None:
583            """Open the dialog to change the display settings."""
584            dialog = DisplaySettingsDialog(self, display_settings=self.plot.display_settings)
585            dialog.open()
586            dialog.accepted.connect(lambda: self.assign_display_settings(dialog.display_settings))
587
588        @pyqtSlot(DisplaySettings)
589        def assign_display_settings(self, display_settings: DisplaySettings) -> None:
590            """Assign a new value to ``self.plot.display_settings`` and give the expression box focus."""
591            self.plot.display_settings = display_settings
592            self.plot.update()
593            self.lineedit_expression_box.setFocus()
594            self.update_render_buttons()
595
596        def show_error_message(self, title: str, text: str, info: str | None = None) -> None:
597            """Show an error message in a dialog box.
598
599            :param str title: The window title of the dialog box
600            :param str text: The simple error message
601            :param info: The more informative error message
602            :type info: Optional[str]
603            """
604            dialog = QMessageBox(self)
```

```
605                dialog.setIcon(QMessageBox.Critical)
606                dialog.setWindowTitle(title)
607                dialog.setText(text)
608
609                if info is not None:
610                    dialog.setInformativeText(info)
611
612                dialog.open()
613
614                # This is `finished` rather than `accepted` because we want to update the buttons no matter what
615                dialog.finished.connect(self.update_render_buttons)
616
617        def is_matrix_too_big(self, matrix: MatrixType) -> bool:
618            """Check if the given matrix will actually fit onto the canvas.
619
620            Convert the elements of the matrix to canvas coords and make sure they fit within Qt's 32-bit integer limit.
621
622            :param MatrixType matrix: The matrix to check
623            :returns bool: Whether the matrix is too big to fit on the canvas
624            """
625            coords: List[Tuple[int, int]] = [self.plot.canvas_coords(*vector) for vector in matrix.T]
626
627            for x, y in coords:
628                if not (-2147483648 <= x <= 2147483647 and -2147483648 <= y <= 2147483647):
629                    return True
630
631            return False
632
633
634    def qapp() -> QCoreApplication:
635        """Return the equivalent of the global :class:`qApp` pointer.
636
637        :raises RuntimeError: If :meth:`QCoreApplication.instance` returns ``None``
638        """
639        instance = QCoreApplication.instance()
640
641        if instance is None:
642            raise RuntimeError('qApp undefined')
643
644        return instance
645
646
647    def main(args: List[str]) -> None:
648        """Run the GUI by creating and showing an instance of :class:`LintransMainWindow`.
649
650        :param List[str] args: The args to pass to :class:`QApplication`
651        """
652        app = QApplication(args)
653        app.setApplicationName('lintrans')
654        app.setApplicationVersion(lintrans.__version__)
655
656        qapp().setStyle(QStyleFactory.create('fusion'))
657
658        window = LintransMainWindow()
659        window.show()
660
661        sys.exit(app.exec_())
```

A.5 gui/settings.py

```
1     # lintrans - The linear transformation visualizer
2     # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4     # This program is licensed under GNU GPLv3, available here:
5     # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7     """This module contains the :class:`DisplaySettings` class, which holds configuration for display."""
8
9     from __future__ import annotations
10
```

```
11    from dataclasses import dataclass
12
13
14    @dataclass
15    class DisplaySettings:
16        """This class simply holds some attributes to configure display."""
17
18        # === Basic stuff
19
20        draw_background_grid: bool = True
21        """This controls whether we want to draw the background grid.
22
23        The background axes will always be drawn. This makes it easy to identify the center of the space.
24        """
25
26        draw_transformed_grid: bool = True
27        """This controls whether we want to draw the transformed grid. Vectors are handled separately."""
28
29        draw_basis_vectors: bool = True
30        """This controls whether we want to draw the transformed basis vectors."""
31
32        # === Animations
33
34        smoothen_determinant: bool = True
35        """This controls whether we want the determinant to change smoothly during the animation.
36
37        .. note::
38            Even if this is True, it will be ignored if we're animating from a positive det matrix to
39            a negative det matrix, or vice versa, because if we try to smoothly animate that determinant,
40            things blow up and the app often crashes.
41        """
42
43        applicative_animation: bool = True
44        """There are two types of simple animation, transitional and applicative.
45
46        Let ``C`` be the matrix representing the currently displayed transformation, and let ``T`` be the target matrix.
47        Transitional animation means that we animate directly from ``C`` from ``T``,
48        and applicative animation means that we animate from ``C`` to ``TC``, so we apply ``T`` to ``C``.
49        """
50
51        animation_time: int = 1200
52        """This is the number of milliseconds that an animation takes."""
53
54        animation_pause_length: int = 400
55        """This is the number of milliseconds that we wait between animations when using comma syntax."""
56
57        # === Matrix info
58
59        draw_determinant_parallelogram: bool = False
60        """This controls whether or not we should shade the parallelogram representing the determinant of the matrix."""
61
62        show_determinant_value: bool = True
63        """This controls whether we should write the text value of the determinant inside the parallelogram.
64
65        The text only gets draw if :attr:`draw_determinant_parallelogram` is also True.
66        """
67
68        draw_eigenvectors: bool = False
69        """This controls whether we should draw the eigenvectors of the transformation."""
70
71        draw_eigenlines: bool = False
72        """This controls whether we should draw the eigenlines of the transformation."""
```

A.6   `gui/validate.py`

```
1    # lintrans - The linear transformation visualizer
2    # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4    # This program is licensed under GNU GPLv3, available here:
5    # <https://www.gnu.org/licenses/gpl-3.0.html>
```

```
6
7     """This simple module provides a :class:`MatrixExpressionValidator` class to validate matrix expression input."""
8
9     from __future__ import annotations
10
11    import re
12    from typing import Tuple
13
14    from PyQt5.QtGui import QValidator
15
16    from lintrans.matrices import parse
17
18
19    class MatrixExpressionValidator(QValidator):
20        """This class validates matrix expressions in a Qt input box."""
21
22        def validate(self, text: str, pos: int) -> Tuple[QValidator.State, str, int]:
23            """Validate the given text according to the rules defined in the :mod:`lintrans.matrices` module."""
24            clean_text = re.sub(parse.NAIVE_CHARACTER_CLASS[:-1] + ',]', '', text)
25
26            if clean_text == '':
27                if parse.validate_matrix_expression(clean_text):
28                    return QValidator.Acceptable, text, pos
29                else:
30                    return QValidator.Intermediate, text, pos
31
32            return QValidator.Invalid, text, pos
```

A.7 matrices/__init__.py

```
1     # lintrans - The linear transformation visualizer
2     # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4     # This program is licensed under GNU GPLv3, available here:
5     # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7     """This package supplies classes and functions to parse, evaluate, and wrap matrices."""
8
9     from . import parse, utility
10    from .utility import create_rotation_matrix
11    from .wrapper import MatrixWrapper
12
13    __all__ = ['create_rotation_matrix', 'MatrixWrapper', 'parse', 'utility']
```

A.8 matrices/parse.py

```
1     # lintrans - The linear transformation visualizer
2     # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4     # This program is licensed under GNU GPLv3, available here:
5     # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7     """This module provides functions to parse and validate matrix expressions."""
8
9     from __future__ import annotations
10
11    import re
12    from dataclasses import dataclass
13    from typing import List, Pattern, Tuple
14
15    from lintrans.typing_ import MatrixParseList
16
17    NAIVE_CHARACTER_CLASS = r'[-+\sA-Z0-9.rot()^{}]'
18
19
20    class MatrixParseError(Exception):
21        """A simple exception to be raised when an error is found when parsing."""
22
```

```
23
24   def compile_naive_expression_pattern() -> Pattern[str]:
25       """Compile the single RegEx pattern that will match a valid matrix expression."""
26       digit_no_zero = '[123456789]'
27       digits = '\\d+'
28       integer_no_zero = digit_no_zero + '(' + digits + ')?'
29       real_number = f'({integer_no_zero}(\\.{digits})?|0\\.{digits})'
30
31       index_content = f'(-?{integer_no_zero}|T)'
32       index = f'(\\^{{{index_content}}}|\\^{index_content})'
33       matrix_identifier = f'([A-Z]|rot\\(-?{real_number}\\)|\\({NAIVE_CHARACTER_CLASS}+\\))'
34       matrix = '(' + real_number + '?' + matrix_identifier + index + '?)'
35       expression = f'^-?{matrix}+((\\+-?|-){matrix}+)*$'
36
37       return re.compile(expression)
38
39
40   # This is an expensive pattern to compile, so we compile it when this module is initialized
41   naive_expression_pattern = compile_naive_expression_pattern()
42
43
44   def find_sub_expressions(expression: str) -> List[str]:
45       """Find all the sub-expressions in the given expression.
46
47       This function only goes one level deep, so may return strings like ``'A(BC)D'``.
48
49       :raises MatrixParseError: If there are unbalanced parentheses
50       """
51       sub_expressions: List[str] = []
52       string = ''
53       paren_depth = 0
54       pointer = 0
55
56       while True:
57           char = expression[pointer]
58
59           if char == '(' and expression[pointer - 3:pointer] != 'rot':
60               paren_depth += 1
61
62               # This is a bit of a manual bodge, but it eliminates extraneous parens
63               if paren_depth == 1:
64                   pointer += 1
65                   continue
66
67           elif char == ')' and re.match(f'{NAIVE_CHARACTER_CLASS}*?rot\\([-\\d.]+$', expression[:pointer]) is None:
68               paren_depth -= 1
69
70           if paren_depth > 0:
71               string += char
72
73           if paren_depth == 0 and string:
74               sub_expressions.append(string)
75               string = ''
76
77           pointer += 1
78
79           if pointer >= len(expression):
80               break
81
82       if paren_depth != 0:
83           raise MatrixParseError('Unbalanced parentheses in expression')
84
85       return sub_expressions
86
87
88   def validate_matrix_expression(expression: str) -> bool:
89       """Validate the given matrix expression.
90
91       This function simply checks the expression against the BNF schema documented in
92       :ref:`expression-syntax-docs`. It is not aware of which matrices are actually defined
93       in a wrapper. For an aware version of this function, use the
94       :meth:`lintrans.matrices.wrapper.MatrixWrapper.is_valid_expression` method.
95
```

212

```
96              :param str expression: The expression to be validated
97              :returns bool: Whether the expression is valid according to the schema
98              """
99              # Remove all whitespace
100             expression = re.sub(r'\s', '', expression)
101
102             match = naive_expression_pattern.match(expression)
103
104             if match is None:
105                 return False
106
107             # Check that the whole expression was matched against
108             if expression != match.group(0):
109                 return False
110
111             try:
112                 sub_expressions = find_sub_expressions(expression)
113             except MatrixParseError:
114                 return False
115
116             if not sub_expressions:
117                 return True
118
119             return all(validate_matrix_expression(m) for m in sub_expressions)
120
121
122     @dataclass
123     class MatrixToken:
124         """A simple dataclass to hold information about a matrix token being parsed."""
125
126         multiplier: str = ''
127         identifier: str = ''
128         exponent: str = ''
129
130         @property
131         def tuple(self) -> Tuple[str, str, str]:
132             """Create a tuple of the token for parsing."""
133             return self.multiplier, self.identifier, self.exponent
134
135
136     class ExpressionParser:
137         """A class to hold state during parsing.
138
139         Most of the methods in this class are class-internal and should not be used from outside.
140
141         This class should be used like this:
142
143         >>> ExpressionParser('3A^-1B').parse()
144         [[('3', 'A', '-1'), ('', 'B', '')]]
145         >>> ExpressionParser('4(M^TA^2)^-2').parse()
146         [[('4', 'M^{T}A^{2}', '-2')]]
147         """
148
149         def __init__(self, expression: str):
150             """Create an instance of the parser with the given expression and initialise variables to use during
                 ↪  parsing."""
151             # Remove all whitespace
152             expression = re.sub(r'\s', '', expression)
153
154             # Check if it's valid
155             if not validate_matrix_expression(expression):
156                 raise MatrixParseError('Invalid expression')
157
158             # Wrap all exponents and transposition powers with {}
159             expression = re.sub(r'(?<=\^)(-?\d+|T)(?=[^}]|$)', r'{\g<0>}', expression)
160
161             # Remove any standalone minuses
162             expression = re.sub(r'-(?=[A-Z])', '-1', expression)
163
164             # Replace subtractions with additions
165             expression = re.sub(r'-(?=\d+\.?\d*([A-Z]|rot))', '+-', expression)
166
167             # Get rid of a potential leading + introduced by the last step
```

```
168                expression = re.sub(r'^\+', '', expression)
169
170            self.expression = expression
171            self.pointer: int = 0
172
173            self.current_token = MatrixToken()
174            self.current_group: List[Tuple[str, str, str]] = []
175
176            self.final_list: MatrixParseList = []
177
178        def __repr__(self) -> str:
179            """Return a simple repr containing the expression."""
180            return f'{self.__class__.__module__}.{self.__class__.__name__}("{self.expression}")'
181
182        @property
183        def char(self) -> str:
184            """Return the char pointed to by the pointer."""
185            return self.expression[self.pointer]
186
187        def parse(self) -> MatrixParseList:
188            """Fully parse the instance's matrix expression and return the :attr:`lintrans.typing_.MatrixParseList`.
189
190            This method uses all the private methods of this class to parse the
191            expression in parts. All private methods mutate the instance variables.
192
193            :returns: The parsed expression
194            :rtype: :attr:`lintrans.typing_.MatrixParseList`
195            """
196            self._parse_multiplication_group()
197
198            while self.pointer < len(self.expression):
199                if self.expression[self.pointer] != '+':
200                    raise MatrixParseError('Expected "+" between multiplication groups')
201
202                self.pointer += 1
203                self._parse_multiplication_group()
204
205            return self.final_list
206
207        def _parse_multiplication_group(self) -> None:
208            """Parse a group of matrices to be multiplied together.
209
210            This method just parses matrices until we get to a ``+``.
211            """
212            # This loop continues to parse matrices until we fail to do so
213            while self._parse_matrix():
214                # Once we get to the end of the multiplication group, we add it the final list and reset the group list
215                if self.pointer >= len(self.expression) or self.char == '+':
216                    self.final_list.append(self.current_group)
217                    self.current_group = []
218                    self.pointer += 1
219
220        def _parse_matrix(self) -> bool:
221            """Parse a full matrix using :meth:`_parse_matrix_part`.
222
223            This method will parse an optional multiplier, an identifier, and an optional exponent. If we
224            do this successfully, we return True. If we fail to parse a matrix (maybe we've reached the
225            end of the current multiplication group and the next char is ``+``), then we return False.
226
227            :returns bool: Success or failure
228            """
229            self.current_token = MatrixToken()
230
231            while self._parse_matrix_part():
232                pass  # The actual execution is taken care of in the loop condition
233
234            if self.current_token.identifier == '':
235                return False
236
237            self.current_group.append(self.current_token.tuple)
238            return True
239
240        def _parse_matrix_part(self) -> bool:
```

```
241            """Parse part of a matrix (multiplier, identifier, or exponent).
242
243            Which part of the matrix we parse is dependent on the current value of the pointer and the expression.
244            This method will parse whichever part of matrix token that it can. If it can't parse a part of a matrix,
245            or it's reached the next matrix, then we just return False. If we succeeded to parse a matrix part, then
246            we return True.
247
248            :returns bool: Success or failure
249            :raises MatrixParseError: If we fail to parse this part of the matrix
250            """
251            if self.pointer >= len(self.expression):
252                return False
253
254            if self.char.isdigit() or self.char == '-':
255                if self.current_token.multiplier != '' \
256                        or (self.current_token.multiplier == '' and self.current_token.identifier != ''):
257                    return False
258
259                self._parse_multiplier()
260
261            elif self.char.isalpha() and self.char.isupper():
262                if self.current_token.identifier != '':
263                    return False
264
265                self.current_token.identifier = self.char
266                self.pointer += 1
267
268            elif self.char == 'r':
269                if self.current_token.identifier != '':
270                    return False
271
272                self._parse_rot_identifier()
273
274            elif self.char == '(':
275                if self.current_token.identifier != '':
276                    return False
277
278                self._parse_sub_expression()
279
280            elif self.char == '^':
281                if self.current_token.exponent != '':
282                    return False
283
284                self._parse_exponent()
285
286            elif self.char == '+':
287                return False
288
289            else:
290                raise MatrixParseError(f'Unrecognised character "{self.char}" in matrix expression')
291
292            return True
293
294        def _parse_multiplier(self) -> None:
295            """Parse a multiplier from the expression and pointer.
296
297            This method just parses a numerical multiplier, which can include
298            zero or one ``.`` character and optionally a ``-`` at the start.
299
300            :raises MatrixParseError: If we fail to parse this part of the matrix
301            """
302            multiplier = ''
303
304            while self.char.isdigit() or self.char in ('.', '-'):
305                multiplier += self.char
306                self.pointer += 1
307
308            try:
309                float(multiplier)
310            except ValueError as e:
311                raise MatrixParseError(f'Invalid multiplier "{multiplier}"') from e
312
313            self.current_token.multiplier = multiplier
```

```
314
315        def _parse_rot_identifier(self) -> None:
316            """Parse a ``rot()``-style identifier from the expression and pointer.
317
318            This method will just parse something like ``rot(12.5)``. The angle number must be a real number.
319
320            :raises MatrixParseError: If we fail to parse this part of the matrix
321            """
322            if match := re.match(r'rot\(([\d.-]+)\)', self.expression[self.pointer:]):
323                # Ensure that the number in brackets is a valid float
324                try:
325                    float(match.group(1))
326                except ValueError as e:
327                    raise MatrixParseError(f'Invalid angle number "{match.group(1)}" in rot-identifier') from e
328
329                self.current_token.identifier = match.group(0)
330                self.pointer += len(match.group(0))
331            else:
332                raise MatrixParseError(f'Invalid rot-identifier "{self.expression[self.pointer:self.pointer + 15]}..."')
333
334        def _parse_sub_expression(self) -> None:
335            """Parse a parenthesized sub-expression as the identifier.
336
337            This method will also validate the expression in the parentheses.
338
339            :raises MatrixParseError: If we fail to parse this part of the matrix
340            """
341            if self.char != '(':
342                raise MatrixParseError('Sub-expression must start with "("')
343
344            self.pointer += 1
345            paren_depth = 1
346            identifier = ''
347
348            while paren_depth > 0:
349                if self.char == '(':
350                    paren_depth += 1
351                elif self.char == ')':
352                    paren_depth -= 1
353
354                if paren_depth == 0:
355                    self.pointer += 1
356                    break
357
358                identifier += self.char
359                self.pointer += 1
360
361            if not validate_matrix_expression(identifier):
362                raise MatrixParseError(f'Invalid sub-expression identifier "{identifier}"')
363
364            self.current_token.identifier = identifier
365
366        def _parse_exponent(self) -> None:
367            """Parse a matrix exponent from the expression and pointer.
368
369            The exponent must be an integer or ``T`` for transpose.
370
371            :raises MatrixParseError: If we fail to parse this part of the token
372            """
373            if match := re.match(r'\^\{(-?\d+|T)\}', self.expression[self.pointer:]):
374                exponent = match.group(1)
375
376                try:
377                    if exponent != 'T':
378                        int(exponent)
379                except ValueError as e:
380                    raise MatrixParseError(f'Invalid exponent "{match.group(1)}"') from e
381
382                self.current_token.exponent = exponent
383                self.pointer += len(match.group(0))
384            else:
385                raise MatrixParseError(f'Invalid exponent "{self.expression[self.pointer:self.pointer + 10]}..."')
386
```

216

```
387
388    def parse_matrix_expression(expression: str) -> MatrixParseList:
389        """Parse the matrix expression and return a :data:`lintrans.typing_.MatrixParseList`.
390
391        :Example:
392
393        >>> parse_matrix_expression('A')
394        [[('', 'A', '')]]
395        >>> parse_matrix_expression('-3M^2')
396        [[('-3', 'M', '2')]]
397        >>> parse_matrix_expression('1.2rot(12)^{3}2B^T')
398        [[('1.2', 'rot(12)', '3'), ('2', 'B', 'T')]]
399        >>> parse_matrix_expression('A^2 + 3B')
400        [[('', 'A', '2')], [('3', 'B', '')]]
401        >>> parse_matrix_expression('-3A^{-1}3B^T - 45M^2')
402        [[('-3', 'A', '-1'), ('3', 'B', 'T')], [('-45', 'M', '2')]]
403        >>> parse_matrix_expression('5.3A^{4} 2.6B^{-2} + 4.6D^T 8.9E^{-1}')
404        [[('5.3', 'A', '4'), ('2.6', 'B', '-2')], [('4.6', 'D', 'T'), ('8.9', 'E', '-1')]]
405        >>> parse_matrix_expression('2(A+B^TC)^2D')
406        [[('2', 'A+B^{T}C', '2'), ('', 'D', '')]]
407
408        :param str expression: The expression to be parsed
409        :returns: A list of parsed components
410        :rtype: :data:`lintrans.typing_.MatrixParseList`
411        """
412        return ExpressionParser(expression).parse()
```

A.9 `matrices/utility.py`

```
1     #  lintrans - The linear transformation visualizer
2     #  Copyright (C) 2022 D. Dyson (DoctorDalek1963)
3     #
4     #  This program is licensed under GNU GPLv3, available here:
5     #  <https://www.gnu.org/licenses/gpl-3.0.html>
6
7     """This module provides simple utility methods for matrix and vector manipulation."""
8
9     from __future__ import annotations
10
11    import math
12    from typing import Tuple
13
14    import numpy as np
15
16    from lintrans.typing_ import MatrixType
17
18
19    def polar_coords(x: float, y: float, *, degrees: bool = False) -> Tuple[float, float]:
20        """Return the polar coordinates of a given (x, y) Cartesian coordinate.
21
22        .. note:: We're returning the angle in the range [0, 2pi)
23        """
24        radius = math.hypot(x, y)
25
26        # PyCharm complains about np.angle taking a complex argument even though that's what it's designed for
27        # noinspection PyTypeChecker
28        angle = float(np.angle(x + y * 1j, degrees))
29
30        if angle < 0:
31            angle += 2 * np.pi
32
33        return radius, angle
34
35
36    def rect_coords(radius: float, angle: float, *, degrees: bool = False) -> Tuple[float, float]:
37        """Return the rectilinear coordinates of a given polar coordinate."""
38        if degrees:
39            angle = np.radians(angle)
40
41        return radius * np.cos(angle), radius * np.sin(angle)
```

```
42
43
44    def rotate_coord(x: float, y: float, angle: float, *, degrees: bool = False) -> Tuple[float, float]:
45        """Rotate a rectilinear coordinate by the given angle."""
46        if degrees:
47            angle = np.radians(angle)
48
49        r, theta = polar_coords(x, y, degrees=degrees)
50        theta = (theta + angle) % (2 * np.pi)
51
52        return rect_coords(r, theta, degrees=degrees)
53
54
55    def create_rotation_matrix(angle: float, *, degrees: bool = True) -> MatrixType:
56        """Create a matrix representing a rotation (anticlockwise) by the given angle.
57
58        :Example:
59
60        >>> create_rotation_matrix(30)
61        array([[ 0.8660254, -0.5      ],
62               [ 0.5      ,  0.8660254]])
63        >>> create_rotation_matrix(45)
64        array([[ 0.70710678, -0.70710678],
65               [ 0.70710678,  0.70710678]])
66        >>> create_rotation_matrix(np.pi / 3, degrees=False)
67        array([[ 0.5      , -0.8660254],
68               [ 0.8660254,  0.5      ]])
69
70        :param float angle: The angle to rotate anticlockwise by
71        :param bool degrees: Whether to interpret the angle as degrees (True) or radians (False)
72        :returns MatrixType: The resultant matrix
73        """
74        rad = np.deg2rad(angle % 360) if degrees else angle % (2 * np.pi)
75        return np.array([
76            [np.cos(rad), -1 * np.sin(rad)],
77            [np.sin(rad), np.cos(rad)]
78        ])
79
80
81    def is_valid_float(string: str) -> bool:
82        """Check if the string is a valid float (or anything that can be cast to a float, such as an int).
83
84        This function simply checks that ``float(string)`` doesn't raise an error.
85
86        .. note:: An empty string is not a valid float, so will return False.
87
88        :param str string: The string to check
89        :returns bool: Whether the string is a valid float
90        """
91        try:
92            float(string)
93            return True
94        except ValueError:
95            return False
96
97
98    def round_float(num: float, precision: int = 5) -> str:
99        """Round a floating point number to a given number of decimal places for pretty printing.
100
101        :param float num: The number to round
102        :param int precision: The number of decimal places to round to
103        :returns str: The rounded number for pretty printing
104        """
105        # Round to ``precision`` number of decimal places
106        string = str(round(num, precision))
107
108        # Cut off the potential final zero
109        if string.endswith('.0'):
110            return string[:-2]
111
112        elif 'e' in string:  # Scientific notation
113            split = string.split('e')
114            # The leading 0 only happens when the exponent is negative, so we know there'll be a minus sign
```

218

```
115            return split[0] + 'e-' + split[1][1:].lstrip('0')
116
117        else:
118            return string
```

A.10 `matrices/wrapper.py`

```
1    # lintrans - The linear transformation visualizer
2    # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4    # This program is licensed under GNU GPLv3, available here:
5    # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7    """This module contains the main :class:`MatrixWrapper` class and a function to create a matrix from an angle."""
8
9    from __future__ import annotations
10
11   import re
12   from copy import copy
13   from functools import reduce
14   from operator import add, matmul
15   from typing import Any, Dict, List, Optional, Tuple, Union
16
17   import numpy as np
18
19   from lintrans.typing_ import is_matrix_type, MatrixType
20   from .parse import parse_matrix_expression, validate_matrix_expression
21   from .utility import create_rotation_matrix
22
23
24   class MatrixWrapper:
25       """A wrapper class to hold all possible matrices and allow access to them.
26
27       .. note::
28          When defining a custom matrix, its name must be a capital letter and cannot be ``I``.
29
30       The contained matrices can be accessed and assigned to using square bracket notation.
31
32       :Example:
33
34       >>> wrapper = MatrixWrapper()
35       >>> wrapper['I']
36       array([[1., 0.],
37              [0., 1.]])
38       >>> wrapper['M']  # Returns None
39       >>> wrapper['M'] = np.array([[1, 2], [3, 4]])
40       >>> wrapper['M']
41       array([[1., 2.],
42              [3., 4.]])
43       """
44
45       def __init__(self):
46           """Initialize a :class:`MatrixWrapper` object with a dictionary of matrices which can be accessed."""
47           self._matrices: Dict[str, Optional[Union[MatrixType, str]]] = {
48               'A': None, 'B': None, 'C': None, 'D': None,
49               'E': None, 'F': None, 'G': None, 'H': None,
50               'I': np.eye(2),  # I is always defined as the identity matrix
51               'J': None, 'K': None, 'L': None, 'M': None,
52               'N': None, 'O': None, 'P': None, 'Q': None,
53               'R': None, 'S': None, 'T': None, 'U': None,
54               'V': None, 'W': None, 'X': None, 'Y': None,
55               'Z': None
56           }
57
58       def __repr__(self) -> str:
59           """Return a nice string repr of the :class:`MatrixWrapper` for debugging."""
60           defined_matrices = ''.join([k for k, v in self._matrices.items() if v is not None])
61           return f'<{self.__class__.__module__}.{self.__class__.__name__} object with ' \
62               f"{len(defined_matrices)} defined matrices: '{defined_matrices}'>"
63
```

```python
64          def __eq__(self, other: Any) -> bool:
65              """Check for equality in wrappers by comparing dictionaries.
66
67              :param Any other: The object to compare this wrapper to
68              """
69              if not isinstance(other, self.__class__):
70                  return NotImplemented
71
72              # We loop over every matrix and check if every value is equal in each
73              for name in self._matrices:
74                  s_matrix = self[name]
75                  o_matrix = other[name]
76
77                  if s_matrix is None and o_matrix is None:
78                      continue
79
80                  elif (s_matrix is None and o_matrix is not None) or \
81                       (s_matrix is not None and o_matrix is None):
82                      return False
83
84                  # This is mainly to satisfy mypy, because we know these must be matrices
85                  elif not is_matrix_type(s_matrix) or not is_matrix_type(o_matrix):
86                      return False
87
88                  # Now we know they're both NumPy arrays
89                  elif np.array_equal(s_matrix, o_matrix):
90                      continue
91
92                  else:
93                      return False
94
95              return True
96
97          def __hash__(self) -> int:
98              """Return the hash of the matrices dictionary."""
99              return hash(self._matrices)
100
101         def __getitem__(self, name: str) -> Optional[MatrixType]:
102             """Get the matrix with the given name.
103
104             If it is a simple name, it will just be fetched from the dictionary. If the name is ``rot(x)``, with
105             a given angle in degrees, then we return a new matrix representing a rotation by that angle.
106
107             .. note::
108                 If the named matrix is defined as an expression, then this method will return its evaluation.
109                 If you want the expression itself, use :meth:`get_expression`.
110
111             :param str name: The name of the matrix to get
112             :returns Optional[MatrixType]: The value of the matrix (could be None)
113
114             :raises NameError: If there is no matrix with the given name
115             """
116             # Return a new rotation matrix
117             if (match := re.match(r'^rot\((-?\d*\.?\d*)\)$', name)) is not None:
118                 return create_rotation_matrix(float(match.group(1)))
119
120             if name not in self._matrices:
121                 if validate_matrix_expression(name):
122                     return self.evaluate_expression(name)
123
124                 raise NameError(f'Unrecognised matrix name "{name}"')
125
126             # We copy the matrix before we return it so the user can't accidentally mutate the matrix
127             matrix = copy(self._matrices[name])
128
129             if isinstance(matrix, str):
130                 return self.evaluate_expression(matrix)
131
132             return matrix
133
134         def __setitem__(self, name: str, new_matrix: Optional[Union[MatrixType, str]]) -> None:
135             """Set the value of matrix ``name`` with the new_matrix.
136
```

220

```
137              The new matrix may be a simple 2x2 NumPy array, or it could be a string, representing an
138              expression in terms of other, previously defined matrices.
139
140              :param str name: The name of the matrix to set the value of
141              :param Optional[Union[MatrixType, str]] new_matrix: The value of the new matrix (could be None)
142
143              :raises NameError: If the name isn't a legal matrix name
144              :raises TypeError: If the matrix isn't a valid 2x2 NumPy array or expression in terms of other defined
     ↪  matrices
145              :raises ValueError: If you attempt to define a matrix in terms of itself
146              """
147              if not (name in self._matrices and name != 'I'):
148                  raise NameError('Matrix name is illegal')
149
150              if new_matrix is None:
151                  self._matrices[name] = None
152                  return
153
154              if isinstance(new_matrix, str):
155                  if self.is_valid_expression(new_matrix):
156                      if name not in new_matrix:
157                          self._matrices[name] = new_matrix
158                          return
159                      else:
160                          raise ValueError('Cannot define a matrix recursively')
161
162              if not is_matrix_type(new_matrix):
163                  raise TypeError('Matrix must be a 2x2 NumPy array')
164
165              # All matrices must have float entries
166              a = float(new_matrix[0][0])
167              b = float(new_matrix[0][1])
168              c = float(new_matrix[1][0])
169              d = float(new_matrix[1][1])
170
171              self._matrices[name] = np.array([[a, b], [c, d]])
172
173          def get_expression(self, name: str) -> Optional[str]:
174              """If the named matrix is defined as an expression, return that expression, else return None.
175
176              :param str name: The name of the matrix
177              :returns Optional[str]: The expression that the matrix is defined as, or None
178
179              :raises NameError: If the name is invalid
180              """
181              if name not in self._matrices:
182                  raise NameError('Matrix must have a legal name')
183
184              matrix = self._matrices[name]
185              if isinstance(matrix, str):
186                  return matrix
187
188              return None
189
190          def is_valid_expression(self, expression: str) -> bool:
191              """Check if the given expression is valid, using the context of the wrapper.
192
193              This method calls :func:`lintrans.matrices.parse.validate_matrix_expression`, but also
194              ensures that all the matrices in the expression are defined in the wrapper.
195
196              :param str expression: The expression to validate
197              :returns bool: Whether the expression is valid in this wrapper
198
199              :raises LinAlgError: If a matrix is defined in terms of the inverse of a singular matrix
200              """
201              # Get rid of the transposes to check all capital letters
202              new_expression = expression.replace('^T', '').replace('^{T}', '')
203
204              # Make sure all the referenced matrices are defined
205              for matrix in [x for x in new_expression if re.match('[A-Z]', x)]:
206                  if self[matrix] is None:
207                      return False
208
```

```
209                    if (expr := self.get_expression(matrix)) is not None:
210                        if not self.is_valid_expression(expr):
211                            return False
212
213                return validate_matrix_expression(expression)
214
215        def evaluate_expression(self, expression: str) -> MatrixType:
216            """Evaluate a given expression and return the matrix evaluation.
217
218            :param str expression: The expression to be parsed
219            :returns MatrixType: The matrix result of the expression
220
221            :raises ValueError: If the expression is invalid
222            """
223            if not self.is_valid_expression(expression):
224                raise ValueError('The expression is invalid')
225
226            parsed_result = parse_matrix_expression(expression)
227            final_groups: List[List[MatrixType]] = []
228
229            for group in parsed_result:
230                f_group: List[MatrixType] = []
231
232                for multiplier, identifier, index in group:
233                    if index == 'T':
234                        m = self[identifier]
235
236                        # This assertion is just so mypy doesn't complain
237                        # We know this won't be None, because we know that this matrix is defined in this wrapper
238                        assert m is not None
239                        matrix_value = m.T
240
241                    else:
242                        matrix_value = np.linalg.matrix_power(self[identifier], 1 if index == '' else int(index))
243
244                    matrix_value *= 1 if multiplier == '' else float(multiplier)
245                    f_group.append(matrix_value)
246
247                final_groups.append(f_group)
248
249            return reduce(add, [reduce(matmul, group) for group in final_groups])
250
251        def get_defined_matrices(self) -> List[Tuple[str, Union[MatrixType, str]]]:
252            """Return a list of tuples containing the name and value of all defined matrices in the wrapper.
253
254            :returns: A list of tuples where the first element is the name, and the second element is the value
255            :rtype: List[Tuple[str, Union[MatrixType, str]]]
256            """
257            matrices = []
258
259            for name, value in self._matrices.items():
260                if value is not None:
261                    matrices.append((name, value))
262
263            return matrices
```

A.11 `typing_/__init__.py`

```
1    # lintrans - The linear transformation visualizer
2    # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4    # This program is licensed under GNU GPLv3, available here:
5    # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7    """This package supplies type aliases for linear algebra and transformations.
8
9    .. note::
10       This package is called ``typing_`` and not ``typing`` to avoid name collisions with the
11       builtin :mod:`typing`. I don't quite know how this collision occurs, but renaming
12       this module fixed the problem.
```

```python
13      """
14
15      from __future__ import annotations
16
17      from sys import version_info
18      from typing import Any, List, Tuple
19
20      from numpy import ndarray
21      from nptyping import NDArray, Float
22
23      if version_info >= (3, 10):
24          from typing import TypeGuard
25
26      __all__ = ['is_matrix_type', 'MatrixType', 'MatrixParseList', 'VectorType']
27
28      MatrixType = NDArray[(2, 2), Float]
29      """This type represents a 2x2 matrix as a NumPy array."""
30
31      VectorType = NDArray[(2,), Float]
32      """This type represents a 2D vector as a NumPy array, for use with :attr:`MatrixType`."""
33
34      MatrixParseList = List[List[Tuple[str, str, str]]]
35      """This is a list containing lists of tuples. Each tuple represents a matrix and is ``(multiplier,
36      matrix_identifier, index)`` where all of them are strings. These matrix-representing tuples are
37      contained in lists which represent multiplication groups. Every matrix in the group should be
38      multiplied together, in order. These multiplication group lists are contained by a top level list,
39      which is this type. Once these multiplication group lists have been evaluated, they should be summed.
40
41      In the tuples, the multiplier is a string representing a real number, the matrix identifier
42      is a capital letter or ``rot(x)`` where x is a real number angle, and the index is a string
43      representing an integer, or it's the letter ``T`` for transpose.
44      """
45
46
47      def is_matrix_type(matrix: Any) -> TypeGuard[NDArray[(2, 2), Float]]:
48          """Check if the given value is a valid matrix type.
49
50          .. note::
51             This function is a TypeGuard, meaning if it returns True, then the
52             passed value must be a :attr:`lintrans.typing_.MatrixType`.
53          """
54          return isinstance(matrix, ndarray) and matrix.shape == (2, 2)
```

A.12 `gui/dialogs/__init__.py`

```python
1      # lintrans - The linear transformation visualizer
2      # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4      # This program is licensed under GNU GPLv3, available here:
5      # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7      """This package provides separate dialogs for the main GUI.
8
9      These dialogs are for defining new matrices in different ways and editing settings.
10     """
11
12     from .define_new_matrix import DefineAsAnExpressionDialog, DefineDialog, DefineNumericallyDialog,
       ↪   DefineVisuallyDialog
13     from .misc import AboutDialog, InfoPanelDialog
14     from .settings import DisplaySettingsDialog
15
16     __all__ = ['AboutDialog', 'DefineAsAnExpressionDialog', 'DefineDialog', 'DefineNumericallyDialog',
17                'DefineVisuallyDialog', 'DisplaySettingsDialog', 'InfoPanelDialog']
```

A.13 `gui/dialogs/define_new_matrix.py`

```python
1      # lintrans - The linear transformation visualizer
2      # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
```

```
3
4      # This program is licensed under GNU GPLv3, available here:
5      # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7      """This module provides an abstract :class:`DefineDialog` class and subclasses, allowing definition of new
↪        matrices."""
8
9      from __future__ import annotations
10
11     import abc
12
13     from numpy import array
14     from PyQt5 import QtWidgets
15     from PyQt5.QtCore import pyqtSlot
16     from PyQt5.QtGui import QDoubleValidator, QKeySequence
17     from PyQt5.QtWidgets import QGridLayout, QHBoxLayout, QShortcut, QSizePolicy, QSpacerItem, QVBoxLayout
18
19     from lintrans.gui.dialogs.misc import FixedSizeDialog
20     from lintrans.gui.plots import DefineVisuallyWidget
21     from lintrans.gui.settings import DisplaySettings
22     from lintrans.gui.validate import MatrixExpressionValidator
23     from lintrans.matrices import MatrixWrapper
24     from lintrans.matrices.utility import is_valid_float, round_float
25     from lintrans.typing_ import MatrixType
26
27     ALPHABET_NO_I = 'ABCDEFGHJKLMNOPQRSTUVWXYZ'
28
29
30     class DefineDialog(FixedSizeDialog):
31         """An abstract superclass for definitions dialogs.
32
33         .. warning:: This class should never be directly instantiated, only subclassed.
34
35         .. note::
36            I would make this class have ``metaclass=abc.ABCMeta``, but I can't because it subclasses :class:`QDialog`,
37            and every superclass of a class must have the same metaclass, and :class:`QDialog` is not an abstract class.
38         """
39
40         def __init__(self, *args, matrix_wrapper: MatrixWrapper, **kwargs):
41             """Create the widgets and layout of the dialog.
42
43             .. note:: ``*args`` and ``**kwargs`` are passed to the super constructor (:class:`QDialog`).
44
45             :param MatrixWrapper matrix_wrapper: The MatrixWrapper that this dialog will mutate
46             """
47             super().__init__(*args, **kwargs)
48
49             self.matrix_wrapper = matrix_wrapper
50             self.setWindowTitle('Define a matrix')
51
52             # === Create the widgets
53
54             self.button_confirm = QtWidgets.QPushButton(self)
55             self.button_confirm.setText('Confirm')
56             self.button_confirm.setEnabled(False)
57             self.button_confirm.clicked.connect(self.confirm_matrix)
58             self.button_confirm.setToolTip('Confirm this as the new matrix<br><b>(Ctrl + Enter)</b>')
59             QShortcut(QKeySequence('Ctrl+Return'), self).activated.connect(self.button_confirm.click)
60
61             self.button_cancel = QtWidgets.QPushButton(self)
62             self.button_cancel.setText('Cancel')
63             self.button_cancel.clicked.connect(self.reject)
64             self.button_cancel.setToolTip('Cancel this definition<br><b>(Escape)</b>')
65
66             self.label_equals = QtWidgets.QLabel()
67             self.label_equals.setText('=')
68
69             self.combobox_letter = QtWidgets.QComboBox(self)
70
71             for letter in ALPHABET_NO_I:
72                 self.combobox_letter.addItem(letter)
73
74             self.combobox_letter.activated.connect(self.load_matrix)
```

```
75
76              # === Arrange the widgets
77
78              self.setContentsMargins(10, 10, 10, 10)
79
80              self.hlay_buttons = QHBoxLayout()
81              self.hlay_buttons.setSpacing(20)
82              self.hlay_buttons.addItem(QSpacerItem(50, 5, hPolicy=QSizePolicy.Expanding, vPolicy=QSizePolicy.Minimum))
83              self.hlay_buttons.addWidget(self.button_cancel)
84              self.hlay_buttons.addWidget(self.button_confirm)
85
86              self.hlay_definition = QHBoxLayout()
87              self.hlay_definition.setSpacing(20)
88              self.hlay_definition.addWidget(self.combobox_letter)
89              self.hlay_definition.addWidget(self.label_equals)
90
91              self.vlay_all = QVBoxLayout()
92              self.vlay_all.setSpacing(20)
93
94              self.setLayout(self.vlay_all)
95
96          @property
97          def selected_letter(self) -> str:
98              """Return the letter currently selected in the combo box."""
99              return str(self.combobox_letter.currentText())
100
101         @abc.abstractmethod
102         @pyqtSlot()
103         def update_confirm_button(self) -> None:
104             """Enable the confirm button if it should be enabled, else, disable it."""
105
106         @pyqtSlot(int)
107         def load_matrix(self, index: int) -> None:
108             """Load the selected matrix into the dialog.
109
110             This method is optionally able to be overridden. If it is not overridden,
111             then no matrix is loaded when selecting a name.
112
113             We have this method in the superclass so that we can define it as the slot
114             for the :meth:`QComboBox.activated` signal in this constructor, rather than
115             having to define that in the constructor of every subclass.
116             """
117
118         @abc.abstractmethod
119         @pyqtSlot()
120         def confirm_matrix(self) -> None:
121             """Confirm the inputted matrix and assign it.
122
123             .. note:: When subclassing, this method should mutate ``self.matrix_wrapper`` and then call
            ↪  ``self.accept()``.
124             """
125
126
127     class DefineVisuallyDialog(DefineDialog):
128         """The dialog class that allows the user to define a matrix visually."""
129
130         def __init__(self, *args, matrix_wrapper: MatrixWrapper, display_settings: DisplaySettings, **kwargs):
131             """Create the widgets and layout of the dialog.
132
133             :param MatrixWrapper matrix_wrapper: The MatrixWrapper that this dialog will mutate
134             """
135             super().__init__(*args, matrix_wrapper=matrix_wrapper, **kwargs)
136
137             self.setMinimumSize(700, 550)
138
139             # === Create the widgets
140
141             self.plot = DefineVisuallyWidget(self, display_settings=display_settings)
142
143             # === Arrange the widgets
144
145             self.hlay_definition.addWidget(self.plot)
146             self.hlay_definition.setStretchFactor(self.plot, 1)
```

```python
147
148                 self.vlay_all.addLayout(self.hlay_definition)
149                 self.vlay_all.addLayout(self.hlay_buttons)
150
151                 # We load the default matrix A into the plot
152                 self.load_matrix(0)
153
154                 # We also enable the confirm button, because any visually defined matrix is valid
155                 self.button_confirm.setEnabled(True)
156
157         @pyqtSlot()
158         def update_confirm_button(self) -> None:
159             """Enable the confirm button.
160
161             .. note::
162                 The confirm button is always enabled in this dialog and this method is never actually used,
163                 so it's got an empty body. It's only here because we need to implement the abstract method.
164             """
165
166         @pyqtSlot(int)
167         def load_matrix(self, index: int) -> None:
168             """Show the selected matrix on the plot. If the matrix is None, show the identity."""
169             matrix = self.matrix_wrapper[self.selected_letter]
170
171             if matrix is None:
172                 matrix = self.matrix_wrapper['I']
173
174             self.plot.visualize_matrix_transformation(matrix)
175             self.plot.update()
176
177         @pyqtSlot()
178         def confirm_matrix(self) -> None:
179             """Confirm the matrix that's been defined visually."""
180             matrix: MatrixType = array([
181                 [self.plot.point_i[0], self.plot.point_j[0]],
182                 [self.plot.point_i[1], self.plot.point_j[1]]
183             ])
184
185             self.matrix_wrapper[self.selected_letter] = matrix
186             self.accept()
187
188
189 class DefineNumericallyDialog(DefineDialog):
190     """The dialog class that allows the user to define a new matrix numerically."""
191
192     def __init__(self, *args, matrix_wrapper: MatrixWrapper, **kwargs):
193         """Create the widgets and layout of the dialog.
194
195         :param MatrixWrapper matrix_wrapper: The MatrixWrapper that this dialog will mutate
196         """
197         super().__init__(*args, matrix_wrapper=matrix_wrapper, **kwargs)
198
199         # === Create the widgets
200
201         # tl = top left, br = bottom right, etc.
202         self.element_tl = QtWidgets.QLineEdit(self)
203         self.element_tl.textChanged.connect(self.update_confirm_button)
204         self.element_tl.setValidator(QDoubleValidator())
205
206         self.element_tr = QtWidgets.QLineEdit(self)
207         self.element_tr.textChanged.connect(self.update_confirm_button)
208         self.element_tr.setValidator(QDoubleValidator())
209
210         self.element_bl = QtWidgets.QLineEdit(self)
211         self.element_bl.textChanged.connect(self.update_confirm_button)
212         self.element_bl.setValidator(QDoubleValidator())
213
214         self.element_br = QtWidgets.QLineEdit(self)
215         self.element_br.textChanged.connect(self.update_confirm_button)
216         self.element_br.setValidator(QDoubleValidator())
217
218         self.matrix_elements = (self.element_tl, self.element_tr, self.element_bl, self.element_br)
219
```

```
220            # === Arrange the widgets
221
222            self.grid_matrix = QGridLayout()
223            self.grid_matrix.setSpacing(20)
224            self.grid_matrix.addWidget(self.element_tl, 0, 0)
225            self.grid_matrix.addWidget(self.element_tr, 0, 1)
226            self.grid_matrix.addWidget(self.element_bl, 1, 0)
227            self.grid_matrix.addWidget(self.element_br, 1, 1)
228
229            self.hlay_definition.addLayout(self.grid_matrix)
230
231            self.vlay_all.addLayout(self.hlay_definition)
232            self.vlay_all.addLayout(self.hlay_buttons)
233
234            # We load the default matrix A into the boxes
235            self.load_matrix(0)
236
237            self.element_tl.setFocus()
238
239        @pyqtSlot()
240        def update_confirm_button(self) -> None:
241            """Enable the confirm button if there are valid floats in every box."""
242            for elem in self.matrix_elements:
243                if not is_valid_float(elem.text()):
244                    # If they're not all numbers, then we can't confirm it
245                    self.button_confirm.setEnabled(False)
246                    return
247
248            # If we didn't find anything invalid
249            self.button_confirm.setEnabled(True)
250
251        @pyqtSlot(int)
252        def load_matrix(self, index: int) -> None:
253            """If the selected matrix is defined, load its values into the boxes."""
254            matrix = self.matrix_wrapper[self.selected_letter]
255
256            if matrix is None:
257                for elem in self.matrix_elements:
258                    elem.setText('')
259
260            else:
261                self.element_tl.setText(round_float(matrix[0][0]))
262                self.element_tr.setText(round_float(matrix[0][1]))
263                self.element_bl.setText(round_float(matrix[1][0]))
264                self.element_br.setText(round_float(matrix[1][1]))
265
266            self.update_confirm_button()
267
268        @pyqtSlot()
269        def confirm_matrix(self) -> None:
270            """Confirm the matrix in the boxes and assign it to the name in the combo box."""
271            matrix: MatrixType = array([
272                [float(self.element_tl.text()), float(self.element_tr.text())],
273                [float(self.element_bl.text()), float(self.element_br.text())]
274            ])
275
276            self.matrix_wrapper[self.selected_letter] = matrix
277            self.accept()
278
279
280 class DefineAsAnExpressionDialog(DefineDialog):
281     """The dialog class that allows the user to define a matrix as an expression of other matrices."""
282
283     def __init__(self, *args, matrix_wrapper: MatrixWrapper, **kwargs):
284         """Create the widgets and layout of the dialog.
285
286         :param MatrixWrapper matrix_wrapper: The MatrixWrapper that this dialog will mutate
287         """
288         super().__init__(*args, matrix_wrapper=matrix_wrapper, **kwargs)
289
290         self.setMinimumWidth(450)
291
292         # === Create the widgets
```

```
293
294            self.lineedit_expression_box = QtWidgets.QLineEdit(self)
295            self.lineedit_expression_box.setPlaceholderText('Enter matrix expression...')
296            self.lineedit_expression_box.textChanged.connect(self.update_confirm_button)
297            self.lineedit_expression_box.setValidator(MatrixExpressionValidator())
298
299            # === Arrange the widgets
300
301            self.hlay_definition.addWidget(self.lineedit_expression_box)
302
303            self.vlay_all.addLayout(self.hlay_definition)
304            self.vlay_all.addLayout(self.hlay_buttons)
305
306            # Load the matrix if it's defined as an expression
307            self.load_matrix(0)
308
309            self.lineedit_expression_box.setFocus()
310
311        @pyqtSlot()
312        def update_confirm_button(self) -> None:
313            """Enable the confirm button if the matrix expression is valid in the wrapper."""
314            text = self.lineedit_expression_box.text()
315            valid_expression = self.matrix_wrapper.is_valid_expression(text)
316
317            self.button_confirm.setEnabled(valid_expression and self.selected_letter not in text)
318
319        @pyqtSlot(int)
320        def load_matrix(self, index: int) -> None:
321            """If the selected matrix is defined an expression, load that expression into the box."""
322            if (expr := self.matrix_wrapper.get_expression(self.selected_letter)) is not None:
323                self.lineedit_expression_box.setText(expr)
324            else:
325                self.lineedit_expression_box.setText('')
326
327        @pyqtSlot()
328        def confirm_matrix(self) -> None:
329            """Evaluate the matrix expression and assign its value to the name in the combo box."""
330            self.matrix_wrapper[self.selected_letter] = self.lineedit_expression_box.text()
331            self.accept()
```

A.14 `gui/dialogs/misc.py`

```
1    #  lintrans - The linear transformation visualizer
2    #  Copyright (C) 2022 D. Dyson (DoctorDalek1963)
3    #
4    #  This program is licensed under GNU GPLv3, available here:
5    #  <https://www.gnu.org/licenses/gpl-3.0.html>
6
7    """This module provides miscellaneous dialog classes like :class:`AboutDialog`."""
8
9    from __future__ import annotations
10
11   import platform
12   from typing import Union
13
14   from PyQt5.QtCore import PYQT_VERSION_STR, QT_VERSION_STR, Qt
15   from PyQt5.QtWidgets import QDialog, QGridLayout, QLabel, QVBoxLayout, QWidget
16
17   import lintrans
18   from lintrans.matrices.utility import round_float
19   from lintrans.matrices import MatrixWrapper
20   from lintrans.typing_ import is_matrix_type, MatrixType
21
22
23   class FixedSizeDialog(QDialog):
24       """A simple superclass to create modal dialog boxes with fixed size.
25
26       We override the :meth:`open` method to set the fixed size as soon as the dialog is opened modally.
27       """
28
```

```python
29          def open(self) -> None:
30              """Override :meth:`QDialog.open` to set the dialog to a fixed size."""
31              super().open()
32              self.setFixedSize(self.size())
33
34
35      class AboutDialog(FixedSizeDialog):
36          """A simple dialog class to display information about the app to the user.
37
38          It only has an :meth:`__init__` method because it only has label widgets, so no other methods are necessary
39          here.
            """
40
41          def __init__(self, *args, **kwargs):
42              """Create an :class:`AboutDialog` object with all the label widgets."""
43              super().__init__(*args, **kwargs)
44
45              self.setWindowTitle('About lintrans')
46
47              # === Create the widgets
48
49              label_title = QLabel(self)
50              label_title.setText(f'lintrans (version {lintrans.__version__})')
51              label_title.setAlignment(Qt.AlignCenter)
52
53              font_title = label_title.font()
54              font_title.setPointSize(font_title.pointSize() * 2)
55              label_title.setFont(font_title)
56
57              label_version_info = QLabel(self)
58              label_version_info.setText(
59                  f'With Python version {platform.python_version()}\n'
60                  f'Qt version {QT_VERSION_STR} and PyQt5 version {PYQT_VERSION_STR}\n'
61                  f'Running on {platform.platform()}'
62              )
63              label_version_info.setAlignment(Qt.AlignCenter)
64
65              label_info = QLabel(self)
66              label_info.setText(
67                  'lintrans is a program designed to help visualise<br>'
68                  '2D linear transformations represented with matrices.<br><br>'
69                  "It's designed for teachers and students and any feedback<br>"
70                  'is greatly appreciated at <a href="https://github.com/DoctorDalek1963/lintrans" '
71                  'style="color: black;">my GitHub page</a><br>or via email '
72                  '(<a href="mailto:dyson.dyson@icloud.com" style="color: black;">dyson.dyson@icloud.com</a>).'
73              )
74              label_info.setAlignment(Qt.AlignCenter)
75              label_info.setTextFormat(Qt.RichText)
76              label_info.setOpenExternalLinks(True)
77
78              label_copyright = QLabel(self)
79              label_copyright.setText(
80                  'This program is free software.<br>Copyright 2021-2022 D. Dyson (DoctorDalek1963).<br>'
81                  'This program is licensed under GPLv3, which can be found '
82                  '<a href="https://www.gnu.org/licenses/gpl-3.0.html" style="color: black;">here</a>.'
83              )
84              label_copyright.setAlignment(Qt.AlignCenter)
85              label_copyright.setTextFormat(Qt.RichText)
86              label_copyright.setOpenExternalLinks(True)
87
88              # === Arrange the widgets
89
90              self.setContentsMargins(10, 10, 10, 10)
91
92              vlay = QVBoxLayout()
93              vlay.setSpacing(20)
94              vlay.addWidget(label_title)
95              vlay.addWidget(label_version_info)
96              vlay.addWidget(label_info)
97              vlay.addWidget(label_copyright)
98
99              self.setLayout(vlay)
100
```

```
101
102   class InfoPanelDialog(FixedSizeDialog):
103       """A simple dialog class to display an info panel that shows all currently defined matrices."""
104
105       def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
106           """Create the dialog box with all the widgets needed to show the information."""
107           super().__init__(*args, **kwargs)
108           self.wrapper = matrix_wrapper
109
110           self.setWindowTitle('Defined matrices')
111
112           grid_layout = QGridLayout()
113           grid_layout.setSpacing(20)
114
115           bold_font = self.font()
116           bold_font.setBold(True)
117
118           name_value_pair: tuple[str, Union[MatrixType, str]]
119
120           # Each defined matrix will get a widget group. Each group will be a label for the name,
121           # a label for '=', and a container widget to either show the matrix numerically, or to
122           # show the expression that it's defined as
123           for i, name_value_pair in enumerate(self.wrapper.get_defined_matrices()):
124               name, value = name_value_pair
125
126               # Create all the widgets first
127               label_name = QLabel(self)
128               label_name.setText(name)
129               label_name.setFont(bold_font)
130
131               label_equals = QLabel(self)
132               label_equals.setText('=')
133
134               widget_matrix = self._get_matrix_widget(value)
135
136               # We want columns of at most 6 widget groups
137               # This column variable manages which column of defined matrices we're on
138               # It's multiplied by 3 because all the widgets are in a single grid layout
139               # I could factor out each triplet of widgets for a defined matrix into a container widget,
140               # but I prefer to keep the widget count lower to reduce any possible lag
141               column = 3 * (i // 6)
142
143               grid_layout.addWidget(
144                   label_name,
145                   i - 2 * column,
146                   column,
147                   Qt.AlignCenter
148               )
149               grid_layout.addWidget(
150                   label_equals,
151                   i - 2 * column,
152                   column + 1,
153                   Qt.AlignCenter
154               )
155               grid_layout.addWidget(
156                   widget_matrix,
157                   i - 2 * column,
158                   column + 2,
159                   Qt.AlignCenter
160               )
161
162           self.setContentsMargins(10, 10, 10, 10)
163           self.setLayout(grid_layout)
164
165       def _get_matrix_widget(self, matrix: Union[MatrixType, str]) -> QWidget:
166           """Return a :class:`QWidget` containing the value of the matrix.
167
168           If the matrix is defined as an expression, it will be a simple :class:`QLabel`.
169           If the matrix is defined as a matrix, it will be a :class:`QWidget` container
170           with multiple :class:`QLabel` objects in it.
171           """
172           if isinstance(matrix, str):
173               label = QLabel(self)
```

```
174                    label.setText(matrix)
175                    return label
176
177            elif is_matrix_type(matrix):
178                # tl = top left, br = bottom right, etc.
179                label_tl = QLabel(self)
180                label_tl.setText(round_float(matrix[0][0]))
181
182                label_tr = QLabel(self)
183                label_tr.setText(round_float(matrix[0][1]))
184
185                label_bl = QLabel(self)
186                label_bl.setText(round_float(matrix[1][0]))
187
188                label_br = QLabel(self)
189                label_br.setText(round_float(matrix[1][1]))
190
191                # The parens need to be bigger than the numbers, but increasing the font size also
192                # makes the font thicker, so we have to reduce the font weight by the same factor
193                font_parens = self.font()
194                font_parens.setPointSize(int(font_parens.pointSize() * 2.5))
195                font_parens.setWeight(int(font_parens.weight() / 2.5))
196
197                label_paren_left = QLabel(self)
198                label_paren_left.setText('(')
199                label_paren_left.setFont(font_parens)
200
201                label_paren_right = QLabel(self)
202                label_paren_right.setText(')')
203                label_paren_right.setFont(font_parens)
204
205                container = QWidget(self)
206                grid_layout = QGridLayout()
207
208                grid_layout.addWidget(label_paren_left, 0, 0, -1, 1)
209                grid_layout.addWidget(label_tl, 0, 1)
210                grid_layout.addWidget(label_tr, 0, 2)
211                grid_layout.addWidget(label_bl, 1, 1)
212                grid_layout.addWidget(label_br, 1, 2)
213                grid_layout.addWidget(label_paren_right, 0, 3, -1, 1)
214
215                container.setLayout(grid_layout)
216
217                return container
218
219            raise ValueError('Matrix was not MatrixType or str')
```

A.15   `gui/dialogs/settings.py`

```
1     # lintrans - The linear transformation visualizer
2     # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4     # This program is licensed under GNU GPLv3, available here:
5     # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7     """This module provides dialogs to edit settings within the app."""
8
9     from __future__ import annotations
10
11    import abc
12    from typing import Dict
13
14    from PyQt5 import QtWidgets
15    from PyQt5.QtGui import QIntValidator, QKeyEvent, QKeySequence
16    from PyQt5.QtWidgets import QCheckBox, QGroupBox, QHBoxLayout, QShortcut, QSizePolicy, QSpacerItem, QVBoxLayout
17
18    from lintrans.gui.dialogs.misc import FixedSizeDialog
19    from lintrans.gui.settings import DisplaySettings
20
21
```

```
22    class SettingsDialog(FixedSizeDialog):
23        """An abstract superclass for other simple dialogs."""
24
25        def __init__(self, *args, **kwargs):
26            """Create the widgets and layout of the dialog, passing ``*args`` and ``**kwargs`` to super."""
27            super().__init__(*args, **kwargs)
28
29            # === Create the widgets
30
31            self.button_confirm = QtWidgets.QPushButton(self)
32            self.button_confirm.setText('Confirm')
33            self.button_confirm.clicked.connect(self.confirm_settings)
34            self.button_confirm.setToolTip('Confirm these new settings<br><b>(Ctrl + Enter)</b>')
35            QShortcut(QKeySequence('Ctrl+Return'), self).activated.connect(self.button_confirm.click)
36
37            self.button_cancel = QtWidgets.QPushButton(self)
38            self.button_cancel.setText('Cancel')
39            self.button_cancel.clicked.connect(self.reject)
40            self.button_cancel.setToolTip('Revert these settings<br><b>(Escape)</b>')
41
42            # === Arrange the widgets
43
44            self.setContentsMargins(10, 10, 10, 10)
45
46            self.hlay_buttons = QHBoxLayout()
47            self.hlay_buttons.setSpacing(20)
48            self.hlay_buttons.addItem(QSpacerItem(50, 5, hPolicy=QSizePolicy.Expanding, vPolicy=QSizePolicy.Minimum))
49            self.hlay_buttons.addWidget(self.button_cancel)
50            self.hlay_buttons.addWidget(self.button_confirm)
51
52            self.vlay_options = QVBoxLayout()
53            self.vlay_options.setSpacing(20)
54
55            self.vlay_all = QVBoxLayout()
56            self.vlay_all.setSpacing(20)
57            self.vlay_all.addLayout(self.vlay_options)
58            self.vlay_all.addLayout(self.hlay_buttons)
59
60            self.setLayout(self.vlay_all)
61
62        @abc.abstractmethod
63        def load_settings(self) -> None:
64            """Load the current settings into the widgets."""
65
66        @abc.abstractmethod
67        def confirm_settings(self) -> None:
68            """Confirm the settings chosen in the dialog."""
69
70
71    class DisplaySettingsDialog(SettingsDialog):
72        """The dialog to allow the user to edit the display settings."""
73
74        def __init__(self, *args, display_settings: DisplaySettings, **kwargs):
75            """Create the widgets and layout of the dialog.
76
77            :param DisplaySettings display_settings: The :class:`lintrans.gui.settings.DisplaySettings` object to mutate
78            """
79            super().__init__(*args, **kwargs)
80
81            self.display_settings = display_settings
82            self.setWindowTitle('Change display settings')
83
84            self.dict_checkboxes: Dict[str, QCheckBox] = dict()
85
86            # === Create the widgets
87
88            # Basic stuff
89
90            self.checkbox_draw_background_grid = QCheckBox(self)
91            self.checkbox_draw_background_grid.setText('Draw &background grid')
92            self.checkbox_draw_background_grid.setToolTip(
93                'Draw the background grid (axes are always drawn)'
94            )
```

```
 95                self.dict_checkboxes['b'] = self.checkbox_draw_background_grid
 96
 97                self.checkbox_draw_transformed_grid = QCheckBox(self)
 98                self.checkbox_draw_transformed_grid.setText('Draw t&ransformed grid')
 99                self.checkbox_draw_transformed_grid.setToolTip(
100                    'Draw the transformed grid (vectors are handled separately)'
101                )
102                self.dict_checkboxes['r'] = self.checkbox_draw_transformed_grid
103
104                self.checkbox_draw_basis_vectors = QCheckBox(self)
105                self.checkbox_draw_basis_vectors.setText('Draw basis &vectors')
106                self.checkbox_draw_basis_vectors.setToolTip(
107                    'Draw the transformed basis vectors'
108                )
109                self.dict_checkboxes['v'] = self.checkbox_draw_basis_vectors
110
111                # Animations
112
113                self.checkbox_smoothen_determinant = QCheckBox(self)
114                self.checkbox_smoothen_determinant.setText('&Smoothen determinant')
115                self.checkbox_smoothen_determinant.setToolTip(
116                    'Smoothly animate the determinant transition during animation (if possible)'
117                )
118                self.dict_checkboxes['s'] = self.checkbox_smoothen_determinant
119
120                self.checkbox_applicative_animation = QCheckBox(self)
121                self.checkbox_applicative_animation.setText('&Applicative animation')
122                self.checkbox_applicative_animation.setToolTip(
123                    'Animate the new transformation applied to the current one,\n'
124                    'rather than just that transformation on its own'
125                )
126                self.dict_checkboxes['a'] = self.checkbox_applicative_animation
127
128                self.label_animation_time = QtWidgets.QLabel(self)
129                self.label_animation_time.setText('Total animation length (ms)')
130                self.label_animation_time.setToolTip(
131                    'How long it takes for an animation to complete'
132                )
133
134                self.lineedit_animation_time = QtWidgets.QLineEdit(self)
135                self.lineedit_animation_time.setValidator(QIntValidator(1, 9999, self))
136
137                self.label_animation_pause_length = QtWidgets.QLabel(self)
138                self.label_animation_pause_length.setText('Animation pause length (ms)')
139                self.label_animation_pause_length.setToolTip(
140                    'How many milliseconds to pause for in comma-separated animations'
141                )
142
143                self.lineedit_animation_pause_length = QtWidgets.QLineEdit(self)
144                self.lineedit_animation_pause_length.setValidator(QIntValidator(1, 999, self))
145
146                # Matrix info
147
148                self.checkbox_draw_determinant_parallelogram = QCheckBox(self)
149                self.checkbox_draw_determinant_parallelogram.setText('Draw &determinant parallelogram')
150                self.checkbox_draw_determinant_parallelogram.setToolTip(
151                    'Shade the parallelogram representing the determinant of the matrix'
152                )
153                self.checkbox_draw_determinant_parallelogram.clicked.connect(self.update_gui)
154                self.dict_checkboxes['d'] = self.checkbox_draw_determinant_parallelogram
155
156                self.checkbox_show_determinant_value = QCheckBox(self)
157                self.checkbox_show_determinant_value.setText('Show de&terminant value')
158                self.checkbox_show_determinant_value.setToolTip(
159                    'Show the value of the determinant inside the parallelogram'
160                )
161                self.dict_checkboxes['t'] = self.checkbox_show_determinant_value
162
163                self.checkbox_draw_eigenvectors = QCheckBox(self)
164                self.checkbox_draw_eigenvectors.setText('Draw &eigenvectors')
165                self.checkbox_draw_eigenvectors.setToolTip('Draw the eigenvectors of the transformations')
166                self.dict_checkboxes['e'] = self.checkbox_draw_eigenvectors
167
```

```
168                    self.checkbox_draw_eigenlines = QCheckBox(self)
169                    self.checkbox_draw_eigenlines.setText('Draw eigen&lines')
170                    self.checkbox_draw_eigenlines.setToolTip('Draw the eigenlines (invariant lines) of the transformations')
171                    self.dict_checkboxes['l'] = self.checkbox_draw_eigenlines
172
173                    # === Arrange the widgets in QGroupBoxes
174
175                    # Basic stuff
176
177                    self.vlay_groupbox_basic_stuff = QVBoxLayout()
178                    self.vlay_groupbox_basic_stuff.setSpacing(20)
179                    self.vlay_groupbox_basic_stuff.addWidget(self.checkbox_draw_background_grid)
180                    self.vlay_groupbox_basic_stuff.addWidget(self.checkbox_draw_transformed_grid)
181                    self.vlay_groupbox_basic_stuff.addWidget(self.checkbox_draw_basis_vectors)
182
183                    self.groupbox_basic_stuff = QGroupBox('Basic stuff', self)
184                    self.groupbox_basic_stuff.setLayout(self.vlay_groupbox_basic_stuff)
185
186                    # Animations
187
188                    self.hlay_animation_time = QHBoxLayout()
189                    self.hlay_animation_time.addWidget(self.label_animation_time)
190                    self.hlay_animation_time.addWidget(self.lineedit_animation_time)
191
192                    self.hlay_animation_pause_length = QHBoxLayout()
193                    self.hlay_animation_pause_length.addWidget(self.label_animation_pause_length)
194                    self.hlay_animation_pause_length.addWidget(self.lineedit_animation_pause_length)
195
196                    self.vlay_groupbox_animations = QVBoxLayout()
197                    self.vlay_groupbox_animations.setSpacing(20)
198                    self.vlay_groupbox_animations.addWidget(self.checkbox_smoothen_determinant)
199                    self.vlay_groupbox_animations.addWidget(self.checkbox_applicative_animation)
200                    self.vlay_groupbox_animations.addLayout(self.hlay_animation_time)
201                    self.vlay_groupbox_animations.addLayout(self.hlay_animation_pause_length)
202
203                    self.groupbox_animations = QGroupBox('Animations', self)
204                    self.groupbox_animations.setLayout(self.vlay_groupbox_animations)
205
206                    # Matrix info
207
208                    self.vlay_groupbox_matrix_info = QVBoxLayout()
209                    self.vlay_groupbox_matrix_info.setSpacing(20)
210                    self.vlay_groupbox_matrix_info.addWidget(self.checkbox_draw_determinant_parallelogram)
211                    self.vlay_groupbox_matrix_info.addWidget(self.checkbox_show_determinant_value)
212                    self.vlay_groupbox_matrix_info.addWidget(self.checkbox_draw_eigenvectors)
213                    self.vlay_groupbox_matrix_info.addWidget(self.checkbox_draw_eigenlines)
214
215                    self.groupbox_matrix_info = QGroupBox('Matrix info', self)
216                    self.groupbox_matrix_info.setLayout(self.vlay_groupbox_matrix_info)
217
218                    # Now arrange the groupboxes
219                    self.vlay_options.addWidget(self.groupbox_basic_stuff)
220                    self.vlay_options.addWidget(self.groupbox_animations)
221                    self.vlay_options.addWidget(self.groupbox_matrix_info)
222
223                    # Finally, we load the current settings and update the GUI
224                    self.load_settings()
225                    self.update_gui()
226
227        def load_settings(self) -> None:
228            """Load the current display settings into the widgets."""
229            # Basic stuff
230            self.checkbox_draw_background_grid.setChecked(self.display_settings.draw_background_grid)
231            self.checkbox_draw_transformed_grid.setChecked(self.display_settings.draw_transformed_grid)
232            self.checkbox_draw_basis_vectors.setChecked(self.display_settings.draw_basis_vectors)
233
234            # Animations
235            self.checkbox_smoothen_determinant.setChecked(self.display_settings.smoothen_determinant)
236            self.checkbox_applicative_animation.setChecked(self.display_settings.applicative_animation)
237            self.lineedit_animation_time.setText(str(self.display_settings.animation_time))
238            self.lineedit_animation_pause_length.setText(str(self.display_settings.animation_pause_length))
239
240            # Matrix info
```

```python
241                self.checkbox_draw_determinant_parallelogram.setChecked(self.display_settings.draw_determinant ⏎
                   ↪   _parallelogram)
242                self.checkbox_show_determinant_value.setChecked(self.display_settings.show_determinant_value)
243                self.checkbox_draw_eigenvectors.setChecked(self.display_settings.draw_eigenvectors)
244                self.checkbox_draw_eigenlines.setChecked(self.display_settings.draw_eigenlines)
245
246            def confirm_settings(self) -> None:
247                """Build a :class:`lintrans.gui.settings.DisplaySettings` object and assign it."""
248                # Basic stuff
249                self.display_settings.draw_background_grid = self.checkbox_draw_background_grid.isChecked()
250                self.display_settings.draw_transformed_grid = self.checkbox_draw_transformed_grid.isChecked()
251                self.display_settings.draw_basis_vectors = self.checkbox_draw_basis_vectors.isChecked()
252
253                # Animations
254                self.display_settings.smoothen_determinant = self.checkbox_smoothen_determinant.isChecked()
255                self.display_settings.applicative_animation = self.checkbox_applicative_animation.isChecked()
256                self.display_settings.animation_time = int(self.lineedit_animation_time.text())
257                self.display_settings.animation_pause_length = int(self.lineedit_animation_pause_length.text())
258
259                # Matrix info
260                self.display_settings.draw_determinant_parallelogram =
                   ↪   self.checkbox_draw_determinant_parallelogram.isChecked()
261                self.display_settings.show_determinant_value = self.checkbox_show_determinant_value.isChecked()
262                self.display_settings.draw_eigenvectors = self.checkbox_draw_eigenvectors.isChecked()
263                self.display_settings.draw_eigenlines = self.checkbox_draw_eigenlines.isChecked()
264
265                self.accept()
266
267            def update_gui(self) -> None:
268                """Update the GUI according to other widgets in the GUI.
269
270                For example, this method updates which checkboxes are enabled based on the values of other checkboxes.
271                """
272                self.checkbox_show_determinant_value.setEnabled(self.checkbox_draw_determinant_parallelogram.isChecked())
273
274            def keyPressEvent(self, event: QKeyEvent) -> None:
275                """Handle a :class:`QKeyEvent` by manually activating toggling checkboxes.
276
277                Qt handles these shortcuts automatically and allows the user to do ``Alt + Key``
278                to activate a simple shortcut defined with ``&``. However, I like to be able to
279                just hit ``Key`` and have the shortcut activate.
280                """
281                letter = event.text().lower()
282                key = event.key()
283
284                if letter in self.dict_checkboxes:
285                    self.dict_checkboxes[letter].animateClick()
286
287                # Return or keypad enter
288                elif key == 0x01000004 or key == 0x01000005:
289                    self.button_confirm.click()
290
291                # Escape
292                elif key == 0x01000000:
293                    self.button_cancel.click()
294
295                else:
296                    event.ignore()
```

A.16 gui/plots/__init__.py

```python
1    # lintrans - The linear transformation visualizer
2    # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4    # This program is licensed under GNU GPLv3, available here:
5    # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7    """This package provides widgets for the visualization plot in the main window and the visual definition dialog."""
8
9    from .classes import BackgroundPlot, VectorGridPlot
```

```
10     from .widgets import DefineVisuallyWidget, VisualizeTransformationWidget
11
12     __all__ = ['BackgroundPlot', 'DefineVisuallyWidget', 'VectorGridPlot', 'VisualizeTransformationWidget']
```

A.17 `gui/plots/classes.py`

```
1     # lintrans - The linear transformation visualizer
2     # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4     # This program is licensed under GNU GPLv3, available here:
5     # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7     """This module provides superclasses for plotting transformations."""
8
9     from __future__ import annotations
10
11     from abc import abstractmethod
12     from typing import Iterable, List, Tuple
13
14     import numpy as np
15     from nptyping import Float, NDArray
16     from PyQt5.QtCore import QPoint, QRectF, Qt
17     from PyQt5.QtGui import QBrush, QColor, QPainter, QPainterPath, QPaintEvent, QPen, QWheelEvent
18     from PyQt5.QtWidgets import QWidget
19
20     from lintrans.typing_ import MatrixType
21
22
23     class BackgroundPlot(QWidget):
24         """This class provides a background for plotting, as well as setup for a Qt widget.
25
26         This class provides a background (untransformed) plane, and all the backend
27         details for a Qt application, but does not provide useful functionality. To
28         be useful, this class must be subclassed and behaviour must be implemented
29         by the subclass.
30
31         .. warning:: This class should never be directly instantiated, only subclassed.
32
33         .. note::
34             I would make this class have ``metaclass=abc.ABCMeta``, but I can't because it subclasses :class:`QWidget`,
35             and every superclass of a class must have the same metaclass, and :class:`QWidget` is not an abstract class.
36         """
37
38         default_grid_spacing: int = 85
39         minimum_grid_spacing: int = 5
40
41         def __init__(self, *args, **kwargs):
42             """Create the widget and setup backend stuff for rendering.
43
44             .. note:: ``*args`` and ``**kwargs`` are passed the superclass constructor (:class:`QWidget`).
45             """
46             super().__init__(*args, **kwargs)
47
48             self.setAutoFillBackground(True)
49
50             # Set the background to white
51             palette = self.palette()
52             palette.setColor(self.backgroundRole(), Qt.white)
53             self.setPalette(palette)
54
55             # Set the grid colour to grey and the axes colour to black
56             self.colour_background_grid = QColor('#808080')
57             self.colour_background_axes = QColor('#000000')
58
59             self.grid_spacing = BackgroundPlot.default_grid_spacing
60             self.width_background_grid: float = 0.3
61
62         @property
63         def canvas_origin(self) -> Tuple[int, int]:
64             """Return the canvas coords of the grid origin.
```

```python
 65
 66            The return value is intended to be unpacked and passed to a :meth:`QPainter.drawLine:iiii` call.
 67
 68            See :meth:`canvas_coords`.
 69
 70            :returns: The canvas coordinates of the grid origin
 71            :rtype: Tuple[int, int]
 72            """
 73            return self.width() // 2, self.height() // 2
 74
 75        def canvas_x(self, x: float) -> int:
 76            """Convert an x coordinate from grid coords to canvas coords."""
 77            return int(self.canvas_origin[0] + x * self.grid_spacing)
 78
 79        def canvas_y(self, y: float) -> int:
 80            """Convert a y coordinate from grid coords to canvas coords."""
 81            return int(self.canvas_origin[1] - y * self.grid_spacing)
 82
 83        def canvas_coords(self, x: float, y: float) -> Tuple[int, int]:
 84            """Convert a coordinate from grid coords to canvas coords.
 85
 86            This method is intended to be used like
 87
 88            .. code::
 89
 90                painter.drawLine(*self.canvas_coords(x1, y1), *self.canvas_coords(x2, y2))
 91
 92            or like
 93
 94            .. code::
 95
 96                painter.drawLine(*self.canvas_origin, *self.canvas_coords(x, y))
 97
 98            See :attr:`canvas_origin`.
 99
100            :param float x: The x component of the grid coordinate
101            :param float y: The y component of the grid coordinate
102            :returns: The resultant canvas coordinates
103            :rtype: Tuple[int, int]
104            """
105            return self.canvas_x(x), self.canvas_y(y)
106
107        def grid_corner(self) -> Tuple[float, float]:
108            """Return the grid coords of the top right corner."""
109            return self.width() / (2 * self.grid_spacing), self.height() / (2 * self.grid_spacing)
110
111        def grid_coords(self, x: int, y: int) -> Tuple[float, float]:
112            """Convert a coordinate from canvas coords to grid coords.
113
114            :param int x: The x component of the canvas coordinate
115            :param int y: The y component of the canvas coordinate
116            :returns: The resultant grid coordinates
117            :rtype: Tuple[float, float]
118            """
119            # We get the maximum grid coords and convert them into canvas coords
120            return (x - self.canvas_origin[0]) / self.grid_spacing, (-y + self.canvas_origin[1]) / self.grid_spacing
121
122        @abstractmethod
123        def paintEvent(self, event: QPaintEvent) -> None:
124            """Handle a :class:`QPaintEvent`.
125
126            .. note:: This method is abstract and must be overridden by all subclasses.
127            """
128
129        def draw_background(self, painter: QPainter, draw_grid: bool) -> None:
130            """Draw the background grid.
131
132            .. note:: This method is just a utility method for subclasses to use to render the background grid.
133
134            :param QPainter painter: The painter to draw the background with
135            :param bool draw_grid: Whether to draw the grid lines
136            """
137            if draw_grid:
```

237

```
138                        painter.setPen(QPen(self.colour_background_grid, self.width_background_grid))
139
140                        # Draw equally spaced vertical lines, starting in the middle and going out
141                        # We loop up to half of the width. This is because we draw a line on each side in each iteration
142                        for x in range(self.width() // 2 + self.grid_spacing, self.width(), self.grid_spacing):
143                            painter.drawLine(x, 0, x, self.height())
144                            painter.drawLine(self.width() - x, 0, self.width() - x, self.height())
145
146                        # Same with the horizontal lines
147                        for y in range(self.height() // 2 + self.grid_spacing, self.height(), self.grid_spacing):
148                            painter.drawLine(0, y, self.width(), y)
149                            painter.drawLine(0, self.height() - y, self.width(), self.height() - y)
150
151                    # Now draw the axes
152                    painter.setPen(QPen(self.colour_background_axes, self.width_background_grid))
153                    painter.drawLine(self.width() // 2, 0, self.width() // 2, self.height())
154                    painter.drawLine(0, self.height() // 2, self.width(), self.height() // 2)
155
156            def wheelEvent(self, event: QWheelEvent) -> None:
157                """Handle a :class:`QWheelEvent` by zooming in or our of the grid."""
158                # angleDelta() returns a number of units equal to 8 times the number of degrees rotated
159                degrees = event.angleDelta() / 8
160
161                if degrees is not None:
162                    new_spacing = max(1, self.grid_spacing + degrees.y())
163
164                    if new_spacing >= self.minimum_grid_spacing:
165                        self.grid_spacing = new_spacing
166
167                event.accept()
168                self.update()
169
170
171    class VectorGridPlot(BackgroundPlot):
172        """This class represents a background plot, with vectors and their grid drawn on top.
173
174        This class should be subclassed to be used for visualization and matrix definition widgets.
175        All useful behaviour should be implemented by any subclass.
176
177        .. warning:: This class should never be directly instantiated, only subclassed.
178        """
179
180        def __init__(self, *args, **kwargs):
181            """Create the widget with ``point_i`` and ``point_j`` attributes.
182
183            .. note:: ``*args`` and ``**kwargs`` are passed to the superclass constructor (:class:`BackgroundPlot`).
184            """
185            super().__init__(*args, **kwargs)
186
187            self.point_i: Tuple[float, float] = (1., 0.)
188            self.point_j: Tuple[float, float] = (0., 1.)
189
190            self.colour_i = QColor('#0808d8')
191            self.colour_j = QColor('#e90000')
192            self.colour_eigen = QColor('#13cf00')
193            self.colour_text = QColor('#000000')
194
195            self.width_vector_line = 1.8
196            self.width_transformed_grid = 0.8
197
198            self.arrowhead_length = 0.15
199
200            self.max_parallel_lines = 150
201
202        @property
203        def matrix(self) -> MatrixType:
204            """Return the assembled matrix of the basis vectors."""
205            return np.array([
206                [self.point_i[0], self.point_j[0]],
207                [self.point_i[1], self.point_j[1]]
208            ])
209
210        @property
```

```python
211         def det(self) -> float:
212             """Return the determinant of the assembled matrix."""
213             return float(np.linalg.det(self.matrix))
214
215         @property
216         def eigs(self) -> Iterable[Tuple[float, NDArray[(1, 2), Float]]]:
217             """Return the eigenvalues and eigenvectors zipped together to be iterated over.
218
219             :rtype: Iterable[Tuple[float, NDArray[(1, 2), Float]]]
220             """
221             values, vectors = np.linalg.eig(self.matrix)
222             return zip(values, vectors.T)
223
224         @abstractmethod
225         def paintEvent(self, event: QPaintEvent) -> None:
226             """Handle a :class:`QPaintEvent`.
227
228             .. note:: This method is abstract and must be overridden by all subclasses.
229             """
230
231         def draw_parallel_lines(self, painter: QPainter, vector: Tuple[float, float], point: Tuple[float, float]) ->
232     ↪   None:
            """Draw a set of evenly spaced grid lines parallel to ``vector`` intersecting ``point``.
233
234             :param QPainter painter: The painter to draw the lines with
235             :param vector: The vector to draw the grid lines parallel to
236             :type vector: Tuple[float, float]
237             :param point: The point for the lines to intersect with
238             :type point: Tuple[float, float]
239             """
240             max_x, max_y = self.grid_corner()
241             vector_x, vector_y = vector
242             point_x, point_y = point
243
244             # If the determinant is 0
245             if abs(vector_x * point_y - vector_y * point_x) < 1e-12:
246                 rank = np.linalg.matrix_rank(
247                     np.array([
248                         [vector_x, point_x],
249                         [vector_y, point_y]
250                     ])
251                 )
252
253                 # If the matrix is rank 1, then we can draw the column space line
254                 if rank == 1:
255                     # If the vector does not have a 0 x or y component, then we can just draw the line
256                     if abs(vector_x) > 1e-12 and abs(vector_y) > 1e-12:
257                         self.draw_oblique_line(painter, vector_y / vector_x, 0)
258
259                     # Otherwise, we have to draw lines along the axes
260                     elif abs(vector_x) > 1e-12 and abs(vector_y) < 1e-12:
261                         painter.drawLine(0, self.height() // 2, self.width(), self.height() // 2)
262
263                     elif abs(vector_x) < 1e-12 and abs(vector_y) > 1e-12:
264                         painter.drawLine(self.width() // 2, 0, self.width() // 2, self.height())
265
266                     # If the vector is (0, 0), then don't draw a line for it
267                     else:
268                         return
269
270                 # If the rank is 0, then we don't draw any lines
271                 else:
272                     return
273
274             elif abs(vector_x) < 1e-12 and abs(vector_y) < 1e-12:
275                 # If both components of the vector are practically 0, then we can't render any grid lines
276                 return
277
278             # Draw vertical lines
279             elif abs(vector_x) < 1e-12:
280                 painter.drawLine(self.canvas_x(0), 0, self.canvas_x(0), self.height())
281
282                 for i in range(max(abs(int(max_x / point_x)), self.max_parallel_lines)):
```

239

```
283                    painter.drawLine(
284                        self.canvas_x((i + 1) * point_x),
285                        0,
286                        self.canvas_x((i + 1) * point_x),
287                        self.height()
288                    )
289                    painter.drawLine(
290                        self.canvas_x(-1 * (i + 1) * point_x),
291                        0,
292                        self.canvas_x(-1 * (i + 1) * point_x),
293                        self.height()
294                    )
295
296            # Draw horizontal lines
297            elif abs(vector_y) < 1e-12:
298                painter.drawLine(0, self.canvas_y(0), self.width(), self.canvas_y(0))
299
300                for i in range(max(abs(int(max_y / point_y)), self.max_parallel_lines)):
301                    painter.drawLine(
302                        0,
303                        self.canvas_y((i + 1) * point_y),
304                        self.width(),
305                        self.canvas_y((i + 1) * point_y)
306                    )
307                    painter.drawLine(
308                        0,
309                        self.canvas_y(-1 * (i + 1) * point_y),
310                        self.width(),
311                        self.canvas_y(-1 * (i + 1) * point_y)
312                    )
313
314            # If the line is oblique, then we can use y = mx + c
315            else:
316                m = vector_y / vector_x
317                c = point_y - m * point_x
318
319                self.draw_oblique_line(painter, m, 0)
320
321                # We don't want to overshoot the max number of parallel lines,
322                # but we should also stop looping as soon as we can't draw any more lines
323                for i in range(1, self.max_parallel_lines + 1):
324                    if not self.draw_pair_of_oblique_lines(painter, m, i * c):
325                        break
326
327        def draw_pair_of_oblique_lines(self, painter: QPainter, m: float, c: float) -> bool:
328            """Draw a pair of oblique lines, using the equation y = mx + c.
329
330            This method just calls :meth:`draw_oblique_line` with ``c`` and ``-c``,
331            and returns True if either call returned True.
332
333            :param QPainter painter: The painter to draw the vectors and grid lines with
334            :param float m: The gradient of the lines to draw
335            :param float c: The y-intercept of the lines to draw. We use the positive and negative versions
336            :returns bool: Whether we were able to draw any lines on the canvas
337            """
338            return any([
339                self.draw_oblique_line(painter, m, c),
340                self.draw_oblique_line(painter, m, -c)
341            ])
342
343        def draw_oblique_line(self, painter: QPainter, m: float, c: float) -> bool:
344            """Draw an oblique line, using the equation y = mx + c.
345
346            We only draw the part of the line that fits within the canvas, returning True if
347            we were able to draw a line within the boundaries, and False if we couldn't draw a line
348
349            :param QPainter painter: The painter to draw the vectors and grid lines with
350            :param float m: The gradient of the line to draw
351            :param float c: The y-intercept of the line to draw
352            :returns bool: Whether we were able to draw a line on the canvas
353            """
354            max_x, max_y = self.grid_corner()
355
```

```python
356            # These variable names are shortened for convenience
357            # myi is max_y_intersection, mmyi is minus_max_y_intersection, etc.
358            myi = (max_y - c) / m
359            mmyi = (-max_y - c) / m
360            mxi = max_x * m + c
361            mmxi = -max_x * m + c
362
363            # The inner list here is a list of coords, or None
364            # If an intersection fits within the bounds, then we keep its coord,
365            # else it is None, and then gets discarded from the points list
366            # By the end, points is a list of two coords, or an empty list
367            points: List[Tuple[float, float]] = [
368                x for x in [
369                    (myi, max_y) if -max_x < myi < max_x else None,
370                    (mmyi, -max_y) if -max_x < mmyi < max_x else None,
371                    (max_x, mxi) if -max_y < mxi < max_y else None,
372                    (-max_x, mmxi) if -max_y < mmxi < max_y else None
373                ] if x is not None
374            ]
375
376            # If no intersections fit on the canvas
377            if len(points) < 2:
378                return False
379
380            # If we can, then draw the line
381            else:
382                painter.drawLine(
383                    *self.canvas_coords(*points[0]),
384                    *self.canvas_coords(*points[1])
385                )
386                return True
387
388    def draw_transformed_grid(self, painter: QPainter) -> None:
389        """Draw the transformed version of the grid, given by the basis vectors.
390
391        .. note:: This method draws the grid, but not the basis vectors. Use :meth:`draw_basis_vectors` to draw
          ↪  them.
392
393        :param QPainter painter: The painter to draw the grid lines with
394        """
395        # Draw all the parallel lines
396        painter.setPen(QPen(self.colour_i, self.width_transformed_grid))
397        self.draw_parallel_lines(painter, self.point_i, self.point_j)
398        painter.setPen(QPen(self.colour_j, self.width_transformed_grid))
399        self.draw_parallel_lines(painter, self.point_j, self.point_i)
400
401    def draw_arrowhead_away_from_origin(self, painter: QPainter, point: Tuple[float, float]) -> None:
402        """Draw an arrowhead at ``point``, pointing away from the origin.
403
404        :param QPainter painter: The painter to draw the arrowhead with
405        :param point: The point to draw the arrowhead at, given in grid coords
406        :type point: Tuple[float, float]
407        """
408        # This algorithm was adapted from a C# algorithm found at
409        # http://csharphelper.com/blog/2014/12/draw-lines-with-arrowheads-in-c/
410
411        # Get the x and y coords of the point, and then normalize them
412        # We have to normalize them, or else the size of the arrowhead will
413        # scale with the distance of the point from the origin
414        x, y = point
415        vector_length = np.sqrt(x * x + y * y)
416
417        if vector_length < 1e-12:
418            return
419
420        nx = x / vector_length
421        ny = y / vector_length
422
423        # We choose a length and find the steps in the x and y directions
424        length = min(
425            self.arrowhead_length * self.default_grid_spacing / self.grid_spacing,
426            vector_length
427        )
```

```
428                    dx = length * (-nx - ny)
429                    dy = length * (nx - ny)
430
431                    # Then we just plot those lines
432                    painter.drawLine(*self.canvas_coords(x, y), *self.canvas_coords(x + dx, y + dy))
433                    painter.drawLine(*self.canvas_coords(x, y), *self.canvas_coords(x - dy, y + dx))
434
435            def draw_position_vector(self, painter: QPainter, point: Tuple[float, float], colour: QColor) -> None:
436                """Draw a vector from the origin to the given point.
437
438                :param QPainter painter: The painter to draw the position vector with
439                :param point: The tip of the position vector in grid coords
440                :type point: Tuple[float, float]
441                :param QColor colour: The colour to draw the position vector in
442                """
443                painter.setPen(QPen(colour, self.width_vector_line))
444                painter.drawLine(*self.canvas_origin, *self.canvas_coords(*point))
445                self.draw_arrowhead_away_from_origin(painter, point)
446
447            def draw_basis_vectors(self, painter: QPainter) -> None:
448                """Draw arrowheads at the tips of the basis vectors.
449
450                :param QPainter painter: The painter to draw the basis vectors with
451                """
452                self.draw_position_vector(painter, self.point_i, self.colour_i)
453                self.draw_position_vector(painter, self.point_j, self.colour_j)
454
455            def draw_determinant_parallelogram(self, painter: QPainter) -> None:
456                """Draw the parallelogram of the determinant of the matrix.
457
458                :param QPainter painter: The painter to draw the parallelogram with
459                """
460                if self.det == 0:
461                    return
462
463                path = QPainterPath()
464                path.moveTo(*self.canvas_origin)
465                path.lineTo(*self.canvas_coords(*self.point_i))
466                path.lineTo(*self.canvas_coords(self.point_i[0] + self.point_j[0], self.point_i[1] + self.point_j[1]))
467                path.lineTo(*self.canvas_coords(*self.point_j))
468
469                color = (16, 235, 253) if self.det > 0 else (253, 34, 16)
470                brush = QBrush(QColor(*color, alpha=128), Qt.SolidPattern)
471
472                painter.fillPath(path, brush)
473
474            def draw_determinant_text(self, painter: QPainter) -> None:
475                """Write the string value of the determinant in the middle of the parallelogram.
476
477                :param QPainter painter: The painter to draw the determinant text with
478                """
479                painter.setPen(QPen(self.colour_text, self.width_vector_line))
480
481                # We're building a QRect that encloses the determinant parallelogram
482                # Then we can center the text in this QRect
483                coords: List[Tuple[float, float]] = [
484                    (0, 0),
485                    self.point_i,
486                    self.point_j,
487                    (
488                        self.point_i[0] + self.point_j[0],
489                        self.point_i[1] + self.point_j[1]
490                    )
491                ]
492
493                xs = [t[0] for t in coords]
494                ys = [t[1] for t in coords]
495
496                top_left = QPoint(*self.canvas_coords(min(xs), max(ys)))
497                bottom_right = QPoint(*self.canvas_coords(max(xs), min(ys)))
498
499                rect = QRectF(top_left, bottom_right)
500
```

```
501                 painter.drawText(
502                     rect,
503                     Qt.AlignHCenter | Qt.AlignVCenter,
504                     f'{self.det:.2f}'
505                 )
506
507         def draw_eigenvectors(self, painter: QPainter) -> None:
508             """Draw the eigenvectors of the displayed matrix transformation.
509
510             :param QPainter painter: The painter to draw the eigenvectors with
511             """
512             for value, vector in self.eigs:
513                 x = value * vector[0]
514                 y = value * vector[1]
515
516                 if x.imag != 0 or y.imag != 0:
517                     continue
518
519                 self.draw_position_vector(painter, (x, y), self.colour_eigen)
520
521                 # Now we need to draw the eigenvalue at the tip of the eigenvector
522
523                 offset = 3
524                 top_left: QPoint
525                 bottom_right: QPoint
526                 alignment_flags: int
527
528                 if x >= 0 and y >= 0:   # Q1
529                     top_left = QPoint(self.canvas_x(x) + offset, 0)
530                     bottom_right = QPoint(self.width(), self.canvas_y(y) - offset)
531                     alignment_flags = Qt.AlignLeft | Qt.AlignBottom
532
533                 elif x < 0 and y >= 0:   # Q2
534                     top_left = QPoint(0, 0)
535                     bottom_right = QPoint(self.canvas_x(x) - offset, self.canvas_y(y) - offset)
536                     alignment_flags = Qt.AlignRight | Qt.AlignBottom
537
538                 elif x < 0 and y < 0:   # Q3
539                     top_left = QPoint(0, self.canvas_y(y) + offset)
540                     bottom_right = QPoint(self.canvas_x(x) - offset, self.height())
541                     alignment_flags = Qt.AlignRight | Qt.AlignTop
542
543                 else:   # Q4
544                     top_left = QPoint(self.canvas_x(x) + offset, self.canvas_y(y) + offset)
545                     bottom_right = QPoint(self.width(), self.height())
546                     alignment_flags = Qt.AlignLeft | Qt.AlignTop
547
548                 painter.setPen(QPen(self.colour_text, self.width_vector_line))
549                 painter.drawText(QRectF(top_left, bottom_right), alignment_flags, f'{value:.2f}')
550
551         def draw_eigenlines(self, painter: QPainter) -> None:
552             """Draw the eigenlines (invariant lines).
553
554             :param QPainter painter: The painter to draw the eigenlines with
555             """
556             painter.setPen(QPen(self.colour_eigen, self.width_transformed_grid))
557
558             for value, vector in self.eigs:
559                 if value.imag != 0:
560                     continue
561
562                 x, y = vector
563
564                 if x == 0:
565                     x_mid = int(self.width() / 2)
566                     painter.drawLine(x_mid, 0, x_mid, self.height())
567
568                 elif y == 0:
569                     y_mid = int(self.height() / 2)
570                     painter.drawLine(0, y_mid, self.width(), y_mid)
571
572                 else:
573                     self.draw_oblique_line(painter, y / x, 0)
```

243

A.18 `gui/plots/widgets.py`

```
1    # lintrans - The linear transformation visualizer
2    # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4    # This program is licensed under GNU GPLv3, available here:
5    # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7    """This module provides the actual widgets that can be used to visualize transformations in the GUI."""
8
9    from __future__ import annotations
10
11   from math import ceil, dist, floor
12   from typing import List, Tuple
13
14   from PyQt5.QtCore import Qt
15   from PyQt5.QtGui import QMouseEvent, QPainter, QPaintEvent
16
17   from lintrans.typing_ import MatrixType
18   from lintrans.gui.settings import DisplaySettings
19   from .classes import VectorGridPlot
20
21
22   class VisualizeTransformationWidget(VectorGridPlot):
23       """This class is the widget that is used in the main window to visualize transformations.
24
25       It handles all the rendering itself, and the only method that the user needs to
26       worry about is :meth:`visualize_matrix_transformation`, which allows you to visualize
27       the given matrix transformation.
28       """
29
30       def __init__(self, *args, display_settings: DisplaySettings, **kwargs):
31           """Create the widget and assign its display settings, passing ``*args`` and ``**kwargs`` to super."""
32           super().__init__(*args, **kwargs)
33
34           self.display_settings = display_settings
35
36       def visualize_matrix_transformation(self, matrix: MatrixType) -> None:
37           """Transform the grid by the given matrix.
38
39           .. warning:: This method does not call ``update()``. This must be done by the caller.
40
41           .. note::
42              This method transforms the background grid, not the basis vectors. This
43              means that it cannot be used to compose transformations. Compositions
44              should be done by the user.
45
46           :param MatrixType matrix: The matrix to transform by
47           """
48           self.point_i = (matrix[0][0], matrix[1][0])
49           self.point_j = (matrix[0][1], matrix[1][1])
50
51       def paintEvent(self, event: QPaintEvent) -> None:
52           """Handle a :class:`QPaintEvent` by drawing the background grid and the transformed grid.
53
54           The transformed grid is defined by the basis vectors i and j, which can
55           be controlled with the :meth:`visualize_matrix_transformation` method.
56           """
57           painter = QPainter()
58           painter.begin(self)
59
60           painter.setRenderHint(QPainter.Antialiasing)
61           painter.setBrush(Qt.NoBrush)
62
63           self.draw_background(painter, self.display_settings.draw_background_grid)
64
65           if self.display_settings.draw_eigenlines:
66               self.draw_eigenlines(painter)
67
68           if self.display_settings.draw_eigenvectors:
69               self.draw_eigenvectors(painter)
70
```

244

```
 71                 if self.display_settings.draw_determinant_parallelogram:
 72                     self.draw_determinant_parallelogram(painter)
 73
 74                     if self.display_settings.show_determinant_value:
 75                         self.draw_determinant_text(painter)
 76
 77                 if self.display_settings.draw_transformed_grid:
 78                     self.draw_transformed_grid(painter)
 79
 80                 if self.display_settings.draw_basis_vectors:
 81                     self.draw_basis_vectors(painter)
 82
 83             painter.end()
 84             event.accept()
 85
 86
 87     class DefineVisuallyWidget(VisualizeTransformationWidget):
 88         """This class is the widget that allows the user to visually define a matrix.
 89
 90         This is just the widget itself. If you want the dialog, use
 91         :class:`lintrans.gui.dialogs.define_new_matrix.DefineVisuallyDialog`.
 92         """
 93
 94         def __init__(self, *args, display_settings: DisplaySettings, **kwargs):
 95             """Create the widget and enable mouse tracking. ``*args`` and ``**kwargs`` are passed to ``super()``."""
 96             super().__init__(*args, display_settings=display_settings, **kwargs)
 97
 98             self.dragged_point: Tuple[float, float] | None = None
 99
100             # This is the distance that the cursor needs to be from the point to drag it
101             self.epsilon: int = 5
102
103         def mousePressEvent(self, event: QMouseEvent) -> None:
104             """Handle a :class:`QMouseEvent` when the user presses a button."""
105             mx = event.x()
106             my = event.y()
107             button = event.button()
108
109             if button != Qt.LeftButton:
110                 event.ignore()
111                 return
112
113             for point in (self.point_i, self.point_j):
114                 px, py = self.canvas_coords(*point)
115                 if abs(px - mx) <= self.epsilon and abs(py - my) <= self.epsilon:
116                     self.dragged_point = point[0], point[1]
117
118             event.accept()
119
120         def mouseReleaseEvent(self, event: QMouseEvent) -> None:
121             """Handle a :class:`QMouseEvent` when the user releases a button."""
122             if event.button() == Qt.LeftButton:
123                 self.dragged_point = None
124                 event.accept()
125             else:
126                 event.ignore()
127
128         def mouseMoveEvent(self, event: QMouseEvent) -> None:
129             """Handle the mouse moving on the canvas."""
130             mx = event.x()
131             my = event.y()
132
133             if self.dragged_point is None:
134                 event.ignore()
135                 return
136
137             x, y = self.grid_coords(mx, my)
138
139             possible_snaps: List[Tuple[int, int]] = [
140                 (floor(x), floor(y)),
141                 (floor(x), ceil(y)),
142                 (ceil(x), floor(y)),
143                 (ceil(x), ceil(y))
```

```
144                     ]
145
146             snap_distances: List[Tuple[float, Tuple[int, int]]] = [
147                 (dist((x, y), coord), coord)
148                 for coord in possible_snaps
149             ]
150
151             for snap_dist, coord in snap_distances:
152                 if snap_dist < 0.1:
153                     x, y = coord
154
155             if self.dragged_point == self.point_i:
156                 self.point_i = x, y
157
158             elif self.dragged_point == self.point_j:
159                 self.point_j = x, y
160
161             self.dragged_point = x, y
162
163             self.update()
164
165             event.accept()
```

B   Testing code

B.1   `matrices/test_parse_and_validate_expression.py`

```
1    # lintrans - The linear transformation visualizer
2    # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4    # This program is licensed under GNU GPLv3, available here:
5    # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7    """Test the :mod:`matrices.parse` module validation and parsing."""
8
9    from typing import List, Tuple
10
11   import pytest
12
13   from lintrans.matrices.parse import (
14       MatrixParseError, find_sub_expressions, parse_matrix_expression, validate_matrix_expression
15   )
16   from lintrans.typing_ import MatrixParseList
17
18   expected_sub_expressions: List[Tuple[str, List[str]]] = [
19       ('2(AB)^-1', ['AB']),
20       ('-3(A+B)^2-C(B^TA)^-1', ['A+B', 'B^TA']),
21       ('rot(45)', []),
22       ('()', []),
23       ('(())', ['()']),
24       ('2.3A^-1(AB)^-1+(BC)^2', ['AB', 'BC']),
25       ('(2.3A^-1(AB)^-1+(BC)^2)', ['2.3A^-1(AB)^-1+(BC)^2']),
26   ]
27
28
29   def test_find_sub_expressions() -> None:
30       """Test the :func:`lintrans.matrices.parse.find_sub_expressions` function."""
31       for inp, output in expected_sub_expressions:
32           assert find_sub_expressions(inp) == output
33
34
35   valid_inputs: List[str] = [
36       'A', 'AB', '3A', '1.2A', '-3.4A', 'A^2', 'A^-1', 'A^{-1}',
37       'A^12', 'A^T', 'A^{5}', 'A^{T}', '4.3A^7', '9.2A^{18}', '0.1A'
38
39       'rot(45)', 'rot(12.5)', '3rot(90)',
40       'rot(135)^3', 'rot(51)^T', 'rot(-34)^-1',
41
42       'A+B', 'A+2B', '4.3A+9B', 'A^2+B^T', '3A^7+0.8B^{16}',
43       'A-B', '3A-4B', '3.2A^3-16.79B^T', '4.752A^{17}-3.32B^{36}',
44       'A-1B', '-A', '-1A'
45
46       '3A4B', 'A^TB', 'A^{T}B', '4A^6B^3',
47       '2A^{3}4B^5', '4rot(90)^3', 'rot(45)rot(13)',
48       'Arot(90)', 'AB^2', 'A^2B^2', '8.36A^T3.4B^12',
49
50       '3.5A^{4}5.6rot(19.2)^T-B^{-1}4.1C^5',
51
52       '(A)', '(AB)^-1', '2.3(3B^TA)^2', '-3.4(9D^{2}3F^-1)^T+C', '(AB)(C)',
53       '3(rot(34)^-7A)^-1+B', '3A^2B+4A(B+C)^-1D^T-A(C(D+E)B)'
54   ]
55
56   invalid_inputs: List[str] = [
57       '', 'rot()', 'A^', 'A^1.2', 'A^{3.4}', '1,2A', 'ro(12)', '5', '12^2', '^T', '^{12}', '.1A',
58       'A^{13}', 'A^3}', 'A^A', '^2', 'A--B', '--A', '+A', '--1A', 'A--B', 'A--1B', '.A', '1.A',
59       '2.3AB)^T', '(AB+)', '-4.6(9A', '-2(3.4A^{-1}-C^)^2', '9.2)', '3A^2B+4A(B+C)^-1D^T-A(C(D+EB)',
60       '3()^2', '4(your mum)^T', 'rot()', 'rot(10.1.1)', 'rot(--2)',
61
62       'This is 100% a valid matrix expression, I swear'
63   ]
64
65
66   @pytest.mark.parametrize('inputs,output', [(valid_inputs, True), (invalid_inputs, False)])
67   def test_validate_matrix_expression(inputs: List[str], output: bool) -> None:
```

```
68              """Test the validate_matrix_expression() function."""
69              for inp in inputs:
70                  assert validate_matrix_expression(inp) == output
71
72
73      expressions_and_parsed_expressions: List[Tuple[str, MatrixParseList]] = [
74          # Simple expressions
75          ('A', [[('', 'A', '')]]),
76          ('A^2', [[('', 'A', '2')]]),
77          ('A^{2}', [[('', 'A', '2')]]),
78          ('3A', [[('3', 'A', '')]]),
79          ('1.4A^3', [[('1.4', 'A', '3')]]),
80          ('0.1A', [[('0.1', 'A', '')]]),
81          ('0.1A', [[('0.1', 'A', '')]]),
82          ('A^12', [[('', 'A', '12')]]),
83          ('A^234', [[('', 'A', '234')]]),
84
85          # Multiplications
86          ('A 0.1B', [[('', 'A', ''), ('0.1', 'B', '')]]),
87          ('A^2 3B', [[('', 'A', '23'), ('', 'B', '')]]),
88          ('4A^{3} 6B^2', [[('4', 'A', '3'), ('6', 'B', '2')]]),
89          ('4.2A^{T} 6.1B^-1', [[('4.2', 'A', 'T'), ('6.1', 'B', '-1')]]),
90          ('-1.2A^2 rot(45)^2', [[('-1.2', 'A', '2'), ('', 'rot(45)', '2')]]),
91          ('3.2A^T 4.5B^{5} 9.6rot(121.3)', [[('3.2', 'A', 'T'), ('4.5', 'B', '5'), ('9.6', 'rot(121.3)', '')]]),
92          ('-1.18A^{-2} 0.1B^{2} 9rot(-34.6)^-1', [[('-1.18', 'A', '-2'), ('0.1', 'B', '2'), ('9', 'rot(-34.6)', '-1')]]),
93
94          # Additions
95          ('A + B', [[('', 'A', '')], [('', 'B', '')]]),
96          ('A + B - C', [[('', 'A', '')], [('', 'B', '')], [('-1', 'C', '')]]),
97          ('A^2 + 0.5B', [[('', 'A', '2')], [('0.5', 'B', '')]]),
98          ('2A^3 + 8B^T - 3C^-1', [[('2', 'A', '3')], [('8', 'B', 'T')], [('-3', 'C', '-1')]]),
99          ('4.9A^2 - 3rot(134.2)^-1 + 7.6B^8', [[('4.9', 'A', '2')], [('-3', 'rot(134.2)', '-1')], [('7.6', 'B', '8')]]),
100
101         # Additions with multiplication
102         ('2.14A^{3} 4.5rot(14.5)^-1 + 8B^T - 3C^-1', [[('2.14', 'A', '3'), ('4.5', 'rot(14.5)', '-1')],
103                                                       [('8', 'B', 'T')], [('-3', 'C', '-1')]]),
104         ('2.14A^{3} 4.5rot(14.5)^-1 + 8.5B^T 5.97C^14 - 3.14D^{-1} 6.7E^T',
105          [[('2.14', 'A', '3'), ('4.5', 'rot(14.5)', '-1')], [('8.5', 'B', 'T'), ('5.97', 'C', '14')],
106           [('-3.14', 'D', '-1'), ('6.7', 'E', 'T')]]),
107
108         # Parenthesized expressions
109         ('(AB)^-1', [[('', 'AB', '-1')]]),
110         ('-3(A+B)^2-C(B^TA)^-1', [[('-3', 'A+B', '2')], [('-1', 'C', ''), ('', 'B^{T}A', '-1')]]),
111         ('2.3(3B^TA)^2', [[('2.3', '3B^{T}A', '2')]]),
112         ('-3.4(9D^{2}3F^-1)^T+C', [[('-3.4', '9D^{2}3F^{-1}', 'T')], [('', 'C', '')]]),
113         ('2.39(3.1A^{-1}2.3B(CD)^-1)^T + (AB^T)^-1', [[('2.39', '3.1A^{-1}2.3B(CD)^{-1}', 'T')], [('', 'AB^{T}',
    ↪    '-1')]])
114     ]
115
116
117     def test_parse_matrix_expression() -> None:
118         """Test the parse_matrix_expression() function."""
119         for expression, parsed_expression in expressions_and_parsed_expressions:
120             # Test it with and without whitespace
121             assert parse_matrix_expression(expression) == parsed_expression
122             assert parse_matrix_expression(expression.replace(' ', '')) == parsed_expression
123
124
125     def test_parse_error() -> None:
126         """Test that parse_matrix_expression() raises a MatrixParseError."""
127         for expression in invalid_inputs:
128             with pytest.raises(MatrixParseError):
129                 parse_matrix_expression(expression)
```

B.2 matrices/matrix_wrapper/conftest.py

```
1     # lintrans - The linear transformation visualizer
2     # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4     # This program is licensed under GNU GPLv3, available here:
```

```python
5    # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7    """A simple conftest.py containing some re-usable fixtures."""
8
9    import numpy as np
10   import pytest
11
12   from lintrans.matrices import MatrixWrapper
13
14
15   def get_test_wrapper() -> MatrixWrapper:
16       """Return a new MatrixWrapper object with some preset values."""
17       wrapper = MatrixWrapper()
18
19       root_two_over_two = np.sqrt(2) / 2
20
21       wrapper['A'] = np.array([[1, 2], [3, 4]])
22       wrapper['B'] = np.array([[6, 4], [12, 9]])
23       wrapper['C'] = np.array([[-1, -3], [4, -12]])
24       wrapper['D'] = np.array([[13.2, 9.4], [-3.4, -1.8]])
25       wrapper['E'] = np.array([
26           [root_two_over_two, -1 * root_two_over_two],
27           [root_two_over_two, root_two_over_two]
28       ])
29       wrapper['F'] = np.array([[-1, 0], [0, 1]])
30       wrapper['G'] = np.array([[np.pi, np.e], [1729, 743.631]])
31
32       return wrapper
33
34
35   @pytest.fixture
36   def test_wrapper() -> MatrixWrapper:
37       """Return a new MatrixWrapper object with some preset values."""
38       return get_test_wrapper()
39
40
41   @pytest.fixture
42   def new_wrapper() -> MatrixWrapper:
43       """Return a new MatrixWrapper with no initialized values."""
44       return MatrixWrapper()
```

B.3　matrices/matrix_wrapper/test_evaluate_expression.py

```python
1    # lintrans - The linear transformation visualizer
2    # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4    # This program is licensed under GNU GPLv3, available here:
5    # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7    """Test the MatrixWrapper evaluate_expression() method."""
8
9    import numpy as np
10   from numpy import linalg as la
11   import pytest
12   from pytest import approx
13
14   from lintrans.matrices import MatrixWrapper, create_rotation_matrix
15   from lintrans.typing_ import MatrixType
16
17   from conftest import get_test_wrapper
18
19
20   def test_simple_matrix_addition(test_wrapper: MatrixWrapper) -> None:
21       """Test simple addition and subtraction of two matrices."""
22
23       # NOTE: We assert that all of these values are not None just to stop mypy complaining
24       # These values will never actually be None because they're set in the wrapper() fixture
25       # There's probably a better way  do this, because this method is a bit of a bodge, but this works for now
26       assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
27               test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
```

```python
28                    test_wrapper['G'] is not None
29
30         assert (test_wrapper.evaluate_expression('A+B') == test_wrapper['A'] + test_wrapper['B']).all()
31         assert (test_wrapper.evaluate_expression('E+F') == test_wrapper['E'] + test_wrapper['F']).all()
32         assert (test_wrapper.evaluate_expression('G+D') == test_wrapper['G'] + test_wrapper['D']).all()
33         assert (test_wrapper.evaluate_expression('C+C') == test_wrapper['C'] + test_wrapper['C']).all()
34         assert (test_wrapper.evaluate_expression('D+A') == test_wrapper['D'] + test_wrapper['A']).all()
35         assert (test_wrapper.evaluate_expression('B+C') == test_wrapper['B'] + test_wrapper['C']).all()
36
37         assert test_wrapper == get_test_wrapper()
38
39
40     def test_simple_two_matrix_multiplication(test_wrapper: MatrixWrapper) -> None:
41         """Test simple multiplication of two matrices."""
42         assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
43                 test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
44                 test_wrapper['G'] is not None
45
46         assert (test_wrapper.evaluate_expression('AB') == test_wrapper['A'] @ test_wrapper['B']).all()
47         assert (test_wrapper.evaluate_expression('BA') == test_wrapper['B'] @ test_wrapper['A']).all()
48         assert (test_wrapper.evaluate_expression('AC') == test_wrapper['A'] @ test_wrapper['C']).all()
49         assert (test_wrapper.evaluate_expression('DA') == test_wrapper['D'] @ test_wrapper['A']).all()
50         assert (test_wrapper.evaluate_expression('ED') == test_wrapper['E'] @ test_wrapper['D']).all()
51         assert (test_wrapper.evaluate_expression('FD') == test_wrapper['F'] @ test_wrapper['D']).all()
52         assert (test_wrapper.evaluate_expression('GA') == test_wrapper['G'] @ test_wrapper['A']).all()
53         assert (test_wrapper.evaluate_expression('CF') == test_wrapper['C'] @ test_wrapper['F']).all()
54         assert (test_wrapper.evaluate_expression('AG') == test_wrapper['A'] @ test_wrapper['G']).all()
55
56         assert test_wrapper.evaluate_expression('A2B') == approx(test_wrapper['A'] @ (2 * test_wrapper['B']))
57         assert test_wrapper.evaluate_expression('2AB') == approx((2 * test_wrapper['A']) @ test_wrapper['B'])
58         assert test_wrapper.evaluate_expression('C3D') == approx(test_wrapper['C'] @ (3 * test_wrapper['D']))
59         assert test_wrapper.evaluate_expression('4.2E1.2A') == approx((4.2 * test_wrapper['E']) @ (1.2 *
       ↪  test_wrapper['A']))
60
61         assert test_wrapper == get_test_wrapper()
62
63
64     def test_identity_multiplication(test_wrapper: MatrixWrapper) -> None:
65         """Test that multiplying by the identity doesn't change the value of a matrix."""
66         assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
67                 test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
68                 test_wrapper['G'] is not None
69
70         assert (test_wrapper.evaluate_expression('I') == test_wrapper['I']).all()
71         assert (test_wrapper.evaluate_expression('AI') == test_wrapper['A']).all()
72         assert (test_wrapper.evaluate_expression('IA') == test_wrapper['A']).all()
73         assert (test_wrapper.evaluate_expression('GI') == test_wrapper['G']).all()
74         assert (test_wrapper.evaluate_expression('IG') == test_wrapper['G']).all()
75
76         assert (test_wrapper.evaluate_expression('EID') == test_wrapper['E'] @ test_wrapper['D']).all()
77         assert (test_wrapper.evaluate_expression('IED') == test_wrapper['E'] @ test_wrapper['D']).all()
78         assert (test_wrapper.evaluate_expression('EDI') == test_wrapper['E'] @ test_wrapper['D']).all()
79         assert (test_wrapper.evaluate_expression('IEIDI') == test_wrapper['E'] @ test_wrapper['D']).all()
80         assert (test_wrapper.evaluate_expression('EI^3D') == test_wrapper['E'] @ test_wrapper['D']).all()
81
82         assert test_wrapper == get_test_wrapper()
83
84
85     def test_simple_three_matrix_multiplication(test_wrapper: MatrixWrapper) -> None:
86         """Test simple multiplication of two matrices."""
87         assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
88                 test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
89                 test_wrapper['G'] is not None
90
91         assert (test_wrapper.evaluate_expression('ABC') == test_wrapper['A'] @ test_wrapper['B'] @
       ↪  test_wrapper['C']).all()
92         assert (test_wrapper.evaluate_expression('ACB') == test_wrapper['A'] @ test_wrapper['C'] @
       ↪  test_wrapper['B']).all()
93         assert (test_wrapper.evaluate_expression('BAC') == test_wrapper['B'] @ test_wrapper['A'] @
       ↪  test_wrapper['C']).all()
94         assert (test_wrapper.evaluate_expression('EFG') == test_wrapper['E'] @ test_wrapper['F'] @
       ↪  test_wrapper['G']).all()
```

250

```
 95        assert (test_wrapper.evaluate_expression('DAC') == test_wrapper['D'] @ test_wrapper['A'] @
           ↪  test_wrapper['C']).all()
 96        assert (test_wrapper.evaluate_expression('GAE') == test_wrapper['G'] @ test_wrapper['A'] @
           ↪  test_wrapper['E']).all()
 97        assert (test_wrapper.evaluate_expression('FAG') == test_wrapper['F'] @ test_wrapper['A'] @
           ↪  test_wrapper['G']).all()
 98        assert (test_wrapper.evaluate_expression('GAF') == test_wrapper['G'] @ test_wrapper['A'] @
           ↪  test_wrapper['F']).all()
 99
100        assert test_wrapper == get_test_wrapper()
101
102
103    def test_matrix_inverses(test_wrapper: MatrixWrapper) -> None:
104        """Test the inverses of single matrices."""
105        assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
106                test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
107                test_wrapper['G'] is not None
108
109        assert (test_wrapper.evaluate_expression('A^{-1}') == la.inv(test_wrapper['A'])).all()
110        assert (test_wrapper.evaluate_expression('B^{-1}') == la.inv(test_wrapper['B'])).all()
111        assert (test_wrapper.evaluate_expression('C^{-1}') == la.inv(test_wrapper['C'])).all()
112        assert (test_wrapper.evaluate_expression('D^{-1}') == la.inv(test_wrapper['D'])).all()
113        assert (test_wrapper.evaluate_expression('E^{-1}') == la.inv(test_wrapper['E'])).all()
114        assert (test_wrapper.evaluate_expression('F^{-1}') == la.inv(test_wrapper['F'])).all()
115        assert (test_wrapper.evaluate_expression('G^{-1}') == la.inv(test_wrapper['G'])).all()
116
117        assert (test_wrapper.evaluate_expression('A^-1') == la.inv(test_wrapper['A'])).all()
118        assert (test_wrapper.evaluate_expression('B^-1') == la.inv(test_wrapper['B'])).all()
119        assert (test_wrapper.evaluate_expression('C^-1') == la.inv(test_wrapper['C'])).all()
120        assert (test_wrapper.evaluate_expression('D^-1') == la.inv(test_wrapper['D'])).all()
121        assert (test_wrapper.evaluate_expression('E^-1') == la.inv(test_wrapper['E'])).all()
122        assert (test_wrapper.evaluate_expression('F^-1') == la.inv(test_wrapper['F'])).all()
123        assert (test_wrapper.evaluate_expression('G^-1') == la.inv(test_wrapper['G'])).all()
124
125        assert test_wrapper == get_test_wrapper()
126
127
128    def test_matrix_powers(test_wrapper: MatrixWrapper) -> None:
129        """Test that matrices can be raised to integer powers."""
130        assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
131                test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
132                test_wrapper['G'] is not None
133
134        assert (test_wrapper.evaluate_expression('A^2') == la.matrix_power(test_wrapper['A'], 2)).all()
135        assert (test_wrapper.evaluate_expression('B^4') == la.matrix_power(test_wrapper['B'], 4)).all()
136        assert (test_wrapper.evaluate_expression('C^{12}') == la.matrix_power(test_wrapper['C'], 12)).all()
137        assert (test_wrapper.evaluate_expression('D^12') == la.matrix_power(test_wrapper['D'], 12)).all()
138        assert (test_wrapper.evaluate_expression('E^8') == la.matrix_power(test_wrapper['E'], 8)).all()
139        assert (test_wrapper.evaluate_expression('F^{-6}') == la.matrix_power(test_wrapper['F'], -6)).all()
140        assert (test_wrapper.evaluate_expression('G^-2') == la.matrix_power(test_wrapper['G'], -2)).all()
141
142        assert test_wrapper == get_test_wrapper()
143
144
145    def test_matrix_transpose(test_wrapper: MatrixWrapper) -> None:
146        """Test matrix transpositions."""
147        assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
148                test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
149                test_wrapper['G'] is not None
150
151        assert (test_wrapper.evaluate_expression('A^{T}') == test_wrapper['A'].T).all()
152        assert (test_wrapper.evaluate_expression('B^{T}') == test_wrapper['B'].T).all()
153        assert (test_wrapper.evaluate_expression('C^{T}') == test_wrapper['C'].T).all()
154        assert (test_wrapper.evaluate_expression('D^{T}') == test_wrapper['D'].T).all()
155        assert (test_wrapper.evaluate_expression('E^{T}') == test_wrapper['E'].T).all()
156        assert (test_wrapper.evaluate_expression('F^{T}') == test_wrapper['F'].T).all()
157        assert (test_wrapper.evaluate_expression('G^{T}') == test_wrapper['G'].T).all()
158
159        assert (test_wrapper.evaluate_expression('A^T') == test_wrapper['A'].T).all()
160        assert (test_wrapper.evaluate_expression('B^T') == test_wrapper['B'].T).all()
161        assert (test_wrapper.evaluate_expression('C^T') == test_wrapper['C'].T).all()
162        assert (test_wrapper.evaluate_expression('D^T') == test_wrapper['D'].T).all()
163        assert (test_wrapper.evaluate_expression('E^T') == test_wrapper['E'].T).all()
```

```
164             assert (test_wrapper.evaluate_expression('F^T') == test_wrapper['F'].T).all()
165             assert (test_wrapper.evaluate_expression('G^T') == test_wrapper['G'].T).all()
166
167             assert test_wrapper == get_test_wrapper()
168
169
170     def test_rotation_matrices(test_wrapper: MatrixWrapper) -> None:
171         """Test that 'rot(angle)' can be used in an expression."""
172             assert (test_wrapper.evaluate_expression('rot(90)') == create_rotation_matrix(90)).all()
173             assert (test_wrapper.evaluate_expression('rot(180)') == create_rotation_matrix(180)).all()
174             assert (test_wrapper.evaluate_expression('rot(270)') == create_rotation_matrix(270)).all()
175             assert (test_wrapper.evaluate_expression('rot(360)') == create_rotation_matrix(360)).all()
176             assert (test_wrapper.evaluate_expression('rot(45)') == create_rotation_matrix(45)).all()
177             assert (test_wrapper.evaluate_expression('rot(30)') == create_rotation_matrix(30)).all()
178
179             assert (test_wrapper.evaluate_expression('rot(13.43)') == create_rotation_matrix(13.43)).all()
180             assert (test_wrapper.evaluate_expression('rot(49.4)') == create_rotation_matrix(49.4)).all()
181             assert (test_wrapper.evaluate_expression('rot(-123.456)') == create_rotation_matrix(-123.456)).all()
182             assert (test_wrapper.evaluate_expression('rot(963.245)') == create_rotation_matrix(963.245)).all()
183             assert (test_wrapper.evaluate_expression('rot(-235.24)') == create_rotation_matrix(-235.24)).all()
184
185             assert test_wrapper == get_test_wrapper()
186
187
188     def test_multiplication_and_addition(test_wrapper: MatrixWrapper) -> None:
189         """Test multiplication and addition of matrices together."""
190             assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
191                     test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
192                     test_wrapper['G'] is not None
193
194             assert (test_wrapper.evaluate_expression('AB+C') ==
195                     test_wrapper['A'] @ test_wrapper['B'] + test_wrapper['C']).all()
196             assert (test_wrapper.evaluate_expression('DE-D') ==
197                     test_wrapper['D'] @ test_wrapper['E'] - test_wrapper['D']).all()
198             assert (test_wrapper.evaluate_expression('FD+AB') ==
199                     test_wrapper['F'] @ test_wrapper['D'] + test_wrapper['A'] @ test_wrapper['B']).all()
200             assert (test_wrapper.evaluate_expression('BA-DE') ==
201                     test_wrapper['B'] @ test_wrapper['A'] - test_wrapper['D'] @ test_wrapper['E']).all()
202
203             assert (test_wrapper.evaluate_expression('2AB+3C') ==
204                     (2 * test_wrapper['A']) @ test_wrapper['B'] + (3 * test_wrapper['C'])).all()
205             assert (test_wrapper.evaluate_expression('4D7.9E-1.2A') ==
206                     (4 * test_wrapper['D']) @ (7.9 * test_wrapper['E']) - (1.2 * test_wrapper['A'])).all()
207
208             assert test_wrapper == get_test_wrapper()
209
210
211     def test_complicated_expressions(test_wrapper: MatrixWrapper) -> None:
212         """Test evaluation of complicated expressions."""
213             assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
214                     test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
215                     test_wrapper['G'] is not None
216
217             assert (test_wrapper.evaluate_expression('-3.2A^T 4B^{-1} 6C^{-1} + 8.1D^{2} 3.2E^4') ==
218                     (-3.2 * test_wrapper['A'].T) @ (4 * la.inv(test_wrapper['B'])) @ (6 * la.inv(test_wrapper['C']))
219                     + (8.1 * la.matrix_power(test_wrapper['D'], 2)) @ (3.2 * la.matrix_power(test_wrapper['E'], 4))).all()
220
221             assert (test_wrapper.evaluate_expression('53.6D^{2} 3B^T - 4.9F^{2} 2D + A^3 B^-1') ==
222                     (53.6 * la.matrix_power(test_wrapper['D'], 2)) @ (3 * test_wrapper['B'].T)
223                     - (4.9 * la.matrix_power(test_wrapper['F'], 2)) @ (2 * test_wrapper['D'])
224                     + la.matrix_power(test_wrapper['A'], 3) @ la.inv(test_wrapper['B'])).all()
225
226             assert test_wrapper == get_test_wrapper()
227
228
229     def test_parenthesized_expressions(test_wrapper: MatrixWrapper) -> None:
230         """Test evaluation of parenthesized expressions."""
231             assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
232                     test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
233                     test_wrapper['G'] is not None
234
235             assert (test_wrapper.evaluate_expression('(A^T)^2') == la.matrix_power(test_wrapper['A'].T, 2)).all()
236             assert (test_wrapper.evaluate_expression('(B^T)^3') == la.matrix_power(test_wrapper['B'].T, 3)).all()
```

252

```
237         assert (test_wrapper.evaluate_expression('(C^T)^4') == la.matrix_power(test_wrapper['C'].T, 4)).all()
238         assert (test_wrapper.evaluate_expression('(D^T)^5') == la.matrix_power(test_wrapper['D'].T, 5)).all()
239         assert (test_wrapper.evaluate_expression('(E^T)^6') == la.matrix_power(test_wrapper['E'].T, 6)).all()
240         assert (test_wrapper.evaluate_expression('(F^T)^7') == la.matrix_power(test_wrapper['F'].T, 7)).all()
241         assert (test_wrapper.evaluate_expression('(G^T)^8') == la.matrix_power(test_wrapper['G'].T, 8)).all()
242
243         assert (test_wrapper.evaluate_expression('(rot(45)^1)^T') == create_rotation_matrix(45).T).all()
244         assert (test_wrapper.evaluate_expression('(rot(45)^2)^T') == la.matrix_power(create_rotation_matrix(45),
            ↪    2).T).all()
245         assert (test_wrapper.evaluate_expression('(rot(45)^3)^T') == la.matrix_power(create_rotation_matrix(45),
            ↪    3).T).all()
246         assert (test_wrapper.evaluate_expression('(rot(45)^4)^T') == la.matrix_power(create_rotation_matrix(45),
            ↪    4).T).all()
247         assert (test_wrapper.evaluate_expression('(rot(45)^5)^T') == la.matrix_power(create_rotation_matrix(45),
            ↪    5).T).all()
248
249         assert (test_wrapper.evaluate_expression('D^3(A+6.2F-0.397G^TE)^-2+A') ==
250                 la.matrix_power(test_wrapper['D'], 3) @ la.matrix_power(
251                     test_wrapper['A'] + 6.2 * test_wrapper['F'] - 0.397 * test_wrapper['G'].T @ test_wrapper['E'],
252                     -2
253                 ) + test_wrapper['A']).all()
254
255         assert (test_wrapper.evaluate_expression('-1.2F^{3}4.9D^T(A^2(B+3E^TF)^-1)^2') ==
256                 -1.2 * la.matrix_power(test_wrapper['F'], 3) @ (4.9 * test_wrapper['D'].T) @
257                 la.matrix_power(
258                     la.matrix_power(test_wrapper['A'], 2) @ la.matrix_power(
259                         test_wrapper['B'] + 3 * test_wrapper['E'].T @ test_wrapper['F'],
260                         -1
261                     ),
262                     2
263                 )).all()
264
265
266     def test_value_errors(test_wrapper: MatrixWrapper) -> None:
267         """Test that evaluate_expression() raises a ValueError for any malformed input."""
268         invalid_expressions = ['', '+', '-', 'This is not a valid expression', '3+4',
269                                'A+2', 'A^', '^2', 'A^-', 'At', 'A^t', '3^2']
270
271         for expression in invalid_expressions:
272             with pytest.raises(ValueError):
273                 test_wrapper.evaluate_expression(expression)
274
275
276     def test_linalgerror() -> None:
277         """Test that certain expressions raise np.linalg.LinAlgError."""
278         matrix_a: MatrixType = np.array([
279             [0, 0],
280             [0, 0]
281         ])
282
283         matrix_b: MatrixType = np.array([
284             [1, 2],
285             [1, 2]
286         ])
287
288         wrapper = MatrixWrapper()
289         wrapper['A'] = matrix_a
290         wrapper['B'] = matrix_b
291
292         assert (wrapper.evaluate_expression('A') == matrix_a).all()
293         assert (wrapper.evaluate_expression('B') == matrix_b).all()
294
295         with pytest.raises(np.linalg.LinAlgError):
296             wrapper.evaluate_expression('A^-1')
297
298         with pytest.raises(np.linalg.LinAlgError):
299             wrapper.evaluate_expression('B^-1')
300
301         assert (wrapper['A'] == matrix_a).all()
302         assert (wrapper['B'] == matrix_b).all()
```

B.4   `matrices/matrix_wrapper/test_misc.py`

```python
# lintrans - The linear transformation visualizer
# Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)

# This program is licensed under GNU GPLv3, available here:
# <https://www.gnu.org/licenses/gpl-3.0.html>

"""Test the miscellaneous methods of the MatrixWrapper class."""

from lintrans.matrices import MatrixWrapper


def test_get_expression(test_wrapper: MatrixWrapper) -> None:
    """Test the get_expression method of the MatrixWrapper class."""
    test_wrapper['N'] = 'A^2'
    test_wrapper['O'] = '4B'
    test_wrapper['P'] = 'A+C'

    test_wrapper['Q'] = 'N^-1'
    test_wrapper['R'] = 'P-40'
    test_wrapper['S'] = 'NOP'

    assert test_wrapper.get_expression('A') is None
    assert test_wrapper.get_expression('B') is None
    assert test_wrapper.get_expression('C') is None
    assert test_wrapper.get_expression('D') is None
    assert test_wrapper.get_expression('E') is None
    assert test_wrapper.get_expression('F') is None
    assert test_wrapper.get_expression('G') is None

    assert test_wrapper.get_expression('N') == 'A^2'
    assert test_wrapper.get_expression('O') == '4B'
    assert test_wrapper.get_expression('P') == 'A+C'

    assert test_wrapper.get_expression('Q') == 'N^-1'
    assert test_wrapper.get_expression('R') == 'P-40'
    assert test_wrapper.get_expression('S') == 'NOP'
```

B.5   `matrices/matrix_wrapper/test_setitem_and_getitem.py`

```python
# lintrans - The linear transformation visualizer
# Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)

# This program is licensed under GNU GPLv3, available here:
# <https://www.gnu.org/licenses/gpl-3.0.html>

"""Test the MatrixWrapper __setitem__() and __getitem__() methods."""

from typing import Any, List

import numpy as np
import pytest
from numpy import linalg as la

from lintrans.matrices import MatrixWrapper
from lintrans.typing_ import MatrixType

valid_matrix_names = 'ABCDEFGHJKLMNOPQRSTUVWXYZ'
invalid_matrix_names = ['bad name', '123456', 'Th15 Is an 1nV@l1D n@m3', 'abc', 'a']

test_matrix: MatrixType = np.array([[1, 2], [4, 3]])


def test_basic_get_matrix(new_wrapper: MatrixWrapper) -> None:
    """Test MatrixWrapper().__getitem__()."""
    for name in valid_matrix_names:
        assert new_wrapper[name] is None
```

```python
29              assert (new_wrapper['I'] == np.array([[1, 0], [0, 1]])).all()
30
31
32      def test_get_name_error(new_wrapper: MatrixWrapper) -> None:
33          """Test that MatrixWrapper().__getitem__() raises a NameError if called with an invalid name."""
34          for name in invalid_matrix_names:
35              with pytest.raises(NameError):
36                  _ = new_wrapper[name]
37
38
39      def test_basic_set_matrix(new_wrapper: MatrixWrapper) -> None:
40          """Test MatrixWrapper().__setitem__()."""
41          for name in valid_matrix_names:
42              new_wrapper[name] = test_matrix
43              assert (new_wrapper[name] == test_matrix).all()
44
45              new_wrapper[name] = None
46              assert new_wrapper[name] is None
47
48
49      def test_set_expression(test_wrapper: MatrixWrapper) -> None:
50          """Test that MatrixWrapper.__setitem__() can accept a valid expression."""
51          test_wrapper['N'] = 'A^2'
52          test_wrapper['O'] = 'BA+2C'
53          test_wrapper['P'] = 'E^T'
54          test_wrapper['Q'] = 'C^-1B'
55          test_wrapper['R'] = 'A^{2}3B'
56          test_wrapper['S'] = 'N^-1'
57          test_wrapper['T'] = 'PQP^-1'
58
59          with pytest.raises(TypeError):
60              test_wrapper['U'] = 'A+1'
61
62          with pytest.raises(TypeError):
63              test_wrapper['V'] = 'K'
64
65          with pytest.raises(TypeError):
66              test_wrapper['W'] = 'L^2'
67
68          with pytest.raises(TypeError):
69              test_wrapper['X'] = 'M^-1'
70
71
72      def test_simple_dynamic_evaluation(test_wrapper: MatrixWrapper) -> None:
73          """Test that expression-defined matrices are evaluated dynamically."""
74          test_wrapper['N'] = 'A^2'
75          test_wrapper['O'] = '4B'
76          test_wrapper['P'] = 'A+C'
77
78          assert (test_wrapper['N'] == test_wrapper.evaluate_expression('A^2')).all()
79          assert (test_wrapper['O'] == test_wrapper.evaluate_expression('4B')).all()
80          assert (test_wrapper['P'] == test_wrapper.evaluate_expression('A+C')).all()
81
82          assert (test_wrapper.evaluate_expression('N^2 + 30') ==
83                  la.matrix_power(test_wrapper.evaluate_expression('A^2'), 2) +
84                  3 * test_wrapper.evaluate_expression('4B')
85                  ).all()
86          assert (test_wrapper.evaluate_expression('P^-1 - 3NO^2') ==
87                  la.inv(test_wrapper.evaluate_expression('A+C')) -
88                  (3 * test_wrapper.evaluate_expression('A^2')) @
89                  la.matrix_power(test_wrapper.evaluate_expression('4B'), 2)
90                  ).all()
91
92          test_wrapper['A'] = np.array([
93              [19, -21.5],
94              [84, 96.572]
95          ])
96          test_wrapper['B'] = np.array([
97              [-0.993, 2.52],
98              [1e10, 0]
99          ])
100         test_wrapper['C'] = np.array([
101             [0, 19512],
```

```
102                 [1.414, 19]
103         ])
104
105         assert (test_wrapper['N'] == test_wrapper.evaluate_expression('A^2')).all()
106         assert (test_wrapper['O'] == test_wrapper.evaluate_expression('4B')).all()
107         assert (test_wrapper['P'] == test_wrapper.evaluate_expression('A+C')).all()
108
109         assert (test_wrapper.evaluate_expression('N^2 + 30') ==
110                 la.matrix_power(test_wrapper.evaluate_expression('A^2'), 2) +
111                 3 * test_wrapper.evaluate_expression('4B')
112                 ).all()
113         assert (test_wrapper.evaluate_expression('P^-1 - 3N0^2') ==
114                 la.inv(test_wrapper.evaluate_expression('A+C')) -
115                 (3 * test_wrapper.evaluate_expression('A^2')) @
116                 la.matrix_power(test_wrapper.evaluate_expression('4B'), 2)
117                 ).all()
118
119
120     def test_recursive_dynamic_evaluation(test_wrapper: MatrixWrapper) -> None:
121         """Test that dynamic evaluation works recursively."""
122         test_wrapper['N'] = 'A^2'
123         test_wrapper['O'] = '4B'
124         test_wrapper['P'] = 'A+C'
125
126         test_wrapper['Q'] = 'N^-1'
127         test_wrapper['R'] = 'P-40'
128         test_wrapper['S'] = 'NOP'
129
130         assert test_wrapper['Q'] == pytest.approx(test_wrapper.evaluate_expression('A^-2'))
131         assert test_wrapper['R'] == pytest.approx(test_wrapper.evaluate_expression('A + C - 16B'))
132         assert test_wrapper['S'] == pytest.approx(test_wrapper.evaluate_expression('A^{2}4BA + A^{2}4BC'))
133
134
135     def test_set_identity_error(new_wrapper: MatrixWrapper) -> None:
136         """Test that MatrixWrapper().__setitem__() raises a NameError when trying to assign to the identity matrix."""
137         with pytest.raises(NameError):
138             new_wrapper['I'] = test_matrix
139
140
141     def test_set_name_error(new_wrapper: MatrixWrapper) -> None:
142         """Test that MatrixWrapper().__setitem__() raises a NameError when trying to assign to an invalid name."""
143         for name in invalid_matrix_names:
144             with pytest.raises(NameError):
145                 new_wrapper[name] = test_matrix
146
147
148     def test_set_type_error(new_wrapper: MatrixWrapper) -> None:
149         """Test that MatrixWrapper().__setitem__() raises a TypeError when trying to set a non-matrix."""
150         invalid_values: List[Any] = [
151                             12,
152                             [1, 2, 3, 4, 5],
153                             [[1, 2], [3, 4]],
154                             True,
155                             24.3222,
156                             'This is totally a matrix, I swear',
157                             MatrixWrapper,
158                             MatrixWrapper(),
159                             np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),
160                             np.eye(100)
161                             ]
162
163         for value in invalid_values:
164             with pytest.raises(TypeError):
165                 new_wrapper['M'] = value
```

B.6 matrices/utility/test_coord_conversion.py

```
1     #  lintrans - The linear transformation visualizer
2     #  Copyright (C) 2022 D. Dyson (DoctorDalek1963)
3     #
```

```
4     #  This program is licensed under GNU GPLv3, available here:
5     #  <https://www.gnu.org/licenses/gpl-3.0.html>
6
7     """Test conversion between polar and rectilinear coordinates in :mod:`lintrans.matrices.utility`."""
8
9     from typing import List, Tuple
10
11    from numpy import pi, sqrt
12    from pytest import approx
13
14    from lintrans.matrices.utility import polar_coords, rect_coords
15
16    expected_coords: List[Tuple[Tuple[float, float], Tuple[float, float]]] = [
17        ((0, 0), (0, 0)),
18        ((1, 1), (sqrt(2), pi / 4)),
19        ((0, 1), (1, pi / 2)),
20        ((1, 0), (1, 0)),
21        ((sqrt(2), sqrt(2)), (2, pi / 4)),
22        ((-3, 4), (5, 2.214297436)),
23        ((4, -3), (5, 5.639684198)),
24        ((5, -0.2), (sqrt(626) / 5, 6.24320662)),
25        ((-1.3, -10), (10.08414597, 4.583113976)),
26        ((23.4, 0), (23.4, 0)),
27        ((pi, -pi), (4.442882938, 1.75 * pi))
28    ]
29
30
31    def test_polar_coords() -> None:
32        """Test that :func:`lintrans.matrices.utility.polar_coords` works as expected."""
33        for rect, polar in expected_coords:
34            assert polar_coords(*rect) == approx(polar)
35
36
37    def test_rect_coords() -> None:
38        """Test that :func:`lintrans.matrices.utility.rect_coords` works as expected."""
39        for rect, polar in expected_coords:
40            assert rect_coords(*polar) == approx(rect)
41
42        assert rect_coords(1, 0) == approx((1, 0))
43        assert rect_coords(1, pi) == approx((-1, 0))
44        assert rect_coords(1, 2 * pi) == approx((1, 0))
45        assert rect_coords(1, 3 * pi) == approx((-1, 0))
46        assert rect_coords(1, 4 * pi) == approx((1, 0))
47        assert rect_coords(1, 5 * pi) == approx((-1, 0))
48        assert rect_coords(1, 6 * pi) == approx((1, 0))
49        assert rect_coords(20, 100) == approx(rect_coords(20, 100 % (2 * pi)))
```

B.7 `matrices/utility/test_float_utility_functions.py`

```
1     # lintrans - The linear transformation visualizer
2     # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4     # This program is licensed under GNU GPLv3, available here:
5     # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7     """Test the utility functions for GUI dialog boxes."""
8
9     from typing import List, Tuple
10
11    import numpy as np
12    import pytest
13
14    from lintrans.matrices.utility import is_valid_float, round_float
15
16    valid_floats: List[str] = [
17        '0', '1', '3', '-2', '123', '-208', '1.2', '-3.5', '4.252634', '-42362.352325',
18        '1e4', '-2.59e3', '4.13e-6', '-5.5244e-12'
19    ]
20
21    invalid_floats: List[str] = [
```

```
22        '', 'pi', 'e', '1.2.3', '1,2', '-', '.', 'None', 'no', 'yes', 'float'
23    ]
24
25
26    @pytest.mark.parametrize('inputs,output', [(valid_floats, True), (invalid_floats, False)])
27    def test_is_valid_float(inputs: List[str], output: bool) -> None:
28        """Test the is_valid_float() function."""
29        for inp in inputs:
30            assert is_valid_float(inp) == output
31
32
33    def test_round_float() -> None:
34        """Test the round_float() function."""
35        expected_values: List[Tuple[float, int, str]] = [
36            (1.0, 4, '1'), (1e-6, 4, '0'), (1e-5, 6, '1e-5'), (6.3e-8, 5, '0'), (3.2e-8, 10, '3.2e-8'),
37            (np.sqrt(2) / 2, 5, '0.70711'), (-1 * np.sqrt(2) / 2, 5, '-0.70711'),
38            (np.pi, 1, '3.1'), (np.pi, 2, '3.14'), (np.pi, 3, '3.142'), (np.pi, 4, '3.1416'), (np.pi, 5, '3.14159'),
39            (1.23456789, 2, '1.23'), (1.23456789, 3, '1.235'), (1.23456789, 4, '1.2346'), (1.23456789, 5, '1.23457'),
40            (12345.678, 1, '12345.7'), (12345.678, 2, '12345.68'), (12345.678, 3, '12345.678'),
41        ]
42
43        for num, precision, answer in expected_values:
44            assert round_float(num, precision) == answer
```

B.8   matrices/utility/test_rotation_matrices.py

```
1     # lintrans - The linear transformation visualizer
2     # Copyright (C) 2021-2022 D. Dyson (DoctorDalek1963)
3
4     # This program is licensed under GNU GPLv3, available here:
5     # <https://www.gnu.org/licenses/gpl-3.0.html>
6
7     """Test functions for rotation matrices."""
8
9     from typing import List, Tuple
10
11    import numpy as np
12    import pytest
13
14    from lintrans.matrices import create_rotation_matrix
15    from lintrans.typing_ import MatrixType
16
17    angles_and_matrices: List[Tuple[float, float, MatrixType]] = [
18        (0, 0, np.array([[1, 0], [0, 1]])),
19        (90, np.pi / 2, np.array([[0, -1], [1, 0]])),
20        (180, np.pi, np.array([[-1, 0], [0, -1]])),
21        (270, 3 * np.pi / 2, np.array([[0, 1], [-1, 0]])),
22        (360, 2 * np.pi, np.array([[1, 0], [0, 1]])),
23
24        (45, np.pi / 4, np.array([
25            [np.sqrt(2) / 2, -1 * np.sqrt(2) / 2],
26            [np.sqrt(2) / 2, np.sqrt(2) / 2]
27        ])),
28        (135, 3 * np.pi / 4, np.array([
29            [-1 * np.sqrt(2) / 2, -1 * np.sqrt(2) / 2],
30            [np.sqrt(2) / 2, -1 * np.sqrt(2) / 2]
31        ])),
32        (225, 5 * np.pi / 4, np.array([
33            [-1 * np.sqrt(2) / 2, np.sqrt(2) / 2],
34            [-1 * np.sqrt(2) / 2, -1 * np.sqrt(2) / 2]
35        ])),
36        (315, 7 * np.pi / 4, np.array([
37            [np.sqrt(2) / 2, np.sqrt(2) / 2],
38            [-1 * np.sqrt(2) / 2, np.sqrt(2) / 2]
39        ])),
40
41        (30, np.pi / 6, np.array([
42            [np.sqrt(3) / 2, -1 / 2],
43            [1 / 2, np.sqrt(3) / 2]
44        ])),
```

258

```
45          (60, np.pi / 3, np.array([
46              [1 / 2, -1 * np.sqrt(3) / 2],
47              [np.sqrt(3) / 2, 1 / 2]
48          ])),
49          (120, 2 * np.pi / 3, np.array([
50              [-1 / 2, -1 * np.sqrt(3) / 2],
51              [np.sqrt(3) / 2, -1 / 2]
52          ])),
53          (150, 5 * np.pi / 6, np.array([
54              [-1 * np.sqrt(3) / 2, -1 / 2],
55              [1 / 2, -1 * np.sqrt(3) / 2]
56          ])),
57          (210, 7 * np.pi / 6, np.array([
58              [-1 * np.sqrt(3) / 2, 1 / 2],
59              [-1 / 2, -1 * np.sqrt(3) / 2]
60          ])),
61          (240, 4 * np.pi / 3, np.array([
62              [-1 / 2, np.sqrt(3) / 2],
63              [-1 * np.sqrt(3) / 2, -1 / 2]
64          ])),
65          (300, 10 * np.pi / 6, np.array([
66              [1 / 2, np.sqrt(3) / 2],
67              [-1 * np.sqrt(3) / 2, 1 / 2]
68          ])),
69          (330, 11 * np.pi / 6, np.array([
70              [np.sqrt(3) / 2, 1 / 2],
71              [-1 / 2, np.sqrt(3) / 2]
72          ]))
73      ]
74
75
76      def test_create_rotation_matrix() -> None:
77          """Test that create_rotation_matrix() works with given angles and expected matrices."""
78          for degrees, radians, matrix in angles_and_matrices:
79              assert create_rotation_matrix(degrees, degrees=True) == pytest.approx(matrix)
80              assert create_rotation_matrix(radians, degrees=False) == pytest.approx(matrix)
81
82              assert create_rotation_matrix(-1 * degrees, degrees=True) == pytest.approx(np.linalg.inv(matrix))
83              assert create_rotation_matrix(-1 * radians, degrees=False) == pytest.approx(np.linalg.inv(matrix))
84
85          assert (create_rotation_matrix(-90, degrees=True) ==
86                  create_rotation_matrix(270, degrees=True)).all()
87          assert (create_rotation_matrix(-0.5 * np.pi, degrees=False) ==
88                  create_rotation_matrix(1.5 * np.pi, degrees=False)).all()
```