Maths 6141 - Numerical Methods

Computer Lab 1

Computer Lab 1

The aim of the first computer lab is to gain basic familiarity with the capabilities of python. To do this you must use it. There are a number of resources with introductions; you should pick the one you find most effective and work through it during the lab, asking for assistance with any points that you do not understand. Once you are confident that you can complete the tasks below you will have some of the basics needed.
There are many ways to use python; on a University iSolutions machine, probably the best is to use spyder. Go to the `Start` menu, then `All Programs`, then `Programming Languages`, then `Anaconda`. Alternatively, type `spyder` into the search bar.

Resources

A lot of information is available online, including material from Prof. Fangohr in Engineering, or a complete book from Prof. Pine at NYU. There are also Johansson's lectures on scientific computing with python.
In particular, take a look at first steps in spyder.
Particular tools (python packages) that we shall use a lot include numpy, scipy and particularly matplotlib.
For those that already know Matlab, there is a detailed wiki showing how to convert to python.

Tasks

Desktop

Work out how to navigate spyder. You should have an editor window, a help window, and a console (look at the first steps in spyder to make sure it is an `ipython` console). Ensure you know how to set the directory that you are working in. Experiment in the console, but make certain to use the editor to save work.

Scalars, vectors, and matrices

In the *console*

1. Check the obvious operations such as `1+1`, `2-1`, `2*1`, and `1/2` do what you expect. Check that assigning the result of a calculation to a variable (e.g. `two=1+1`, and then type `two` to check its value) does what you expect.

2. Vectors can represent a mathematical vector $\boldsymbol{x}$ or a function of one variable $x(t)$ and specific values of t. python does not know the difference and will treat them the same. It *does not* know the difference between row and column vectors. Ensure that you have imported numpy (`import numpy as np`). Enter the vector
$$\boldsymbol{x} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$
using `x = np.array([1,2])`. Check its shape (`x.shape`). Compute its transpose (`y=x.T`) and check its shape as well.

3. As with vectors, python arrays can represent genuine matrices, or vector fields such as $\boldsymbol{x}(t)$, or just a collection of numbers with a multidimensional structure. Arrays are at the heart of numpy with a number of important operations. Enter the matrix

$$A = \begin{pmatrix} 2 & 3 \\ 6 & 5 \end{pmatrix}.$$

   You should understand why `A = np.array([[2, 3],[6, 5]])` is the notation to use.

4. Try using the function `len` and check the attributes `size` and `shape` on your defined vectors and arrays (e.g. `len(x)`, or `A.size`). Understand when and how a vector is just a special type of array.

5. Check directly that $\boldsymbol{x}$ is an eigenvector of A by computing $A\boldsymbol{x}$ in one operation. (Look up the `dot` function in numpy).

6. Compute the eigenvectors and eigenvalues of A using the numpy linear algebra function `eig` (see `numpy.linalg.eig`).

7. Set A to equal its transpose.

8. Functions and operations can be applied to vectors and matrices. Sometimes the result is obvious (e.g., `np.sin(x)` computes the sine of each entry of the vector `x`). Sometimes you need to think: `A**2` squares each entry, but `np.dot(A,A)` computes the matrix square. Experiment to see which operations apply to each entry and which behave like matrix multiplication.

9. Compute the eigenvectors of the new A and set $\boldsymbol{x}$ equal to its first eigenvector using array subscripting ("colon notation").

10. Not all useful linear algebra functions are given by numpy. Import scipy and explore its linear algebra functions (`scipy.linalg`), checking that you can calculate the matrix exponential and the condition number of a matrix.

Scripts

Once you have completed the above, use the *Editor* to write a script that does the same. Reset the console (using `%reset`) and run your script. Note any warnings that the Editor gives you. Understand the use of semicolons to suppress output.

Plotting

matplotlib is the key set of plotting and visualization tools. This is essential for understanding the results of numerical methods, and you should be familiar with the production of legible figures that clearly show the key information needed to convey both the interesting results produced, and also the correctness of the methods used. You should check the python scripts accompanying the lecture slides on Blackboard to see how particular figures are produced. The following basic tasks illustrate some of the key steps required. The standard starting point is to `import matplotlib.pyplot as plt`.

1. Create two arrays `x` and `y`, containing 80 and 60 points respectively, equally spaced between $[0, 1]$ and $[0, 2]$ respectively. Do this using the numpy command `linspace`.

2. Create vectors containing $\sin(x)$ and $\exp(-y^2)$. In different windows plot, using default arguments, these functions against the appropriate arguments.

3. Work out how to add axis labels, titles and legends to the plots.

4. Work out how to change the font size of the labels, titles and legends.

5. Work out how to change the plotting style used. Plot one of the functions using both markers, such as black crosses, and a non-standard line, such as a thick red dashed line.

6. Try changing the axis of one of the plots so that only half the range is shown. Ensure that you can do this using both the GUI tools and with command line arguments.

7. Investigate the different logarithmic plots `semilogx`, `semilogy` and `loglog`, being careful to only use them when it is sensible.

8. Briefly look at other plotting commands such as bar graphs and histograms. You should take care with the differences in the arguments used.

As above, you should ensure that every step that you take can be reproduced by putting the commands in to a script which produces the same results each time.

1. Starting from your 1d vectors `x` and `y`, create 2d arrays `X` and `Y` using the function `meshgrid`. These are to be the coordinate points for producing simple 3d plots.

2. Create another array `Z` from
$$z = \cos(2\pi x y^2).$$

3. Produce different types of plot, investigating the difference between the python commands `plot_surface`, `contour`, `plot_wireframe`, and any others you might find.

4. Investigate how to rotate the viewpoint.

5. With the surface plots, try the effect of different colourmaps.

As you know the unit circle can be expressed in parametric form

$$x = \cos\theta$$
$$y = \sin\theta$$

with $\theta \in [0, 2\pi]$.

1. In the same window plot x vs. θ and y vs. θ.

2. In a different window produce two subplots of x and y vs. θ.

3. Now go back to the previous plot and add axis labels and a legend.

4. Plot the unit circle as a line in a 3-d plot (set the z coordinate to zero).

5. Create a random 2×2 matrix A.

6. Show the effect of A on points on the unit circle. That is, denote the coordinates of the unit circle as
$$\boldsymbol{x} = \begin{pmatrix} x \\ y \end{pmatrix},$$
compute the new coordinates $\boldsymbol{x}' = A\boldsymbol{x}$, and plot the result in the same way as you plotted the unit circle above.

7. Investigate the effect of powers of A (A^2, A^5, etc.) on the unit circle as above. What does this tell you about the eigenvalues of A?