Solución Tarea 5
FISI 6510

Guillermo Fidalgo

9 de marzo de 2021

Exercise 5.7: Consider the integral

$$I = \int_0^1 \sin^2 \sqrt{100x}\, \mathrm{d}x$$

a) Write a program that uses the adaptive trapezoidal rule method of Section 5.3 and Eq. (5.34) to calculate the value of this integral to an approximate accuracy of $\epsilon = 10^{-6}$ (i.e., correct to six digits after the decimal point). Start with one single integration slice and work up from there to two, four, eight, and so forth. Have your program print out the number of slices, its estimate of the integral, and its estimate of the error on the integral, for each value of the number of slices N, until the target accuracy is reached. (Hint: You should find the result is around $I = 0.45$.)

b) Now modify your program to evaluate the same integral using the Romberg integration technique described in this section. Have your program print out a triangular table of values, as on page 161, of all the Romberg estimates of the integral. Calculate the error on your estimates using Eq. (5.49) and again continue the calculation until you reach an accuracy of $\epsilon = 10^{-6}$. You should find that the Romberg method reaches the required accuracy considerably faster than the trapezoidal rule alone.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    return (np.sin((100*x)**.5))**2

def trapezoid(f,a=0,b=1,h=1,N=1):
    h=(b-a)/N
    s = 0.5*f(a) + 0.5*f(b)
    for k in range(1,N):
        s += f(a+k*h)
    return h*s
```

a)

[2]:
```
N = 1 # Número inicial de pasos
a=0
b=1
```

```
h=(b-a)/N
I=[]
k =0

epsilon,err= 1e-6,1

I.append(trapezoid(f,N=1)) # valor inicial $I_1$
print("N = {:4}\t I = {:.7f}\t error = N/A".format(N,I[0]))

while abs(err) > epsilon:
    N=2*N
    h = h/2
    s=0
    for n in range(1,N,2):
        s+= f(a+n*h)
    i= .5*I[k] + h*s
    I.append(i)
    k+=1
    err =  (I[k] - I[k-1])/3
    print("N = {:4}\t I = {:.7f}\t error = {:.7f}".format(N,i,err))
```

```
N =    1         I = 0.1479795   error = N/A
N =    2         I = 0.3252319   error = 0.0590841
N =    4         I = 0.5122829   error = 0.0623503
N =    8         I = 0.4029974   error = -0.0364285
N =   16         I = 0.4301034   error = 0.0090353
N =   32         I = 0.4484147   error = 0.0061038
N =   64         I = 0.4539129   error = 0.0018328
N =  128         I = 0.4553485   error = 0.0004785
N =  256         I = 0.4557113   error = 0.0001209
N =  512         I = 0.4558022   error = 0.0000303
N = 1024         I = 0.4558249   error = 0.0000076
N = 2048         I = 0.4558306   error = 0.0000019
N = 4096         I = 0.4558321   error = 0.0000005
```

b)

[3]:
```
def romberg(f,a,b,h,N,precision=1e-6):
    R=[]
    I1=trapezoid(f,a,b,h,N)
    R.append([I1])
    ERR=[]
    err=1
    i=1
    while abs(err) > precision:
        N=2*N
```

```
        h=h/2
        s=0
        for n in range(1,N,2):
            s+= f(a+n*h)
        I_1m= .5*I1 + h*s
        R.append([I_1m])
        I1=I_1m


        for m in range(1,i+1):
            err= (1/(4**m-1))*(R[i][m-1] - R[i-1][m-1])
            ERR.append(err)
            R_im= R[i][m-1] + err
            R[i].append(R_im)
        i+=1

    return R,ERR,N
```

[4]:

```
N=1
a=0
b=1
h=(b-a)/N

r=romberg(f,a,b,h,N)


for i in r[0]:
    print(i)
print("\nN = {} steps".format(r[2]))
```

```
[0.147979484546652]
[0.3252319078064746, 0.38431604889308213]
[0.5122828507233315, 0.5746331650289505, 0.5873209727713417]
[0.4029974484782483, 0.3665689810632206, 0.35269803546550527, 0.34897386185747614]
[0.43010336929474696, 0.4391386762335798, 0.4439766559116038, 0.4454255229028117, 0.4458037647108326]
[0.44841466578746997, 0.4545184312850443, 0.45554374828847527, 0.455727352929378, 0.4557677522628155, 0.45577749223109704]
[0.4539129312153758, 0.45574568635801105, 0.4558275033628755, 0.45583200741167545, 0.45583241782140993, 0.45583248103309965, 0.4558324944613785]

N = 64 steps
```

Exercise 5.8: Write a program that uses the adaptive Simpson's rule method of Section 5.3 and Eqs. (5.35) to (5.39) to calculate the same integral as in Exercise 5.7, again to an approximate accuracy of $\epsilon = 10^{-6}$. Starting this time with two integration slices, work up from there to four, eight, and so forth, printing out the results at each step until the required accuracy is reached. You should find you reach that accuracy for a significantly smaller number of slices than with the trapezoidal rule calculation in part (a) of Exercise 5.7, but a somewhat larger number than with the Romberg integration of part (b).

[5]:
```
def adap_simp(f,a=0,b=1,N=2,h=.5):
    fa=f(a)
    fb=f(b)
    integral=0

    for k in range(2,N-1,2):
        integral=f(a+k*h)+integral

    S=(integral*2 + fa+fb)*(1/3)


    integral=0
    for k in range(1,N,2):
        integral=f(a+k*h)+integral
    T=integral*(2/3)


    integral=h*(S+2*T)
    I=integral
    return I,S,T
```

[24]:
```
N=2
h=.5
a=0
b=1

I,S,T=adap_simp(f,N=N,h=h,a=a,b=b)
err = 1
tol=1e-6
print("I = {:.7f}\t N= {}\t error is {:.7f}".format(I,N,err))
while abs(err)>tol:

    N=2*N
    h=h/2
    S=S+T
    _,_,T=adap_simp(f,a,b,h=h,N=N)

    Inew = h*(S+2*T)
```

```
    err=(1/15)*(Inew-I)
    I=Inew
    print("I = {:.7f}\t N= {}\t error is {:.7f}".format(I,N,err))
```

```
I = 0.3843160    N= 2    error is 1.0000000
I = 0.5746332    N= 4    error is 0.0126878
I = 0.3665690    N= 8    error is -0.0138709
I = 0.4391387    N= 16   error is 0.0048380
I = 0.4545184    N= 32   error is 0.0010253
I = 0.4557457    N= 64   error is 0.0000818
I = 0.4558270    N= 128  error is 0.0000054
I = 0.4558322    N= 256  error is 0.0000003
```