Lab4

May 20, 2019

Instructor(s): Dr. Kenneth Duru
First Semester 2019
Mathematical Sciences Institute
Australian National University

1 Math3511/6111, Scientific Computing

This Lab book must be submitted by **20th May 5pm**. Late Submissions will incur a 5% penalty per working day. Assignment submissions will close on the **27th May 5pm**. Submissions after this time will be invalid.

2 Lab 4: Quadrature

2.1 A. Summary

In this lab you will explore the accuracy of the different quadrature methods. You will explore the Newton-Cotes method using the method of undetermined coefficients. You will also be expected to debug existing code.

The tutors will provide an explanation (and demonstration) of the Newton-Cotes method using the method of undetermined coefficients.

Your task is to implement the Square law and the Trapezoidal law to approximate the following integral

$$\int_{-1}^{1} e^{-3x}\, dx$$

and comment on the accuracy of the two methods.

2.2 B. Labbook -- here comes the part which you include or modify (student)

2.2.1 B1. Left Riemann sum (student) [34 pts]

1) The cell below implements the left Riemann sum for approximating integrals. Before running anything, read through the code and try to understand what it's trying to do. Write a short description on what each procedure is trying to achieve.

Unfortunately, due to the author's wild inexperience with Python, he has made several errors in the following code. Your task is to find each error in the code below, give a brief explaination as to why the error occurs and how to fix the error.

Once you are done fixing the errors in the code, use the square law to approximate the integral

$$\int_{-1}^{1} e^{-3x}\, dx$$

with the partition of the interval ranging from 1 to 100 points. Using the expert knowledge you have gained in the previous labs, along with `scipy.quad` as a reference solution, plot the error of approximation against the size of the partition. Comment on what you observe.

2.2.2 My Report.

Your discussion goes here.

Description of the code:

Firstly we define the function and calculate the reference integral value using scipy.quad.

Secondly we define the function calculating integral using Riemann sum:

Riemannian sum

$$Q(f) = \sum_{k=0}^{n} (x_{k+1} - x_k)\, f(x_k)$$

Then we calculate the integral for partition size from 2 to 100 and their numerical error compared with reference integral value.

Finally we make a plot of numerical error vs partition size.

The original code has serveral errors

errors and their correction

1.numpy -> np in f = lambda x : numpy.exp(-3*x)

This error occurs because he forget he has already import numpy as np.

2 forget to approx = 0

This error occurs because he forget initialise the value before making sums.

3.for i in len(xpts)->for i in range(np.size(xpts)-1)

This error occurs because i is in an array, not a number.

4.-3x->3*

This error occurs because he forgets a operation between 3 and x.

5 errvec = ()->errvec=[]

This error occurs because he mistaken expression empty list.

6.errvec.append(abs(reference - Rect(i))->errvec.append(abs(reference - Rect(i

This error occurs because he forget a right braket correspinding to the left braket of append.

7.def Rect(start = -1, end = 1, f = f, partition_size):->def Rect(partition_size, start = -1, end = 1, f = f):

This error occurs because undefined parameter should come first.

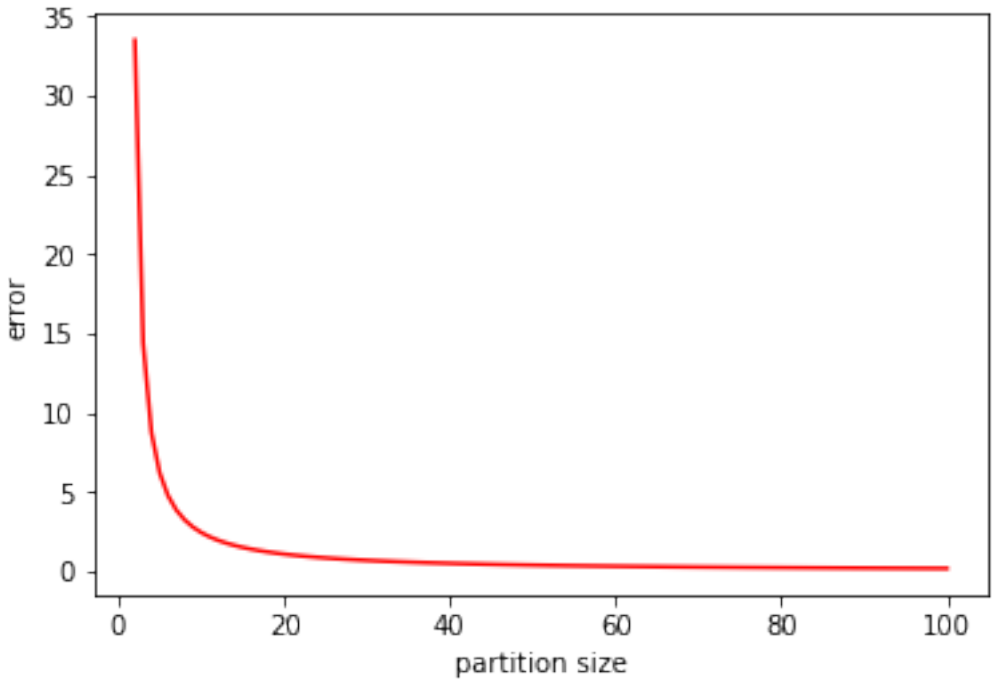
8.plot(size,errvec,'r-'); ->plt.plot(size,errvec,'r-');

This error occurs bacause he forget the plt before plot.

```
In [10]: %matplotlib inline
         import numpy as np
         import scipy as sp
         import math
         import scipy.integrate as integrate
         import pylab as plt
```

```
f = lambda x : np.exp(-3*x) #fix 1

# reference solution of the integral
reference = integrate.quad(lambda x: np.exp(-3*x),-1,1)[0] #fix 1,4


# python function implementing the Riemann sum
def Rect(partition_size, start = -1, end = 1, f = f):
    xpts = np.linspace(start,end,partition_size)
    ypts = f(xpts)
    approx = 0 #fix 2
    for i in range(np.size(xpts)-1):    #fix 3
        approx = approx + (xpts[i+1]-xpts[i])*f(xpts[i])
    return approx

# Initializing list of error
errvec = []#fix 6
for i in range(2,101):
    errvec.append(abs(reference - Rect(i))) #fix 7

# plot of the errors
size = list(range(2,101)) #fix 5
plt.plot(size,errvec,'r-');
plt.xlabel('partition size')
plt.ylabel('error')
```

Out[10]: Text(0,0.5,'error')



comments: As the size of partition increases, the error of square law decreases.

As the error bound for Riemann sum is:

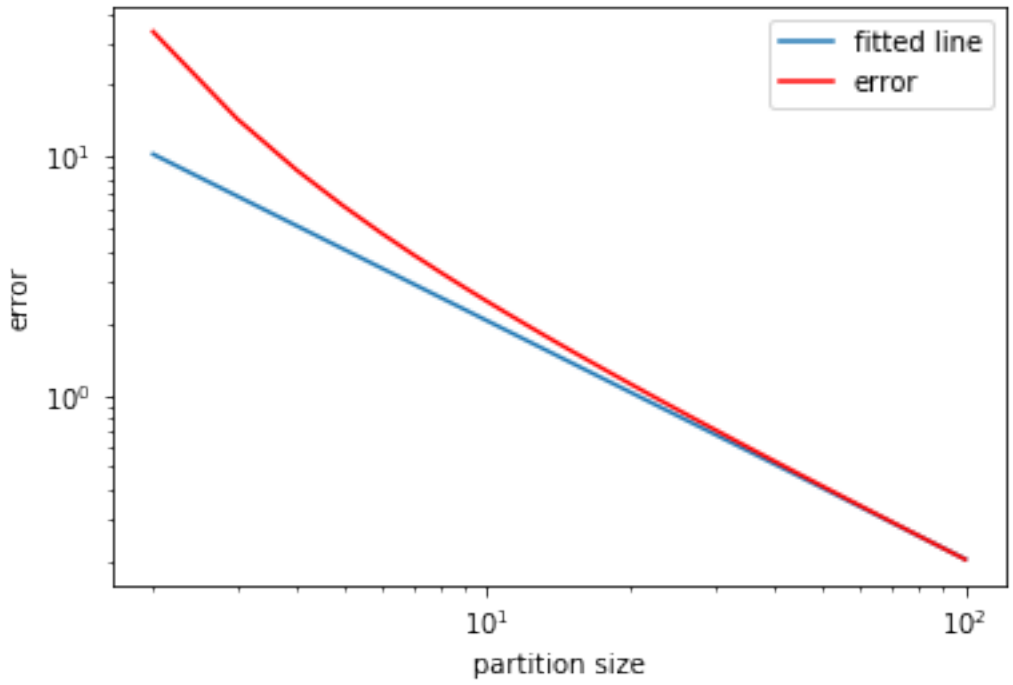
$$|e(x)| \leq Mh \sum_{k=1}^{n} (x_k - x_{k-1}) = M(b-a)h$$

where M is Lipschitz constant of f.

The error for Riemann sum should be of order $O(N^{-1})$, where N is partition size.

The following plot compare the error vs $\frac{20.5}{N}$ in loglog space. The fit between two curves states that the the error for Riemann sum should be of order $O(N^{-1})$

```
In [33]: err=20.5/np.array(size)
         plt.loglog(size,err, label='fitted line')
         plt.loglog(size,errvec,'r-', label='error');
         plt.xlabel('partition size')
         plt.ylabel('error')
         plt.legend()

Out[33]: <matplotlib.legend.Legend at 0x1f1c504bdd8>
```



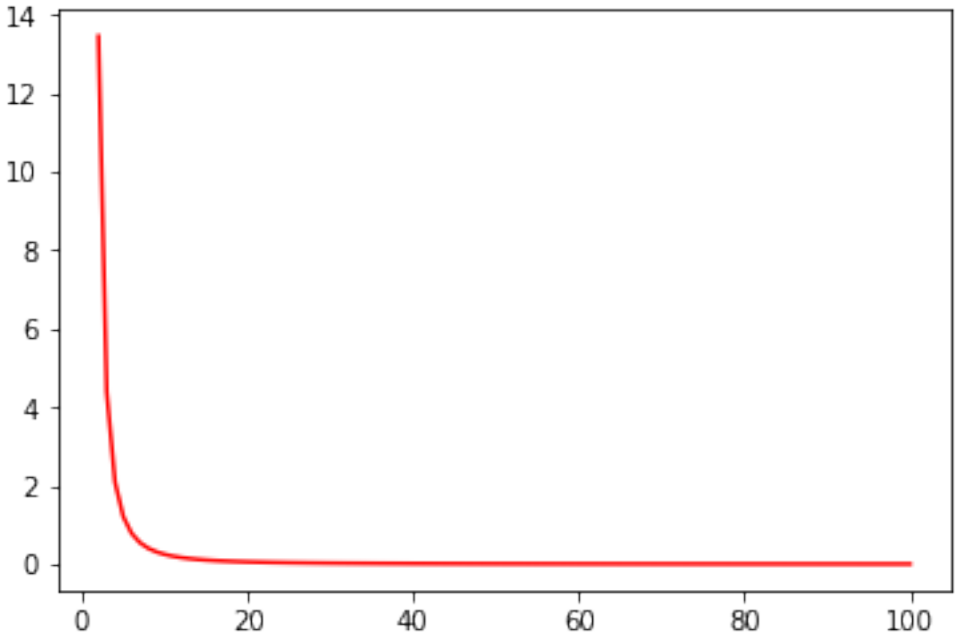
2.2.3 B2. Composite Trapezoidal rule (student) [33 pts]

2) Repeat the accuracy study from point 1 using the Trapezoidal rule. Observe how the error reduces in comparison to the Riemannian sum; comment on wether this is expected.

2.2.4 My Report.

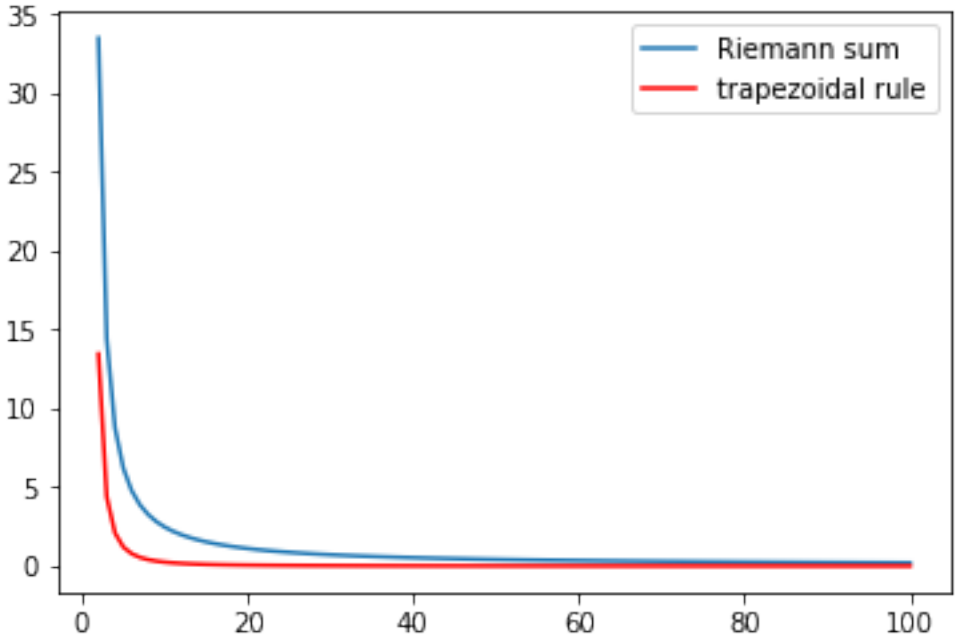
Your discussion goes here.

In [23]:
```
%matplotlib inline
import numpy as np
import scipy as sp
import math
import scipy.integrate as integrate
import pylab as plt

f = lambda x : np.exp(-3*x) #fix 1

# reference solution of the integral
reference = integrate.quad(lambda x: np.exp(-3*x),-1,1)[0] #fix 1,4


# python function implementing the Riemann sum
def Trape(partition_size, start = -1, end = 1, f = f):
    xpts = np.linspace(start,end,partition_size)
    ypts = f(xpts)
    approx = 0 #fix 2
    for i in range(np.size(xpts)-1):   #fix 3
        approx = approx + (xpts[i+1]-xpts[i])/2*(f(xpts[i])+f(xpts[i+1]))
    return approx

# Initializing list of error
errvec_trap = []#fix 6
for i in range(2,101):
    errvec_trap.append(abs(reference - Trape(i))) #fix 7

# plot of the errors
size = np.linspace(2,100,99) #fix 5
plt.plot(size,errvec_trap,'r-');
```



In [35]:
```
plt.plot(size,errvec,label="Riemann sum");
plt.plot(size,errvec_trap,'r-',label="trapezoidal rule");
plt.legend()
```

Out[35]: <matplotlib.legend.Legend at 0x1f1c69cbd30>



comments:

1 As the size of partition increases, the error of Trapezoidal rule decreases.

2 Error reduces faster in comparison to the Riemannian sum; this is expected as The error for Trapezoidal rule should be of order $O(N^{-2})$ while for Riemann sum it is $O(N^{-1})$. Hence the error for Trapezoidal rule will decrease faster.

Addtional analysis: As the error for Trapezoidal rule is

$$e = T(f,h) - I(f) = \frac{h^2}{12}(f'(b) - f'(a)) + O(h^3)$$

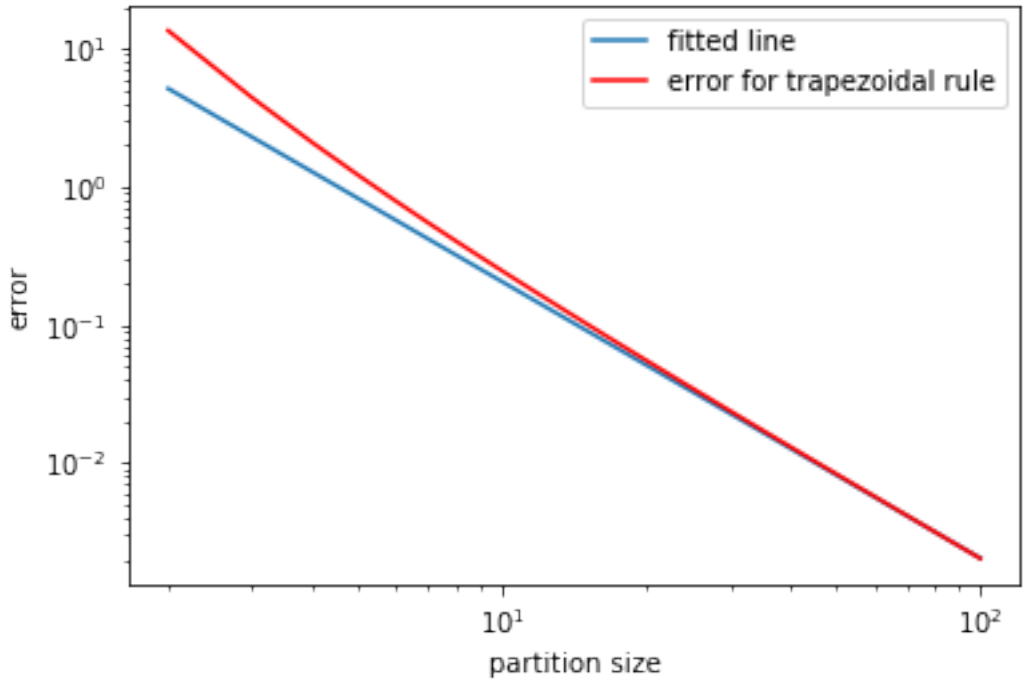
The error for Trapezoidal rule should be of order $O(N^{-2})$, where N is partition size.

The following plot compare the error vs $\frac{20.5}{N^2}$ in loglog space. The fit between two curves states that the the error for Trapezoidal rule should be of order $O(N^{-2})$

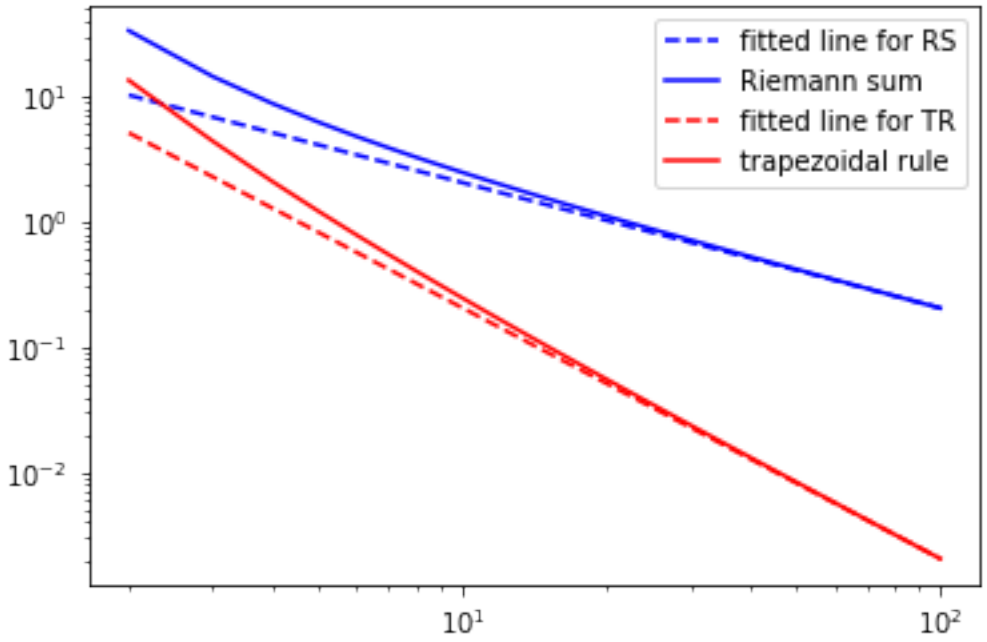
We also make a plot to compare the error for Riemann sum and trapezoidal rule in the last.

```
In [71]: err_2=20.5/np.array(size)**2
         plt.loglog(size,err_2,label='fitted line')
         plt.loglog(size,errvec_trap,'r-',label='error for trapezoidal rule');
         plt.xlabel('partition size')
         plt.ylabel('error')
         plt.legend()
```

```
Out[71]: <matplotlib.legend.Legend at 0x1f1c7d6d5c0>
```



```
In [40]: plt.loglog(size,err,'b--',label='fitted line for RS')
         plt.loglog(size,errvec,'b',label="Riemann sum");
         plt.loglog(size,err_2,'r--',label='fitted line for TR')
         plt.loglog(size,errvec_trap,'r-',label="trapezoidal rule");
         plt.legend()

Out[40]: <matplotlib.legend.Legend at 0x1f1c6a0d0f0>
```



2.2.5 B3. Newton Cotes Method (student) [33 pts]

3) Use the Newton-Cotes formula

$$\int_a^b f(x) = \sum_{i=0}^{4} A_i f(x_i)$$

to estimate the integral

$$\int_{-1}^{1} e^{-3x}$$

with 5 evenly spaced grid points (compare to your reference value).

(Hint: Use the method of undetermined coefficients to solve for the A_i, by substituting in $f = 1, x, x^2, x^3, x^4$ and demanding that the result of the integral be exact)

Repeat with 7 and 9 points. Comment on the improvement to your approximations.

2.2.6 My Report.

Your discussion goes here.

I use the method in lecture to calculate the weights

equations

$$\sum_{k=0}^{n} k^j w_k = \frac{n^{j+1}}{j+1}, \quad j = 0, \ldots, n$$

matrix is Vandermonde matrix

```
In [72]: from scipy.linalg import lu_factor, lu_solve
         # compute Newton-Cotes weights for 5,7 and 9 points
         w = np.zeros((3,10))
         for i, n in enumerate((4,6,8)):
             print("\n n = {}:".format(n),end='    ')
             x = np.linspace(0,n,n+1)
             Van = np.transpose(np.vander(x,increasing=True))
             b = np.zeros(n+1)
             for j in range(n+1):
                 b[j]=n**(j+1)/(j+1)
             lu = lu_factor(Van)
             ww = lu_solve(lu,b)
             w[i,0:n+1]=ww
             for j in range(n):
                 print("w{} = {:4.2f}".format(j,w[i,j]),end='    ')
```

```
 n = 4:    w0 = 0.31    w1 = 1.42    w2 = 0.53    w3 = 1.42
 n = 6:    w0 = 0.29    w1 = 1.54    w2 = 0.19    w3 = 1.94    w4 = 0.19    w5 = 1.54
 n = 8:    w0 = 0.28    w1 = 1.66    w2 = -0.26    w3 = 2.96    w4 = -1.28    w5 = 2.96    w6 = -0.26
```

Then calculate the integral using the quadrature rule:

$$Q(f, a, b) = \sum_{i=0}^{n} w_i f(x_i)$$

```
In [73]: %matplotlib inline
         import numpy as np
         import scipy as sp
         import math
         import scipy.integrate as integrate
         import pylab as plt

         f = lambda x : np.exp(-3*x) #fix 1

         # reference solution of the integral
         reference = integrate.quad(lambda x: np.exp(-3*x),-1,1)[0] #fix 1,4
```

```
# python function implementing Newton-Cotes methods
for i, n in enumerate((4,6,8)):
    Q = np.sum(w[i,:n+1]*np.exp(-3*np.linspace(-1,1,n+1)))*2/n
    print(("n = {}, Q = {:8.7f}, Error = {:6.1e}"\
           .format(n,Q,Q-reference)))
```

```
n = 4, Q = 6.7444774, Error = 6.6e-02
n = 6, Q = 6.6812933, Error = 2.7e-03
n = 8, Q = 6.6786691, Error = 8.6e-05
```

Comments: As the number of evenly spaced grid points increases, the error decreases.

Analysis: From lecture note, in the case of even n the first (constant) term does not contribute to the error and one gets error formula:

$$|E| \leq \frac{h^{n+3}}{2(n+1)!} |f^{(n+2)}(\xi)| \int_0^n |w(x)|\, dx$$

from which we can see that error bound is decreasing with increasing n