311 – Numerical Computations
Lab 10: Regression/Interpolation

A)Polynomial Representation in Numpy

```
p1 = np.poly1d([7, 3,0])
p2 = np.poly1d([5, 0,0,-9, 2, 10])

print ("Polynomial 1:\n", p1)
print ("Polynomial 2:\n", p2)



print ("Evaluation:" , p1(3), p2(1))

print ("Coefficients of P2: ", p2.coeffs)
```

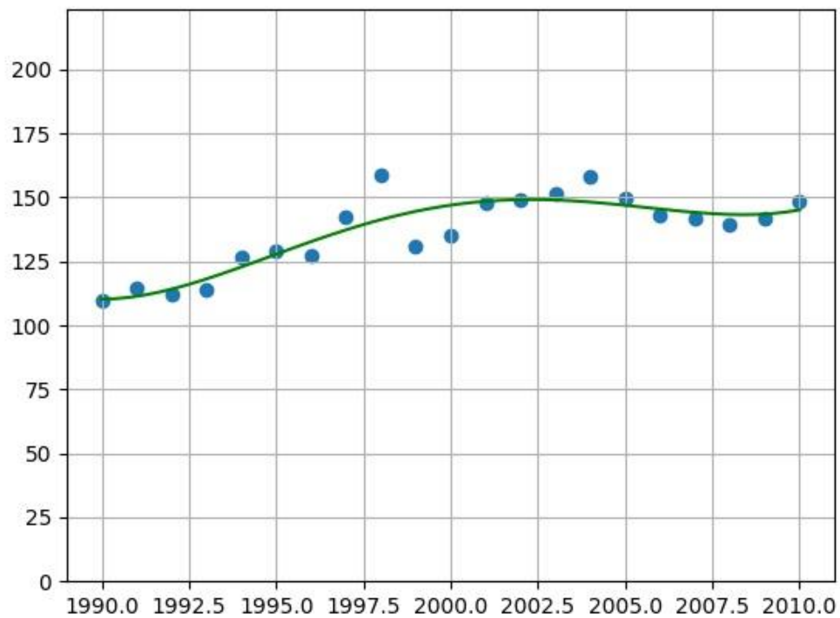
Output:

```
Polynomial 1:

   2
7 x + 3 x
Polynomial 2:

   5     2
5 x -  9 x + 2 x + 10
Evaluation: 72 8

Coefficients of P2:  [ 5  0  0 -9  2 10]
```

B) Polynomial Regression

```
import numpy as np
import matplotlib.pyplot as plt

X =    [1990,1991,1992,1993,1994,1995,1996,1997,
        1998,1999,2000,2001,2002,2003,2004,2005,
        2006,2007,2008,2009,2010]
Y    = [109.43,114.32,112.15,113.9, 126.83,128.81,127.23,142.32,
        158.67,130.79,135.36,147.72, 149.29,151.52,158.41,149.63,
        142.78,141.91,139.31,142.05,148.33]

poly_coeff = np.polyfit(X, Y, 4)
p= np.poly1d(poly_coeff)
print("Estimation example:", p(2020))
# For the chart:
x = np.linspace(X[0], X[len(X)-1])
pdraw = p(x)

plt.scatter(X,Y)

plt.plot(x, pdraw ,color="green")

plt.grid( )
plt.ylim(0,myPoly.max()*1.5 )
plt.show( )
```

(See Output Figure on Next Page)



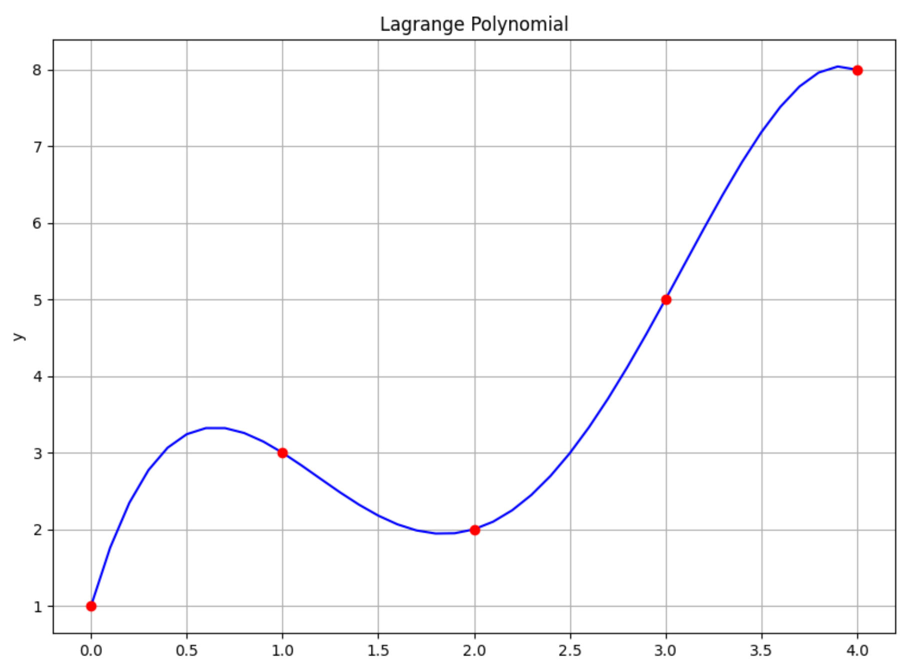
Task1: Apply Regression (of degree 4) on the data in the attached File. Let the program predict the value of y at x=2007, 2012 and 2018.

Also let the program draws a chart like the above chart.

C) Interpolation:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate

x = [0, 1, 2, 3, 4]
y = [1, 3, 2, 5 ,8]

f = scipy.interpolate.lagrange(x, y)
print(f(1.5))
print(f(7))                    #!!!!!!!!!!!
# for the chart
x_new = np.linspace(0, 4, 40)
#equivalently: x_new = np.arange(0, 4.1, 0.1)
plt.plot(x_new, f(x_new), 'b', x, y, 'ro')
plt.title('Lagrange Polynomial')
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
plt.show( )
```

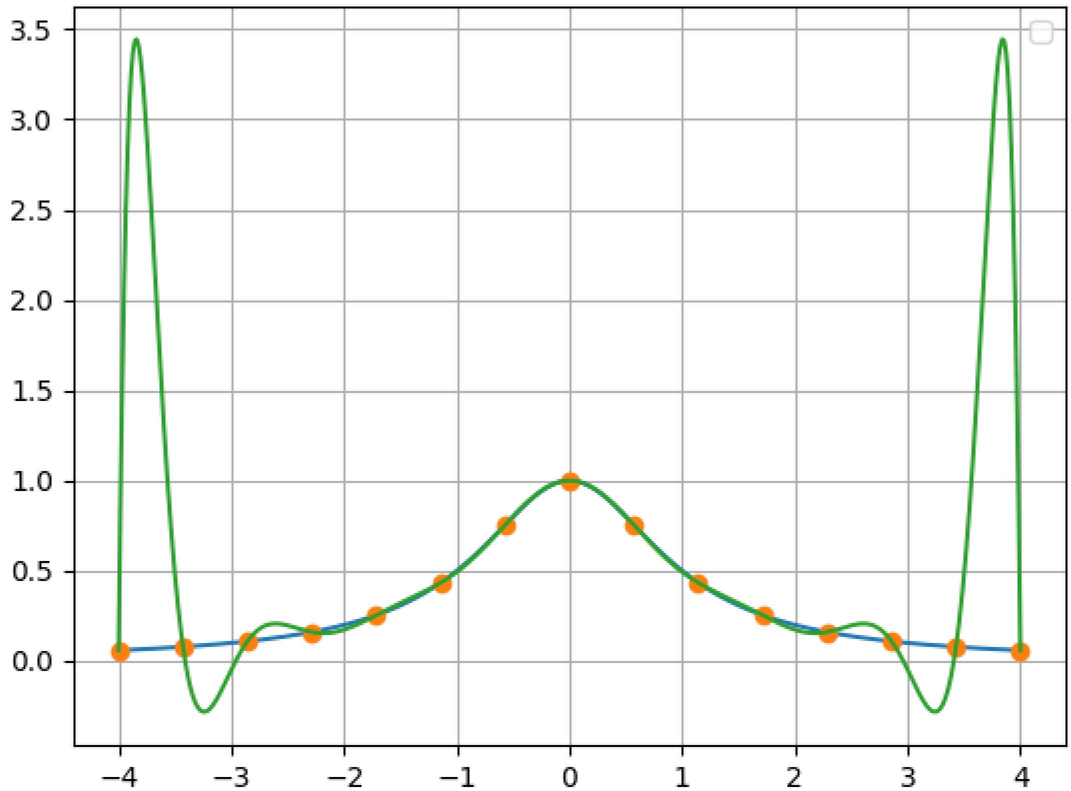
Output:



D) Oscillation at the edges (Runge's Phenomenon):

Runge's phenomenon is a problem of oscillation at the **edges** of an interval that occurs when using polynomial interpolation with polynomials of **high degree** over a set of equispaced interpolation points.

Output: # Python Code is at next page:



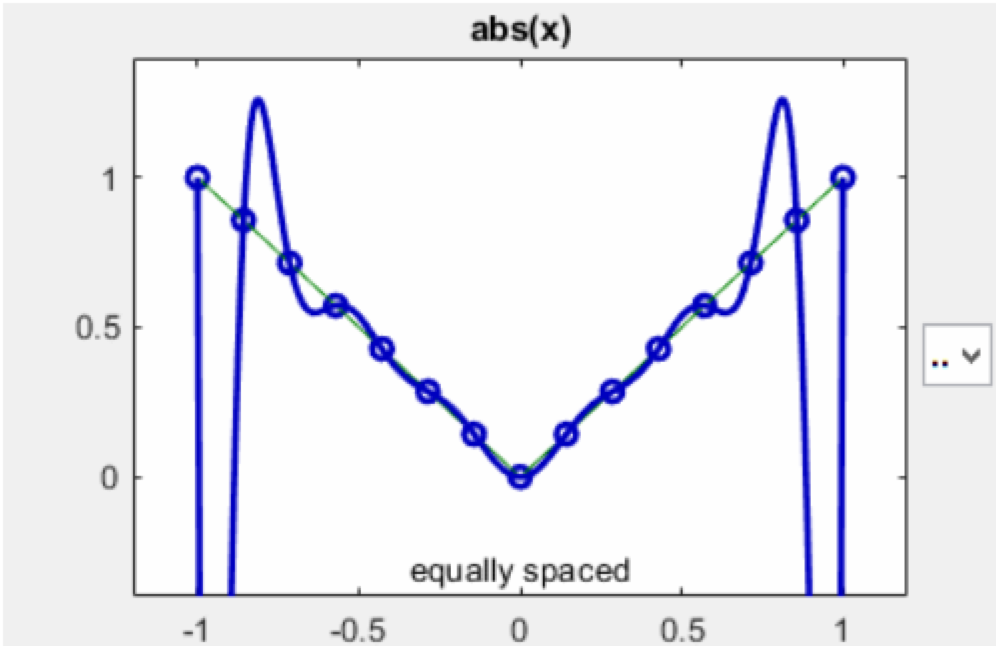
In such case: your estimate point should be near the center.

In case you are adding more points (to obtain more accuracy for the estimation of a certain point), add half of these points to the left of your point and the other half to the right.

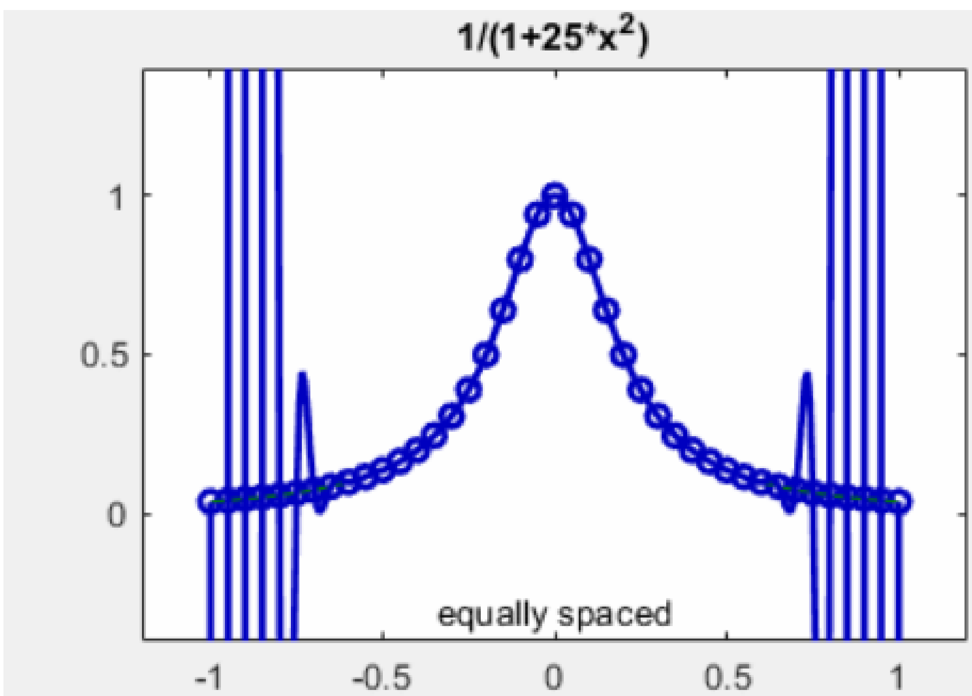
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate

def f(x):
   return 1/(1 + x**2)      #in most sources, they use
                            #   f(x) = 1 / (1 + 25*x**2)

x_data = np.linspace(-4,4,15)
y_data = f(x_data)
p = scipy.interpolate.lagrange(x_data,y_data)
xs = np.linspace(-4,4,500)
ys = f(xs)
plt.plot(xs,ys)
plt.plot(x_data,y_data,'o')
ys = p(xs)
plt.plot(xs,ys)
plt.grid()
plt.legend()
plt.show()
```

Task2: Write a program that draws a chart similar to the abs function (on the next page).

More Examples:



15 point



41 point

These two photos are from:

https://blogs.mathworks.com/cleve/2018/12/10/explore-runges-polynomial-interpolation-phenomenon/