LAB 2 Interpolation

NUMERICAL COMPUTING

In [ ]:
```
import numpy as np
import sympy as sp
import scipy as sc
from scipy.interpolate import lagrange
import matplotlib.pyplot as plt
plt.style.use('seaborn-poster')
```

Interpolation

Lagrange Polynomial

Lagrange polynomial interpolation finds a single polynomial that goes through all the data points. This polynomial is referred to as a Lagrange polynomial, L(x), and as an interpolation function, it should have the property $L(x_i) = y_i$ for every point in the data set. For computing Lagrange polynomials, it is useful to write them as a linear combination of Lagrange basis polynomials,

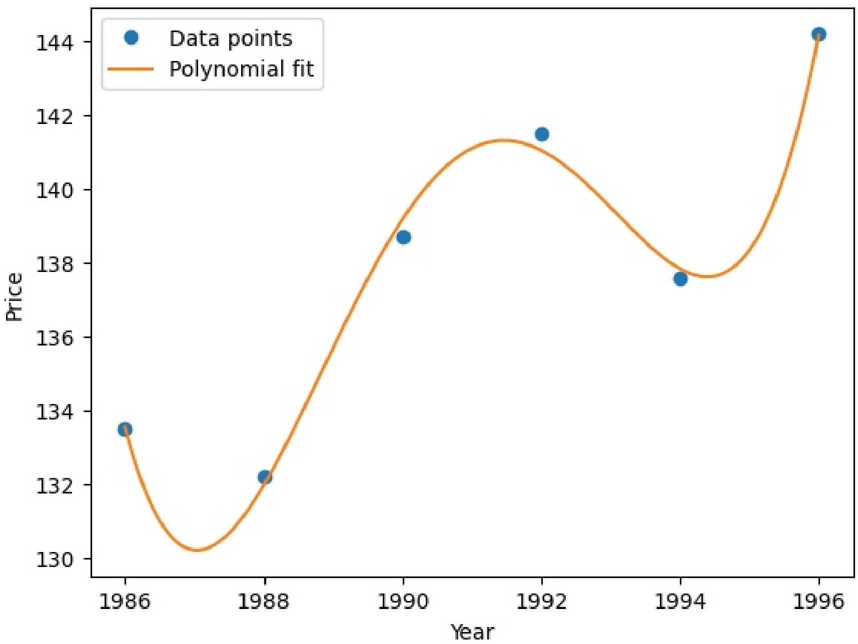
$$P_i(x) = \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j},$$

$$L(x) = \sum_{i=1}^{n} y_i P_i(x)$$

$a_0 + a_1 x + a_2 x^2$

In [13]:
```
%matplotlib inline
```

In [27]:
```
year = np.array([1986, 1988, 1990, 1992, 1994, 1996])
price = np.array([133.5, 132.2, 138.7, 141.5, 137.6, 144.2])

# Fit a polynomial of degree 4 to the data
a = np.polyfit(year, price, 4)

x1 = np.linspace(1986, 1996, 200)
p = np.polyval(a, x1)

plt.plot(year, price, 'o', label='Data points')
plt.plot(x1, p, '-', label='Polynomial fit')
plt.xlabel('Year')
plt.ylabel('Price')
plt.legend()
plt.show()
```



Practice to make a polynmial in numpy.

```
import numpy as np
x=[1,2,3]
np.poly1d(x)
```

In [16]:
```
xx=[1.,-4]
Poly1=np.poly1d(xx,True)
print(Poly1)
```

```
   2
1 x + 3 x - 4
```

In [4]:
```
x=[1,2,3] # here 1,2,3 are coefficients of the polynomial in descending order
Poly1=np.poly1d(x)
print(Poly1)

Poly2=np.poly1d(x,True) #another format to print polynomial
print(Poly2)
```

```
   2
1 x + 2 x + 3
   3     2
1 x - 6 x + 11 x - 6
```

In [5]:
```
Poly1=np.poly1d(a)
print(Poly1)
```

```
         4         3             2
0.03737 x - 297.6 x + 8.889e+05 x - 1.18e+09 x + 5.872e+11
```

In [6]:
```
x=sp.symbols('x')
p = sp.Poly(xx, x)
p
```

Out[6]: $\mathrm{Poly}(1.0x - 4.0, x, domain = \mathrm{R})$

In [7]:
```
p.eval(19)
```

Out[7]: 15.0

Function for getting Lagrange Polynmial

In [8]:
```
# Function to calculate Lagrange polynomial
def lagrange_poly(x, y):
    n = len(x)
    p = np.poly1d(0.0)
    for i in range(n):
        L = np.poly1d(y[i])
        for j in range(n):
            if j != i:
                L *= np.poly1d([1.0, -x[j]]) / (x[i] - x[j])
        p += L
    return p
```

In [9]:
```
L=np.poly1d([1.0, -20]) / (0 -20)
print(L)
```

```
-0.05 x + 1
```

$$\frac{x-20}{0-20} * \frac{x-40}{0-40} * \frac{x-60}{0-60} * \frac{x-80}{0-80}$$

In [145…
```
x = [0, 20,40,60, 80, 100]
y = [26.0, -48.6, 61.6, -71.2, 74.8, -75.2]
p = lagrange_poly(x, y)
print(p)
```

```
            5            4          3         2
-5.329e-06 x + 0.001313 x - 0.1132 x + 3.985 x - 47.81 x + 26
```

For Interpolating at a specific point

In [10]:
```
# Interpolate at a specific point
point = float(input("Enter x-coordinate to interpolate: "))
interp_value = p(point)

# Print Lagrange polynomial and interpolated value
print("Lagrange polynomial is:")
print(p)
print("Interpolated value at x =", point, "is:", interp_value)
```

```
Lagrange polynomial is:
Poly(1.0*x - 4.0, x, domain='RR')
Interpolated value at x = 1.2 is: -2.80000000000000
```

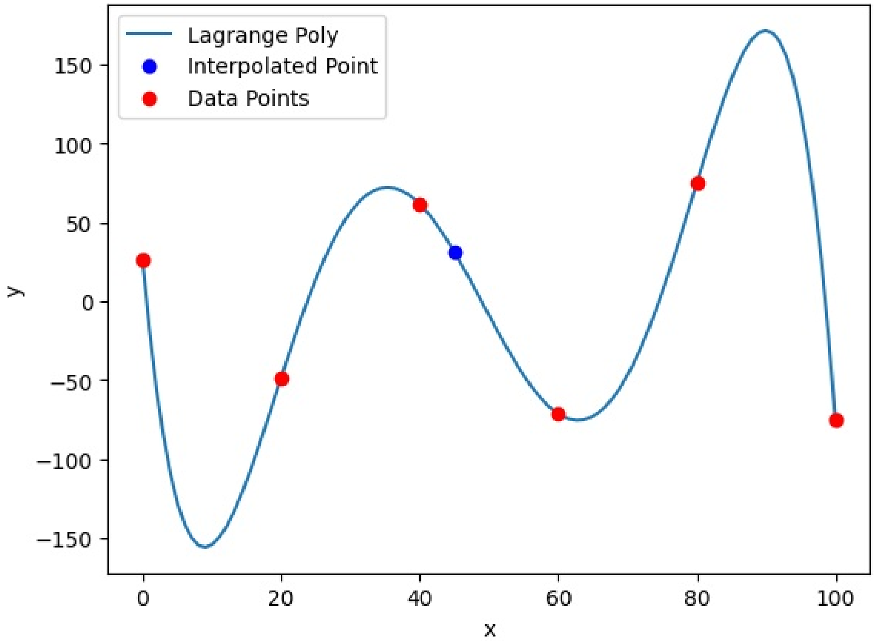
TASK # 01

Solve the above problem manually (Hand written & verify the polynomial & Interpolated value at x = 50 Show all the necessary steps and submitted in the form of PDF along with the project/.ipynb file at GCR

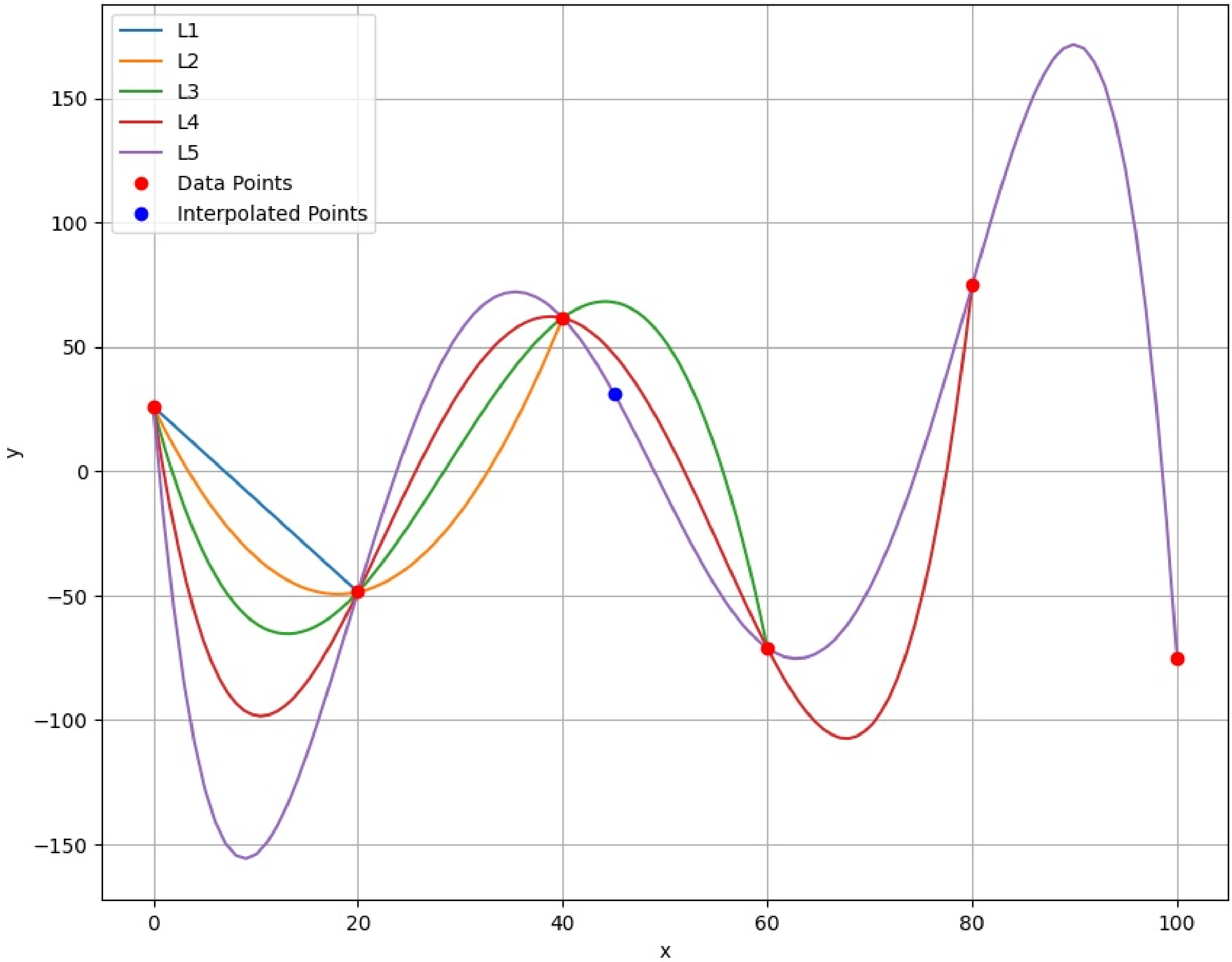
Plotting of Lagrange Polynomial

In [28]:
```
import matplotlib.pyplot as plt
xi=45
yi=31.29079589843832
p = lagrange_poly(x[0:6], y[0:6])
print(p)
xp=np.linspace(0,x[5],100)
yp=p(xp)

plt.plot(xp, yp, label='Lagrange Poly')
plt.plot(xi, yi, 'bo', label='Interpolated Point')
plt.plot(x[0:6], y[0:6], 'ro', label='Data Points')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

```
           5            4          3         2
-5.329e-06 x + 0.001313 x - 0.1132 x + 3.985 x - 47.81 x + 26
```



In [29]:
```
fig = plt.figure(figsize = (10,8))
x = [0, 20,40,60, 80, 100]
y = [26.0, -48.6, 61.6,-71.2, 74.8, -75.2]
n=5
for i in range(1,n+1,1):
  p = lagrange_poly(x[0:i+1], y[0:i+1])
  xp=np.linspace(0,x[i],100)
  yp=p(xp)
  plt.plot(xp, yp, label = f"L{i}")
plt.plot(x,y,'ro',label="Data Points")
plt.plot(xi,yi,'bo',label="Interpolated Points")
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```



In [ ]:

Scipy Implimentation of Lagrange Polynomial

Instead we calculate everything from scratch, in scipy, we can use the lagrange function directly to interpolate the data. Let's see the above example

In [12]:
```
# Define the data points
x = np.array([0, 20, 40, 60, 80, 100])
y = np.array([26.0, -48.6, 61.6, -71.2, 74.8, -75.2])

# Define the Lagrange Polynomial
f = lagrange(x, y)

# Find P(50) by evaluating the polynomial at x=50
p_50 = f(50)
print("P(50) =", p_50)
```

```
P(50) = -8.76015624999853
```

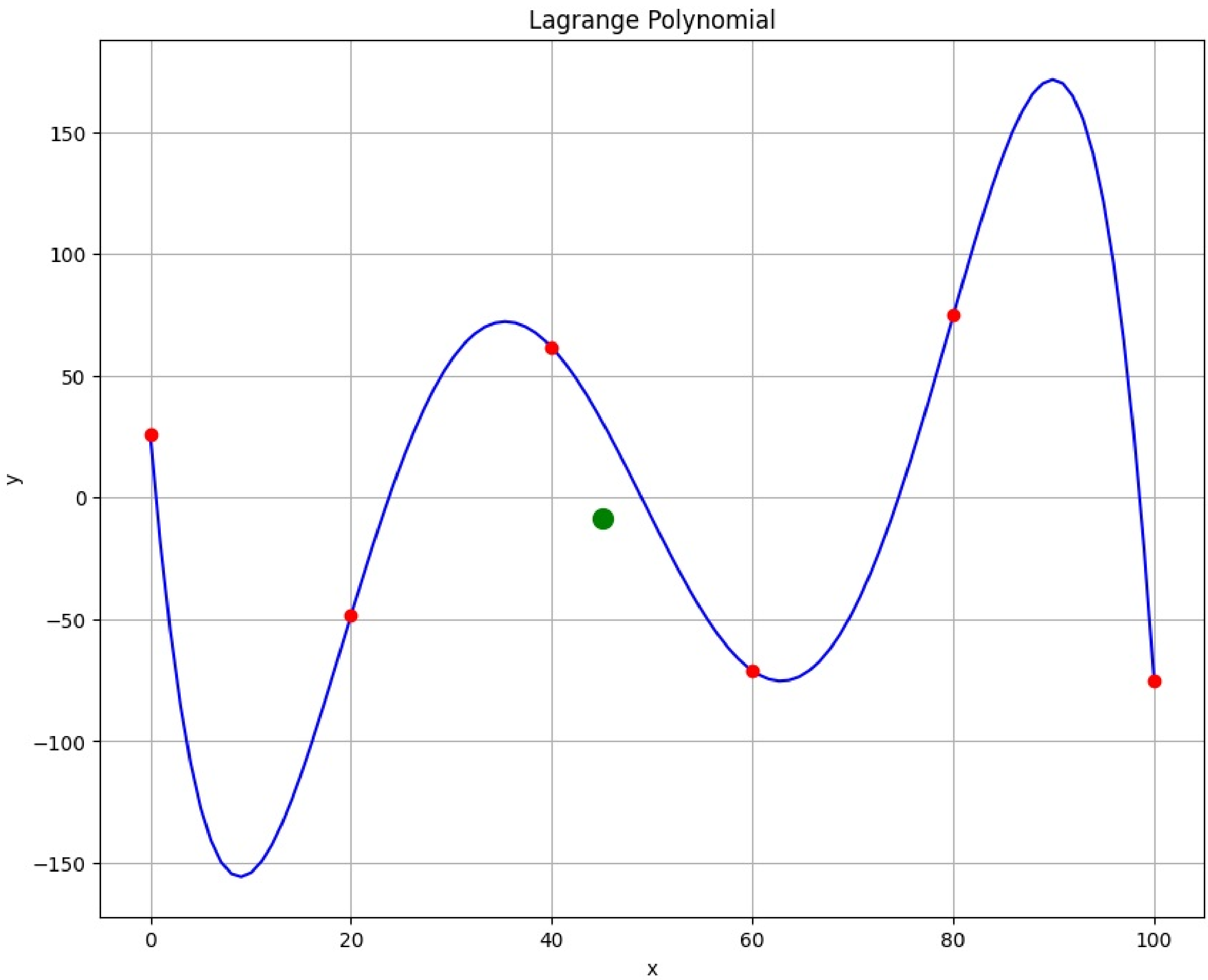
In [37]:
```
print(f)
```

```
              5            4          3         2
-5.329e-06 x + 0.001313 x - 0.1132 x + 3.985 x - 47.81 x + 26
```

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import lagrange

# Define the data points
x = np.array([0, 20, 40, 60, 80, 100])
y = np.array([26.0, -48.6, 61.6, -71.2, 74.8, -75.2])

# Define the Lagrange Polynomial
f = lagrange(x, y)

# Find P(50) by evaluating the polynomial at x=50
p_50 = f(50)
print("P(50) =", p_50)
```

```python
# Plot the Lagrange Polynomial and the data points
x_new = np.linspace(0, 100, 100)
fig = plt.figure(figsize = (10,8))
plt.plot(x_new, f(x_new), 'b', x, y, 'ro')
plt.plot(45, p_45, 'go', markersize=10)
plt.title('Lagrange Polynomial')
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
P(50) = -8.76015624999853
```



Newton divided differences

$$
\begin{array}{cccccc}
x_0 & y_0 & & & & \\
 & & f[x_1,x_0] & & & \\
x_1 & y_1 & & f[x_2,x_1,x_0] & & \\
 & & f[x_2,x_1] & & f[x_3,x_2,x_1,x_0] & \\
x_2 & y_2 & & f[x_3,x_2,x_1] & & f[x_4,x_3,x_2,x_1,x_0] \\
 & & f[x_3,x_2] & & f[x_4,x_3,x_2,x_1] & \\
x_3 & y_3 & & f[x_4,x_3,x_2] & & \\
 & & f[x_4,x_3] & & & \\
x_4 & y_4 & & & &
\end{array}
$$

Each element in the table can be calculated using the two previous elements (to the left). In reality, we can calculate each element and store them into a diagonal matrix, that is the coefficients matrix can be write as:

:

$$
\begin{array}{ccccc}
y_0 & f[x_1,x_0] & f[x_2,x_1,x_0] & f[x_3,x_2,x_1,x_0] & f[x_4,x_3,x_2,x_1,x_0] \\
y_1 & f[x_2,x_1] & f[x_3,x_2,x_1] & f[x_4,x_3,x_2,x_1] & 0 \\
y_2 & f[x_3,x_2] & f[x_4,x_3,x_2] & 0 & 0 \\
y_3 & f[x_4,x_3] & 0 & 0 & 0 \\
y_4 & 0 & 0 & 0 & 0
\end{array}
$$

Task # 02

Use the below code and apply the following alteration and show them along with plot

(i) Take input from user and show interpolation at that point along with its plot.

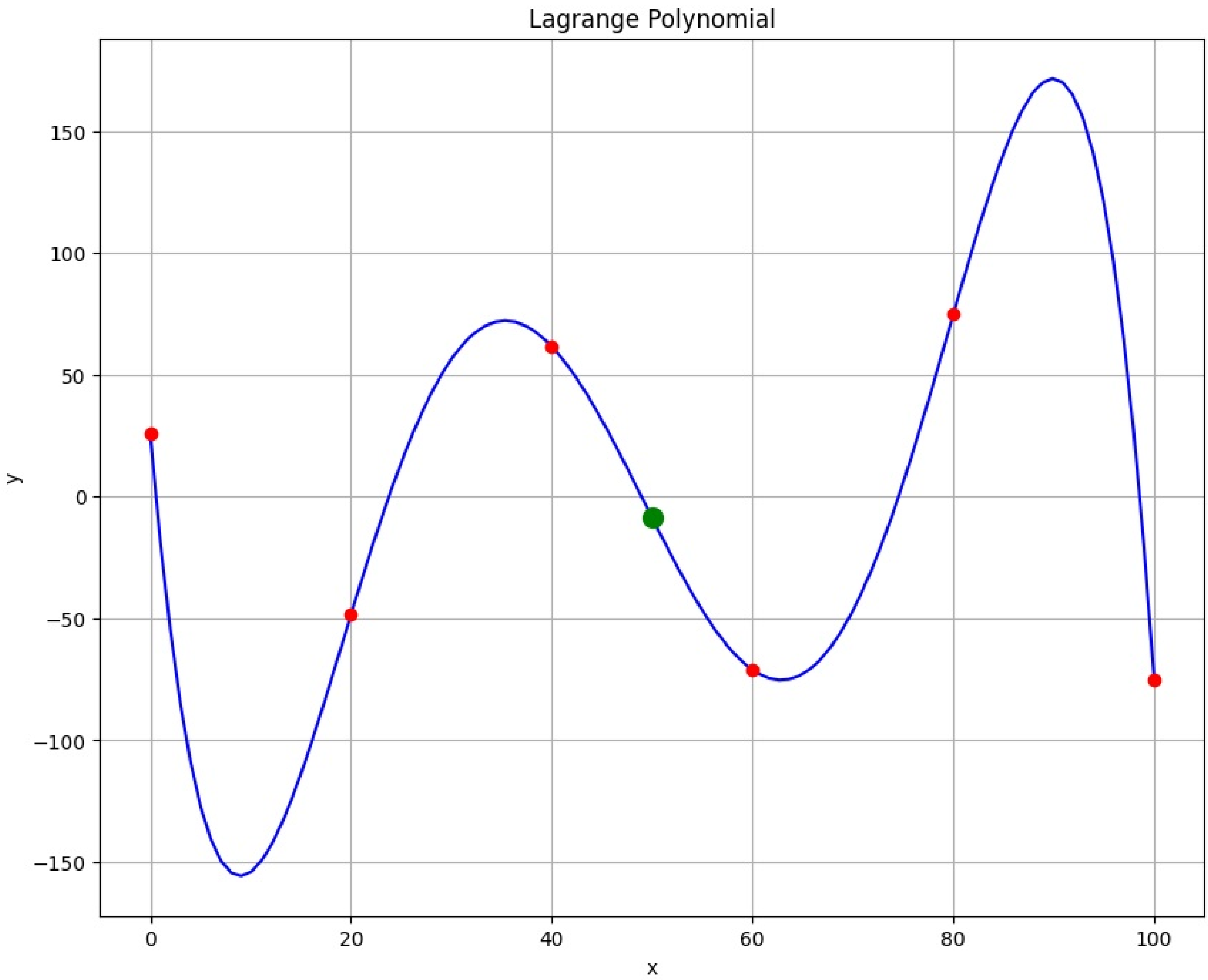
(ii) Also add a code that will display the polynomial too.

Code for Newton divided difference Method

In [27]:

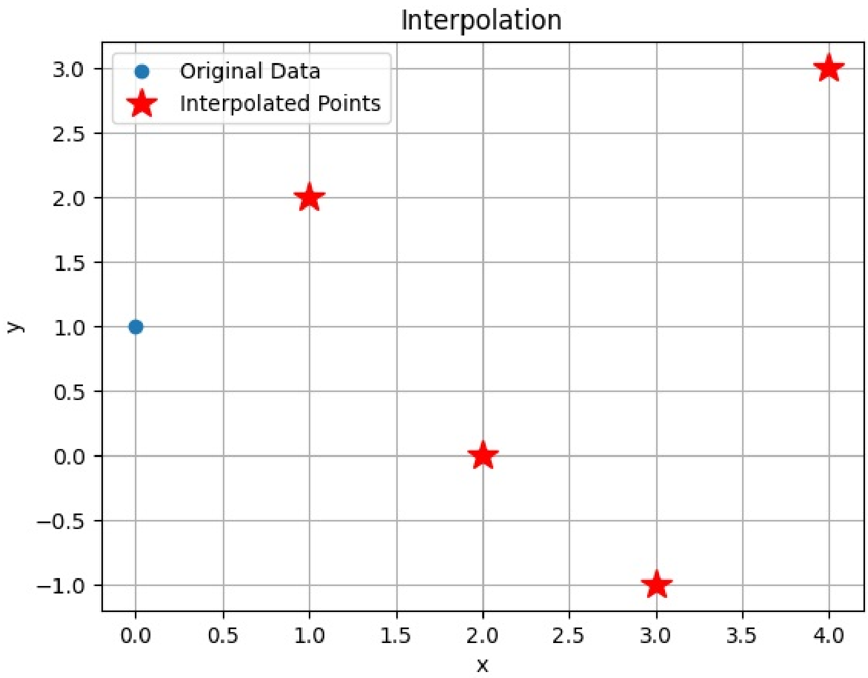
```
#TASK 2 SOLUTION

import numpy as np
import matplotlib.pyplot as plt

def divided_difference_table(x, y):
    n = len(x)
    F = [[0] * n for i in range(n)]
    for i in range(n):
        F[i][0] = y[i]
    for j in range(1, n):
        for i in range(j, n):
            F[i][j] = (F[i][j-1] - F[i-1][j-1]) / (x[i] - x[i-j])
    return F

def newton_div_dif_poly(x,y,xi):
   F=divided_difference_table(x,y) # Saving divided difference in a variable F
   n=len(x)
   prod=np.poly1d(1)
   N=np.poly1d(F[0][0])
   for i in range(1,n):
     prod=np.poly1d(x[0:i],True)
     N+=np.poly1d(F[i][i]*(prod.c))

   return N,N(xi) #returning the polynomial and approximated value


x = [0, 20,40,60, 80, 100]
y = [26.0, -48.6, 61.6, -71.2, 74.8, -75.2]

#taking input from user for point of interpolation
point=float(input("Enter the point of interpolation: "))
polynomial=newton_div_dif_poly(x, y,point)[0]
approx_value=newton_div_dif_poly(x, y,point)[1]

#printing polynomial and value at point of interpolation
print("Polynomial is:")
print(polynomial)
print("Interpolated value= ", approx_value)

#plot
x_new = np.linspace(0, 100, 100)
fig = plt.figure(figsize = (10,8))
plt.plot(x_new, polynomial(x_new), 'b', x, y, 'ro')
plt.plot(point, approx_value, 'go', markersize=10)
plt.title('Lagrange Polynomial')
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
Polynomial is:
            5            4           3         2
-5.329e-06 x + 0.001313 x - 0.1132 x + 3.985 x - 47.81 x + 26
Interpolated value=  -8.760156250001728
```



In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

def interpolate_and_plot(x_data, y_data):
  """
  Performs user-driven interpolation, plots the results, and displays the polynomial.

  - Combines divided differences and np.polyfit for flexibility.
  - Handles single or multiple input for interpolation.
  - Plots original data, interpolated points, and displays polynomial equation.
  """

  # Get input from user for interpolation point(s)
  x_interp_str = input("Enter x-value(s) for interpolation (comma-separated): ")
  x_interp = np.fromstring(x_interp_str, sep=",", dtype=float)

  # Option 1: Use divided differences for single input (optional)
  if len(x_interp) == 1:
    # Calculate divided differences table (optional)
    coef = divided_diff(x_data, y_data)

    # Interpolate using divided differences (optional)
    # y_interp = newton_poly(coef, x_data, x_interp)

  # Option 2: Use np.polyfit for all cases (recommended)
  polynomial = np.poly1d(np.polyfit(x_data, y_data, len(x_data) - 1))
```

```
  y_interp = polynomial(x_interp)

  # Create a string representation of the polynomial
  polynomial_str = str(polynomial)

  # Plot the original data points and the interpolated points
  plt.plot(x_data, y_data, 'o', label='Original Data')
  plt.plot(x_interp, y_interp, 'r*', markersize=15, label='Interpolated Points')
  plt.xlabel('x')
  plt.ylabel('y')
  plt.title('Interpolation')  # Consider a more specific title if using both methods
  plt.legend()
  plt.grid(True)
  plt.show()

  print("Polynomial:", polynomial_str)

# Optional functions for divided differences (uncomment if desired)
def divided_diff(x, y):
  """
  Calculates the divided differences table.

  Args:
      x (np.ndarray): Array of x-values.
      y (np.ndarray): Array of y-values.

  Returns:
      np.ndarray: Divided differences table.
  """

  n = len(y)
  coef = np.zeros([n, n])
  # The first column is y
  coef[:, 0] = y

  for j in range(1, n):
    for i in range(n - j):
      coef[i][j] = (coef[i + 1][j - 1] - coef[i][j - 1]) / (x[i + j] - x[i])

  return coef

def newton_poly(coef, x_data, x):
  """
  Evaluates the Newton polynomial at x.

  Args:
      coef (np.ndarray): Divided differences table.
      x_data (np.ndarray): Array of x-values used for interpolation.
      x (float): Value at which to evaluate the polynomial.

  Returns:
      float: Interpolated value.
  """

  n = len(x_data) - 1
  p = coef[n]
  for k in range(1, n + 1):
    p = coef[n - k] + (x - x_data[n - k]) * p
  return p

# Example usage
x_data = np.array([0, 1, 2, 3])
y_data = np.array([1, 2, 0, -1])
interpolate_and_plot(x_data, y_data)
```




```
Polynomial:          3         2
0.6667 x - 3.5 x + 3.833 x + 1
```

In [152…

```
x = [0, 20,40,60, 80, 100]
y = [26.0, -48.6, 61.6, -71.2, 74.8, -75.2]
a_s = divided_diff(x, y)
```

In [153…

```
a_s
```

Out[153…

```
array([[ 2.60000000e+01, -3.73000000e+00,  2.31000000e-01,
        -8.91250000e-03,  2.47291667e-04, -5.32864583e-06],
       [-4.86000000e+01,  5.51000000e+00, -3.03750000e-01,
         1.08708333e-02, -2.85572917e-04,  0.00000000e+00],
       [ 6.16000000e+01, -6.64000000e+00,  3.48500000e-01,
        -1.19750000e-02,  0.00000000e+00,  0.00000000e+00],
       [-7.12000000e+01,  7.30000000e+00, -3.70000000e-01,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [ 7.48000000e+01, -7.50000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [-7.52000000e+01,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00]])
```

In [155…

```
x = [0, 20,40,60, 80, 100]
y = [26.0, -48.6, 61.6, -71.2, 74.8, -75.2]
a_s = divided_diff(x, y)[0, :]
a_s
```

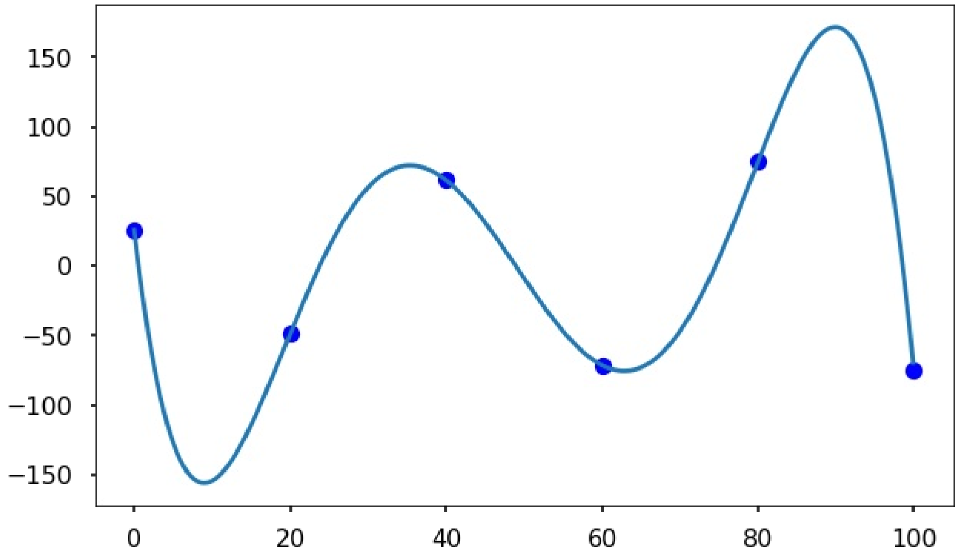
Out[155…

```
array([ 2.60000000e+01, -3.73000000e+00,  2.31000000e-01, -8.91250000e-03,
        2.47291667e-04, -5.32864583e-06])
```

In [159…

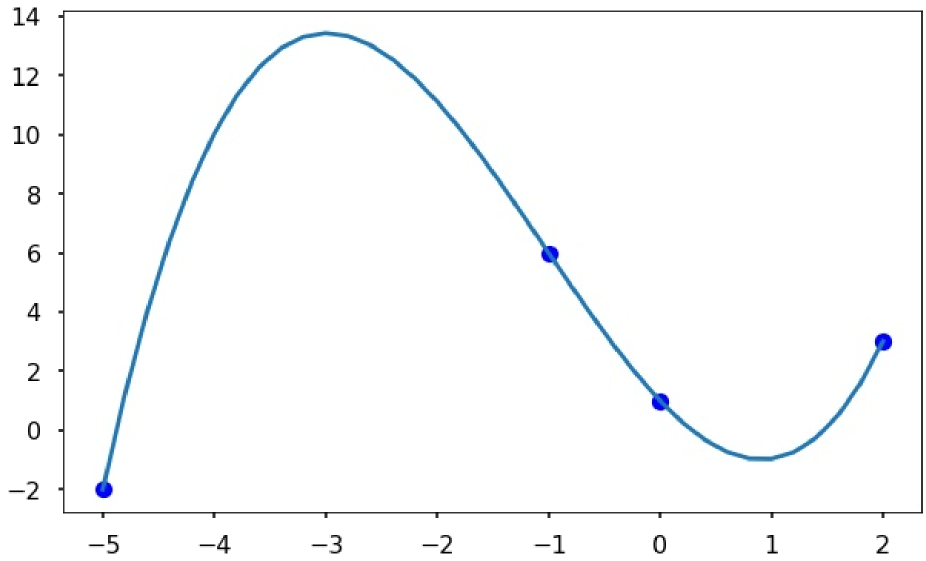
```
# evaluate on new data points
x_new = np.arange(0, 100, .1)
y_new = newton_poly(a_s, x, x_new)
fig = plt.figure(figsize = (10,6))

plt.plot(x, y, 'bo')
plt.plot(x_new, y_new)
```

Out[159… [<matplotlib.lines.Line2D at 0x237013ef248>]



In [166…
```
x = np.array([-5, -1, 0, 2])
y = np.array([-2, 6, 1, 3])
# get the divided difference coef
a_s = divided_diff(x, y)[0, :]

# evaluate on new data points
x_new = np.arange(-5, 2.1, 0.2)
y_new = newton_poly(a_s, x, x_new)

plt.figure(figsize = (10, 6))
plt.plot(x, y, 'bo')
plt.plot(x_new, y_new)
```

Out[166… [<matplotlib.lines.Line2D at 0x237013343c8>]



Task # 03 (A)

Use the above code by adding divided difference table code i.e. it will show divided difference table.

Task # 03 (B)

With the help of **"pandas"** as shown at the starting of this lab session, read code from provided csv file & Write a code for Newton's forward divided difference. Print the polynomial and plot the interpolating point too.

Task # 03 (C)

Do part 3(B) manually (Mentioned all steps and verify the result.

In [19]:
```
#TASK 3A SOLUTION
```

```python
import numpy as np
from tabulate import tabulate

def divided_difference_table(x, y):
    n = len(x)
    F = [['-----'] * n for i in range(n)]
    for i in range(n):
        F[i][0] = y[i]
    for j in range(1, n):
        for i in range(j, n):
            F[i][j] = (F[i][j-1] - F[i-1][j-1]) / (x[i] - x[i-j])
    return F

def newton_div_dif_poly(x,y,xi):
   F=divided_difference_table(x,y) # Saving divided difference in a variable F
   n=len(x)

   prod=np.poly1d(1)
   N=np.poly1d(F[0][0])
   for i in range(1,n):
     prod=np.poly1d(x[0:i],True)
     N+=np.poly1d(F[i][i]*(prod.c))
   print("Polynomial is:")
   print(N)
   print("Interpolated value= ",N(xi))

   return


#SOLUTION TASK 3A
def print_divided_difference_table(x,y):
  F=divided_difference_table(x,y) # Saving divided difference in a variable F
  n=len(x)
  #initialising table
  table=[[i] * (n+2) for i in range(n)] #0th column with i
  for i in range(0,n):                  #1st column with x
   table[i][1] = x[i]
  #filling rest of the spaces with F(f(x) and divided differences)
  for j in range(0,n):
   for i in range(0,n):
    table[i][j+2]=F[i][j]
  #printing table
  print(tabulate(table,headers=['i','xi','f[xi]','f[xi-1,xi]','f[xi-2,xi-1,xi]','f[xi-3,...,xi]','f[xi-4,...,xi

  return



x = [0,40,80, 120, 160]
y = [26.0, -48.6, 61.6, -71.2, 74.8, -75.2]
newton_div_dif_poly(x, y,45)
print_divided_difference_table(x,y)
```

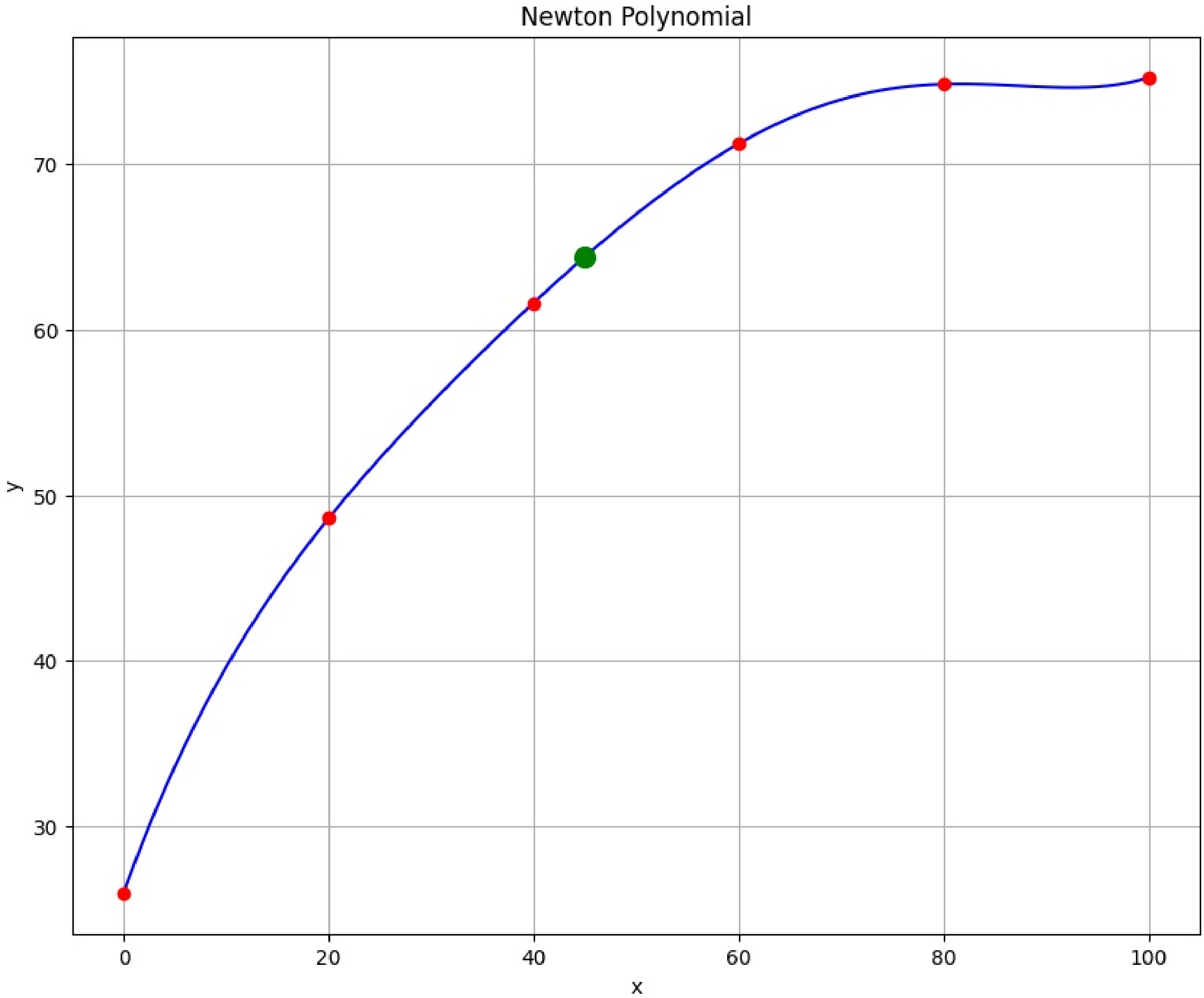
```
Polynomial is:
           4            3           2
1.546e-05 x - 0.004823 x + 0.4635 x - 13.67 x + 26
Interpolated value=  -27.029467773437347
|   i |   xi |   f[xi] | f[xi-1,xi]          | f[xi-2,xi-1,xi]     | f[xi-3,...,xi]         | f[xi-4,...,xi]
|
|-----|------|---------|---------------------|---------------------|------------------------|-------------------
-----|
|   0 |    0 |    26   | -----               | -----               | -----                  | -----
|
|   1 |   40 |   -48.6 | -1.8649999999999998 | -----               | -----                  | -----
|
|   2 |   80 |    61.6 | 2.755               | 0.05774999999999999 | -----                  | -----
|
|   3 |  120 |   -71.2 | -3.3200000000000003 | -0.0759375          | -0.0011140624999999998 | -----
|
|   4 |  160 |    74.8 | 3.65                | 0.08712500000000001 | 0.0013588541666666666  | 1.5455729166666666
e-05 |
```

In [28]:

```python
#TASK 3B SOLUTION

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Read data from CSV file
df = pd.read_csv('interpolation.csv')

# Convert data to numpy arrays
x = df['x'].values
```

```
y = df['y'].values
print("x=",x)
print("y=",y)

# CODE FOR FORWARD DIVIDED DIFFERENCE
def forward_divided_difference_table(x, y):
    n = len(x)
    F = [[0] * n for i in range(n)]
    for i in range(n):
        F[i][0] = y[i]
    for j in range(1, n):
        for i in range(j, n):
             F[i][j] = (F[i][j-1] - F[i-1][j-1]) / (x[i] - x[i-j])
    return F

def forward_newton_div_dif_poly(x,y,xi):
   F=forward_divided_difference_table(x,y) # Saving divided difference in a variable F
   n=len(x)
   h=x[1]-x[0]
   s=np.poly1d([1/h,x[0]/h])
   prod=np.poly1d(1)
   N=np.poly1d(F[0][0])
   for i in range(1,n):
     prod*=(s-(i-1))
     N+=np.poly1d(F[i][i]*(prod.c)*(h**i))

   return N,N(xi)


#function call and saving return values
polynomial=forward_newton_div_dif_poly(x, y,45)[0]
approx_value=forward_newton_div_dif_poly(x, y,45)[1]

#printing polynomial and value at point of interpolation
print("Polynomial is:")
print(polynomial)
print("Interpolated value= ", approx_value)

#plot
x_new = np.linspace(0, 100, 100)
fig = plt.figure(figsize = (10,8))
plt.plot(x_new, polynomial(x_new), 'b', x, y, 'ro')
plt.plot(45, approx_value, 'go', markersize=10)
plt.title('Newton Polynomial')
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
x= [  0  20  40  60  80 100]
y= [26.  48.6 61.6 71.2 74.8 75.2]
Polynomial is:
           5              4              3            2
3.698e-08 x - 9.688e-06 x + 0.0009219 x - 0.04463 x + 1.725 x + 26
Interpolated value=  64.3791259765625
```



Code to read data from a CSV file

```
import pandas as pd
import numpy as np

# Read data from CSV file
df = pd.read_csv('data.csv')

# Convert data to numpy arrays
x = df['x values'].values
y = df['y values'].values
```

In [9]:
```python
f = open('interpolation.csv', "r")
print(f.read())
```

```
x,y
0,26
20,48.6
40,61.6
60,71.2
80,74.8
100,75.2
```

In [58]:
```python
f = open('interpolation.csv', "r")

XY=f.readlines()
x=[]
y=[]
for i in XY:
    if 'x,y' in i:
        continue
    ii=i.split(',')
    x.append(float(ii[0]))
    y.append(float(ii[1]))
print("X values:", x)
print("Y values:", y)
```

```
X values: [0.0, 20.0, 40.0, 60.0, 80.0, 100.0]
Y values: [26.0, 48.6, 61.6, 71.2, 74.8, 75.2]
```

In [167…
```python
XY
```

Out[167…
```
['x,y\n',
 '0,26\n',
 '20,48.6\n',
 '40,61.6\n',
 '60,71.2\n',
 '80,74.8\n',
 '100,75.2\n']
```

In [120…
```python
import csv

x = []
y = []

with open('interpolation.csv', 'r') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        if 'x' in row[0]:
            continue
        x.append(float(row[0]))
        y.append(float(row[1]))

print("X values:", x)
print("Y values:", y)
```

```
X values: [0.0, 20.0, 40.0, 60.0, 80.0, 100.0]
Y values: [26.0, 48.6, 61.6, 71.2, 74.8, 75.2]
```

In [ ]:
```python
y
```