EMAT30008 Scientific Computing
Week 13: Python exercises

Before starting these exercises, create a new repository on GitHub called Exercises. Then, using the `git clone` command, create a local version of this repository on your computer. Create a new file for each exercise, e.g. called Exercise1.py, Exercise2.py, etc. As you complete these exercises, be sure to commit your changes regularly. Once you complete an exercise, push the changes to GitHub.

1. The value of π can be approximated using the Leibniz formula:

$$\pi \approx \pi_N = \sum_{n=0}^{N} \frac{8}{(4n+1)(4n+3)}$$

   where N is a large number. Taking the limit as $N \to \infty$ produces the exact value of π, but this requires evaluating an infinite number of terms, which is impossible on a computer. Therefore, we can only approximate the value of π by using a finite number of terms in the sum.

   (a) Use this formula to compute approximations to π by taking $N = 100$, $N = 1,000$, and $N = 10,000$.

   (b) Given that $\pi = 3.141592653589793...$, what is the error of the approximation in each of these cases? **Note**: the error is defined as $|\pi - \pi_N|$. The function `abs` can be used to compute the absolute value in Python. The `math` package provides a variable for π called `pi` that can be imported using the code `from math import pi`.

   (c) What value of N is needed to produce an error that is less than 10^{-7}?

   (d) Now that you've finished the exercise, don't forget to push your repository to GitHub!

2. (a) Given two vectors in the form of two lists, e.g. $\vec{a} = [1, 2, 3]$ and $\vec{b} = [6, 5, 4]$, write a Python function that returns the dot product of these vectors.

   (b) Given two matrices in the form of nested lists (lists of lists), e.g. $A = [[1, 2], [3, 4]]$, write a Python function that returns the product of these two matrices.

   (c) Edit your code so that error messages are printed if the vectors and matrices do not have consistent sizes.

3. Create a NumPy array called `a` that stores the array [5, 4, 9, 2, 0, 4, 7, 2].

   (a) Print the last entry of `a`. **Hint**: You can use the index -1 to access the last entry of lists, strings, NumPy arrays, etc.

   (b) Print the values of `a[1:6]` and explain the output. Now try printing the values of `a[:-2]` and `a[::2]`. What do these do?

   (c) Change the last entry of `a` to -9 and print the result. Now run the command `a[0:3] = 1` and print the result. How has this altered `a`?

4. (a) Create a NumPy array r that contains 20 random numbers between 1 and 9 that from a uniform distribution. Print the result.

   (b) **Logical indexing** provides a quick way to access and modify entries in a NumPy array that satisfy certain criteria. In this question, we'll use logical indexing to replace all of the entries in r that are smaller than 5 with 0. First, run the command `idx = r < 5`. Print the value of `idx`. Explain the result you see. Now run the command `r[idx] = 0` and print the value of r. What has happened?

5. Solve the linear system of equations $Ax = b$ where

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 2 \end{pmatrix} \tag{1}$$

Print the solution x. Then compute $Ax - b$ and print the result. Can you explain the values you see?

6. (a) Use NumPy's `linspace` function to create an array called t that contains 500 values between 0 and 5. Create a second array called y that stores the values of $y = t^2 e^{-2t}$. **Hint**: use the `exp` function to compute the exponential of a NumPy array.

   (b) Plot y as a function of x. Add labels to the x and y axes. Edit the line colour and thickness and font sizes to your preference.

   (c) Find the maximum value of y. **Note**: this is a simple way of finding the maximum of a function.

   (d) Use logical indexing or otherwise to find the value of t at which y is maximal. Does this match up with what you see in your plot?

7. The Moore–Penrose inverse, or **pseudoinverse**, provides a means to define the inverse of singular and non-square matrices. For a matrix A with linearly independent columns, the pseudoinverse is defined as $A^+ = (A^T A)^{-1} A^T$, where A^T is the transpose of A.

   (a) Write a Python function that returns the pseudoinverse of a matrix A.

   (b) Consider the matrix

$$A = \begin{pmatrix} 1 & -1 \\ 2 & 4 \\ 1 & 1 \\ 3 & 8 \end{pmatrix}. \tag{2}$$

   Compute $A^+ A$ and show that this is equal to I, the identity matrix.

   (c) Now consider the over-determined system of linear equations given by

$$\begin{pmatrix} 1 & -1 \\ 2 & 4 \\ 1 & 1 \\ 3 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 6 \\ 8 \end{pmatrix}. \tag{3}$$

   Use the pseudoinverse to compute the solution x. Hint: left multiply the system of equations $Ax = b$ by A^+ and use $A^+ A = I$ to show that $x = A^+ b$.

   It should come as a surprise that you are able to compute x because over-determined systems of equations usually do not have a solution. Indeed, performing Gaussian elimination will show that there is no vector x^* that solves the linear system (3). So, what does the vector $x = A^+ b$ correspond to then? It is the closest approximation to the vector x^* that would solve (3). More specifically, the vector $x = A^+ b$ minimises the error $||Ax - b||$, where $||\cdot||$ is the Euclidean norm. Minimising $||Ax - b||$ is known as a **least-squares problem**.

8. **Newton's method** is a way of finding the solution x to nonlinear equations of the form $f(x) = 0$. For a single equation, Newton's method works as follows. First, propose an initial guess of the solution x_0. Then, create successive approximations to the solution using the recursive formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \ldots. \tag{4}$$

until $|f(x_n)| < \epsilon$, where ϵ is some user-defined tolerance (e.g. 10^{-10}).

In this exercise, you will create a Python module called `solvers` that contains code to run Newton's method. You will then use a Python script to import the module and solve the equation $f(x) = \cos(x) - x = 0$.

(a) Create a Python file called solvers.py. This will be file for your module. In this file, write a Python function that implements Newton's method using the recursive formula above. Note: your module should not run Newton's method; this is what the script is for (see part (b)).

(b) Create a Python script called main.py that: (i) imports your `solvers` module, (ii) has Python functions to evaluate $f(x)$ and $f'(x)$, and (iii) calls the function for Newton's method. Run your script to show that the solution to $\cos(x) - x$ is $x \simeq 0.7390851332$.

(c) Solve the equation $\cos(x) - x$ using the `root` function in SciPy.

(d) Bonus: implement the bisection method and secant method in your `solvers` module and call them from within the main.py script.

If you made it to here, then well done! I hope you remembered to make lots of commits and to push your repository to GitHub after completing each exercise!