PHYS 222 Homework: Approximating a Function

Fahed Abu Shaer

September 2023

1 Linear Interpolation

I generated random points using that resembled the curve of $f(x) = x^2$, and I tried to interpolate and estimate the curve using the equation of a line $y = ax+b$ where $a = \frac{y_{i+1}-y_i}{x_{i+1}-x_i}$ and $b = y_i - ax_i$.

The code I wrote is

```
x = np.linspace(-10, 10, 10)
noise = np.random.randint(0, 10)
y = x ** 2 + noise
plt.scatter(x, y)
for i in range(len(x) - 1):
    x1 = x[i]
    x2 = x[i+1]
    y1 = x1 ** 2 + noise
    y2 = x2 ** 2 + noise
    a = (y2 - y1)/(x2 - x1)
    b = y1 - a * x1
    y1 = a * x1 + b
    y2 = a * x2 +b
    plt.plot([x1, x2], [y1, y2])
plt.ylim(-1, 115)
plt.show()
```

This is the graph that I got from this code.

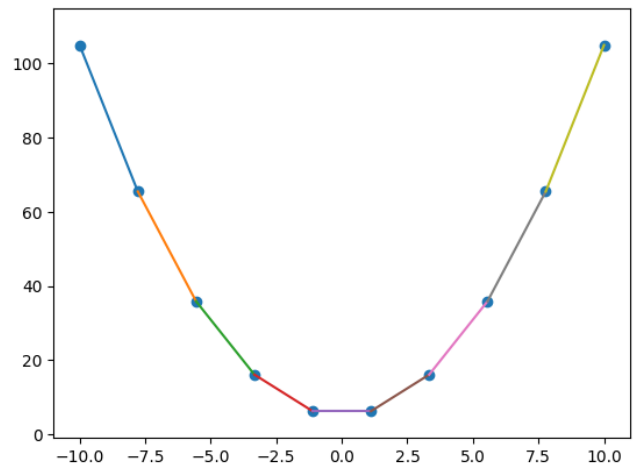

Figure 1: Linear Interpolation

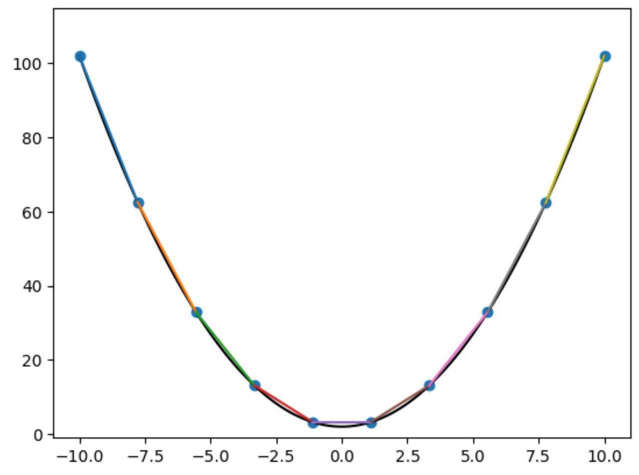

Figure 2: Linear Interpolation and the Function

The lines are a decent approximate of the function, however it is very sharp, but it doesn't fit the function that much, and the error is high.

2 Lagrange Polynomials Interpolation

On the same function, I used Lagrange polynomials to fit the data points where the polynomials are $\prod \frac{x-x_m}{x_j-x_m}$ The code that I wrote and are is:

```
    x = np.linspace(-10, 10, 100)
noise = np.random.randint(0, 10)
y = x ** 2 + noise
plt.plot(x,y, color = 'k')
x = np.linspace(-10, 10, 10)
y = x ** 2 + noise
plt.scatter(x, y)
for i in range(len(x) - 2):
    x1 = x[i]
    x2 = x[i+1]
    x3 = x[i+2]
    y1 = x1 ** 2 + noise
    y2 = x2 ** 2 + noise
    y3 = x3 ** 2 + noise
    L1 = ((x - x2)/(x1 - x2)) * ((x - x3)/(x1 - x3))
    L2 = ((x - x1)/(x2 - x1)) * ((x - x3)/(x2 - x3))
    L3 = ((x - x1)/(x3 - x1)) * ((x - x2)/(x3 - x2))
    y = L1 * y1 + L2 * y2 + L3 * y3
    plt.plot(x, y)
plt.ylim(-10, 115)
plt.show()
```

The graphs that I got when I ran the code:

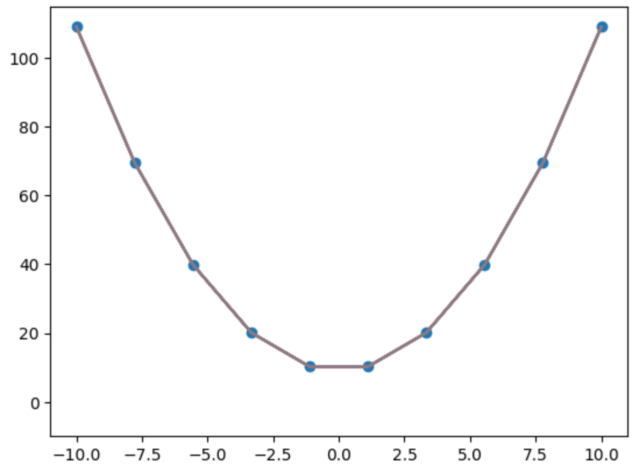

Figure 3: 2nd Order Lagrange Interpolation

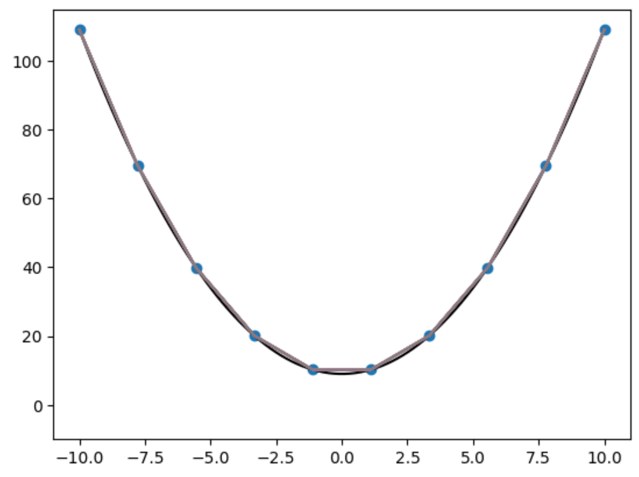

Figure 4: Lagrange Interpolation and the Function

This method yielded a smoother graph that fit the data point, but the margin of error is still big.

3 The Aitken's Method

I used the Aitken's method to estimate the function $f(x) = \frac{1}{1+x^2}$, and this the Aitken function I wrote:

```
def Aitkin (x, P, first, last):
    if first == last:
        return P[first]
    elif first != last:
        return ((x - x[last])/(x[first] - x[last])) * Aitkin(x, P, first, last - 1)
             + ((x - x[first])/(x[last] - x[first])) * Aitkin(x, P, first + 1, last)
```

and this is the results I got:

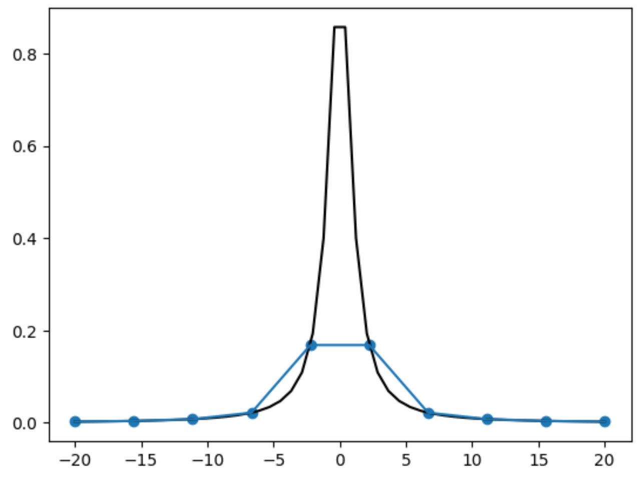

Figure 5: Aitken's Method for Approximating $f(x) = \frac{1}{1+x^2}$

The blue graph is the approximation and the black is the actual function, so we can see that at some points, the graphs matched, but at other points the approximation was off.

4 Quadratic Splines

I generated random data points and constructed a system of equations that describes the 2nd order polynomial between two data points. Then I solved the system and got the coefficients of the polynomial and plotted it. The curve between the first two and last two data points should be first order due to some conditions we took while constructing the system of equations. I implemented the following code:

```
def linear_inter(x, y):
x1 = x[0]
x2 = x[1]
y1 = y[0]
y2 = y[1]
a = (y2 - y1)/(x2 - x1)
b = y1 - a * x1
y1 = a * x1 + b
y2 = a * x2 +b
x = [x1, x2]
y = [y1, y2]
plt.plot(x, y)
return a, b
```

```
def quad_splines(x, y, C0, C1, C2):
X = [[0,0,0],[0,0,0],[0,0,0]]
Y = [y[0], y[1], 2 * C1 * x[1] + C2]
for i in range(2):
    X[i][0] = x[i] ** 2
    X[i][1] = x[i]
    X[i][2] = 1
X[2][0] = 2 * x[1]
X[2][1] = 1
X[2][2] = 0
C = np.linalg.solve(X, Y)
return C

x = np.linspace(-20, 20, 10)
y = np.random.uniform(-20, 20, 10)
l = len(x)
plt.scatter(x, y)
b1, c1 = linear_inter([x[0], x[1]], [y[0], y[1]])
C = [0, b1, c1]
for i in range(1, l - 2):
    var = np.array([x[i], x[i + 1]])
    fun = np.array([y[i], y[i + 1]])
    C = quad_splines(var, fun, C[0], C[1], C[2])
    P = C[0] * x ** 2 + C[1] * x + C[2]
    plt.plot([x[i], x[i+1]], [P[i], P[i+1]])
x1, x2, y1, y2 = x[l-2], x[l-1], y[l-2], y[l-1]
b2, c2 = linear_inter([x1, x2], [y1, y2])
plt.show()
```

This is the graph I obtained:

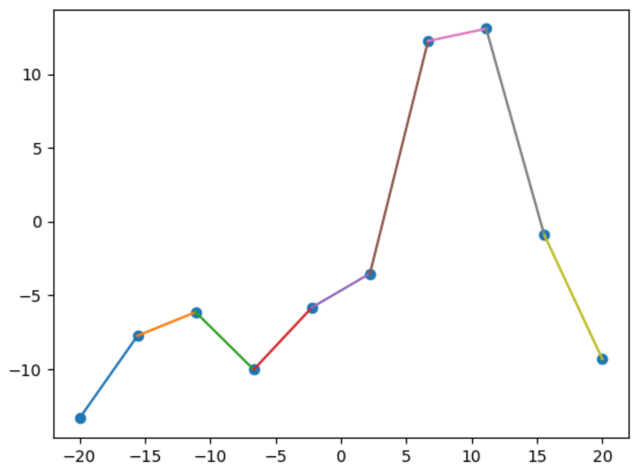

Figure 6: Approximation of a Random Data Set Using Quadratic Splines

The curves between each two points appears to be of the first order, not second order which was what I was trying to achieve. So, I assume there is something wrong with my code, and I am trying to find what it is.

5 Cubic Splines

I imported a package from the Python library that implements approximation using cubic splines, and this is the code:

```
x = np.linspace(-20, 20, 10)
y = np.random.uniform(-20, 20, 10)
plt.scatter(x, y)
P = scp.interpolate.CubicSpline(x, y)
plt.plot(x, P(x))
plt.show()
```

This is the results I got:

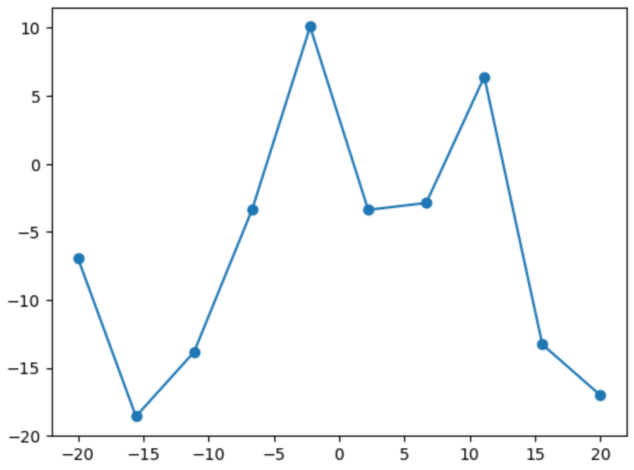

Figure 7: Approximation of a Random Data Set Using Cubic Splines

This yielded similar results to the one I ran to get the approximation using quadratic splines. This raised the question of how different orders of the polynomials used looked like? Maybe I used the function incorrectly which gave me incorrect results.