APPM 4600 Lab 7

10 October 2024

The code for this lab can be seen on github here and is included below.

1 Prelab

We have the polynomial

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n,$$

which we wish to use to interpolate the data $\{x_j, f(x_j)\}_{j=0}^n$. We plug these data points into the polynomial and have the system

$$\vec{y} = V\vec{a},$$

where V is the vadermonde matrix

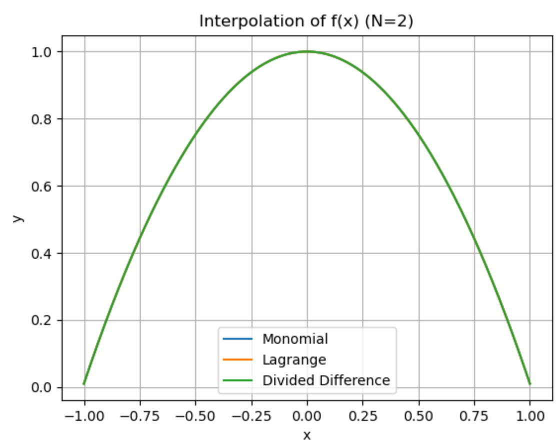
$$V = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ & & \dots & & \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix}.$$

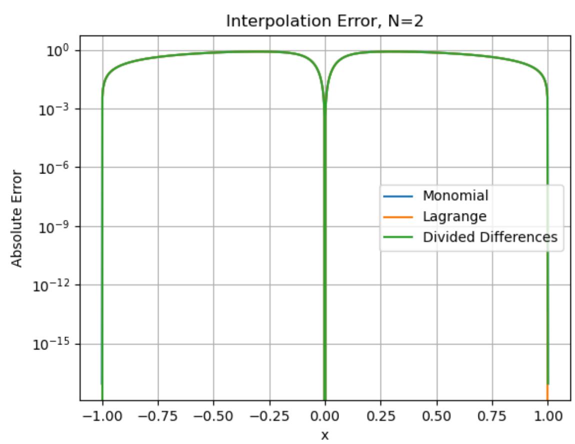
Thus, the coefficients $\vec{a}$ are given by

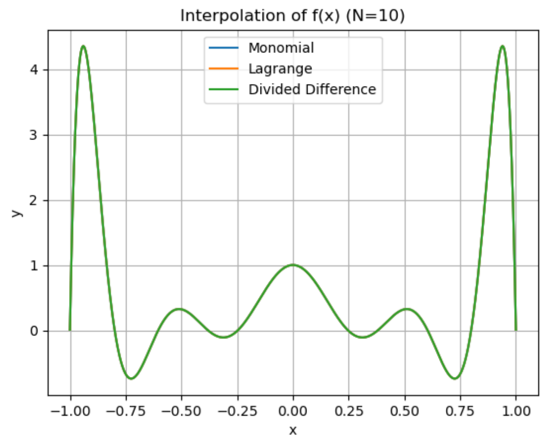
$$\vec{a} = V^{-1}\vec{y}.$$

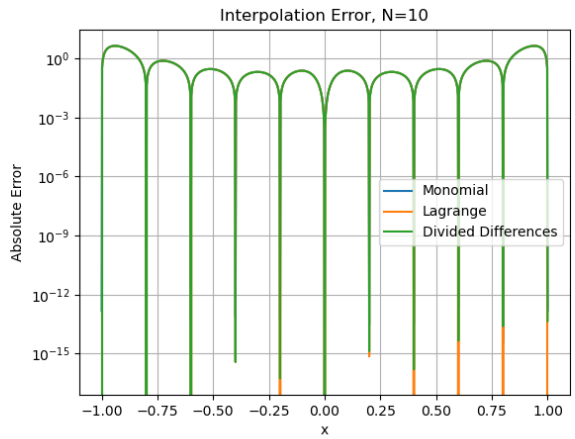
2 Different Interpolation Techniques

The function f is interpolated with the three different methods. Results are shown below for different values of N. Notice that the three methods generally give a very similar polynomial. This is expected, since we know that this polynomial is unique (this was shown in class).

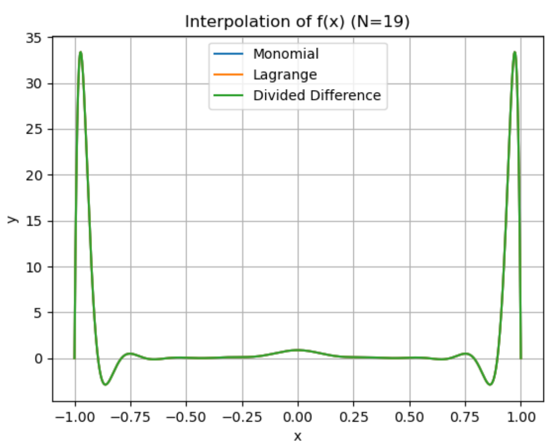


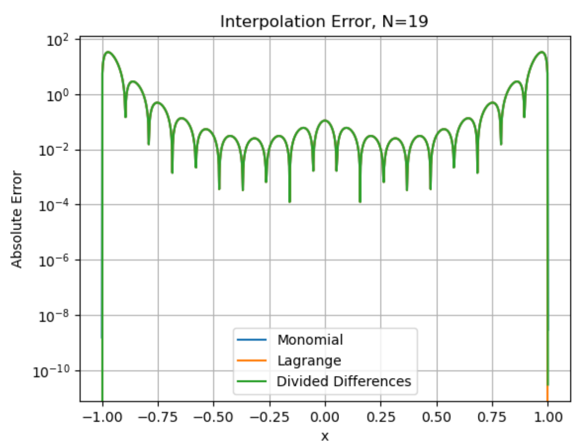






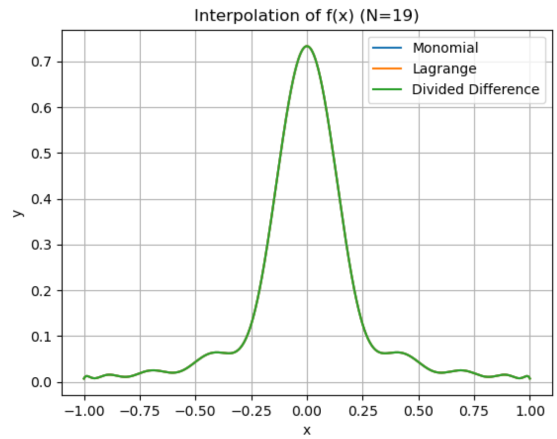
When N becomes large, the polynomials do a poor job of approximating the function near the edges of the domain. In particular, for $N = 19$ (shown below), the error grows very large near $x = -1$ and $x = 1$.

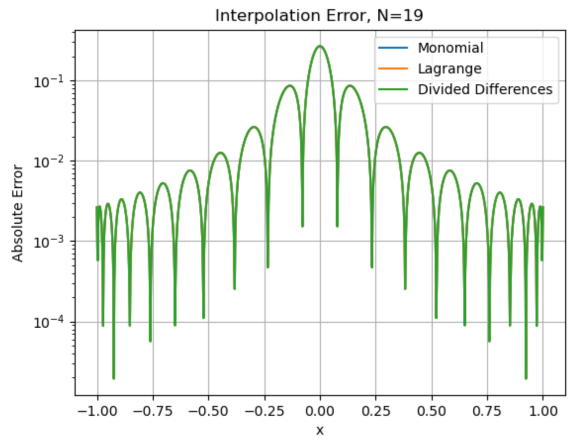




3 Improving the approximation

We place more points near the edges of the interval (as described in lab) and largely eliminate the *Runge* phenomenon. Shown below is the result for $N = 19$ with more points at the edges of the domain. Notice that the error is smallest in the areas where we place more points.






```python
#! /usr/bin/env python3
import numpy as np
import numpy.linalg as la
from numpy.linalg import inv
from numpy.linalg import norm
import matplotlib.pyplot as plt
import math

def  eval_monomial(xeval,coef,N,Neval):

    yeval = coef[0]*np.ones(Neval+1)

#    print('yeval = ', yeval)

    for j in range(1,N+1):
      for i in range(Neval+1):
#        print('yeval[i] = ', yeval[i])
#        print('a[j] = ', a[j])
#        print('i = ', i)
#        print('xeval[i] = ', xeval[i])
        yeval[i] = yeval[i] + coef[j]*xeval[i]**j

    return yeval


def Vandermonde(xint,N):

    V = np.zeros((N+1,N+1))

    ''' fill the first column'''
    for j in range(N+1):
       V[j][0] = 1.0

    for i in range(1,N+1):
        for j in range(N+1):
           V[j][i] = xint[j]**i

    return V

def eval_lagrange(xeval,xint,yint,N):

    lj = np.ones(N+1)
```

```
    for count in range(N+1):
       for jj in range(N+1):
           if (jj != count):
              lj[count] = lj[count]*(xeval - xint[jj])/(xint[count]-xint[jj])

    yeval = 0.

    for jj in range(N+1):
       yeval = yeval + yint[jj]*lj[jj]

    return(yeval)

def lagrange_interp(xj, yj, xeval, N):
    yeval = np.zeros((xeval.size, 1))
    for i in range(xeval.size):
        yeval[i] = eval_lagrange(xeval[i], xj, yj, N)

    return yeval

''' create divided difference matrix'''
def dividedDiffTable(x, y, n):

    for i in range(1, n):
        for j in range(n - i):
            y[j][i] = ((y[j][i - 1] - y[j + 1][i - 1]) /
                                     (x[j] - x[i + j]));
    return y;

def evalDDpoly(xval, xint,y,N):
    ''' evaluate the polynomial terms'''
    ptmp = np.zeros(N+1)

    ptmp[0] = 1.
    for j in range(N):
      ptmp[j+1] = ptmp[j]*(xval-xint[j])

    '''evaluate the divided difference polynomial'''
    yeval = 0.
    for j in range(N+1):
       yeval = yeval + y[0][j]*ptmp[j]

    return yeval

def dd_interp(xj, yj, xeval, N):
    yeval = np.zeros((xeval.size, 1))

    y = np.zeros((N+1, N+1))
    for j in range(N+1):
        y[j][0] = yj[j]
    y = dividedDiffTable(xj, y, N+1)
    for kk in range(xeval.size):
        yeval[kk] = evalDDpoly(xeval[kk], xj, y, N)

    return yeval

def question3_1(N, Neval):
    #xj = np.linspace(-1, 1, N+1)
    j = np.linspace(1, N+1, N+1)
```

```python
    xj = np.cos((2*j-1)*math.pi / (2*(N+1)))

    f = lambda x: 1/(1 + (10*x)**2)
    yj = f(xj)

    xeval = np.linspace(-1, 1, Neval+1)
    # monomial interpolation
    V = Vandermonde(xj, N)
    Vinv = inv(V)
    coef = Vinv @ yj
    ymono = eval_monomial(xeval, coef, N, Neval)
    # Lagrange interpolation
    y_lagrange = lagrange_interp(xj, yj, xeval, N)
    # Divided Differences
    y_dd = dd_interp(xj, yj, xeval, N)

    plt.clf()
    plt.plot(xeval, ymono, label="Monomial")
    plt.plot(xeval, y_lagrange, label="Lagrange")
    plt.plot(xeval, y_dd, label="Divided Difference")

    plt.legend()
    plt.grid()
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Interpolation of f(x) (N=" + str(N) + ")")
    plt.savefig("lab7_uneven_interp_N" + str(N) + ".png")

    plt.clf()
    plt.semilogy(xeval, np.abs(f(xeval) - ymono), label="Monomial")

    plt.semilogy(xeval, np.abs(f(xeval) - np.transpose(y_lagrange))[0], label="Lagrange")
    plt.semilogy(xeval, np.abs(f(xeval) - np.transpose(y_dd))[0], label="Divided Differences")
    plt.legend()
    plt.grid()
    plt.xlabel("x")
    plt.ylabel("Absolute Error")
    plt.title("Interpolation Error, N=" + str(N))
    plt.savefig("lab7_uneven_interp_error_N" + str(N) + ".png")

for N in range(2, 11):
    print(N)
    question3_1(N, 1000)

question3_1(19, 1000)
```