

# lintrans

by D. Dyson

Centre Name: The Duston School  
Centre Number: 123456  
Candidate Number: 123456

# Contents

<b>1</b>	<b>Analysis</b>	<b>1</b>
1.1	Computational Approach . . . . .	1
1.2	Stakeholders . . . . .	2
1.3	Research on existing solutions . . . . .	2
1.3.1	MIT ‘Matrix Vector’ Mathlet . . . . .	2
1.3.2	Linear Transformation Visualizer . . . . .	3
1.3.3	Desmos app . . . . .	3
1.3.4	Visualizing Linear Transformations . . . . .	4
1.4	Essential features . . . . .	4
1.5	Limitations . . . . .	5
1.6	Hardware and software requirements . . . . .	6
1.6.1	Hardware . . . . .	6
1.6.2	Software . . . . .	6
1.7	Success criteria . . . . .	7
<b>2</b>	<b>Design</b>	<b>8</b>
2.1	Problem decomposition . . . . .	8
2.2	Structure of the solution . . . . .	8
2.2.1	The main project . . . . .	8
2.2.2	The gui subpackages . . . . .	10
2.3	Algorithm design . . . . .	10
2.4	Usability features . . . . .	10
2.5	Variables and validation . . . . .	11
2.6	Iterative test data . . . . .	12
2.7	Post-development test data . . . . .	13
<b>3</b>	<b>Development</b>	<b>14</b>
3.1	Matrices backend . . . . .	14
3.1.1	MatrixWrapper class . . . . .	14
	<b>References</b>	<b>17</b>
<b>A</b>	<b>Project code</b>	<b>18</b>
A.1	__main__.py . . . . .	18
A.2	__init__.py . . . . .	18
A.3	matrices/wrapper.py . . . . .	18
A.4	matrices/__init__.py . . . . .	22
A.5	matrices/parse.py . . . . .	22
A.6	typing/__init__.py . . . . .	24
A.7	gui/main_window.py . . . . .	25
A.8	gui/settings.py . . . . .	32
A.9	gui/__init__.py . . . . .	33
A.10	gui/validate.py . . . . .	33
A.11	gui/dialogs/settings.py . . . . .	34
A.12	gui/dialogs/define_new_matrix.py . . . . .	37
A.13	gui/dialogs/__init__.py . . . . .	42
A.14	gui/plots/widgets.py . . . . .	42
A.15	gui/plots/classes.py . . . . .	45
A.16	gui/plots/__init__.py . . . . .	53
<b>B</b>	<b>Testing code</b>	<b>53</b>
B.1	matrices/test_rotation_matrices.py . . . . .	53
B.2	matrices/test_parse_and_validate_expression.py . . . . .	54
B.3	matrices/matrix_wrapper/test_misc.py . . . . .	55
B.4	matrices/matrix_wrapper/conftest.py . . . . .	56
B.5	matrices/matrix_wrapper/test_evaluate_expression.py . . . . .	57
B.6	matrices/matrix_wrapper/test_setitem_and_getitem.py . . . . .	60

B.7	<code>gui/test_dialog_utility_functions.py</code> . . . . .	63
-----	---	----

# 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[1], 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  $2 \times 2$  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 be 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. Many 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.

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, 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'[2] 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 draggable 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.

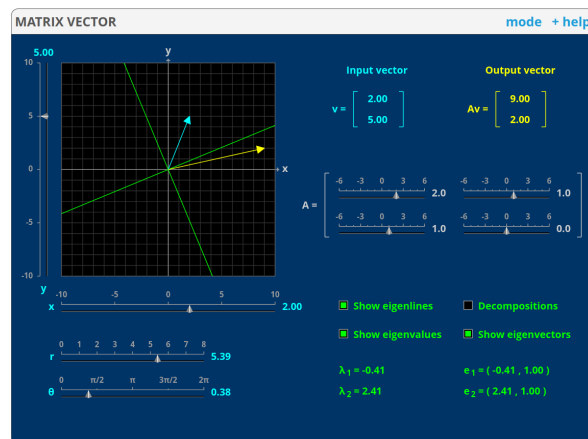


Figure 1: The MIT 'Matrix Vector' Mathlet

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  $2 \times 2$  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

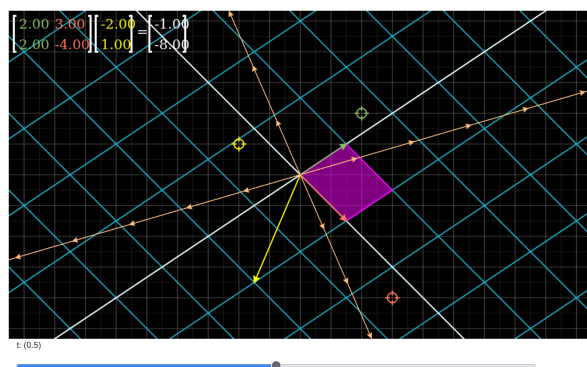


Figure 2: 'Linear Transformation Visualizer' halfway through an animation

Another web app that I found was one simply called 'Linear Transformation Visualizer' by Shad Sharma[3]. 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 determinant. 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 determinants 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[4], 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.

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.

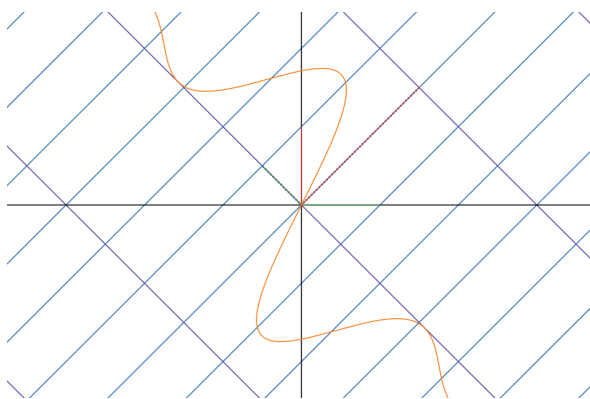


Figure 3: The Desmos app halfway through an animation, rendering  $f(x) = \frac{\sin^2 x}{x}$  in orange

### 1.3.4 Visualizing Linear Transformations

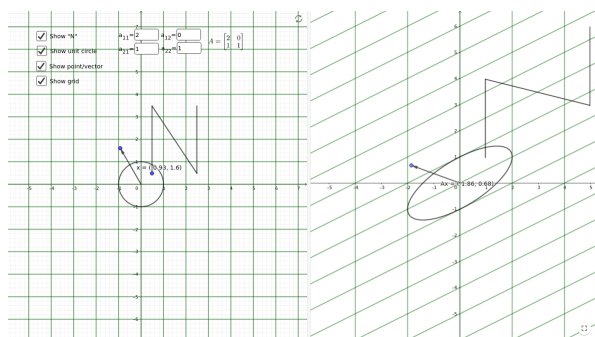


Figure 4: The GeoGebra applet rendering its default matrix

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

The last solution that I want to talk about is a GeoGebra applet simply titled 'Visualizing Linear Transformations'[5]. 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 aforementioned N (which the user can translate to anywhere on the grid), the unit circle, the input/output 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

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  $\mathbf{C}$  is the matrix for the currently displayed transformation, and  $\mathbf{T}$  is the matrix for the target transformation, then we could either animate from  $\mathbf{C}$  to  $\mathbf{T}$  or we could animate from  $\mathbf{C}$  to  $\mathbf{TC}$ . 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. 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  $\mathbf{I}$  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  $\mathbb{R}^2 \rightarrow \mathbb{R}^2$  - 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, \$BINARY\_SIZE 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 \$BINARY\_SIZE 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  $1920 \times 1080$  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'<sup>1</sup> or later, or any modern Linux distro<sup>2</sup>. Basically, it just requires an operating system that is compatible with Python 3.10 and 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.

Python 3.10 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 themselves, then they will need Python 3.10 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. `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; and `pyqt5` is a library that just allows interop between Python and Qt5, which is originally a C++ library.

---

<sup>1</sup>Python 3.10 won't compile on any earlier versions of macOS[6]

<sup>2</sup>Specifying a Linux version is practically impossible. Python 3.10 isn't available in many package repositories, but 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[7]

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<sup>3</sup>; `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 render any valid matrix expression
4. It must be able to animate any valid matrix expression
5. It must be able to display information about the currently rendered transformation (determinant, eigenlines, etc.)
6. It must be able to save and load sessions (defined matrices, display settings, etc.)
7. 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 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.

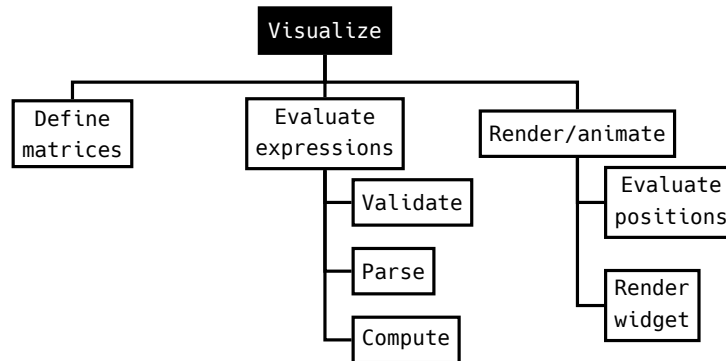
---

<sup>3</sup>Python has weak, dynamic typing with optional type annotations but `mypy` enforces these static type annotations

## 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.

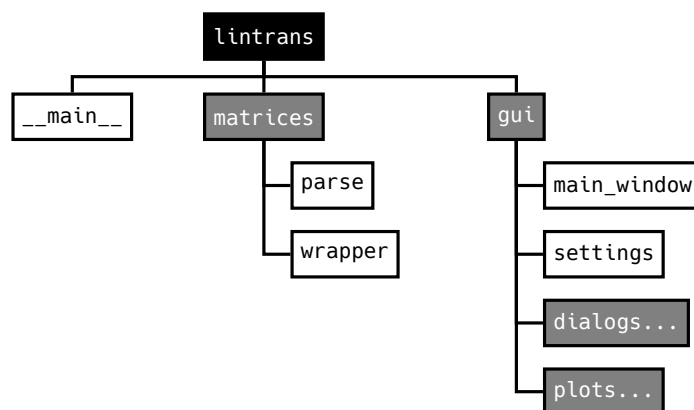
Rendering and animating will likely be the largest part in reality, but I've only decomposed it into simple blocks here. Evaluating positions involves evaluating the matrix expression that the user has input and using the columns of the resultant matrix to find the new positions of the basis vectors, 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

#### 2.2.1 The main project

I have decomposed my solution like so:



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.

`matrices` is the package that will allow the user to define, validate, parse, evaluate, and use matrices. The `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 elsewhere. The `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 matrices are actually defined - as well the ability to evaluate matrix expressions, in addition to its basic behaviour of setting and getting matrices. This `matrices` package will also have a `create_rotation_matrix` function that will generate a rotation matrix from an angle using the formula  $\begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$ . It will be in the `wrapper` module since it's related to defining and manipulating matrices, but it will be exported and accessible as `lintrans.matrices.create_rotation_matrix`.

`gui` is the package that will contain all the frontend code for everything GUI-related. `main_window` is the module that will contain a `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. This module will also contain a simple `main()` function to instantiate and launch the application GUI.

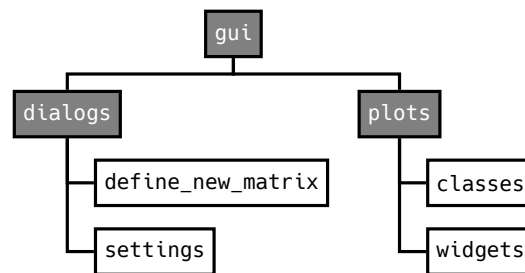
The `settings` module will contain a `DisplaySettings` dataclass<sup>4</sup> that will represent the settings for visualizing transformations. The `LintransMainWindow` class will have an instance of this class and check against it when rendering things. The user will be able to open a dialog to change these display settings, which will update the main window's instance of this class.

The `settings` module will also have a `GlobalSettings` class, which will represent the global settings for the application, such as the logging level, where to save the logs, whether to ask the user if they want to be prompted with a tutorial whenever they open the app, etc. This class will have defaults for everything, but the constructor will try to read these settings from a config file if possible. This allows for persistent settings between sessions. This config file will be `~/.config/lintrans.conf` on Unix-like systems, including macOS, and `C:\Users\%USER%\AppData\Roaming\lintrans\config.txt` on Windows. This difference is to remain consistent with operating system conventions<sup>5</sup>.

<sup>4</sup>This is the Python equivalent of a `struct` or `record` in other languages

<sup>5</sup>And also to avoid confusing Windows users with a `.conf` file

### 2.2.2 The gui subpackages



The `dialogs` subpackage will contain modules with different dialog classes. It will have a `define_new_matrices` module, which will have a `DefinedDialog` abstract superclass. It will also contain classes that inherit from this superclass and provide dialogs for defining new matrices visually, numerically, and as an expression in terms of other matrices. Additionally, this subpackage will contain a `settings` module, which will provide a `SettingsDialog` superclass and a `DisplaySettingsDialog` class, which will allow the user to configure the aforementioned display settings. It will also have a `GlobalSettingsDialog` class, which will similarly allow the user to configure the app's global settings through a dialog.

The `plots` subpackage will have a `classes` module and a `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 `plots` subpackage will also contain a `widgets` module, which will have the classes `VisualizeTransformationWidget` and `DefineVisuallyWidget`, both of which will 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.

It's also worth noting here that I don't currently know how I'm going to implement the transformation of arbitrary polygons. It will likely consist of an attribute in `VisualizeTransformationWidget` which is a list of points, and these points can be dragged around with mouse event handlers and then the transformed versions can be rendered, but I'm not yet sure about how I'm going to implement it.

## 2.3 Algorithm design

This section will be completed later.

## 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[8], 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[9] to make sure that my app is accessible to people with different forms of colour blindness<sup>6</sup>.

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.

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

This project won't actually have many variables. The main ones will be instance attributes on the `LintransMainWindow` class. It will have a `MatrixWrapper` instance, a `DisplaySettings` instance, and a `GlobalSettings` 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<sup>7</sup> and the matrices will be of type `MatrixType`. This will be a custom type alias representing a  $2 \times 2$  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<sup>8</sup>. 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

---

<sup>6</sup>I actually had to clone a fork of this project[10] to get it working on Ubuntu 20.04 and adapt it slightly to create a working jar file

<sup>7</sup>I would make these char but Python only has a str type for strings

<sup>8</sup>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()`

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 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

In unit testing, 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, like `is_valid_float`.

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^45.6rot(19.2^T-B^-14.1C^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 parse error, 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

This section will be completed later.



### 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<sup>9</sup>. The second comment is the file name and line numbers. If the line numbers are omitted, then the snippet is the whole file. 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[12] to allow indexing into a `MatrixWrapper` object like `wrapper['M']`. This simplifies using the class.

```

1  # 29ec1fedbf307e3b7ca731c4a381535fec899b0b
2  # src/lintrans/matrices/wrapper.py
3
4  """A module containing a simple MatrixWrapper class to wrap matrices and context."""
5
6  import numpy as np
7
8  from lintrans.typing import MatrixType
9
10
11 class MatrixWrapper:
12     """A simple wrapper class to hold all possible matrices and allow access to them."""
13
14     def __init__(self):
15         """Initialise a MatrixWrapper object with a matrices dict."""
16         self._matrices: dict[str, MatrixType | None] = {
17             'A': None, 'B': None, 'C': None, 'D': None,
18             'E': None, 'F': None, 'G': None, 'H': None,
19             'I': np.eye(2), # I is always defined as the identity matrix
20             'J': None, 'K': None, 'L': None, 'M': None,
21             'N': None, 'O': None, 'P': None, 'Q': None,
22             'R': None, 'S': None, 'T': None, 'U': None,
23             'V': None, 'W': None, 'X': None, 'Y': None,
24             'Z': None
25         }
26
27     def __getitem__(self, name: str) -> MatrixType | None:
28         """Get the matrix with 'name' from the dictionary.
29
30         Raises:
31             KeyError:
32                 If there is no matrix with the given name
33         """
34         return self._matrices[name]
35
36     def __setitem__(self, name: str, new_matrix: MatrixType) -> None:
37         """Set the value of matrix 'name' with the new_matrix.
38
39         Raises:
40             ValueError:
41                 If 'name' isn't a valid matrix name
42         """
43         name = name.upper()
44
45         if name == 'I' or name not in self._matrices:
46             raise NameError('Matrix name must be a capital letter and cannot be "I"')
47
48         self._matrices[name] = new_matrix

```

<sup>9</sup>A history of all commits can be found in the GitHub repository[11]

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`.

```

1  # 29ec1fedbf307e3b7ca731c4a381535fec899b0b
2  # tests/test_matrix_wrapper.py
3
4  """Test the MatrixWrapper class."""
5
6  import numpy as np
7  import pytest
8  from lintrans.matrices import MatrixWrapper
9
10 valid_matrix_names = 'ABCDEFGHJKLMNOPQRSTUVWXYZ'
11 test_matrix = np.array([[1, 2], [4, 3]])
12
13
14 @pytest.fixture
15 def wrapper() -> MatrixWrapper:
16     """Return a new MatrixWrapper object."""
17     return MatrixWrapper()
18
19
20 def test_get_matrix(wrapper) -> None:
21     """Test MatrixWrapper.__getitem__()."""
22     for name in valid_matrix_names:
23         assert wrapper[name] is None
24
25     assert (wrapper['I'] == np.array([[1, 0], [0, 1]])).all()
26
27
28 def test_get_name_error(wrapper) -> None:
29     """Test that MatrixWrapper.__getitem__() raises a KeyError if called with an invalid name."""
30     with pytest.raises(KeyError):
31         _ = wrapper['bad name']
32         _ = wrapper['123456']
33         _ = wrapper['Th15 Is an 1nV@l1D n@m3']
34         _ = wrapper['abc']
35
36
37 def test_set_matrix(wrapper) -> None:
38     """Test MatrixWrapper.__setitem__()."""
39     for name in valid_matrix_names:
40         wrapper[name] = test_matrix
41         assert (wrapper[name] == test_matrix).all()
42
43
44 def test_set_identity_error(wrapper) -> None:
45     """Test that MatrixWrapper.__setitem__() raises a NameError when trying to assign to I."""
46     with pytest.raises(NameError):
47         wrapper['I'] = test_matrix
48
49
50 def test_set_name_error(wrapper) -> None:
51     """Test that MatrixWrapper.__setitem__() raises a NameError when trying to assign to an invalid name."""
52     with pytest.raises(NameError):
53         wrapper['bad name'] = test_matrix
54         wrapper['123456'] = test_matrix
55         wrapper['Th15 Is an 1nV@l1D n@m3'] = test_matrix
56         wrapper['abc'] = test_matrix

```

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.

## References

- [1] Grant Sanderson (3blue1brown). *Essence of Linear Algebra*. 6th Aug. 2016. URL: [https://www.youtube.com/playlist?list=PLZHQ0b0WTQDPD3MizzM2xVFitgF8hE\\_ab](https://www.youtube.com/playlist?list=PLZHQ0b0WTQDPD3MizzM2xVFitgF8hE_ab).
- [2] H. Hohn et al. *Matrix Vector*. MIT. 2001. URL: <https://mathlets.org/mathlets/matrix-vector/>.
- [3] Shad Sharma. *Linear Transformation Visualizer*. 4th May 2017. URL: <https://shad.io/MatVis/>.
- [4] *2D linear transformation*. URL: <https://www.desmos.com/calculator/upooihuy4s>.
- [5] je1324. *Visualizing Linear Transformations*. 15th Mar. 2018. URL: <https://www.geogebra.org/m/YCZa8TAH>.
- [6] *Python 3.10 Downloads*. Python Software Foundation. URL: <https://www.python.org/downloads/release/python-3100/>.
- [7] *Qt5 for Linux/X11*. URL: <https://doc.qt.io/qt-5/linux.html>.
- [8] *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>.
- [9] Nathaniel Vaughn Kelso and Bernie Jenny. *Color Oracle*. Version 1.3. URL: <https://colororacle.org/>.
- [10] Alanocallaghan. *color-oracle-java*. Version 1.3. URL: <https://github.com/Alanocallaghan/color-oracle-java>.
- [11] D. Dyson (DoctorDalek1963). *lintrans*. GitHub. URL: <https://github.com/DoctorDalek1963/lintrans>.
- [12] *Python 3 Data model - special methods*. Python Software Foundation. URL: <https://docs.python.org/3/reference/datamodel.html#special-method-names>.

## A Project code

### A.1 \_\_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 very simply runs the app by calling :func:`lintrans.gui.main_window.main`.
10 This allows the user to run the app like ``python -m lintrans`` from the command line.
11 """
12
13
14 import sys
15
16 from lintrans.gui import main_window
17
18 if __name__ == '__main__':
19     main_window.main(sys.argv)
```

### A.2 \_\_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 __all__ = ['gui', 'matrices', 'typing_']
12
13 __version__ = '0.2.0'
```

### A.3 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, Optional, Union
16
17 import numpy as np
18
19 from .parse import parse_matrix_expression, validate_matrix_expression
20 from lintrans.typing_ import is_matrix_type, MatrixType
21
22
23 class MatrixWrapper:
```

```

24     """A wrapper class to hold all possible matrices and allow access to them.
25
26     .. note::
27         When defining a custom matrix, its name must be a capital letter and cannot be ``I``.
28
29     The contained matrices can be accessed and assigned to using square bracket notation.
30
31     :Example:
32
33     >>> wrapper = MatrixWrapper()
34     >>> wrapper['I']
35     array([[1., 0.],
36            [0., 1.]])
37     >>> wrapper['M'] # Returns None
38     >>> wrapper['M'] = np.array([[1, 2], [3, 4]])
39     >>> wrapper['M']
40     array([[1., 2.],
41            [3., 4.]])
42     """
43
44     def __init__(self):
45         """Initialise a :class:`MatrixWrapper` object with a dictionary of matrices which can be accessed."""
46         self._matrices: dict[str, Optional[Union[MatrixType, str]]] = {
47             'A': None, 'B': None, 'C': None, 'D': None,
48             'E': None, 'F': None, 'G': None, 'H': None,
49             'I': np.eye(2), # I is always defined as the identity matrix
50             'J': None, 'K': None, 'L': None, 'M': None,
51             'N': None, 'O': None, 'P': None, 'Q': None,
52             'R': None, 'S': None, 'T': None, 'U': None,
53             'V': None, 'W': None, 'X': None, 'Y': None,
54             'Z': None
55         }
56
57     def __repr__(self) -> str:
58         """Return a nice string repr of the :class:`MatrixWrapper` for debugging."""
59         defined_matrices = ''.join([k for k, v in self._matrices.items() if v is not None])
60         return f'<{self.__class__.__module__}.{self.__class__.__name__} object with ' \
61             f'{len(defined_matrices)} defined matrices: '{defined_matrices}'>'
62
63     def __eq__(self, other: Any) -> bool:
64         """Check for equality in wrappers by comparing dictionaries.
65
66         :param Any other: The object to compare this wrapper to
67         """
68         if not isinstance(other, self.__class__):
69             return NotImplemented
70
71         # We loop over every matrix and check if every value is equal in each
72         for name in self._matrices:
73             s_matrix = self[name]
74             o_matrix = other[name]
75
76             if s_matrix is None and o_matrix is None:
77                 continue
78
79             elif (s_matrix is None and o_matrix is not None) or \
80                  (s_matrix is not None and o_matrix is None):
81                 return False
82
83             # This is mainly to satisfy mypy, because we know these must be matrices
84             elif not is_matrix_type(s_matrix) or not is_matrix_type(o_matrix):
85                 return False
86
87             # Now we know they're both NumPy arrays
88             elif np.array_equal(s_matrix, o_matrix):
89                 continue
90
91             else:
92                 return False
93
94         return True
95
96     def __hash__(self) -> int:

```

```

97         """Return the hash of the matrices dictionary."""
98         return hash(self._matrices)
99
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 (may 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*)\d*)', name)) is not None:
117             return create_rotation_matrix(float(match.group(1)))
118
119         if name not in self._matrices:
120             raise NameError(f'Unrecognised matrix name "{name}"')
121
122         # We copy the matrix before we return it so the user can't accidentally mutate the matrix
123         matrix = copy(self._matrices[name])
124
125         if isinstance(matrix, str):
126             return self.evaluate_expression(matrix)
127
128         return matrix
129
130     def __setitem__(self, name: str, new_matrix: Optional[Union[MatrixType, str]]) -> None:
131         """Set the value of matrix ``name`` with the new_matrix.
132
133         The new matrix may be a simple 2x2 NumPy array, or it could be a string, representing an
134         expression in terms of other, previously defined matrices.
135
136         :param str name: The name of the matrix to set the value of
137         :param Optional[Union[MatrixType, str]] new_matrix: The value of the new matrix (may be None)
138
139         :raises NameError: If the name isn't a legal matrix name
140         :raises TypeError: If the matrix isn't a valid 2x2 NumPy array or expression in terms of other defined
141         ↪ matrices
142         :raises ValueError: If you attempt to define a matrix in terms of itself
143         """
144         if not (name in self._matrices and name != 'I'):
145             raise NameError('Matrix name is illegal')
146
147         if new_matrix is None:
148             self._matrices[name] = None
149             return
150
151         if isinstance(new_matrix, str):
152             if self.is_valid_expression(new_matrix):
153                 if name not in self._matrices:
154                     self._matrices[name] = new_matrix
155                     return
156             else:
157                 raise ValueError('Cannot define a matrix recursively')
158
159         if not is_matrix_type(new_matrix):
160             raise TypeError('Matrix must be a 2x2 NumPy array')
161
162         # All matrices must have float entries
163         a = float(new_matrix[0][0])
164         b = float(new_matrix[0][1])
165         c = float(new_matrix[1][0])
166         d = float(new_matrix[1][1])
167
168         self._matrices[name] = np.array([[a, b], [c, d]])

```

```

169 def get_expression(self, name: str) -> Optional[str]:
170     """If the named matrix is defined as an expression, return that expression, else return None.
171
172     :param str name: The name of the matrix
173     :returns Optional[str]: The expression that the matrix is defined as, or None
174
175     :raises NameError: If the name is invalid
176     """
177     if name not in self._matrices:
178         raise NameError('Matrix must have a legal name')
179
180     matrix = self._matrices[name]
181     if isinstance(matrix, str):
182         return matrix
183
184     return None
185
186 def is_valid_expression(self, expression: str) -> bool:
187     """Check if the given expression is valid, using the context of the wrapper.
188
189     This method calls :func:`lintrans.matrices.parse.validate_matrix_expression`, but also
190     ensures that all the matrices in the expression are defined in the wrapper.
191
192     :param str expression: The expression to validate
193     :returns bool: Whether the expression is valid in this wrapper
194
195     :raises LinAlgError: If a matrix is defined in terms of the inverse of a singular matrix
196     """
197     # Get rid of the transposes to check all capital letters
198     new_expression = expression.replace('^T', '').replace('^{T}', '')
199
200     # Make sure all the referenced matrices are defined
201     for matrix in [x for x in new_expression if re.match('[A-Z]', x)]:
202         if self[matrix] is None:
203             return False
204
205         if (expr := self.get_expression(matrix)) is not None:
206             if not self.is_valid_expression(expr):
207                 return False
208
209     return validate_matrix_expression(expression)
210
211 def evaluate_expression(self, expression: str) -> MatrixType:
212     """Evaluate a given expression and return the matrix evaluation.
213
214     :param str expression: The expression to be parsed
215     :returns MatrixType: The matrix result of the expression
216
217     :raises ValueError: If the expression is invalid
218     """
219     if not self.is_valid_expression(expression):
220         raise ValueError('The expression is invalid')
221
222     parsed_result = parse_matrix_expression(expression)
223     final_groups: list[list[MatrixType]] = []
224
225     for group in parsed_result:
226         f_group: list[MatrixType] = []
227
228         for matrix in group:
229             if matrix[2] == 'T':
230                 m = self[matrix[1]]
231
232                 # This assertion is just so mypy doesn't complain
233                 # We know this won't be None, because we know that this matrix is defined in this wrapper
234                 assert m is not None
235                 matrix_value = m.T
236
237             else:
238                 matrix_value = np.linalg.matrix_power(self[matrix[1]],
239                                                         1 if (index := matrix[2]) == '' else int(index))
240
241             matrix_value *= 1 if (multiplier := matrix[0]) == '' else float(multiplier)

```



```

242         f_group.append(matrix_value)
243
244         final_groups.append(f_group)
245
246     return reduce(add, [reduce(matmul, group) for group in final_groups])
247
248
249 def create_rotation_matrix(angle: float, *, degrees: bool = True) -> MatrixType:
250     """Create a matrix representing a rotation (anticlockwise) by the given angle.
251
252     :Example:
253
254     >>> create_rotation_matrix(30)
255     array([[ 0.8660254, -0.5      ],
256           [ 0.5      ,  0.8660254]])
257     >>> create_rotation_matrix(45)
258     array([[ 0.70710678, -0.70710678],
259           [ 0.70710678,  0.70710678]])
260     >>> create_rotation_matrix(np.pi / 3, degrees=False)
261     array([[ 0.5      , -0.8660254],
262           [ 0.8660254,  0.5      ]])
263
264     :param float angle: The angle to rotate anticlockwise by
265     :param bool degrees: Whether to interpret the angle as degrees (True) or radians (False)
266     :returns MatrixType: The resultant matrix
267     """
268     rad = np.deg2rad(angle) if degrees else angle
269     return np.array([
270         [np.cos(rad), -1 * np.sin(rad)],
271         [np.sin(rad), np.cos(rad)]
272     ])

```

#### A.4 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
10 from .wrapper import create_rotation_matrix, MatrixWrapper
11
12 __all__ = ['create_rotation_matrix', 'MatrixWrapper', 'parse']

```

#### A.5 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 typing import Pattern
13
14 from lintrans.typing import MatrixParseList
15
16
17 class MatrixParseError(Exception):
18     """A simple exception to be raised when an error is found when parsing."""
19

```

```

20
21 def compile_valid_expression_pattern() -> Pattern[str]:
22     """Compile the single RegEx pattern that will match a valid matrix expression."""
23     digit_no_zero = '[123456789]'
24     digits = '\\d+'
25     integer_no_zero = digit_no_zero + '(' + digits + ')?'
26     real_number = f'({integer_no_zero}(\\.({digits})?|0?\\.({digits}))'
27
28     index_content = f'(-?{integer_no_zero}|T)'
29     index = f'\\^{{{index_content}}}|\\^{index_content}'
30     matrix_identifier = f'([A-Z]|rot\\((-?{real_number}\\))'
31     matrix = '(' + real_number + '?' + matrix_identifier + index + '?)'
32     expression = f'^~?{matrix}+((\\+|-){matrix}+)*$'
33
34     return re.compile(expression)
35
36
37 # This is an expensive pattern to compile, so we compile it when this module is initialized
38 valid_expression_pattern = compile_valid_expression_pattern()
39
40
41 def validate_matrix_expression(expression: str) -> bool:
42     """Validate the given matrix expression.
43
44     This function simply checks the expression against the BNF schema documented in
45     :ref:`expression-syntax-docs`. It is not aware of which matrices are actually defined
46     in a wrapper. For an aware version of this function, use the
47     :meth:`lintrans.matrices.wrapper.MatrixWrapper.is_valid_expression` method.
48
49     :param str expression: The expression to be validated
50     :returns bool: Whether the expression is valid according to the schema
51     """
52     # Remove all whitespace
53     expression = re.sub(r'\\s', '', expression)
54
55     match = valid_expression_pattern.match(expression)
56
57     if match is None:
58         return False
59
60     # Check if the whole expression was matched against
61     return expression == match.group(0)
62
63
64 def parse_matrix_expression(expression: str) -> MatrixParseList:
65     """Parse the matrix expression and return a :data:`lintrans.typing_.MatrixParseList`.
66
67     :Example:
68
69     >>> parse_matrix_expression('A')
70     [[(' ', 'A', ' ')]]
71     >>> parse_matrix_expression('-3M^2')
72     [[(' -3', 'M', '2')]]
73     >>> parse_matrix_expression('1.2rot(12)^{3}2B^T')
74     [[('1.2', 'rot(12)', '3'), ('2', 'B', 'T')]]
75     >>> parse_matrix_expression('A^2 + 3B')
76     [[(' ', 'A', '2'), [('3', 'B', ' ')]]
77     >>> parse_matrix_expression('-3A^{-1}3B^T - 45M^2')
78     [[(' -3', 'A', ' -1'), ('3', 'B', 'T'), [(' -45', 'M', '2')]]
79     >>> parse_matrix_expression('5.3A^{4} 2.6B^{-2} + 4.6D^T 8.9E^{-1}')
80     [[('5.3', 'A', '4'), ('2.6', 'B', ' -2'), [('4.6', 'D', 'T'), ('8.9', 'E', ' -1')]]
81
82     :param str expression: The expression to be parsed
83     :returns: A list of parsed components
84     :rtype: :data:`lintrans.typing_.MatrixParseList`
85     """
86     # Remove all whitespace
87     expression = re.sub(r'\\s', '', expression)
88
89     # Check if it's valid
90     if not validate_matrix_expression(expression):
91         raise MatrixParseError('Invalid expression')
92

```

```

93     # Wrap all exponents and transposition powers with {}
94     expression = re.sub(r'(?<=\\^)(-?\\d+|T)(?=[^]]|$)', r'{\\g<0>}', expression)
95
96     # Remove any standalone minuses
97     expression = re.sub(r'-(?=[A-Z])', '-1', expression)
98
99     # Replace subtractions with additions
100    expression = re.sub(r'-(?=[\\d+\\.]?\\d*([A-Z]|rot))', '+-', expression)
101
102    # Get rid of a potential leading + introduced by the last step
103    expression = re.sub(r'^\\+', '', expression)
104
105    return [
106        [
107            # The tuple returned by re.findall is (multiplier, matrix identifier, full index, stripped index),
108            # so we have to remove the full index, which contains the {}
109            (t[0], t[1], t[3])
110            for t in re.findall(r'(-?\\d*\\.)?\\d*([A-Z]|rot\\(-?\\d+\\.)?\\d*\\)(\\^{(-?\\d+|T)})?', group)
111        ]
112        # We just split the expression by '+' to have separate groups
113        for group in expression.split('+')
114    ]

```

## A.6 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 :external:mod:`typing`. I don't quite know how this collision occurs, but renaming
12     this module fixed the problem.
13 """
14
15 from __future__ import annotations
16
17 from typing import Any, TypeGuard
18
19 from numpy import ndarray
20 from nptyping import NDArray, Float
21
22 __all__ = ['is_matrix_type', 'MatrixType', 'MatrixParseList']
23
24 MatrixType = NDArray[(2, 2), Float]
25 """This type represents a 2x2 matrix as a NumPy array."""
26
27 MatrixParseList = list[list[tuple[str, str, str]]]
28 """This is a list containing lists of tuples. Each tuple represents a matrix and is ``(multiplier,
29 matrix_identifier, index)`` where all of them are strings. These matrix-representing tuples are
30 contained in lists which represent multiplication groups. Every matrix in the group should be
31 multiplied together, in order. These multiplication group lists are contained by a top level list,
32 which is this type. Once these multiplication group lists have been evaluated, they should be summed.
33
34 In the tuples, the multiplier is a string representing a real number, the matrix identifier
35 is a capital letter or ``rot(x)`` where x is a real number angle, and the index is a string
36 representing an integer, or it's the letter ``T`` for transpose.
37 """
38
39
40 def is_matrix_type(matrix: Any) -> TypeGuard[NDArray[(2, 2), Float]]:
41     """Check if the given value is a valid matrix type.
42
43     .. note::
44         This function is a TypeGuard, meaning if it returns True, then the
45         passed value must be a :attr:`lintrans.typing_.MatrixType`.

```

```

46     """
47     return isinstance(matrix, ndarray) and matrix.shape == (2, 2)

```

## A.7 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 sys
12  import webbrowser
13  from copy import deepcopy
14  from typing import Type
15
16  import numpy as np
17  from numpy import linalg
18  from numpy.linalg import LinAlgError
19  from PyQt5 import QtWidgets
20  from PyQt5.QtCore import pyqtSlot, QThread
21  from PyQt5.QtGui import QKeySequence
22  from PyQt5.QtWidgets import QApplication, QHBoxLayout, QMainWindow, QMessageBox,
23      QShortcut, QSizePolicy, QSpacerItem, QVBoxLayout
24
25  from lintrans.matrices import MatrixWrapper
26  from lintrans.matrices.parse import validate_matrix_expression
27  from lintrans.typing import MatrixType
28  from .dialogs import DefineAsAnExpressionDialog, DefineDialog, DefineNumericallyDialog, DefineVisuallyDialog
29  from .dialogs.settings import DisplaySettingsDialog
30  from .plots import VisualizeTransformationWidget
31  from .settings import DisplaySettings
32  from .validate import MatrixExpressionValidator
33
34
35  class LintransMainWindow(QMainWindow):
36      """This class provides a main window for the GUI using the Qt framework.
37
38      This class should not be used directly, instead call :func:`lintrans.gui.main_window.main` to create the GUI.
39      """
40
41      def __init__(self):
42          """Create the main window object, and create and arrange every widget in it.
43
44          This doesn't show the window, it just constructs it. Use :func:`lintrans.gui.main_window.main` to show the
45          ↩ GUI.
46          """
47          super().__init__()
48          self.matrix_wrapper = MatrixWrapper()
49
50          self.setWindowTitle('lintrans')
51          self.setMinimumSize(1000, 750)
52
53          self.animating: bool = False
54          self.animating_sequence: bool = False
55
56          # === Create menubar
57
58          self.menubar = QtWidgets.QMenuBar(self)
59
60          self.menu_file = QtWidgets.QMenu(self.menubar)
61          self.menu_file.setTitle('&File')
62
63          self.menu_help = QtWidgets.QMenu(self.menubar)
64          self.menu_help.setTitle('&Help')

```

```

65
66     self.action_new = QtWidgets.QAction(self)
67     self.action_new.setText('&New')
68     self.action_new.setShortcut('Ctrl+N')
69     self.action_new.triggered.connect(lambda: print('new'))
70
71     self.action_open = QtWidgets.QAction(self)
72     self.action_open.setText('&Open')
73     self.action_open.setShortcut('Ctrl+O')
74     self.action_open.triggered.connect(lambda: print('open'))
75
76     self.action_save = QtWidgets.QAction(self)
77     self.action_save.setText('&Save')
78     self.action_save.setShortcut('Ctrl+S')
79     self.action_save.triggered.connect(lambda: print('save'))
80
81     self.action_save_as = QtWidgets.QAction(self)
82     self.action_save_as.setText('Save as...')
83     self.action_save_as.triggered.connect(lambda: print('save as'))
84
85     self.action_tutorial = QtWidgets.QAction(self)
86     self.action_tutorial.setText('&Tutorial')
87     self.action_tutorial.setShortcut('F1')
88     self.action_tutorial.triggered.connect(lambda: print('tutorial'))
89
90     self.action_docs = QtWidgets.QAction(self)
91     self.action_docs.setText('&Docs')
92     self.action_docs.triggered.connect(
93         lambda: webbrowser.open_new_tab('https://doctordalek1963.github.io/lintrans/docs/index.html')
94     )
95
96     self.action_about = QtWidgets.QAction(self)
97     self.action_about.setText('&About')
98     self.action_about.triggered.connect(lambda: print('about'))
99
100     # TODO: Implement these actions and enable them
101     self.action_new.setEnabled(False)
102     self.action_open.setEnabled(False)
103     self.action_save.setEnabled(False)
104     self.action_save_as.setEnabled(False)
105     self.action_tutorial.setEnabled(False)
106     self.action_about.setEnabled(False)
107
108     self.menu_file.addAction(self.action_new)
109     self.menu_file.addAction(self.action_open)
110     self.menu_file.addSeparator()
111     self.menu_file.addAction(self.action_save)
112     self.menu_file.addAction(self.action_save_as)
113     self.menu_file.addSeparator()
114     self.menu_file.addAction(self.action_about)
115
116     self.menu_help.addAction(self.action_tutorial)
117     self.menu_help.addAction(self.action_docs)
118
119     self.menubar.addAction(self.menu_file.menuAction())
120     self.menubar.addAction(self.menu_help.menuAction())
121
122     self.setMenuBar(self.menubar)
123
124     # === Create widgets
125
126     # Left layout: the plot and input box
127
128     self.plot = VisualizeTransformationWidget(DisplaySettings(), self)
129
130     self.lineedit_expression_box = QtWidgets.QLineEdit(self)
131     self.lineedit_expression_box.setPlaceholderText('Enter matrix expression...')
132     self.lineedit_expression_box.setValidator(MatrixExpressionValidator(self))
133     self.lineedit_expression_box.textChanged.connect(self.update_render_buttons)
134
135     # Right layout: all the buttons
136
137     # Misc buttons

```

```

138
139     self.button_create_polygon = QtWidgets.QPushButton(self)
140     self.button_create_polygon.setText('Create polygon')
141     # self.button_create_polygon.clicked.connect(self.create_polygon)
142     self.button_create_polygon.setToolTip('Define a new polygon to view the transformation of')
143
144     # TODO: Implement this and enable button
145     self.button_create_polygon.setEnabled(False)
146
147     self.button_change_display_settings = QtWidgets.QPushButton(self)
148     self.button_change_display_settings.setText('Change\ndisplay settings')
149     self.button_change_display_settings.clicked.connect(self.dialog_change_display_settings)
150     self.button_change_display_settings.setToolTip(
151         "Change which things are rendered and how they're rendered<br><b>(Ctrl + D)</b>"
152     )
153     QShortcut(QKeySequence('Ctrl+D'), self).activated.connect(self.button_change_display_settings.click)
154
155     self.button_reset_zoom = QtWidgets.QPushButton(self)
156     self.button_reset_zoom.setText('Reset zoom')
157     self.button_reset_zoom.clicked.connect(self.reset_zoom)
158     self.button_reset_zoom.setToolTip('Reset the zoom level back to normal<br><b>(Ctrl + Shift + R)</b>')
159     QShortcut(QKeySequence('Ctrl+Shift+R'), self).activated.connect(self.button_reset_zoom.click)
160
161     # Define new matrix buttons and their groupbox
162
163     self.button_define_visually = QtWidgets.QPushButton(self)
164     self.button_define_visually.setText('Visually')
165     self.button_define_visually.setToolTip('Drag the basis vectors<br><b>(Alt + 1)</b>')
166     self.button_define_visually.clicked.connect(lambda: self.dialog_define_matrix(DefineVisuallyDialog))
167     QShortcut(QKeySequence('Alt+1'), self).activated.connect(self.button_define_visually.click)
168
169     self.button_define_numerically = QtWidgets.QPushButton(self)
170     self.button_define_numerically.setText('Numerically')
171     self.button_define_numerically.setToolTip('Define a matrix just with numbers<br><b>(Alt + 2)</b>')
172     self.button_define_numerically.clicked.connect(lambda: self.dialog_define_matrix(DefineNumericallyDialog))
173     QShortcut(QKeySequence('Alt+2'), self).activated.connect(self.button_define_numerically.click)
174
175     self.button_define_as_expression = QtWidgets.QPushButton(self)
176     self.button_define_as_expression.setText('As an expression')
177     self.button_define_as_expression.setToolTip('Define a matrix in terms of other matrices<br><b>(Alt +
178     ↪ 3)</b>')
179     self.button_define_as_expression.clicked.connect(lambda:
180     ↪ self.dialog_define_matrix(DefineAsAnExpressionDialog))
181     QShortcut(QKeySequence('Alt+3'), self).activated.connect(self.button_define_as_expression.click)
182
183     self.vlay_define_new_matrix = QVBoxLayout()
184     self.vlay_define_new_matrix.setSpacing(20)
185     self.vlay_define_new_matrix.addWidget(self.button_define_visually)
186     self.vlay_define_new_matrix.addWidget(self.button_define_numerically)
187     self.vlay_define_new_matrix.addWidget(self.button_define_as_expression)
188
189     self.groupbox_define_new_matrix = QtWidgets.QGroupBox('Define a new matrix', self)
190     self.groupbox_define_new_matrix.setLayout(self.vlay_define_new_matrix)
191
192     # Render buttons
193
194     self.button_reset = QtWidgets.QPushButton(self)
195     self.button_reset.setText('Reset')
196     self.button_reset.clicked.connect(self.reset_transformation)
197     self.button_reset.setToolTip('Reset the visualized transformation back to the identity<br><b>(Ctrl +
198     ↪ R)</b>')
199     QShortcut(QKeySequence('Ctrl+R'), self).activated.connect(self.button_reset.click)
200
201     self.button_render = QtWidgets.QPushButton(self)
202     self.button_render.setText('Render')
203     self.button_render.setEnabled(False)
204     self.button_render.clicked.connect(self.render_expression)
205     self.button_render.setToolTip('Render the expression<br><b>(Ctrl + Enter)</b>')
206     QShortcut(QKeySequence('Ctrl+Return'), self).activated.connect(self.button_render.click)
207
208     self.button_animate = QtWidgets.QPushButton(self)
209     self.button_animate.setText('Animate')
210     self.button_animate.setEnabled(False)

```

```

208     self.button_animate.clicked.connect(self.animate_expression)
209     self.button_animate.setToolTip('Animate the expression<br><b>(Ctrl + Shift + Enter)</b>')
210     QShortcut(QKeySequence('Ctrl+Shift+Return'), self).activated.connect(self.button_animate.click)
211
212     # === Arrange widgets
213
214     self.vlay_left = QVBoxLayout()
215     self.vlay_left.addWidget(self.plot)
216     self.vlay_left.addWidget(self.lineedit_expression_box)
217
218     self.vlay_misc_buttons = QVBoxLayout()
219     self.vlay_misc_buttons.setSpacing(20)
220     self.vlay_misc_buttons.addWidget(self.button_create_polygon)
221     self.vlay_misc_buttons.addWidget(self.button_change_display_settings)
222     self.vlay_misc_buttons.addWidget(self.button_reset_zoom)
223
224     self.vlay_render = QVBoxLayout()
225     self.vlay_render.setSpacing(20)
226     self.vlay_render.addWidget(self.button_reset)
227     self.vlay_render.addWidget(self.button_animate)
228     self.vlay_render.addWidget(self.button_render)
229
230     self.vlay_right = QVBoxLayout()
231     self.vlay_right.setSpacing(50)
232     self.vlay_right.addLayout(self.vlay_misc_buttons)
233     self.vlay_right.addItem(QSpacerItem(100, 2, hPolicy=QSizePolicy.Minimum, vPolicy=QSizePolicy.Expanding))
234     self.vlay_right.addWidget(self.groupbox_define_new_matrix)
235     self.vlay_right.addItem(QSpacerItem(100, 2, hPolicy=QSizePolicy.Minimum, vPolicy=QSizePolicy.Expanding))
236     self.vlay_right.addLayout(self.vlay_render)
237
238     self.hlay_all = QHBoxLayout()
239     self.hlay_all.setSpacing(15)
240     self.hlay_all.addLayout(self.vlay_left)
241     self.hlay_all.addLayout(self.vlay_right)
242
243     self.central_widget = QWidget()
244     self.central_widget.setLayout(self.hlay_all)
245     self.central_widget.setContentsMargins(10, 10, 10, 10)
246
247     self.setCentralWidget(self.central_widget)
248
249     def update_render_buttons(self) -> None:
250         """Enable or disable the render and animate buttons according to whether the matrix expression is valid."""
251         text = self.lineedit_expression_box.text()
252
253         # Let's say that the user defines a non-singular matrix A, then defines B as A^-1
254         # If they then redefine A and make it singular, then we get a LinAlgError when
255         # trying to evaluate an expression with B in it
256         # To fix this, we just do naive validation rather than aware validation
257         if ',' in text:
258             self.button_render.setEnabled(False)
259
260             try:
261                 valid = all(self.matrix_wrapper.is_valid_expression(x) for x in text.split(','))
262             except LinAlgError:
263                 valid = all(validate_matrix_expression(x) for x in text.split(','))
264
265             self.button_animate.setEnabled(valid)
266
267         else:
268             try:
269                 valid = self.matrix_wrapper.is_valid_expression(text)
270             except LinAlgError:
271                 valid = validate_matrix_expression(text)
272
273             self.button_render.setEnabled(valid)
274             self.button_animate.setEnabled(valid)
275
276     @pyqtSlot()
277     def reset_zoom(self) -> None:
278         """Reset the zoom level back to normal."""
279         self.plot.grid_spacing = self.plot.default_grid_spacing
280         self.plot.update()

```

```

281
282 @pyqtSlot()
283 def reset_transformation(self) -> None:
284     """Reset the visualized transformation back to the identity."""
285     self.plot.visualize_matrix_transformation(self.matrix_wrapper['I'])
286     self.animating = False
287     self.animating_sequence = False
288     self.plot.update()
289
290 @pyqtSlot()
291 def render_expression(self) -> None:
292     """Render the transformation given by the expression in the input box."""
293     try:
294         matrix = self.matrix_wrapper.evaluate_expression(self.lineEdit_expression_box.text())
295
296     except LinAlgError:
297         self.show_error_message('Singular matrix', 'Cannot take inverse of singular matrix')
298         return
299
300     if self.is_matrix_too_big(matrix):
301         self.show_error_message('Matrix too big', "This matrix doesn't fit on the canvas")
302         return
303
304     self.plot.visualize_matrix_transformation(matrix)
305     self.plot.update()
306
307 @pyqtSlot()
308 def animate_expression(self) -> None:
309     """Animate from the current matrix to the matrix in the expression box."""
310     self.button_render.setEnabled(False)
311     self.button_animate.setEnabled(False)
312
313     matrix_start: MatrixType = np.array([
314         [self.plot.point_i[0], self.plot.point_j[0]],
315         [self.plot.point_i[1], self.plot.point_j[1]]
316     ])
317
318     text = self.lineEdit_expression_box.text()
319
320     # If there's commas in the expression, then we want to animate each part at a time
321     if ',' in text:
322         current_matrix = matrix_start
323         self.animating_sequence = True
324
325         # For each expression in the list, right multiply it by the current matrix,
326         # and animate from the current matrix to that new matrix
327         for expr in text.split(',')[::-1]:
328             try:
329                 new_matrix = self.matrix_wrapper.evaluate_expression(expr) @ current_matrix
330             except LinAlgError:
331                 self.show_error_message('Singular matrix', 'Cannot take inverse of singular matrix')
332                 return
333
334             if not self.animating_sequence:
335                 break
336
337             self.animate_between_matrices(current_matrix, new_matrix)
338             current_matrix = new_matrix
339
340             # Here we just redraw and allow for other events to be handled while we pause
341             self.plot.update()
342             QApplication.processEvents()
343             QThread.sleep(self.plot.display_settings.animation_pause_length)
344
345             self.animating_sequence = False
346
347     # If there's no commas, then just animate directly from the start to the target
348     else:
349         # Get the target matrix and it's determinant
350         try:
351             matrix_target = self.matrix_wrapper.evaluate_expression(text)
352
353         except LinAlgError:

```



```

354         self.show_error_message('Singular matrix', 'Cannot take inverse of singular matrix')
355         return
356
357     # The concept of applicative animation is explained in /gui/settings.py
358     if self.plot.display_settings.applicative_animation:
359         matrix_target = matrix_target @ matrix_start
360
361     # If we want a transitional animation and we're animating the same matrix, then restart the animation
362     # We use this check rather than equality because of small floating point errors
363     elif (abs(matrix_start - matrix_target) < 1e-12).all():
364         matrix_start = self.matrix_wrapper['I']
365
366     # We pause here for 200 ms to make the animation look a bit nicer
367     self.plot.visualize_matrix_transformation(matrix_start)
368     self.plot.update()
369     QApplication.processEvents()
370     QThread.msleep(200)
371
372     self.animate_between_matrices(matrix_start, matrix_target)
373
374     self.update_render_buttons()
375
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 normalise
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
446
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         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)
470
471     @pyqtSlot()
472     def assign_matrix_wrapper(self) -> None:
473         """Assign a new value to ``self.matrix_wrapper`` and give the expression box focus."""
474         self.matrix_wrapper = self.sender().matrix_wrapper
475         self.linedit_expression_box.setFocus()
476         self.update_render_buttons()
477
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))
484
485     @pyqtSlot(DisplaySettings)
486     def assign_display_settings(self, display_settings: DisplaySettings) -> None:
487         """Assign a new value to ``self.plot.display_settings`` and give the expression box focus."""
488         self.plot.display_settings = display_settings
489         self.plot.update()
490         self.linedit_expression_box.setFocus()
491         self.update_render_buttons()
492
493     def show_error_message(self, title: str, text: str, info: str | None = None) -> None:
494         """Show an error message in a dialog box.
495
496         :param str title: The window title of the dialog box
497         :param str text: The simple error message
498         :param info: The more informative error message

```

```

499         :type info: Optional[str]
500         """
501         dialog = QMessageBox(self)
502         dialog.setIcon(QMessageBox.Critical)
503         dialog.setWindowTitle(title)
504         dialog.setText(text)
505
506         if info is not None:
507             dialog.setInformativeText(info)
508
509         dialog.open()
510
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)
513
514     def is_matrix_too_big(self, matrix: MatrixType) -> bool:
515         """Check if the given matrix will actually fit onto the canvas.
516
517         Convert the elements of the matrix to canvas coords and make sure they fit within Qt's 32-bit integer limit.
518
519         :param MatrixType matrix: The matrix to check
520         :returns bool: Whether the matrix fits on the canvas
521         """
522         coords: list[tuple[int, int]] = [self.plot.canvas_coords(*vector) for vector in matrix.T]
523
524         for x, y in coords:
525             if not (-2147483648 <= x <= 2147483647 and -2147483648 <= y <= 2147483647):
526                 return True
527
528         return False
529
530
531     def main(args: list[str]) -> None:
532         """Run the GUI by creating and showing an instance of :class:`LintransMainWindow`.
533
534         :param list[str] args: The args to pass to :class:`QApplication` (normally ``sys.argv``)
535         """
536         app = QApplication(args)
537         window = LintransMainWindow()
538         window.show()
539         sys.exit(app.exec_())

```

## A.8 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       smoothen_determinant: bool = True
19       """This controls whether we want the determinant to change smoothly during the animation.
20
21       .. note::
22           Even if this is True, it will be ignored if we're animating from a positive det matrix to
23           a negative det matrix, or vice versa, because if we try to smoothly animate that determinant,
24           things blow up and the app often crashes.
25       """
26

```

```

27     applicative_animation: bool = True
28     """There are two types of simple animation, transitional and applicative.
29
30     Let ``C`` be the matrix representing the currently displayed transformation, and let ``T`` be the target matrix.
31     Transitional animation means that we animate directly from ``C`` from ``T``,
32     and applicative animation means that we animate from ``C`` to ``TC``, so we apply ``T`` to ``C``.
33     """
34
35     animation_pause_length: int = 400
36     """This is the number of milliseconds that we wait between animations when using comma syntax."""
37
38     draw_determinant_parallelogram: bool = False
39     """This controls whether or not we should shade the parallelogram representing the determinant of the matrix."""
40
41     draw_determinant_text: bool = True
42     """This controls whether we should write the text value of the determinant inside the parallelogram.
43
44     The text only gets draw if :attr:`draw_determinant_parallelogram` is also True.
45     """
46
47     draw_eigenvectors: bool = False
48     """This controls whether we should draw the eigenvectors of the transformation."""
49
50     draw_eigenlines: bool = False
51     """This controls whether we should draw the eigenlines of the transformation."""

```

## A.9 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.10 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
13 from PyQt5.QtGui import QValidator
14
15 from lintrans.matrices import parse
16
17 class MatrixExpressionValidator(QValidator):
18     """This class validates matrix expressions in an Qt input box."""
19
20     def validate(self, text: str, pos: int) -> tuple[QValidator.State, str, int]:
21         """Validate the given text according to the rules defined in the :mod:`lintrans.matrices` module."""
22         clean_text = re.sub(r'[\sA-Z\d.rot()^{};+-]', '', text)
23
24         if clean_text == '':
25

```

```

26         if parse.validate_matrix_expression(clean_text):
27             return QValidator.Acceptable, text, pos
28         else:
29             return QValidator.Intermediate, text, pos
30
31     return QValidator.Invalid, text, pos

```

## A.11 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
13  from PyQt5 import QtWidgets
14  from PyQt5.QtGui import QIntValidator, QKeyEvent, QKeySequence
15  from PyQt5.QtWidgets import QCheckBox, QDialog, QGroupBox, QHBoxLayout, QShortcut, QSizePolicy, QSpacerItem,
   ↳ QVBoxLayout
16
17  from lintrans.gui.settings import DisplaySettings
18
19
20  class SettingsDialog(QDialog):
21     """An abstract superclass for other simple dialogs."""
22
23     def __init__(self, *args, **kwargs):
24         """Create the widgets and layout of the dialog, passing ``*args`` and ``**kwargs`` to super."""
25         super().__init__(*args, **kwargs)
26
27         # === Create the widgets
28
29         self.button_confirm = QtWidgets.QPushButton(self)
30         self.button_confirm.setText('Confirm')
31         self.button_confirm.clicked.connect(self.confirm_settings)
32         self.button_confirm.setToolTip('Confirm these new settings<br><b>(Ctrl + Enter)</b>')
33         QShortcut(QKeySequence('Ctrl+Return'), self).activated.connect(self.button_confirm.click)
34
35         self.button_cancel = QtWidgets.QPushButton(self)
36         self.button_cancel.setText('Cancel')
37         self.button_cancel.clicked.connect(self.reject)
38         self.button_cancel.setToolTip('Revert these settings<br><b>(Escape)</b>')
39
40         # === Arrange the widgets
41
42         self.setContentsMargins(10, 10, 10, 10)
43
44         self.hlay_buttons = QHBoxLayout()
45         self.hlay_buttons.setSpacing(20)
46         self.hlay_buttons.addItem(QSpacerItem(50, 5, hPolicy=QSizePolicy.Expanding, vPolicy=QSizePolicy.Minimum))
47         self.hlay_buttons.addWidget(self.button_cancel)
48         self.hlay_buttons.addWidget(self.button_confirm)
49
50         self.vlay_options = QVBoxLayout()
51         self.vlay_options.setSpacing(20)
52
53         self.vlay_all = QVBoxLayout()
54         self.vlay_all.setSpacing(20)
55         self.vlay_all.addLayout(self.vlay_options)
56         self.vlay_all.addLayout(self.hlay_buttons)
57
58         self.setLayout(self.vlay_all)
59
60     @abc.abstractmethod

```

```

61     def load_settings(self) -> None:
62         """Load the current settings into the widgets."""
63
64     @abc.abstractmethod
65     def confirm_settings(self) -> None:
66         """Confirm the settings chosen in the dialog."""
67
68
69 class DisplaySettingsDialog(SettingsDialog):
70     """The dialog to allow the user to edit the display settings."""
71
72     def __init__(self, display_settings: DisplaySettings, *args, **kwargs):
73         """Create the widgets and layout of the dialog.
74
75         :param DisplaySettings display_settings: The :class:`lintrans.gui.settings.DisplaySettings` object to mutate
76         """
77         super().__init__(*args, **kwargs)
78
79         self.display_settings = display_settings
80         self.setWindowTitle('Change display settings')
81
82         self.dict_checkboxes: dict[str, QCheckBox] = dict()
83
84         # === Create the widgets
85
86         # Animations
87
88         self.checkbox_smooththen_determinant = QCheckBox(self)
89         self.checkbox_smooththen_determinant.setText('&Smoothen determinant')
90         self.checkbox_smooththen_determinant.setToolTip(
91             'Smoothly animate the determinant transition during animation (if possible)'
92         )
93         self.dict_checkboxes['s'] = self.checkbox_smooththen_determinant
94
95         self.checkbox_applicative_animation = QCheckBox(self)
96         self.checkbox_applicative_animation.setText('&Applicative animation')
97         self.checkbox_applicative_animation.setToolTip(
98             'Animate the new transformation applied to the current one,\n'
99             'rather than just that transformation on its own'
100        )
101        self.dict_checkboxes['a'] = self.checkbox_applicative_animation
102
103        self.label_animation_pause_length = QtWidgets.QLabel(self)
104        self.label_animation_pause_length.setText('Animation pause length (ms)')
105        self.label_animation_pause_length.setToolTip(
106            'How many milliseconds to pause for in comma-separated animations'
107        )
108
109        self.lineedit_animation_pause_length = QtWidgets.QLineEdit(self)
110        self.lineedit_animation_pause_length.setValidator(QIntValidator(1, 999, self))
111
112        # Matrix info
113
114        self.checkbox_draw_determinant_parallelogram = QCheckBox(self)
115        self.checkbox_draw_determinant_parallelogram.setText('Draw &determinant parallelogram')
116        self.checkbox_draw_determinant_parallelogram.setToolTip(
117            'Shade the parallelogram representing the determinant of the matrix'
118        )
119        self.checkbox_draw_determinant_parallelogram.clicked.connect(self.update_gui)
120        self.dict_checkboxes['d'] = self.checkbox_draw_determinant_parallelogram
121
122        self.checkbox_draw_determinant_text = QCheckBox(self)
123        self.checkbox_draw_determinant_text.setText('Draw determinant &text')
124        self.checkbox_draw_determinant_text.setToolTip(
125            'Write the text value of the determinant inside the parallelogram'
126        )
127        self.dict_checkboxes['t'] = self.checkbox_draw_determinant_text
128
129        self.checkbox_draw_eigenvectors = QCheckBox(self)
130        self.checkbox_draw_eigenvectors.setText('Draw &eigenvectors')
131        self.checkbox_draw_eigenvectors.setToolTip('Draw the eigenvectors of the transformations')
132        self.dict_checkboxes['e'] = self.checkbox_draw_eigenvectors
133

```

```

134     self.checkbox_draw_eigenlines = QCheckBox(self)
135     self.checkbox_draw_eigenlines.setText('Draw eigen&lines')
136     self.checkbox_draw_eigenlines.setToolTip('Draw the eigenlines (invariant lines) of the transformations')
137     self.dict_checkboxes['l'] = self.checkbox_draw_eigenlines
138
139     # === Arrange the widgets in QGroupBoxes
140
141     # Animations
142
143     self.hlay_animation_pause_length = QHBoxLayout()
144     self.hlay_animation_pause_length.addWidget(self.label_animation_pause_length)
145     self.hlay_animation_pause_length.addWidget(self.lineedit_animation_pause_length)
146
147     self.vlay_groupbox_animations = QVBoxLayout()
148     self.vlay_groupbox_animations.setSpacing(20)
149     self.vlay_groupbox_animations.addWidget(self.checkbox_smooththen_determinant)
150     self.vlay_groupbox_animations.addWidget(self.checkbox_applicative_animation)
151     self.vlay_groupbox_animations.addLayout(self.hlay_animation_pause_length)
152
153     self.groupbox_animations = QGroupBox('Animations', self)
154     self.groupbox_animations.setLayout(self.vlay_groupbox_animations)
155
156     # Matrix info
157
158     self.vlay_groupbox_matrix_info = QVBoxLayout()
159     self.vlay_groupbox_matrix_info.setSpacing(20)
160     self.vlay_groupbox_matrix_info.addWidget(self.checkbox_draw_determinant_parallelogram)
161     self.vlay_groupbox_matrix_info.addWidget(self.checkbox_draw_determinant_text)
162     self.vlay_groupbox_matrix_info.addWidget(self.checkbox_draw_eigenvectors)
163     self.vlay_groupbox_matrix_info.addWidget(self.checkbox_draw_eigenlines)
164
165     self.groupbox_matrix_info = QGroupBox('Matrix info', self)
166     self.groupbox_matrix_info.setLayout(self.vlay_groupbox_matrix_info)
167
168     self.vlay_options.addWidget(self.groupbox_animations)
169     self.vlay_options.addWidget(self.groupbox_matrix_info)
170
171     # Finally, we load the current settings and update the GUI
172     self.load_settings()
173     self.update_gui()
174
175     def load_settings(self) -> None:
176         """Load the current display settings into the widgets."""
177         # Animations
178         self.checkbox_smooththen_determinant.setChecked(self.display_settings.smoothen_determinant)
179         self.checkbox_applicative_animation.setChecked(self.display_settings.applicative_animation)
180         self.lineedit_animation_pause_length.setText(str(self.display_settings.animation_pause_length))
181
182         # Matrix info
183         self.checkbox_draw_determinant_parallelogram.setChecked(
184             ↪ self.display_settings.draw_determinant_parallelogram)
185         self.checkbox_draw_determinant_text.setChecked(self.display_settings.draw_determinant_text)
186         self.checkbox_draw_eigenvectors.setChecked(self.display_settings.draw_eigenvectors)
187         self.checkbox_draw_eigenlines.setChecked(self.display_settings.draw_eigenlines)
188
189     def confirm_settings(self) -> None:
190         """Build a :class:`lintrans.gui.settings.DisplaySettings` object and assign it."""
191         # Animations
192         self.display_settings.smoothen_determinant = self.checkbox_smooththen_determinant.isChecked()
193         self.display_settings.applicative_animation = self.checkbox_applicative_animation.isChecked()
194         self.display_settings.animation_pause_length = int(self.lineedit_animation_pause_length.text())
195
196         # Matrix info
197         self.display_settings.draw_determinant_parallelogram =
198             ↪ self.checkbox_draw_determinant_parallelogram.isChecked()
199         self.display_settings.draw_determinant_text = self.checkbox_draw_determinant_text.isChecked()
200         self.display_settings.draw_eigenvectors = self.checkbox_draw_eigenvectors.isChecked()
201         self.display_settings.draw_eigenlines = self.checkbox_draw_eigenlines.isChecked()
202
203         self.accept()
204
205     def update_gui(self) -> None:
206         """Update the GUI according to other widgets in the GUI.

```

```

205
206     For example, this method updates which checkboxes are enabled based on the values of other checkboxes.
207     """
208     self.checkbox_draw_determinant_text.setEnabled(self.checkbox_draw_determinant_parallelogram.isChecked())
209
210     def keyPressEvent(self, event: QKeyEvent) -> None:
211         """Handle a :class:`QKeyEvent` by manually activating toggling checkboxes.
212
213         Qt handles these shortcuts automatically and allows the user to do ``Alt + Key``
214         to activate a simple shortcut defined with ``&``. However, I like to be able to
215         just hit ``Key`` and have the shortcut activate.
216         """
217         letter = event.text().lower()
218         key = event.key()
219
220         if letter in self.dict_checkboxes:
221             self.dict_checkboxes[letter].animateClick()
222
223         # Return or keypad enter
224         elif key == 0x01000004 or key == 0x01000005:
225             self.button_confirm.click()
226
227         # Escape
228         elif key == 0x01000000:
229             self.button_cancel.click()
230
231         else:
232             event.ignore()

```

## A.12 gui/dialogs/define\_new\_matrix.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 an abstract :class:`DefinedDialog` class and subclasses, allowing definition of new
8  ↪ matrices."""
9
10 from __future__ import annotations
11
12 import abc
13
14 from numpy import array
15 from PyQt5 import QtWidgets
16 from PyQt5.QtCore import pyqtSlot
17 from PyQt5.QtGui import QDoubleValidator, QKeySequence
18 from PyQt5.QtWidgets import QDialog, QGridLayout, QHBoxLayout, QShortcut, QSizePolicy, QSpacerItem, QVBoxLayout
19
20 from lintrans.gui.plots import DefineVisuallyWidget
21 from lintrans.gui.validate import MatrixExpressionValidator
22 from lintrans.matrices import MatrixWrapper
23 from lintrans.typing_ import MatrixType
24
25 ALPHABET_NO_I = 'ABCDEFGHJKLMNOPQRSTUVWXYZ'
26
27 def is_valid_float(string: str) -> bool:
28     """Check if the string is a valid float (or anything that can be cast to a float, such as an int).
29
30     This function simply checks that ``float(string)`` doesn't raise an error.
31
32     .. note:: An empty string is not a valid float, so will return False.
33
34     :param str string: The string to check
35     :returns bool: Whether the string is a valid float
36     """
37     try:
38         float(string)

```



```

39         return True
40     except ValueError:
41         return False
42
43
44 def round_float(num: float, precision: int = 5) -> str:
45     """Round a floating point number to a given number of decimal places for pretty printing.
46
47     :param float num: The number to round
48     :param int precision: The number of decimal places to round to
49     :returns str: The rounded number for pretty printing
50     """
51     # Round to ``precision`` number of decimal places
52     string = str(round(num, precision))
53
54     # Cut off the potential final zero
55     if string.endswith('.0'):
56         return string[:-2]
57
58     elif 'e' in string: # Scientific notation
59         split = string.split('e')
60         # The leading 0 only happens when the exponent is negative, so we know there'll be a minus sign
61         return split[0] + 'e-' + split[1][1:].rstrip('0')
62
63     else:
64         return string
65
66
67 class DefineDialog(QDialog):
68     """An abstract superclass for definitions dialogs.
69
70     .. warning:: This class should never be directly instantiated, only subclassed.
71
72     .. note::
73         I would make this class have ``metaclass=abc.ABCMeta``, but I can't because it subclasses :class:`QDialog`,
74         and a every superclass of a class must have the same metaclass, and :class:`QDialog` is not an abstract
75     ↩ class.
76     """
77
78     def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
79         """Create the widgets and layout of the dialog.
80
81         .. note:: ``*args`` and ``**kwargs`` are passed to the super constructor (:class:`QDialog`).
82
83         :param MatrixWrapper matrix_wrapper: The MatrixWrapper that this dialog will mutate
84         """
85         super().__init__(*args, **kwargs)
86
87         self.matrix_wrapper = matrix_wrapper
88         self.setWindowTitle('Define a matrix')
89
90         # === Create the widgets
91
92         self.button_confirm = QtWidgets.QPushButton(self)
93         self.button_confirm.setText('Confirm')
94         self.button_confirm.setEnabled(False)
95         self.button_confirm.clicked.connect(self.confirm_matrix)
96         self.button_confirm.setToolTip('Confirm this as the new matrix<br><b>(Ctrl + Enter)</b>')
97         QShortcut(QKeySequence('Ctrl+Return'), self).activated.connect(self.button_confirm.click)
98
99         self.button_cancel = QtWidgets.QPushButton(self)
100         self.button_cancel.setText('Cancel')
101         self.button_cancel.clicked.connect(self.reject)
102         self.button_cancel.setToolTip('Cancel this definition<br><b>(Escape)</b>')
103
104         self.label_equals = QtWidgets.QLabel()
105         self.label_equals.setText('=')
106
107         self.combobox_letter = QtWidgets.QComboBox(self)
108
109         for letter in ALPHABET_NO_I:
110             self.combobox_letter.addItem(letter)

```

```

111         self.combobox_letter.activated.connect(self.load_matrix)
112
113         # === Arrange the widgets
114
115         self.setContentsMargins(10, 10, 10, 10)
116
117         self.hlay_buttons = QHBoxLayout()
118         self.hlay_buttons.setSpacing(20)
119         self.hlay_buttons.addItem(QSpacerItem(50, 5, hPolicy=QSizePolicy.Expanding, vPolicy=QSizePolicy.Minimum))
120         self.hlay_buttons.addWidget(self.button_cancel)
121         self.hlay_buttons.addWidget(self.button_confirm)
122
123         self.hlay_definition = QHBoxLayout()
124         self.hlay_definition.setSpacing(20)
125         self.hlay_definition.addWidget(self.combobox_letter)
126         self.hlay_definition.addWidget(self.label_equals)
127
128         self.vlay_all = QVBoxLayout()
129         self.vlay_all.setSpacing(20)
130
131         self.setLayout(self.vlay_all)
132
133     @property
134     def selected_letter(self) -> str:
135         """Return the letter currently selected in the combo box."""
136         return str(self.combobox_letter.currentText())
137
138     @abc.abstractmethod
139     @pyqtSlot()
140     def update_confirm_button(self) -> None:
141         """Enable the confirm button if it should be enabled, else, disable it."""
142
143     @pyqtSlot(int)
144     def load_matrix(self, index: int) -> None:
145         """Load the selected matrix into the dialog.
146
147         This method is optionally able to be overridden. If it is not overridden,
148         then no matrix is loaded when selecting a name.
149
150         We have this method in the superclass so that we can define it as the slot
151         for the :meth:`QComboBox.activated` signal in this constructor, rather than
152         having to define that in the constructor of every subclass.
153         """
154
155     @abc.abstractmethod
156     @pyqtSlot()
157     def confirm_matrix(self) -> None:
158         """Confirm the inputted matrix and assign it.
159
160         .. note:: When subclassing, this method should mutate ``self.matrix_wrapper`` and then call
161         ↪ ``self.accept()``.
162         """
163
164     class DefineVisuallyDialog(DefineDialog):
165         """The dialog class that allows the user to define a matrix visually."""
166
167         def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
168             """Create the widgets and layout of the dialog.
169
170             :param MatrixWrapper matrix_wrapper: The MatrixWrapper that this dialog will mutate
171             """
172             super().__init__(matrix_wrapper, *args, **kwargs)
173
174             self.setMinimumSize(700, 550)
175
176             # === Create the widgets
177
178             self.plot = DefineVisuallyWidget(self)
179
180             # === Arrange the widgets
181
182             self.hlay_definition.addWidget(self.plot)

```

```

183         self.hlay_definition.setStretchFactor(self.plot, 1)
184
185         self.vlay_all.addLayout(self.hlay_definition)
186         self.vlay_all.addLayout(self.hlay_buttons)
187
188         # We load the default matrix A into the plot
189         self.load_matrix(0)
190
191         # We also enable the confirm button, because any visually defined matrix is valid
192         self.button_confirm.setEnabled(True)
193
194     @pyqtSlot()
195     def update_confirm_button(self) -> None:
196         """Enable the confirm button.
197
198         .. note::
199             The confirm button is always enabled in this dialog and this method is never actually used,
200             so it's got an empty body. It's only here because we need to implement the abstract method.
201         """
202
203     @pyqtSlot(int)
204     def load_matrix(self, index: int) -> None:
205         """Show the selected matrix on the plot. If the matrix is None, show the identity."""
206         matrix = self.matrix_wrapper[self.selected_letter]
207
208         if matrix is None:
209             matrix = self.matrix_wrapper['I']
210
211         self.plot.visualize_matrix_transformation(matrix)
212         self.plot.update()
213
214     @pyqtSlot()
215     def confirm_matrix(self) -> None:
216         """Confirm the matrix that's been defined visually."""
217         matrix: MatrixType = array([
218             [self.plot.point_i[0], self.plot.point_j[0]],
219             [self.plot.point_i[1], self.plot.point_j[1]]
220         ])
221
222         self.matrix_wrapper[self.selected_letter] = matrix
223         self.accept()
224
225
226 class DefineNumericallyDialog(DefinedDialog):
227     """The dialog class that allows the user to define a new matrix numerically."""
228
229     def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
230         """Create the widgets and layout of the dialog.
231
232         :param MatrixWrapper matrix_wrapper: The MatrixWrapper that this dialog will mutate
233         """
234         super().__init__(matrix_wrapper, *args, **kwargs)
235
236         # === Create the widgets
237
238         # tl = top left, br = bottom right, etc.
239         self.element_tl = QtWidgets.QLineEdit(self)
240         self.element_tl.textChanged.connect(self.update_confirm_button)
241         self.element_tl.setValidator(QDoubleValidator())
242
243         self.element_tr = QtWidgets.QLineEdit(self)
244         self.element_tr.textChanged.connect(self.update_confirm_button)
245         self.element_tr.setValidator(QDoubleValidator())
246
247         self.element_bl = QtWidgets.QLineEdit(self)
248         self.element_bl.textChanged.connect(self.update_confirm_button)
249         self.element_bl.setValidator(QDoubleValidator())
250
251         self.element_br = QtWidgets.QLineEdit(self)
252         self.element_br.textChanged.connect(self.update_confirm_button)
253         self.element_br.setValidator(QDoubleValidator())
254
255         self.matrix_elements = (self.element_tl, self.element_tr, self.element_bl, self.element_br)

```

```

256
257     # == Arrange the widgets
258
259     self.grid_matrix = QGridLayout()
260     self.grid_matrix.setSpacing(20)
261     self.grid_matrix.addWidget(self.element_tl, 0, 0)
262     self.grid_matrix.addWidget(self.element_tr, 0, 1)
263     self.grid_matrix.addWidget(self.element_bl, 1, 0)
264     self.grid_matrix.addWidget(self.element_br, 1, 1)
265
266     self.hlay_definition.addLayout(self.grid_matrix)
267
268     self.vlay_all.addLayout(self.hlay_definition)
269     self.vlay_all.addLayout(self.hlay_buttons)
270
271     # We load the default matrix A into the boxes
272     self.load_matrix(0)
273
274     self.element_tl.setFocus()
275
276 @pyqtSlot()
277 def update_confirm_button(self) -> None:
278     """Enable the confirm button if there are valid floats in every box."""
279     for elem in self.matrix_elements:
280         if not is_valid_float(elem.text()):
281             # If they're not all numbers, then we can't confirm it
282             self.button_confirm.setEnabled(False)
283             return
284
285     # If we didn't find anything invalid
286     self.button_confirm.setEnabled(True)
287
288 @pyqtSlot(int)
289 def load_matrix(self, index: int) -> None:
290     """If the selected matrix is defined, load its values into the boxes."""
291     matrix = self.matrix_wrapper[self.selected_letter]
292
293     if matrix is None:
294         for elem in self.matrix_elements:
295             elem.setText('')
296
297     else:
298         self.element_tl.setText(round_float(matrix[0][0]))
299         self.element_tr.setText(round_float(matrix[0][1]))
300         self.element_bl.setText(round_float(matrix[1][0]))
301         self.element_br.setText(round_float(matrix[1][1]))
302
303     self.update_confirm_button()
304
305 @pyqtSlot()
306 def confirm_matrix(self) -> None:
307     """Confirm the matrix in the boxes and assign it to the name in the combo box."""
308     matrix: MatrixType = array([
309         [float(self.element_tl.text()), float(self.element_tr.text())],
310         [float(self.element_bl.text()), float(self.element_br.text())]
311     ])
312
313     self.matrix_wrapper[self.selected_letter] = matrix
314     self.accept()
315
316
317 class DefineAsAnExpressionDialog(Definedialog):
318     """The dialog class that allows the user to define a matrix as an expression of other matrices."""
319
320     def __init__(self, matrix_wrapper: MatrixWrapper, *args, **kwargs):
321         """Create the widgets and layout of the dialog.
322
323         :param MatrixWrapper matrix_wrapper: The MatrixWrapper that this dialog will mutate
324         """
325         super().__init__(matrix_wrapper, *args, **kwargs)
326
327         self.setMinimumWidth(450)
328

```

```

329         # === Create the widgets
330
331         self.lineedit_expression_box = QtWidgets.QLineEdit(self)
332         self.lineedit_expression_box.setPlaceholderText('Enter matrix expression...')
333         self.lineedit_expression_box.textChanged.connect(self.update_confirm_button)
334         self.lineedit_expression_box.setValidator(MatrixExpressionValidator())
335
336         # === Arrange the widgets
337
338         self.hlay_definition.addWidget(self.lineedit_expression_box)
339
340         self.vlay_all.addLayout(self.hlay_definition)
341         self.vlay_all.addLayout(self.hlay_buttons)
342
343         # Load the matrix if it's defined as an expression
344         self.load_matrix(0)
345
346         self.lineedit_expression_box.setFocus()
347
348     @pyqtSlot()
349     def update_confirm_button(self) -> None:
350         """Enable the confirm button if the matrix expression is valid in the wrapper."""
351         text = self.lineedit_expression_box.text()
352         valid_expression = self.matrix_wrapper.is_valid_expression(text)
353
354         self.button_confirm.setEnabled(valid_expression and self.selected_letter not in text)
355
356     @pyqtSlot(int)
357     def load_matrix(self, index: int) -> None:
358         """If the selected matrix is defined an expression, load that expression into the box."""
359         if (expr := self.matrix_wrapper.get_expression(self.selected_letter)) is not None:
360             self.lineedit_expression_box.setText(expr)
361         else:
362             self.lineedit_expression_box.setText('')
363
364     @pyqtSlot()
365     def confirm_matrix(self) -> None:
366         """Evaluate the matrix expression and assign its value to the name in the combo box."""
367         self.matrix_wrapper[self.selected_letter] = self.lineedit_expression_box.text()
368         self.accept()

```

### A.13 gui/dialogs/\_\_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 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,
13 ↪ DefineVisuallyDialog
14 from .settings import DisplaySettingsDialog
15
16 __all__ = ['DefineAsAnExpressionDialog', 'DefineDialog', 'DefineNumericallyDialog',
17            'DefineVisuallyDialog', 'DisplaySettingsDialog']

```

### A.14 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 PyQt5.QtCore import Qt
12 from PyQt5.QtGui import QMouseEvent, QPainter, QPaintEvent
13
14 from .classes import VectorGridPlot
15 from lintrans.typing_ import MatrixType
16 from lintrans.gui.settings import DisplaySettings
17
18
19 class VisualizeTransformationWidget(VectorGridPlot):
20     """This class is the widget that is used in the main window to visualize transformations.
21
22     It handles all the rendering itself, and the only method that the user needs to
23     worry about is :meth:`visualize_matrix_transformation`, which allows you to visualise
24     the given matrix transformation.
25     """
26
27     def __init__(self, display_settings: DisplaySettings, *args, **kwargs):
28         """Create the widget and assign its display settings, passing ``*args`` and ``**kwargs`` to super."""
29         super().__init__(*args, **kwargs)
30
31         self.display_settings = display_settings
32
33     def visualize_matrix_transformation(self, matrix: MatrixType) -> None:
34         """Transform the grid by the given matrix.
35
36         .. warning:: This method does not call ``update()``. This must be done by the caller.
37
38         .. note::
39             This method transforms the background grid, not the basis vectors. This
40             means that it cannot be used to compose transformations. Compositions
41             should be done by the user.
42
43         :param MatrixType matrix: The matrix to transform by
44         """
45         self.point_i = (matrix[0][0], matrix[1][0])
46         self.point_j = (matrix[0][1], matrix[1][1])
47
48     def paintEvent(self, event: QPaintEvent) -> None:
49         """Handle a :class:`QPaintEvent` by drawing the background grid and the transformed grid.
50
51         The transformed grid is defined by the basis vectors i and j, which can
52         be controlled with the :meth:`visualize_matrix_transformation` method.
53         """
54         painter = QPainter()
55         painter.begin(self)
56
57         painter.setRenderHint(QPainter.Antialiasing)
58         painter.setBrush(Qt.NoBrush)
59
60         self.draw_background(painter)
61         self.draw_transformed_grid(painter)
62         self.draw_basis_vectors(painter)
63
64         if self.display_settings.draw_eigenlines:
65             self.draw_eigenlines(painter)
66
67         if self.display_settings.draw_eigenvectors:
68             self.draw_eigenvectors(painter)
69
70         if self.display_settings.draw_determinant_parallelogram:
71             self.draw_determinant_parallelogram(painter)
72
73             if self.display_settings.draw_determinant_text:
74                 self.draw_determinant_text(painter)
75
76         painter.end()
77         event.accept()
78

```

```

79
80 class DefineVisuallyWidget(VisualizeTransformationWidget):
81     """This class is the widget that allows the user to visually define a matrix.
82
83     This is just the widget itself. If you want the dialog, use
84     :class:`lintrans.gui.dialogs.define_new_matrix.DefineVisuallyDialog`.
85     """
86
87     def __init__(self, *args, **kwargs):
88         """Create the widget and enable mouse tracking. ``*args`` and ``**kwargs`` are passed to ``super()``."""
89         super().__init__(*args, **kwargs)
90
91         self.dragged_point: tuple[float, float] | None = None
92
93         # This is the distance that the cursor needs to be from the point to drag it
94         self.epsilon: int = 5
95
96     def paintEvent(self, event: QPaintEvent) -> None:
97         """Handle a :class:`QPaintEvent` by drawing the background grid and the transformed grid.
98
99         The transformed grid is defined by the basis vectors  $i$  and  $j$ ,
100         which can be dragged around in the widget.
101         """
102         painter = QPainter()
103         painter.begin(self)
104
105         painter.setRenderHint(QPainter.Antialiasing)
106         painter.setBrush(Qt.NoBrush)
107
108         self.draw_background(painter)
109         self.draw_transformed_grid(painter)
110         self.draw_basis_vectors(painter)
111
112         painter.end()
113         event.accept()
114
115     def mousePressEvent(self, event: QMouseEvent) -> None:
116         """Handle a QMouseEvent when the user pressed a button."""
117         mx = event.x()
118         my = event.y()
119         button = event.button()
120
121         if button != Qt.LeftButton:
122             event.ignore()
123             return
124
125         for point in (self.point_i, self.point_j):
126             px, py = self.canvas_coords(*point)
127             if abs(px - mx) <= self.epsilon and abs(py - my) <= self.epsilon:
128                 self.dragged_point = point[0], point[1]
129
130         event.accept()
131
132     def mouseReleaseEvent(self, event: QMouseEvent) -> None:
133         """Handle a QMouseEvent when the user release a button."""
134         if event.button() == Qt.LeftButton:
135             self.dragged_point = None
136             event.accept()
137         else:
138             event.ignore()
139
140     def mouseMoveEvent(self, event: QMouseEvent) -> None:
141         """Handle the mouse moving on the canvas."""
142         mx = event.x()
143         my = event.y()
144
145         if self.dragged_point is not None:
146             x, y = self.grid_coords(mx, my)
147
148             if self.dragged_point == self.point_i:
149                 self.point_i = x, y
150
151             elif self.dragged_point == self.point_j:

```

```

152         self.point_j = x, y
153
154         self.dragged_point = x, y
155
156         self.update()
157
158         event.accept()
159
160         event.ignore()

```

## A.15 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
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 a every superclass of a class must have the same metaclass, and :class:`QWidget` is not an abstract
36      ↩ class.
37      """
38
39      default_grid_spacing: int = 85
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     self.grid_spacing = BackgroundPlot.default_grid_spacing
59     self.width_background_grid: float = 0.3
60
61     @property
62     def canvas_origin(self) -> tuple[int, int]:
63         """Return the canvas coords of the grid origin.
64
65         The return value is intended to be unpacked and passed to a :meth:`QPainter.drawLine:iiii` call.
66
67         See :meth:`canvas_coords`.
68
69         :returns: The canvas coordinates of the grid origin
70         :rtype: tuple[int, int]
71         """
72         return self.width() // 2, self.height() // 2
73
74     def canvas_x(self, x: float) -> int:
75         """Convert an x coordinate from grid coords to canvas coords."""
76         return int(self.canvas_origin[0] + x * self.grid_spacing)
77
78     def canvas_y(self, y: float) -> int:
79         """Convert a y coordinate from grid coords to canvas coords."""
80         return int(self.canvas_origin[1] - y * self.grid_spacing)
81
82     def canvas_coords(self, x: float, y: float) -> tuple[int, int]:
83         """Convert a coordinate from grid coords to canvas coords.
84
85         This method is intended to be used like
86
87         .. code::
88
89             painter.drawLine(*self.canvas_coords(x1, y1), *self.canvas_coords(x2, y2))
90
91         or like
92
93         .. code::
94
95             painter.drawLine(*self.canvas_origin, *self.canvas_coords(x, y))
96
97         See :attr:`canvas_origin`.
98
99         :param float x: The x component of the grid coordinate
100        :param float y: The y component of the grid coordinate
101        :returns: The resultant canvas coordinates
102        :rtype: tuple[int, int]
103        """
104        return self.canvas_x(x), self.canvas_y(y)
105
106     def grid_corner(self) -> tuple[float, float]:
107         """Return the grid coords of the top right corner."""
108         return self.width() / (2 * self.grid_spacing), self.height() / (2 * self.grid_spacing)
109
110     def grid_coords(self, x: int, y: int) -> tuple[float, float]:
111         """Convert a coordinate from canvas coords to grid coords.
112
113         :param int x: The x component of the canvas coordinate
114         :param int y: The y component of the canvas coordinate
115         :returns: The resultant grid coordinates
116         :rtype: tuple[float, float]
117         """
118         # We get the maximum grid coords and convert them into canvas coords
119         return (x - self.canvas_origin[0]) / self.grid_spacing, (-y + self.canvas_origin[1]) / self.grid_spacing
120
121     @abstractmethod
122     def paintEvent(self, event: QPaintEvent) -> None:
123         """Handle a :class:`QPaintEvent`.
124
125         .. note:: This method is abstract and must be overridden by all subclasses.
126         """
127
128     def draw_background(self, painter: QPainter) -> None:
129         """Draw the background grid.
130

```

```

131     .. note:: This method is just a utility method for subclasses to use to render the background grid.
132
133     :param QPainter painter: The painter to draw the background with
134     """
135     # Draw the grid
136     painter.setPen(QPen(self.colour_background_grid, self.width_background_grid))
137
138     # We draw the background grid, centered in the middle
139     # We deliberately exclude the axes - these are drawn separately
140     for x in range(self.width() // 2 + self.grid_spacing, self.width(), self.grid_spacing):
141         painter.drawLine(x, 0, x, self.height())
142         painter.drawLine(self.width() - x, 0, self.width() - x, self.height())
143
144     for y in range(self.height() // 2 + self.grid_spacing, self.height(), self.grid_spacing):
145         painter.drawLine(0, y, self.width(), y)
146         painter.drawLine(0, self.height() - y, self.width(), self.height() - y)
147
148     # Now draw the axes
149     painter.setPen(QPen(self.colour_background_axes, self.width_background_grid))
150     painter.drawLine(self.width() // 2, 0, self.width() // 2, self.height())
151     painter.drawLine(0, self.height() // 2, self.width(), self.height() // 2)
152
153     def wheelEvent(self, event: QWheelEvent) -> None:
154         """Handle a :class:`QWheelEvent` by zooming in or out of the grid."""
155         # angleDelta() returns a number of units equal to 8 times the number of degrees rotated
156         degrees = event.angleDelta() / 8
157
158         if degrees is not None:
159             self.grid_spacing = max(1, self.grid_spacing + degrees.y())
160
161         event.accept()
162         self.update()
163
164
165     class VectorGridPlot(BackgroundPlot):
166         """This class represents a background plot, with vectors and their grid drawn on top.
167
168         This class should be subclassed to be used for visualization and matrix definition widgets.
169         All useful behaviour should be implemented by any subclass.
170
171         .. warning:: This class should never be directly instantiated, only subclassed.
172         """
173
174         def __init__(self, *args, **kwargs):
175             """Create the widget with ``point_i`` and ``point_j`` attributes.
176
177             .. note:: ``*args`` and ``**kwargs`` are passed to the superclass constructor (:class:`BackgroundPlot`).
178             """
179             super().__init__(*args, **kwargs)
180
181             self.point_i: tuple[float, float] = (1., 0.)
182             self.point_j: tuple[float, float] = (0., 1.)
183
184             self.colour_i = QColor('#0808d8')
185             self.colour_j = QColor('#e90000')
186             self.colour_eigen = QColor('#13cf00')
187             self.colour_text = QColor('#000000')
188
189             self.width_vector_line = 1.8
190             self.width_transformed_grid = 0.8
191
192             self.arrowhead_length = 0.15
193
194             self.max_parallel_lines = 150
195
196         @property
197         def matrix(self) -> MatrixType:
198             """Return the assembled matrix of the basis vectors."""
199             return np.array([
200                 [self.point_i[0], self.point_j[0]],
201                 [self.point_i[1], self.point_j[1]]
202             ])
203

```

```

204     @property
205     def det(self) -> float:
206         """Return the determinant of the assembled matrix."""
207         return float(np.linalg.det(self.matrix))
208
209     @property
210     def eigs(self) -> Iterable[tuple[float, NDArray[(1, 2), Float]]]:
211         """Return the eigenvalues and eigenvectors zipped together to be iterated over.
212
213         :rtype: Iterable[tuple[float, NDArray[(1, 2), Float]]]
214         """
215         values, vectors = np.linalg.eig(self.matrix)
216         return zip(values, vectors.T)
217
218     @abstractmethod
219     def paintEvent(self, event: QPaintEvent) -> None:
220         """Handle a :class:`QPaintEvent`.
221
222         .. note:: This method is abstract and must be overridden by all subclasses.
223         """
224
225     def draw_parallel_lines(self, painter: QPainter, vector: tuple[float, float], point: tuple[float, float]) ->
↪ None:
226         """Draw a set of evenly spaced grid lines parallel to ``vector`` intersecting ``point``.
227
228         :param QPainter painter: The painter to draw the lines with
229         :param vector: The vector to draw the grid lines parallel to
230         :type vector: tuple[float, float]
231         :param point: The point for the lines to intersect with
232         :type point: tuple[float, float]
233         """
234         max_x, max_y = self.grid_corner()
235         vector_x, vector_y = vector
236         point_x, point_y = point
237
238         # If the determinant is 0
239         if abs(vector_x * point_y - vector_y * point_x) < 1e-12:
240             rank = np.linalg.matrix_rank(
241                 np.array([
242                     [vector_x, point_x],
243                     [vector_y, point_y]
244                 ])
245             )
246
247             # If the matrix is rank 1, then we can draw the column space line
248             if rank == 1:
249                 if abs(vector_x) < 1e-12:
250                     painter.drawLine(self.width() // 2, 0, self.width() // 2, self.height())
251                 elif abs(vector_y) < 1e-12:
252                     painter.drawLine(0, self.height() // 2, self.width(), self.height() // 2)
253                 else:
254                     self.draw_oblique_line(painter, vector_y / vector_x, 0)
255
256             # If the rank is 0, then we don't draw any lines
257             else:
258                 return
259
260             elif abs(vector_x) < 1e-12 and abs(vector_y) < 1e-12:
261                 # If both components of the vector are practically 0, then we can't render any grid lines
262                 return
263
264             # Draw vertical lines
265             elif abs(vector_x) < 1e-12:
266                 painter.drawLine(self.canvas_x(0), 0, self.canvas_x(0), self.height())
267
268                 for i in range(max(abs(int(max_x / point_x)), self.max_parallel_lines)):
269                     painter.drawLine(
270                         self.canvas_x((i + 1) * point_x),
271                         0,
272                         self.canvas_x((i + 1) * point_x),
273                         self.height()
274                     )
275                 painter.drawLine(

```

```

276         self.canvas_x(-1 * (i + 1) * point_x),
277         0,
278         self.canvas_x(-1 * (i + 1) * point_x),
279         self.height()
280     )
281
282     # Draw horizontal lines
283     elif abs(vector_y) < 1e-12:
284         painter.drawLine(0, self.canvas_y(0), self.width(), self.canvas_y(0))
285
286         for i in range(max(abs(int(max_y / point_y)), self.max_parallel_lines)):
287             painter.drawLine(
288                 0,
289                 self.canvas_y((i + 1) * point_y),
290                 self.width(),
291                 self.canvas_y((i + 1) * point_y)
292             )
293             painter.drawLine(
294                 0,
295                 self.canvas_y(-1 * (i + 1) * point_y),
296                 self.width(),
297                 self.canvas_y(-1 * (i + 1) * point_y)
298             )
299
300     # If the line is oblique, then we can use  $y = mx + c$ 
301     else:
302         m = vector_y / vector_x
303         c = point_y - m * point_x
304
305         self.draw_oblique_line(painter, m, 0)
306
307         # We don't want to overshoot the max number of parallel lines,
308         # but we should also stop looping as soon as we can't draw any more lines
309         for i in range(1, self.max_parallel_lines + 1):
310             if not self.draw_pair_of_oblique_lines(painter, m, i * c):
311                 break
312
313     def draw_pair_of_oblique_lines(self, painter: QPainter, m: float, c: float) -> bool:
314         """Draw a pair of oblique lines, using the equation  $y = mx + c$ .
315
316         This method just calls :meth:`draw_oblique_line` with ``c`` and ``-c``,
317         and returns True if either call returned True.
318
319         :param QPainter painter: The painter to draw the vectors and grid lines with
320         :param float m: The gradient of the lines to draw
321         :param float c: The y-intercept of the lines to draw. We use the positive and negative versions
322         :returns bool: Whether we were able to draw any lines on the canvas
323         """
324         return any([
325             self.draw_oblique_line(painter, m, c),
326             self.draw_oblique_line(painter, m, -c)
327         ])
328
329     def draw_oblique_line(self, painter: QPainter, m: float, c: float) -> bool:
330         """Draw an oblique line, using the equation  $y = mx + c$ .
331
332         We only draw the part of the line that fits within the canvas, returning True if
333         we were able to draw a line within the boundaries, and False if we couldn't draw a line
334
335         :param QPainter painter: The painter to draw the vectors and grid lines with
336         :param float m: The gradient of the line to draw
337         :param float c: The y-intercept of the line to draw
338         :returns bool: Whether we were able to draw a line on the canvas
339         """
340         max_x, max_y = self.grid_corner()
341
342         # These variable names are shortened for convenience
343         # myi is max_y_intersection, mmyi is minus_max_y_intersection, etc.
344         myi = (max_y - c) / m
345         mmyi = (-max_y - c) / m
346         mx_i = max_x * m + c
347         mmxi = -max_x * m + c
348

```

```

349     # The inner list here is a list of coords, or None
350     # If an intersection fits within the bounds, then we keep its coord,
351     # else it is None, and then gets discarded from the points list
352     # By the end, points is a list of two coords, or an empty list
353     points: list[tuple[float, float]] = [
354         x for x in [
355             (myi, max_y) if -max_x < myi < max_x else None,
356             (mmyi, -max_y) if -max_x < mmyi < max_x else None,
357             (max_x, mxi) if -max_y < mxi < max_y else None,
358             (-max_x, mmxi) if -max_y < mmxi < max_y else None
359         ] if x is not None
360     ]
361
362     # If no intersections fit on the canvas
363     if len(points) < 2:
364         return False
365
366     # If we can, then draw the line
367     else:
368         painter.drawLine(
369             *self.canvas_coords(*points[0]),
370             *self.canvas_coords(*points[1])
371         )
372         return True
373
374     def draw_transformed_grid(self, painter: QPainter) -> None:
375         """Draw the transformed version of the grid, given by the basis vectors.
376
377         .. note:: This method draws the grid, but not the basis vectors. Use :meth:`draw_basis_vectors` to draw
378         ↪ them.
379
380         :param QPainter painter: The painter to draw the grid lines with
381         """
382         # Draw all the parallel lines
383         painter.setPen(QPen(self.colour_i, self.width_transformed_grid))
384         self.draw_parallel_lines(painter, self.point_i, self.point_j)
385         painter.setPen(QPen(self.colour_j, self.width_transformed_grid))
386         self.draw_parallel_lines(painter, self.point_j, self.point_i)
387
388     def draw_arrowhead_away_from_origin(self, painter: QPainter, point: tuple[float, float]) -> None:
389         """Draw an arrowhead at ``point``, pointing away from the origin.
390
391         :param QPainter painter: The painter to draw the arrowhead with
392         :param point: The point to draw the arrowhead at, given in grid coords
393         :type point: tuple[float, float]
394         """
395         # This algorithm was adapted from a C# algorithm found at
396         # http://csharpshelper.com/blog/2014/12/draw-lines-with-arrowheads-in-c/
397
398         # Get the x and y coords of the point, and then normalize them
399         # We have to normalize them, or else the size of the arrowhead will
400         # scale with the distance of the point from the origin
401         x, y = point
402         vector_length = np.sqrt(x * x + y * y)
403
404         if vector_length < 1e-12:
405             return
406
407         nx = x / vector_length
408         ny = y / vector_length
409
410         # We choose a length and find the steps in the x and y directions
411         length = min(
412             self.arrowhead_length * self.default_grid_spacing / self.grid_spacing,
413             vector_length
414         )
415         dx = length * (-nx - ny)
416         dy = length * (nx - ny)
417
418         # Then we just plot those lines
419         painter.drawLine(*self.canvas_coords(x, y), *self.canvas_coords(x + dx, y + dy))
420         painter.drawLine(*self.canvas_coords(x, y), *self.canvas_coords(x - dx, y + dx))

```

```

421 def draw_position_vector(self, painter: QPainter, point: tuple[float, float], colour: QColor) -> None:
422     """Draw a vector from the origin to the given point.
423
424     :param QPainter painter: The painter to draw the position vector with
425     :param point: The tip of the position vector in grid coords
426     :type point: tuple[float, float]
427     :param QColor colour: The colour to draw the position vector in
428     """
429     painter.setPen(QPen(colour, self.width_vector_line))
430     painter.drawLine(*self.canvas_origin, *self.canvas_coords(*point))
431     self.draw_arrowhead_away_from_origin(painter, point)
432
433 def draw_basis_vectors(self, painter: QPainter) -> None:
434     """Draw arrowheads at the tips of the basis vectors.
435
436     :param QPainter painter: The painter to draw the basis vectors with
437     """
438     self.draw_position_vector(painter, self.point_i, self.colour_i)
439     self.draw_position_vector(painter, self.point_j, self.colour_j)
440
441 def draw_determinant_parallelogram(self, painter: QPainter) -> None:
442     """Draw the parallelogram of the determinant of the matrix.
443
444     :param QPainter painter: The painter to draw the parallelogram with
445     """
446     if self.det == 0:
447         return
448
449     path = QPainterPath()
450     path.moveTo(*self.canvas_origin)
451     path.lineTo(*self.canvas_coords(*self.point_i))
452     path.lineTo(*self.canvas_coords(self.point_i[0] + self.point_j[0], self.point_i[1] + self.point_j[1]))
453     path.lineTo(*self.canvas_coords(*self.point_j))
454
455     color = (16, 235, 253) if self.det > 0 else (253, 34, 16)
456     brush = QBrush(QColor(*color, alpha=128), Qt.SolidPattern)
457
458     painter.fillPath(path, brush)
459
460 def draw_determinant_text(self, painter: QPainter) -> None:
461     """Write the string value of the determinant in the middle of the parallelogram.
462
463     :param QPainter painter: The painter to draw the determinant text with
464     """
465     painter.setPen(QPen(self.colour_text, self.width_vector_line))
466
467     # We're building a QRect that encloses the determinant parallelogram
468     # Then we can center the text in this QRect
469     coords: list[tuple[float, float]] = [
470         (0, 0),
471         self.point_i,
472         self.point_j,
473         (
474             self.point_i[0] + self.point_j[0],
475             self.point_i[1] + self.point_j[1]
476         )
477     ]
478
479     xs = [t[0] for t in coords]
480     ys = [t[1] for t in coords]
481
482     top_left = QPoint(*self.canvas_coords(min(xs), max(ys)))
483     bottom_right = QPoint(*self.canvas_coords(max(xs), min(ys)))
484
485     rect = QRectF(top_left, bottom_right)
486
487     painter.drawText(
488         rect,
489         Qt.AlignHCenter | Qt.AlignVCenter,
490         f'{self.det:.2f}'
491     )
492
493 def draw_eigenvectors(self, painter: QPainter) -> None:

```

```

494         """Draw the eigenvectors of the displayed matrix transformation.
495
496         :param QPainter painter: The painter to draw the eigenvectors with
497         """
498         for value, vector in self.eigs:
499             x = value * vector[0]
500             y = value * vector[1]
501
502             if x.imag != 0 or y.imag != 0:
503                 continue
504
505             self.draw_position_vector(painter, (x, y), self.colour_eigen)
506
507             # Now we need to draw the eigenvalue at the tip of the eigenvector
508
509             offset = 3
510             top_left: QPoint
511             bottom_right: QPoint
512             alignment_flags: int
513
514             if x >= 0 and y >= 0: # Q1
515                 top_left = QPoint(self.canvas_x(x) + offset, 0)
516                 bottom_right = QPoint(self.width(), self.canvas_y(y) - offset)
517                 alignment_flags = Qt.AlignLeft | Qt.AlignBottom
518
519             elif x < 0 and y >= 0: # Q2
520                 top_left = QPoint(0, 0)
521                 bottom_right = QPoint(self.canvas_x(x) - offset, self.canvas_y(y) - offset)
522                 alignment_flags = Qt.AlignRight | Qt.AlignBottom
523
524             elif x < 0 and y < 0: # Q3
525                 top_left = QPoint(0, self.canvas_y(y) + offset)
526                 bottom_right = QPoint(self.canvas_x(x) - offset, self.height())
527                 alignment_flags = Qt.AlignRight | Qt.AlignTop
528
529             else: # Q4
530                 top_left = QPoint(self.canvas_x(x) + offset, self.canvas_y(y) + offset)
531                 bottom_right = QPoint(self.width(), self.height())
532                 alignment_flags = Qt.AlignLeft | Qt.AlignTop
533
534             painter.setPen(QPen(self.colour_text, self.width_vector_line))
535             painter.drawText(QRectF(top_left, bottom_right), alignment_flags, f'{value:.2f}')
536
537     def draw_eigenlines(self, painter: QPainter) -> None:
538         """Draw the eigenlines (invariant lines).
539
540         :param QPainter painter: The painter to draw the eigenlines with
541         """
542         painter.setPen(QPen(self.colour_eigen, self.width_transformed_grid))
543
544         for value, vector in self.eigs:
545             if value.imag != 0:
546                 continue
547
548             x, y = vector
549
550             if x == 0:
551                 x_mid = int(self.width() / 2)
552                 painter.drawLine(x_mid, 0, x_mid, self.height())
553
554             elif y == 0:
555                 y_mid = int(self.height() / 2)
556                 painter.drawLine(0, y_mid, self.width(), y_mid)
557
558             else:
559                 self.draw_oblique_line(painter, y / x, 0)

```

## A.16 gui/plots/\_\_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 provides widgets for the visualization plot in the main window and the visual definition dialog."""
8
9  from . import classes
10 from .widgets import DefineVisuallyWidget, VisualizeTransformationWidget
11
12 __all__ = ['classes', 'DefineVisuallyWidget', 'VisualizeTransformationWidget']

```

## B Testing code

### B.1 matrices/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  import numpy as np
10 import pytest
11
12 from lintrans.matrices import create_rotation_matrix
13 from lintrans.typing_ import MatrixType
14
15 angles_and_matrices: list[tuple[float, float, MatrixType]] = [
16     (0, 0, np.array([[1, 0], [0, 1]])),
17     (90, np.pi / 2, np.array([[0, -1], [1, 0]])),
18     (180, np.pi, np.array([[ -1, 0], [0, -1]])),
19     (270, 3 * np.pi / 2, np.array([[0, 1], [-1, 0]])),
20     (360, 2 * np.pi, np.array([[1, 0], [0, 1]])),
21
22     (45, np.pi / 4, np.array([
23         [np.sqrt(2) / 2, -1 * np.sqrt(2) / 2],
24         [np.sqrt(2) / 2, np.sqrt(2) / 2]
25     ])),
26     (135, 3 * np.pi / 4, np.array([
27         [-1 * np.sqrt(2) / 2, -1 * np.sqrt(2) / 2],
28         [np.sqrt(2) / 2, -1 * np.sqrt(2) / 2]
29     ])),
30     (225, 5 * np.pi / 4, np.array([
31         [-1 * np.sqrt(2) / 2, np.sqrt(2) / 2],
32         [-1 * np.sqrt(2) / 2, -1 * np.sqrt(2) / 2]
33     ])),
34     (315, 7 * np.pi / 4, np.array([
35         [np.sqrt(2) / 2, np.sqrt(2) / 2],
36         [-1 * np.sqrt(2) / 2, np.sqrt(2) / 2]
37     ])),
38
39     (30, np.pi / 6, np.array([
40         [np.sqrt(3) / 2, -1 / 2],
41         [1 / 2, np.sqrt(3) / 2]
42     ])),
43     (60, np.pi / 3, np.array([
44         [1 / 2, -1 * np.sqrt(3) / 2],
45         [np.sqrt(3) / 2, 1 / 2]
46     ])),
47     (120, 2 * np.pi / 3, np.array([
48         [-1 / 2, -1 * np.sqrt(3) / 2],

```



```

49     [np.sqrt(3) / 2, -1 / 2]
50     ])),
51     (150, 5 * np.pi / 6, np.array([
52         [-1 * np.sqrt(3) / 2, -1 / 2],
53         [1 / 2, -1 * np.sqrt(3) / 2]
54     ])),
55     (210, 7 * np.pi / 6, np.array([
56         [-1 * np.sqrt(3) / 2, 1 / 2],
57         [-1 / 2, -1 * np.sqrt(3) / 2]
58     ])),
59     (240, 4 * np.pi / 3, np.array([
60         [-1 / 2, np.sqrt(3) / 2],
61         [-1 * np.sqrt(3) / 2, -1 / 2]
62     ])),
63     (300, 10 * np.pi / 6, np.array([
64         [1 / 2, np.sqrt(3) / 2],
65         [-1 * np.sqrt(3) / 2, 1 / 2]
66     ])),
67     (330, 11 * np.pi / 6, np.array([
68         [np.sqrt(3) / 2, 1 / 2],
69         [-1 / 2, np.sqrt(3) / 2]
70     ]))
71 ]
72
73
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))

```

## B.2 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 matrices.parse module validation and parsing."""
8
9 import pytest
10
11 from lintrans.matrices.parse import MatrixParseError, parse_matrix_expression, validate_matrix_expression
12 from lintrans.typing_ import MatrixParseList
13
14 valid_inputs: list[str] = [
15     'A', 'AB', '3A', '1.2A', '-3.4A', 'A^2', 'A^-1', 'A^{-1}',
16     'A^12', 'A^T', 'A^{5}', 'A^{T}', '4.3A^7', '9.2A^{18}', '.1A'
17
18     'rot(45)', 'rot(12.5)', '3rot(90)',
19     'rot(135)^3', 'rot(51)^T', 'rot(-34)^-1',
20
21     'A+B', 'A+2B', '4.3A+9B', 'A^2+B^T', '3A^7+0.8B^{16}',
22     'A-B', '3A-4B', '3.2A^3-16.79B^T', '4.752A^{17}-3.32B^{36}',
23     'A-1B', '-A', '-1A'
24
25     '3A4B', 'A^TB', 'A^{T}B', '4A^6B^3',
26     '2A^{3}4B^5', '4rot(90)^3', 'rot(45)rot(13)',
27     'Arot(90)', 'AB^2', 'A^2B^2', '8.36A^T3.4B^12',
28
29     '3.5A^{4}5.6rot(19.2)^T-B^{-1}4.1C^5'
30 ]
31
32 invalid_inputs: list[str] = [
33     '', 'rot()', 'A^', 'A^{3.4}', 'A^{3.4}', '1,2A', 'ro(12)', '5', '12^2', '^T', '^12}',
34     'A^{13}', 'A^3}', 'A^A', '^2', 'A--B', '--A', '+A', '--1A', 'A--B', 'A--1B', '.A', '1.A'

```

```

35
36     'This is 100% a valid matrix expression, I swear'
37 ]
38
39
40 @pytest.mark.parametrize('inputs,output', [(valid_inputs, True), (invalid_inputs, False)])
41 def test_validate_matrix_expression(inputs: list[str], output: bool) -> None:
42     """Test the validate_matrix_expression() function."""
43     for inp in inputs:
44         assert validate_matrix_expression(inp) == output
45
46
47 expressions_and_parsed_expressions: list[tuple[str, MatrixParseList]] = [
48     # Simple expressions
49     ('A', [((' ', 'A', ' ')]]),
50     ('A^2', [((' ', 'A', '2')])),
51     ('A^{2}', [((' ', 'A', '2')])),
52     ('3A', [(('3', 'A', ' ')]]),
53     ('1.4A^3', [(('1.4', 'A', '3')])),
54     ('0.1A', [(('0.1', 'A', ' ')]]),
55     ('.1A', [((' ', 'A', ' ')]]),
56     ('A^{12}', [((' ', 'A', '12')])),
57     ('A^{234}', [((' ', 'A', '234')])),
58
59     # Multiplications
60     ('A .1B', [((' ', 'A', ' '), ('1', 'B', ' ')]]),
61     ('A^2 3B', [((' ', 'A', '23'), (' ', 'B', ' ')]]),
62     ('4A^{3} 6B^2', [(('4', 'A', '3'), ('6', 'B', '2')])),
63     ('4.2A^{T} 6.1B^{-1}', [(('4.2', 'A', 'T'), ('6.1', 'B', '-1')])),
64     ('-1.2A^2 rot(45)^2', [(('1.2', 'A', '2'), (' ', 'rot(45)', '2')])),
65     ('3.2A^T 4.5B^{5} 9.6rot(121.3)', [(('3.2', 'A', 'T'), ('4.5', 'B', '5'), ('9.6', 'rot(121.3)', ' ')]]),
66     ('-1.18A^{-2} 0.1B^{2} 9rot(-34.6)^{-1}', [(('1.18', 'A', '-2'), ('0.1', 'B', '2'), ('9', 'rot(-34.6)', '-1')])),
67
68     # Additions
69     ('A + B', [((' ', 'A', ' '), (' ', 'B', ' ')]]),
70     ('A + B - C', [((' ', 'A', ' '), (' ', 'B', ' '), ('-1', 'C', ' ')]]),
71     ('A^2 + .5B', [((' ', 'A', '2'), ('.5', 'B', ' ')]]),
72     ('2A^3 + 8B^T - 3C^{-1}', [(('2', 'A', '3'), ('8', 'B', 'T'), ('-3', 'C', '-1')])),
73     ('4.9A^2 - 3rot(134.2)^{-1} + 7.6B^8', [(('4.9', 'A', '2'), ('-3', 'rot(134.2)', '-1'), ('7.6', 'B', '8')])),
74
75     # Additions with multiplication
76     ('2.14A^{3} 4.5rot(14.5)^{-1} + 8B^T - 3C^{-1}', [(('2.14', 'A', '3'), ('4.5', 'rot(14.5)', '-1'),
77                                                         [('8', 'B', 'T'), ('-3', 'C', '-1')])),
78     ('2.14A^{3} 4.5rot(14.5)^{-1} + 8.5B^T 5.97C^{14} - 3.14D^{^{-1}} 6.7E^T',
79      [(('2.14', 'A', '3'), ('4.5', 'rot(14.5)', '-1'), [('8.5', 'B', 'T'), ('5.97', 'C', '14'),
80                                                         [('-3.14', 'D', '-1'), ('6.7', 'E', 'T')]])),
81 ]
82
83
84 def test_parse_matrix_expression() -> None:
85     """Test the parse_matrix_expression() function."""
86     for expression, parsed_expression in expressions_and_parsed_expressions:
87         # Test it with and without whitespace
88         assert parse_matrix_expression(expression) == parsed_expression
89         assert parse_matrix_expression(expression.replace(' ', '')) == parsed_expression
90
91
92 def test_parse_error() -> None:
93     """Test that parse_matrix_expression() raises a MatrixParseError."""
94     for expression in invalid_inputs:
95         with pytest.raises(MatrixParseError):
96             parse_matrix_expression(expression)

```

### B.3 matrices/matrix\_wrapper/test\_misc.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 miscellaneous methods of the MatrixWrapper class."""
8
9 from lintrans.matrices import MatrixWrapper
10
11
12 def test_get_expression(test_wrapper: MatrixWrapper) -> None:
13     """Test the get_expression method of the MatrixWrapper class."""
14     test_wrapper['N'] = 'A^2'
15     test_wrapper['O'] = '4B'
16     test_wrapper['P'] = 'A+C'
17
18     test_wrapper['Q'] = 'N^-1'
19     test_wrapper['R'] = 'P-40'
20     test_wrapper['S'] = 'NOP'
21
22     assert test_wrapper.get_expression('A') is None
23     assert test_wrapper.get_expression('B') is None
24     assert test_wrapper.get_expression('C') is None
25     assert test_wrapper.get_expression('D') is None
26     assert test_wrapper.get_expression('E') is None
27     assert test_wrapper.get_expression('F') is None
28     assert test_wrapper.get_expression('G') is None
29
30     assert test_wrapper.get_expression('N') == 'A^2'
31     assert test_wrapper.get_expression('O') == '4B'
32     assert test_wrapper.get_expression('P') == 'A+C'
33
34     assert test_wrapper.get_expression('Q') == 'N^-1'
35     assert test_wrapper.get_expression('R') == 'P-40'
36     assert test_wrapper.get_expression('S') == 'NOP'

```

## B.4 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:
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.5 matrices/matrix\_wrapper/test\_evaluate\_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 MatrixWrapper evaluate_expression() method."""
8
9  import numpy as np
10 from numpy import linalg as la
11 import pytest
12
13 from lintrans.matrices import MatrixWrapper, create_rotation_matrix
14 from lintrans.typing_ import MatrixType
15
16 from confest import get_test_wrapper
17
18
19 def test_simple_matrix_addition(test_wrapper: MatrixWrapper) -> None:
20     """Test simple addition and subtraction of two matrices."""
21
22     # NOTE: We assert that all of these values are not None just to stop mypy complaining
23     # These values will never actually be None because they're set in the wrapper() fixture
24     # There's probably a better way do this, because this method is a bit of a bodge, but this works for now
25     assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
26         test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
27         test_wrapper['G'] is not None
28
29     assert (test_wrapper.evaluate_expression('A+B') == test_wrapper['A'] + test_wrapper['B']).all()
30     assert (test_wrapper.evaluate_expression('E+F') == test_wrapper['E'] + test_wrapper['F']).all()
31     assert (test_wrapper.evaluate_expression('G+D') == test_wrapper['G'] + test_wrapper['D']).all()
32     assert (test_wrapper.evaluate_expression('C+C') == test_wrapper['C'] + test_wrapper['C']).all()
33     assert (test_wrapper.evaluate_expression('D+A') == test_wrapper['D'] + test_wrapper['A']).all()
34     assert (test_wrapper.evaluate_expression('B+C') == test_wrapper['B'] + test_wrapper['C']).all()
35
36     assert test_wrapper == get_test_wrapper()
37
38
39 def test_simple_two_matrix_multiplication(test_wrapper: MatrixWrapper) -> None:
40     """Test simple multiplication of two matrices."""
41     assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
42         test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
43         test_wrapper['G'] is not None
44
45     assert (test_wrapper.evaluate_expression('AB') == test_wrapper['A'] @ test_wrapper['B']).all()
46     assert (test_wrapper.evaluate_expression('BA') == test_wrapper['B'] @ test_wrapper['A']).all()
47     assert (test_wrapper.evaluate_expression('AC') == test_wrapper['A'] @ test_wrapper['C']).all()
48     assert (test_wrapper.evaluate_expression('DA') == test_wrapper['D'] @ test_wrapper['A']).all()
49     assert (test_wrapper.evaluate_expression('ED') == test_wrapper['E'] @ test_wrapper['D']).all()
50     assert (test_wrapper.evaluate_expression('FD') == test_wrapper['F'] @ test_wrapper['D']).all()
51     assert (test_wrapper.evaluate_expression('GA') == test_wrapper['G'] @ test_wrapper['A']).all()
52     assert (test_wrapper.evaluate_expression('CF') == test_wrapper['C'] @ test_wrapper['F']).all()
53     assert (test_wrapper.evaluate_expression('AG') == test_wrapper['A'] @ test_wrapper['G']).all()
54
55     assert test_wrapper == get_test_wrapper()
56
57
58 def test_identity_multiplication(test_wrapper: MatrixWrapper) -> None:
59     """Test that multiplying by the identity doesn't change the value of a matrix."""

```

```

60     assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
61           test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
62           test_wrapper['G'] is not None
63
64     assert (test_wrapper.evaluate_expression('I') == test_wrapper['I']).all()
65     assert (test_wrapper.evaluate_expression('AI') == test_wrapper['A']).all()
66     assert (test_wrapper.evaluate_expression('IA') == test_wrapper['A']).all()
67     assert (test_wrapper.evaluate_expression('GI') == test_wrapper['G']).all()
68     assert (test_wrapper.evaluate_expression('IG') == test_wrapper['G']).all()
69
70     assert (test_wrapper.evaluate_expression('EID') == test_wrapper['E'] @ test_wrapper['D']).all()
71     assert (test_wrapper.evaluate_expression('IED') == test_wrapper['E'] @ test_wrapper['D']).all()
72     assert (test_wrapper.evaluate_expression('EDI') == test_wrapper['E'] @ test_wrapper['D']).all()
73     assert (test_wrapper.evaluate_expression('IEIDI') == test_wrapper['E'] @ test_wrapper['D']).all()
74     assert (test_wrapper.evaluate_expression('EI*3D') == test_wrapper['E'] @ test_wrapper['D']).all()
75
76     assert test_wrapper == get_test_wrapper()
77
78
79 def test_simple_three_matrix_multiplication(test_wrapper: MatrixWrapper) -> None:
80     """Test simple multiplication of two matrices."""
81     assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
82           test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
83           test_wrapper['G'] is not None
84
85     assert (test_wrapper.evaluate_expression('ABC') == test_wrapper['A'] @ test_wrapper['B'] @
86           ↪ test_wrapper['C']).all()
87     assert (test_wrapper.evaluate_expression('ACB') == test_wrapper['A'] @ test_wrapper['C'] @
88           ↪ test_wrapper['B']).all()
89     assert (test_wrapper.evaluate_expression('BAC') == test_wrapper['B'] @ test_wrapper['A'] @
90           ↪ test_wrapper['C']).all()
91     assert (test_wrapper.evaluate_expression('EFG') == test_wrapper['E'] @ test_wrapper['F'] @
92           ↪ test_wrapper['G']).all()
93     assert (test_wrapper.evaluate_expression('DAC') == test_wrapper['D'] @ test_wrapper['A'] @
94           ↪ test_wrapper['C']).all()
95     assert (test_wrapper.evaluate_expression('GAE') == test_wrapper['G'] @ test_wrapper['A'] @
96           ↪ test_wrapper['E']).all()
97     assert (test_wrapper.evaluate_expression('FAG') == test_wrapper['F'] @ test_wrapper['A'] @
98           ↪ test_wrapper['G']).all()
99     assert (test_wrapper.evaluate_expression('GAF') == test_wrapper['G'] @ test_wrapper['A'] @
100           ↪ test_wrapper['F']).all()
101
102     assert test_wrapper == get_test_wrapper()
103
104 def test_matrix_inverses(test_wrapper: MatrixWrapper) -> None:
105     """Test the inverses of single matrices."""
106     assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
107           test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
108           test_wrapper['G'] is not None
109
110     assert (test_wrapper.evaluate_expression('A^{-1}') == la.inv(test_wrapper['A'])).all()
111     assert (test_wrapper.evaluate_expression('B^{-1}') == la.inv(test_wrapper['B'])).all()
112     assert (test_wrapper.evaluate_expression('C^{-1}') == la.inv(test_wrapper['C'])).all()
113     assert (test_wrapper.evaluate_expression('D^{-1}') == la.inv(test_wrapper['D'])).all()
114     assert (test_wrapper.evaluate_expression('E^{-1}') == la.inv(test_wrapper['E'])).all()
115     assert (test_wrapper.evaluate_expression('F^{-1}') == la.inv(test_wrapper['F'])).all()
116     assert (test_wrapper.evaluate_expression('G^{-1}') == la.inv(test_wrapper['G'])).all()
117
118     assert (test_wrapper.evaluate_expression('A^{-1}') == la.inv(test_wrapper['A'])).all()
119     assert (test_wrapper.evaluate_expression('B^{-1}') == la.inv(test_wrapper['B'])).all()
120     assert (test_wrapper.evaluate_expression('C^{-1}') == la.inv(test_wrapper['C'])).all()
121     assert (test_wrapper.evaluate_expression('D^{-1}') == la.inv(test_wrapper['D'])).all()
122     assert (test_wrapper.evaluate_expression('E^{-1}') == la.inv(test_wrapper['E'])).all()
123     assert (test_wrapper.evaluate_expression('F^{-1}') == la.inv(test_wrapper['F'])).all()
124     assert (test_wrapper.evaluate_expression('G^{-1}') == la.inv(test_wrapper['G'])).all()
125
126     assert test_wrapper == get_test_wrapper()
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 \

```

```

125         test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
126         test_wrapper['G'] is not None
127
128     assert (test_wrapper.evaluate_expression('A^2') == la.matrix_power(test_wrapper['A'], 2)).all()
129     assert (test_wrapper.evaluate_expression('B^4') == la.matrix_power(test_wrapper['B'], 4)).all()
130     assert (test_wrapper.evaluate_expression('C^{12}') == la.matrix_power(test_wrapper['C'], 12)).all()
131     assert (test_wrapper.evaluate_expression('D^{12}') == la.matrix_power(test_wrapper['D'], 12)).all()
132     assert (test_wrapper.evaluate_expression('E^8') == la.matrix_power(test_wrapper['E'], 8)).all()
133     assert (test_wrapper.evaluate_expression('F^{6}') == la.matrix_power(test_wrapper['F'], 6)).all()
134     assert (test_wrapper.evaluate_expression('G^{-2}') == la.matrix_power(test_wrapper['G'], -2)).all()
135
136     assert test_wrapper == get_test_wrapper()
137
138
139 def test_matrix_transpose(test_wrapper: MatrixWrapper) -> None:
140     """Test matrix transpositions."""
141     assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
142            test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
143            test_wrapper['G'] is not None
144
145     assert (test_wrapper.evaluate_expression('A^{T}') == test_wrapper['A'].T).all()
146     assert (test_wrapper.evaluate_expression('B^{T}') == test_wrapper['B'].T).all()
147     assert (test_wrapper.evaluate_expression('C^{T}') == test_wrapper['C'].T).all()
148     assert (test_wrapper.evaluate_expression('D^{T}') == test_wrapper['D'].T).all()
149     assert (test_wrapper.evaluate_expression('E^{T}') == test_wrapper['E'].T).all()
150     assert (test_wrapper.evaluate_expression('F^{T}') == test_wrapper['F'].T).all()
151     assert (test_wrapper.evaluate_expression('G^{T}') == test_wrapper['G'].T).all()
152
153     assert (test_wrapper.evaluate_expression('A^T') == test_wrapper['A'].T).all()
154     assert (test_wrapper.evaluate_expression('B^T') == test_wrapper['B'].T).all()
155     assert (test_wrapper.evaluate_expression('C^T') == test_wrapper['C'].T).all()
156     assert (test_wrapper.evaluate_expression('D^T') == test_wrapper['D'].T).all()
157     assert (test_wrapper.evaluate_expression('E^T') == test_wrapper['E'].T).all()
158     assert (test_wrapper.evaluate_expression('F^T') == test_wrapper['F'].T).all()
159     assert (test_wrapper.evaluate_expression('G^T') == test_wrapper['G'].T).all()
160
161     assert test_wrapper == get_test_wrapper()
162
163
164 def test_rotation_matrices(test_wrapper: MatrixWrapper) -> None:
165     """Test that 'rot(angle)' can be used in an expression."""
166     assert (test_wrapper.evaluate_expression('rot(90)') == create_rotation_matrix(90)).all()
167     assert (test_wrapper.evaluate_expression('rot(180)') == create_rotation_matrix(180)).all()
168     assert (test_wrapper.evaluate_expression('rot(270)') == create_rotation_matrix(270)).all()
169     assert (test_wrapper.evaluate_expression('rot(360)') == create_rotation_matrix(360)).all()
170     assert (test_wrapper.evaluate_expression('rot(45)') == create_rotation_matrix(45)).all()
171     assert (test_wrapper.evaluate_expression('rot(30)') == create_rotation_matrix(30)).all()
172
173     assert (test_wrapper.evaluate_expression('rot(13.43)') == create_rotation_matrix(13.43)).all()
174     assert (test_wrapper.evaluate_expression('rot(49.4)') == create_rotation_matrix(49.4)).all()
175     assert (test_wrapper.evaluate_expression('rot(-123.456)') == create_rotation_matrix(-123.456)).all()
176     assert (test_wrapper.evaluate_expression('rot(963.245)') == create_rotation_matrix(963.245)).all()
177     assert (test_wrapper.evaluate_expression('rot(-235.24)') == create_rotation_matrix(-235.24)).all()
178
179     assert test_wrapper == get_test_wrapper()
180
181
182 def test_multiplication_and_addition(test_wrapper: MatrixWrapper) -> None:
183     """Test multiplication and addition of matrices together."""
184     assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
185            test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
186            test_wrapper['G'] is not None
187
188     assert (test_wrapper.evaluate_expression('AB+C') ==
189            test_wrapper['A'] @ test_wrapper['B'] + test_wrapper['C']).all()
189     assert (test_wrapper.evaluate_expression('DE-D') ==
190            test_wrapper['D'] @ test_wrapper['E'] - test_wrapper['D']).all()
191     assert (test_wrapper.evaluate_expression('FD+AB') ==
192            test_wrapper['F'] @ test_wrapper['D'] + test_wrapper['A'] @ test_wrapper['B']).all()
193     assert (test_wrapper.evaluate_expression('BA-DE') ==
194            test_wrapper['B'] @ test_wrapper['A'] - test_wrapper['D'] @ test_wrapper['E']).all()
195
196     assert (test_wrapper.evaluate_expression('2AB+3C') ==

```

```

198         (2 * test_wrapper['A']) @ test_wrapper['B'] + (3 * test_wrapper['C'])).all()
199     assert (test_wrapper.evaluate_expression('4D7.9E-1.2A') ==
200            (4 * test_wrapper['D']) @ (7.9 * test_wrapper['E']) - (1.2 * test_wrapper['A'])).all()
201
202     assert test_wrapper == get_test_wrapper()
203
204
205 def test_complicated_expressions(test_wrapper: MatrixWrapper) -> None:
206     """Test evaluation of complicated expressions."""
207     assert test_wrapper['A'] is not None and test_wrapper['B'] is not None and test_wrapper['C'] is not None and \
208            test_wrapper['D'] is not None and test_wrapper['E'] is not None and test_wrapper['F'] is not None and \
209            test_wrapper['G'] is not None
210
211     assert (test_wrapper.evaluate_expression('-3.2A^T 4B^{-1} 6C^{-1} + 8.1D^{2} 3.2E^4') ==
212            (-3.2 * test_wrapper['A'].T) @ (4 * la.inv(test_wrapper['B'])) @ (6 * la.inv(test_wrapper['C']))
213            + (8.1 * la.matrix_power(test_wrapper['D'], 2)) @ (3.2 * la.matrix_power(test_wrapper['E'], 4))).all()
214
215     assert (test_wrapper.evaluate_expression('53.6D^{2} 3B^T - 4.9F^{2} 2D + A^3 B^{-1}') ==
216            (53.6 * la.matrix_power(test_wrapper['D'], 2)) @ (3 * test_wrapper['B'].T)
217            - (4.9 * la.matrix_power(test_wrapper['F'], 2)) @ (2 * test_wrapper['D'])
218            + la.matrix_power(test_wrapper['A'], 3) @ la.inv(test_wrapper['B'])).all()
219
220     assert test_wrapper == get_test_wrapper()
221
222
223 def test_value_errors(test_wrapper: MatrixWrapper) -> None:
224     """Test that evaluate_expression() raises a ValueError for any malformed input."""
225     invalid_expressions = ['', '+', '-', 'This is not a valid expression', '3+4',
226                           'A+2', 'A^', '^2', 'A^-', 'At', 'A^t', '3^2']
227
228     for expression in invalid_expressions:
229         with pytest.raises(ValueError):
230             test_wrapper.evaluate_expression(expression)
231
232
233 def test_linalgerror() -> None:
234     """Test that certain expressions raise np.linalg.LinAlgError."""
235     matrix_a: MatrixType = np.array([
236         [0, 0],
237         [0, 0]
238     ])
239
240     matrix_b: MatrixType = np.array([
241         [1, 2],
242         [1, 2]
243     ])
244
245     wrapper = MatrixWrapper()
246     wrapper['A'] = matrix_a
247     wrapper['B'] = matrix_b
248
249     assert (wrapper.evaluate_expression('A') == matrix_a).all()
250     assert (wrapper.evaluate_expression('B') == matrix_b).all()
251
252     with pytest.raises(np.linalg.LinAlgError):
253         wrapper.evaluate_expression('A^{-1}')
254
255     with pytest.raises(np.linalg.LinAlgError):
256         wrapper.evaluate_expression('B^{-1}')
257
258     assert (wrapper['A'] == matrix_a).all()
259     assert (wrapper['B'] == matrix_b).all()

```

## B.6 matrices/matrix\_wrapper/test\_setitem\_and\_getitem.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 MatrixWrapper __setitem__() and __getitem__() methods."""
8
9 import numpy as np
10 from numpy import linalg as la
11 import pytest
12 from typing import Any
13
14 from lintrans.matrices import MatrixWrapper
15 from lintrans.typing_ import MatrixType
16
17 valid_matrix_names = 'ABCDEFGHJKLMNOPQRSTUVWXYZ'
18 invalid_matrix_names = ['bad name', '123456', 'Th15 Is an 1nV@l1D n@m3', 'abc', 'a']
19
20 test_matrix: MatrixType = np.array([[1, 2], [4, 3]])
21
22
23 def test_basic_get_matrix(new_wrapper: MatrixWrapper) -> None:
24     """Test MatrixWrapper().__getitem__()."""
25     for name in valid_matrix_names:
26         assert new_wrapper[name] is None
27
28     assert (new_wrapper['I'] == np.array([[1, 0], [0, 1]])).all()
29
30
31 def test_get_name_error(new_wrapper: MatrixWrapper) -> None:
32     """Test that MatrixWrapper().__getitem__() raises a NameError if called with an invalid name."""
33     for name in invalid_matrix_names:
34         with pytest.raises(NameError):
35             _ = new_wrapper[name]
36
37
38 def test_basic_set_matrix(new_wrapper: MatrixWrapper) -> None:
39     """Test MatrixWrapper().__setitem__()."""
40     for name in valid_matrix_names:
41         new_wrapper[name] = test_matrix
42         assert (new_wrapper[name] == test_matrix).all()
43
44     new_wrapper[name] = None
45     assert new_wrapper[name] is None
46
47
48 def test_set_expression(test_wrapper: MatrixWrapper) -> None:
49     """Test that MatrixWrapper.__setitem__() can accept a valid expression."""
50     test_wrapper['N'] = 'A^2'
51     test_wrapper['O'] = 'BA+2C'
52     test_wrapper['P'] = 'E^T'
53     test_wrapper['Q'] = 'C^-1B'
54     test_wrapper['R'] = 'A^{2}3B'
55     test_wrapper['S'] = 'N^-1'
56     test_wrapper['T'] = 'PQP^-1'
57
58     with pytest.raises(TypeError):
59         test_wrapper['U'] = 'A+1'
60
61     with pytest.raises(TypeError):
62         test_wrapper['V'] = 'K'
63
64     with pytest.raises(TypeError):
65         test_wrapper['W'] = 'L^2'
66
67     with pytest.raises(TypeError):
68         test_wrapper['X'] = 'M^-1'
69
70
71 def test_simple_dynamic_evaluation(test_wrapper: MatrixWrapper) -> None:
72     """Test that expression-defined matrices are evaluated dynamically."""
73     test_wrapper['N'] = 'A^2'
74     test_wrapper['O'] = '4B'
75     test_wrapper['P'] = 'A+C'
76
77     assert (test_wrapper['N'] == test_wrapper.evaluate_expression('A^2')).all()
78     assert (test_wrapper['O'] == test_wrapper.evaluate_expression('4B')).all()

```



```

79     assert (test_wrapper['P'] == test_wrapper.evaluate_expression('A+C')).all()
80
81     assert (test_wrapper.evaluate_expression('N^2 + 30') ==
82             la.matrix_power(test_wrapper.evaluate_expression('A^2'), 2) +
83             3 * test_wrapper.evaluate_expression('4B')
84             ).all()
85     assert (test_wrapper.evaluate_expression('P^-1 - 3N0^2') ==
86             la.inv(test_wrapper.evaluate_expression('A+C')) -
87             (3 * test_wrapper.evaluate_expression('A^2')) @
88             la.matrix_power(test_wrapper.evaluate_expression('4B'), 2)
89             ).all()
90
91     test_wrapper['A'] = np.array([
92         [19, -21.5],
93         [84, 96.572]
94     ])
95     test_wrapper['B'] = np.array([
96         [-0.993, 2.52],
97         [1e10, 0]
98     ])
99     test_wrapper['C'] = np.array([
100         [0, 19512],
101         [1.414, 19]
102     ])
103
104     assert (test_wrapper['N'] == test_wrapper.evaluate_expression('A^2')).all()
105     assert (test_wrapper['O'] == test_wrapper.evaluate_expression('4B')).all()
106     assert (test_wrapper['P'] == test_wrapper.evaluate_expression('A+C')).all()
107
108     assert (test_wrapper.evaluate_expression('N^2 + 30') ==
109             la.matrix_power(test_wrapper.evaluate_expression('A^2'), 2) +
110             3 * test_wrapper.evaluate_expression('4B')
111             ).all()
112     assert (test_wrapper.evaluate_expression('P^-1 - 3N0^2') ==
113             la.inv(test_wrapper.evaluate_expression('A+C')) -
114             (3 * test_wrapper.evaluate_expression('A^2')) @
115             la.matrix_power(test_wrapper.evaluate_expression('4B'), 2)
116             ).all()
117
118
119 def test_recursive_dynamic_evaluation(test_wrapper: MatrixWrapper) -> None:
120     """Test that dynamic evaluation works recursively."""
121     test_wrapper['N'] = 'A^2'
122     test_wrapper['O'] = '4B'
123     test_wrapper['P'] = 'A+C'
124
125     test_wrapper['Q'] = 'N^-1'
126     test_wrapper['R'] = 'P-4O'
127     test_wrapper['S'] = 'NOP'
128
129     assert test_wrapper['Q'] == pytest.approx(test_wrapper.evaluate_expression('A^-2'))
130     assert test_wrapper['R'] == pytest.approx(test_wrapper.evaluate_expression('A + C - 16B'))
131     assert test_wrapper['S'] == pytest.approx(test_wrapper.evaluate_expression('A^{2}4BA + A^{2}4BC'))
132
133
134 def test_set_identity_error(new_wrapper: MatrixWrapper) -> None:
135     """Test that MatrixWrapper().__setitem__() raises a NameError when trying to assign to I."""
136     with pytest.raises(NameError):
137         new_wrapper['I'] = test_matrix
138
139
140 def test_set_name_error(new_wrapper: MatrixWrapper) -> None:
141     """Test that MatrixWrapper().__setitem__() raises a NameError when trying to assign to an invalid name."""
142     for name in invalid_matrix_names:
143         with pytest.raises(NameError):
144             new_wrapper[name] = test_matrix
145
146
147 def test_set_type_error(new_wrapper: MatrixWrapper) -> None:
148     """Test that MatrixWrapper().__setitem__() raises a TypeError when trying to set a non-matrix."""
149     invalid_values: list[Any] = [
150         12,
151         [1, 2, 3, 4, 5],

```

```

152         [[1, 2], [3, 4]],
153         True,
154         24.3222,
155         'This is totally a matrix, I swear',
156         MatrixWrapper,
157         MatrixWrapper(),
158         np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),
159         np.eye(100)
160     ]
161
162     for value in invalid_values:
163         with pytest.raises(TypeError):
164             new_wrapper['M'] = value

```

## B.7 gui/test\_dialog\_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  import numpy as np
10 import pytest
11
12 from lintrans.gui.dialogs.define_new_matrix import is_valid_float, round_float
13
14 valid_floats: list[str] = [
15     '0', '1', '3', '-2', '123', '-208', '1.2', '-3.5', '4.252634', '-42362.352325',
16     '1e4', '-2.59e3', '4.13e-6', '-5.5244e-12'
17 ]
18
19 invalid_floats: list[str] = [
20     '', 'pi', 'e', '1.2.3', '1,2', '-', '.', 'None', 'no', 'yes', 'float'
21 ]
22
23
24 @pytest.mark.parametrize('inputs,output', [(valid_floats, True), (invalid_floats, False)])
25 def test_is_valid_float(inputs: list[str], output: bool) -> None:
26     """Test the is_valid_float() function."""
27     for inp in inputs:
28         assert is_valid_float(inp) == output
29
30
31 def test_round_float() -> None:
32     """Test the round_float() function."""
33     expected_values: list[tuple[float, int, str]] = [
34         (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'),
35         (np.sqrt(2) / 2, 5, '0.70711'), (-1 * np.sqrt(2) / 2, 5, '-0.70711'),
36         (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'),
37         (1.23456789, 2, '1.23'), (1.23456789, 3, '1.235'), (1.23456789, 4, '1.2346'), (1.23456789, 5, '1.23457'),
38         (12345.678, 1, '12345.7'), (12345.678, 2, '12345.68'), (12345.678, 3, '12345.678'),
39     ]
40
41     for num, precision, answer in expected_values:
42         assert round_float(num, precision) == answer

```