EE5011: Assignment 0

Lagrange Interpolation

J.Phani Jayanth - EE19B026

25 August 2022

Objective

- Interpolation of the function $f(x) = sin(x)$ at the values $0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \frac{5\pi}{4}, \frac{3\pi}{2}, \frac{7\pi}{4}, 2\pi$.
- Implementing Lagrange Interpolation in C with the following calling sequence: *float lintp(float **xx, float *yy, float x, int n)*, that performs an n^{th} order interpolation.
- Evaluating the function on 100 uniformly spaced values between 0 and 2π and writing the output to a text file.
- Running the C executable through the Python script by importing the *os* package.
- Repeating the above with gaussian noise (n) added to the function, i.e., $f(x) = sin(x) + n$, with a variance of σ^2, generated using $sigma * randn(N)$.
- Plotting the error between the actual and interpolated function for both the cases.

Lagrange Interpolation

There exists an interpolating polynomial of degree $N-1$ through N points $y_1 = f(x_1), y_2 = f(x_2), ..., y_N = f(x_N)$. This is given by the Lagrange's classical formula:

$$P(x) = \frac{(x-x_2)(x-x_3)...(x-x_N)}{(x_1-x_2)(x_1-x_3)...(x_1-x_N)}y_1 + \frac{(x-x_1)(x-x_3)...(x-x_N)}{(x_2-x_1)(x_2-x_3)...(x_2-x_N)}y_2 + ... + \frac{(x-x_1)(x-x_2)...(x-x_{N-1})}{(x_N-x_1)(x_N-x_2)...(x_N-x_{N-1})}y_N$$

The above formula consists of N terms, each a polynomial of degree $N-1$ and each constructed to be zero at all of the x_i except one, at which it is constructed to be y_i.

The Lagrange Interpolation function is implemented in C as shown below:

```
float LagrangeInterpolation(float *xx, float *yy, float x, int n) {
    float y = 0;
    for (int i = 0; i < n; i++){
        float L = 1;
        for (int j = 0; j < n; j++)
        {
            if (i != j)
            {
                L *= (x - xx[j]) / (xx[i] - xx[j]);
            }
        }
        y += L * yy[i];
    }
    return y;
}
```

We interpolate the function $f(x) = sin(x)$ at the following nine values: $0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \frac{5\pi}{4}, \frac{3\pi}{2}, \frac{7\pi}{4}, 2\pi$. Since the function is sampled at nine values, we perform an 8^{th} order interpolation, i.e., $n = 8$.

The program is run to evaluate the *Lagrange Interpolation* function on 100 uniformly spaced values between 0 and 2π. The interpolated outputs are then written to text file *output.txt*. This is done as follows:

```
FILE *fp;
fp = fopen("output.txt", "w");

float XX[9] = {0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2};
float *xx = XX;

float YY_[9] = {0,0.7071,1,0.7071,0,-0.7071,-1,-0.7071,0};  // Without Noise
float *yy = YY_;

for (int i = 0; i <= 100; i++)
    {
        y = LagrangeInterpolation(xx, yy, x, 8);    // n = 8
        fprintf(fp, "%f %f\n", x, y);
        x += 0.02;  // For 100 samples between 0 & 2
    }
```

The obtained interpolated values are plotted against the actual function as shown below:

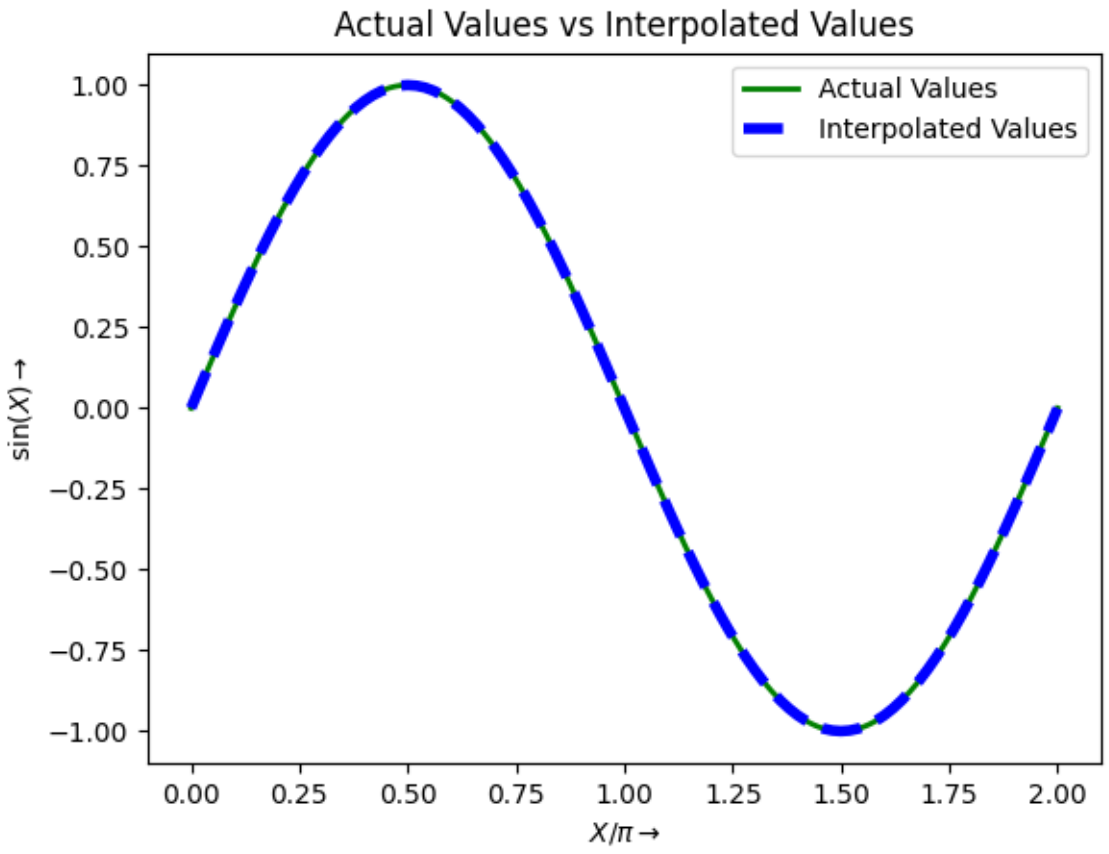

Figure 1: Obtained Interpolated Values vs the Actual function $f(x) = sin(x)$

The plot shown in Figure 1 is plotted by compiling and running the C code through a python script. This is done by importing the *os* package. The *output.txt* is read using *numpy.loadtxt()*. The following function is written in Python for the above-mentioned purposes:

```
import os

def CompileCCode(filename, N):
    os.system("gcc -o " + filename + ".out " + filename + ".c")
    os.system("./" + filename + ".out " + str(N))

    TEXT = np.loadtxt("output.txt")
    INP = TEXT[:,0]
    INTOUT = TEXT[:,1]

    return INP, INTOUT
```

To evaluate the acquired *Lagrange Interpolation*, we plot the error in the interpolated value when compared to the actual value, with a semilog-Y axis.

```
DELTA = np.sin(np.pi*INP) - INTOUT
print("Absolute Error in Interpolation: " + str(np.sum(np.abs(DELTA))))

# Plotting the Absolute Error in Interpolation
title(label = "Absolute Error in Interpolation")
xlabel(r"$X/\pi \rightarrow$")
ylabel(r"Error $\rightarrow$")
scatter(X_, np.zeros(len(X_)), marker = 'o', color = "blue")
semilogy(INP, DELTA, label = "Absolute Error", color = "green")
legend()
show()
```

The plot is obtained as follows:

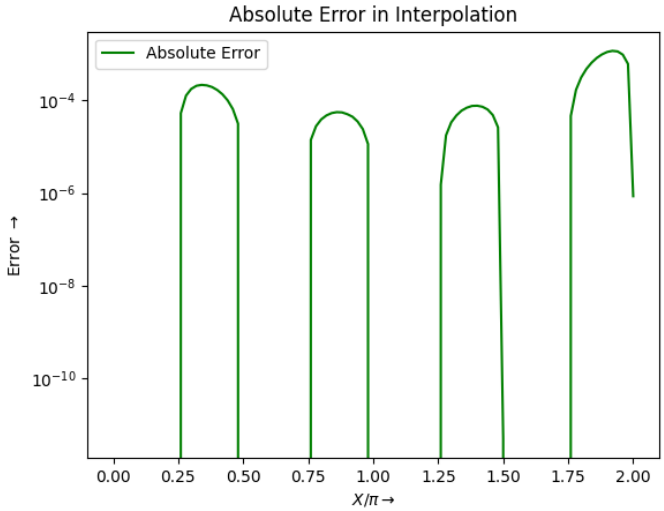

Figure 2: Absolute Error in Interpolation - Semilog-Y

From the plot, we can observe that the error is very minimal, i.e., the interpolation was quite efficient. The total absolute error obtained in the interpolation is ≈ 0.0227.

Addition of Gaussian Noise to $f(x)$

We repeat the experiment with $f(x) = sin(x) + n$, where n is normally distributed noise with a variance of σ^2. The noise is generated using $sigma * randn(N)$, and is added to the function $f(x) = sin(x)$. The σ has been varied from 0.01 to 0.25, in steps of 0.02.

```
SIGMA = np.linspace(0.01, 0.25, 13)

X_ = np.array([0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2])
Y_ = np.sin(np.pi*X_)

OUT = []
for sig in SIGMA:
    X = np.random.randn(9)*sig
    Y = list(np.round(Y_ + X, 5))
    OUT.append(Y)
```

The noisy function is then interpolated across various values of σ. These interpolated values are plotted against $f(x) = sin(x)$, as shown below:

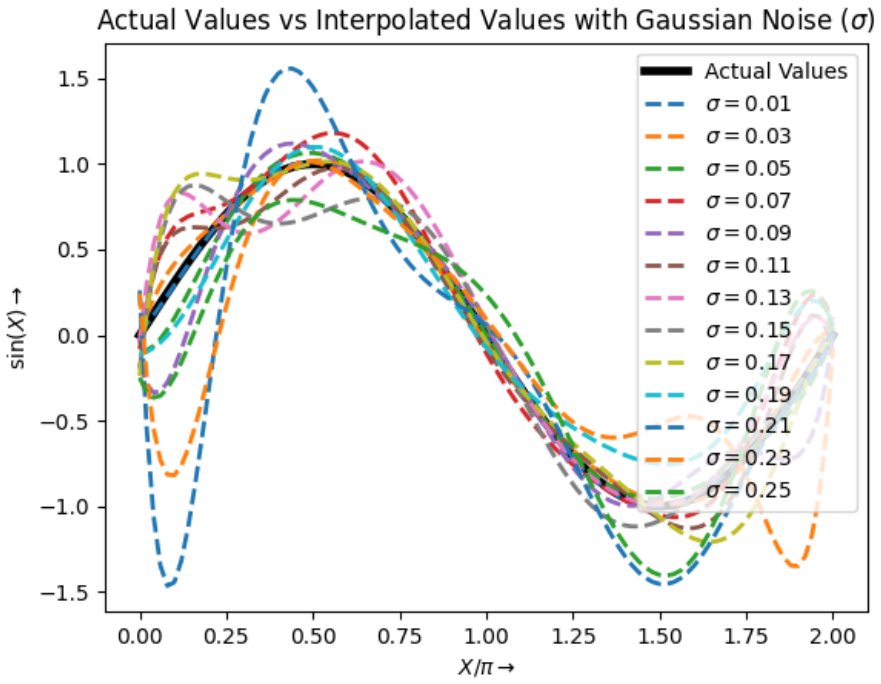

Figure 3: Obtained Interpolated Values with Gaussian noise of various σ

The error in interpolation is also plotted with semilog-Y axis, as shown below:

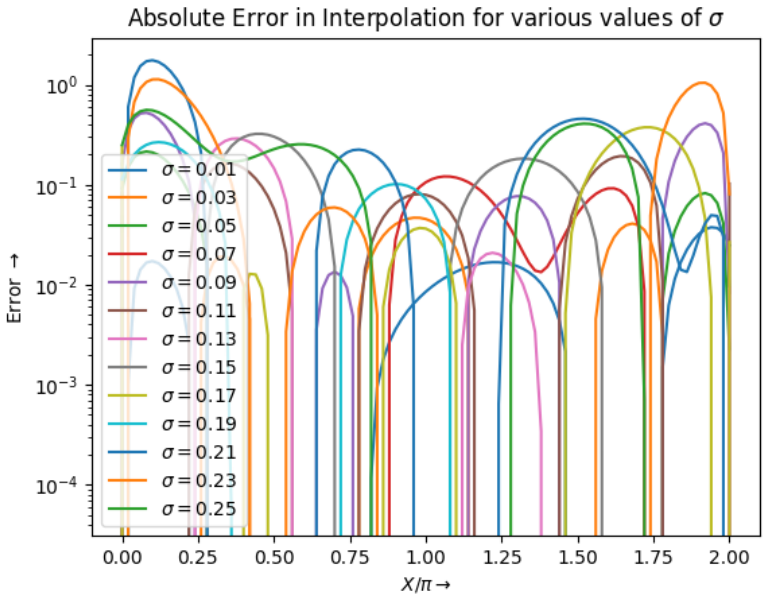

Figure 4: Absolute Error in Interpolation with Gaussian noise of various σ - Semilog-Y

The total absolute error is also plotted against σ, as shown in the following figure:

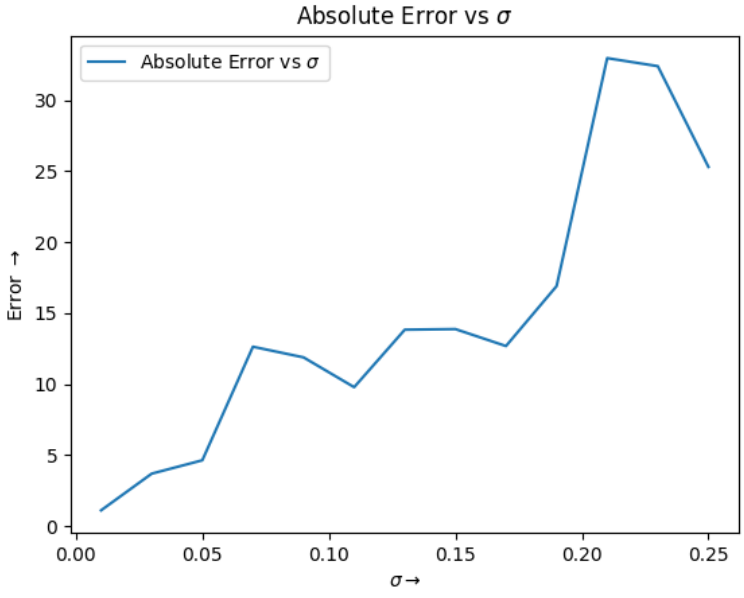

Figure 5: Absolute Error in Interpolation vs σ

From the plot, we can observe that, on an average, the absolute error increases with increase in variance (σ^2) of the noise.

```
SIGMA = np.linspace(0.01, 0.25, 13)
X = np.arange(0,2,1e-5)      # X Values
Y_ACTUAL = np.sin(np.pi*X)  # Actual Y Values
plot(X, Y_ACTUAL, label = "Actual Values", color = "black", linewidth = 4)

for i in range(len(SIGMA)):
    INP, INTOUT = CompileCCode("LagrangeInterpolation", i+1)
    plot(INP, INTOUT, label = r"$\sigma = $" + str(SIGMA[i]), linewidth = 2,
   linestyle = "--")
title("Actual Values vs Interpolated Values with Gaussian Noise ($\sigma$)")
xlabel(r"$X/\pi \rightarrow$")
ylabel(r"$\sin(X) \rightarrow$")
legend()
savefig("LagrangeInterpolationGaussNoise.png")
show()

ABSNOISE = []
for i in range(len(SIGMA)):
    INP, INTOUT = CompileCCode("LagrangeInterpolation", i+1)
    DELTA = np.sin(np.pi*INP) - INTOUT
    ABSNOISE.append(np.sum(np.abs(DELTA)))
    scatter(X_, np.zeros(len(X_)), marker = 'o', color = "blue")
    semilogy(INP, DELTA, label = r"$\sigma = $" + str(round(SIGMA[i],3)))
title("Absolute Error in Interpolation for various values of $\sigma$")
xlabel(r"$X/\pi \rightarrow$")
ylabel(r"Error $\rightarrow$")
legend()
savefig("LagrangeInterpolationErrorAbs1.png")
show()

plot(SIGMA, ABSNOISE, label = "Absolute Error vs $\sigma$")
title("Absolute Error vs " + r"$\sigma$")
xlabel(r"$\sigma \rightarrow$")
ylabel(r"Error $\rightarrow$")
legend()
savefig("LagrangeInterpolationErrorAbsSigma.png")
show()
```

Conclusion

- We utilized Lagrange Interpolation to perform an 8^{th} order interpolation of the function $f(x) = sin(x)$.
- The actual function values and the obtained interpolated values are analysed and the absolute error in *Lagrange Interpolation* is computed.
- Additionally, we studied the affect of addition of *Gaussian Noise* on Interpolation, and noticed that the absolute error in interpolation increases with increase in the variance (σ^2) of the noise.