Assignment-0

Shivanshu Shekhar

August 23, 2023

1 Assignments

1. Revise your Python skills, especially scientific python (numpy, scipy & pylab)

2. We wish to interpolate $f(x) = \sin(x)$ from a table of function values sampled at $0, \pi/4, \pi/2, 3\pi/4, \pi, 5\pi/4, 3\pi/2, 7\pi/4, 2\pi$. Write a Lagrange Interpolation function in C with the following calling sequence that will perform an n^{th} order interpolation at the point $x.xx$ and yy are the table of values that are interpolated over. They are of dimension n. Also, create a main program in C that will allow the C code to run independently of Python. Use $n = 8$ and generate the interpolated values for this problem.

```
float lintp(float *xx,float *yy,float x,int n)
```

3. Run the program and evaluate the function on 100 uniformly spaced values between 0 and 2π Write out the answers to a text file *output.txt* with each line containing x y.

4. Write a python script that uses system via

   ‘
```
import os
os.system("...")
```

   to run the C executable. It should then use np.loadtxt to read in *output.txt* and plot both the interpolated values as well as the exact function $\sin(x)$ in a plot. Another plot should plot the error between the interpolated and exact function vs x.

5. Repeat the above with $f(x) = \sin(x) + n$ where n is normally distributed noise with a variance σ^2 Use **randn(N)*sigma** to generate N such random numbers. Compare the quality of the interpolation as the amount of noise increases. Can you explain what you see?

2 Lagrange Interpolation

Lagrange interpolation is a mathematical method used to approximate a function based on a set of data points. It's a popular technique in numerical analysis and computational mathematics.

The primary goal of Lagrange interpolation is to construct a polynomial function that passes through a given set of data points. If you have a set of $n+1$ data points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, where each x_i is a distinct x-coordinate and y_i is the corresponding y-coordinate, Lagrange interpolation seeks to find a polynomial $P(x)$ of degree at most n that satisfies $P(x_i) = y_i$ for each i from 0 to n.

The Lagrange interpolation polynomial can be defined as:

$$P(x) = \sum_{i=0}^{n} y_i \cdot L_i(x)$$

Where $L_i(x)$ is the ith Lagrange basis polynomial, defined as:

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

These basis polynomials have a useful property: $L_i(x_j) = \delta_{ij}$, where δ_{ij} is the Kronecker delta, which is 1 if $i = j$ and 0 otherwise. This property ensures that the Lagrange polynomial passes through the given data points, as desired.

We implemented the this logic in C as shown below:

```
float lintp(float *xx, float *yy, float x, int n){
    double ret = 0;

    for(int i=0;i<=n;i++){
        double L = 1;
        for(int j=0;j<=n;j++) if(i!=j) L*= (x-xx[j])/(xx[i]-xx[j]);

        ret += L * yy[i];
    }

    return ret;
}
```

We use these lines of code to interpolate $f(x) = \sin(x)$. Since we are given the 9 sampled values we perform 8^{th} order interpolation.

We calculate the interpolated values at 100 equally spaced points and write them to *output.txt* in the same format as specified by the question using the following lines of code:

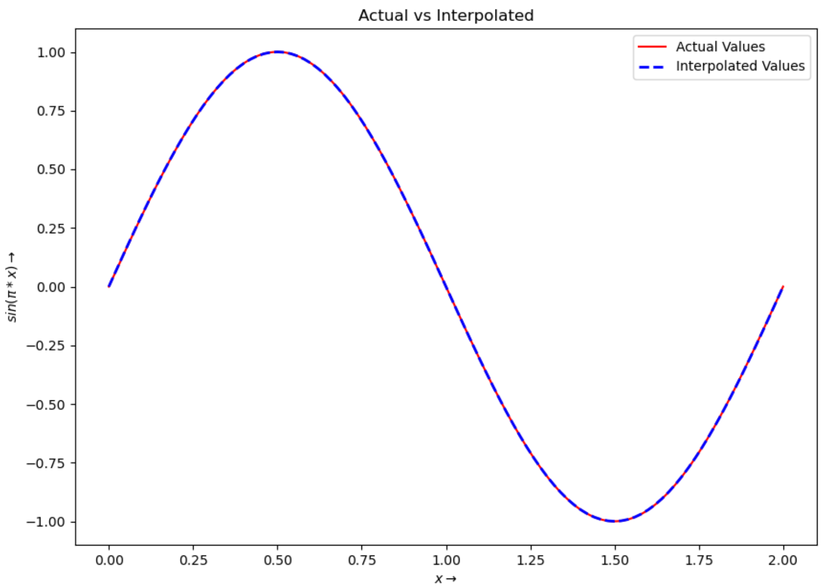
```
for(int i=0;i<=100;i++){
    if(flag == 0) y = lintp(xx, yy, x, 8);

    else y = lintp(xx, yyNoise[flag-1], x, 8);
```

```
    fprintf(output, "%f %f\n", x, y);

    x += 0.02; // 100 uniform samples in (0, 2)
}
```

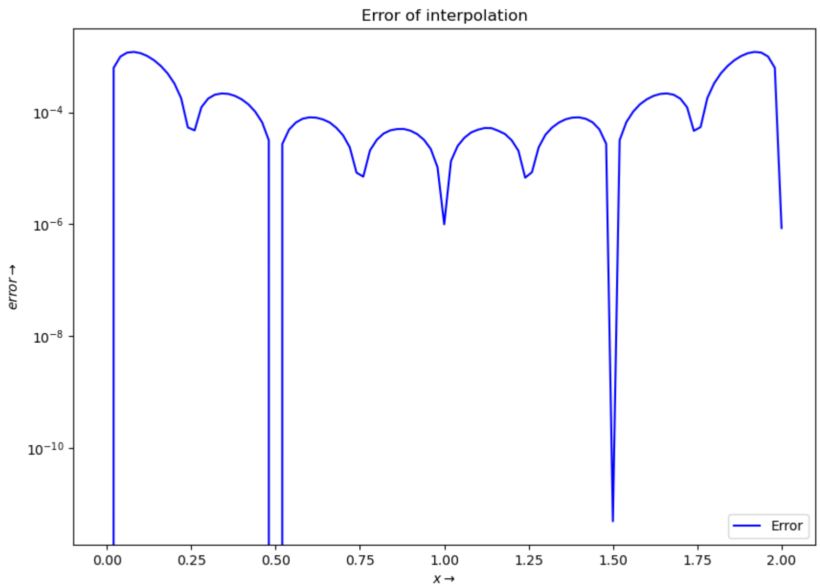
We read the *output.txt* in python using the np.loadtxt function and get the generated values to conduct further analysis in python. We also have the main calling script in python, we run the C code using os.system command. Below is the function in python that we used to call the C code.

```
def runC(name: str, flag: int):
    os.system('gcc -o ' + name + ".out " + name + '.c' + ' -lm')
    os.system('./' + name + '.out' + ' tmp.txt ' + str(flag))

    data = np.loadtxt('output.txt')
    return data[:, 0], data[:, 1]
```

To compare the interpolated points with the actual values, we will plot them in the same graph as shown in Figure: 1.



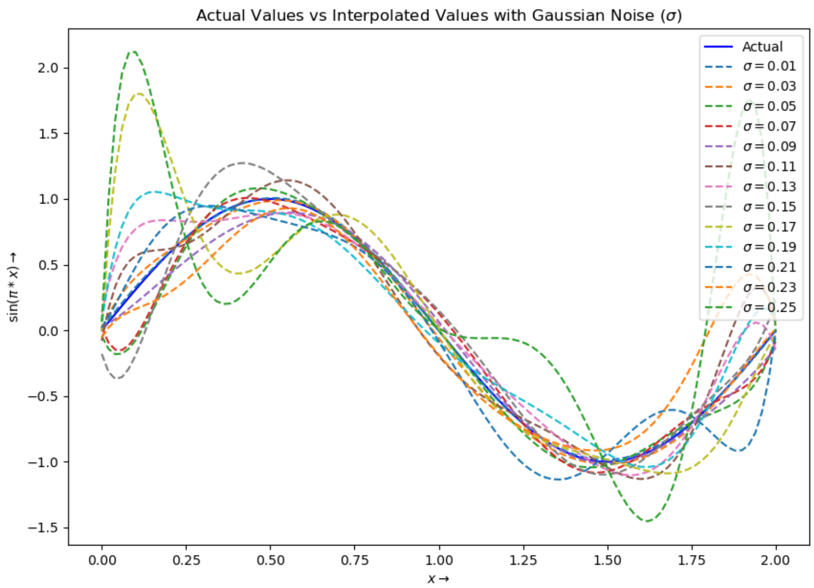
To compare the error in the interpolation we also plot the error between the actual and interpolated values in a semilog-Y plot, as shown in Figure: 2 we can see that the absolute error in the interpolation is very small.



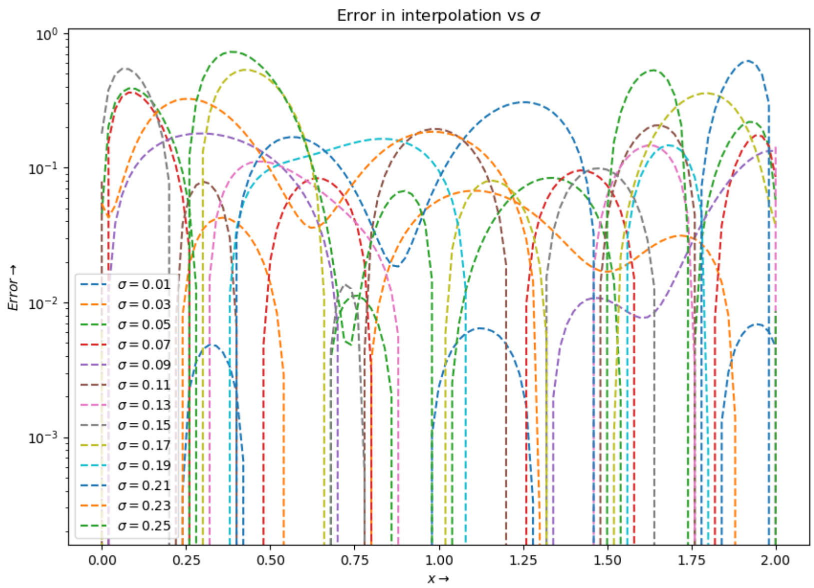
3 Adding Gaussian Noise

We repeat the above experiments with the following function $f(x) = \sin(x) + n$ where n is the Gaussian noise. The noise is generated using np.randn(N)*sigma), and $\sigma \in (0.01, 0.25)$ with steps of 0.02.

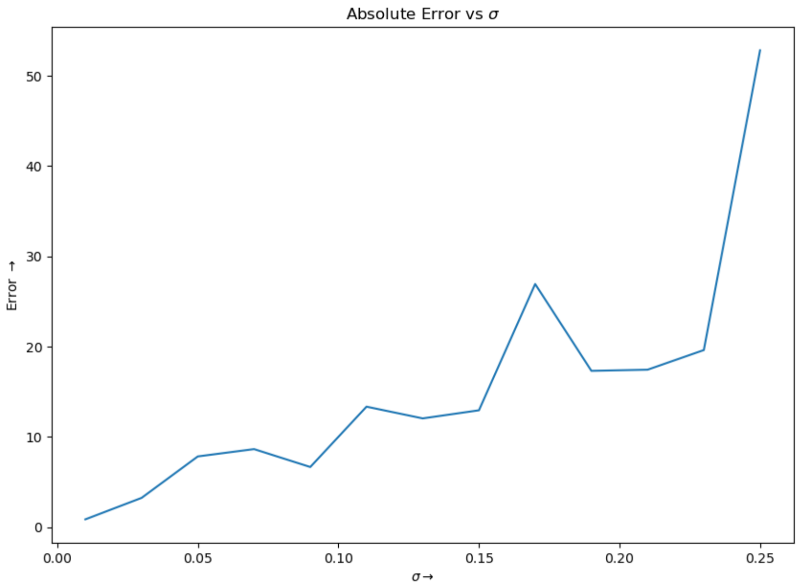
The function affected by noise is subsequently interpolated for different values of σ. These interpolated values are then graphed in comparison to $f(x) = \sin(x)$, as depicted in Figure: 3.



We plot the error in interpolation in a similar way as for the noise counterpart in Figure: 4.



Finally, we plot the absolute error variations as we change the σ in Figure: 5.



4 Conclusion

- Utilized Lagrange Interpolation for 8th-order interpolation of the function $f(x) = \sin(x)$.
- Analyzed actual function values and obtained interpolated values; computed the absolute error in Lagrange interpolation.
- Investigated the impact of introducing Gaussian Noise on interpolation.
- Observed that the absolute error in interpolation rises with increased variance of the noise.