Numerical Methods Homework 8

Adam Kit - 3707437

7 June 2020

1 Lagrange Interpolation

We aim to interpolate 1

$$f(x) = \frac{1}{1 + 2x^2} \tag{1}$$

using Lagrange's interpolation formula. The formula for Lagrange interpolation is a linear combination of polynomials [1]:

$$L(x) = \sum_{j=0}^{k} y_j l_j(x)$$

where l_j is the polynomial:

$$l_j(x) = \prod_{0 \leq m \leq k; m \neq j} \frac{x - x_m}{x_j - x_m}$$

where $0 \leq j \leq k$. To make this realistically coded, I have a function make the coeffient γ_j that is similar to l_j:

$$\gamma_j = \prod_{0 \leq m \leq j l m \neq j} \frac{y_j}{x_j - x_m}$$

and then using another function, I multiply each γ_j with its respective polynomial across a domain of which to interpolate on. The results when the x-values are the points $(-1, -0.5, 0, 0.5, 1)$ are seen in Figure 1.

1.1 Chebyshev Roots

The chebyshev are given by the equation:

$$x_k = cos\left(\frac{2k - 1}{2n}\pi\right)$$

for $k = 1, ..., n$, where n is the number of input values that we are interpolating with. The result for $n = 5$ is seen below in Figure 2.

1.2 Cheb vs the World

In Figure 3 we see that with 10 data points along the interval $[-1 : 1]$ are better interpolated when they are the Cheb. roots as opposed to evenly spaced data points.

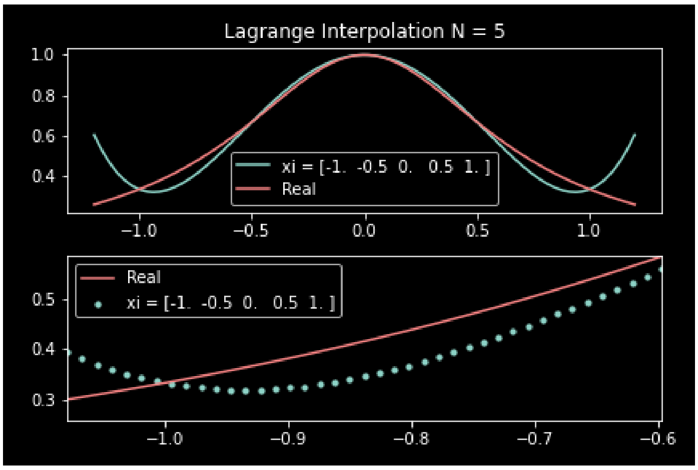

Figure 1: Lagrange Interpolation where $x_i = [(-1.0, -0.5, 0.0, 0.05, 1.0)]$

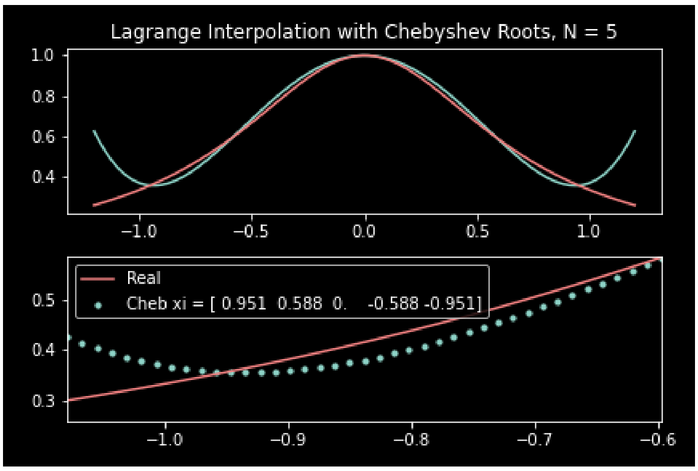

Figure 2: Lagrange Interpolation where x_i are found using the chebyshev root polynomial equation

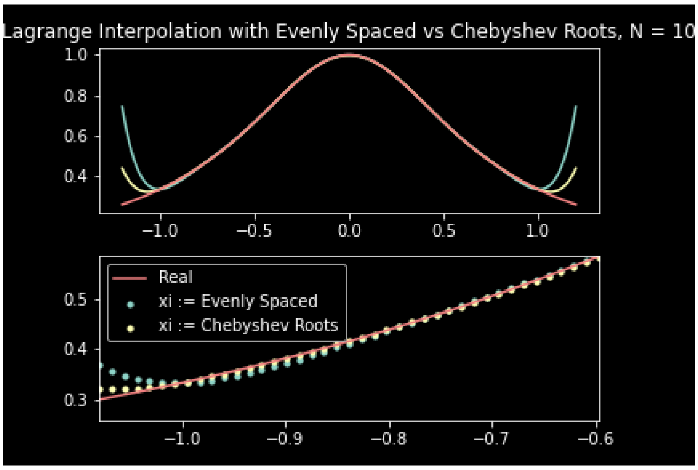

Figure 3: Lagrange Interpolation comparing when x_i are found using the chebyshev root polynomial equation vs an evenly spaced interval of $n = 10$ data points along the interval $[-1 : 1]$

2 Inverse Lagrange

So this problem can be elegently solved by just inserting the x-values into the y-values place in our function used in 1, and vice versus. The results are obtained when doing this for the functions found in *lagrange-formula.py* found in this *repository*, and is visualized in Figure 4.

3 Approximate Solutions using Jacobis Iteration Method

Jacobi's iteration method is represented in the function displayed in Figure 5, and the tensorflow as well as the method can be found in *javobitf.py*.

$$\vec{x} = [1, 2, 3, 0]$$

References

[1] Waring, Edward. *Problems Concerning Interpolations.* Philosophical Transactions of the Royal Society, 1779.

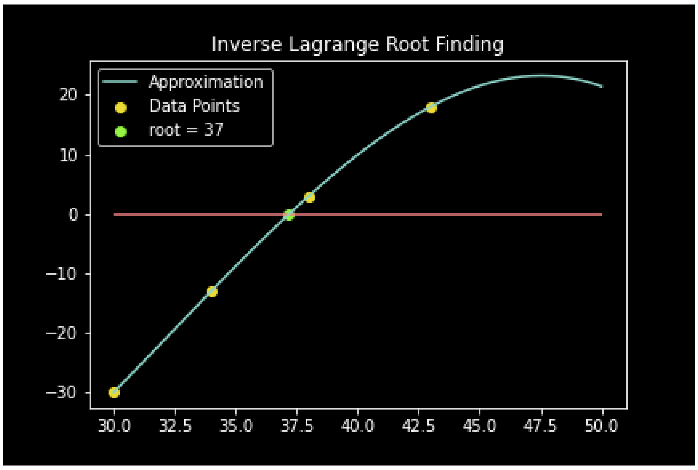

Figure 4: The polynomial is approximated with lagrange interpolation, and the root is found using the inverse. Code is found in *inverse-lagrange.py*

```python
def jacobi(A, b, N=25, x=None):
    """
    Jacobi Iteration for Solving Systems of Equations
    Ax = B
    @params
    A: list or array: L.H.S of equation, matrix to be decomposed
    b: list or array:  R.H.S of equation, end values
    N: int: number of iterations
    x: list or array: initial guesses, default = None
    """
    if x is None:
        x = np.zeros(len(A[0]))

    D = np.diag(A)
    R = A - np.diagflat(D)

    for i in range(N):
        x = (b - np.dot(R, x))/D
    return x
```

Figure 5: The function to iterate the solving of a system of equations via the Jacobi Diagonilization method.