

```

"""
Homework 7 Problem 1
Develop an algorithm which, for a given function f(x), interval bounds a and b with a < b,
and a prescribed number of subintervals n, applies the multiple application trapezoidal
rule to approximate integrating f(x) dx from a to b.

@author: Jacob Needham
"""
import math

def function(x):
    y = x**0.1 * (1.2 - x)*(1 - math.exp(20*(x-1)))
    return y

def multiTrap(a,b,n):
    h = (b-a)/n
    x = a
    total = function(x)
    for i in range(1,n):
        x += h
        total += 2*function(x)
    total += function(b)
    I = (b-a)* total/(2*n)
    return I

def main():
    print("The value found for 600 iterations:",multiTrap(0,1,600))

main()

```

```
1 """
2 Homework 7 Problem 1
3 Develop an algorithm which, for a given function f(x), interval bounds a and b with a < b,
4 and a prescribed number of subintervals n, applies the multiple application trapezoidal
5 rule to approximate integrating f(x) dx from a to b.
6
7 @author: Jacob Needham
8 """
9 import math
10
11 def function(x):
12     y = x**0.1 * (1.2 - x)*(1 - math.exp(20*(x-1)))
13     return y
14
15 def multiTrap(a,b,n):
16     h = (b-a)/n
17     x = a
18     total = function(x)
19     for i in range(1,n):
20         x += h
21         total += 2*function(x)
22     total += function(b)
23     I =(b-a)* total/(2*n)
24     return I
25
26 def main():
27     print("The value found for 600 iterations:",multiTrap(0,1,600))
28
29 main()
```

In [82]: runfile('C:/Users/Needh/Documents/Spring 2019/Numerical Methods/HW 7/Problem 1.py', wdir='C:/Users/Needh/Documents/Spring 2019/Numerical Methods/HW 7')
The value found for 600 iterations: 0.6018571071751161

In [83]:

```
"""
```

Homework 7 Problem 2

Develop an algorithm which, for a given function $f(x)$, interval bounds a and b with $a < b$, and a prescribed number of subintervals n , approximates integrating $f(x) dx$ from a to b according to the following procedure:

- (a) If $n = 1$, it applies the trapezoidal rule.
- (b) If n is even, it applies the multiple application Simpson's 1/3 rule.
- (c) If $n \geq 3$ and n is odd, it applies the multiple application Simpson's 1/3 rule on the first $n - 3$ subintervals, and applies the Simpson's 3/8 rule on the last three subintervals.

@author: Jacob Needham

```
"""
```

```
import math
```

```
def function(x):
```

```
    y = x**0.1 * (1.2 - x)*(1 - math.exp(20*(x-1)))
```

```
    return y
```

```
def multiTrap(n,a,b):
```

```
    h = (b-a)/n
```

```
    x = a
```

```
    total = function(x)
```

```
    for i in range(1,n):
```

```
        x += h
```

```
        total += 2*function(x)
```

```
    total += function(b)
```

```
    I = (b-a)* total/(2*n)
```

```
    return I
```

```
def simp13(a,b,n,h):
```

```
    x=a
```

```
    total = function(x)
```

```
    #h=(b-a)/n
```

```
    for i in range(2,n-1,2):
```

```
        x += h
```

```
        total = total + 4*function(x)
```

```
        x += h
```

```
        total = total + 2*function(x)
```

```
    x += h
```

```
    total = total + 4*function(x)
```

```
    x += h
```

```
    total = total + function(x)
```

```
    I = (b-a)*total/(3*n)
```

```
    return I
```

```
def simp38(a,b,n,h):
```

```
    x=a
```

```
    dx = (b-a)/n
```

```
    total = function(x)
```

```
    x += dx
```

```
    total += 3*function(x)
```

```
    x += dx
```

```
    total += 3*function(x)
```

```
    x += dx
```

```
    total += function(x)
```

```
    ans = (b-a)*total/8
```

```

    return ans

def printF(area):
    print("The approximate area under the curve is", area)

def main():
    a = 0
    b = 1
    n = 700
    h = (b-a)/n

    if n == 1:
        area = multiTrap(n,a,b)
        printF(area)

    if n%2 == 0:
        area = simp13(a,b,n,h)
        printF(area)

    if n%2 != 0:
        areaA = simp13(a,b-2*h,n,h)
        areaB = simp38(b-2*h,b,n,h)
        printF(areaA+areaB)

main()

```

```
1 """
2 Homework 7 Problem 2
3 Develop an algorithm which, for a given function f(x), interval bounds a and b with
4 a < b, and a prescribed number of subintervals n, approximates integrating f(x) dx from
5 a to b according to the following procedure:
6 (a) If n = 1, it applies the trapezoidal rule.
7 (b) If n is even, it applies the multiple application Simpson's 1/3 rule.
8 (c) If n ≥ 3 and n is odd, it applies the multiple application Simpson's 1/3 rule
9 on the first n - 3 subintervals, and applies the Simpson's 3/8 rule on the
10 last three subintervals.
11
12 @author: Jacob Needham
13 """
14 import math
15
16 def function(x):
17     y = x**0.1 * (1.2 - x)*(1 - math.exp(20*(x-1)))
18     return y
19
20 def multiTrap(n,a,b):
21     h = (b-a)/n
22     x = a
23     total = function(x)
24     for i in range(1,n):
25         x += h
26         total += 2*function(x)
27     total += function(b)
28     I =(b-a)* total/(2*n)
29     return I
30
31 def simp13(a,b,n,h):
32     x=a
33     total = function(x)
34     #h=(b-a)/n
35     for i in range(2,n-1,2):
36         x += h
37         total = total + 4*function(x)
38         x += h
39         total = total + 2*function(x)
40     x += h
41     total = total + 4*function(x)
42     x += h
```

```
In [84]: runfile('C:/Users/Needh/Documents/Spring 2019/
Numerical Methods/HW 7/Problem 2.py', wdir='C:/Users/
Needh/Documents/Spring 2019/Numerical Methods/HW 7')
The approximate area under the curve is
0.6020681796156371

In [85]:
```

```

"""
Homework 7 Problem 3
Develop an algorithm which, for a given function f(x), interval bounds a and b with a < b,
and error tolerance per subinterval tol, applies adaptive quadrature to approximate the
integration of f(x) dx from a to b based on the pseudocode that was presented in class
and can be found on page 642 of the textbook.

@author: Jacob Needham
"""

import math

def function(x):
    y = x**0.1 * (1.2 - x)*(1 - math.exp(20*(x-1)))
    return y

def quadapt(a,b,tol):
    c = (a+b)/2
    Fa = function(a)
    Fb = function(b)
    Fc = function(c)
    quadap = qStep(a,b,tol,Fa,Fb,Fc)
    return quadap

def qStep(a,b,tol,Fa,Fb,Fc):
    h1 = b-a
    h2 = h1/2
    c = (a+b)/2
    Fd = function((a+c)/2)
    Fe = function((c + b)/2)
    I1 = h1/6*(Fa +4*Fc + Fb)
    I2 = h2/6*(Fa + 4*Fd + 2*Fc + 4*Fe + Fb)
    if abs(I2 - I1) <= tol:
        I = I2 + (I2 - I1)/15
    else :
        Ia = qStep(a,c,tol,Fa,Fc,Fd)
        Ib = qStep(c,b,tol,Fc,Fb,Fe)
        I = Ia + Ib
    return I

def main():
    a=0
    b=1
    tol = .00001
    print("The value found for a tolerance of .0001:",quadapt(a,b,tol))

main()

```

```
1 """
2 Homework 7 Problem 3
3 Develop an algorithm which, for a given function f(x), interval bounds a and b with a < b,
4 and error tolerance per subinterval tol, applies adaptive quadrature to approximate the
5 integration of f(x) dx from a to b based on the pseudocode that was presented in class
6 and can be found on page 642 of the textbook.
7
8 @author: Jacob Needham
9 """
10 import math
11
12 def function(x):
13     y = x**0.1 * (1.2 - x)*(1 - math.exp(20*(x-1)))
14     return y
15
16 def quadapt(a,b,tol):
17     c = (a+b)/2
18     Fa = function(a)
19     Fb = function(b)
20     Fc = function(c)
21     quadap = qStep(a,b,tol,Fa,Fb,Fc)
22     return quadap
23
24 def qStep(a,b,tol,Fa,Fb,Fc):
25     h1 = b-a
26     h2 = h1/2
27     c = (a+b)/2
28     Fd = function((a+c)/2)
29     Fe = function((c + b)/2)
30     I1 = h1/6*(Fa +4*Fc + Fb)
31     I2 = h2/6*(Fa + 4*Fd + 2*Fc + 4*Fe + Fb)
32     if abs(I2 -I1) <= tol:
33         I = I2 + (I2 - I1)/15
34     else :
35         Ia = qStep(a,c,tol,Fa,Fc,Fd)
36         Ib = qStep(c,b,tol,Fc,Fb,Fe)
37         I = Ia + Ib
38     return I
39
40 def main():
41     a=0
42     b=1
43     tol=0.0001
```

```
In [89]: runfile('C:/Users/Needh/Documents/Spring 2019/
Numerical Methods/HW 7/Problem 3.py', wdir='C:/Users/
Needh/Documents/Spring 2019/Numerical Methods/HW 7')
The value found for a tolerance of .0001:
0.6022912886645673

In [90]:
```



```
"""
```

Homework 7 Problem 4

Apply the algorithms you developed in questions 1-3 above to approximate $f(x) = x^{0.1} * (1.2 - x)(1 - e^{20(x-1)})$ dx from 0 to 1, for varying values of n and tol. Note that this integral is not easy to evaluate analytically! Using the true value of 0.602298, plot Et as a function of n for the algorithms you developed in questions 1 and 2, and plot Et as a function of tol for the algorithm you developed for question 3. Use your best judgement to determine appropriate ranges of values for n and tol to be included in the plots.

@author: Jacob Needham

```
"""
```

```
import math
import numpy as np
from collections import OrderedDict
import matplotlib.pyplot as graph

def function(x):
    y = x**0.1 * (1.2 - x)*(1 - math.exp(20*(x-1)))
    return y

def multiTrap(a,b,n):
    h = (b-a)/n
    x = a
    total = function(x)
    for i in range(1,n):
        x += h
        total += 2*function(x)
    total += function(b)
    I = (b-a)* total/(2*n)
    return I

def simp13(a,b,n,h):
    x=a
    total = function(x)
    #h=(b-a)/n
    for i in range(2,n-1,2):
        x += h
        total = total + 4*function(x)
        x += h
        total = total + 2*function(x)
    x += h
    total = total + 4*function(x)
    x += h
    total = total + function(x)
    I = (b-a)*total/(3*n)
    return I

def simp38(a,b,n,h):
    x=a
    dx = (b-a)/n
    total = function(x)
    x += dx
    total += 3*function(x)
    x += dx
```



```

total += 3*function(x)
x += dx
total += function(x)
ans = (b-a)*total/8
return ans

def quadapt(a,b,tol):
    c = (a+b)/2
    Fa = function(a)
    Fb = function(b)
    Fc = function(c)
    quadap = qStep(a,b,tol,Fa,Fb,Fc)
    return quadap

def qStep(a,b,tol,Fa,Fb,Fc):
    h1 = b-a
    h2 = h1/2
    c = (a+b)/2
    Fd = function((a+c)/2)
    Fe = function((c + b)/2)
    I1 = h1/6*(Fa +4*Fc + Fb)
    I2 = h2/6*(Fa + 4*Fd + 2*Fc + 4*Fe + Fb)
    if abs(I2 -I1) <= tol:
        I = I2 + (I2 - I1)/15
    else :
        Ia = qStep(a,c,tol,Fa,Fc,Fd)
        Ib = qStep(c,b,tol,Fc,Fb,Fe)
        I = Ia + Ib
    return I

def plotSpace ():
    a = 0
    b = 1
    true = 0.602298
    for n in range(1,50):
        ea = (true - multiTrap(0,1,n))/true *100
        graph.plot(n,ea,'b.', label = 'Multi Trap')
        h = (b-a)/n
        if n == 1:
            area = multiTrap(n,a,b)
            eb = (true-area)/true *100
            graph.plot(n,eb,'r.')

        if n%2 ==0:
            area = simp13(a,b,n,h)
            ec = (true-area)/true *100
            graph.plot(n,ec,'r.', label = 'Simpsons 1/3')

        if n%2 != 0:
            areaA = simp13(a,b-2*h,n,h)
            areaB = simp38(b-2*h,b,n,h)
            ed = (true -(areaA+areaB))/true *100
            graph.plot(n,ed,'g.', label = 'Simpsons 1/3 and 3/8')

graph.ylim(0,100)

```

```

graph.xlim(0,50)

#setting up graph
graph.xlabel('x - axis')
graph.ylabel('y - axis')
graph.title('Homeowrk 7.4:\n %Error Et vs Number of Iterations ')

#grpahing
graph.legend()

handles, labels = graph.gca().get_legend_handles_labels()
by_label = OrderedDict(zip(labels, handles))
graph.legend(by_label.values(), by_label.keys())
graph.show()

plotSpace()

def plotSpace2 ():
    a = 0
    b = 1
    true = 0.602298
    for n in range(100000,1,-100):
        ea = (true - quadapt(a,b,n/100000))/true *100
        graph.plot(n/100000,ea,'k.', label = 'Quadapt')

graph.ylim(0,25)
graph.xlim(0,1)

#setting up graph
graph.xlabel('x - axis')
graph.ylabel('y - axis')
graph.title('Homeowrk 7.4:\n %Error Et vs Number of Iterations')

#grpahing
graph.legend()

handles, labels = graph.gca().get_legend_handles_labels()
by_label = OrderedDict(zip(labels, handles))
graph.legend(by_label.values(), by_label.keys())
graph.show()

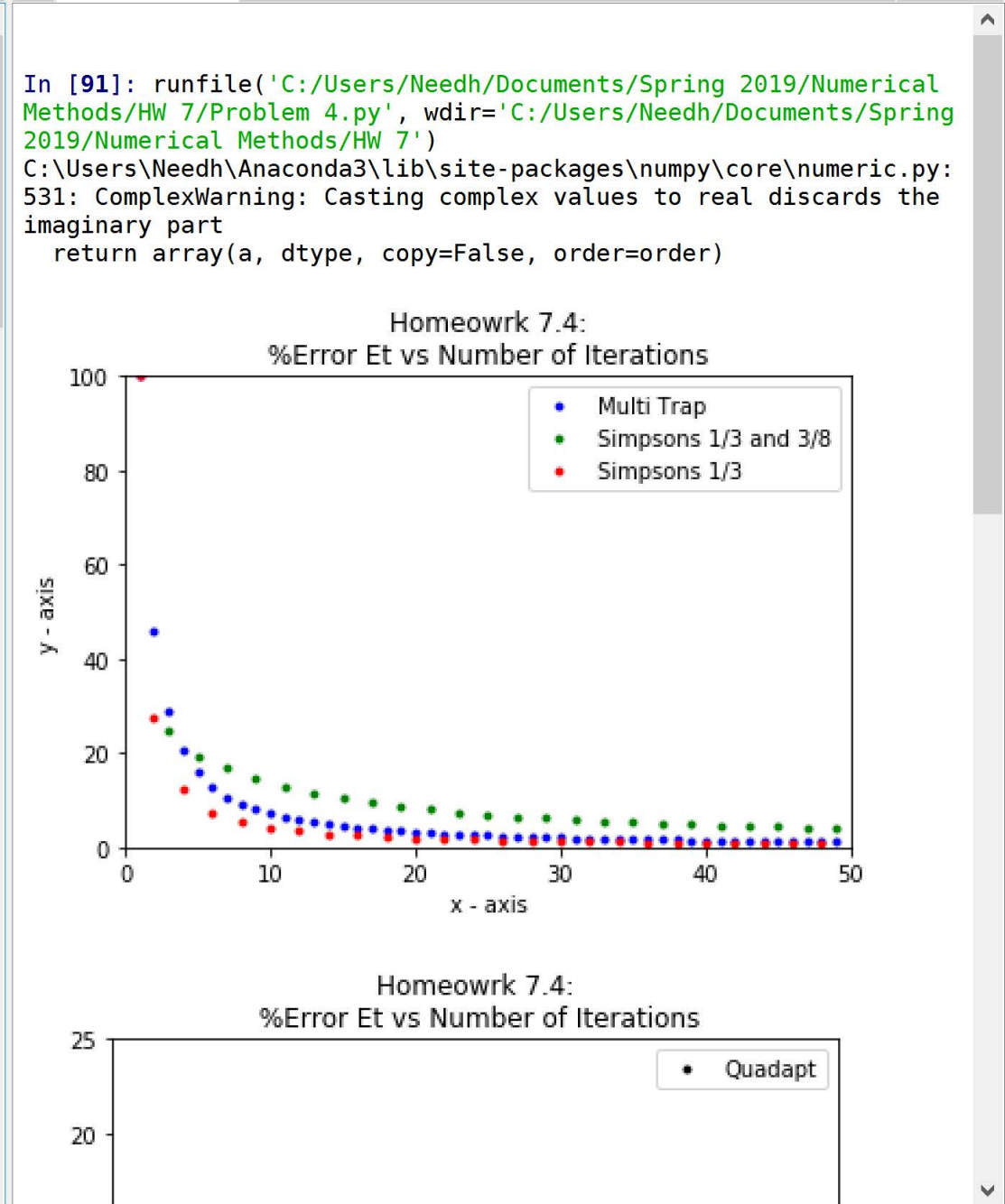
plotSpace2()

```

```

1 """
2 Homework 7 Problem 4
3 Apply the algorithms you developed in questions 1-3 above to approximate
4  $f(x) = x^{0.1} * (1.2 - x)(1 - e^{20(x-1)})$  dx from 0 to 1, for varying values of n and tol.
5 Note that this integral is not easy to evaluate analytically! Using the true value of
6 0.602298, plot Et as a function of n for the algorithms you developed in questions
7 1 and 2, and plot Et as a function of tol for the algorithm you developed for question 3.
8 Use your best judgement to determine appropriate ranges of values for n and tol to be
9 included in the plots.
10
11 @author: Jacob Needham
12 """
13
14 import math
15 import numpy as np
16 from collections import OrderedDict
17 import matplotlib.pyplot as graph
18
19
20 def function(x):
21     y = x**0.1 * (1.2 - x)*(1 - math.exp(20*(x-1)))
22     return y
23
24 def multiTrap(a,b,n):
25     h = (b-a)/n
26     x = a
27     total = function(x)
28     for i in range(1,n):
29         x += h
30         total += 2*function(x)
31     total += function(b)
32     I = (b-a)* total/(2*n)
33     return I
34
35 def simp13(a,b,n,h):
36     x=a
37     total = function(x)
38     #h=(b-a)/n
39     for i in range(2,n-1,2):
40         x += h
41         total = total + 4*function(x)
42         x += h

```



```
1 """
2 Homework 7 Problem 4
3 Apply the algorithms you developed in questions 1-3 above to approximate
4  $f(x) = x^{0.1} * (1.2 - x)(1 - e^{20(x-1)})$  dx from 0 to 1, for varying values of n and tol.
5 Note that this integral is not easy to evaluate analytically! Using the true value of
6 0.602298, plot Et as a function of n for the algorithms you developed in questions
7 1 and 2, and plot Et as a function of tol for the algorithm you developed for question 3.
8 Use your best judgement to determine appropriate ranges of values for n and tol to be
9 included in the plots.
10
11 @author: Jacob Needham
12 """
13
14 import math
15 import numpy as np
16 from collections import OrderedDict
17 import matplotlib.pyplot as graph
18
19
20 def function(x):
21     y = x**0.1 * (1.2 - x)*(1 - math.exp(20*(x-1)))
22     return y
23
24 def multiTrap(a,b,n):
25     h = (b-a)/n
26     x = a
27     total = function(x)
28     for i in range(1,n):
29         x += h
30         total += 2*function(x)
31     total += function(b)
32     I =(b-a)* total/(2*n)
33     return I
34
35 def simp13(a,b,n,h):
36     x=a
37     total = function(x)
38     #h=(b-a)/n
39     for i in range(2,n-1,2):
40         x += h
41         total = total + 4*function(x)
42         x += h
```

