```python
"""
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()
```

Problem 1.py | Problem 2.py | Problem 3.py | Problem 4.py | hayden trial.py | HW 2.3.py

Console 2/A

```python
"""
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()
```

```
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]:
```

Permissions: **RW** End-of-lines: **CRLF** Encoding: **UTF-8** Line: **1** Column: **4** Memory: **58 %**

```python
"""
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 ≥ 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()
```

Problem 1.py | Problem 2.py | Problem 3.py | Problem 4.py | hayden trial.py | HW 2.3.py

```
"""
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 ≥ 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
```

Console 2/A

```
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]:
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 1 Column: 4 Memory: 60 %

```python
"""
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()
```

Problem 1.py | Problem 2.py | Problem 3.py | Problem 4.py | hayden trial.py | HW 2.3.py

```python
"""
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
```

Console 2/A

```
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]:
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 44 Column: 52 Memory: 60 %

```python
"""
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)
```

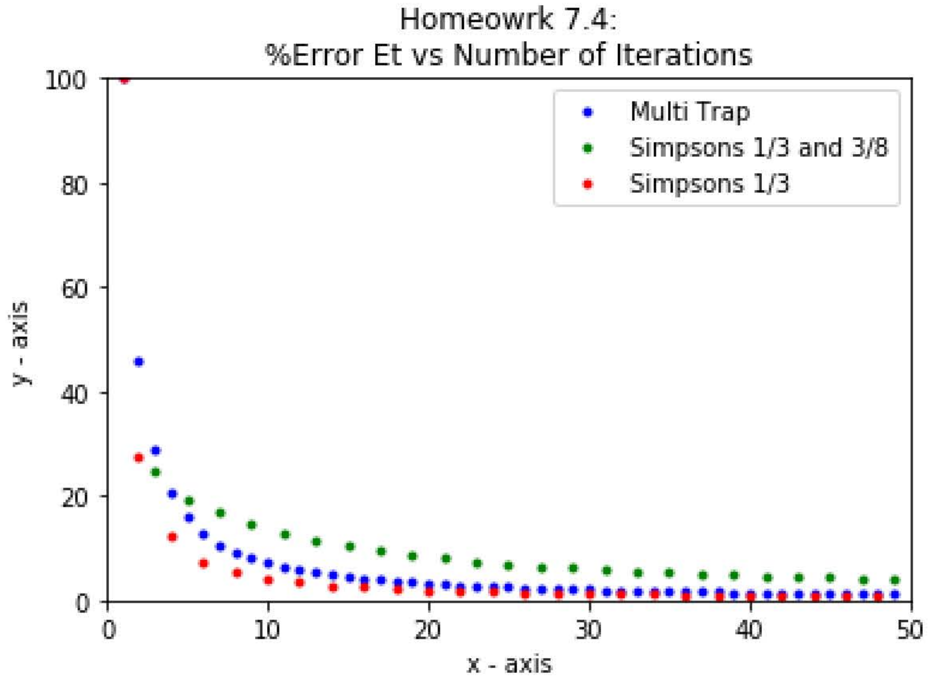
```
    graph.xlim(0,50)

    #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()
```

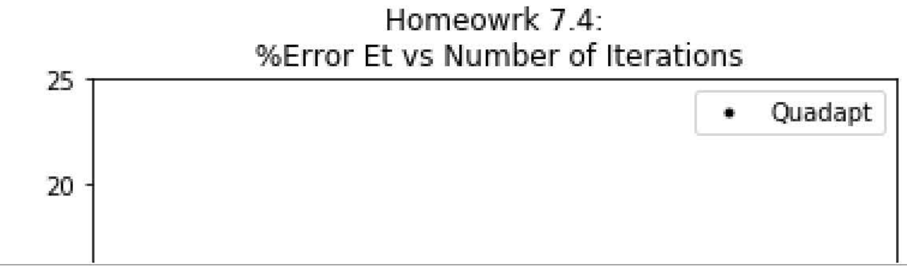
Problem 1.py | Problem 2.py | Problem 3.py | Problem 4.py | hayden trial.py | HW 2.3.py

Console 2/A

```python
"""
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
```

```
In [91]: runfile('C:/Users/Needh/Documents/Spring 2019/Numerical Methods/HW 7/Problem 4.py', wdir='C:/Users/Needh/Documents/Spring 2019/Numerical Methods/HW 7')
C:\Users\Needh\Anaconda3\lib\site-packages\numpy\core\numeric.py:531: ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



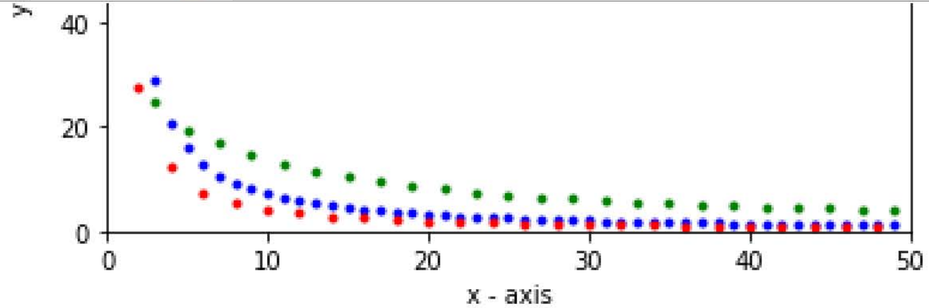


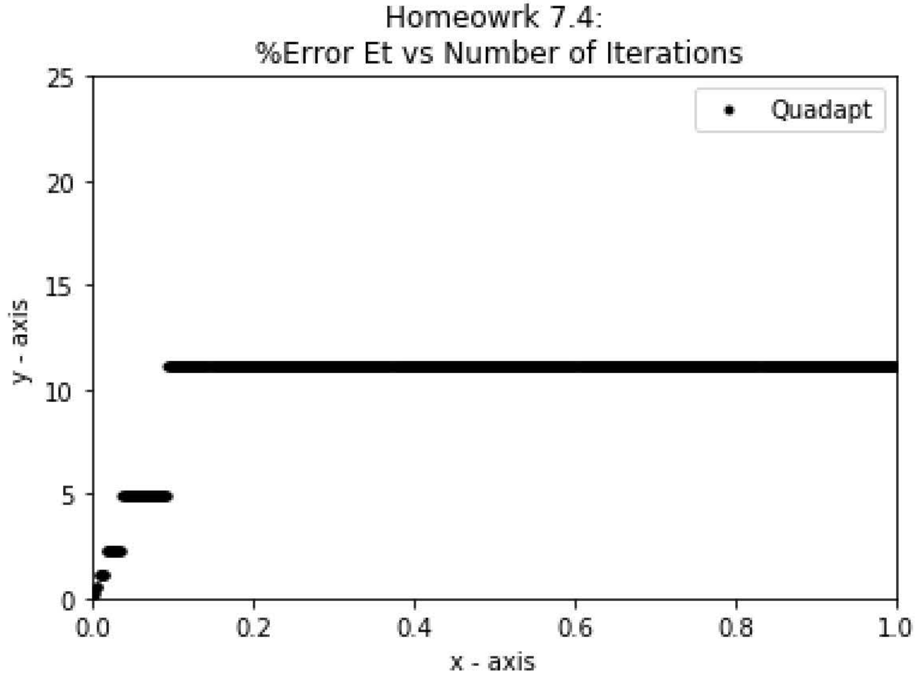
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 1 Column: 4 Memory: 60 %

Problem 1.py Problem 2.py Problem 3.py Problem 4.py hayden trial.py HW 2.3.py

Console 2/A

```python
"""
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
```





In [92]:

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 1 Column: 4 Memory: 60 %