# Project Report

## Numeric Computing – 107269

# THE LENGTH OF A CURVE

## INTRODUCTION:

In this experiment, we will find the length of two curves generated from the same points – one curve is a polynomial interpolant and another one is a spline interpolant.

**Motivation behind the experiment**: In 1901, Runge conducted a numerical experiment to show that higher order interpolation is a bad idea. It was shown that as you use higher order interpolants to approximate $f(x)=1/(1+25x^2)$ in [-1,1], the differences between the original function and the interpolants becomes worse. This concept also becomes the basis why we use splines rather than polynomial interpolation to find smooth paths to travel through several discrete points.

## METHOD:

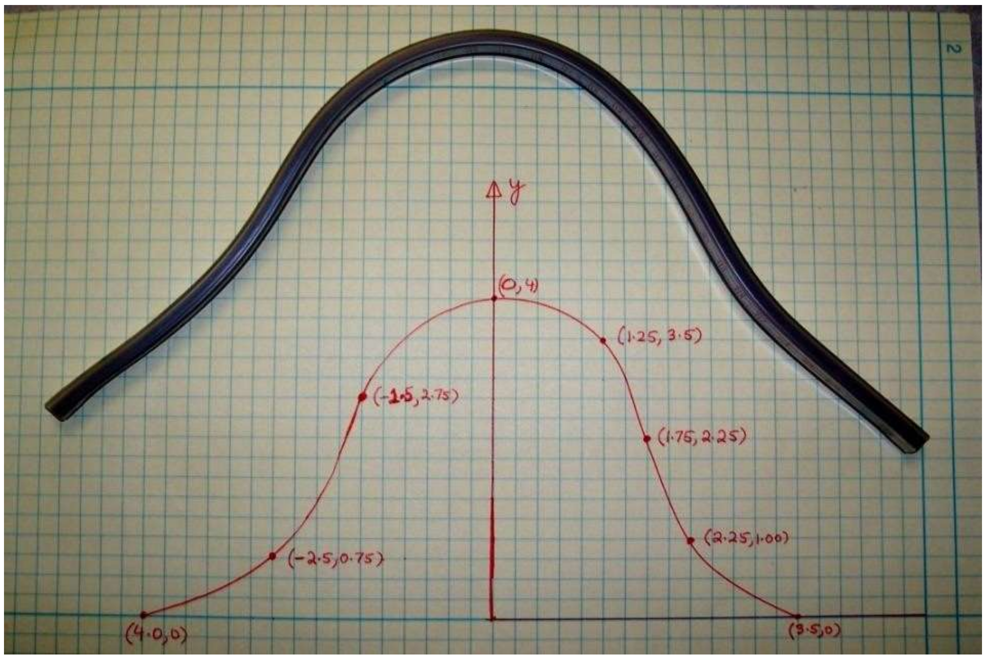
**What do we do:** A flexible curve (see Figure) made of lead-core construction with graduations in both millimeters and inches is provided? The student needs to draw a curve similar in shape to the Runge's curve on the provided graphing paper as shown. It just needs to be similar in shape – the student can make the *x*-domain shorter and the maximum *y*-value larger or vice-versa. The student just needs to make sure that there is a one-to-one correspondence of values.

## Tools using: Python

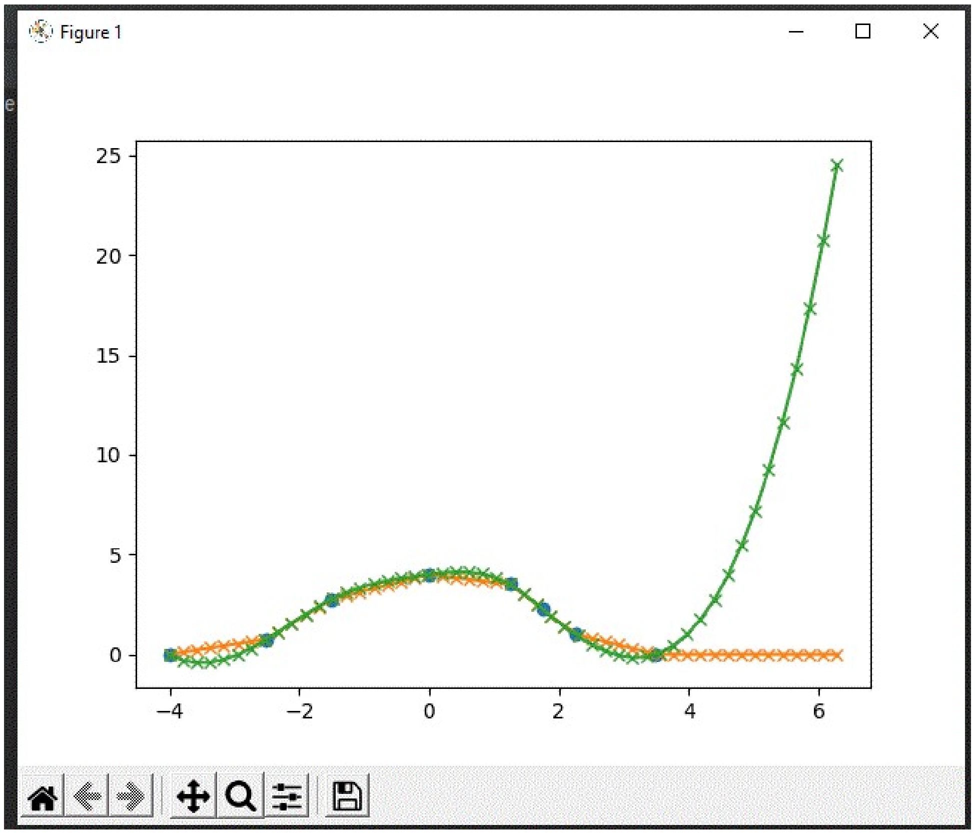
## Project Code:

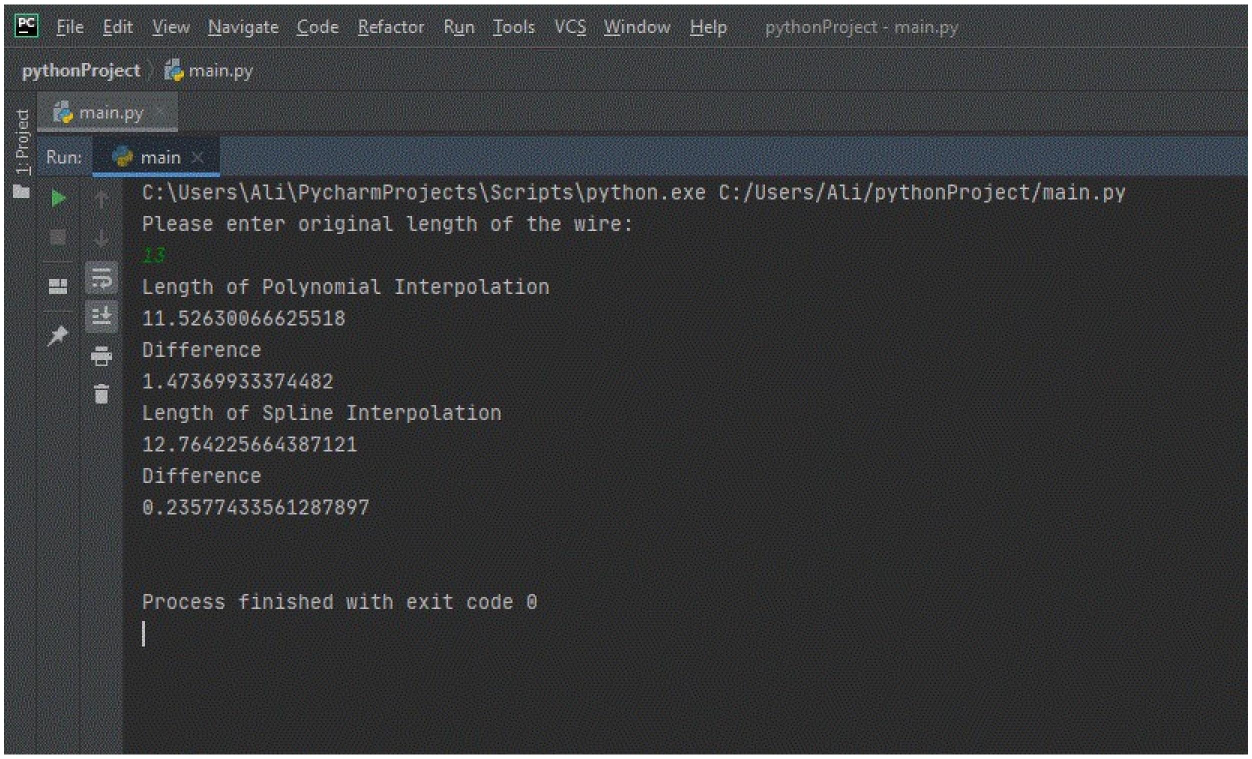
```python
#Created By:
#MUHAMMAD UMAR KHAN 10619
#AWAB KHAN 8116
#SHAIKH RAMEEZ AZHAR 10394

import numpy as np
import matplotlib.pyplot as plt
x_pts = [-4.0, -2.5, -1.5, 0, 1.25, 1.75, 2.25, 3.5]
y_pts = [0, 0.75, 2.75, 4, 3.5, 2.25, 1.00, 0]
print("Please enter original length of the wire:")
lent = float(input())
x_vals = np.linspace(-4, 2 * np.pi, 50)
y_vals = np.interp(x_vals, x_pts, y_pts)
plt.plot(x_pts, y_pts, 'o')
plt.plot(x_vals, y_vals, '-x')
plt.show()

import numpy as np
from scipy import interpolate

splines = interpolate.splrep(x_pts, y_pts)
x_vals1 = np.linspace(-4, 2 * np.pi, 50)
y_vals1 = interpolate.splev(x_vals, splines)
#-------------------------------------------------------------------------------
def length(x, y, x1, l):
    a = []
    for i in range(49):
        if (x[i] >= x1[7]):
            break
        else:
            t = (x[i + 1] - x[i]) * (x[i + 1] - x[i])
            g = (y[i + 1] - y[i]) * (y[i + 1] - y[i])
            a.append(np.sqrt(t + g))

    result = 0
    for i in range(len(a)):
        result = result + a[i]
    print(result)
    print("Difference")
    print(l - result)
#------------------------------------------------------------------------------

plt.plot(x_pts, y_pts, 'o', x_vals, y_vals, '-x', x_vals1, y_vals1, '-x')
plt.show()
print("Length of Polynomial Interpolation")
length(x_vals, y_vals, x_pts, lent)
print("Length of Spline Interpolation")
length(x_vals1, y_vals1, x_pts, lent)
print()
```

## Actual Graph:



## SNAPSHOT OF OUTPUT:





```
C:\Users\Ali\PycharmProjects\Scripts\python.exe C:/Users/Ali/pythonProject/main.py
Please enter original length of the wire:
13
Length of Polynomial Interpolation
11.52630066625518
Difference
1.47369933374482
Length of Spline Interpolation
12.764225664387121
Difference
0.23577433561287897


Process finished with exit code 0
```

## RESULT:

The final result revealed that spline interpolation is the best method and it gives out smooth curves as compared to polynomial interpolation. In polynomial interpolation we had an equation with points to plot the graph which proved to give a rough curve and spline interpolation gave a perfect curve with a value as close to the length of the wire given as an input. The error calculated through difference was very less in spline interpolation and in polynomial interpolation the error was almost double therefore spline interpolation proved to be the best experiment plot.

## CONCLUSION:

Lastly it is observed that if we want to go through some smooth discrete points, the spline interpolation is the best way to do it and polynomial interpolation is a bad and an old idea.