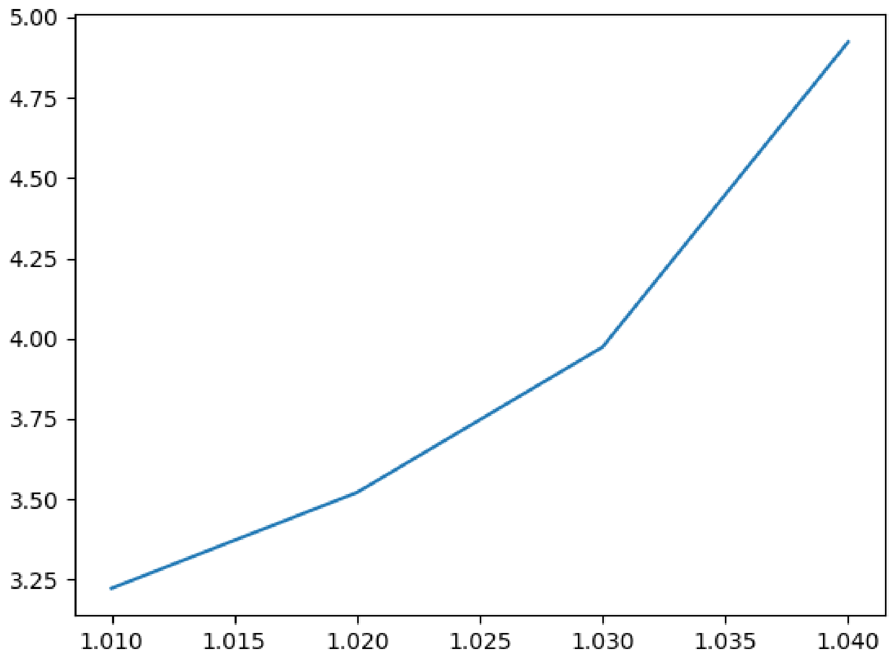
1. Write and execute a program to solve this initial-value problem: $x' = e^{xt} + \cos(x - t)$, $x(1) = 3$. Use the fourth-order Runge-Kutta formulas with $h = 0.01$. Stop the computation just before the solution overflows. Plot the solution.

Output:



Code (1):

```

import numpy as np
import matplotlib.pyplot as plt

#problem 1

fprime=lambda t,x: np.exp(x*t)+np.cos((x-t))
t=1
x=3
h=0.01
M=4
tref=[]
xref=[]
error=[]

for i in range(0,M):
    F1=h*fprime(t,x)
    F2=h*fprime(t+(h/2),x+(F1/2))
```

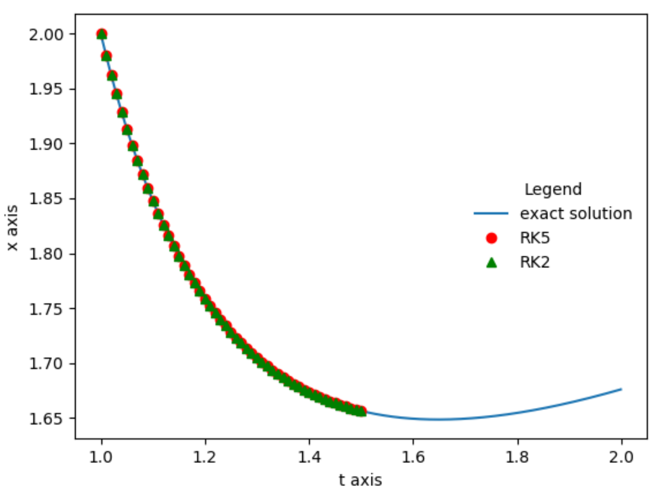
```
    F3=h*fprime(t+(h/2),x+(F2/2))
    F4=h*fprime(t+h,x+F3)
    x=x+(F1+2*F2+2*F3+F4)/6
    t=t+h
    tref.append(t)
    xref.append(x)

plt.figure(1)
plt.plot(tref,xref)
plt.xlabel("delta t")
plt.ylabel("Xn(t)")
plt.title('Runge-Kuta 4th order')
plt.show()
```

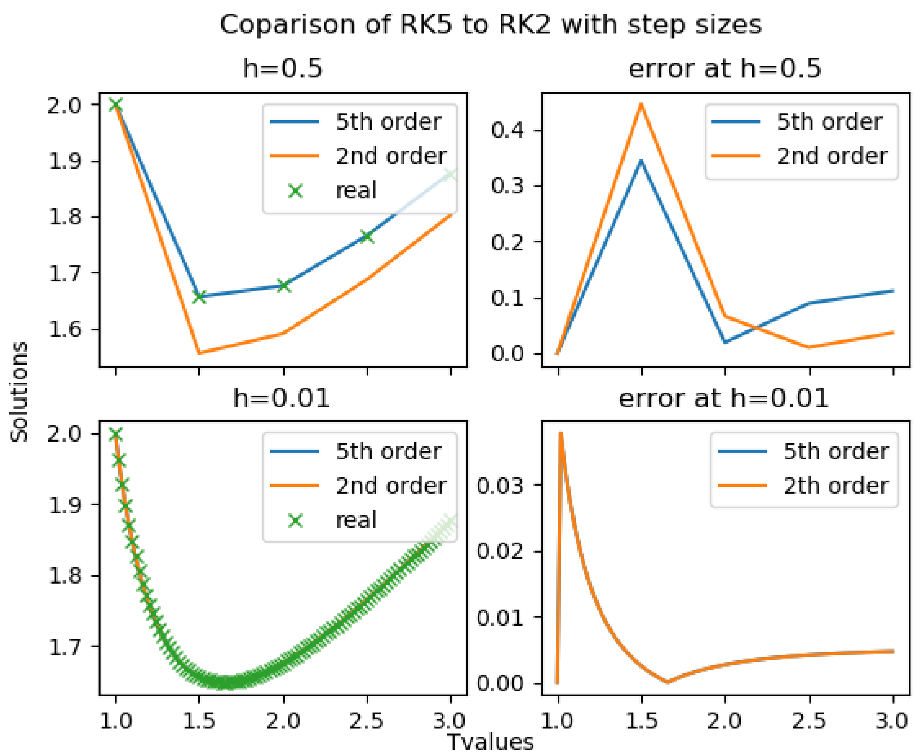
2. Numerically Compare the following fifth-order Runge-Kutta with the classical Runge-Kutta (second order) method on a problem with a known solution: $x(t+h) = x(t) + \frac{1}{24}F_1 + \frac{5}{48}F_4 + \frac{27}{56}F_5 + \frac{125}{336}F_6$ where

$$\begin{cases} F_1 = hf(t,x) \\ F_2 = hf(t+h/2, x+F_1/2) \\ F_3 = hf(t+h/2, x+F_1/4+F_2/4) \\ F_4 = hf(t+h, x-F_2+2F_3) \\ F_5 = hf(t+2h/3, x+7F_1/27+10F_2/27+F_4/27) \\ F_6 = hf(t+h/5, x+28F_1/625-F_2/5+546F_3/625+54F_4/625-378F_5/625) \end{cases}$$

The problem can be $x' = t^{-2}(tx - x^2)$, $x(1) = 2$ on the interval $[1,3]$ with the exact solution $x(t) = (1/2 + \ln t)^{-1}t$.

Output:





Code (2):

```

import numpy as np
import matplotlib.pyplot as plt
import math as m

#problem 2

def Runge_Kutta_5(f,fprime,t,x,h,M):
    xref=[x]
    error=[0]
    tlist=[t]
    for i in range(1,M+1):
        F1=h*fprime(t,x)
        F2=h*fprime(t+0.5*h,x+0.5*F1)
        F3=h*fprime(t+0.5*h,x+0.25*F1+0.25*F2)
        F4=h*fprime(t+h,x-F2+2*F3)
        F5=h*fprime(t+(2/3)*h,x+(7/27)*F1+(10/27)*F2+(1/27)*F4)
        F6=h*fprime(t+(1/5)*h,x+(28/625)*F1-(1/5)*F2+(546/625)
            *F3+(54/625)*F4-(378/625)*F5)
        x=x+(1/24)*F1+(5/48)*F4+(27/56)*F5+(125/336)*F6
        error.append(abs(f(t)-x))
        t=t+h
        tlist.append(t)
        xref.append(x)
```

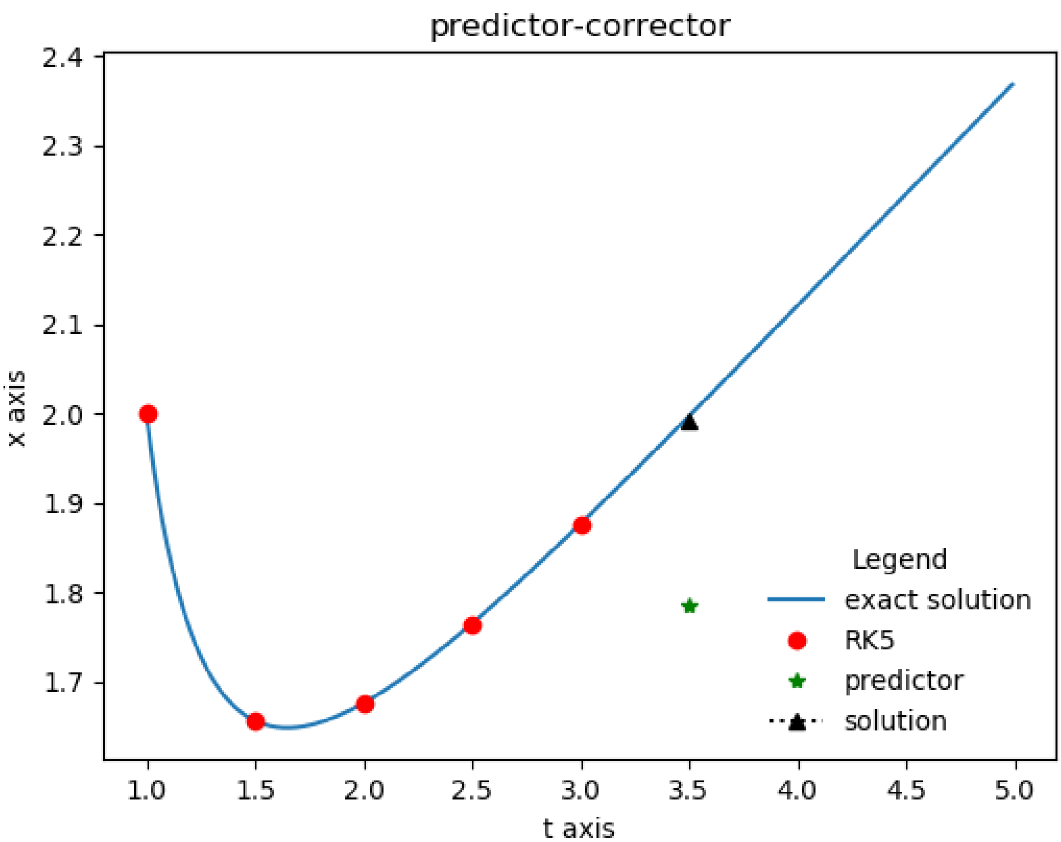
```
        i+=1
    return(xref,error,tlist)

def Runge_Kutta_2(f,fprime,t,x,h,M):
    xref=[x]
    error=[0]
    tlist=[t]
    for i in range(1,M+1):
        F1=h*fprime(t,x)
        F2=h*fprime(t+h,x+F1)
        x=x+(F1+F2)/2
        error.append(abs(f(t)-x))
        xref.append(x)
        i+=1
        t=t+h
        tlist.append(t)
    return(xref,error,tlist)

f=lambda t: t/((1/2)+m.log(t))
fprime= lambda t,x: (t**-2)*(t*x-x**2)
M1=4
t=1
x=2
h1=(3-1)/M1

X15,E15,T15=Runge_Kutta_5(f,fprime,t,x,h1,M1)
X12,E12,T12=Runge_Kutta_2(f,fprime,t,x,h1,M1)

M2=8
h2=(3-1)/M2

X25,E25,T25=Runge_Kutta_5(f,fprime,t,x,h2,M2)
X22,E22,T22=Runge_Kutta_2(f,fprime,t,x,h2,M2)

M3=100
h3=(3-1)/M3

X35,E35,T35=Runge_Kutta_5(f,fprime,t,x,h3,M3)
X32,E32,T32=Runge_Kutta_2(f,fprime,t,x,h3,M3)

M4=1000
h4=(3-1)/M4

X45,E45,T45=Runge_Kutta_5(f,fprime,t,x,h4,M4)
X42,E42,T42=Runge_Kutta_2(f,fprime,t,x,h4,M4)

real1=[f(i) for i in T15]
```

```
real2=[f(i) for i in T25]
real3=[f(i) for i in T35]
real4=[f(i) for i in T45]

print(E15)
#Creates two subplots and unpacks the output array immediately
fig, axrr = plt.subplots(2, 2, sharex='all' )
fig.suptitle("Coparison of RK5 to RK2 with step sizes")
fig.text(0.5, 0.04, 'Tvalues', ha='center')
fig.text(0.04, 0.5,'Solutions',va='center',rotation='vertical')
axrr[0,0].plot(T15, X15,'-',label='5th order')
axrr[0,0].plot(T12, X12,'-',label='2nd order')
axrr[0,0].plot(T12, real1,'x',label='real')
axrr[0,0].legend(loc='upper right')
axrr[0,0].set_title('h=0.5')
axrr[0,1].plot(T15, E15,'-',label='5th order')
axrr[0,1].plot(T12, E12,'-',label='2nd order')
axrr[0,1].legend(loc='upper right')
axrr[0,1].set_title('error at h=0.5')
axrr[1,0].plot(T35, X35,'-',label='5th order')
axrr[1,0].plot(T32, X32,'-',label='2nd order')
axrr[1,0].plot(T32, real3,'x',label='real')
axrr[1,0].legend(loc='upper right')
axrr[1,0].set_title('h=0.01')
axrr[1,1].plot(T35, E35,'-',label='5th order')
axrr[1,1].plot(T32, E32,'-',label='2th order')
axrr[1,1].legend(loc='upper right')
axrr[1,1].set_title('error at h=0.01')
plt.savefig("error comparison of 2nd and 5th order.png")
plt.show()
```

3. Implement the fifth-order predictor-corrector method using above fifth order Runge-Kutta together with Adams-Bashforth and Adams-Moulton formulas. Check your code using the above example.

Implementation 1:

Output:



Code (3.1):

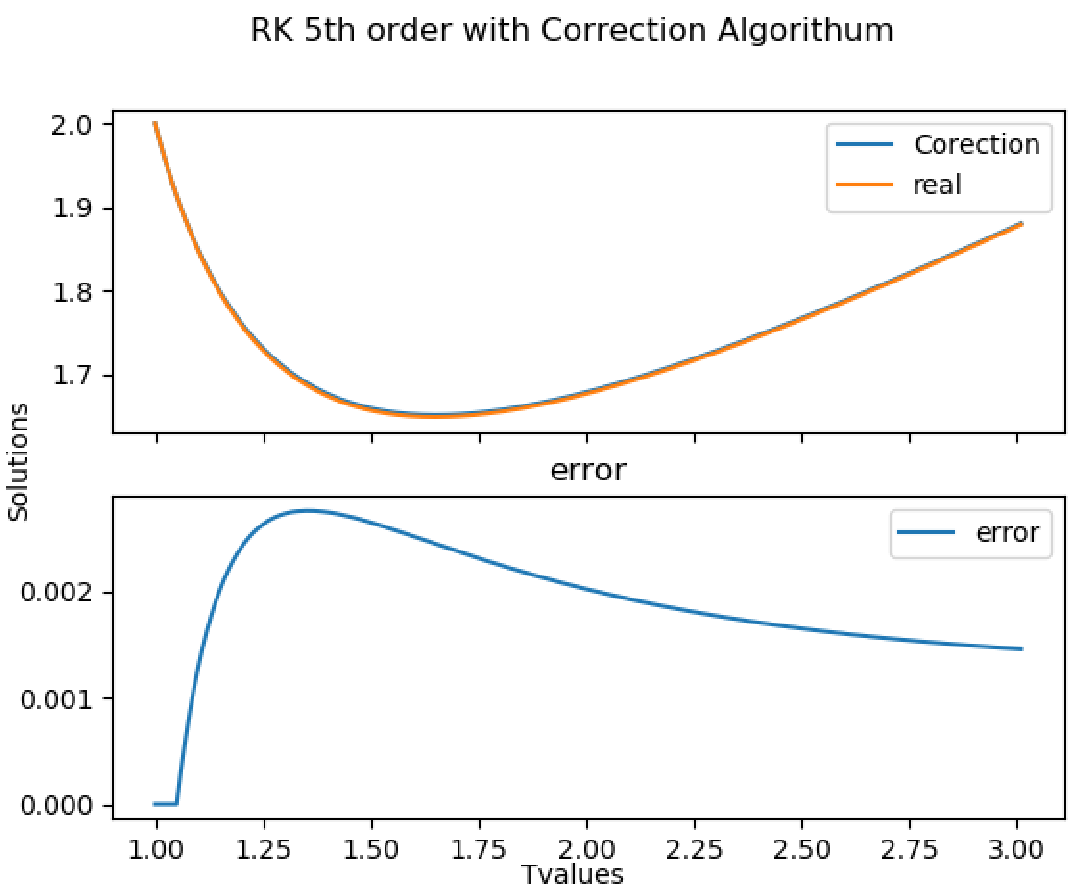
```
import numpy as np
import matplotlib.pyplot as plt
import math as m

# ODE to be solved x'(t) = f(t, x(t))
f = lambda t,x: (t**-2)*(t*x-x**2)

# exact solution to ODE
xt = lambda t: t/((1/2)+ np.log(t))

# initial conditions
t0 = 1
x0 = 2
```

```

# number of iterations
M = 4

# step size h
h = 0.5

# get initial data to start predictor-corrector scheme (RK5)
def Runge_Kutta_5(f,t,x,h,M):
    xref=[x]
    tlist=[t]
    for i in range(1,M+1):
        F1=h*f(t,x)
        F2=h*f(t+0.5*h,x+0.5*F1)
        F3=h*f(t+0.5*h,x+0.25*F1+0.25*F2)
        F4=h*f(t+h,x-F2+2*F3)
        F5=h*f(t+(2/3)*h,x+(7/27)*F1+(10/27)*F2+(1/27)*F4)
        F6=h*f(t+(1/5)*h,x+(28/625)*F1-(1/5)*F2+(546/625)*F3
            +(54/625)*F4-(378/625)*F5)
        x=x+(1/24)*F1+(5/48)*F4+(27/56)*F5+(125/336)*F6
        t=t+h
        tlist.append(t)
        xref.append(x)
        i+=1
    return(xref,tlist)

X,T=Runge_Kutta_5(f,t0,x0,h,M)

# adams-bashforth formula to calculate x_n+1* (predictor)
def adams_bash(f,X,T,h):
    x_nplus1_star = X[4] + (h/720)*(1901*f(T[4],X[4])
        - 2774*f(T[3],X[3]) + 2616*f(T[2],X[2])
        - 1274*f(T[1],X[1]) + 251*f(T[0],X[0]))
    return(x_nplus1_star)

predictor = adams_bash(f,X,T,h)

# adams-moulton formula to calculate x_n+1
def adams_moulton(predictor,f,X,T,h):
    x_nplus1 = X[4] + (h/720)*(1901*f(T[4]+h, predictor)
        - 2774*f(T[4],X[4]) + 2616*f(T[3],X[3])
        - 1274*f(T[2],X[2]) + 251*f(T[1],X[1]))
    return(x_nplus1)

x_nplus1 = adams_moulton(predictor,f,X,T,h)

t = np.arange(1,5,.01)
```

```
plt.title("predictor-corrector")
plt.xlabel("t axis")
plt.ylabel("x axis")
plt.plot(t,xt(t), label = "exact solution")
plt.plot(T,X, 'ro', label = "RK5")
plt.plot(T[len(T)-1] + h, predictor, 'g*',label = "predictor")
plt.plot(T[len(T)-1] + h, x_nplus1, '^k:',label = "solution")
plt.grid(False)
plt.legend(loc="lower right", title="Legend", frameon=False)
plt.show()
```

Implementation 2:

Output:



Code (3.2):

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

# ODE to be solved x'(t) = f(t, x(t))
f= lambda t,x: (t**-2)*(t*x-x**2)

# exact solution to ODE
xt=lambda t: t/((1/2)+m.log(t))

# initial conditions
t = 1
x = 2

# number of iterations
M=200

# step size h
h = 0.01

def fixers(f,ft,t,x,h,M):
    xref=[x]
    tlist=[t]
    error=[0]
    for i in range(0,M+1):
        if len(xref)<=5:
            F1=h*f(t,x)
            F2=h*f(t+0.5*h,x+0.5*F1)
            F3=h*f(t+0.5*h,x+0.25*F1+0.25*F2)
            F4=h*f(t+h,x-F2+2*F3)
            F5=h*f(t+(2/3)*h,x+(7/27)*F1+(10/27)*F2+(1/27)*F4)
            F6=h*f(t+(1/5)*h,x+(28/625)*F1-(1/5)*F2
                +(546/625)*F3+(54/625)*F4-(378/625)*F5)
            x=x+(1/24)*F1+(5/48)*F4+(27/56)*F5
                +(125/336)*F6
            t=t+h
            i+=1
            tlist.append(t)
            xref.append(x)
            error.append(abs(ft(t)-x))
        else:
            xprime = xref[i] + (h/720)*(1901
                *f(tlist[i],xref[i])
                - 2774*f(tlist[i-1],xref[i-1])
                + 2616*f(tlist[i-2],xref[i-2])
                + 273*f(tlist[i-3],xref[i-3])
```

```
                + 251*f(tlist[i-4],xref[i-4]))
            x=xref[i]+(h/720)*(251*f(tlist[i]+h,xprime)
                +646*f(tlist[i],xref[i])
                -264*f(tlist[i-1],xref[i-1])
                +106*f(tlist[i-2],xref[i-2])
                -19*f(tlist[i-3],
                    xref[i-3]))
            t+=h
            i+=1
            tlist.append(t)
            xref.append(x)
            error.append(abs(ft(t)-x))
    return(xref,tlist,error)


X,T,E=fixers(f,xt,t,x,h,M)
real=[xt(i) for i in T]
fig,(ax1,ax2) = plt.subplots(2, 1, sharex='all' )
fig.text(0.5, 0.04, 'Tvalues', ha='center')
fig.text(0.04,0.5,'Solutions',va='center',rotation='vertical')
fig.suptitle('RK 5th order with Correction Algorithum')
ax1.plot(T,X,label='Corection')
ax1.plot(T,real,label='real')
ax1.legend(loc='upper right')
ax2.plot(T,E,label='error')
ax2.legend(loc='upper right')
ax2.set_title('error')
plt.savefig("Corrector method.png")
plt.show()
```