sec 5.3 Q9a

```
import math

def func1 ( t, y ):
    return (2/t) * y + t*t*math.exp(t)
def func2 ( t, y ):
    return (2/t*t) * y + 4*t*math.exp(t) + t*t*math.exp(t)

def second_order(t0, y, h, t):
    while t0 < t:
        temp = y
        y = y + h * func1(t0, y)+(h*h/2)*func2(t0,y)
        t0 = t0 + h
    print("Approximate solution at t = ", t, " is ", "%.6f" % y)
    return y



def exactfunc(t):
  return t*t*(math.exp(t)-math.exp(1))


t0 = 1
y0 = 0
h = 0.1
t = 1.1
approx = second_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.1  is  0.339785
The exact value at t =  1.1  is  0.345920
0.006134647982359187
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.2
approx = second_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.2  is  0.852733
The exact value at t =  1.2  is  0.866643
0.01390937658245417
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.3
approx = second_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.3  is  1.585067
The exact value at t =  1.3  is  1.607215
0.022147909352181916
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.4
approx = second_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.4  is  2.591293
The exact value at t =  1.4  is  2.620360
0.029066816179772026
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.5
approx = second_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at x =  1.5  is  3.935496
The exact value at t =  1.5  is  3.967666
0.032170203136305364
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.5
approx = second_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.5  is  3.935496
The exact value at t =  1.5  is  3.967666
```

```
0.032170203136305364
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.6
approx = second_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.6  is  5.692834
The exact value at t =  1.6  is  5.720962
0.028127946470894116
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.7
approx = second_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.7  is  7.951238
The exact value at t =  1.7  is  7.963873
0.012635212279533903
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.8
approx = second_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.8  is  10.813374
The exact value at t =  1.8  is  10.793625
0.01974933101766574
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.9
approx = second_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.9  is  14.398871
The exact value at t =  1.9  is  14.323082
0.07578957070101389
```

```
t0 = 1
y0 = 0
h = 0.1
t = 2.0
approx = second_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  2.0  is  18.846883
The exact value at t =  2.0  is  18.683097
0.16378565572325243
```

sec 5.3 Q9b

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

# xi -> corresponds to the new data point
# n -> represents the number of known data points
def interpolate(f: list, xi: int, n: int) -> float:
    result = 0.0
    for i in range(n):
        term = f[i].y
        for j in range(n):
            if j != i:
                term = term * (xi - f[j].x) / (f[i].x - f[j].x)
        result += term
    return result


#(i)
# x is the same as t
#(t0, y0) = (1,0)
#(t1, y1) = (1.1, 0.339785)
d1 = [Point(1,0), Point(1.1, 0.339785)]
approx = interpolate(d1, 1.04, 2)
print(approx)
exact = exactfunc(1.04)
error = abs(exact-approx)
error
```

```
0.135914
0.01592650293865608
```

```
#(ii)
d1 = [Point(1.5,3.935496), Point(1.6, 5.692834)]
approx = interpolate(d1, 1.55, 2)
print(approx)
exact = exactfunc(1.55)
error = abs(exact-approx)
error
```

```
4.814165
0.025529979198597452
```

```
#(iii)
d1 = [Point(1.9,14.398871), Point(2, 18.846883)]
approx = interpolate(d1, 1.97, 2)
print(approx)
exact = exactfunc(1.97)
error = abs(exact-approx)
error
```

```
17.512479399999997
0.23318096444233305
```

sec 5.3 Q9c

```
def func3(t,y):
  return 6*math.exp(t)+6*t*math.exp(t)+t*t*math.exp(t)

def func4(t,y):
  return 12*math.exp(t)+8*t*math.exp(t)+t*t*math.exp(t)

def forth_order(t0, y, h, t):
    while t0 < t:
        temp = y
        y = y + h * func1(t0, y)+(h*h/2)*func2(t0,y)+(h*h*h/6)*func3(t0,y)+(h*h*h*h
        t0 = t0 + h
    print("Approximate solution at t = ", t, " is ", "%.6f" % y)
    return y


t0 = 1
y0 = 0
h = 0.1
t = 1.1
approx = forth_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.1  is  0.345913
The exact value at t =  1.1  is  0.345920
7.18769404106645e-06
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.2
approx = forth_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.2  is  0.867226
The exact value at t =  1.2  is  0.866643
0.0005835378550950177
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.3
approx = forth_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.3  is  1.610540
The exact value at t =  1.3  is  1.607215
0.0033252364853941785
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.4
approx = forth_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.4  is  2.630781
The exact value at t =  1.4  is  2.620360
0.010421213179569566
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.5
approx = forth_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.5  is  3.992502
```

```
The exact value at t =  1.5  is  3.967666
0.024836152274084533
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.6
approx = forth_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.6  is  5.771386
The exact value at t =  1.6  is  5.720962
0.05042436070094247
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.7
approx = forth_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.7  is  8.055951
The exact value at t =  1.7  is  7.963873
0.09207748019471751
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.8
approx = forth_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.8  is  10.949519
The exact value at t =  1.8  is  10.793625
0.1558943698307207
```

```
t0 = 1
y0 = 0
h = 0.1
t = 1.9
approx = forth_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  1.9  is  14.572458
The exact value at t =  1.9  is  14.323082
0.24937624042997975
```

```
t0 = 1
y0 = 0
h = 0.1
t = 2.0
approx = forth_order(t0, y0, h, t)
exact = exactfunc(t)
print("The exact value at t = " ,t, " is ", "%.6f"% exactfunc(t))
error = abs(exact-approx)
error
```

```
Approximate solution at t =  2.0  is  19.064748
The exact value at t =  2.0  is  18.683097
0.3816505485780155
```

Sec5.4 Q3(b)

```
def exactfunc(t):
  return t*math.tan(math.log(t))
def func(t,y):
  return 1+y/t+(y/t)*(y/t)

import numpy as np
def modified_euler_method( f, a, b, N, IV ):
    h = (b-a)/float(N)
    t = np.arange( a, b+h, h )
    y = np.zeros((N+1,))
    exact = np.zeros((N+1,))
    error = np.zeros((N+1,))
    t[0] = a
    y[0] = IV
    for i in range(1,N+1):
        f1 = f( t[i-1], y[i-1] )
        f2 = f( t[i], y[i-1] + h * f1 )
        y[i] = y[i-1] + h * ( f1 + f2 ) / 2.0
        exact[i] = exactfunc(t[i])
        error[i] = abs(exactfunc(t[i])-y[i])
    return t, y, exact, error
modified_euler_method( func, 1.0, 3.0, 10, 0 )
# code edited from: http://connor-johnson.com/2014/02/21/numerical-solutions-to-ode
```

```
(array([1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4, 2.6, 2.8, 3. ]),
 array([0.        , 0.21944444, 0.48504947, 0.80401161, 1.18485597,
        1.63842289, 2.17887721, 2.82506509, 3.60252472, 4.54661355,
        5.70756991]),
 array([0.        , 0.22124277, 0.48968166, 0.81275274, 1.19943864,
        1.66128176, 2.21350181, 2.87655142, 3.67847533, 4.65866506,
        5.87409998]),
```

```
 array([0.        , 0.00179833, 0.00463219, 0.00874113, 0.01458267,
        0.02285886, 0.0346246 , 0.05148633, 0.07595061, 0.11205151,
        0.16653007]))
```

sec 5.4 Q3(d)

```
def func(t,y):
  return -5*y + 5*t*t + 2*t
def exactfunc(t):
  return t*t + (1/3) * math.exp(-5*t)
modified_euler_method( func, 0.0, 1.0, 10, 1/3 )
```

```
(array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 array([0.33333333, 0.22083333, 0.17427083, 0.17641927, 0.21651204,
        0.28782003, 0.38613752, 0.50883595, 0.65427247, 0.82142029,
        1.00963768]),
 array([0.        , 0.21217689, 0.16262648, 0.16437672, 0.20511176,
        0.27736167, 0.37659569, 0.50006579, 0.64610521, 0.813703  ,
        1.00224598]),
 array([0.        , 0.00865645, 0.01164435, 0.01204255, 0.01140028,
        0.01045836, 0.00954183, 0.00877015, 0.00816725, 0.00771729,
        0.0073917 ]))
```

sec 5.4 Q7(b)

```
def exactfunc(t):
  return t*math.tan(math.log(t))
def func(t,y):
  return 1+y/t+(y/t)*(y/t)

def midpoint_method( f, a, b, N, IV ):
    h = (b-a)/float(N)                  # determine step-size
    t = np.arange( a, b+h, h )
    y = np.zeros((N+1,))
    exact = np.zeros((N+1,))
    error = np.zeros((N+1,))
    t[0] = a
    y[0] = IV
    for i in range(1,N+1):              # apply Midpoint Method
        y[i] = y[i-1] + h*f(t[i-1]+h/2.0, y[i-1]+h*f(t[i-1], y[i-1])/ 2.0 )
        exact[i] = exactfunc(t[i])
        error[i] = abs(exactfunc(t[i])-y[i])
    return t, y, exact, error
midpoint_method(func,1.0,3.0,10,0)
# code edited from: http://connor-johnson.com/2014/02/21/numerical-solutions-to-ode
```

```
(array([1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4, 2.6, 2.8, 3. ]),
 array([0.        , 0.21983471, 0.48617705, 0.80618489, 1.18843926,
        1.64388892, 2.18686089, 2.83643571, 3.61849255, 4.56889438,
        5.73864747]),
 array([0.        , 0.22124277, 0.48968166, 0.81275274, 1.19943864,
```

```
          1.66128176, 2.21350181, 2.87655142, 3.67847533, 4.65866506,
          5.87409998]),
   array([0.        , 0.00140806, 0.00350462, 0.00656785, 0.01099938,
          0.01739283, 0.02664092, 0.04011571, 0.05998278, 0.08977067,
          0.13545251]))
```

sec 5.4 Q7(d)

```
def func(t,y):
  return -5*y + 5*t*t + 2*t
def exactfunc(t):
  return t*t + (1/3) * math.exp(-5*t)
midpoint_method(func, 0.0, 1.0, 10, 1/3 )
```

```
  (array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
   array([0.33333333, 0.21958333, 0.17223958, 0.17389974, 0.21368734,
          0.28480459, 0.38300287, 0.50562679, 0.65101674, 0.81813547,
          1.00633467]),
   array([0.        , 0.21217689, 0.16262648, 0.16437672, 0.20511176,
          0.27736167, 0.37659569, 0.50006579, 0.64610521, 0.813703  ,
          1.00224598]),
   array([0.        , 0.00740645, 0.0096131 , 0.00952302, 0.00857558,
          0.00744292, 0.00640718, 0.005561  , 0.00491153, 0.00443247,
          0.00408868]))
```

sec5.4 Q15(b)

```
def exactfunc(t):
  return t*math.tan(math.log(t))
def func(t,y):
  return 1+y/t+(y/t)*(y/t)

def runge_kutta_4_method( f, a, b, N, IV ):
    h = (b-a)/float(N)                  # determine step-size
    t = np.arange( a, b+h, h )
    y = np.zeros((N+1,))
    exact = np.zeros((N+1,))
    error = np.zeros((N+1,))
    t[0] = a
    y[0] = IV
    for i in range(1,N+1):              # apply Fourth Order Runge-Kutta Method
        k1 = h * f( t[i-1], y[i-1] )
        k2 = h * f( t[i-1] + h / 2.0, y[i-1] + k1 / 2.0 )
        k3 = h * f( t[i-1] + h / 2.0, y[i-1] + k2 / 2.0 )
        k4 = h * f( t[i], y[i-1] + k3 )
        y[i] = y[i-1] + ( k1 + 2.0 * k2 + 2.0 * k3 + k4 ) / 6.0
        exact[i] = exactfunc(t[i])
        error[i] = abs(exactfunc(t[i])-y[i])
    return t, y, exact, error
runge_kutta_4_method(func,1.0,3.0,10,0)
# code edited from: http://connor-johnson.com/2014/02/21/numerical-solutions-to-ode
```

```
# Code edited from: http://connor-johnson.com/2014/02/18/numerical-solutions-to-ode
```

```
(array([1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4, 2.6, 2.8, 3. ]),
 array([0.        , 0.22124571, 0.48968417, 0.81275216, 1.19943202,
        1.66126512, 2.21346932, 2.87649411, 3.678379  , 4.65850628,
        5.87383857]),
 array([0.        , 0.22124277, 0.48968166, 0.81275274, 1.19943864,
        1.66128176, 2.21350181, 2.87655142, 3.67847533, 4.65866506,
        5.87409998]),
 array([0.00000000e+00, 2.93453118e-06, 2.50269242e-06, 5.78537532e-07,
        6.61868670e-06, 1.66402729e-05, 3.24969043e-05, 5.73054533e-05,
        9.63347526e-05, 1.58774557e-04, 2.61408351e-04]))
```

sec 5.4 Q15(d)

```
def func(t,y):
  return -5*y + 5*t*t + 2*t
def exactfunc(t):
  return t*t + (1/3) * math.exp(-5*t)
runge_kutta_4_method(func, 0.0, 1.0, 10, 1/3 )
```

```
(array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 array([0.33333333, 0.21228299, 0.16276546, 0.16451654, 0.20524051,
        0.27747666, 0.37669808, 0.50015795, 0.64618959, 0.8137817 ,
        1.00232067]),
 array([0.        , 0.21217689, 0.16262648, 0.16437672, 0.20511176,
        0.27736167, 0.37659569, 0.50006579, 0.64610521, 0.813703  ,
        1.00224598]),
 array([0.00000000e+00, 1.06099540e-04, 1.38977328e-04, 1.39820702e-04,
        1.28744116e-04, 1.14994496e-04, 1.02388524e-04, 9.21538833e-05,
        8.43754935e-05, 7.87045663e-05, 7.46866646e-05]))
```

sec 5.4 Q17(b)

```
def exactfunc(t):
  return t*math.tan(math.log(t))
def func(t,y):
  return 1+y/t+(y/t)*(y/t)
#(t0, y0) = (2.0,1.63842289)
#(t1, y1) = (2.2,2.17887721)
d1 = [Point(2.0,1.63842289), Point(2.2,2.17887721)]
approx = interpolate(d1, 2.1, 2)
print(approx)
exact = exactfunc(2.1)
error = abs(exact-approx)
error
```

```
1.9086500499999999
0.016311601019705302
```

```
d1 = [Point(2.6,3.60252472), Point(2.8,4.54661355)]
approx = interpolate(d1, 2.75, 2)
```

```
approx = interpolate(d1, 2.75, 2)
print(approx)
exact = exactfunc(2.75)
error = abs(exact-approx)
error
```

```
4.3105913425
0.08357840069680122
```

sec 5.4 Q17(d)

```
def func(t,y):
  return -5*y + 5*t*t + 2*t
def exactfunc(t):
  return t*t + (1/3) * math.exp(-5*t)

d1 = [Point(0.5,0.28782003), Point(0.6,0.38613752)]
approx = interpolate(d1, 0.54, 2)
print(approx)
exact = exactfunc(0.54)
error = abs(exact-approx)
error
```

```
0.32714702600000006
0.013145188420083442
```

```
d1 = [Point(0.9,0.82142029), Point(1.0,1.00963768)]
approx = interpolate(d1, 0.94, 2)
print(approx)
exact = exactfunc(0.94)
error = abs(exact-approx)
error
```

```
0.8967072459999998
0.010075486966101277
```