Mathematical Physics - III (Practical)
Paper: PHS-A-CC-4-8-P

1. Solution of ODE/PDE:
(a) Initial value problem: Modified-Euler and Runge-Kutta second order and fourth order methods.
(b) Boundary value problems: Finite difference method with fixed step size.
Application to simple physical problems.

(a) Initial value problem: Modified-Euler and Runge-Kutta second order and fourth order methods.

Ordinary differential equations (ODE)

Let us consider a first order differential equation with initial condition:

$$y' = f(x,y), y(x_0) = y_0 \tag{1}$$

The differential equation (1) is equivalent to the following integral equation

$$y = y_0 + \int_{x_0}^{x} f(x,y)dx \tag{2}$$

The equation (2) can be used to obtain a solution by successive approximations.

In general

$$y_n = y_0 + \int_{x_0}^{x} f(x,y_{n-1})dx \tag{3}$$

This scheme of successive approximation is used to write algorithms for solving a first order ordinary differential equation.

Euler Method:

Formally the Euler's method algorithm is

$$y_{n+1} = y_n + hf(x_n, y_n)$$

The midpoint method uses an Euler step to evaluate the derivative in the midpoint of the interval as follows:

$$y_0 = a$$

$$y_{n+1} = y_n + h\left[f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}f(x_n, y_n)\right)\right]$$

The modified Euler method is the average of the midpoint method and the Euler method.

$$y_0 = a$$

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_n + hf(x_n, y_n))]$$

Runge-Kutta methd:

2nd Order:

General 2nd order Runge-Kutta scheme:

$$K_1 = h * f(x_n, y_n)$$

$$K_2 = h\, f\big((x_n + \alpha h), (y_n + \beta K_1)\big)$$

$$y_{n+1} = y_n + a_1 K_1 + a_2 K_2$$

With

$$a_1 + a_2 = 1$$

$$\propto a_2 = \beta a_2 = \frac{1}{2}$$

Since we have three equations and four unknown $(a_1, a_2, \propto, \beta)$ there are infinitely many solutions

The most popular are:

(i) $\alpha = \beta = \frac{1}{2}$ and $a_1 = 0, a_2 = 1$

$$K_1 = h * f(x_n, y_n)$$

$$K_2 = h\, f\left(\left(x_n + \frac{h}{2}\right), (y_n + \frac{1}{2}K_1)\right)$$

$$y_{n+1} = y_n + K_2$$

(ii) $\alpha = \beta = 1$ and $a_1 = \frac{1}{2}, a_2 = \frac{1}{2}$

$$K_1 = h * f(x_n, y_n)$$

$$K_2 = h\, f\big((x_n + h), (y_n + K_1)\big)$$

$$y_{n+1} = y_n + \frac{1}{2}(K_1 + K_2)$$

(iii)

$\alpha = \beta = \frac{2}{3}$ and $a_1 = \frac{1}{4}, a_2 = \frac{3}{4}$

$$K_1 = h * f(x_n, y_n)$$

$$K_2 = h\, f\left(\left(x_n + \frac{2}{3}h\right), (y_n + \frac{2}{3}K_1)\right)$$

$$y_{n+1} = y_n + \frac{1}{4}(K_1 + 3K_2)$$

$$\therefore\ y_{n+1} = y_n + \frac{h}{4}\left[f(x_n, y_n) + 3f(x_n + \frac{2}{3}h, y_n + \frac{2}{3}hf(x_n, y_n))\right]$$

<u>4th Order Runge-Kutta Scheme:</u>

The most widely used Runge-Kutta scheme is the 4th order scheme RK4 based on Simpson's rule

$$y_{n+1} = y_n + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

Where

$$K_1 = h * f(x_n, y_n)$$

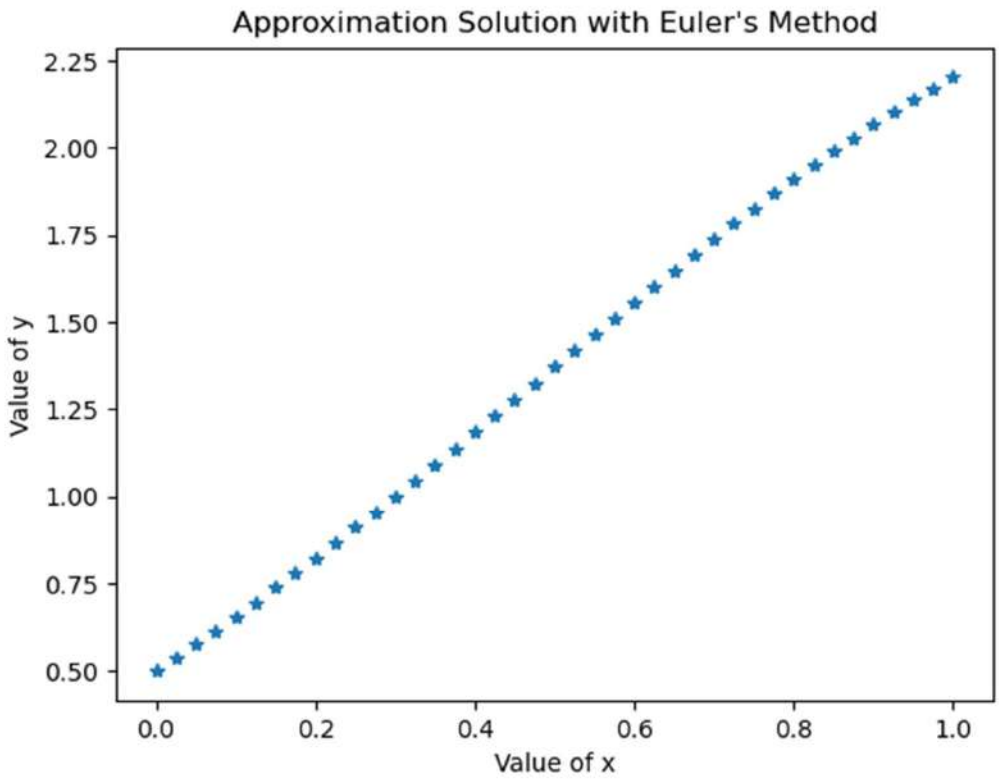
$$K_2 = h\, f\left(\left(x_n + \frac{h}{2}\right), (y_n + \frac{1}{2}K_1)\right)$$

$$K_3 = h\, f\left(\left(x_n + \frac{h}{2}\right), (y_n + \frac{1}{2}K_2)\right)$$

$$K_4 = h\, f\big((x_n + h), (y_n + K_3)\big)$$

(1).

```
# Euler method
import numpy as np
from matplotlib import pyplot as plt

def f(x,y): return y-2*x**2+1
#def f(x,y): return (1+x)*y + 1 - 3*x + pow(x,2)
#def f(x,y): return y-x**2+1
#def f(x,y): return (x+y+x*y)
#def f(x,y): return x+2*y


a=0
b=1
N=40
h=(b-a)/float (N)
#print (h)
y=np.zeros((N+1, ))
#print (y)
x=np.arange(a, b+h, h)
#print (x)
x[0]=0
y[0]=0.5
for i in range (1, N+1):
   y[i]=y[i-1]+h*f(x[i-1], y[i-1])
#print (x[N],y[N])
##print("x_n\t    y_n")
##for i in range(1, N+1):
##        print(x[i],"\t",format(y[i],'6f'))
plt.plot(x,y,'*', lw=1)
plt.xlabel("Value of x")
plt.ylabel("Value of y")
plt.title("Approximation Solution with Euler's Method")
plt.savefig('Euler.png')
plt.show()
```

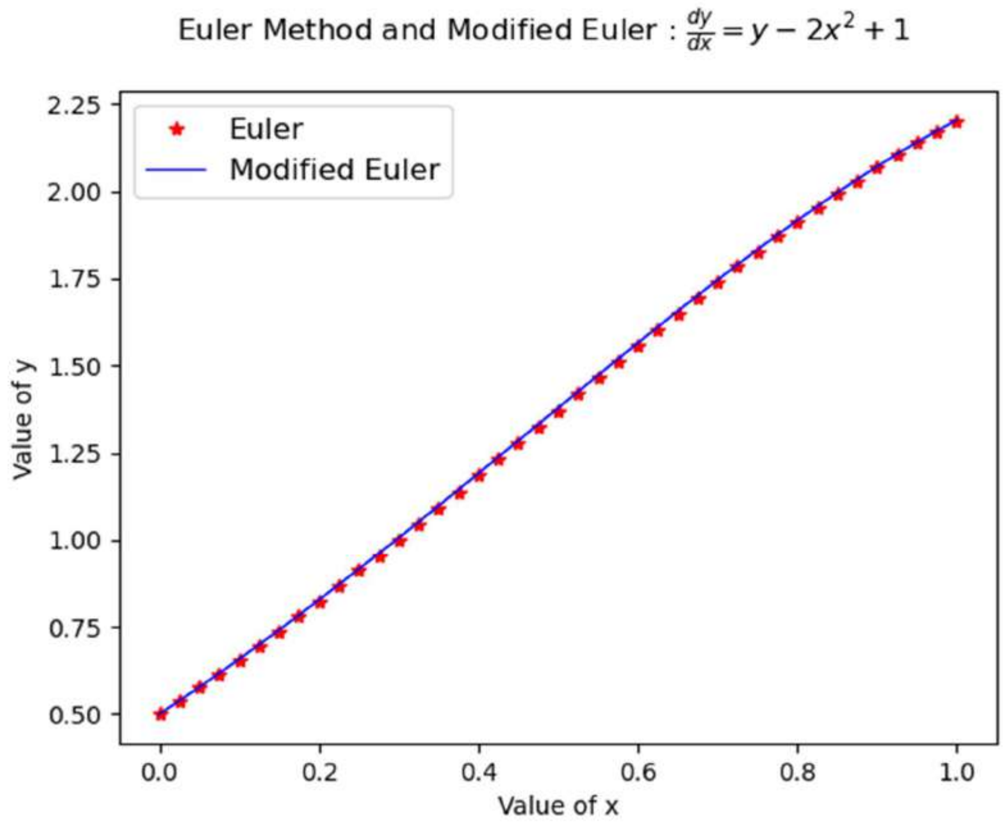


(2).

```
# Euler method and Modified Euler Method
import numpy as np
from matplotlib import pyplot as plt


def f(x,y): return y-2*x**2+1
#def f(x,y): return (1+x)*y + 1 - 3*x + pow(x,2)
#def f(x,y): return y-x**2+1
```

```
#def f(x,y): return (x+y+x*y)
#def f(x,y): return x+2*y
a=0
b=1
N=40
h=(b-a)/float (N)
#print (h)
y=np.zeros((N+1, ))
#print (y)
x=np.arange(a, b+h, h)
#print (x)
x[0]=0
y[0]=0.5
for i in range (1, N+1):
    y[i]=y[i-1]+h*f(x[i-1], y[i-1])
#print (x[N],y[N])
##print("x_n\t    y_n")
##for i in range(1, N+1):
##        print(x[i],"\t",format(y[i],'6f'))
yc=np.zeros((N+1, ))
#print (yc)
xc=np.arange(a, b+h, h)
#print (xc)
xc[0]=0
yc[0]=0.5

for i in range (1, N+1):
    f1=f(xc[i-1], yc[i-1])
    f2=f( xc[i], yc[i-1]+h*f1 )
    yc[i]=yc[i-1]+h*(f1+f2)/2.0
```

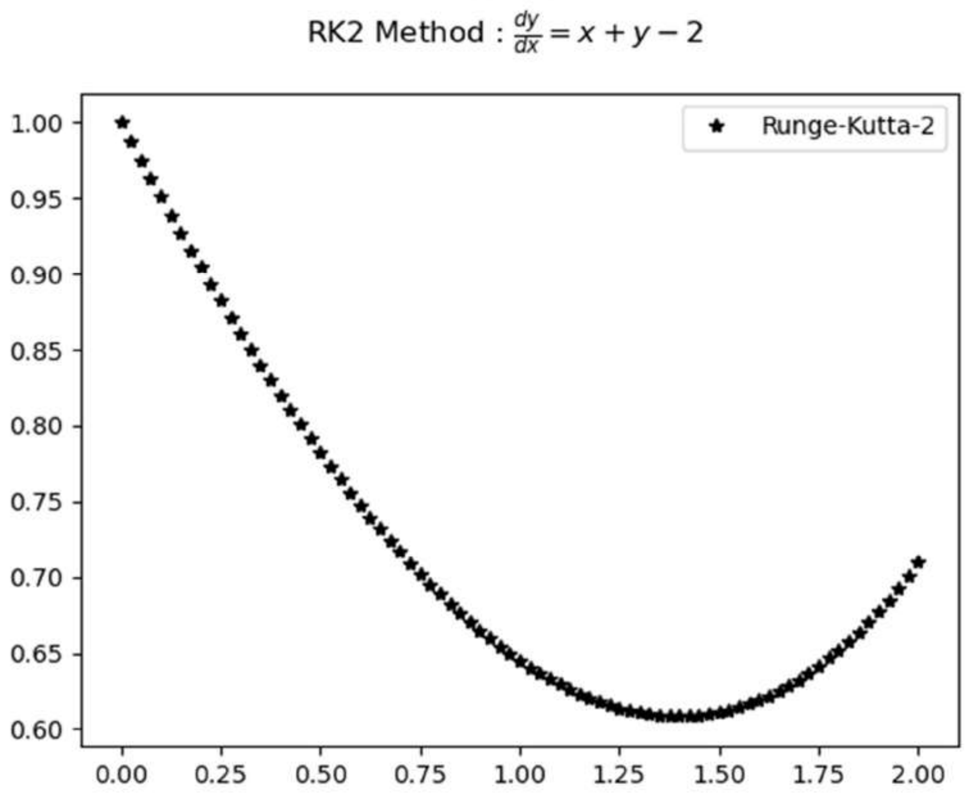
```
##print (xc[N], yc[N])
##print("xc_n\t    yc_n")
##for i in range(1, N+1):
##        print(xc[i],"\t",format(yc[i],'6f'))

plt.figure()
plt.plot(x,y,'*', c='r', lw=1, label='Euler')
plt.plot(xc,yc,'-', c='b', lw=1, label='Modified Euler')
plt.xlabel("Value of x")
plt.ylabel("Value of y")
plt.suptitle(r'Euler Method and Modified Euler : $\frac{dy}{dx} = y-2x^2+1$',size=12)
plt.legend(loc='best', prop={'size':12})
plt.savefig('eu-meu.png')
plt.show()
```



(3).(a).

```
# Runge Kutta 2nd order dy/dx=x+y-2
import numpy as np
import matplotlib.pyplot as plt

def f(x,y):
    return x+y-2

a = 0
b = 2
N = 80

##x = 0.0
##y = 1.0



h = ((b-a)/float(N)) # determine step-size
print (h)
y = np.zeros((N+1,))
x = np.arange( a, b+h, h ) # create mesh

#print (x, y)

x[0]=0.0
y[0]=1.0

for i in range(1,N+1):
    k1 = h * f( x[i-1], y[i-1] )
    k2 = h * f( x[i-1] + h / 2.0, y[i-1] + k1 / 2.0 )
    y[i] = y[i-1] + (1.0/6.0)*(k1+2*k2)
```

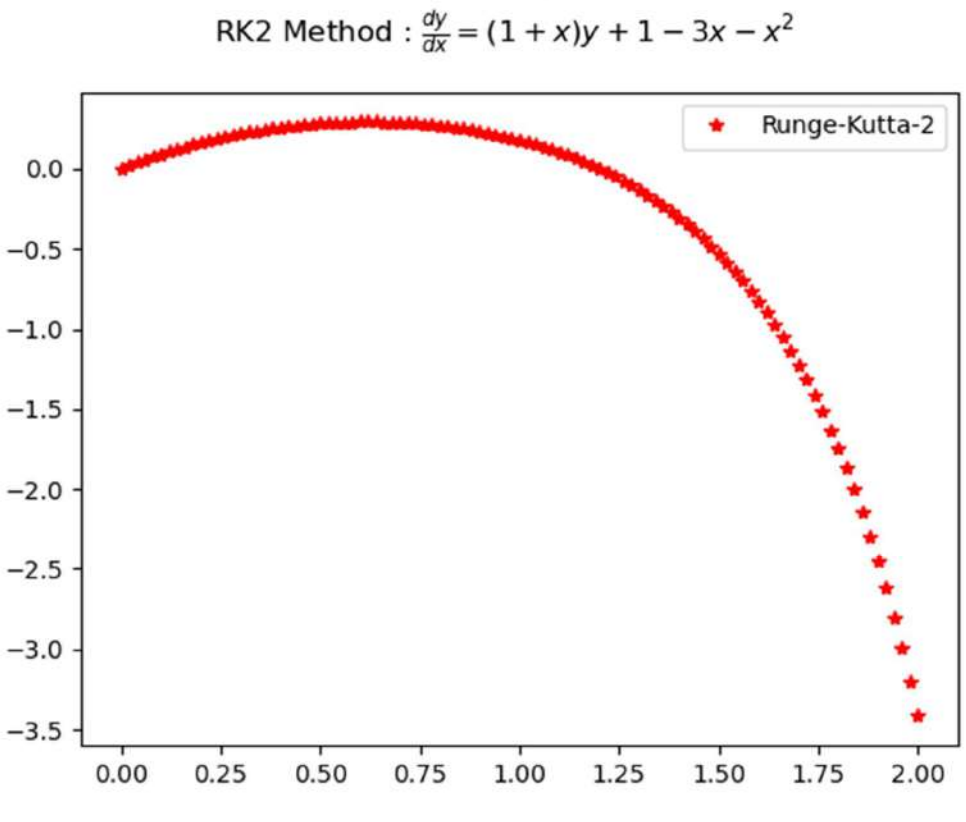
```
print (x[N],y[N])
##print("x_n\t    y_n")
##for i in range(1, N+1):
##      print(x[i],"\t",format(y[i],'6f'))

#print (x,y)

plt.figure()
plt.plot (x, y, '*', lw=2, c='k', label='Runge-Kutta-2')
plt.suptitle(r'RK2 Method : $\frac{dy}{dx} = x+y-2$',size=12)
plt.legend(loc='best')
plt.show()
```



(b).

```
# Runge Kutta 2nd order dy/dx=(1+x)*y+1-3*x+x*x
import numpy as np
import matplotlib.pyplot as plt

def f(x,y):
    return (1+x)*y+1-3*x+x*x

a = 0
b = 2
N = 100

##x = 0.0
##y = 1.0



h = ((b-a)/float(N)) # determine step-size
print (h)
y = np.zeros((N+1,))
x = np.arange( a, b+h, h ) # create mesh

#print (x, y)

x[0]=0.0
y[0]=0.0

for i in range(1,N+1):
    k1 = h * f( x[i-1], y[i-1] )
    k2 = h * f( x[i-1] + h, y[i-1] + k1 )
    y[i] = y[i-1] + (1.0/2.0)*(k1+k2)
```

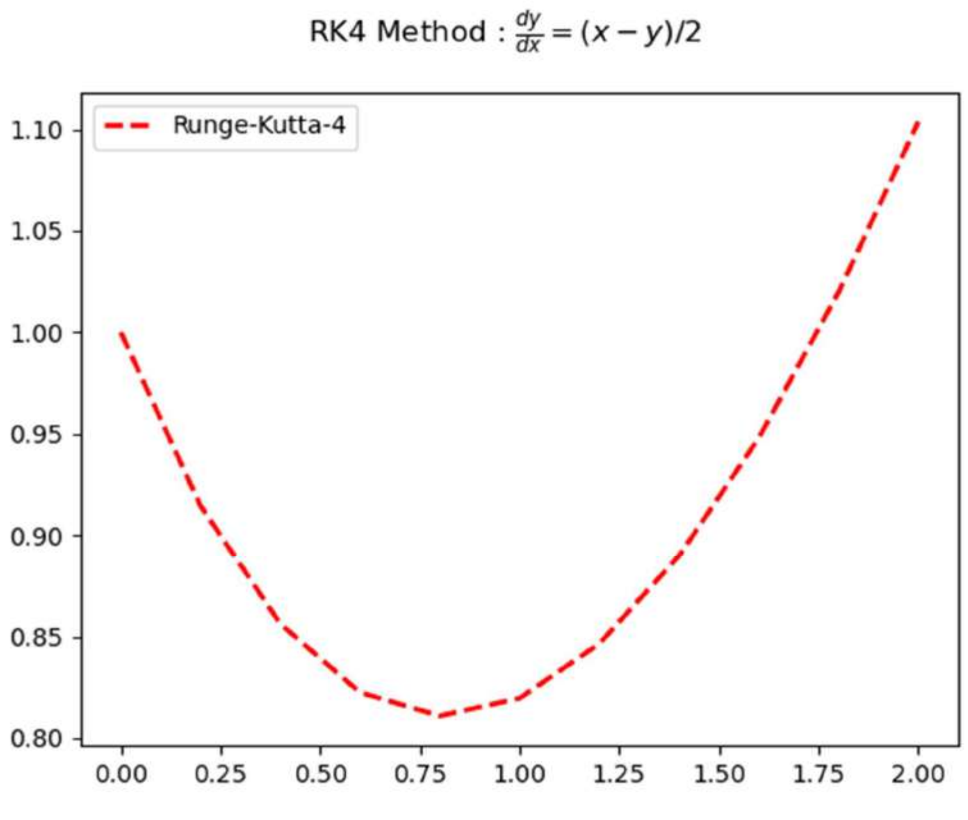
```
print (x[N],y[N])
##print("x_n\t    y_n")
##for i in range(1, N+1):
##      print(x[i],"\t",format(y[i],'6f'))

#print (x,y)

plt.figure()
plt.plot (x, y, '*', lw=2, c='r', label='Runge-Kutta-2')
plt.suptitle(r'RK2 Method : $\frac{dy}{dx} = (1+x)y+1-3x-x^2$',size=12)
plt.legend(loc='best')
plt.show()
```



(4). (a).

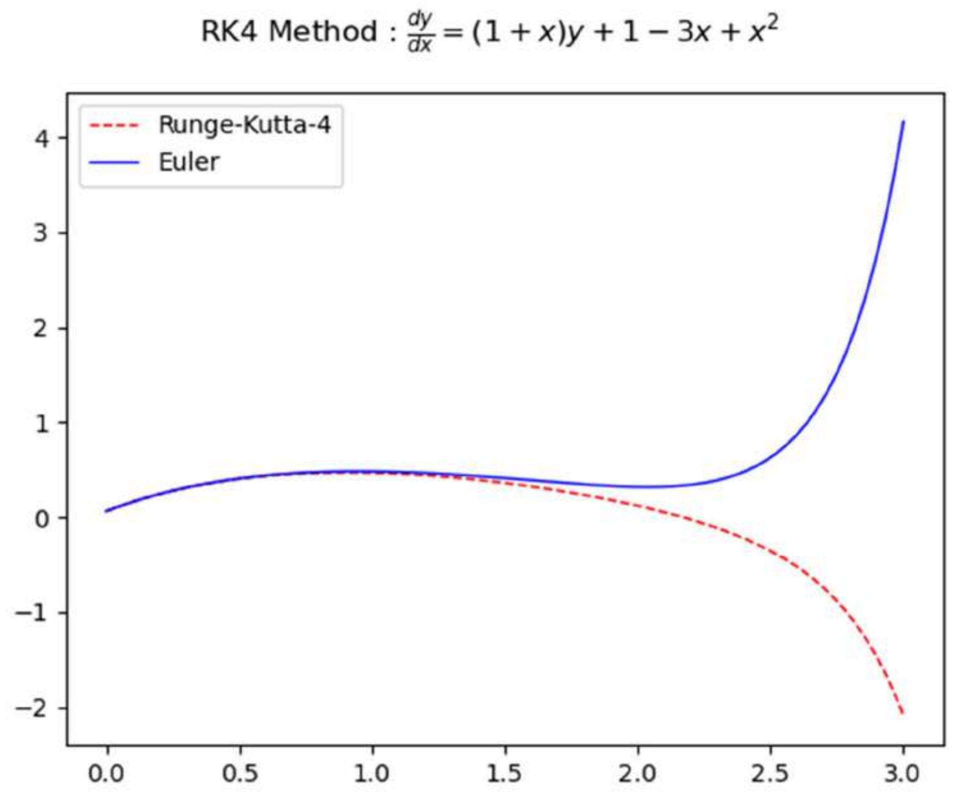
```
# Runge Kutta 4th order dy/dx=(x-y)/2
import numpy as np
import matplotlib.pyplot as plt

def f(x,y):
    return (x-y)/2
a = 0
b = 2
N = 10

##x = 0.0
##y = 1.0
h = ((b-a)/float(N)) # determine step-size
print (h)
y = np.zeros((N+1,))
x = np.arange( a, b+h, h ) # create mesh

#print (x, y)

x[0]=0.0
y[0]=1.0

for i in range(1,N+1):
    k1 = h * f( x[i-1], y[i-1] )
    k2 = h * f( x[i-1] + h / 2.0, y[i-1] + k1 / 2.0 )
    k3 = h * f( x[i-1] + h / 2.0, y[i-1] + k2 / 2.0 )
    k4 = h * f( x[i-1]+h, y[i-1] + k3 )
    y[i] = y[i-1] + ( k1 + 2.0 * k2 + 2.0 * k3 + k4 ) / 6.0
```

```
print (x[N],y[N])
print("x_n\t    y_n")
for i in range(1, N+1):
     print(x[i],"\t",format(y[i],'6f'))
#print (x,y)
plt.figure()
plt.plot (x, y, '--', lw=2, c='r', label='Runge-Kutta-4')
plt.suptitle(r'RK4 Method : $\frac{dy}{dx} = (x-y)/2$',size=12)
plt.legend(loc='best')
plt.show()
```



4.(b) # Runge Kutta 4th order dy/dx=(1+x)*y + 1 - 3*x + pow(x,2)

```
import numpy as np
import matplotlib.pyplot as plt
def f(x,y):
    return (1+x)*y + 1 - 3*x + pow(x,2)

a = 0
b = 3
N = 300

h = (b-a)/float(N) # determine step-size
print (h)

ye=np.zeros((N+1, ))
xe=np.arange(a, b+h, h)
xe[0]=0.0
ye[0]=0.065

for i in range (1, N+1):
    ye[i]=ye[i-1]+h*f(xe[i-1], ye[i-1])



x = np.arange( a, b+h, h ) # create mesh
y = np.zeros((N+1,)) # initialize x

x[0] = 0.0
y[0] = 0.065
for i in range(1,N+1):
    k1 = h * f( x[i-1], y[i-1] )
    k2 = h * f( x[i-1] + h / 2.0, y[i-1] + k1 / 2.0 )
    k3 = h * f( x[i-1] + h / 2.0, y[i-1] + k2 / 2.0 )
    k4 = h * f( x[i-1]+h, y[i-1] + k3 )
    y[i] = y[i-1] + ( k1 + 2.0 * k2 + 2.0 * k3 + k4 ) / 6.0


##print (x[N],y[N])
##print("x_n\t    y_n")
##for i in range(1, N+1):
##       print(x[i],"\t",format(y[i],'6f'))
plt.figure()
plt.plot (x, y, '--', lw=1.0, c='r', label='Runge-Kutta-4')
plt.plot(xe,ye,'-', c='b', lw=1, label='Euler')
plt.suptitle(r'RK4 Method : $\frac{dy}{dx} = (1+x)y+1-3x+x^2$',size=12)
plt.legend(loc='best')
plt.show()
```

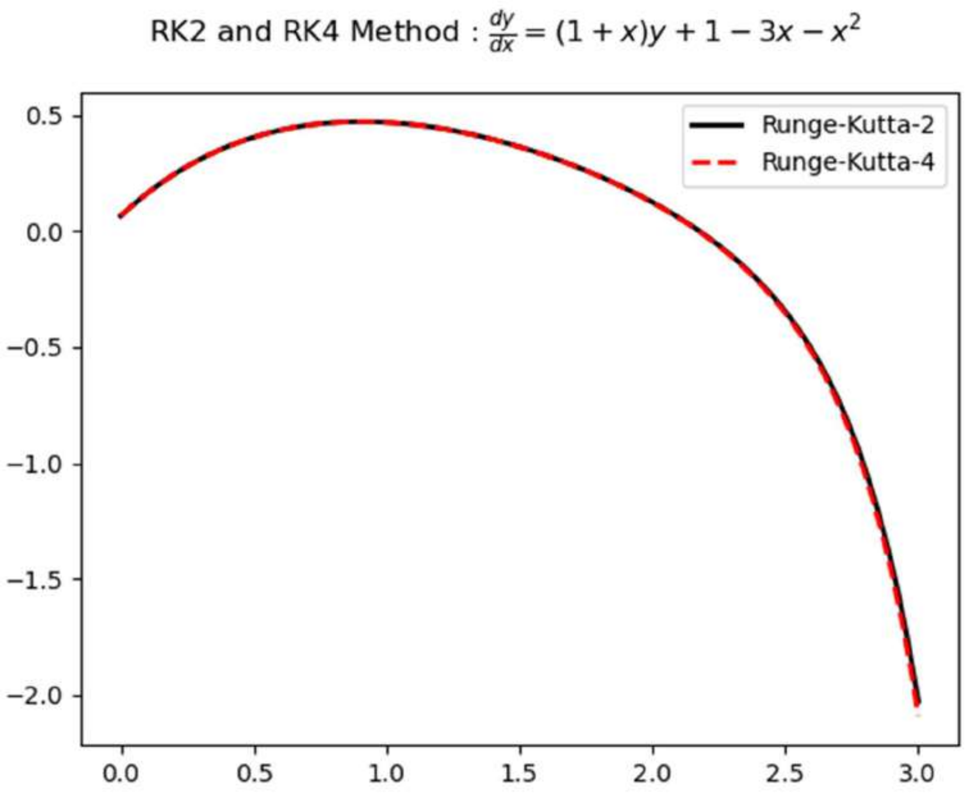


(c).

```
# comparison between Runge Kutta 2nd and 4th order dy/dx=(1+x)*y+1-3*x+x*x
import numpy as np
import matplotlib.pyplot as plt

def f(x,y):
    return (1+x)*y+1-3*x+x*x

a = 0
b = 3
N = 300

##x = 0.0
##y = 1.0



h = ((b-a)/float(N)) # determine step-size
print (h)
y = np.zeros((N+1,))
x = np.arange( a, b+h, h ) # create mesh

#print (x, y)

x[0]=0.0
y[0]=0.065

for i in range(1,N+1):
    k1 = h * f( x[i-1], y[i-1] )
    k2 = h * f( x[i-1] + h, y[i-1] + k1 )
    y[i] = y[i-1] + (1.0/2.0)*(k1+k2)
```

```
print (x[N],y[N])
##print("x_n\t    y_n")
##for i in range(1, N+1):
##      print(x[i],"\t",format(y[i],'6f'))

#print (x,y)




x4 = np.arange( a, b+h, h ) # create mesh
y4 = np.zeros((N+1,)) # initialize x
#print (x, y)

x4[0] = 0.0
y4[0] = 0.065

for i in range(1,N+1):
    k41 = h * f( x4[i-1], y4[i-1] )
    k42 = h * f( x4[i-1] + h / 2.0, y4[i-1] + k41 / 2.0 )
    k43 = h * f( x4[i-1] + h / 2.0, y4[i-1] + k42 / 2.0 )
    k44 = h * f( x4[i], y4[i-1] + k43 )
    y4[i] = y4[i-1] + ( k41 + 2.0 * k42 + 2.0 * k43 + k44 ) / 6.0


print (x4[N],y4[N])
##print("x4_n\t    y4_n")
##for i in range(1, N+1):
```

```
##      print(x4[i],"\t",format(y4[i],'6f'))


#print (x4,y4)


plt.figure()
plt.plot (x, y, '-', lw=2, c='k', label='Runge-Kutta-2')
plt.plot (x4, y4, '--', lw=2, c='r', label='Runge-Kutta-4')
plt.suptitle(r'RK2 and RK4 Method : $\frac{dy}{dx} =(1+x)y+1-3x-x^2 $',size=12)
plt.legend(loc='best')
plt.show()
```


RK2 and RK4 Method : $\frac{dy}{dx} = (1+x)y + 1 - 3x - x^2$
Runge-Kutta-2
Runge-Kutta-4