Mathematical Physics - III (Practical)
Complex Analysis
Paper: PHS-A-CC-4-8-P

6. Complex analysis:

(a) Integrate $\int_0^\infty \frac{\sin x}{x} dx$ numerically and check with computer integration.
(b) Root finding:
  (i) Compute the nth roots of unity for n= 2, 3, and 4
  (ii) Find the two square roots of -5+12i

Scientific python provides several integration routines. A general purpose tool to solve integrals I of the kind $I = \int_a^b f(x)dx$ is provided by the quad() function of the scipy.integrate module.

Most numerical methods for computing this integral split up the original integral into a sum of several integrals, each covering a smaller part of the original integration interval [a,b]. This re-writing of the integral is based on a selection of integration points $x_i, i = 0, 1, \ldots . n$ that are distributed at an interval [a,b]. Integration points may, or may not, be evenly distributed. An even distribution simplifies expressions and is often sufficient, so we will mostly restrict ourselves to that choice. The different integration methods will differ in the way they approximate each integral. The following three methods are used to find the approximation of a definite integral.

1. Rectangle Rule
2. Trapezoidal Rule
3. Simpson's $\frac{1}{3}^{rd}$ Rule

(a) Integrate $\int_0^\infty \frac{\sin x}{x} dx$

```
import numpy as np
import scipy.integrate as sci

t = 1E-6

def f(x): return np.sin(x)/x

def simps(p, q):

   deltax  = (q-p)*0.5
   s1 = f(p) + f(q)
   s4 = f(p+deltax)
   I0 = 1E-16
   I1 = (s1+4.0*s4)*deltax/3.0
```

```
    i=2; s2=0.0;
    while(abs((I1-I0)/I1) > 0.1*t):
        x   = p+deltax*0.5
        s2 += s4
        s4  = 0.0
        for j in range(0,i):
            s4 += f(x)
            x += deltax
        deltax *= 0.5
        i *= 2
        I0 = I1
        I1 = (s1+2.0*s2+4.0*s4)*deltax/3.0
    return I1

m = float(input('Enter integrals lower limit : '))

I0 = 1E-16

while True:
   q = m
   p = 1.0/m
   I1 = simps(p,q)
   if(abs(I0-I1) < t):
      break
   I0 = I1
   m += 500;

print ('Integral_',p,'^',q,' sin(x)/x dx (numerical) = ', I1)


x=np.linspace(p, q, int(q*10))

print ('Integral_',p,'^',q,' sin(x)/x dx (direct) = ', sci.simps(f(x),x))
```

Output: Lower limit: 0.01

```
========= RESTART: C:/Python38-32/semIV-CC8/sem4/int_sine.py =========
Enter integrals lower limit: 0.01
Integral_ 5.5555524691375165e-05 ^ 18000.010000000002
sin(x)/x dx (numerical) = 1.570726765676473
Integral_ 5.5555524691375165e-05 ^ 18000.010000000002
sin(x)/x dx (direct) = 1.5707128865530906
>>>
```

(b).Root finding:

(iii) Compute the nth roots of unity for n= 2, 3, and 4

(iv) Find the two square roots of -5+12i,

```
import cmath

def nroot(k,n): return cmath.exp(2*cmath.pi*1j*k/n)
n = input("Compute the nth root of unity for n = ");
n=int(n)
for k in range(n):
   print (nroot(k,n))



cn=-5+12j
print ('Square root of', cn, ' is ', cmath.sqrt(cn))


#one can write

x=-5.0
y=12.0
z=complex(x,y)
print (z)
print (cmath.sqrt(z))
```

Output:

```
>>>
======= RESTART: C:/Python38-32/semIV-CC8/sem4/c-root.py  ===============
Compute the nth root of unity for n = 2
(1+0j)
(-1+1.2246467991473532e-16j)
Square root of (-5+12j)  is  (2+3j)
(-5+12j)
(2+3j)
>>>

======== RESTART: C:/Python38-32/semIV-CC8/sem4/c-root.py ===============
Compute the nth root of unity for n = 3
(1+0j)
(-0.4999999999999998+0.8660254037844387j)
(-0.5000000000000004-0.8660254037844384j)
Square root of (2+3j)  is  (1.6741492280355401+0.8959774761298381j)
>>>

>>>
======== RESTART: C:/Python38-32/semIV-CC8/sem4/c-root.py ===============
Compute the nth root of unity for n = 4
(1+0j)
(6.123233995736766e-17+1j)
(-1+1.2246467991473532e-16j)
(-1.8369701987210297e-16-1j)
Square root of (8-3j)  is  (2.8761088075138543-0.5215379877427584j)
>>>
```