Working with Scipy -- linear algebra in Python

Introducing SciPy

SciPy is a Python library that provides algorithms for a vast array of scientific computation. This primarily includes tools for integration, optimization, and solvers for eigenvalue problems, differential equations and several other problems in numerical computing.

See the tutorial below to learn more about SciPy:

https://docs.scipy.org/doc/scipy/tutorial/index.html

What is the difference with **NumPy**?

NumPy: Performs basic operations such as sorting, indexing, etc. Contains a variety of functions but these are not defined in depth. Arrays are multi-dimensional arrays of objects which are of the same type.

SciPy: Is built on NumPy. Used for complex operations such as algebraic functions, various numerical algorithms, etc. Contains detailed versions of the functions like linear algebra. Uses the concept of array and data structures from NumPy. Faster than NumPy for specialised operations.

1. Linear algebra

We use the **scipy.linalg** functions for simple linear algebra algorithms.

One can also use the **numpy.linalg** to perform similar functions. However, this is a subset of the SciPy functionalities.

See this page for more functions: https://numpy.org/doc/stable/reference/routines.linalg.html?highlight=np%20linalg

Determinant of a matrix

In [1]:
```
import numpy as np
from scipy import linalg as la

A = np.random.randint(5,size=(3, 3)) # Random array with shape (3,3) filled with integer elements from 0 to 5
print ('Matrix A is:\n', A,'\n')

print (np.linalg.det(A))  # Compute the determinant of a square matrix
print (la.det(A))
```

```
Matrix A is:
 [[2 0 4]
 [0 1 0]
 [0 0 3]]

6.0
6.0
```

Eigenvalue and eigenvector of a matrix

In [2]:
```
A = np.random.randint(5,size=(3, 3)) # Random array with shape (3,3) filled with integer elements from 0 to 5
print ('Matrix A is:\n', A,'\n')

e_val, e_vec = la.eig(A)  # Compute the eigenvalues and right eigenvectors of a square array
print (e_val)
print ()
print (e_vec)
```

```
Matrix A is:
 [[4 2 3]
 [1 3 2]
 [4 1 4]]

[8.33710855+0.j         1.33144573+0.16259521j 1.33144573-0.16259521j]

[[ 0.63794998+0.j          0.30326276-0.05166461j  0.30326276+0.05166461j]
 [ 0.37216758+0.j          0.64266625+0.09358944j  0.64266625-0.09358944j]
 [ 0.67417439+0.j         -0.69540174+0.j         -0.69540174-0.j        ]]
```

Multiplicative inverse of a matrix

In [3]:
```
A = np.random.randint(5,size=(3, 3)) # Random array with shape (3,3) filled with integer elements from 0 to 5
print ('Matrix A is:\n', A,'\n')

A_inv = la.inv(A)  # Compute the eigenvalues and right eigenvectors of a square array
print (A_inv)  # Compute the eigenvalues and right eigenvectors of a square array

A @ A_inv # Gives a numerically accurate identity matrix
```

```
Matrix A is:
 [[4 4 1]
 [1 4 0]
 [0 3 0]]

[[ 0.          1.         -1.33333333]
 [ 0.          0.          0.33333333]
 [ 1.         -4.          4.        ]]
```

Out[3]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Matrix exponentiation

e^A, where A is a square matrix

In [5]:
```
exp_A = la.expm(A)
print (exp_A,'\n')

B = np.zeros((2, 2))
print (la.expm(B))
```

```
[[225.9816958  465.54225729  37.79359773]
 [103.97630644 225.9816958   16.54567718]
 [ 49.63703153 113.38079318   8.62459618]]

[[1. 0.]
 [0. 1.]]
```

Several other intersting operations in the linear algebra package

Please go through each of them and practice:

https://docs.scipy.org/doc/scipy/reference/linalg.html

Note that you are expected to study these for future tests.

2. Numerical integration

Using the **integrate** package in SciPy

$$I = \int_{x=0}^{x=1} ax + bx^2 \, dx$$

scipy.integrate.quad(func, a, b)

In [ ]:
```
from scipy import integrate

def I(x, a, b):
    return a*x**2 + b

a,b = 1,0

integrate.quad(I,0,10,args=(a,b)) # Returns the integration with possible upper bound on the error
```

One can also **double** integrate

$$I = \int_{x=0}^{x=2} \int_{y=0}^{y=1} xy^2 \, dy \, dx$$

scipy.integrate.dblquad(func, a0, b0, a1, b1)

In [11]:
```
from scipy import integrate

def f(y,x):                             # Note the order of x and y, while defining the function
    return x*y**2

integrate.dblquad(f, 0, 2, 0, 1)        # The integration limits of x are given first, followed by limits of y
```

Out[11]:
```
(0.6666666666666667, 7.401486830834377e-15)
```

In [12]:
```
def f(y, x):
    return 1

integrate.dblquad(f, 0, np.pi/4, np.sin, np.cos)  # Note that limits of y can be two functions of x
```

Out[12]:
```
(0.41421356237309503, 1.1083280054755938e-14)
```

Trapezium or trapezoidal rule

https://en.wikipedia.org/wiki/Trapezoidal_rule

To integrate $I = \int_{x_0}^{x_1} f(x)\,dx$, the idea is to slice the function into several trapeziums and calculate the area.

For example, using a single trapezium:

$\int_{x_0}^{x_1} f(x) \approx (x_1 - x_0) \times \frac{1}{2}(f(x_1) + f(x_0))$

Using several trapeziums

$\int_{x_0}^{x_N} f(x) \approx \sum_{i=1}^{N} \frac{f(x_{i-1}) + f(x_i)}{2} \Delta x_i$

In [14]:
```
def f(x):
    return x**2

N = 1000
x_val = np.linspace(0,10,N)
dx = x_val[1]-x_val[0]

int_sum = sum([dx/2*(f(x_val[i])+f(x_val[i+1])) for i in range(len(x_val)-1)])

print (int_sum)
```

```
333.33350033383414
```

Monte Carlo intergration

https://en.wikipedia.org/wiki/Monte_Carlo_integration

Let us consider the integration:

$I = \int_\Omega f(x)dx$, where the volume, $V = \int_\Omega dx$.

In MC integration, we sample N uniformly random points $\{x_i\} \in \Omega$. In the limit of large N, we get

$I \approx \frac{V}{N} \sum_{i=1}^{N} f(x_i) \equiv V \times \langle f(x) \rangle.$

Let us consider, $f(x) = x^2$ between $x = 0$ and $x = 10$. Here, $v = 10 - 0 = 10$.

In [16]:
```
N = 100000
V = 10 - 0
x = np.random.uniform(0,10,N)  # we choose N points between 0 and 10

I = V/N*sum(f(x))

print (I)  # Check if this is correct

print (integrate.quad(f,0,10))
```

```
333.73607175889447
(333.33333333333337, 3.700743415417189e-12)
```

3. Tasks for today

Solve the following problems.

- **Newton-Raphson method to find the root of a function**

  The root of a function $f(x)$ is defined as $x = \alpha$, such that $f(\alpha) = 0$. The Newton-Raphson method helps find the root of a continuous function in an iterative manner. Starting with a guess x_n for the root, it predicts a better guess x_{n+1}, where

  $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

  Here, $f'(x)$ is the numerical derivate of the function $f(x)$, given by

  $$f'(x) \equiv \frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

  Find a function, that can be iteratively find the root of the function, $f(x) = \sin(x) + 3\cos(x)$, using the Newton-Raphson method upto an accuracy of 10^{-6}, starting from an initial random guess $x_0 \in [0, \pi]$. The function takes $f(x)$, x_0 and h, as its arguments. The output of the function should be a tuple containing the approximate root of $f(x)$ and the number of iterations n needed to find it.

  Hint: If the function is unable to find the root after a fixed number of iterations than you should restart it with a different initial guess.

- Using **matplotlib**, plot the function $f(x) = \sin(x) + 3\cos(x)$, in the range $x \in [0, 5\pi]$. Use atleast 1000 plot points. Compare this with the plot $f(x) = \cos(x)$.

- Integrate the function $f(x) = x^2$, using the trapezium and Monte-Carlo rule, between $x = 2$ and $x = 100$. Also, compare this with the **integrate.quad** function from SciPy.

- Using both trapezium and Monte-Carlo methods, integrate $x \log x$, between $x = 0$ and $x = 1$. Compare with exact and integrate.quad.

- Consider the **Zassenhaus formula** that is a successive approximation to the matrix exponent of $t(A + B)$. It is given by (to second order) by

  $$e^{t(A+B)} = e^{tA} e^{tB} e^{-\frac{t^2}{2}[A,B]} \dots .$$

  Generate two random 3 x 3 matrices A and B, and using SciPy write a code that verifies the Zassenhaus formula for a small value of t. Show that the approximation is better for smaller t. Also, show that if you set either A or B as the identity matrix, then the relation holds for all value of t.

- The composite Simpson's rule is a numerical integration technique that builds on the trapezoidal method. Here, the integration interval $[a, b]$ is divided into n equal and even divisions (**be careful here**), such that the integration is given by,

  $$\begin{aligned}\int_a^b f(x)dx &\approx \frac{\Delta x}{3} \sum_{\substack{j=0 \\ j\ even}}^{n-2} \left[f(x_j) + 4f(x_{j+1}) + f(x_{j+2})\right] \\ &= \frac{\Delta x}{3}[f(x_0) + 4 \sum_{\substack{j=1 \\ j\ odd}}^{n-1} f(x_j) + 2 \sum_{\substack{j=2 \\ j\ even}}^{n-2} f(x_j) + f(x_n)],\end{aligned}$$

where $\Delta x = x_{i+1} - x_i,\ \forall\ i \in [0, n)$. Write a function that implements the Simpson's method and integrate the function $f(x) = x^2 \log(x)$, between $x = 1$ and $x = 10$.