Computational Methods

(Course code: CCC.528)

Record of Practical Work

Submitted by

Piyush R.Maharana
(21msphcp02**)**

for the degree of

M.Sc. Computational Physics

Department of Computational Sciences
School of Basic Sciences
Central University of Punjab
Bathinda-151 401, Punjab, India

Jan 2023

List of Exercises

Languages used: **Python and FORTRAN**

S.No.	Title of the Exercise	Signature
1	**Numerical Differentiation:** *Central,Backward Forward*	
2	**Numerical Integration:** *Simpson,Trapezoidal,Quadrature,Monte-Carlo*	
3	**Interpolation:** *Lagrange,Cubic Spline*	
4	**Matrix Methods:** *Gauss Seidel,Gauss Jacobi*	
5	**Root Finding:** *Bisection,Newton-Raphson,Fixed Point,Secant,Chebyshev,Halley*	
6	**Gradient Descent:** *1D and 2D*	
7	**Polynomial evaluation:** *Horner's method*	
8	**Partial Differential Equations:** *2D Heat Equation*	

Piyush R.Maharana
M.Sc Computational Physics
21msphcp02

Numerical Differentiation

Forward

$$f'(x_j) = \frac{f(x_{j+1}) - f(x_j)}{h} + O(h)$$

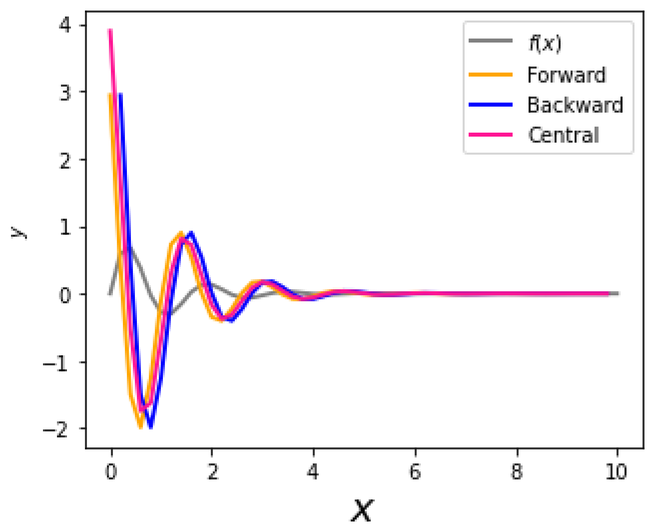
Backward

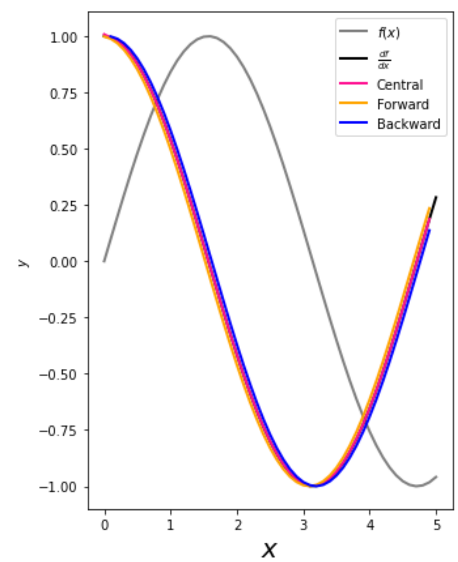
$$f'(x_j) = \frac{f(x_j) - f(x_{j-1})}{h} + O(h)$$

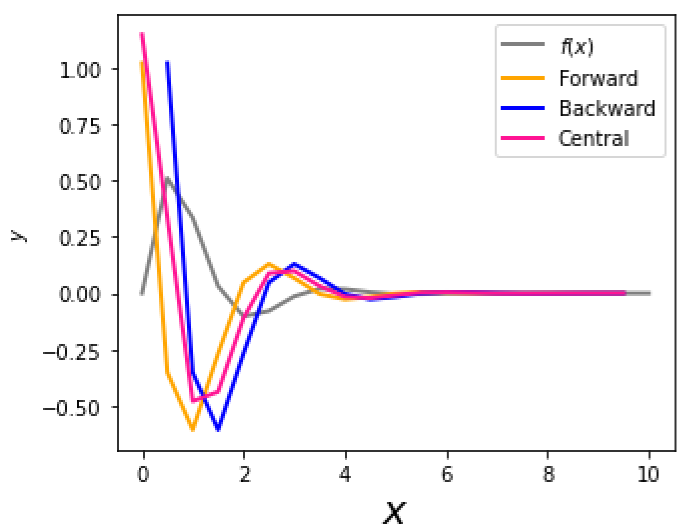
Central

$$f'(x_j) = \frac{f(x_{j+1}) - f(x_{j-1})}{h} + O(h^2)$$

```python
import numpy as np
import matplotlib.pyplot as plt
x=[] ; y=[]
lower=0 ; upper=10; step=0.2
N=int(((upper-lower)/step))
dy = [0]*N ; back=[0]*N ; forw=[0]*N
secder =[0]*N
def f(x):
  return np.sin(4*x)*np.exp(-x)
for i in range(0,N+1):
  x_step = lower+i*step
  x.append(x_step)
  y.append(f(x[i]))
for i in range(1,N-1):
 #derivative nikalo
  dy[i]=(y[i+1] - y[i-1]) / (x[i+1] - x[i-1])  #central difference formula
for i in range(1,N):
  back[i]=(y[i] - y[i-1]) / (x[i] - x[i-1])    #backward difference
for i in range(0,N):
  forw[i]=(y[i+1] - y[i]) / (x[i+1] - x[i])    #forward difference
#for i in range(0,N):
#  secder[i]=(y[i+2]-2*y[i+1] - y[i])/((x[i+1] - x[i])**2)
#interpolate karo bhai log
dy[0] = dy[1] + (dy[2]-dy[1])/(x[2]-x[1])*(x[0]-x[1])
dy[N-1] = dy[N-2] + (dy[N-2]-dy[N-3])/(x[N-2]-x[N-3])*(x[N-1]-x[N-2])
plt.figure(figsize=(5,4))
print('h=',step)
plt.plot(x,y,color='grey',linewidth=2.0,label=r'$f(x)$')
plt.plot(x[:-1],forw,color='orange',linewidth=2.0,label='Forward')
plt.plot(x[1:-1],back[1:],color='blue',linewidth=2.0,label='Backward')
plt.plot(x[:-1],dy,color='deeppink',linewidth=2.0,label='Central')
plt.xlabel('$x$',fontsize=20)
plt.ylabel('$y$')
plt.legend(loc='upper right')
```



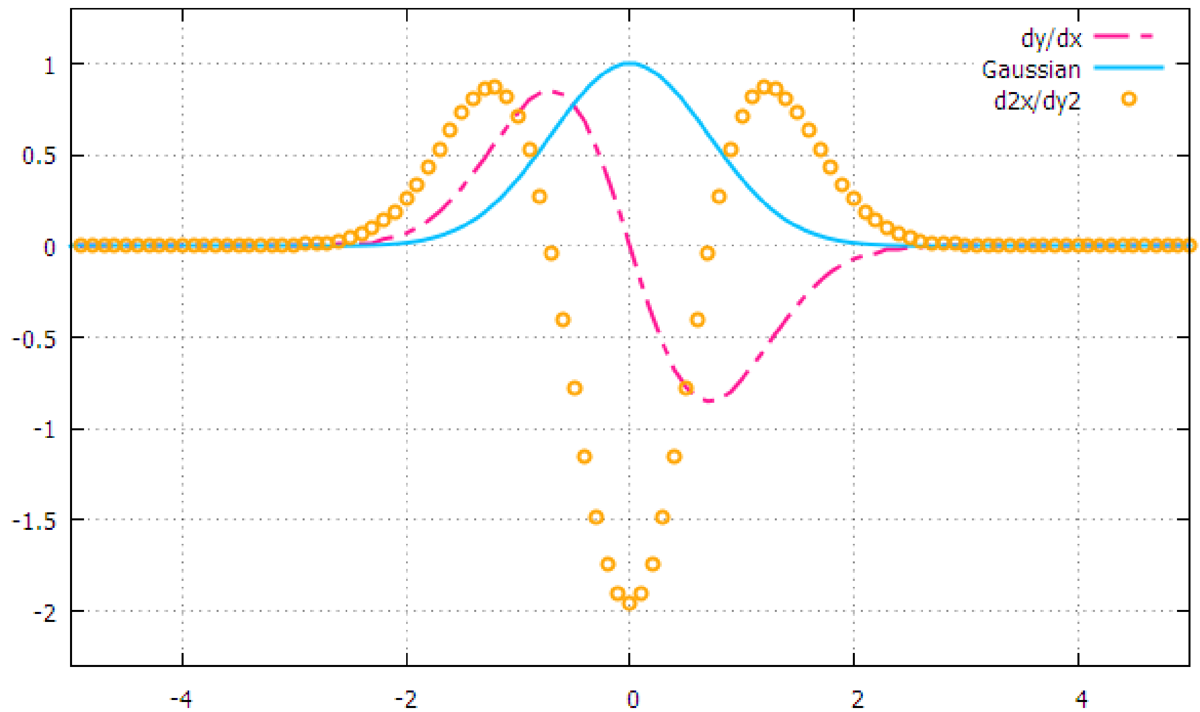


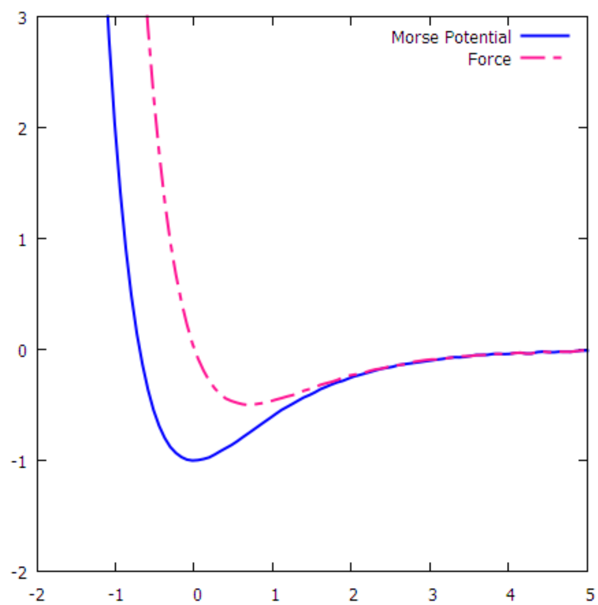


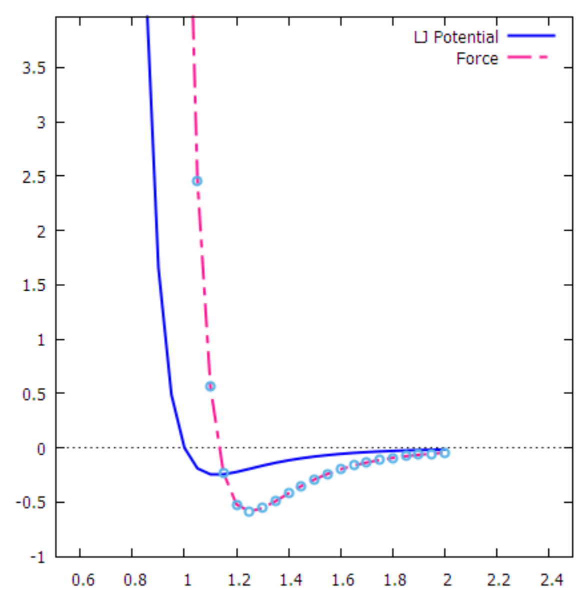

```
program deriv
  implicit none
  ! declare variables
  integer              :: i,upper,lower,N
  real                 :: step
  !real, dimension(N) :: x, y, dy
  real, allocatable :: x(:), y(:),dy(:),dy2(:),diff(:)

  print*,'Enter value of upper'
  read*, upper
  print*,'Enter value of lower'
  read*, lower
  print*,'Enter step size'
  read*,step
  N=(upper-lower)/step
```

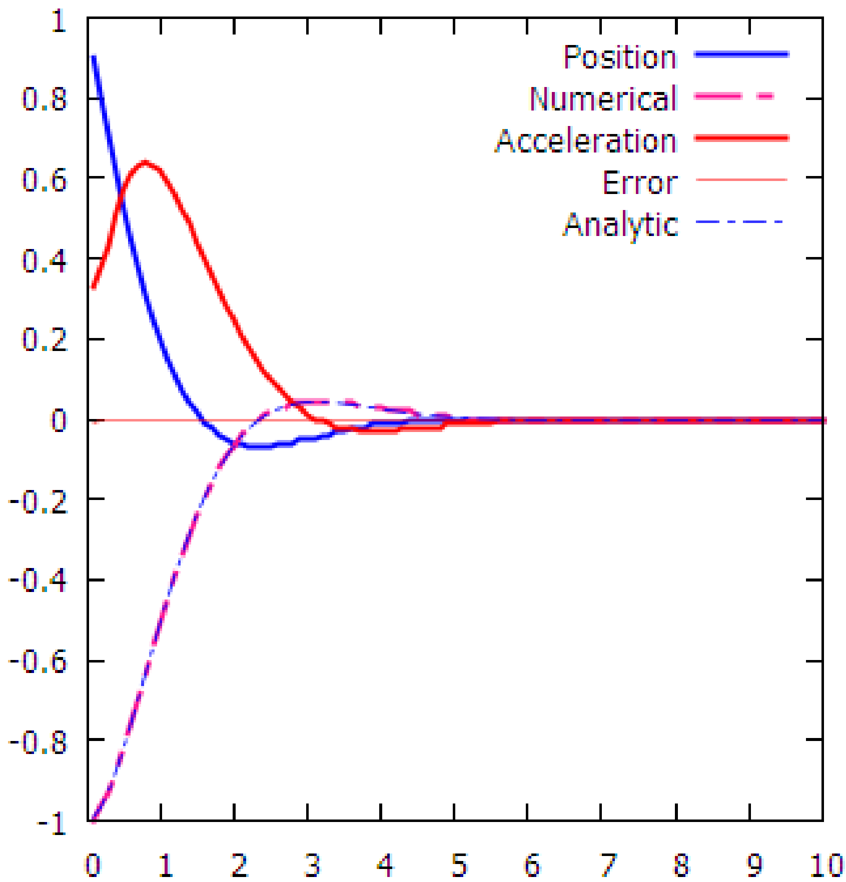
```
  allocate (x(N),y(N),dy(N),dy2(N),diff(N))    !allocate array size
  !assign data
  do i  =1,N
   x(i) = lower+i*step     !fill grid spaces
  end do
  !x = (/ 0., 0.1, 0.2, 0.3, 0.5, 0.6, 0.8, 0.9, 1. /)    !Array constructor
  do i  =1,N
   y(i) = EXP(-(x(i)**2))
  end do
  ! compute derivation
  do i = 2, N-1
     dy(i) = (y(i+1) - y(i-1)) / (x(i+1) - x(i-1))  !derivative of function
  end do
  ! compute first and last derivation using linear extrapolation
  dy(1) = dy(2) + (dy(3)-dy(2))/(x(3)-x(2))*(x(1)-x(2))
  dy(N) = dy(N-1) + (dy(N-1)-dy(N-2))/(x(N-1)-x(N-2))*(x(N)-x(N-1))
  do i = 1, N
    diff(i)=dy(i)-((-2*x(i))*EXP(-(x(i)**2)))
  end do
 !print the results
 !write (*,'(4a10)') 'x', 'y=f(x)', 'dy/dx','d2y/dx2'
 write (*,'(5a10)') 'x', 'y=f(x)', 'dy/dx','diff','d2y/dx2'
  do i = 1, N
    write(*,'(5f10.2)') x(i), y(i), dy(i),diff(i),dy2(i)
  end do
end program deriv
```

```
  !Double Derivative
  do i = 2, N-1
     dy2(i) = (dy(i+1) - dy(i-1)) / (x(i+1) - x(i-1))  !derivative
  end do
  !Linear extrapolation
  dy2(1) = dy2(2) + (dy2(3)-dy2(2))/(x(3)-x(2))*(x(1)-x(2))
  dy2(N) = dy2(N-1) + (dy2(N-1)-dy2(N-2))/(x(N-1)-x(N-2))*(x(N)-x(N-1))
  !Error calculator
  do i = 1, N
    diff(i)=dy(i)-((-2*x(i))*EXP(-(x(i)**2)))
  end do
```


dy/dx
Gaussian
d2x/dy2


Morse Potential
Force

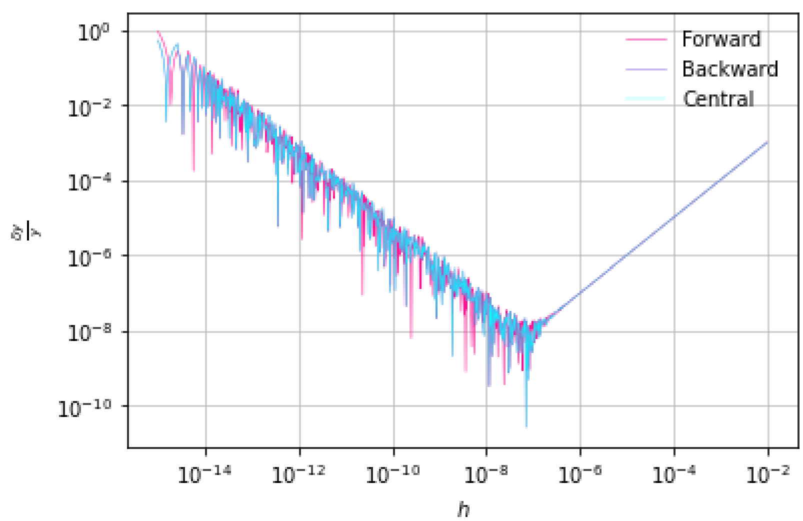

LJ Potential
Force



$$\mathbf{x(t) = e^{-x}\cos(x)}$$

```
#Nov 19,2022
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    """our function to numerically differentiate"""
    return x*x*x
def forwi(x, h):
    """a discrete approximation to the derivative at x"""
    return (f(x+h) - f(x))/h
def backi(x, h):
    return (f(x) - f(x-h))/h
def centri(x, h):
    return (f(x) - f(x-h))/h
def fprime(x):
    return 3*x*x
hs = np.logspace(-15, -2, 1000)   # generate a set of h's from 1.e-16 to 0.1
#x = np.pi/3.0                     # we'll look at the error at pi/3
x=10.0
```

```
forward  = forwi(x, hs)           #compute the numerical difference for all h's
backward = backi(x, hs)
central  = centri(x,hs)
ans = fprime(x)                  # get the analytic derivative
err1 = np.abs(forward  - ans)/ans  # compute the relative error
err2 = np.abs(backward - ans)/ans
err3 = np.abs(central  - ans)/ans
#fig = plt.figure()               # plot the error vs h
#ax = fig.add_subplot(111)
plt.grid(alpha=0.69)
plt.loglog(hs, err1,color='deeppink',linewidth=0.5,label='Forward')
plt.loglog(hs, err2,color='mediumpurple',linewidth=0.5,label='Backward')
plt.loglog(hs, err3,color='cyan',linewidth=0.3,label='Central')
plt.legend(loc='upper right',frameon=False)
plt.xlabel(r'$h$')
plt.ylabel(r'$\frac{\delta y}{y}$')
```



Numerical Integration

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

```
def f(x):
    return 1 + 0.25 * x * np.sin(np.pi * x)
a = 0.5
b = 1.5
# compute the 3 different approximations
I_r = f(a) * (b - a)                                  #rectangular
I_t = 0.5 * (f(a) + f(b)) * (b - a)                   #trapezoidal
I_s = (b - a) / 6.0 * (f(a) + 4 * f((a + b)/2) + f(b))   #Simpson
I_a = 1 - 1/(2 * np.pi**2)                            #analytical
```

```
print(I_r, I_t, I_s, I_a)
```

Trapezoid

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} h\frac{f(x_i)+f(x_{i+1})}{2}$$

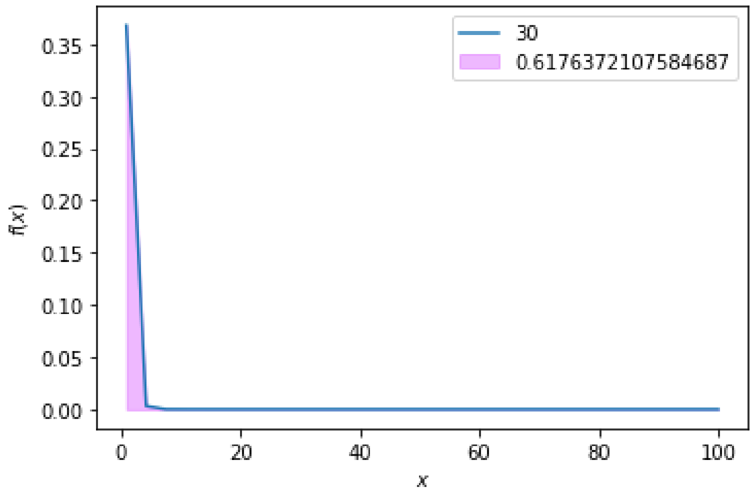
$$\int_a^b f(x)dx \approx \frac{h}{2}\left(f(x_0)+2\left(\sum_{i=1}^{n-1} f(x_i)\right)+f(x_n)\right).$$

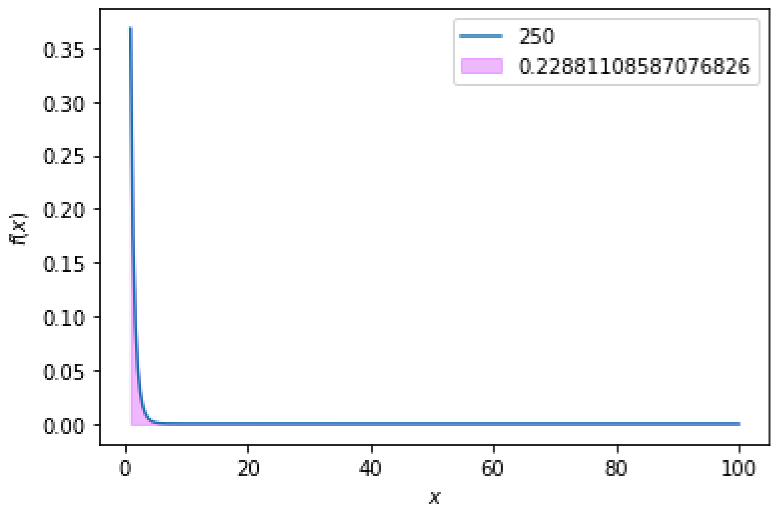
```
#Last update Piyush 22 Dec 2022
PROGRAM numtrapz
IMPLICIT NONE
real,allocatable :: x(:),y(:)
integer          :: N,i,num
real             :: sumtrap
!counter gin bhai
N = 0
OPEN (1, file ='data.txt')
DO
    READ (1,*, END=10)
    N = N + 1
END DO
10 CLOSE (1)
!print*,N
allocate (x(N),y(N))  !allocate the arrays from the text file
!Reading the coefficients from the file
open (unit = 1, file ='data.txt', status ='old')

DO i = 1,N
     read (1,*) x(i),y(i)
END DO
close (1)
DO i = 1,N
    write(*,*) x(i),y(i)
END DO

sumtrap =0
DO i=2,N
   sumtrap = sumtrap+ 0.5*(y(i-1)+y(i))*(x(i)-x(i-1))
END DO
WRITE(*,*)sumtrap
```

```
END PROGRAM numtrapz
```

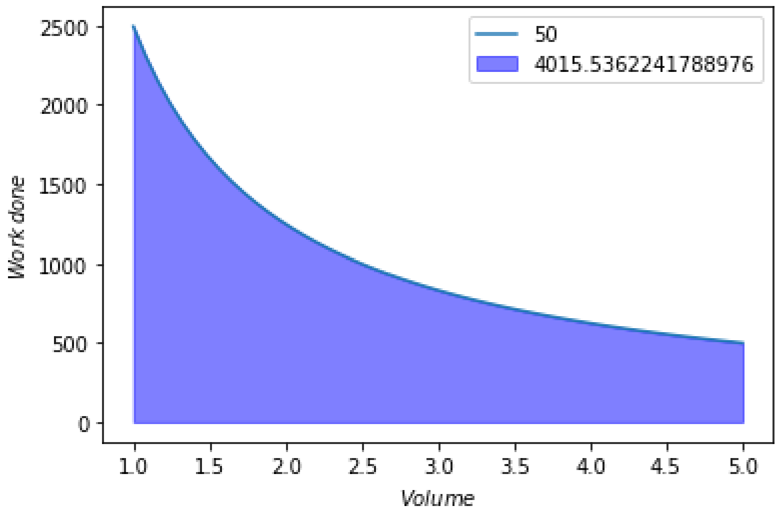
```python
import numpy as np
import matplotlib.pyplot as plt
a = 1; b = 100
N = 250
x = np.linspace(a,b,N+1)
#y = 1 + 0.25*x*np.sin(np.pi*x);
y  = np.exp(-x)/x
y_right = y[1:] #Riemann Right sum
y_left = y[:-1] #Riemann Left sum
dx = (b - a)/N
A = (dx/2)*np.sum(y_right + y_left)
print("A =",A)
plt.plot(x,y,label=N)
plt.fill_between(x,y,color='#DF73FF',alpha=0.5,label=A)
plt.xlabel(r'$x$')
plt.ylabel(r'$f(x)$')
plt.legend()
```

$$\int_1^{100} \frac{e^{-x}}{x} dx \approx 0.219$$






```
V1 = 1
V2 = 5
n = 1; R = 8.314; T = 300
N = 50
V = np.linspace(V1,V2,N+1)
W = (n*R*T)/V
y_right = W[1:]
y_left = W[:-1]
dx = (V2 - V1)/N
A = (dx/2)*np.sum(y_right + y_left)
print("A =",A)
plt.plot(V,W,label=N)
plt.fill_between(V,(n*R*T)/V,color='blue',alpha=0.5,label=A)
plt.xlabel(r'$Volume$')
plt.ylabel(r'$Work\:done$')
plt.legend()
```



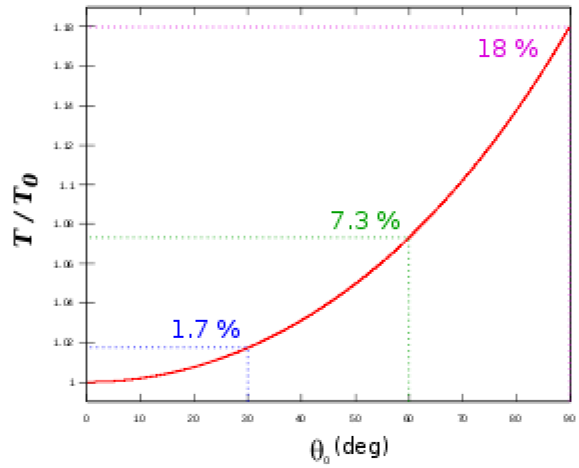
$$W = \int_{V_1}^{V_2} \frac{nRT}{V} dV$$

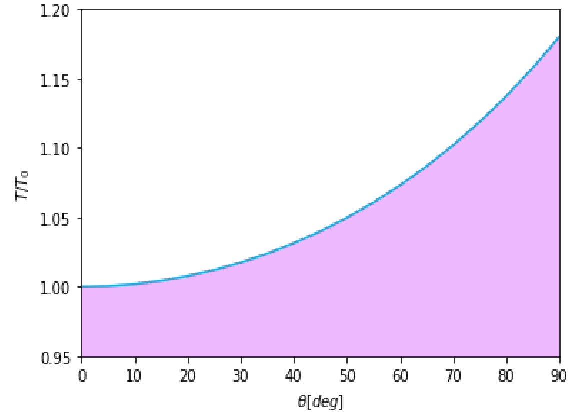
Solving the elliptic integral for pendulum motion

$$T = \frac{2T_0}{\pi} K(k), \qquad \text{where} \quad k = \sin\frac{\theta_0}{2}.$$

$$K(k) = F\left(\frac{\pi}{2}, k\right) = \int_0^{\frac{\pi}{2}} \frac{du}{\sqrt{1 - k^2 \sin^2 u}}.$$

```
import numpy as np
import matplotlib.pyplot as plt
pi =3.14
a = 0; b = pi/2
N = 50
x = np.linspace(a,b,N+1)
tau=[];x1=[]
for angle in range(0,91,5):
 k = np.sin(angle*0.0174533/2)
 y  = (1-((k**2)*np.sin(x)**2))**(-0.5)
 y_right = y[1:] #Riemann Right sum
 y_left = y[:-1] #Riemann Left sum
 dx = (b - a)/N
 A = (dx/2)*np.sum(y_right + y_left)
 tau.append((2/pi)*A)
 x1.append(angle)

plt.ylim([0.95,1.2])
plt.xlim([0,90])
plt.fill_between(x1,tau,color='#DF73FF',alpha=0.5)
plt.xlabel(r'$\theta [deg]$')
plt.ylabel(r'$T/T_{0}$')
```





Power series solution for the elliptic integral

$$\sin\frac{\theta_0}{2} = \frac{1}{2}\theta_0 - \frac{1}{48}\theta_0^3 + \frac{1}{3\,840}\theta_0^5 - \frac{1}{645\,120}\theta_0^7 + \cdots.$$

$$T = 2\pi\sqrt{\frac{\ell}{g}}\left(1 + \frac{1}{16}\theta_0^2 + \frac{11}{3\,072}\theta_0^4 + \frac{173}{737\,280}\theta_0^6 + \frac{22\,931}{1\,321\,205\,760}\theta_0^8 + \frac{1\,319\,183}{951\,268\,147\,200}\theta_0^{10} + \frac{233\,526\,463}{2\,009\,078\,326\,886\,400}\theta_0^{12} + \cdots\right),$$

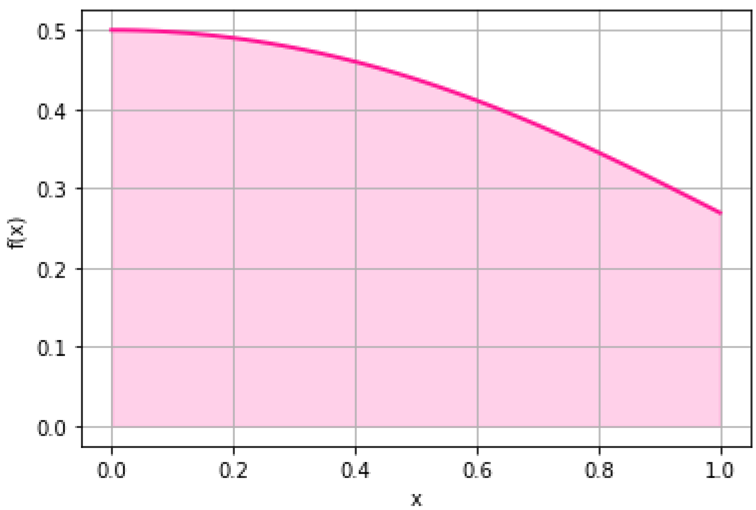
Simpson

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 4\left(\sum_{i=1,i\ \text{odd}}^{n-1} f(x_i)\right) + 2\left(\sum_{i=2,i\ \text{even}}^{n-2} f(x_i)\right) + f(x_n)\right]$$

```
import numpy as np
import matplotlib.pyplot as plt
a = 0 ; b =1
n = 51 ;h = (b-a)/(n-1)
x = np.linspace(a, b, n)

f = 1/(1+np.exp(x**2))
I_simp  = (h/3) * (f[0] + 4*sum(f[1:n-1:2]) + 2*sum(f[:n-2:2]) + f[n-1])
I_trap  = (h/2)*(f[0]+2*sum(f[1:n-2:1])+f[n-1])

I_ana    = 0.41946
err_simp =  I_ana - I_simp
err_trap =  I_ana - I_trap

plt.plot(x,f,color='deeppink',linewidth=2.0,label='f(x)')
plt.fill_between(x, f, 0, color='deeppink', alpha=.2)
plt.xlabel('x')
plt.ylabel('f(x)')
```

```
plt.grid()
print('I_s =',I_simp)
print('I_t =',I_trap)
print('Error_I_s=',err_simp)
print('Error_I_t=',err_trap)
```

```
I_s = 0.4261331478942992

I_t = 0.4139174099321641

Error_I_s= -0.006673147894299181

Error_I_t= 0.005542590067835917
```



$$\int_0^1 \frac{1}{1+e^{x^2}} dx \approx 0.41946$$

$$\begin{aligned}\int_a^b f(x)\,dx &\approx \frac{3}{8}h\sum_{i=1}^{n/3}\left[f(x_{3i-3})+3f(x_{3i-2})+3f(x_{3i-1})+f(x_{3i})\right]\\ &= \frac{3}{8}h\big[f(x_0)+3f(x_1)+3f(x_2)+2f(x_3)+3f(x_4)+3f(x_5)+2f(x_6)+\cdots+2f(x_{n-3})+3f(x_{n-2})+3f(x_{n-1})+f(x_n)\big]\\ &= \frac{3}{8}h\left[f(x_0)+3\sum_{i=1,\ 3\nmid i}^{n-1} f(x_i)+2\sum_{i=1}^{n/3-1} f(x_{3i})+f(x_n)\right].\end{aligned}$$

```
#simpson 3/8
PROGRAM simpsimp38
IMPLICIT NONE
INTEGER::j,n
REAL::a,b,h,I,f

a=0;b=1;n=100
```

```fortran
h=(b-a)/n
I=f(a)+f(b)
DO j=1,n-1
  IF (MOD(j,3)==0) THEN
    I=I+(2*f(a+(j*h)))
  ELSE
    I=I+(3*f(a+(j*h)))
  END IF
END DO
I=(3/8.)*h*I

WRITE(6,9) "I=",I
9 FORMAT (a,F9.6)
END PROGRAM

REAL function f(x1)
  f=x1**2
return
end function
```

Quadrature

Gauss Legendre

```python
#gauss quadrature #general mode
import numpy as np
def f(x):
    return 1/(x**2+5)
a = 0
b = 2*3.14
x_1 = [0]                             ; w_1 = [2]
x_2 = [-0.577,0.577]                  ; w_2 = [1.000,1.000]
x_3 = [0.000,-0.744,0.744]            ; w_3 = [0.888,0.555,0.555]
x_4 = [-0.339,0.339,-0.861,0.861]  ; w_4 = [0.652,0.652,0.347,0.347]
x_5=  [-0.90618,-0.538469,0,0.538469,0.909618]  ; w_5 =
[0.236927,0.478629,0.56889,0.478629,0.236927]
integral=0
N=4
if N == 1:
    integral += w_1[0]*(b-a)*0.5*(f((0.5*((b-a)*x_1[0]+(b+a)))))
elif N == 2:
  for i in range(len(x_2)):
      integral += +w_2[i]*(b-a)*0.5*(f((0.5*((b-a)*x_2[i]+(b+a)))))
elif N == 3:
  for i in range(len(x_3)):
      integral += w_3[i]*(b-a)*0.5*(f((0.5*((b-a)*x_3[i]+(b+a)))))
```

```python
elif N == 4:
    for i in range(len(x_4)):
      integral += w_4[i]*(b-a)*0.5*(f((0.5*((b-a)*x_4[i]+(b+a)))))
else :
    for i in range(len(x_5)):
      integral += w_5[i]*(b-a)*0.5*(f((0.5*((b-a)*x_5[i]+(b+a)))))
print(integral)
```

FORTRAN:

```fortran
#FORTRAN gauss code
program gaussquad
  implicit none
  ! declare variables
  integer            :: i,N
  real               :: f,integral,a,b
  real, dimension(1) :: x_1,w_1
  real, dimension(2) :: x_2,w_2
  real, dimension(3) :: x_3,w_3
  real, dimension(4) :: x_4,w_4

  x_1=(/0/)                            ; w_1 = (/2/)
  x_2=(/-0.577,0.577/)                 ; w_2 = (/1.000,1.000/)
  x_3=(/0.000,-0.744,0.744/)           ; w_3 = (/0.888,0.555,0.555/)
  x_4=(/-0.339,0.339,-0.861,0.861/) ; w_4 = (/0.652,0.652,0.347,0.347/)

  a = 0.0
  b = 3.0
  N = 4
  do i = 1,N
     integral = integral+ w_4(i)*(b-a)*0.5*(f((0.5*((b-a)*x_4(i)+(b+a)))))
  end do
 write (6,9) 'I=',integral
9 FORMAT(a3,f10.6)
end program gaussquad

REAL function f(x1)
REAL::x1
  f=x1*EXP(x1)
return
end function
```

Other quadrature methods can be implemented too just by changing the weights in the initial lists

For example Gauss-Chebyshev and Gauss-Hermite.

```python
#Gauss Hermite quadrature
import numpy as np
import matplotlib.pyplot as plt
```

```
def f(x):
   return 1/(1+x**2)
a = 0
b = 1
x_2 = [-0.7071068,0.7071068]                        ; w_2 = [0.8862269,0.8862269]
x_3 = [0.0000,1.2247449,-1.2247449]                 ; w_3 = [1.1816359,0.2954090,-0.2954090]
x_4 = [0.5246476,-0.5246476,1.6506801,-1.6506081]  ; w_4 = [0.8049141,0.8049141,0.0813128,0.0813128]
```

```
#Gauss Chebyshev quadrature
import numpy as np
import matplotlib.pyplot as plt

def f(x):
   return 1/(1+x**2)
a = 0
b = 1
x_2 = [-0.7071068,0.7071068]                        ; w_2 = [1.5707963,1.5707963]
x_3 = [0.0000,0.8660254,-0.8660254]                 ; w_3 = [1.0471976,1.0471976,1.047196]
x_4 = [0.3826834,-0.3826834,0.9238795,-0.9238795]  ; w_4 = [0.7853982,0.7853982,0.7853982,0.7853982]
```

$$\int_0^3 \frac{1}{2+x^2}dx \approx 0.79923265$$

N=4

Gauss-Legendre **0.7977961**

Gauss-Hermite **1.298547**

Gauss-Chebyshev **0.696479**

$$\int_{0.5}^{1.5} 1+\frac{x}{4}\sin(\pi x)dx \approx 0.949$$

Gauss-Legendre **0.948779**

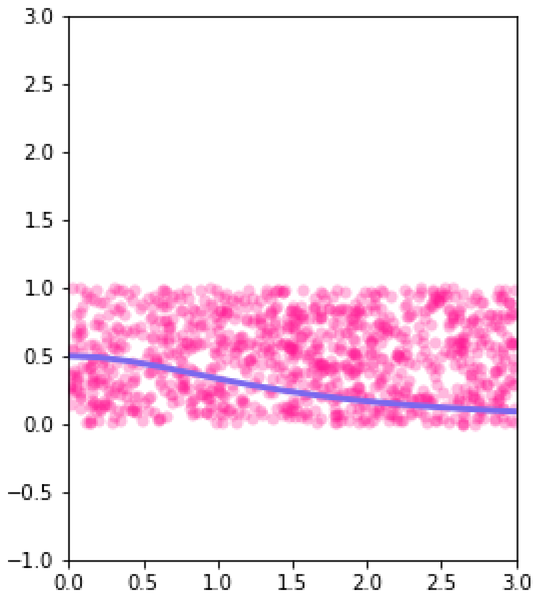
Gauss-Hermite **0.83891978**

Gauss-Chebyshev **1.4598072**

Monte-Carlo

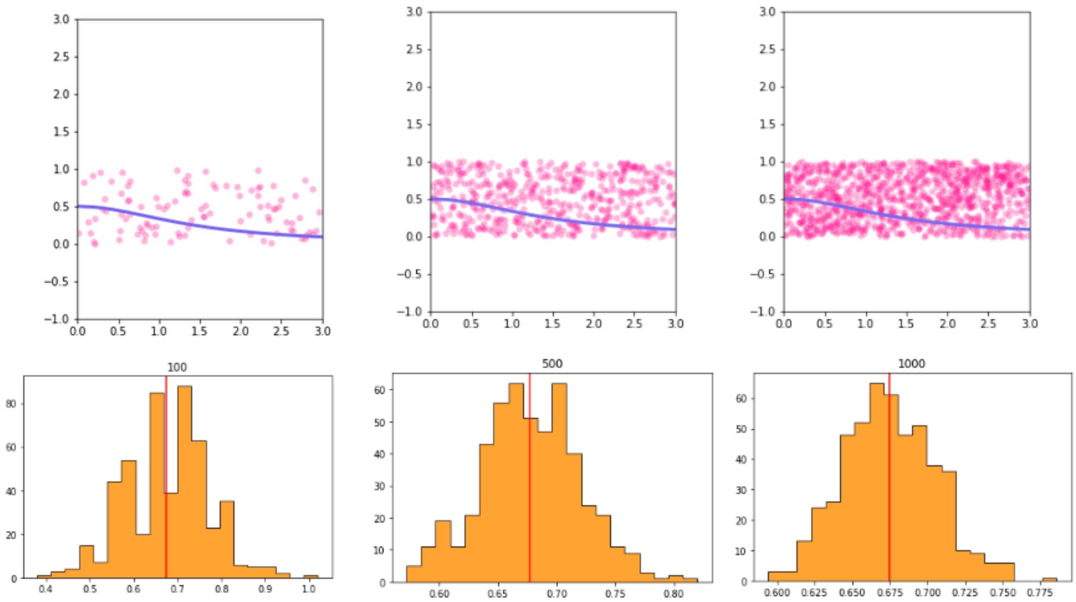
```
#19 Nov 2022
from random import uniform
import matplotlib.pyplot as plt
x_ran,y_ran=[],[]
```

```python
def funci(x):
  return 1/(2+x**2)
def monte_carlo_integrate(f, a, b, c, d, num_points):
    inside_count = 0
    for i in range(num_points):
        x = uniform(a,b)
        y = uniform(c,d)
        x_ran.append(x);y_ran.append(y)
        if 0 <= y <= f(x):
          inside_count += 1
        elif f(x) <= y <= 0:
          inside_count -= 1
    return inside_count/num_points*(b-a)*(d-c)

print(monte_carlo_integrate(funci,0,3,0,1,1000))
x1=np.linspace(0,10)
plt.figure(figsize=(4,5));
plt.xlim([0,3]); plt.ylim([-1,3])
plt.plot(x1,funci(x1),color='mediumslateblue',Linewidth=3.0)
plt.scatter(x_ran,y_ran,linewidth=0.007,c ="deeppink",alpha=0.3)
```



$$\int_0^3 \frac{1}{2+x^2}dx \approx 0.79923265$$

Monte Carlo



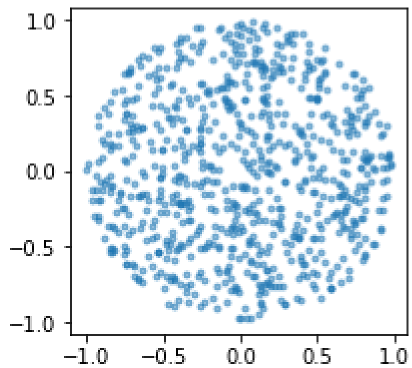
N= 100 I=0.684280 **N= 500 I=0.674392** **N=1000 I=0.674552**

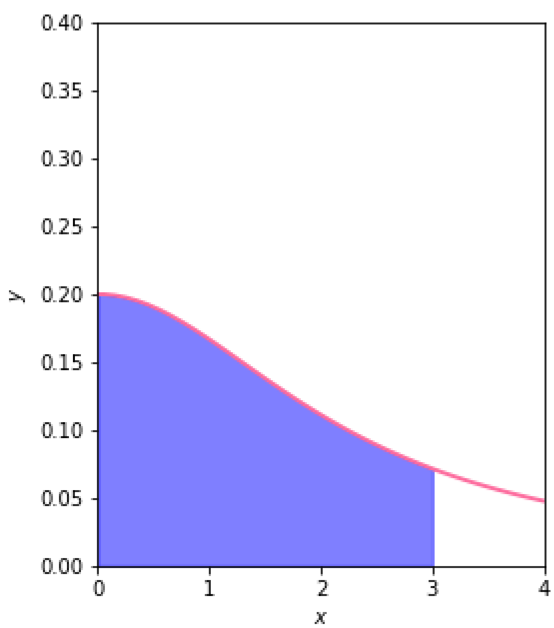
```
#Code to evaluate the MC integration N times and determine the average and plot the histogram
monty=[]
N=500
def funci(x):
  return x**2
for k in range(0,N):
  monty.append(monte_carlo_integrate(funci,0,1,0,1,500))
sum=0
for i in range(0, len(monty)):
   sum = sum + monty[i];
average=sum/len(monty)

plt.hist(monty,histtype='stepfilled',color='darkorange',alpha=0.8,edgecolor='black'
, bins=20,label='Chem')
plt.axvline(average,color='red')
```

Determining the value of Pi

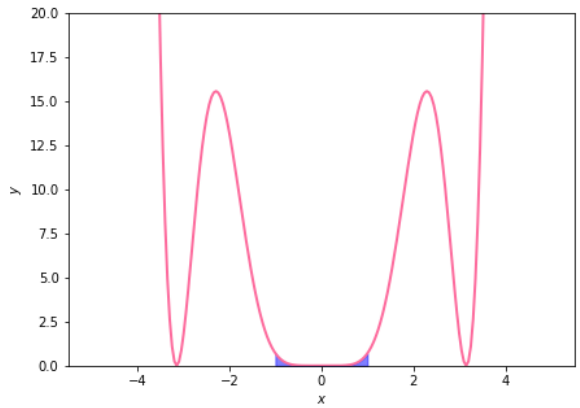
```
#Last update Piyush
import numpy
import matplotlib.pyplot as plt
#plt.style.use("bmh")
#%config InlineBackend.figure_formats=["png"]
```

```
N = 10_00
inside = []
for i in range(N):
    x = numpy.random.uniform(-1, 1)
    y = numpy.random.uniform(-1, 1)
    if numpy.sqrt(x**2 + y**2) < 1:
        inside.append((x, y))
plt.figure(figsize=(3, 3))
plt.scatter([x[0] for x in inside], [x[1] for x in inside], marker=".", alpha=0.5);
pi=4 * len(inside)/float(N)
print(pi)
```

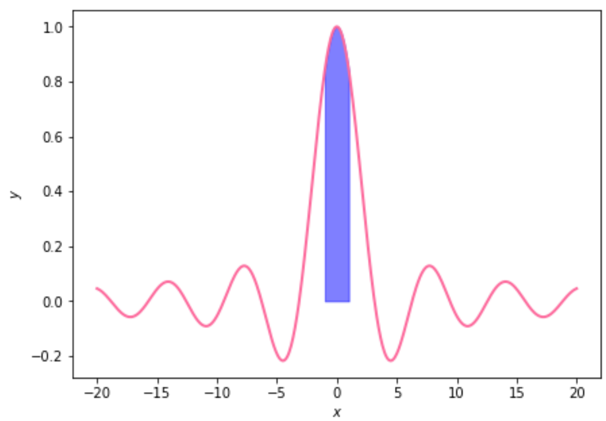




$$\int_0^3 \frac{1}{2+x^2}dx \approx 0.79923265$$

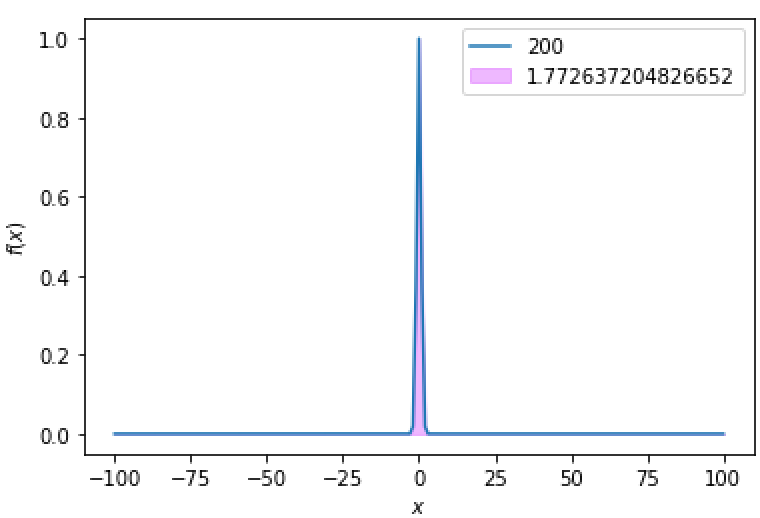


$$\int_{-1}^{1} x^4 sin^2 x dx \approx 0.21925$$

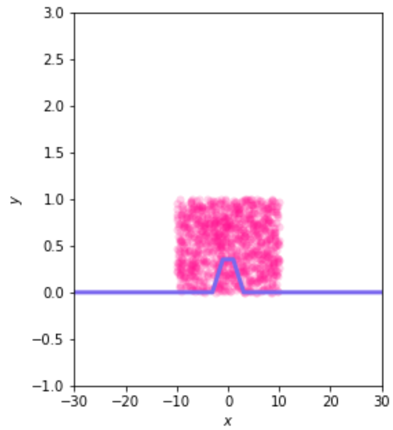


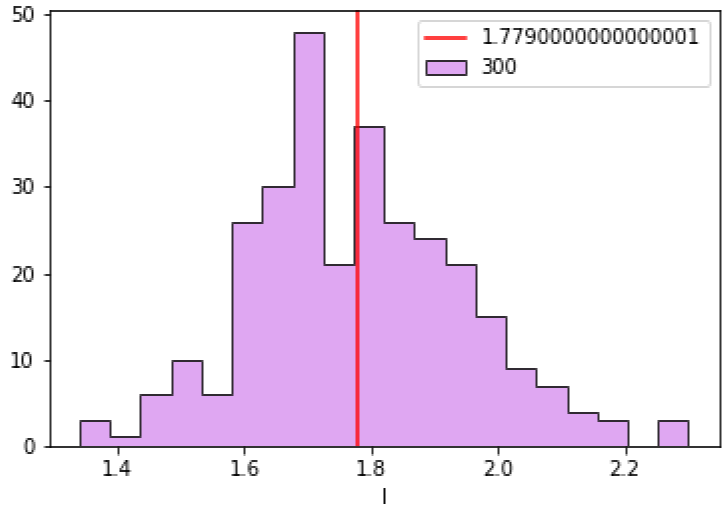
$$\int_{-\infty}^{\infty} e^{-x^2} dx \approx 1.7724538$$

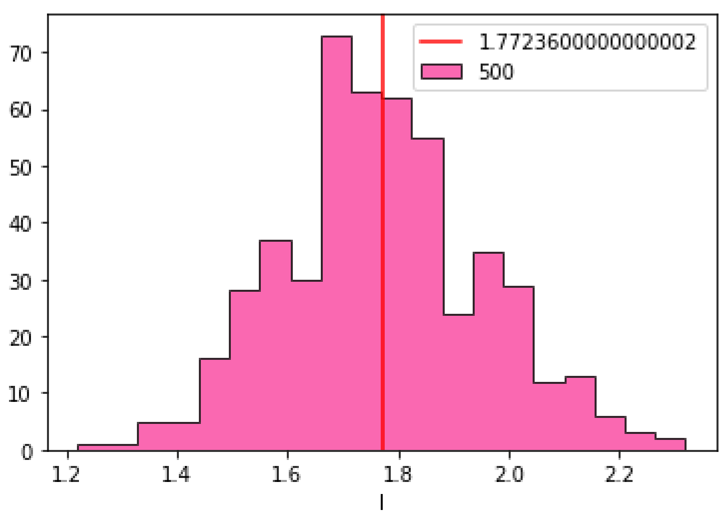
Trapezoid



Monte Carlo

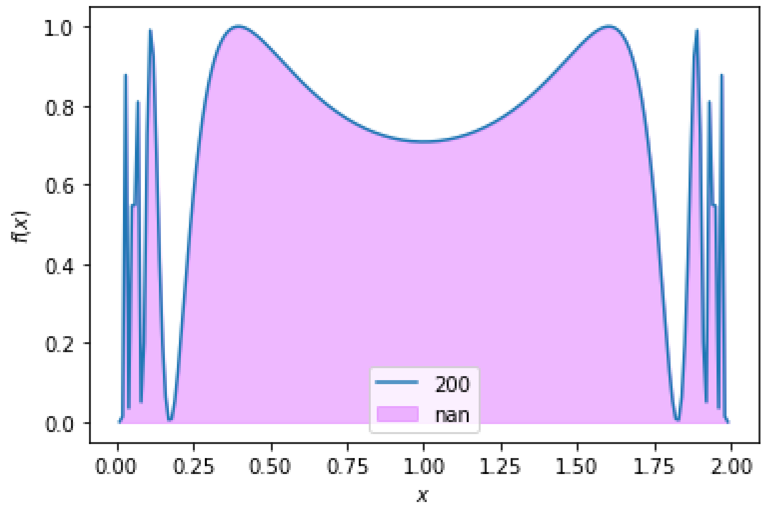






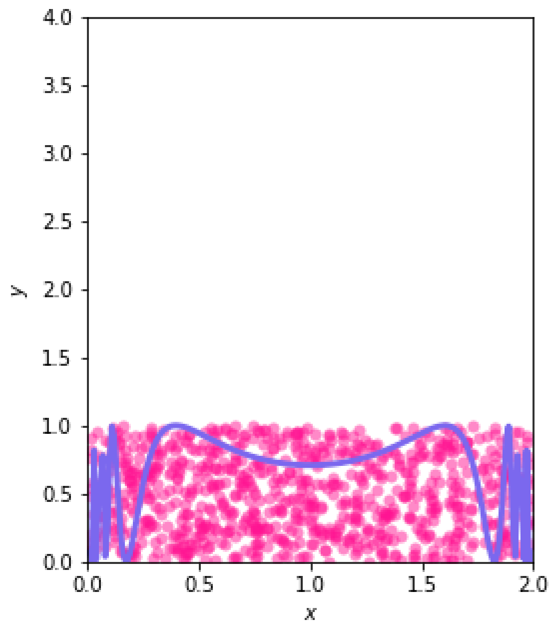
$$I = \int_0^2 \sin^2\left(\frac{1}{x(2-x)}\right) dx$$

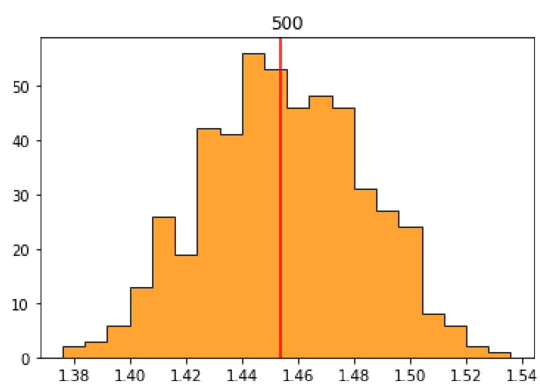
Trapezoid



Monte Carlo

1.4539480000000011





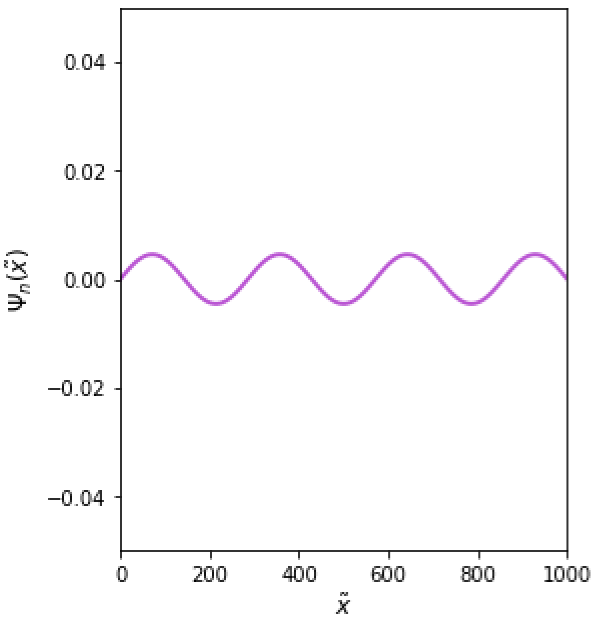
Differential Equations

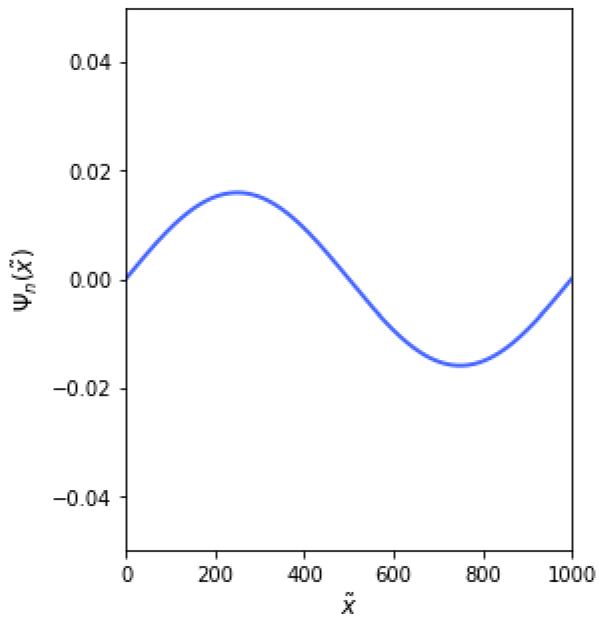
Numerov Algorithm

$$\frac{-\hbar^2}{2m}\frac{\partial^2}{\partial x^2}\psi(x)+V(x)\psi(x)=E\psi(x)$$

$$\Psi_{n+1} = \frac{2(1-\frac{5}{12}l^2k_{n+1}^2)\Psi_n - (1+\frac{1}{12}l^2k_{n-1}^2)\Psi_{n-1}}{1+\frac{1}{12}l^2k_{n+1}^2}$$

```
#Last update piyush 4 Oct 2022 Particle in a box
import numpy as np
import matplotlib.pyplot as plt
N   = 1000
psi = np.zeros(N)              # wavefunction
x   = np.linspace(0,1000,N)    # grid points
v   = (-1)*np.ones(N)
g2  = 200            #gamma square
ep  = 1.418053       # intiial energy
k2  = g2*(ep-v)
l2 = (1.0/(N-1))**2
def wavefunction(ep,N):   #Numerov Algorithm
  psi[0] = 0
  psi[1] = 1e-4
  for i in range(2,N):
     psi[i] =
(2*(1-(5.0/12)*l2*k2[i-1])*psi[i-1]-(1+(1.0/12)*l2*k2[i-2])*psi[i-2])/(1+(1.0/12)*l
2*k2[i])
  return psi
plt.figure(figsize=(4,5));
plt.xlim([0,N]); plt.ylim([-0.05,0.05])
#plt.plot(x,wavefunction(-0.95065,N),linewidth=2.0,color='magenta')
#plt.plot(x,wavefunction(-0.80260,N),linewidth=2.0,color='cornflowerblue')
plt.plot(x,wavefunction(ep,N),linewidth=2.0,color='mediumorchid')
plt.ylabel(r'$\Psi_{n}(\tilde{x})$',fontsize=12)
plt.xlabel(r'$\tilde{x}$',fontsize=12)
```



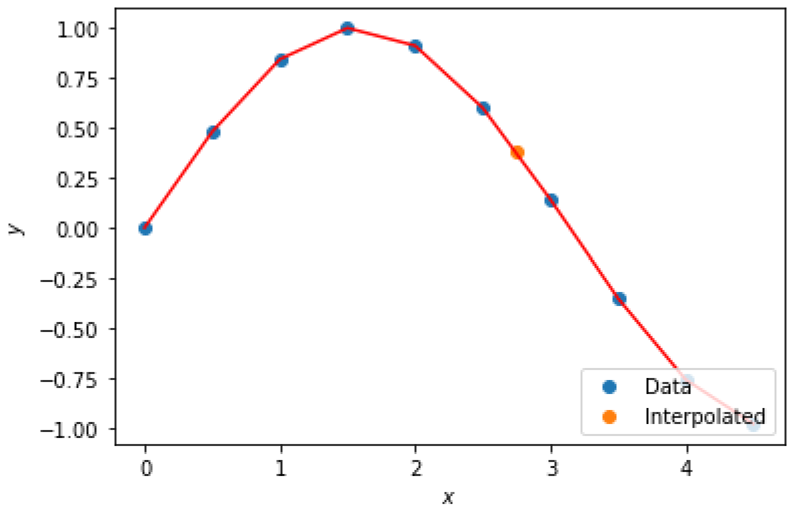


Interpolation

Lagrange

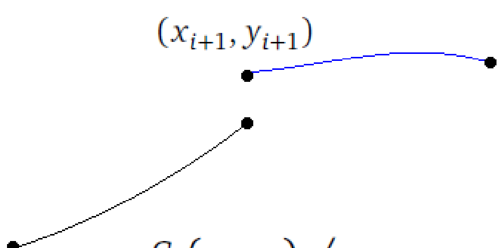
```
#Lagrange Interpolation
#Last update:Piyush
import numpy as np
import matplotlib.pyplot as plt
#x=[1,1.5,2,3.2,4.5]    #If you have a dataset
#y=[5,8.2,9.2,11,16]
N = 10
x = np.zeros((N))       #If you want a functional form
y = np.zeros((N))
for i in range(len(x)):
  a=i/2
  x[i]=a
  y[i]=np.sin(a)
#xp = float(input('Enter interpolation point: '))
xp = 2.75; yp = 0
for i in range(len(x)):
    prod = 1
    for j in range(len(x)):
      if i != j:
            prod = prod * (xp - x[j])/(x[i] - x[j])
    yp = yp + prod * y[i]
print('Interpolated value at %.3f is %.3f.' % (xp, yp))
```

```python
plt.scatter(x,y,label='Data')
plt.plot(x,y,linewidth=1.5,color='red')
plt.xlabel(r'$x$');plt.ylabel(r'$y$')
plt.scatter(xp,yp,label='Interpolated')
plt.legend(loc='lower right')
```

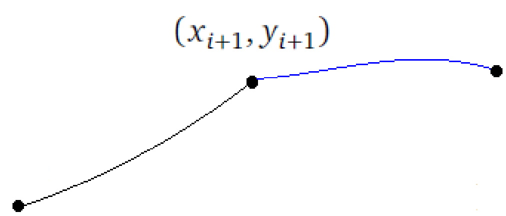


Spline

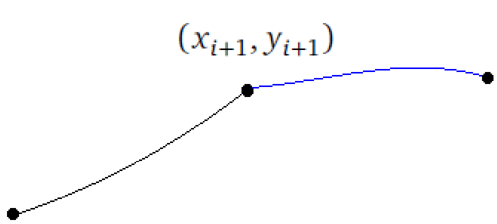
$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3.$$



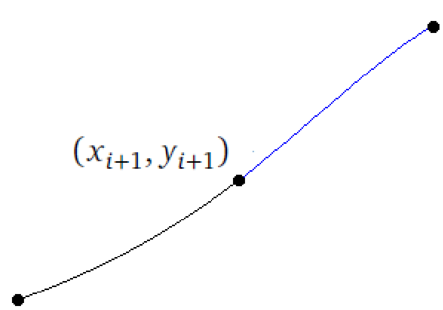
$$S_i(x_{i+1}) \neq y_{i+1}$$



$$S_i(x_{i+1}) = y_{i+1}$$



$$S_i'(x_{i+1}) \neq S_{i+1}'(x_{i+1})$$



$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$$

$$
\left[\begin{array}{cccccccccc|c}
1 & 0 & 0 & 0 & 0 \cdots & 0 & 0 & 0 & 0 & 0 \\
h_1 & 2(h_1+h_2) & h_2 & 0 & 0 \cdots & 0 & 0 & 0 & 0 & 3\left(\frac{H_2}{h_2}-\frac{H_1}{h_1}\right) \\
0 & h_2 & 2(h_2+h_3) & h_3 & 0 \cdots & 0 & 0 & 0 & 0 & 3\left(\frac{H_3}{h_3}-\frac{H_2}{h_2}\right) \\
\vdots & \vdots & \vdots & \vdots & \vdots \ddots & \vdots & \vdots & \vdots & \vdots & \\
0 & 0 & 0 & 0 & 0 \cdots & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & 3\left(\frac{H_{n-1}}{h_{n-1}}-\frac{H_{n-2}}{h_{n-2}}\right) \\
0 & 0 & 0 & 0 & 0 \cdots & 0 & 0 & 0 & 1 & 0
\end{array}\right]
$$

Tridiagonal Matrices: Thomas Algorithm

W. T. Lee*
MS6021, Scientific Computation, University of Limerick

The Thomas algorithm is an efficient way of solving tridiagonal matrix systems. It is based on LU decomposition in which the matrix system $Mx = r$ is rewritten as $LUx = r$ where L is a lower triangular matrix and U is an upper triangular matrix. The system can be efficiently solved by setting $Ux = \rho$ and then solving first $L\rho = r$ for ρ and then $Ux = \rho$ for x. The Thomas algorithm consists of two steps. In Step 1 decomposing the matrix into $M = LU$ *and* solving $L\rho = r$ are accomplished in a single downwards sweep, taking us straight from $Mx = r$ to $Ux = \rho$. In step 2 the equation $Ux = \rho$ is solved for x in an upwards sweep.

$$
b_i = \frac{H_i}{h_i} - \frac{h_i}{3}(2c_i + c_{i+1}), \qquad d_i = \frac{c_{i+1} - c_i}{3h_i}.
$$

```
#Spline Interpolation Piyush Oct 12,2022
import numpy as np
import matplotlib.pyplot as plt

#input data
x=[1,3,4,5,9] ; y=[2,5,8,7,15]
N=len(x)-1

h =[0]*N   ; H =[0]*N   #x diff and y diff
b =[0]*N   ; d =[0]*N

for i in range(0,N):
  h[i]=x[i+1]-x[i]
  H[i]=y[i+1]-y[i]
A = np.zeros((5,5))
print("h");print(h)
print("H");print(H)

A[0][0]=1; A[N,N]=1 #Boundary conditions
```

```
#Constructing A
for w in range(1,N):
  A[w][w+1]   = h[w]
for k in range(0,N-1):
  A[k+1][k]   = h[k]
for l in range(1,N):
  A[l][l] = 2*(A[l][l-1]+A[l][l+1])
print("Matrix A");print(A)

B=np.zeros((5,1))
for s in range(1,len(B)-1):
  B[s][0]=3*((H[s]/h[s])-(H[s-1]/h[s-1]))
print("Matrix B");print(B)

#Thomas Algorithm to solve tridiagonal matrix
gamma  = [0]*N
rho    = [0]*(N+1)
gamma[0] = A[0][1]/A[0][0]
rho[0]   = B[0][0]/A[0][0]

for o in range(1,N):
  gamma[o]=A[o][o+1]/(A[o][o]-gamma[o-1]*A[o][o-1])
for z in range(1,N+1):
   rho[z]=(B[z][0]-rho[z-1]*A[z][z-1])/(A[z][z]-gamma[z-1]*A[z][z-1])
print("rho");print(rho)
print("gamma"); print(gamma)

c=[0]*(N+1)
c[-1]=rho[-1]
for t in reversed(range(0,N)):
  c[t]=rho[t]-gamma[t]*c[t+1]
print("c");print(c)

#calculate coefficients
b = [(H[g]/h[g])-(h[g]/3)*(2*c[g]+c[g+1]) for g in range(0,N)]
print("b");print(b)
d = [(c[e+1]-c[e])/(3*h[e]) for e in range(0,N)]
print("d");print(d)
```

I. STAGE 1

In the first stage the matrix equation $Mx = r$ is converted to the form $Ux = \rho$. Initially the matrix equation looks like:

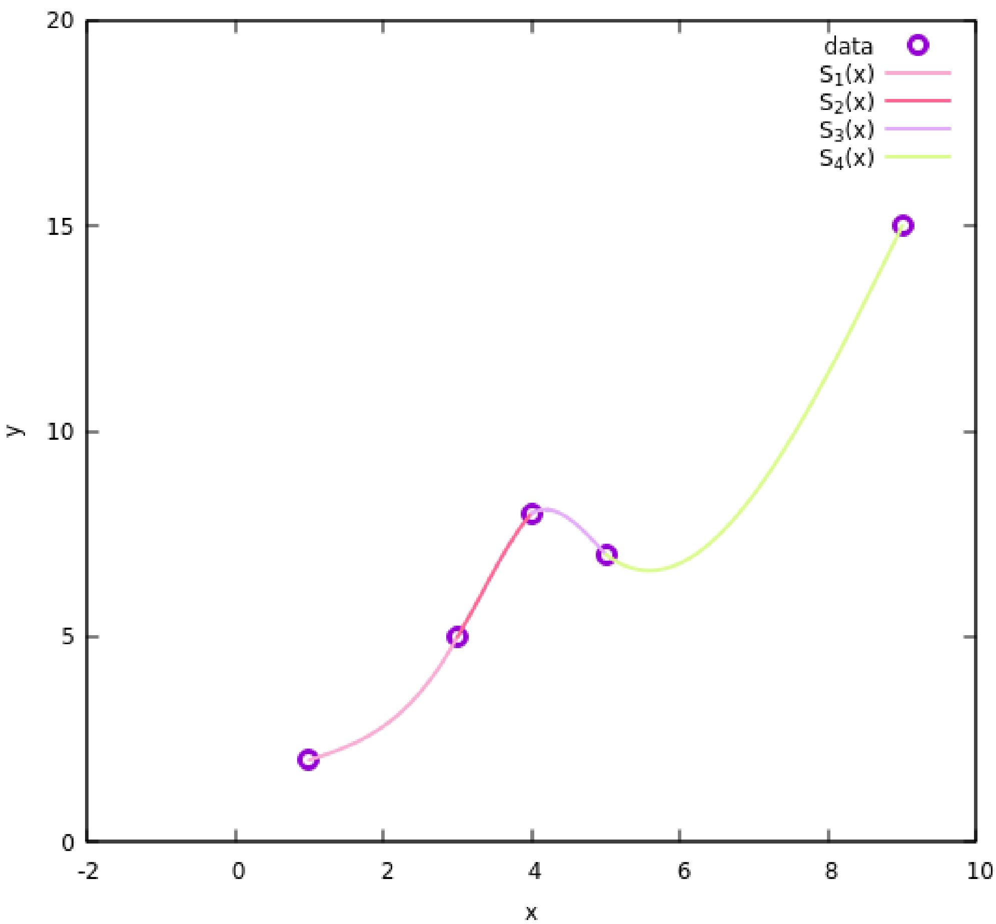
$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 \\ 0 & 0 & a_4 & b_4 & c_4 & 0 \\ 0 & 0 & 0 & a_5 & b_5 & c_5 \\ 0 & 0 & 0 & 0 & a_6 & b_6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{pmatrix}$$

Row 1: $x_1 + \gamma_1 x_2 = \rho_1$.
Rearrange to get: $x_1 = \rho_1 - \gamma_1 x_2$.

$$\begin{pmatrix} 1 & \gamma_1 & 0 & 0 & 0 & 0 \\ 0 & 1 & \gamma_2 & 0 & 0 & 0 \\ 0 & 0 & 1 & \gamma_3 & 0 & 0 \\ 0 & 0 & 0 & 1 & \gamma_4 & 0 \\ 0 & 0 & 0 & 0 & 1 & \gamma_5 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ \rho_4 \\ \rho_5 \\ \rho_6 \end{pmatrix}$$

At this point x, the solution to the matrix equation, is fully determined.

Thomas algorithm was used to solve the tridiagonal matrix A to determine the cubic spline coefficients.

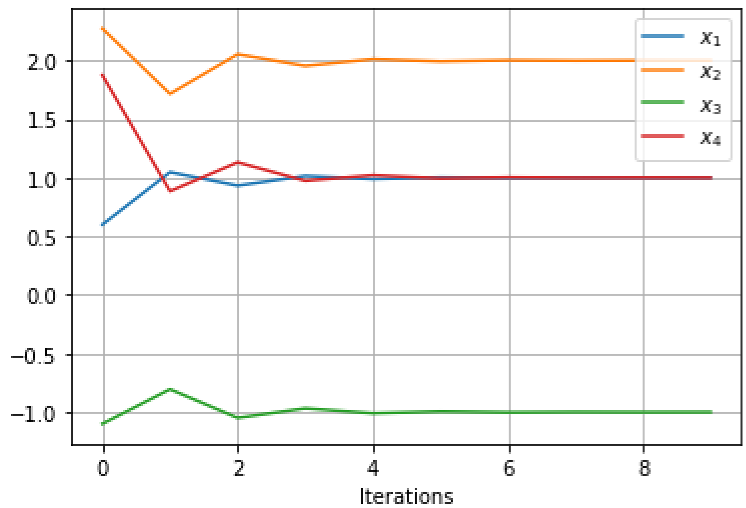


Matrix Solvers

Jacobi

```
#Jacobi's Method
import matplotlib.pyplot as plt
x1=0;x2=0;x3=0;x4=0
data_x1=[];data_x2=[];data_x3=[];data_x4=[]
for i in range (10):
    a=(6+x2-2*x3)/10
```

```
    b=(25+x1+x3-3*x4)/11
    c=(-11-2*x1+x2+x4)/10
    d=(15-3*x2+x3)/8
    x1=a;x2=b;x3=c;x4=d
    data_x1.append(x1);data_x2.append(x2);data_x3.append(x3);data_x4.append(x4)
print(x1,x2,x3,x4)
plt.plot (data_x1, label=r'$x_{1}$')
plt.plot (data_x2, label=r'$x_{2}$')
plt.plot (data_x3, label=r'$x_{3}$')
plt.plot (data_x4, label=r'$x_{4}$')
plt.xlabel('Iterations');plt.legend();plt.grid();plt.show()
```



$$\begin{bmatrix} 27 & 6 & -1 \\ 6 & 15 & 2 \\ 1 & 1 & 54 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 54 \\ 72 \\ 110 \end{bmatrix}$$

Gauss Seidel

```
#Gauss Seidel Iteration
#Last update:Piyush
N=5 #number of iterations
# 27x+6y-z=54
# 6x+15y+2z=72
# x+y+54z=110
x=0;y=0;z=0
for i in range(N):
  x = (1/27)*(54-6*y+z)
  y = (1/15)*(72-6*x-2*z)
  z = (1/54)*(110-x-y)
print('x',x)
print('y',y)
print('z',z)
```

```
x 1.1663468914798172
```

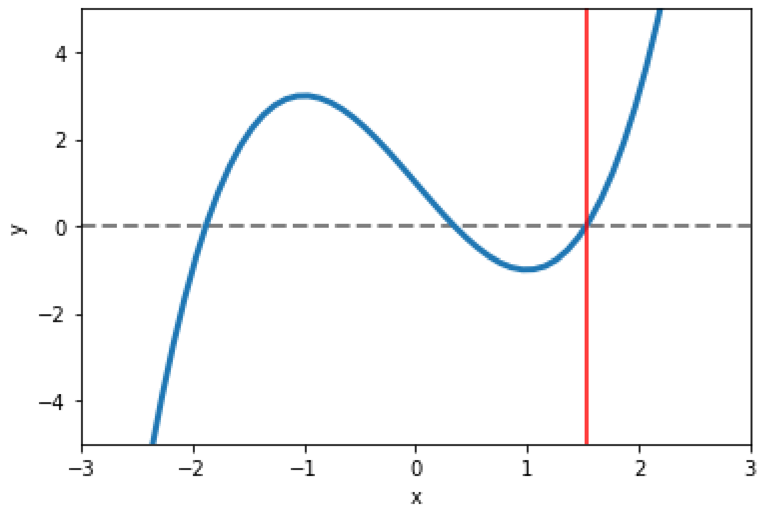
```
y 4.074797518245446
```

```
z 1.9399788072273099
```

Root Finding

Bisection

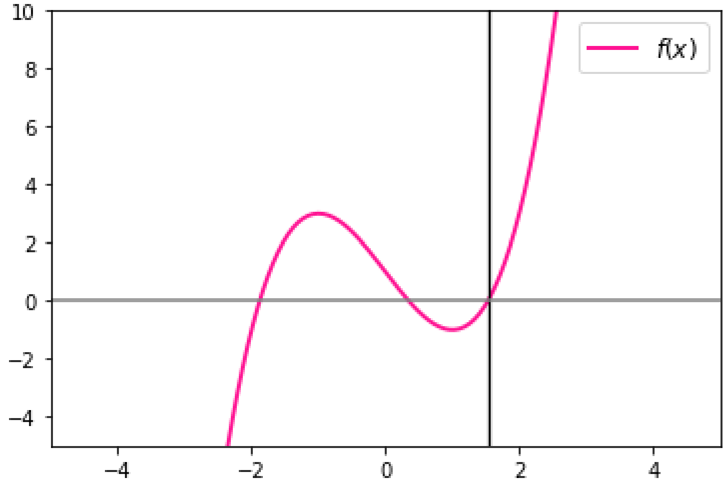
```
#Last update:Piyush
import numpy as np
import matplotlib.pyplot as plt

def bisectionhaitaakat(f, a, b, sehlo):
    if (np.sign(f(a)))*(np.sign(f(b))) > 0:
          print("can't do it")
    mid = (a + b)/2 #midpoint #sehlo is tolerance
    if np.abs(f(mid)) < sehlo:     #jab tak root mil jaye
        return mid   #root mil gaya that's it
    elif np.sign(f(a)) == np.sign(f(mid)):
        print(mid)
        return bisectionhaitaakat(f, mid, b,sehlo)
    elif np.sign(f(b)) == np.sign(f(mid)):
        print(mid)
        return bisectionhaitaakat(f, a, mid,sehlo)

x=np.linspace(-5,5,100)
def f(x):
   return x**3-3*x+1            #yeh raha function
plt.xlabel('x');plt.ylabel('y')
plt.xlim([-3,3.0]);plt.ylim([-5,5])
plt.axhline(0, color='grey',linestyle='--',linewidth=2)

root= bisectionhaitaakat(f,1,2, 1e-5)   #function call kiya
print("root =", root)
print("f(root) =", f(root)) #ideally f(root) has to be zero
plt.plot(x,f(x),'-',linewidth=3.0)
plt.axvline(root,color='red',linewidth=1.5)
```



$$n = \left\lceil \log_2 \left(\frac{b_0 - a_0}{\epsilon} \right) - 1 \right\rceil.$$

Newton-Raphson

```
#Last update:Piyush
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-5,10,200)
def f(x):
  return x**3-3*x+1
def f_prime(x):
  return 3*x**2-3
def newtoroot(f, df, x0, tol):
    if abs(f(x0)) < tol:
        return x0
    else:
        print(x0)
        return newtoroot(f, df, x0 - f(x0)/df(x0), tol)

result = newtoroot(f,f_prime, 1.2, 1e-5)
print("root =", result)
plt.figure(figsize=(6,4));plt.xlim([-5,5]);plt.ylim([-5,10])
plt.plot(x,f(x),label=r'$f(x)$',linewidth=2.0,color='deeppink')
plt.axvline(x=result,color='black',linewidth=1.3)
plt.axhline(y=0,color='grey')
plt.legend(fontsize='12')
```



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

```
1.2
```

```
1.8606060606060604
```

```
1.6088541478282465
```

```
1.5379627005114658
```

```
1.5321277008797534
```

```
root = 1.5320888879510492
```

```
start = []
num   = 10
N     = 5
for i in range(num):
  c=random.randrange(-N, N)
  if(f_prime(c)>0):
    start.append(c)
print(start)

result=[]
for i in range(len(start)):
  result = newtoroot(f,f_prime,start[i], 1e-5)
  print("root =", result)
```

```
[-3, -2, 4, 3, 3, 3]
```

```
root = -1.8793852418279906
```

```
root = -1.879385244836671
```

```
root = 1.5320897486564136
```

```
root = 1.5320888862613389
```

```
root = 1.5320888862613389
```

```
root = 1.5320888862613389
```

```
PROGRAM NEWTON
!This program uses the Newton method to find the root
IMPLICIT NONE
!REAL(KIND = 8) :: DL,DX,X0,X
REAL          :: A,B,tol,DX,X0,X1,DF,F
INTEGER       :: I

      tol = 1.0E-06
      A   = 1.0
      B   = 2.0
      DX = B-A
      X0 = (A+B)/2.0 !midpoint

      I = 0
  DO 100 WHILE (ABS(DX).GT.tol)
      X1 = X0 - F(X0)/DF(X0)
      DX = X1 - X0
      X0 = X1
      I = I + 1
  100 END DO
      WRITE (6,9) I,X0,DX
      STOP
  9 FORMAT (I4,F16.8)
END PROGRAM NEWTON

FUNCTION F(X)
 F = EXP(X)*ALOG(X) - X*X
RETURN
END

FUNCTION DF(X)
  DF = EXP(X)*(ALOG(X) + 1./X) - 2.*X
RETURN
END
```

Chebyshev Method

```
#Chebyshev Method
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-5,10,200)
```

```python
def f(x):            #function
 return x**3-3*x+1
def f_prime(x):      #first derivative
 return 3*x**2-3
def f_2prime(x):     #second derivative
 return 6*x
def chebyroot(f, df, x0, tol):  #chebyshev
   if abs(f(x0)) < tol:
       return x0
   else:
       print(x0)
       return chebyroot(f, df, x0 -
f(x0)/df(x0)-0.5*f_2prime(x0)*((f(x0)**2)/(f_prime(x0))**3), tol)

result = chebyroot(f,f_prime, -1.3, 1e-5)
print("root =", result)
plt.figure(figsize=(6,4));plt.xlim([-5,5]);plt.ylim([-5,10])
plt.plot(x,f(x),label=r'$f(x)$',linewidth=2.0,color='#0ABAB5')
plt.axvline(x=result,color='black',linewidth=1.3);plt.axhline(y=0,color='grey')
plt.legend(fontsize='12')
```

```fortran
PROGRAM CHEBYSHEV
IMPLICIT NONE
REAL          :: A,B,tol,DX,X0,X1,DF,F,D2F
INTEGER       :: I

      tol = 1.0E-05
      A  = -3.0 ; B  = 0.0
      DX = B-A
      X0 = (A+B)/2.0 !midpoint
      I = 0
  DO 21 WHILE (ABS(DX).GT.tol)
      X1 = X0 - F(X0)/DF(X0)-0.5*D2F(X0)*((F(X0))**2/DF(X0)**3)
      DX = X1 - X0
      X0 = X1
      I = I + 1  !step
  21 END DO
      WRITE (6,9) I,X0,DX
      STOP
  9 FORMAT (I4,F16.8)
END PROGRAM CHEBYSHEV

FUNCTION F(X)
  F = X**3-3*X+1
RETURN
END
```

```
FUNCTION DF(X)
  DF = 3*X**2-3
RETURN
END

FUNCTION D2F(X)
  D2F = 6*X
RETURN
END
```

Halley's Method

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2[f'(x_n)]^2 - f(x_n)f''(x_n)}$$

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-5,10,200)

def f(x):           #function
 return x**3-3*x+1
def f_prime(x):     #first derivative
 return 3*x**2-3
def f_2prime(x):    #second derivative
 return 6*x
def chebyroot(f, df, x0, tol):  #halley
   if abs(f(x0)) < tol:
       return x0
   else:
       print(x0)
       return halleyroot(f, df, x0 -
(2*f(x0)*f_prime(x0))/(2*(f_prime(x0)**2)-f(x0)*f_2prime(x0)), tol)

result = halleyroot(f,f_prime, -1.3, 1e-5)
print("root =", result)
plt.figure(figsize=(6,4));plt.xlim([-5,5]);plt.ylim([-5,10])
plt.plot(x,f(x),label=r'$f(x)$',linewidth=2.0,color='#0ABAB5')
plt.axvline(x=result,color='black',linewidth=1.3);plt.axhline(y=0,color='grey')
plt.legend(fontsize='12')
```

Babylonian Method to find square root

$$x - \frac{f(x)}{f'(x)} = x - \frac{x^2-2}{2x} = \frac{x^2+2}{2x} = \frac{x+\frac{2}{x}}{2}.$$

Nth root finding

```
def f(x):           #function
 return x**k-a
def f_prime(x):   #first derivative
 return k*x**(k-1)
def newtoroot(f, df, x0, tol):  #newton raphson method
   if abs(f(x0)) < tol:
       return x0
   else:
       print(x0)
       return newtoroot(f, df, x0 - f(x0)/df(x0), tol)
```

Secant

$$x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)},$$

$$x_3 = x_2 - f(x_2)\frac{x_2 - x_1}{f(x_2) - f(x_1)},$$

$$\vdots$$

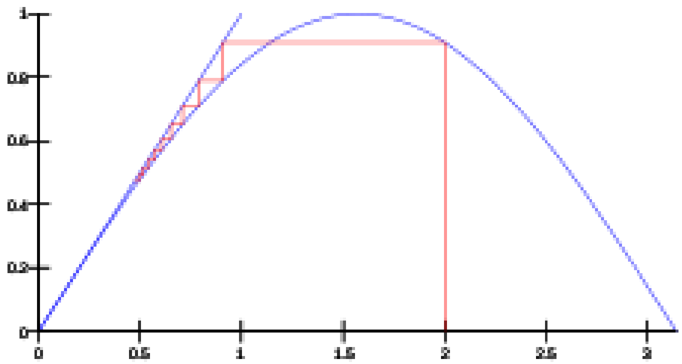
$$x_n = x_{n-1} - f(x_{n-1})\frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}.$$

```
PROGRAM SECANTROOT
      TOL = 1.0E-06
      A   = 10.0
      B   = 30.0
      DX = (B-A)/10.0
      X0 = (A+B)/2.0
      CALL SECANT (DL,X0,DX,ISTEP)
      END PROGRAM SECANTROOT

      SUBROUTINE SECANT (DL,X0,DX,ISTEP)
      !Subroutine for the root of f(x)=0 with the secant method.
      I = 0
```

```
      X1 = X0 + DX
      DO 21  WHILE (ABS(DX).GT.TOL) !stop the loop when
      X2 = X1 - F(X1)*(X1-X0)/(F(X1) - F(X0))
      X0 = X1
      X1 = X2
      DX = X1 - X0
      I = I + 1
      WRITE (6,9) I,X0,DX
      9 FORMAT (I4,F16.8)
 21 END DO
      RETURN
      END

      FUNCTION F(X)
      F = X**2-612
      RETURN
      END
```

Fixed Point

```
program FixiePixie
IMPLICIT NONE
REAL::f
INTEGER::N,i
real, allocatable :: x(:)
N=10
allocate(x(N))
x(1) = 0.0

do i=1,N
      x(i+1) = f(x(i))
end do

do i = 1,N
      write(*,*) x(i)
  end do

end program FixiePixie

REAL function f(x1)
REAL::x1
  f=0.015625*(32*x1**5+31)
return
end function
```



$$x_{n+1} = sin(x_n); x_0 = 2$$

```
pos=[]
x = 2
for i in range(20):
   x = np.sin(x)
   pos.append(x)
   print(x)
```

$$(AX)_n = x_{n+2} - \frac{(\Delta x_{n+1})^2}{\Delta^2 x_n} = x_{n+2} - \frac{(x_{n+2} - x_{n+1})^2}{(x_{n+2} - x_{n+1}) - (x_{n+1} - x_n)}$$

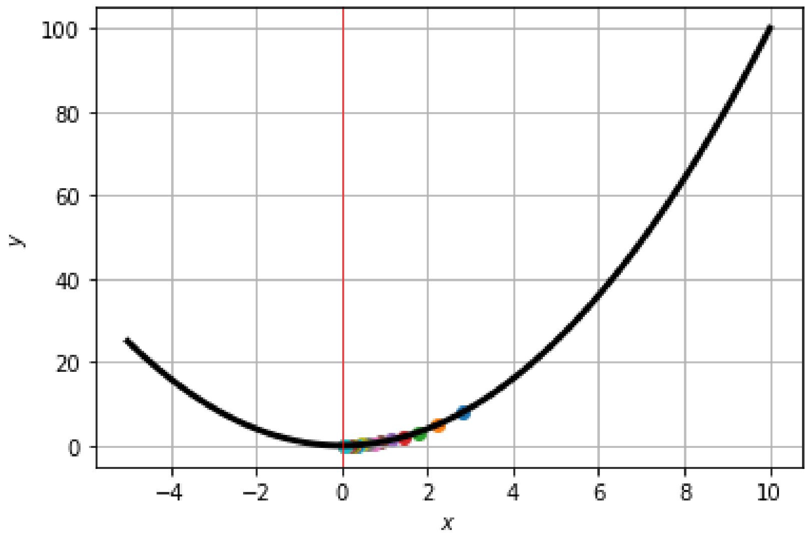
```
import numpy as np
import matplotlib.pyplot as plt
pos=[]
x = 2.0
#This loop gives initial successive approximations to the root zeta
for i in range(3):
   #x = (10/(4+x))**(0.5)
   x = 1/(1+x**2)
   pos.append(x)
   print(x)

#Aitken's Delta 2 process
for i in range(1,15):
 zeta=pos[i+1]-((pos[i+1]-pos[i])**2)/(pos[i+1]-2*pos[i]+pos[i-1])
 pos.append(zeta)
 print(zeta)
```

Optimization

Gradient Descent

1D

```
#1D Gradient Descent
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize
from scipy.misc import derivative
x = np.linspace(-5,10,100)
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.grid();
def f(x):                     #Function
    return -x**2*np.sin(2*x)
dx = derivative(f,x)          #Derivative
alpha = 0.1    #Learning rate
x_new = 2.5    #yaha se start karo
N     = 20     #itni baar karo
plt.plot(x,f(x),linewidth=3,color='black')
for i in range(N):
    x_old = x_new
    x_new += -alpha*derivative(f,x_old)
    print(x_new,f(x_new)) #plot ke liye
    plt.scatter(x_new,f(x_new))
print("local minimum: %.3f" % x_new)
#scipy dhoondh minimum ab
result = optimize.minimize_scalar(f)
x_min = result.x
plt.axvline(x=x_min,color='red',linewidth=0.6)
print("scipy minimum",x_min)
```



Algorithm 1: Newton's method (Optimization)

initialize $x^{(0)}$

for k **in** *1, to max-iter,* **do**

$p^{(k)} = -H_f(x^{(k)}) \backslash \nabla f(x^{(k)})^T$ `// since we interpret the gradient as a row vector, this is a column`

if $||p^{(k)}|| < \epsilon_{\text{tol}}$ **then**

break

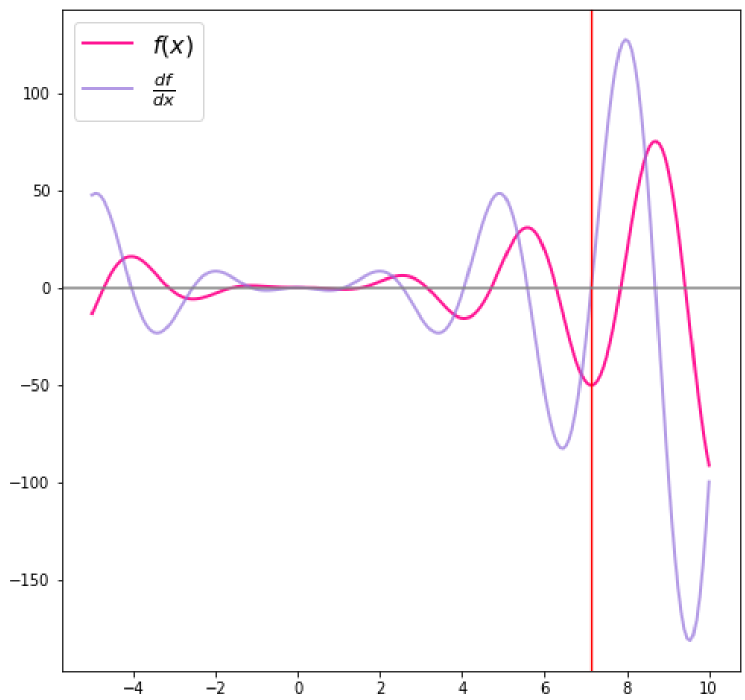
end

$x^{(k+1)} \leftarrow x^{(k)} + p(k)$

end

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-5,10,200)
def f(x):
  return -x**2*np.sin(2*x)
def f_prime(x):
  return -2*x*(np.sin(2*x)+x*np.cos(2*x))
def f_prime2(x):
  return 2*(2*x**2-1)*np.sin(2*x)-8*x*np.cos(2*x)

def newtoroot(f, df, x0, tol):
    if abs(f(x0)) < tol:
        return x0
    else:
        return newtoroot(f, df, x0 - f(x0)/df(x0), tol)
result = newtoroot(f_prime, f_prime2, 6.9, 1e-5)
print("x_min|x_max =", result)
plt.figure(figsize=(8,8))
```

```
plt.plot(x,f(x),label=r'$f(x)$',linewidth=2.0,color='deeppink')
plt.plot(x,f_prime(x),label=r'$\frac{df}{dx}$',linewidth=2.0,color='mediumpurple',a
lpha=0.7)
#plt.plot(x,f_prime2(x),label=r'$\frac{d^2f}{dx^2}$',linewidth=2.0,color='turquoise
',alpha=0.5)
plt.axvline(x=result,color='red',linewidth=1.3)
plt.axhline(y=0,color='grey')
plt.legend(fontsize='16')
```

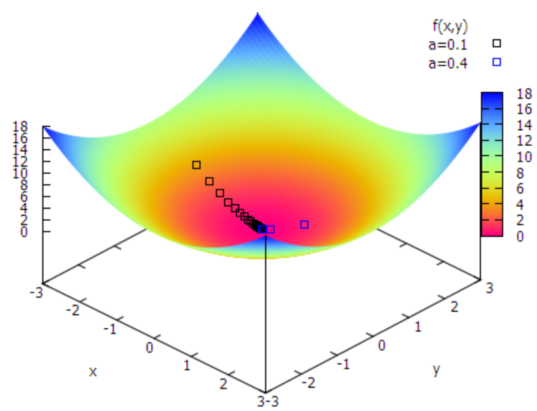


2D gradient descent

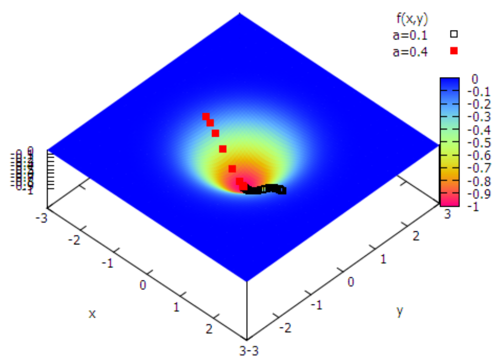
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-5,5,100)
y=np.linspace(-5,5,100)
def f(x,y):
    return x**2.0+y**2.0
def dx(x,y):
     return 2.0*x
def dy(x,y):
     return 2.0 *y
# Gradient Descent
alpha = 0.1 # learning rate
x1_0 = 1.5 # start point
x2_0 = 2.8
```

```
plt.scatter(x1_0,x2_0)
N=20
for i in range(N):
    tmp_x1_0 = x1_0 - alpha * dx(x1_0,x2_0)
    tmp_x2_0 = x2_0 - alpha * dy(x1_0,x2_0)
    x1_0 = tmp_x1_0
    x2_0 = tmp_x2_0
    print(x1_0,x2_0,f(x1_0,x2_0))
    plt.scatter(x1_0, x2_0)
#print(x1_0,x2_0,f(x1_0,x2_0))
```

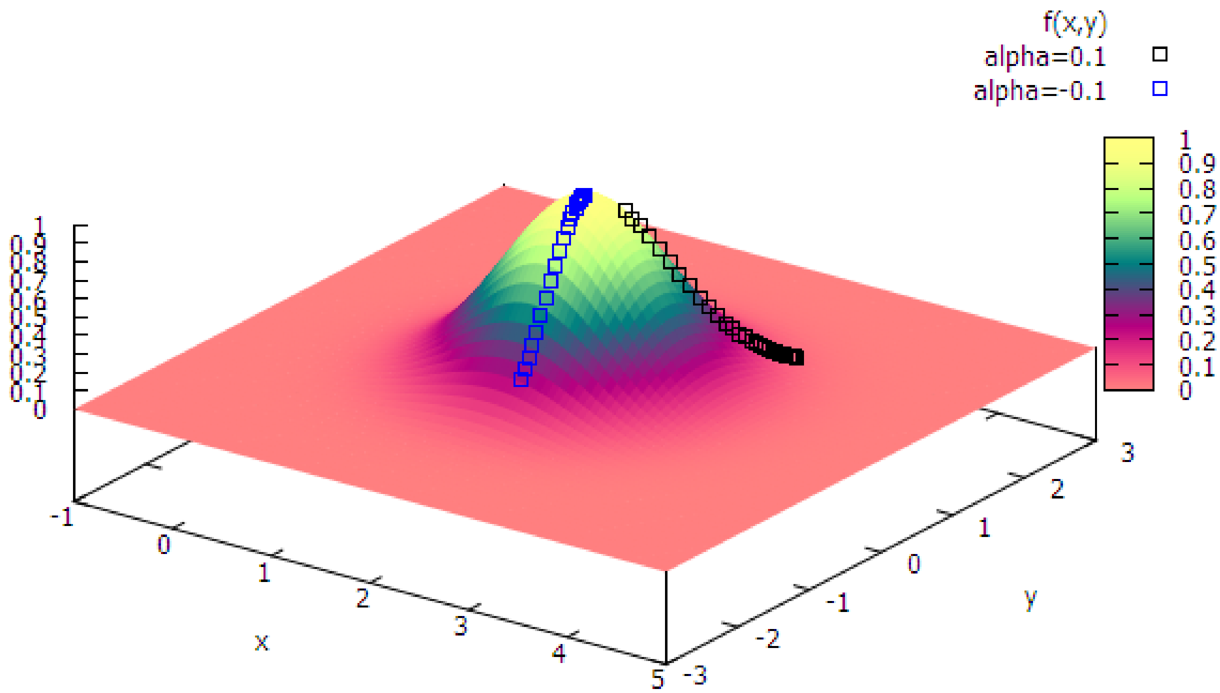
$$f(x,y) = x^2 + y^2$$



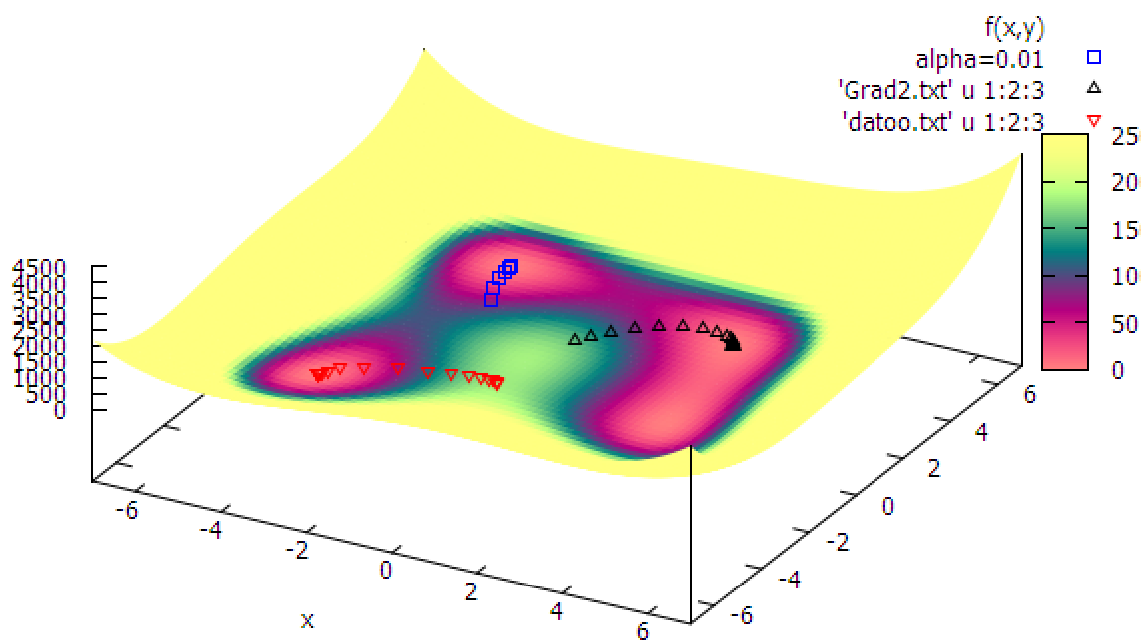
$$f(x,y) = -e^{-x^2-y^2}$$



$$f(x,y) = e^{-(x-2)^2-y^2}$$



$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$



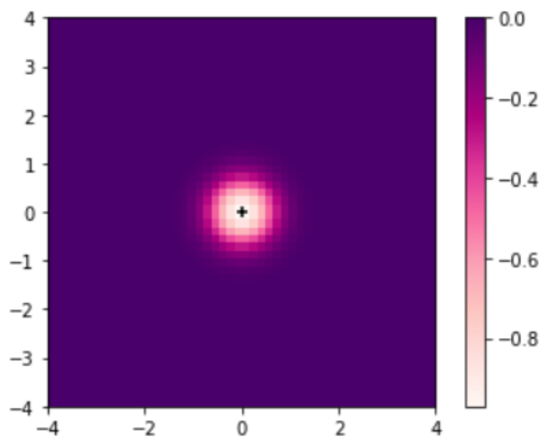
$$\alpha = 0.01$$

$$\frac{\partial f(x,y)}{\partial x} = 2(2x(x^2 + y - 11) + x + y^2 - 7)$$

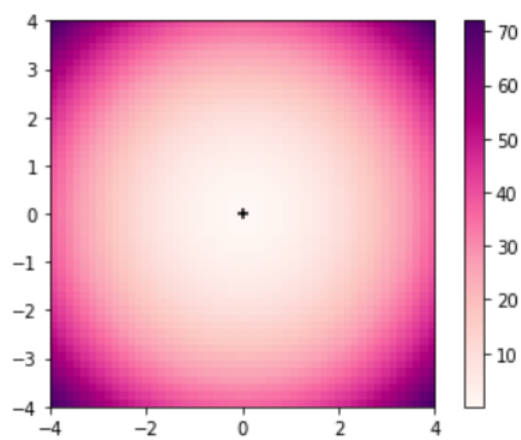
$$\frac{\partial f(x,y)}{\partial y} = 2(x^2 + 2y(x + y^2 - 7) + y - 11)$$

```
from scipy import optimize
x_mini = optimize.minimize(funcy, x0=[-3, 0])
plt.imshow(funcy([x1, y1]), extent=[-4, 4, -4, 4], origin="lower",cmap='RdPu')
plt.colorbar(label=r'$f(x,y)$')
plt.scatter(x_mini.x[0], x_mini.x[1],color='black',marker='+')  #minimum
```

$$f(x,y) = -e^{-x^2-y^2}$$



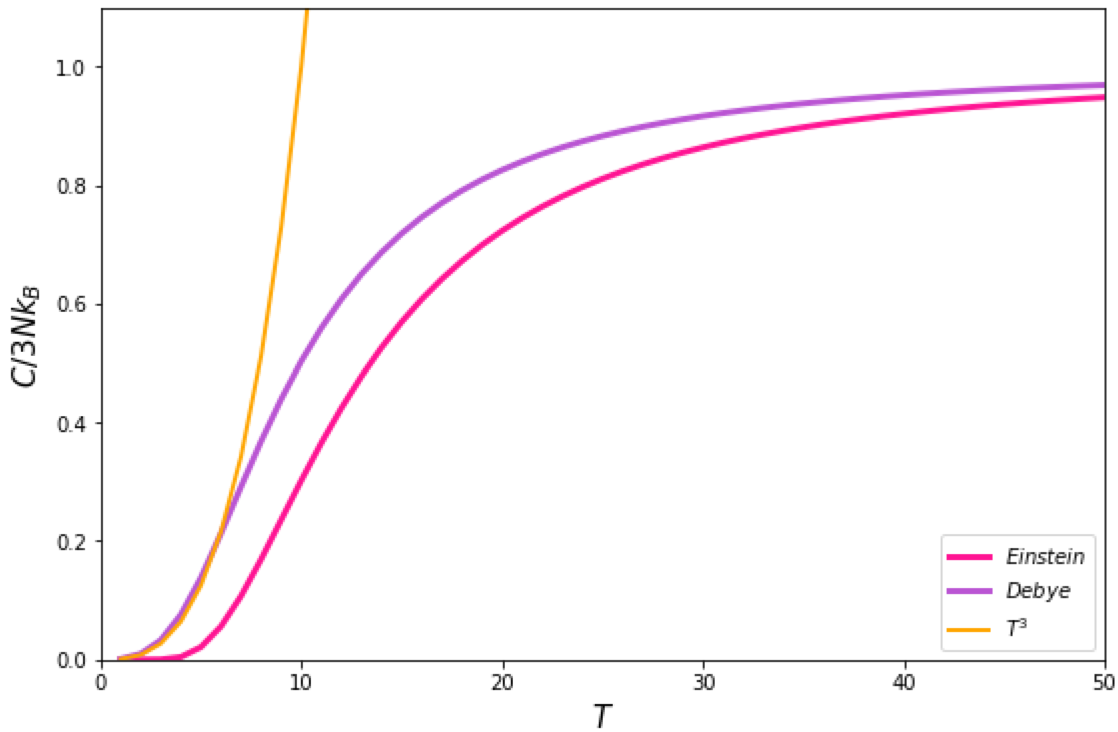
$$f(x,y) = x^2 + y^2$$



Debye Heat

```
# The Heat capacities but after they went to art class
```

```
from scipy.integrate import quad   # For quad integration
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

#Temperatures
T=np.linspace(1,100, 100)
T_e=np.float_(40.10)      #Einstein Temperature
T_d=np.float_(40.00)       #Debye Temperature

def Dulong(T):
    return [1]*len(T)    #Dulong Petit Law C_v=3NK_B;
                         #Normalised it so this is 1.

def Einstein(T,T_e):     #Einstein Function
    return ((T_e/T)**2)*(np.exp(T_e/T)/(np.exp(T_e/T)-1)**2)

#Defining the function for the integral in Debye function
def thatintegralindebye(x):
    return ((x**4)*np.exp(x))/((np.exp(x)-1)**2)
    #Defining the function for the integral part in Debye Function

#Using scipy to integrate the integral and evaluating Debye
def Debye(T,T_d):
    deby=list()
    for t in T:
        deby.append(3*((t/T_d)**3)
*np.float_(quad(thatintegralindebye,0,T_d/t)[0]))
    return (np.array(deby))

plt.figure(figsize=(9, 6))
plt.plot( T, Einstein(T,T_e), linewidth=3,label='$Einstein$',color='deeppink')
#plt.plot( T, Dulong(T), linewidth=3,label='$Dulong-Petit$',color='darkturquoise')
plt.plot( T, Debye(T,T_d),linewidth=3, label='$Debye$',color='mediumorchid')
plt.plot( T,(1e-3)*T**3,label='$T^{3} $',lw=2,color='orange')
#plt.plot( T,100*T**(-2)*np.exp(-(1/T)),label='$T^{2}e^{-1/T} $')
#plt.axvline(x=T_e,label='$T_e$',color='skyblue')
#plt.axvline(x=T_d,label='$T_d$',color='limegreen')
plt.xlim([0,50])
plt.ylim([0,1.1])
plt.xlabel("$T$",fontsize=15)
plt.ylabel("$C/3N k_B$",fontsize=15)
plt.legend(loc='lower right')
plt.show()
```



Polynomial evaluation

Horner's Method

```
program hornerpoly
implicit none
real,allocatable ::coeff(:),y(:)
integer::N,i,num

real :: x=3
!counter gin bhai
N = 0
OPEN (1, file ='data.txt')
DO
       READ (1,*, END=10)
       N = N + 1
END DO
10 CLOSE (1)
print*,N

allocate (coeff(N),y(N))  !allocate the arrays from the text file
!Reading the coefficients from the file

open (unit = 1, file ='data.txt', status ='old')
do i = 1,N
```

```
      read (1,*) coeff(i)
end do
close (1)

num = SIZE(coeff)
y(1)=coeff(n)
do i = 2,num
      y(i)=coeff(num-i+1)+x*y(i-1)
end do
print*,y(num)
end program hornerpoly
```

```
#17 Sep 2022
import numpy as np
data=np.loadtxt('data.txt')
coeff=data[0:]
#coeff=[3,-1,2,-4,0,1]   #coefficient matrix
#x = int(input("Enter value of x:"))
x=3
n = len(coeff)-1
y = np.zeros_like(coeff)
y[0]=coeff[n]
for i in range(1,n+1): # run loop for till a_n # access coeff array backwards
 y[i]=coeff[n-i]+x*y[i-1]
print(y[n])
```

File reader:

5 A 1 2 5

5 B 2 3 4

6 C 0 0 1

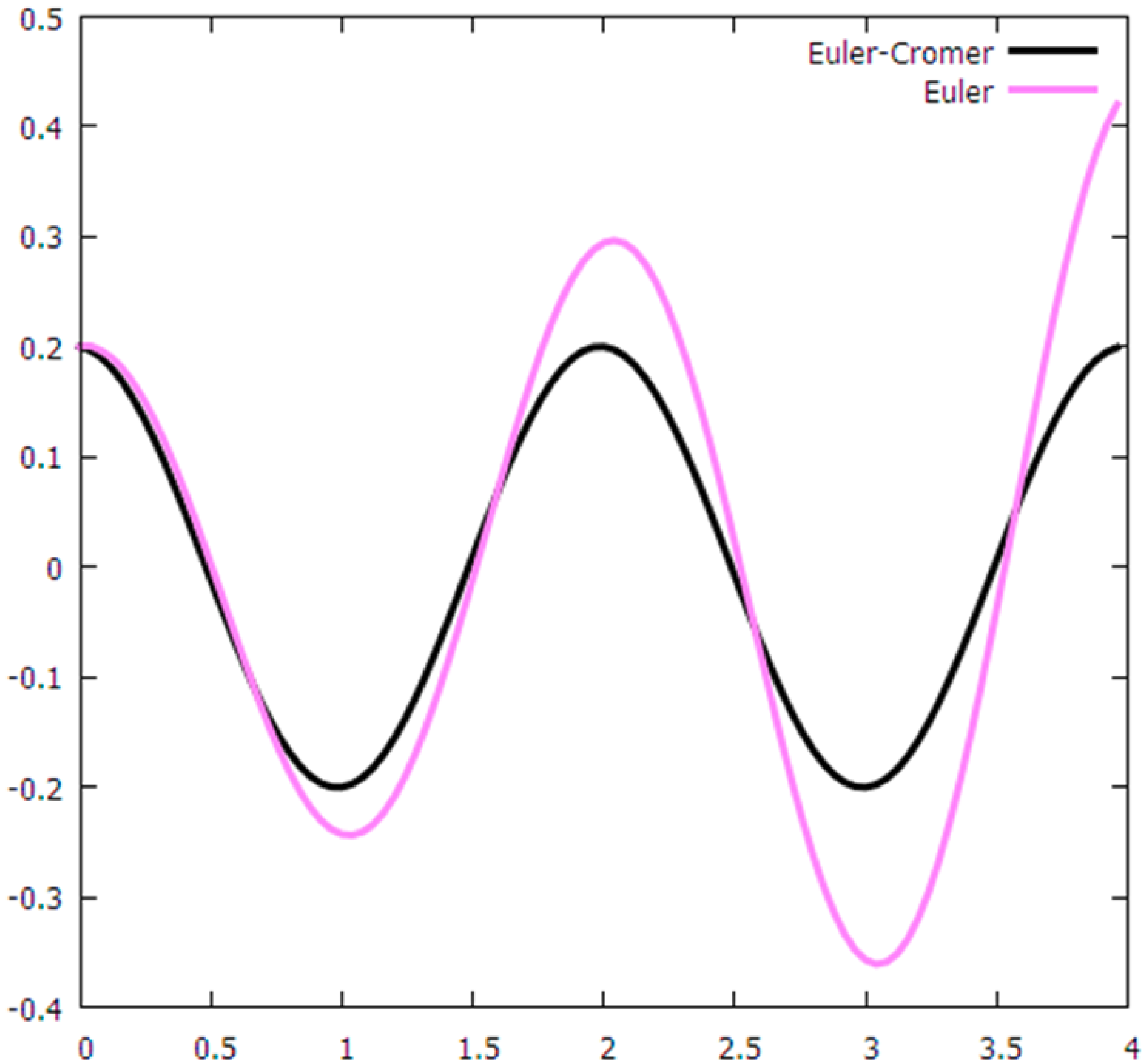
7 E 5 1 2

```
program filereader
implicit none
real,allocatable ::coeff(:),x(:),y(:),z(:)
character(len=10),allocatable:: ar(:)
integer::N,i,num
N = 0
OPEN (1, file ='data.txt')
DO
      READ (1,*, END=10)
      N = N + 1
```

```
END DO
10 CLOSE (1)
print*,N
allocate (coeff(N),ar(N),x(N),y(N),z(N))  !allocate the arrays from the text file
!Reading the coefficients from the file
open (unit = 1, file ='data.txt', status ='old')
do i = 1,N
       read (1,*) coeff(i),ar(i),x(i),y(i),z(i)
end do
close (1)

do i =1,N
       write(*,*) coeff(i),ar(i),x(i),y(i),z(i)
end do
end program filereader
```

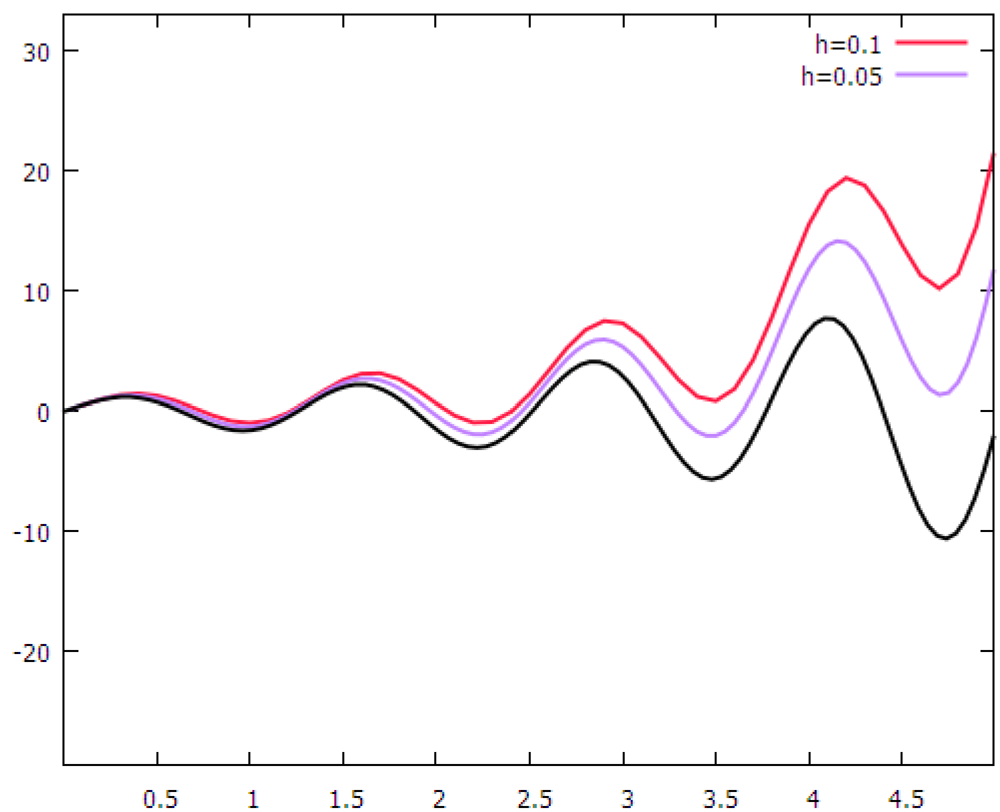
Differential Equations

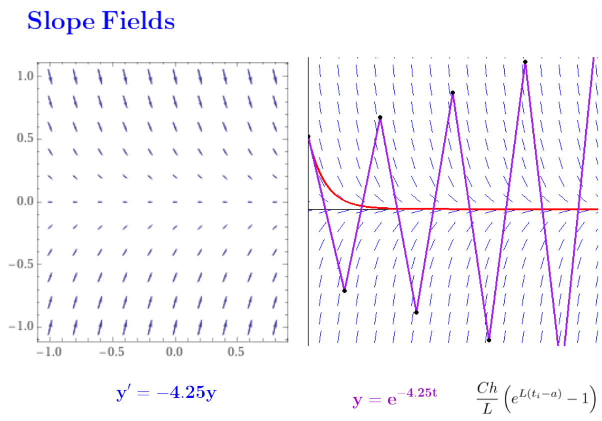
```
clc;clear;
length= 1;        %pendulum length
g=9.8;            %surface acceleration
n = 250;
dt = 0.04;
%q=0.5
omega = zeros(n,1);
theta = zeros(n,1);
time = zeros(n,1);
theta(1)=0.2;
for step = 1:n-1 % loop over timesteps
%omega(step+1) = omega(step) - (g/length)*theta(step)*dt;  %Euler
omega(step+1) = omega(step) - (g/length)*theta(step)*dt-q*omega(step)*dt; %Damped
theta(step+1) = theta(step)+omega(step+1)*dt                             %Euler
Cromer
time(step+1) = time(step) + dt;
end
plot(time,theta,'r' );
xlabel('time (seconds) ');
ylabel('theta (radians)');
```



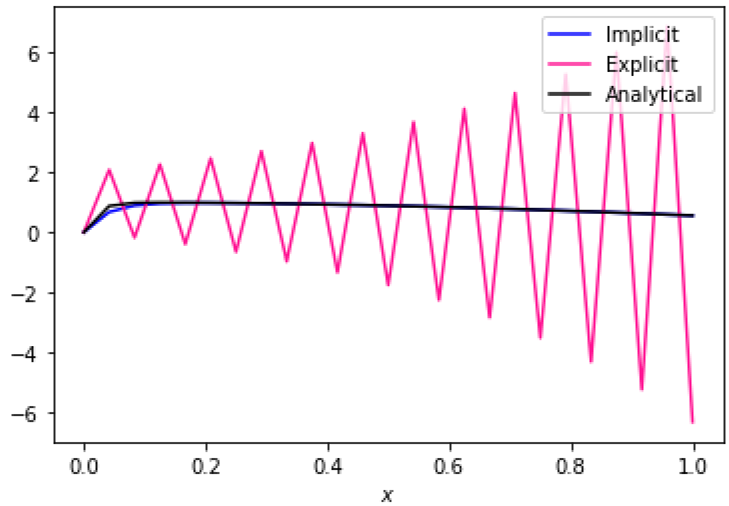

```
#Last update:Piyush   8 Aug 2022
#required: try making a code that automatically determines the
#derivative instead of relying on the user input
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0,4,100)
def euler_explicit(f_prime, y_0, a, b, h):#defining the euler method
   N=int((b-a)/h)        #Number of steps
   x = a ; y = y_0       #Initial Values to the equation y(0) and x(0)
   x_out,y_out =[],[]
   for i in range(N):
       y = y + h*f_prime(x, y)  #y_n+1 = y_n + h * f
       x = x + h
       x_out.append(x)
       y_out.append(y)
   return x_out, y_out
```

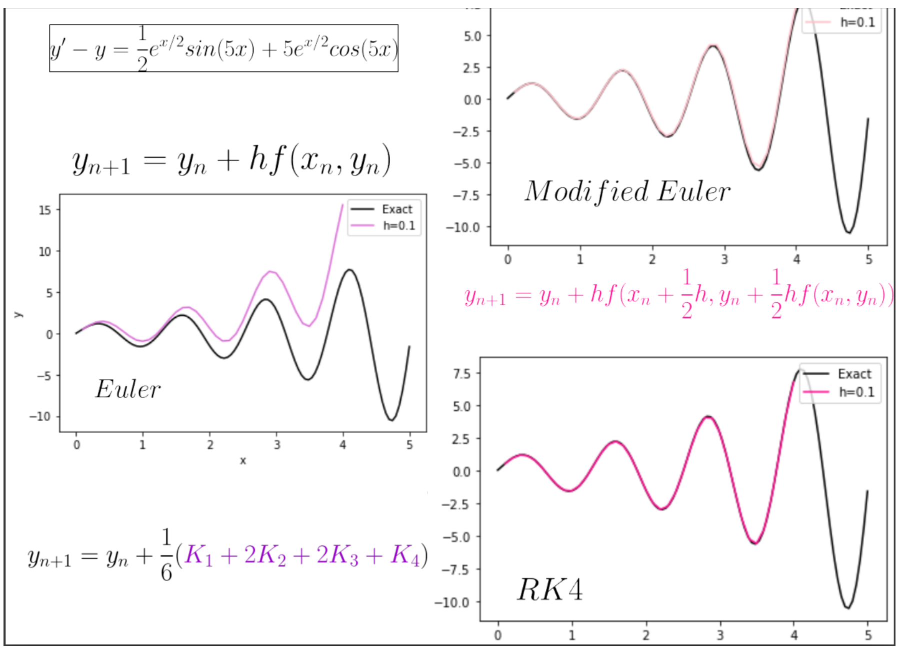
```
def solution(x):
 return np.exp(x/2)*np.sin(5*x)
def f_prime(x, y):
 return -0.5*np.exp(x/2)*np.sin(5*x)+5*np.exp(x/2)*np.cos(5*x)+y
  #return y*np.exp(y)+np.exp(y)
x_euler, y_euler = euler_explicit(f_prime, 0, 0, 3, 0.1)  #call the function and
store the values in x and y
#plt.xlim([0,2])#plt.ylim([0.7,1.1])
plt.figure(figsize=(7,6))
plt.plot(x,solution(x),linewidth=3.0,color='mediumslateblue',label='Exact')
plt.plot(x_euler,y_euler,linewidth=3.0,color='orchid',label='Euler-Explicit')
plt.xlabel(r'$x$',fontsize=15);plt.ylabel(r'$y$',fontsize=15)
plt.legend(loc='upper right')
```





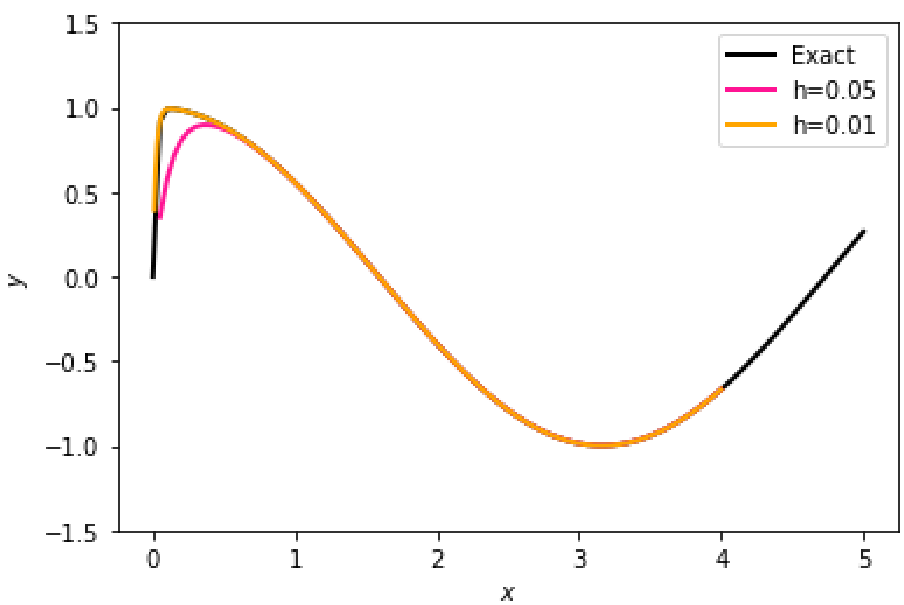

```
#Euler Explicit and Euler Implicit
#Last update Piyush
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 1, 25)   #gridpoints
def euler_explicit(x):
       y = np.zeros_like(x)  #empty array
       h = x[1] - x[0]   #step size
       for i in range(1, len(x)):
           y[i] = y[i-1] -50*h*(y[i-1] - np.cos(x[i]))
       return y
def euler_implicit(x):
       y = np.zeros_like(x)  #empty array
       h = x[1] - x[0]    #step size
       for i in range(1, len(x)):
           y[i] = (y[i-1] + 50*h*np.cos(x[i])) / (50*h + 1)
       return y
def solution(x):
       return (50/2501)*(np.sin(x) + 50*np.cos(x)) - (2500/2501)*np.exp(-50*x)
plt.plot(x,euler_implicit(x),color='blue',label='Implicit')
plt.plot(x,euler_explicit(x),color='deeppink',label='Explicit')
plt.plot(x,solution(x),color='black',label='Analytical')
plt.xlabel(r'$x$')
plt.legend(loc='upper right')
```






```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 5, 100)
def RK4(f_prime, y_0, a, b, h):
   N=int((b-a)/h)  #Number of steps
   x = a ; y = y_0
   x_out,y_out =[],[]
   for i in range(N):
       k1 = h*f_prime(x,y)
       k2 = h*f_prime(x+0.5*h,y+0.5*k1)
       k3 = h*f_prime(x+0.5*h,y+0.5*k2)
       k4 = h*f_prime(x+h,y+k3)

       y = y+ (1/6)*(k1+2*k2+2*k3+k4)   #RK-4
```

```
        x = x + h
        #print(x,y)
        x_out.append(x)
        y_out.append(y)
    return x_out, y_out
def f_prime(x, y):  #Differential equation
 return 50*(np.cos(x)-y)
def solution(x):    #Analytical solution
 return (50/2501)*(np.sin(x)+50*np.cos(x)-50*np.exp(-50*x))  #y(0)=0
x_RK, y_RK = RK4(f_prime, 0, 0, 4, 0.05)
x_RK1, y_RK1 = RK4(f_prime, 0, 0, 4, 0.01)

plt.xlabel(r'$x$');plt.ylabel(r'$y$')
plt.ylim([-1.5,1.5])
plt.plot(x,solution(x),color='black',linewidth=2.0,label='Exact')
plt.plot(x_RK,y_RK,color='deeppink',linewidth=2.0,label='h=0.05')
plt.plot(x_RK1,y_RK1,color='orange',linewidth=2.0,label='h=0.01')
plt.legend(loc='upper right')
```



$$y_{n+1} = y_n + \frac{3}{2}hf(t_{n+1}, y_{n+1}) - \frac{1}{2}hf(t_n, y_n)$$

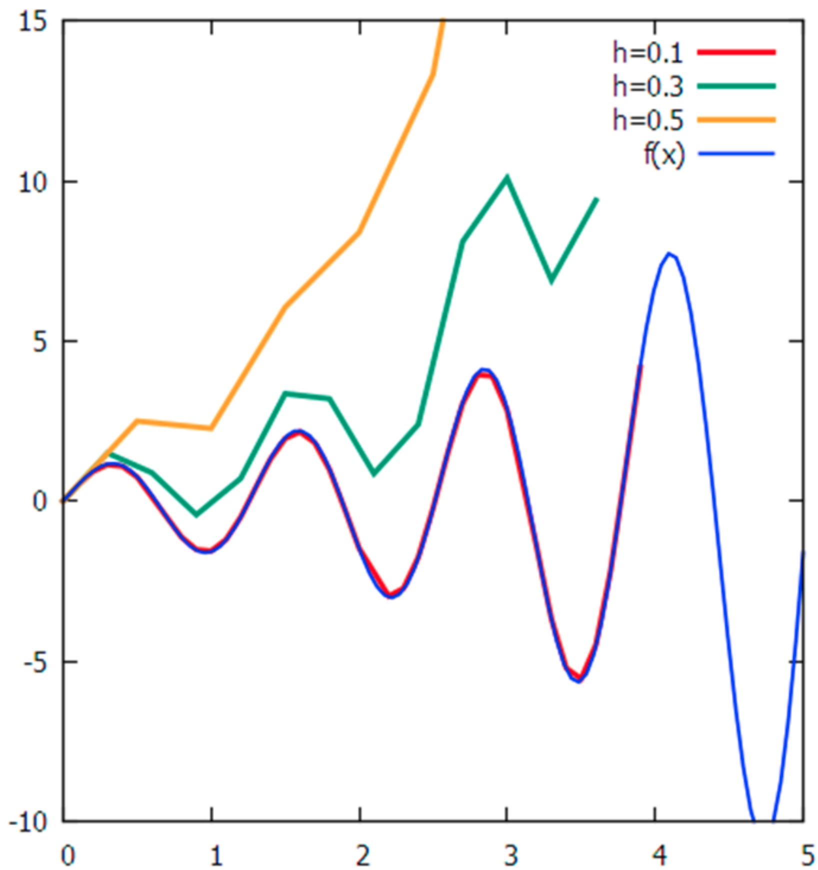
$Adams - Bashforth - Predictor$

$$y_{n+1} = y_n + \frac{h}{2}(f(t_{n+1}, y_{n+1}) + f(t_n, y_n))$$

$Adams - Moulton - Corrector$

```
program Predictor
IMPLICIT NONE
REAL::a,b,h,y_0,f_prime
INTEGER::N,i
real, allocatable :: x(:), y(:)
a=0
b=4
h=0.1
y_0=0
N=INT((b-a)/h) !Number of steps
allocate(x(N),y(N))
x(1) = a
y(1) = y_0
x(2) = a+h       !Adam Bashforth 2 Step Method
y(2) = y_0+h*f_prime(x(1),y(1))
do i=2,N
   y(i+1) = y(i) +
(h/2)*(3*f_prime(x(i),y(i))-f_prime(x(i-1),y(i-1)))
   x(i+1) = x(i) + h
   y(i+1) = y(i) + (h/2)*(f_prime(x(i+1),y(i+1))+f_prime(x(i),y(i)))

end do
do i = 1,N
   write(*,*) x(i),y(i)
   x(i)=x(i)+h
 end do
end program Predictor
```

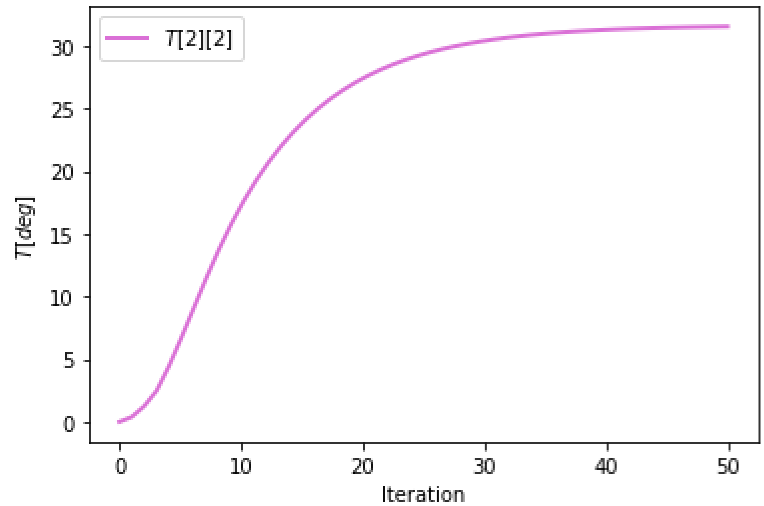
```
REAL function f_prime(x1,y1)
REAL::x1,y1
 f_prime=-0.5*EXP(x1/2)*SIN(5*x1)+5*EXP(x1/2)*COS(5*x1)+y1
return
end function
```

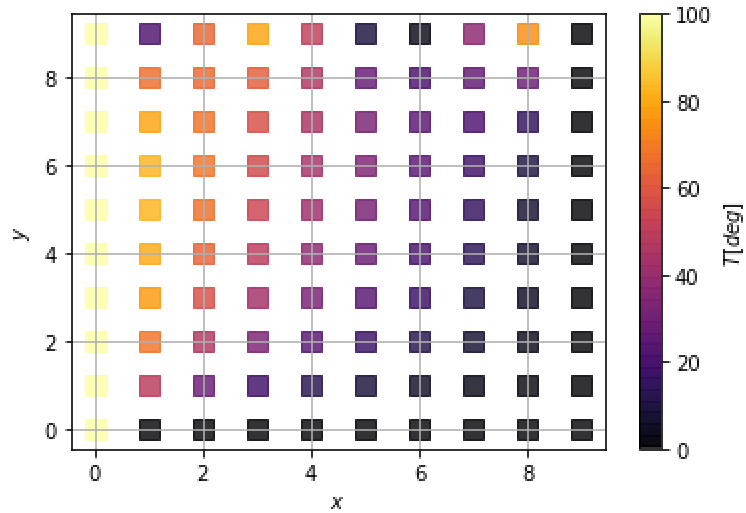


Heat Equation :Laplace 2D Equation

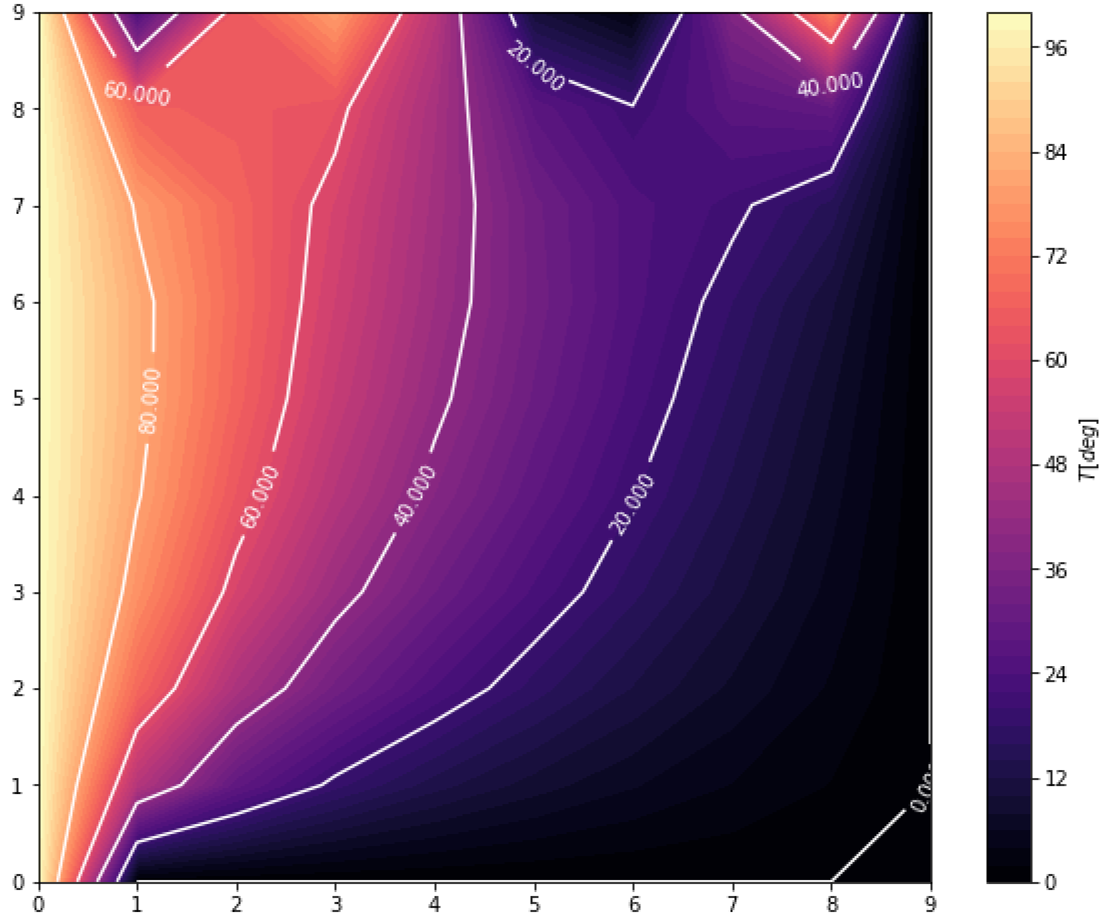
```
import numpy as np
import matplotlib.pyplot as plt
N = 50     #max iter
# Set Dimension and delta
lenX = lenY = 10
delta = 1
# Boundary condition
```

```
#Ttop      = 30
Tbottom   = 0
Tleft     = 100
Tright    = 0
Tguess = 0   #intial guess for internal grid
X, Y = np.meshgrid(np.arange(0, lenX), np.arange(0, lenY))
T = np.empty((lenX, lenY))
T.fill(Tguess)

x2=np.linspace(0,lenX,lenX)  #setting boundary condition
for m in range(0,lenX):
 T[(lenY-1):, :]  = 80*np.sin(x2/2)**2
T[:1, :]          = Tbottom
T[:, (lenX-1):]   = Tright
T[:, :1]          = Tleft
Tcen,Tcen1=[],[]
for k in range(0,N):
   Tcen.append(T[5][5])
   #Tcen1.append(T[7][3])
   for i in range(1, lenX-1, delta):
       for j in range(1, lenY-1, delta):
           T[i, j] = 0.25 * (T[i+1][j] + T[i-1][j] + T[i][j+1] +
T[i][j-1])
print(T)
x1 = np.linspace(0,N,N)
plt.xlabel('Iteration')
plt.ylabel(r'$T[deg]$')
plt.plot(x1,Tcen,linewidth=2.0,color='orchid',label=r'$T[2][2]$')
#plt.plot(x1,Tcen1,linewidth=2.0,color='#50C878',label=r'$T[7][3]$')
plt.legend()
```

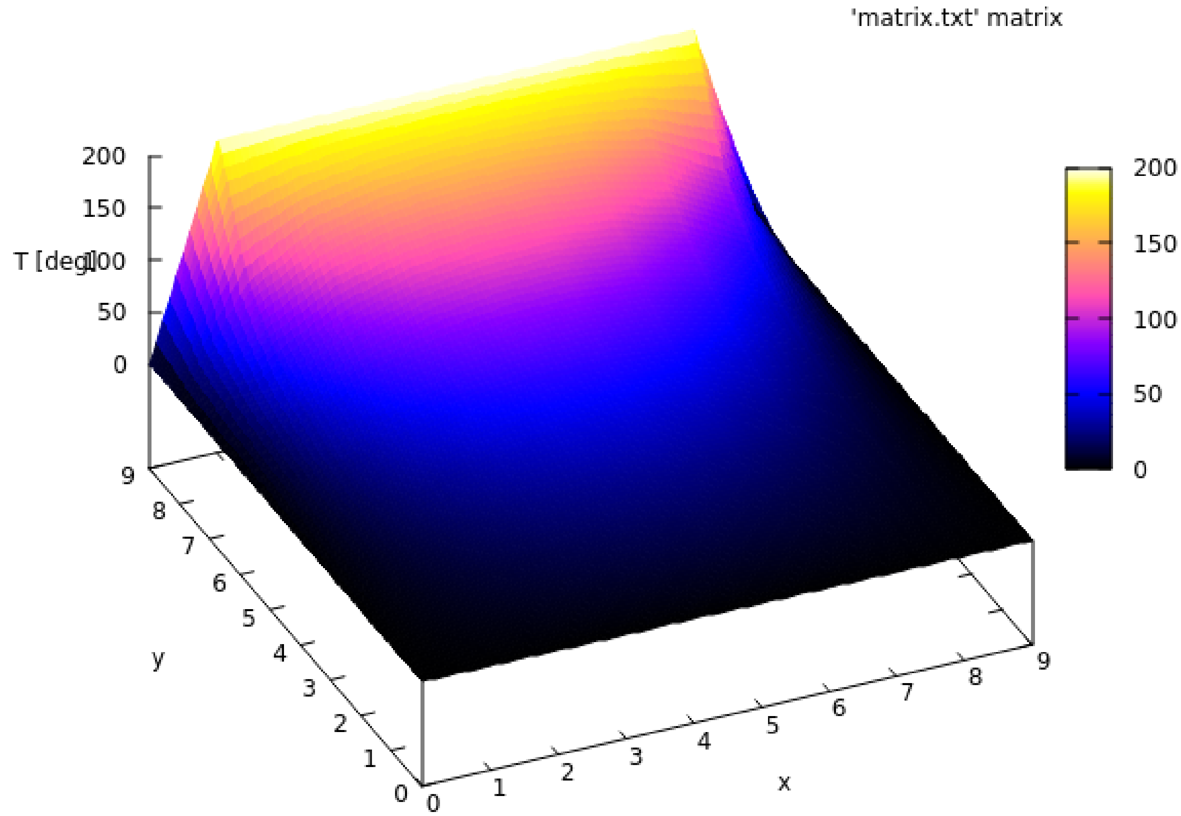





```
plt.figure(figsize=(10,8))
contours = plt.contour(X, Y, T, 5,colors='white') # 5 contours
plt.clabel(contours, inline=True, fontsize=10.0)
plt.contourf(X, Y, T, 60, cmap=plt.cm.magma)
plt.colorbar(label=r'$T [deg]$')
```

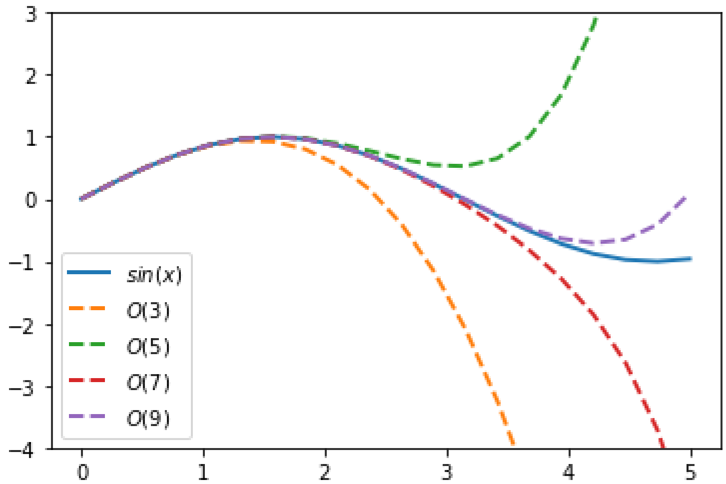


60.000
20.000
40.000
80.000
60.000
40.000
20.000
0.000
0
1
2
3
4
5
6
7
8
9
96
84
72
60
48
36
24
12
0
T[deg]



Taylor Series

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0,5,20)
plt.ylim(-4, 3)
y1 = x  - x**3/6
y2 = y1 + x**5/120
y3 = y2 - x**7/5040
y4 = y3 + x**9/362880
plt.plot(x, np.sin(x),'-',linewidth=2.0,label=r'$sin(x)$')
plt.plot(x,y1,'--',linewidth=2.0,label=r'$O(3)$')
plt.plot(x,y2, '--',linewidth=2.0,label=r'$O(5)$')
plt.plot(x,y3, '--',linewidth=2.0,label=r'$O(7)$')
plt.plot(x,y4, '--',linewidth=2.0,label=r'$O(9)$')
plt.legend(loc='lower left')
```



3
2
1
0
−1
−2
−3
−4
0
1
2
3
4
5
sin(x)
O(3)
O(5)
O(7)
O(9)