Lecture 3:

Example1: A root of $x^3 - 10x^2 + 5 = 0$ lies in the interval (0, 1). Use rootsearch to compute this root with four-digit accuracy.

```
from numpy import sign

def rootsearch(f,a,b,dx):

  x1 = a; f1 = f(a)

  x2 = a + dx; f2 = f(x2)

  while sign(f1) == sign(f2):

    if x1  >=  b: return None,None

    x1 = x2; f1 = f2

    x2 = x1 + dx; f2 = f(x2)

    print('\nx1 is:', np.round(x1,3), '  f(x1) = ', np.round(f1,3), '  x2 is: ', np.round(x2,3),'  f(x2) = ',
np.round(f2,3) )

  else:

    return x1,x2

    print(x1,x2)

import numpy as np


def f(x): return x**3 - 10.0*x**2 + 5.0

x1 = 0.0; x2 = 1.0

for i in range(4):

  print('\n\nstep:',i,':')

  dx = (x2 - x1)/10.0

  x1,x2 = rootsearch(f,x1,x2,dx)

  print('\ndx=',dx)

x = (x1 + x2)/2.0
```

```
print('x =', np.round(x,4))
```

Example 2: Use bisection to find the root of $x^3 - 10x^2 + 5 = 0$ that lies in the interval (0,1) to four-digit accuracy.

```
import numpy as np

import math

# import error

from numpy import sign


def bisection(f,x1,x2,switch=1,tol=1.0e-9):

    f1 = f(x1)

    if f1 == 0.0: return x1

    f2 = f(x2)

    if f2 == 0.0: return x2

    if sign(f1) == sign(f2):

        error.err('Root is not bracketed')

    n = int(math.ceil(math.log(abs(x2 - x1)/tol)/math.log(2.0)))


    for i in range(n):

        x3 = 0.5*(x1 + x2); f3 = f(x3)

        if (switch == 1) and (abs(f3) > abs(f1)) \

                    and (abs(f3) > abs(f2)):

            return None

        if f3 == 0.0: return x3

        if sign(f2)!= sign(f3): x1 = x3; f1 = f3

        else: x2 = x3; f2 = f3
```

```
    return (x1 + x2)/2.0


# from bisection import *

def f(x): return x**3 - 10.0*x**2 + 5.0

x = bisection(f, 0.0, 1.0, tol = 1.0e-4)

print('x =', np.round(x,4))
```

Lecture 4:

Example 1: Use the Newton-Raphson method to obtain successive approximations of $\sqrt{2}$ as the ratio of two integers.

```
def f(x): return x**2-2

def df(x): return 2*x

def newtonRaphson(x, tol=1.0e-7):

    for i in range(30):

        dx=-f(x)/df(x)

        x = x+dx

        if abs(dx) < tol: return x,i

    print('Too many iterations\n')

root,numIter = newtonRaphson(2.0)

print('Root=',root)

print('Number of iterations=',numIter)
```

Example 2: Find the smallest positive zero of

$$f(x) = x^4 - 6.4x^3 + 6.45x^2 + 20.538x - 31.752$$

```
def f(x): return x**4-6.4*x**3+6.45*x**2+20.538*x-31.752

def df(x): return 4*x**3-19.2*x**2+12.9*x+20.538

def newtonRaphson(x, tol=1.0e-9):

    for i in range(30):

        dx=-f(x)/df(x)

        x = x+dx
```

```
        if abs(dx) < tol: return x,i
    print('Too many iterations\n')
root,numIter = newtonRaphson(2.0)
print('Root=',root)
print('Number of iterations=',numIter)
```

Example 3: Use Newton's method to find solutions accurate to within 10^{-4} for the following problems.

$x^3 - 2x^2 - 5 = 0$, [1,4]

```
def f(x):
    return x**3 - 2*x**2 - 5

def df(x):
    return 3*x**2 - 4*x

def newtonRaphson(f, df, a, b, tol=1.0e-9):
    from numpy import sign

    fa = f(a)
    fb = f(b)

    if abs(fa) < tol:
        return a, 0
    if abs(fb) < tol:
        return b, 0
```

```
if sign(fa) == sign(fb):
    raise ValueError("Root is not bracketed")

x = 0.5 * (a + b)

for i in range(1, 31):
    fx = f(x)
    dfx = df(x)

    if abs(fx) < tol:
        return x, i

    # Bracket update
    if sign(fa) != sign(fx):
        b = x
        fb = fx
    else:
        a = x
        fa = fx

    try:
        dx = -fx / dfx
    except ZeroDivisionError:
        dx = b - a  # fallback to bisection
```

```
        x_new = x + dx

        # Keep x within [a, b]
        if (x_new - a) * (x_new - b) > 0:
            dx = 0.5 * (b - a)
            x_new = a + dx

        x = x_new

        if abs(dx) < tol * max(abs(x), 1.0):
            return x, i

    raise RuntimeError("Too many iterations in Newton-Raphson")

# Call the function
root, iterations = newtonRaphson(f, df, a=1.0, b=4.0)
print("Root =", root)
print("Iterations =", iterations)
```

Systems of Equations:

Example 4: Find a solution of

$$\sin x + y^2 + \ln z - 7 = 0$$

$$3x + 2^y - z^3 + 1 = 0$$

$$x + y + z - 5 = 0$$

Using newtonRaphson2. Start with the point (1,1,1).

```python
import numpy as np
import math


def newtonRaphson2(f, x, tol=1.0e-9):
    def jacobian(f, x):
        h = 1.0e-4
        n = len(x)
        jac = np.zeros((n, n))
        f0 = f(x)
        for i in range(n):
            x1 = x.copy()
            x1[i] += h
            f1 = f(x1)
            jac[:, i] = (f1 - f0) / h
        return jac, f0


    for i in range(30):
        jac, f0 = jacobian(f, x)
        if math.sqrt(np.dot(f0, f0) / len(x)) < tol:
            return x, i
        dx = np.linalg.solve(jac, -f0)
        x = x + dx
        if math.sqrt(np.dot(dx, dx)) < tol * max(np.max(np.abs(x)), 1.0):
            return x, i
```

```
    raise RuntimeError("Too many iterations")


# Define the system of equations
def f(x):
    f = np.zeros(len(x))
    f[0] = math.sin(x[0]) + x[1]**2 + math.log(x[2]) - 7.0
    f[1] = 3.0*x[0] + 2.0**x[1] - x[2]**3 + 1.0
    f[2] = x[0] + x[1] + x[2] - 5.0
    return f


x0 = np.array([1.0, 1.0, 1.0])
root, iterations = newtonRaphson2(f, x0)
print("Root:", root)
print("Iterations:", iterations)
print("f(root):", f(root))
input("\nPress return to exit")
```

Lecture 5:

<u>Newton-Gregory Forward interpolation formula</u>

Example: The following table given the population of a town during the last six censuses. Using the Newtown's interpolation formula estimate the population in 1923.

Year (x)	1911	1921	1931	1941	1951	1961
Population (y) (in thousands)	12	15	20	27	39	52

```
import math


years = [1911, 1921, 1931, 1941, 1951, 1961]

populations = [12, 15, 20, 27, 39, 52]


h = years[1] - years[0]


x = 1923

x0 = years[0]

t = (x - x0) / h


n = len(populations)

diff_table = [populations[:]]

for i in range(1, n):

    column = []

    for j in range(n - i):

        delta = diff_table[i-1][j+1] - diff_table[i-1][j]
```

```
        column.append(delta)
        diff_table.append(column)


def newtons_forward(t, diff_table):
    result = diff_table[0][0]
    u_term = 1
    for i in range(1, len(diff_table)):
        u_term *= (t - i + 1)
        term = (u_term * diff_table[i][0]) / math.factorial(i)
        result += term
        return result


estimated_population = newtons_forward(t, diff_table)
print(f"Estimated population in {x} is approximately {estimated_population:.2f} thousand")
```

Newton-Gregory Backward interpolation formula:

Example: Using Newton's backward formula, find the value of $l^{-1.9}$ from the following table of values of l^{-x}.

x	1.00	1.25	1.50	1.75	2.00
l^{-x}	0.3679	0.2865	0.2231	0.1738	0.1353

```
import math
x_values = [1.00, 1.25, 1.50, 1.75, 2.00]
y_values = [0.3679, 0.2865, 0.2231, 0.1738, 0.1353]


h = x_values[1] - x_values[0]
```

```
x = 1.9

n = len(x_values)

u = (x - x_values[-1])/h

diff_table = [y_values[:]]


for i in range(1, n):

    column = []

    for j in range(n - i):

        delta = diff_table[i-1][j+1] - diff_table[i-1][j]

        column.append(delta)

        diff_table.append(column)


def newtons_backward(u, diff_table):

    result = diff_table[0][-1]

    u_term = 1

    for i in range(1, len(diff_table)):

        u_term *= (u + i - 1)

        term = (u_term * diff_table[i][-1]) / math.factorial(i)

        result += term

        return result


estimated_value = newtons_backward(u, diff_table)

print(f"Estimated value of x ln(x) at x = {x} is approximately {estimated_value:.4f}")
```

Lecture 6:

Example: Find the value of $e^{-1.7425}$ by Gauss Forward formula, given that

x	1.72	1.73	1.74	1.75	1.76
e^{-x}	0.17907	0.17728	0.17552	0.17377	0.17204

```
import numpy as np


x_values = [1.72, 1.73, 1.74, 1.75, 1.76]

y_values = [0.17907, 0.17728, 0.17552, 0.17377, 0.17204]

n = len(x_values)


diff_table = np.zeros((n, n))

diff_table[:, 0] = y_values


for j in range(1, n):

    for i in range(n - j):

        diff_table[i][j] = diff_table[i + 1][j - 1] - diff_table[i][j - 1]


x = 1.7425

x0 = x_values[2]

h = x_values[1] - x_values[0]

p = (x - x0) / h


result = diff_table[2][0]
```

```
p_term = 1

fact = 1


for k in range(1, 4):

    if k == 1:

        p_term *= p

    elif k == 2:

        p_term *= (p - 1)

    elif k == 3:

        p_term *= (p + 1)

    fact *= k

    result += (p_term / fact) * diff_table[2 - (k // 2)][k]


print(f"Estimated value of e^(-1.7425) using Gauss Forward Interpolation: {result:.6f}")
```

Example: Using Gauss backward interpolation formula, find $Sin45^0$ from the following table.

θ	20	30	40	50	60	70	80
$Sin\theta$	0.34202	0.50200	0.64279	0.76604	0.86603	0.93969	0.98481

```
import numpy as np

x_values = [20, 30, 40, 50, 60, 70, 80]

y_values = [0.3420, 0.5000, 0.6428, 0.7660, 0.8660, 0.9397, 0.9848]


n = len(x_values)

diff_table = np.zeros((n, n))
```

```
diff_table[:, 0] = y_values

for j in range(1, n):
    for i in range(n-j):
        diff_table[i][j] = diff_table[i][j - 1] - diff_table[i - 1][j - 1]

x = 45
x_n_index = x_values.index(50)
x_n = x_values[x_n_index]
h = 10
p = (x - x_n) / h

result = diff_table[x_n_index][0]
p_term = 1
fact = 1

for k in range(1, 5):  # up to 4th order
    p_term *= (p + (k - 1))
    fact *= k
    result += (p_term / fact) * diff_table[x_n_index][k]

print(f"Estimated sin(45°) using Gauss Backward Interpolation: {result:.6f}")
```

3. Stirling's Interpolation:

```
import numpy as np


x_values = [0.0, 0.5, 1.0, 1.5, 2.0, 2.5]

y_values = [0.0, 0.19146, 0.34134, 0.43319, 0.47725, 0.49379]


n = len(x_values)

diff_table = np.zeros((n, n))

diff_table[:, 0] = y_values


for j in range(1, n):

    for i in range(n - j):

        diff_table[i][j] = diff_table[i + 1][j - 1] - diff_table[i][j - 1]

x = 1.22

h = 0.5

p = (x - x_values[2]) / h


result = diff_table[2][0]


factorial = 1

p_term = 1

sign = 1


for k in range(1, 5):  # up to 4th order
```

```
    factorial *= k
    if k % 2 == 1:
        term = (diff_table[2 - k//2][k] + diff_table[2 - k//2 + 1][k]) / 2
        p_term *= (p ** k)
    else:
        term = diff_table[2 - k//2][k]
        p_term *= (p ** k)
    result += p_term * term / factorial

print(f"Estimated f(1.22) using Stirling's Interpolation: {result:.6f}")
```

Example: Apply Stirling's and Bessel's formula to find the value of $f(1.22)$ from the following table which gives the values of $f(x)=\frac{1}{\sqrt{2\pi}}\int_0^x e^{-x^2/2}dx$ at intervals of $h=0.5$ from $x=0$ to 2.5.

x	0.0	0.5	1.0	1.5	2.0	2.5
$f(x)$	0.0	0.19146	0.34134	0.43319	0.47725	0.49379

```
import numpy as np
x_values = [0.0, 0.5, 1.0, 1.5, 2.0, 2.5]
y_values = [0.0, 0.19146, 0.34134, 0.43319, 0.47725, 0.49379]

n = len(x_values)
h = 0.5
```

```
diff_table = np.zeros((n, n))

diff_table[:, 0] = y_values


for j in range(1, n):

    for i in range(n - j):

        diff_table[i][j] = diff_table[i + 1][j - 1] - diff_table[i][j - 1]


x_interp = 1.22

p = (x_interp - (x_values[2] + x_values[3]) / 2) / h


y0 = diff_table[2][0]

y1 = diff_table[3][0]

delta_y0 = diff_table[2][1]

delta_y1 = diff_table[3][1]

delta2_y0 = diff_table[2][2]

delta3_y0 = diff_table[2][3]

delta4_y0 = diff_table[2][4]


f_x = (y0 + y1)/2 \

    + p * (delta_y1 - delta_y0)/2 \

    + (p**2) * delta2_y0 / 2 \

    + (p*(p**2 - 1)) * (delta3_y0)/6 \

    + ((p**2) * (p**2 - 1)) * delta4_y0 / 24


print(f"Estimated f(1.22) using Bessel's Interpolation: {f_x:.6f}")
```

Lecture 7:

Example1: Using Lagrange's interpolation formula find $f(4)$ from the following data.

$$f(0) = 2,\ f(1) = 3,\ f(2) = 12,\ f(15) = 3587$$

```
x=[0,1,2,15]

y=[2,3,12,3587]

xv=4

def lagrange_interpolation(x,y,xv):

    n=len(x)

    result=0.0

    for i in range(n):

        term=y[i]

        for j in range(n):

            if i!=j:

                term *= (xv-x[j])/(x[i]-x[j])

        result += term

    return result

estimated_value=lagrange_interpolation(x,y,xv)

print(f"Estimated value of f(4): {estimated_value:.2f}")
```