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 [1]:
```
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 [3]:
```
print(a[0])
```
```
1
```

In [4]:
```
a[2] = 5.0
print(a)
```
```
[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

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

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, e.g. [row, column]

In [6]:
```
print(A[0,1])
```
```
2
```

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]:
```
x = np.linspace(0, 1, 5)
print(x)
```
```
[0.   0.25 0.5  0.75 1.  ]
```

`ones` creates arrays filled with ones

In [8]:
```
np.ones(3)
```
Out[8]:
```
array([1., 1., 1.])
```

In [9]:
```
np.ones((3,4)) # passing a tuple for a 2D array
```
Out[9]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

`zeros` creates arrays filled with zeros

In [10]:
```
np.zeros((3,3))
```
Out[10]:
```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

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

In [11]:
```
np.eye(3)
```
Out[11]:
```
array([[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` creates an array with random numbers between 0 and 1 from a uniform distribution

In [29]:
```
np.random.random(4)
```
Out[29]:
```
array([0.13036799, 0.06443515, 0.42820514, 0.26604323])
```

In [30]:
```
np.random.random((2, 2))
```
Out[30]:
```
array([[0.34070661, 0.3636238 ],
       [0.10869408, 0.06688991]])
```

`random.randint(a, b, dims)` creates an array of size *dims* with random integers between a and b

In [31]:
```
np.random.randint(1, 10, 3)
```
Out[31]:
```
array([4, 9, 3])
```

In [32]:
```
np.random.randint(1, 10, (3, 2)) # using tuple
```
Out[32]:
```
array([[7, 5],
       [8, 4],
       [4, 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 [12]:
```
l = [1, 2, 3, 4, 5, 6]
[e + 1 for e in l]
```
Out[12]:
```
[2, 3, 4, 5, 6, 7]
```

With NumPy, such operations become trivial

In [13]:
```
l = np.array([1, 2, 3, 4, 5, 6])
l + 1
```
Out[13]:
```
array([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 [14]:
```
a = np.array([1, 2, 3])
b = np.array([3, 2, 1])
```

In [15]:
```
a + b
```
Out[15]:
```
array([4, 4, 4])
```

In [16]:
```
c = 0.5 * a
print(c)
```
```
[0.5 1.  1.5]
```

In [17]:
```
a.dot(b)
```
Out[17]:
```
10
```

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

In [18]:
```
a * b
```
Out[18]:
```
array([3, 4, 3])
```

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

In [19]:
```
a = np.array([1, 2, 3])
c = np.array([1, 1, 1, 1])
a * c
```
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-19-032aee7d0f56> 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,)
```

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 [22]:
```
A = np.array([[1, 2], [3, 4]])
B = np.array([[1, 4], [6, 2]])
```

In [23]:
```
A + B
```
Out[23]:
```
array([[2, 6],
       [9, 6]])
```

In [24]:
```
A @ B
```
Out[24]:
```
array([[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 [25]:
```
A * B
```
Out[25]:
```
array([[ 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 [26]:
```
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 [27]:
```
A = np.array([[1, 2], [4, 1]])
b = np.array([3, 1])
x = np.linalg.solve(A, b)
print(x)
```
```
[-0.14285714  1.57142857]
```

We can check this answer by computing $Ax - b$, which should be zero

In [28]:
```
A @ x - b
```
Out[28]:
```
array([0., 0.])
```

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