CA_lab_integration_romberg

October 26, 2020

#Romberg's Integration #### Computational Astrophysics * Shivam Kumaran * SC17B122 * 26 Oct 2020

Open Program

[ ]:
```
import numpy as np
```

0.1 Defining Composite trapezoidal rule

[ ]:
```
def integral(f,a_0,b_0,n,kind='simp'):
    h = (b_0-a_0)/n
    t0 = f(a_0)+f(b_0)

    if(kind=='tpz'):
        t1 = sum([f(a_0+k*h) for k in range(1,n)])
        val = (h/2)*(t0+2*t1)
    return(val)
```

1 Calculation of Romberg's integration

For given value of (m,n)
using recurrent function

[ ]:
```
def calc_r(f,a,b,i,j):
    if(j<=i):
        if(j==1):
            #print('calculating: ' , i, j)
            val = integral(f,a,b,int(2**(i-1)), kind='tpz')
            return(val)
        else:
            r_i_jm1 = calc_r(f,a,b,i,j-1)
            r_im1_jm1 =  calc_r(f,a,b,i-1,j-1)
            r_ij =( 4**(j-1)*r_i_jm1 - r_im1_jm1 ) / (4**(j-1)-1)
        return r_ij
    else:
        print('Wrong indices')
        return(0)
```

2 Matrix for Romberg's integration

Returns Matrix corresponding to Romberg's integration value,
Size of this matrix is limited to that of the given accuracy or
The limiting size provided by the user (smaller of the two)

[ ]:
```
def romberg_mat(f,a,b,m,n,e):
    '''
    f : integrand function
    a,b : integral limits
    m ,n :  maximum order
    e : desired accuracy
    returns: romberg matrix of size corresponding
            to size computed upto given error value ,
            or the maximum order given
    '''
    err = 1
    prev = calc_r(f,a,b,1,1)
    mat = np.zeros((m,n))
    mat[0][0] = prev
    j_max = 0
    for i in range(2,m+1):
        for j in range(1,i+1):
            if(j_max<j):
                j_max = j
            #print(i,j)
            nxt = calc_r(f,a,b,i,j)
            mat[i-1][j-1] = nxt
            err = abs(prev-nxt)
            if(err<e):
                #print(i,j)
                mat = mat[:i,:j_max]
                #order = 2**np.shape(mat)[0] , np.shape(mat)[1]+1
                return mat , (i,j)
            else:
                prev = nxt

    raise ValueError('Accuracy could not be achieved with given order limit')
```

[ ]:
```
def tabulate(res , order):
    m , n = res.shape
    print('# intervals , Order >'+str(np.arange(1,n+1)))
    for i in range(0,m):
        if(i+1==order[0]):
            print('{}\t : {}'.format(2**i , res[i,:order[1]]))
        else:
            print('{}\t : {}'.format(2**i , res[i,:i+1]))
```

```
    print("required accuracy achieved at order :{}".format(str(order)) )
#for i in range(1,10):

#    print(calc_r(f,0,1,i,i))
```

#Problem 01

2.0.1 Function I

[ ]:
```
n_max , m_max = 7,7
a ,b = 0,1
e = 1e-8
def f(x):
    val = np.exp(-1*x**2)
    return val
res , order = romberg_mat(f,a,b,n_max,m_max,e)
tabulate(res,order)
integ_val = res[order[0]-1][order[1]-1]
print('integration Value is :{:.8f}'.format(integ_val))
```

```
# intervals , Order >[1 2 3 4]
1         : [0.68393972]
2         : [0.73137025 0.74718043]
4         : [0.7429841  0.74685538 0.74683371]
8         : [0.74586561 0.74682612 0.74682417 0.74682402]
16        : [0.7465846  0.74682426 0.74682413 0.74682413]
required accuracy achieved at order :(5, 4)
integration Value is :0.74682413
```

2.0.2 Function II

[ ]:
```
np.set_printoptions(linewidth=140)
def f_b(k):
    def f(x):
        if(k==1):
            if(np.sin(x)==1):
                val = 0
        else:
            val=1/((1-k*(np.sin(x))**2)**0.5)
        return val
    return f
k = 0.5
n_max , m_max = 20,20
e = 1e-8
a , b = 0 ,  np.pi/2
res , order = romberg_mat(f_b(k),a,b,n_max,m_max,e)
tabulate(res , order)
integ_val = res[order[0]-1][order[1]-1]
```

```
print('integration Value is :{:.8f}'.format(integ_val))
```

```
# intervals , Order >[1 2 3 4]
1         : [1.8961189]
2         : [1.85495913 1.84123921]
4         : [1.85407523 1.85378059 1.85461669]
8         : [1.85407468 1.85407449 1.85409409 1.85408579]
16        : [1.85407468 1.85407468]
required accuracy achieved at order :(5, 2)
integration Value is :1.85407468
```

Function II , K dependency

[ ]:
```
k_list = np.linspace(0 ,  0.99999 , 10)
print('K \t \t (n,m) \t \t Integration Value')
print('_____________________________________________')
for k in k_list:
    res , order = romberg_mat(f_b(k),a,b,n_max,m_max,e)
    integ_val = res[order[0]-1][order[1]-1]
    print('{:.6f} \t {} \t , {:.8f}'.format(k, order ,integ_val))
```

```
K                (n,m)           Integration Value

-----------------------------------------------
0.000000         (2, 1)          , 1.57079633
0.111110         (4, 2)          , 1.61738624
0.222220         (4, 2)          , 1.67100320
0.333330         (4, 2)          , 1.73391483
0.444440         (5, 2)          , 1.80966414
0.555550         (5, 2)          , 1.90423606
0.666660         (5, 2)          , 2.02895033
0.777770         (6, 2)          , 2.20946813
0.888880         (6, 2)          , 2.52858806
0.999990         (12, 10)        , 7.14279279
```

2.0.3 Conclusion

We see that the number of interval required depends on value of K. With K approaching 1 , the order required also increases. And we have no finite value of integration for k=1.