Introduction to Computer Programming

Week 8.3: Scientific computing with NumPy

Bristol

Scientific computing

Many real-world problems are so complex that they do not have an exact solution.

Scientific computing is concerned with the development of algorithms to find **approximate** solutions to these problems.

Many of these algorithms involve calculations with large collections of numbers (vectors and matrices)

NumPy

NumPy is a Python library that enables large collections of numbers to be stored as **arrays**.

Arrays provide a way to store vectors, matrices, and other types of numerical data.

NumPy provides very fast mathematical functions that can operate on these arrays.

Advantages of using NumPy:

- Memory efficient and very fast
- Used in other libraries (e.g. data science, machine learning)
- Extensive built-in functionality (e.g. linear algebra, statistics)

Getting started

It is common to import NumPy using the command

In [2]:
```
import numpy as np
```

Defining arrays

Arrays are defined using the `array` function.

A vector (1D array) can be created by passing a list to `array`

Example: Create an array for the vector $a = (1, 2, 3)$

In [2]:
```
a = np.array([1, 2, 3])
```

Like lists, elements in arrays are accessed using square brackets.

In NumPy, the first element of an array has index 0.

In [3]:
```
# print the first entry in a
print(a[0])
```
```
1
```

In [4]:
```
# print a and then change the third entry to 5
print(a)
a[2] = 5.0
print(a)
```
```
[1 2 3]
[1 2 5]
```

A matrix (2D array) can be created by passing `array` a nested list.

Each inner list will be a row of the matrix

Example: Define the matrix

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

In [5]:
```
A = np.array( [[1, 2], [3, 4]] )
print(A)
```
```
[[1 2]
 [3 4]]
```

Elements in a 2D array can be accessed using square brackets with indices separated by a comma.

The first index is for the row and the second is for the column

Remember, indexing starts at 0.

In [6]:
```
# print the matrix A
print(A, "\n")

# print the entry in the first row, second column
print(A[0,1])
```
```
[[1 2]
 [3 4]]

2
```

Exercise:

Create the array

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 9 & 8 & 7 \\ 2 & 4 & 8 \end{pmatrix}$$

Add the entry in the first row, second column to the entry in the third row, first column

Solution:

In [6]:
```
# creating the array
A = np.array([[1, 2, 3], [9, 8, 7], [2, 4, 8]])

# summing the entries in the 1st row, 2nd column and 3rd row, 1st column
s = A[0, 1] + A[2, 0]

# print the result
print(s)
```
```
4
```

Some useful functions for creating arrays

`linspace(a, b, N)` creates a 1D array with N uniformly spaced entries between a and b (inclusive)

In [7]:
```
# create an array with 5 entries between 0 and 1
x = np.linspace(0, 1, 5)
print(x)
```
```
[0.   0.25 0.5  0.75 1.  ]
```

`ones(dims)` creates arrays filled with ones, where `dims` is an integer or a tuple of integers that describes the dimensions of the array

In [8]:
```
# create a 1D array of length 3 filled with ones
print(np.ones(3))
```
```
[1. 1. 1.]
```

In [9]:
```
# tuples are used to create multi-dimensional arrays of ones
print(np.ones((3,4)))
```
```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

`zeros(dims)` creates arrays filled with zeros

In [10]:
```
# create a 3 x 3 array of zeros by passing a tuple as an argument
print(np.zeros((3,3)))
```
```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

`eye(N)` creates the $N \times N$ identity matrix

In [11]:
```
# create a 3 x 3 identity matrix
I = np.eye(3)
print(I)
```
```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Arrays of random numbers

There are several NumPy functions for creating arrays of random numbers

`random.random(dims)` creates an array with random numbers between 0 and 1 from a uniform distribution

In [13]:
```
# tuples are used to create random matrices
R = np.random.random((2, 2))
print(R)
```
```
[[0.69551691 0.95840253]
 [0.59427865 0.10006244]]
```

`random.randint(a, b, dims)` creates an array with random integers between a and $b - 1$

In [14]:
```
# create a vector with three random integers between 1 and 9
r = np.random.randint(1, 10, 3)
print(r)
```
```
[6 5 5]
```

In [15]:
```
# create a 3 x 2 matrix with random integers between 1 and 9
R = np.random.randint(1, 10, (3, 2))
print(R)
```
```
[[9 5]
 [6 4]
 [1 2]]
```

Operations on arrays

If we were using lists, then we'd have to use `for` loops or list comprehensions to carry out operations

In [16]:
```
l = [1, 2, 3, 4, 5, 6]
l2 = [e + 1 for e in l]
print(l2)
```
```
[2, 3, 4, 5, 6, 7]
```

With NumPy, such operations become trivial

In [17]:
```
l = np.array([1, 2, 3, 4, 5, 6])
l2 = l + 1
print(l2)
```
```
[2 3 4 5 6 7]
```

Example: Define the vectors $a = (1, 2, 3)$ and $b = (3, 2, 1)$. Compute $a + b$, $c = 0.5a$, and the dot product $a \cdot b$

In [18]:
```
# defining the vectors
a = np.array([1, 2, 3])
b = np.array([3, 2, 1])
```

In [19]:
```
# computing a + b and printing the result
print(a + b)
```
```
[4 4 4]
```

In [20]:
```
# computing c = 0.5a and printing the result
c = 0.5 * a
print(c)
```
```
[0.5 1.  1.5]
```

In [21]:
```
# computing a.b
print(np.dot(a, b))
```
```
10
```

Question: What happens if we multiply the two vectors a and b?

In [22]:
```
# printing a and b
print('a =', a, '\nb =', b)
print('a*b =', a * b)
```
```
a = [1 2 3]
b = [3 2 1]
a*b = [3 4 3]
```

Answer: The * operator performs element-by-element multiplication. The vectors must be the same size for this to work correctly

In [23]:
```
a = np.array([1, 2, 3])
c = np.array([1, 1, 1, 1])
a * c
```
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-23-032aee7d0f56> in <module>
      1 a = np.array([1, 2, 3])
      2 c = np.array([1, 1, 1, 1])
----> 3 a * c

ValueError: operands could not be broadcast together with shapes (3,) (4,)
```

Exercise:

Create a vector of length 20 where each entry is a uniformly distributed random number between 3 and 4

Solution:

In [9]:
```
# length of vector
N = 20

# create the vector and print
v = 3 * np.ones(N) + np.random.random(N)
print(v)
```
```
[3.18143449 3.93654101 3.49077634 3.68221602 3.91360176 3.94104543
 3.77369478 3.73381185 3.78941319 3.54143179 3.26961989 3.10603569
 3.69753823 3.38327862 3.47370143 3.38718846 3.88500859 3.30974368
 3.20560635 3.42833972]
```

Matrix operations

Matrices can be added using + and multiplied using @

Example: Consider the matrices

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 4 \\ 6 & 2 \end{pmatrix}$$

Compute $A + B$ and AB

In [24]:
```
A = np.array([[1, 2], [3, 4]])
B = np.array([[1, 4], [6, 2]])
```

In [25]:
```
print(A + B)
```
```
[[2 6]
 [9 6]]
```

In [26]:
```
print(A @ B)
```
```
[[13  8]
 [27 20]]
```

Warning: It is very tempting to use * for matrix multiplication, but this computes the element-wise product

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 4 \\ 6 & 2 \end{pmatrix}$$

In [27]:
```
print(A * B)
```
```
[[ 1  8]
 [18  8]]
```

Applying mathematical functions to arrays

NumPy comes with mathematical functions that can operate on arrays.

Example: compute $y = \sin(x)$ at 10 equally spaced points between 0 and 2π

In [28]:
```
x = np.linspace(0, 2 * np.pi, 10)
y = np.sin(x)
print(np.round(y,2))
```
```
[ 0.    0.64  0.98  0.87  0.34 -0.34 -0.87 -0.98 -0.64 -0.  ]
```

Other functions include cos, tan, arccos, arcsin, exp, log, and more

Linear algebra with NumPy

NumPy can perform some linear algebra calculations.

Example: Use `np.linalg.solve` to solve the system $Ax = b$ when

$$A = \begin{pmatrix} 1 & 2 \\ 4 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$$

In [29]:
```
# define A and b
A = np.array([[1, 2], [4, 1]])
b = np.array([3, 1])

# solve for the vector x and print the result
x = np.linalg.solve(A, b)
print(x)
```
```
[-0.14285714  1.57142857]
```

Summary

NumPy is a library for the creation and manipulation of arrays

It comes loaded with functions for operating on these arrays

It also has functions for linear algebra