Practical 1: Using Python

1.1 Matrices and maths

Using the Python numpy package, create a script called "practical1_matrix.py" and complete the following exercises therein.

Define:

$$A = \begin{bmatrix} 2 & 9 & 0 & 0 \\ 0 & 4 & 1 & 4 \\ 7 & 5 & 5 & 1 \\ 7 & 8 & 7 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} -1 \\ 6 \\ 0 \\ 9 \end{bmatrix}, \quad a = \begin{bmatrix} 3 & 6 & 0 & 9 \end{bmatrix}$$

1. Calculate the following:

a) $A \cdot b$

b) $a + 4$

c) $b \cdot a$

d) $a \cdot b^T$

e) $A \cdot a^T$

2. What do you notice about the matrices you create by typing 'A*A' and 'A**2'?. Contrast this with what you get when you carry out matrix multiplication using the numpy functions, numpy.dot(A,A) and numpy.matmul(A,A).

3. Define a variable 'A_{slice}' and assign to it the 2nd and 3rd columns of A.

4. Linear systems or systems of equations defined in the form $Ax = b$ can be solved using Python's 'numpy.linalg.solve()' function. Use this to define and calculate x using the A and b given above.

1.2 Eigenvalues

Create a program called "practical1_eigen.py" and complete the following exercises therein. Copy your definitions for the matrix A and vector B from the previous exercise.

1. Construct a randomly generated 2x2 matrix of positive integers, where the integers range randomly between 1 and 50

2. Find the maximum and minimum elements in the matrix A.

3. Sort the values of the vector b.

4. Find the eigenvalues and eigenvectors of the matrix $B = A^{-1}$. Store the eigenvalues as a column vector called 'evals'.

5. Define I as the 4x4 identity matrix. Calculate the determinant of the matrix defined by $B - \lambda_j I$ for $j = 1, 2, 3, 4$. (Note: λ_1 is the first eigenvalue, λ_2 the second and so on).

1.3 Loops

Create a program called "practical1_loops.py" and complete the following exercises therein.

1. Define a 5x5 matrix called H with all entries equal to zero. Hint: Look up the Python 'numpy.zeros' command.

2. The NxN Hilbert matrix H, has entries defined by $H_{i,j} = 1/(i + j - 1)$ for $i, j = 1, 2, 3...N$. Create a double for-loop, a so-called 'nested' for-loop, to calculate the entries of the N=5 Hilbert matrix and check your answer using the built-in Python command 'hilbert'.

1.4 Plotting

Create a program called "practical1_plotting.py" and complete the following exercises therein.

1. Define a vector x using the Python 'numpy.linspace' command. It should have range from $x = 0$ to $x = 5$ in 101 points.

2. Define a vector y as the square of every element in x, i.e. define the 101 point vector $y = x**2$.

3. Repeat point 2, but this time define $z = x**3$.

4. Using 'matplotlib.pyplot', plot x vs. y and x vs. z on the same graph.

5. Add a legend showing which lines represent the quadratic and cubic functions respectively. Hint: When defining labels in the 'legend', the names must be strings so rather than inputting (label1,label2), it has to be (['label1','label2']). See if you can work out how to use mathematical notation in your legend, otherwise have a look at the worked solution.

1.5 Functions

Create a program called "practical1_functions.py" and complete the following exercises.

1. At the top of your program, create a new function called 'myfunction' which takes a value 'a' and return the value '$a + 1$'.

2. Below the function in your program, define a variable $a = 1$. You can now call your function by defining a new variable b as 'b = myfunction(a)'. What is the value of b?

3. Create a for-loop that goes from $i = 1$ to $i = 10$. In each loop iteration, call the function again for b, using b itself as the input: 'b = myfunction(b)'. What is the value of b after the loop?

1.6 Carbon-dating project

In your lecture notes you will find an introduction to carbon dating. The equation describing the decay of the carbon-14 isotope as a function of time is:

$$C_{14}(t) = C_{14}(0)\ \exp(-\lambda t) \tag{1}$$

Where $C_{14}(t)$ is the amount of carbon-14 at time t and λ is the decay constant.

For a given ratio $r(t) = C_{14}/C_{12}$ between the two carbon isotopes found in a sample, where t is the time since decay began $t = 0$, the age of the sample t is given by:

$$t = -\frac{1}{\lambda} \log\left(\frac{r(t)}{r(0)}\right) \tag{2}$$

With these equations in mind, create a program called "practical1_carbon.py" and complete the following exercises.

1. Using 1 and the decay constant $\lambda = 1.21 \cdot 10^{-4}$, create a function that calculates C_{14} given a time t and an initial amount $C_{14}(0)$.

2. Using this function, calculate the decay of carbon-14 starting from an initial amount of $C_{14}(0) = 500$ at a minimum of 3000 discrete points over the course of $t_{max} = 30000$ years. Plot the amount of carbon-14 vs. time to visualise the exponential decay.

3. Now using 2, create a function that accepts two ratios, $r(t)$ and $r(0)$ and calculates the age of a corresponding sample.

4. With a constant start ratio $r(0) = 1.5 \cdot 10^{-12}$ and a list of ratios r from different samples:

$$samples = \begin{bmatrix} 1.4 \cdot 10^{-12} \\ 1.1 \cdot 10^{-12} \\ 0.8 \cdot 10^{-12} \\ 0.4 \cdot 10^{-12} \\ 0.2 \cdot 10^{-12} \\ 0.1 \cdot 10^{-12} \\ 0.01 \cdot 10^{-12} \end{bmatrix}$$

Calculate the age of every sample in the fewest possible lines of code.