h

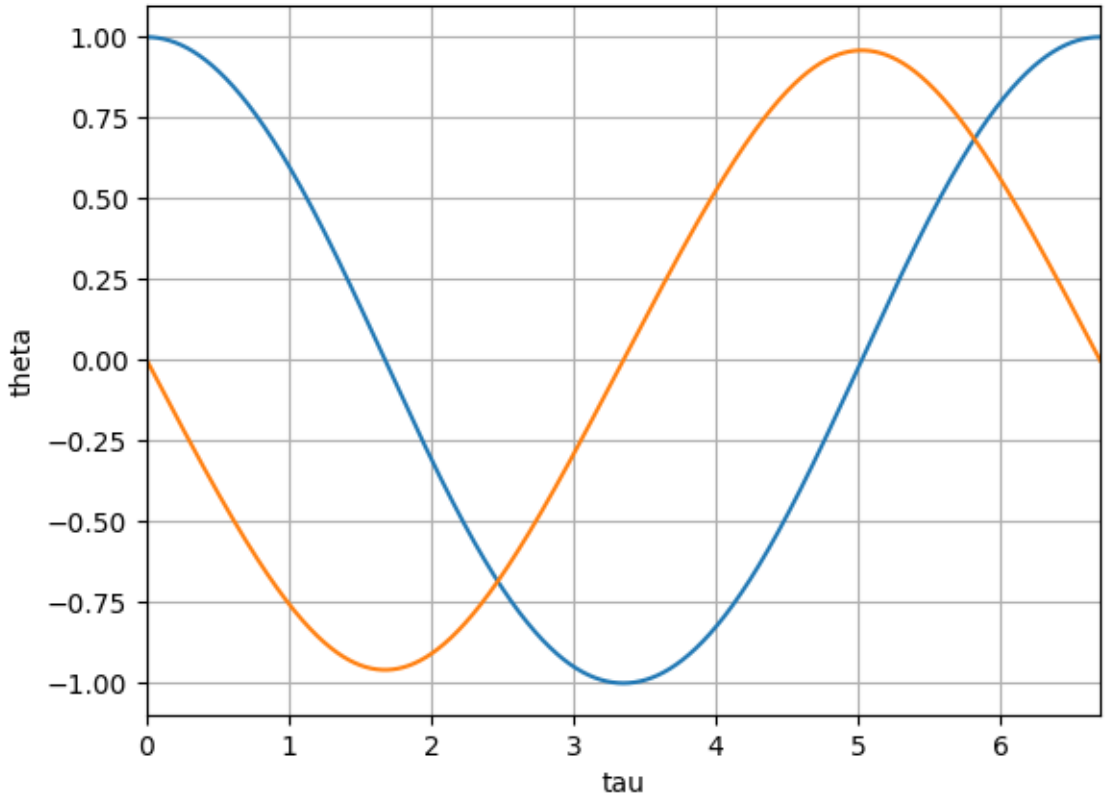

Figure 1: Q38, (Blue is θ, orange is $\frac{d\theta}{d\tau}$)

1 Question 36

-0.008333

2 Question 37

41.853968

3 Question 38

Period: 2.132676 π*sqrt(L/g)

h

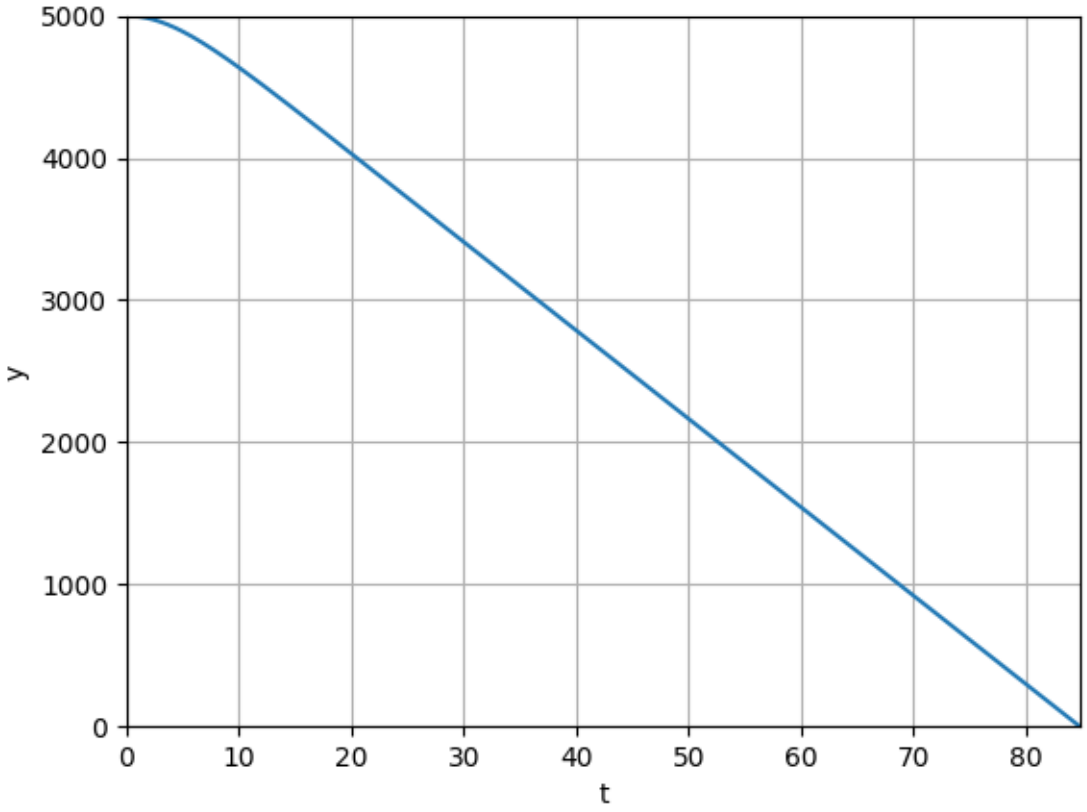

Figure 2: Q39

4 Question 39

Fall time: 84.8 s

h

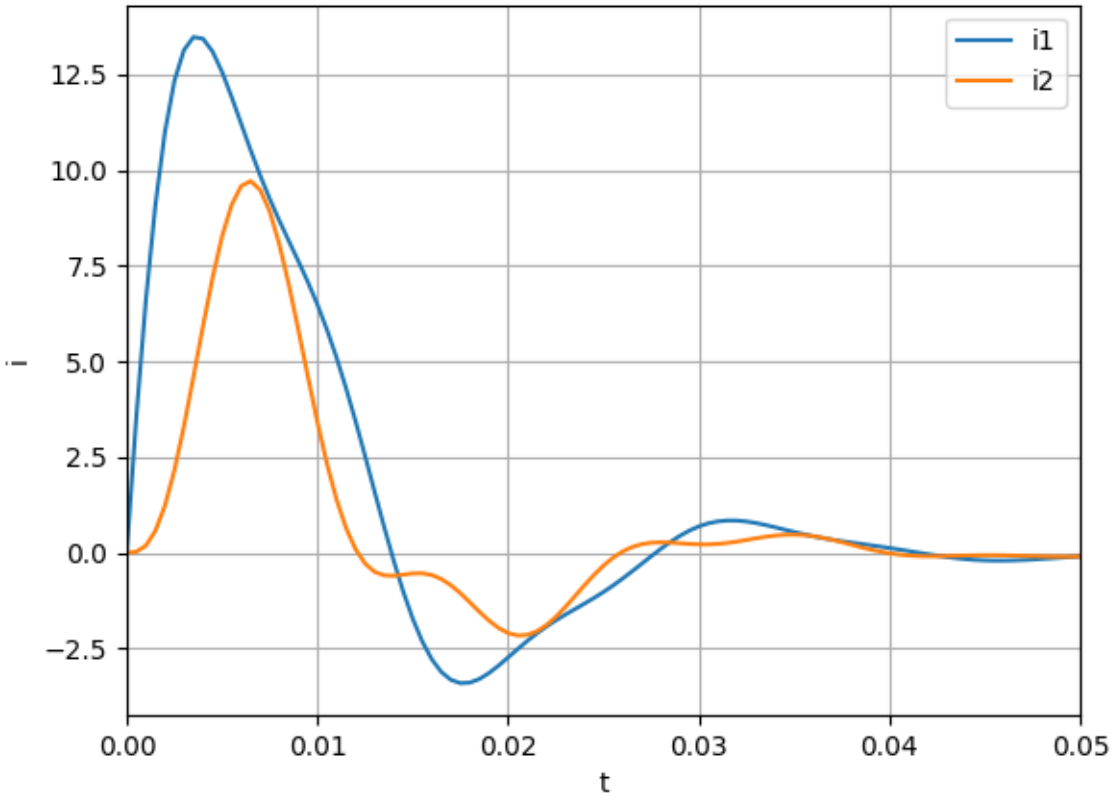

Figure 3: Q40

5 Question 40

t	i1 (A)	i2 (A)	t	i1 (A)	i2 (A)	t	i1 (A)	i2 (A)
0.	0.	0.	0.01	6.490725	3.417276	0.02	-2.738427	-2.099997
0.0005	3.536607	0.024685	0.0105	5.837555	2.332363	0.0205	-2.520513	-2.163295
0.001	6.585396	0.183408	0.011	5.11096	1.40035	0.021	-2.309815	-2.149322
0.0015	9.08104	0.569832	0.0115	4.310259	0.65078	0.0215	-2.113353	-2.058568
0.002	10.995325	1.228839	0.012	3.444894	0.093059	0.022	-1.933884	-1.898054
0.0025	12.336338	2.156731	0.0125	2.533425	-0.281918	0.0225	-1.770496	-1.680243
0.003	13.14442	3.306639	0.013	1.601541	-0.498496	0.023	-1.619492	-1.421492
0.0035	13.485553	4.598047	0.0135	0.679394	-0.591835	0.0235	-1.47545	-1.140233
0.004	13.443082	5.929022	0.014	-0.201401	-0.602905	0.024	-1.332329	-0.855117
0.0045	13.108728	7.189587	0.0145	-1.010762	-0.573461	0.0245	-1.184499	-0.583286
0.005	12.573854	8.274739	0.015	-1.722803	-0.541507	0.025	-1.027591	-0.338963
0.0055	11.921798	9.095765	0.0155	-2.317921	-0.537677	0.0255	-0.859103	-0.132448
0.006	11.22192	9.588857	0.016	-2.78415	-0.58279	0.026	-0.678714	0.030394
0.0065	10.525758	9.72037	0.0165	-3.117685	-0.686723	0.0265	-0.488314	0.148172
0.007	9.865451	9.488492	0.017	-3.322615	-0.848565	0.027	-0.291761	0.223557
0.0075	9.254324	8.9215	0.0175	-3.409945	-1.057919	0.0275	-0.094435	0.262525
0.008	8.68933	8.073088	0.018	-3.396071	-1.297084	0.028	0.09736	0.273345
0.0085	8.154892	7.015518	0.0185	-3.300914	-1.54381	0.0285	0.27706	0.265463
0.009	7.627585	5.831513	0.019	-3.145911	-1.77427	0.029	0.438492	0.248382
0.0095	7.081067	4.605807	0.0195	-2.952096	-1.965927	0.0295	0.576452	0.230671

t	i1 (A)	i2 (A)	t	i1 (A)	i2 (A)
0.03	0.687147	0.219185	0.04	0.124711	-0.013087
0.0305	0.768475	0.218543	0.0405	0.085708	-0.04744
0.031	0.820121	0.230907	0.041	0.045338	-0.071318
0.0315	0.843474	0.256043	0.0415	0.004388	-0.085465
0.032	0.841401	0.291631	0.042	-0.036029	-0.091336
0.0325	0.81789	0.333765	0.0425	-0.074609	-0.090871
0.033	0.777636	0.377573	0.043	-0.109999	-0.086245
0.0335	0.725591	0.417876	0.0435	-0.140943	-0.079617
0.034	0.666544	0.449822	0.044	-0.166405	-0.072918
0.0345	0.60476	0.469421	0.0445	-0.185666	-0.067675
0.035	0.543704	0.473936	0.045	-0.198376	-0.064911
0.0355	0.485879	0.462116	0.0455	-0.204575	-0.065095
0.036	0.432776	0.434239	0.046	-0.204672	-0.068165
0.0365	0.384923	0.392	0.0465	-0.199386	-0.073596
0.037	0.342027	0.338261	0.047	-0.189671	-0.080524
0.0375	0.30318	0.276694	0.0475	-0.176616	-0.087876
0.038	0.267097	0.211382	0.048	-0.161354	-0.094523
0.0385	0.232365	0.146405	0.0485	-0.144962	-0.099415
0.039	0.197671	0.085469	0.049	-0.128388	-0.1017
0.0395	0.16199	0.031595	0.0495	-0.112391	-0.100809
			0.05	-0.097507	-0.096499]

```
import numpy as np
from math import *
np.set_printoptions(suppress=True, precision=6)

#P36: Page 245 Problem 13
#Hint: See the predefined functions in
#"module triangleQuad" provided in the book.
def triangleQuad(f,xc,yc):
  #area coordinates
    alpha = np.array([[1/3,1/3,1/3], \
                                        [.2, .2, .6], \
                                        [.6,.2,.2],\
                                        [.2,.6,.2]])
    W = np.array([-27,25,25,25])/48
    #Cubic weights
    x = np.dot(alpha,xc)#Convert to new coord basis
    y = np.dot(alpha,yc)
    #Find det for area coords?
    A = ( xc[1]*yc[2] - xc[2]*yc[1] \
            - xc[0]*yc[2] + xc[2]*yc[0] \
            + xc[0]*yc[1] - xc[1]*yc[0])/2.0
    sum = 0.0
    for i in range(4):
        sum += W[i]*f(x[i],y[i])
    return A*sum

def P36():
    def f(x,y):
        return (1-x)*(y-x)*y
    xc = np.array([0,1,1.])
    yc = np.array([0,0,1.])
    print("+-----+")
    print("| P36 |")
    print("+-----+")
    print(triangleQuad(f,xc,yc))

def triTest():
    def f(x,y):
        return (x**2 + y**2)/2.0 \
                    -(x**3 - 3.0*x*y**2)/6.0 \
                    -2.0/3.0
    xCorner = np.array([ -1.0, 2.0, -1.0])
    yCorner = np.array([ -sqrt(3.0), 0.0, sqrt(3.0)])
    print("Integral =",triangleQuad(f,xCorner,yCorner))
```

```
    #-1.55884572681

#P37: Page 245 Problem 11
#Hint: See the predefined functions in "module gaussQuad2"
#provided in the book.
from hw5 import gaussNodes

def gaussQuad2(f,x,y,m):
    def jac(x,y,s,t):#Find jacobian for change of basis
        J = np.zeros((2,2,))
        mt, pt, ms, ps = 1.0-t, 1.0+t, 1.0-s, 1.0+s
        J[0,0] = mt*(-x[0]+x[1]) + pt*(x[2]-x[3])
        J[0,1] = mt*(-y[0]+y[1]) + pt*(y[2]-y[3])
        J[1,0] = ms*(-x[0]+x[3]) + ps*(-x[1]+x[2])
        J[1,1] = ms*(-y[0]+y[3]) + ps*(-y[1]+y[2])
        #|J|
        return (J[0,0]*J[1,1]-J[0,1]*J[1,0])/16.0
    def map(x,y,s,t):
        N=np.zeros(4)
        #map coords to quadrilateral
        mt, pt, ms, ps = 1.0-t, 1.0+t, 1.0-s, 1.0+s
        N[0] = ms*mt/4.0
        N[1] = ps*mt/4.0
        N[2] = ps*pt/4.0
        N[3] = ms*pt/4.0
        xCoord = np.dot(N,x)
        yCoord = np.dot(N,y)
        return xCoord,yCoord
    s,A = gaussNodes(m)#Find nodes
    sum = 0.0
    for i in range(m):
        for j in range(m):
            xCoord,yCoord = map(x,y,s[i],s[j])
            #Do summation with adjusted coords
            sum += A[i]*A[j]*jac(x,y,s[i],s[j])*f(xCoord,yCoord)
    return sum

def P37():
    def f(x,y):
        return x*y*(2-x**2)*(2-x*y)
    x = np.array([-3,1,3,-1.])
    y = np.array([-2,-2,2,2.])
    print("+-----+")
    print("| P37 |")
    print("+-----+")
```

```
    print(gaussQuad2(f,x,y,20))

def gQtest():
    def f(x,y):
        return (x**2 + y**2)/2.0 \
                        - (x**3 - 3.0*x*y**2)/6.0 \
                        - 2.0/3.0
    x = np.array([-1.0,2.0,2.0,-1.0])
    y = np.array([-sqrt(3.0),0.0,0.0,sqrt(3.0)])
    m = 3#eval(input("Integration order ==> "))
    print("Integral =", gaussQuad2(f,x,y,m))#-1.55884572681

#P38: Page 263 Problem 7
#Hint: See the predefined functions in "module run_kut4"
#provided in the book. Also, look at Example 7.4 to have an idea.
def integrate(F,x,y,xStop,h):
    def run_kut4(F,x,y,h):
      #RK4 Equations
        K0 = h*F(x,y)
        K1 = h*F(x + h/2.0, y + K0/2.0)
        K2 = h*F(x + h/2.0, y + K1/2.0)
        K3 = h*F(x + h, y + K2)
        return (K0 + 2.0*K1 + 2.0*K2 + K3)/6.0
    X, Y = np.array(x),np.array(y)
    while x < xStop: #Iterate through range (dont overstep)
        h = min(h,xStop-x)
        y = y + run_kut4(F,x,y,h)#find new y
        x = x + h#find new x
        X = np.append(X, x)
        Y = np.vstack((Y, y))
    return X,Y

import matplotlib.pyplot as plt

def P38():
    print("+-----+")
    print("| P38 |")
    print("+-----+")
    def F(x,y):#tau, theta
        F = np.zeros(2)
        F[0] = y[1]
        F[1] = -1*sin(y[0])#-0.1*y[1]-x
        return F
    x, xStop = 0.0, 3*pi#Range for X
    y = np.array([1.0,0.0])#Initial Conditions
```

```
    h = 0.001#Step size
    X,Y = integrate(F,x,y,xStop,h)
    for i in range(10,len(X)):
        if Y[i-1,1] >= 0 and Y[i,1]<=0:#Check to see if completed
            #one full period
            T = X[i]
            print("Period:",round(T/pi,6),"pi*sqrt(L/g)")
            break
    plt.plot(X,Y[:,0],X,Y[:,1])#plot
    plt.grid(True)
    plt.xlabel('tau'); plt.ylabel('theta')
    plt.xlim(x,T)
    plt.savefig("figures/hw6p38.png")#show()

#P39: Page 264 Problem 8
#Hint: See the predefined functions in "module run_kut4"
# provided in the book.
def P39():
    print("+-----+")
    print("| P39 |")
    print("+-----+")
    def F(x,y):#t,y
        F = np.zeros(2)
        g,CD,m=-9.80665,0.2028,80
        F[0] = y[1]
        F[1] = g+CD/m*y[1]*y[1]
        return F
    x, xStop = 0.0, 100# 0.5
    y = np.array([5000,0.0])#[1, 0])
    h = 0.1
    X,Y = integrate(F,x,y,xStop,h)
    for i in range(10,len(X)):
        if Y[i-1,0] >= 0 and Y[i,0]<=0:
            T = X[i]
            print("Fall time:",round(T,6),"s")
            break
    plt.clf()
    plt.plot(X,Y[:,0])
    plt.grid(True)
    plt.xlabel('t'); plt.ylabel('y')
    plt.xlim(x,T)
    plt.ylim(0,5000)
    plt.savefig("figures/hw6p39.png")#show()

#P40: Page 268 Problem 22
```

```
#You do not need to submit the plot.
#The program should contain a function which will return
#two arrays containing the values of time and currents
#respectively, used for the plot. Hint: See the predefined
#functions in "module run_kut4" provided in the book.
def P40():
    print("+-----+")
    print("| P40 |")
    print("+-----+")
    def F(x,y):#y=[q1,i1,q2,i2]
        F = np.zeros(4)
        E,R,L,C=9,0.25,1.2*(10**-3), 5*(10**-3)
        F[0] = y[1]#q1'=i1
        F[1] = (E-R*y[1]-(y[0]-y[2])/C)/L#q1''=i1'
        F[2] = y[3]#q2'=i2
        F[3] = (-R*y[3]-(y[2]-y[0])/C-y[2]/C)/L#q2''=i2'
        return F
    x, xStop = 0.0, 0.05# 0.5
    y = np.array([0.0,0.0,0.0,0.0])#[1, 0])
    h = 0.0005
    X,Y = integrate(F,x,y,xStop,h)

    plt.clf()
    plt.plot(X,Y[:,1],X,Y[:,3])
    plt.grid(True)
    plt.xlabel('t'); plt.ylabel('i')
    plt.xlim(x,xStop)
    plt.legend(["i1","i2"])

    plt.savefig("figures/hw6p40.png")#show()
    print(np.hstack((X.reshape(len(X),1),Y[:,(1,3)])))
    print("s, A, A")

if __name__ == "__main__":
    P36()
    P37()
    P38()
    P39()
    P40()
```