DOING PHYSICS WITH PYTHON

Ian Cooper

matlabvisualphysics@gmail.com

QUANTUM MECHANICS
A NUMERICAL APPROACH
FIRST AND SECOND DERIVATIVES AS OPERATORS

DOWNLOAD DIRECTORIES FOR MATLAB SCRIPTS

qm003B.py

Google drive

GitHub

Operators play a significant role in quantum mechanics. Operators in quantum mechanics are mathematical entities used to represent physical processes that result in the change of the state vector of the system. In this article, we will consider operators as matrices to compute the first derivative and second derivates of functions of the form $f(x)$.

FIRST AND SECOND DERIVATIVES OF A ONE VARIABLE FUNCTION

Consider the one variable function $y(x) \equiv f(x)$. Then the **first derivative** of the function $y(x)$ is

$$(1) \qquad \frac{dy(x)}{dx} \equiv \frac{dy}{dx} = \lim_{\Delta x \to 0}\left(\frac{y(x+\Delta x) - y(x)}{\Delta x}\right)$$

The first derivative tells us how rapidly the function $y(x)$ varies when we change the value of x by a tiny amount dx.

$$(2) \qquad dy = \left(\frac{dy}{dx}\right)dx$$

If we change x by the amount dx then the change in y is dy and is proportional to dx with the constant of proportionality equal to the first derivative dy/dx. Graphically the first derivative gives the **slope** or **gradient** of the tangent of the curve $y(x)$ verses x.

We can approximate the first derivative at x by the following:

$$(3a) \qquad \left.\frac{dy}{dx}\right|_x \approx \frac{y(x+\Delta x) - y(x)}{\Delta x} \qquad \text{forward approximation}$$

$$(3b) \qquad \left.\frac{dy}{dx}\right|_x \approx \frac{y(x) - y(x-\Delta x)}{\Delta x} \qquad \text{backward approximation}$$

$$(3c) \qquad \left.\frac{dy}{dx}\right|_x \approx \frac{y(x+\Delta x) - y(x-\Delta x)}{2\Delta x} \qquad \text{central difference approximation}$$

Mathematically, equations 3 are only correct in the limit $\Delta x \to 0$. In calculating derivatives numerically, Δx has to be small enough to provide sufficient accuracy of the result. There is an optimal value for Δx, since, if Δx is too small you get round-off errors. For most applications, the central difference approximation is preferred over the forward or backward methods. It is more difficult to achieve good accuracy in numerical differentiation compared to numerical integration. The main reason comes from the fact that we are taking the ratio of two differences.

The second derivative can be approximated by the equation

$$(4) \qquad \left.\frac{d^2y}{dx^2}\right|_n \approx \frac{y(n+1)-2y(n)+y(n-1)}{\Delta x^2}$$

DIFFERENTIATION USING MATRICES

The process of differentiation can be considered as an operator acting upon a function. A differential operator is represented by a square *N*x*N* matrix that acts on the function **y** given by a *N*x1 column vector to give the derivative of the function **y'** also as an *N*x1 column vector.

The matrix elements can be easily deduced from the definition of the first derivative and second derivative approximations.

The first derivative matrix equation

$$\begin{bmatrix} y'_1 \\ y'_2 \\ y'_3 \\ \dots \\ y'_{N-1} \\ y'_N \end{bmatrix} = \frac{1}{2\Delta x} \begin{bmatrix} -2 & 2 & 0 & \dots & 0 & 0 \\ -1 & 0 & 1 & \dots & 0 & 0 \\ 0 & -1 & 0 & \dots & 0 & 0 \\ \dots & & & & & \\ 0 & 0 & \dots & -1 & 0 & 1 \\ 0 & 0 & \dots & 0 & -2 & 2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_{N-1} \\ y_N \end{bmatrix}$$

The second derivative matrix equation

$$\begin{bmatrix} y''_1 \\ y''_2 \\ y''_3 \\ \dots \\ y''_{N-1} \\ y''_N \end{bmatrix} = \frac{1}{\Delta x^2} \begin{bmatrix} 1 & -2 & 1 & \dots & 0 & 0 \\ 1 & -2 & 1 & \dots & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ \dots & & & & & \\ 0 & 0 & \dots & -2 & 0 & 1 \\ 0 & 0 & \dots & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_{N-1} \\ y_N \end{bmatrix}$$

The differentiation processes can be written as functions in Python (**qm003B.py**)

```
def firstDer(N,dx):
    v  = ones(N-1)
    M1 = diag(-v,-1)
    M2 = diag(v,1)
    M = M1+M2
    M[0,0] = -2; M[0,1] = 2; M[N-1,N-2] = -2; M[N-1,N-1] = 2
    MF = M/(2*dx)
    return MF

def secondDer(N,dx):
     v = -2*ones(N)
     M1 = np.diag(v)
     v = np.ones(N-1)
     M2 = np.diag(v,1)
     M3 = np.diag(v,-1)
     M = M1+M2+M3
     M[0,0] = 1; M[0,1] = -2; M[0,2] = 1
     M[N-1,N-3] = 1; M[N-1,N-2] = -2; M[N-1,N-1]=1
     MS = M/(dx**2)
     return MS
```

Example **qm003B.py**

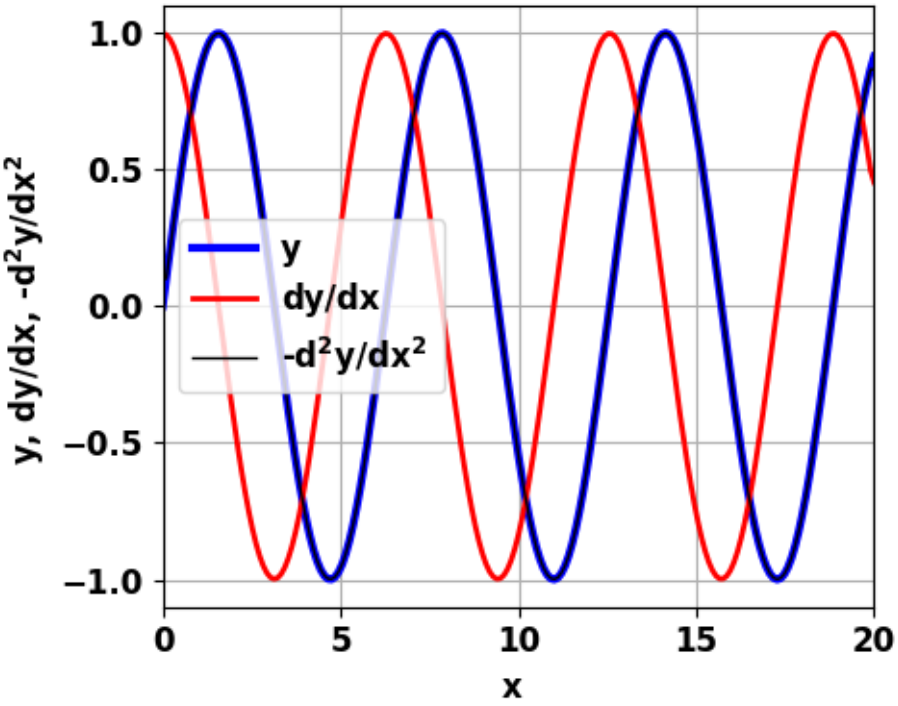
The first and second derivatives of the function

$$y = \sin(x) \quad N = 199 \quad x \in [0, 20]$$

```
#%% Function to be differentiated
N = 199                 # Number of array elements
x1 = 0; x2 = 20         # Function limits
x = linspace(x1,x2,N)   # x array
dx = x[2]-x[1]
y  = sin(x)             #  Function to be differentiated
MF = firstDer(N,dx)     #  1st derivative matrix
FD = MF@y               # 1st derivative
MS = secondDer(N,dx)    # 2nd derivative matrix
SD = MS@y               # Second derivative


#%%  GRAPHICS
plt.rcParams['font.size'] = 12
plt.rcParams["figure.figsize"] = (5,4)
fig, ax = plt.subplots(1)
ax.xaxis.grid(); ax.yaxis.grid()
ax.set_ylabel('y, dy/dx, -d$^2$y/dx$^2$ ',color= 'black')
ax.set_xlabel('x ',color = 'black')
ax.set_xlim([0, 20]); ax.set_ylim([-1.1, 1.1])
```

```
ax.set_yticks(np.arange(-1,1.2,0.50))
fig.tight_layout()
ax.plot(x,y,'b', lw = 3, label = 'y')
ax.plot(x,FD,'r',lw = 2, label = 'dy/dx')
ax.plot(x,-SD,'k',lw = 1, label = '-d$^2$y/dx$^2$ ' )
ax.legend()
# fig.savefig('a1.png')
```



Analytic derivatives

$$y = \sin(x) \quad dy/dx = \cos(x) \quad -d^2y/dx^2 = \sin(x) = y$$

So, with N = 199, there is excellent agreement between the numerical estimates for the derivatives compared to the analytical results. If N is too small, then there will be poor agreement.