University of Colorado
Boulder

HW7

Jesse Hettleman
APPM 4600

College of Engineering and Applied Science
Department of Applied Math

October 18, 2024

1 Problem 1

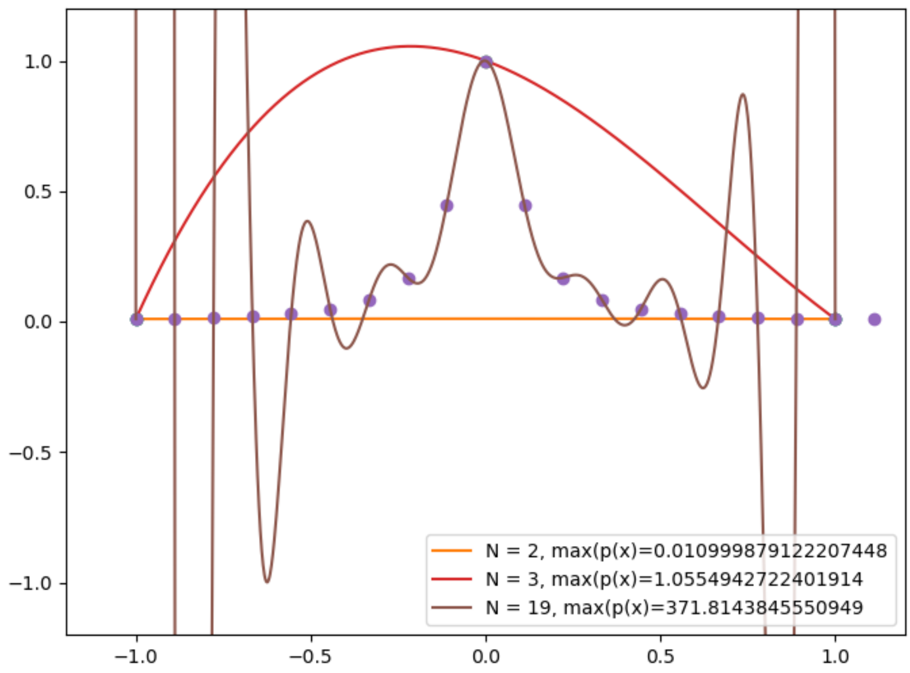
1.a

In order to find the coefficients c of the monomial interpolation, I solved the system $V\mathbf{c} = \mathbf{y}$. To do this, I first created a function that creates the Vandermonde matrix V by filling the i^{th} row with x_i to the power of the j^{th} column. Then, I computed the inverse of this Vandermonde matrix, and multiplied it by $\mathbf{y}$ to find c in the equation $\mathbf{c} = V^{-1}\mathbf{y}$. The resulting vector $\mathbf{c}$ contains the coefficients for the monomial interpolation. See 4 for code conducting these calculations.

1.b

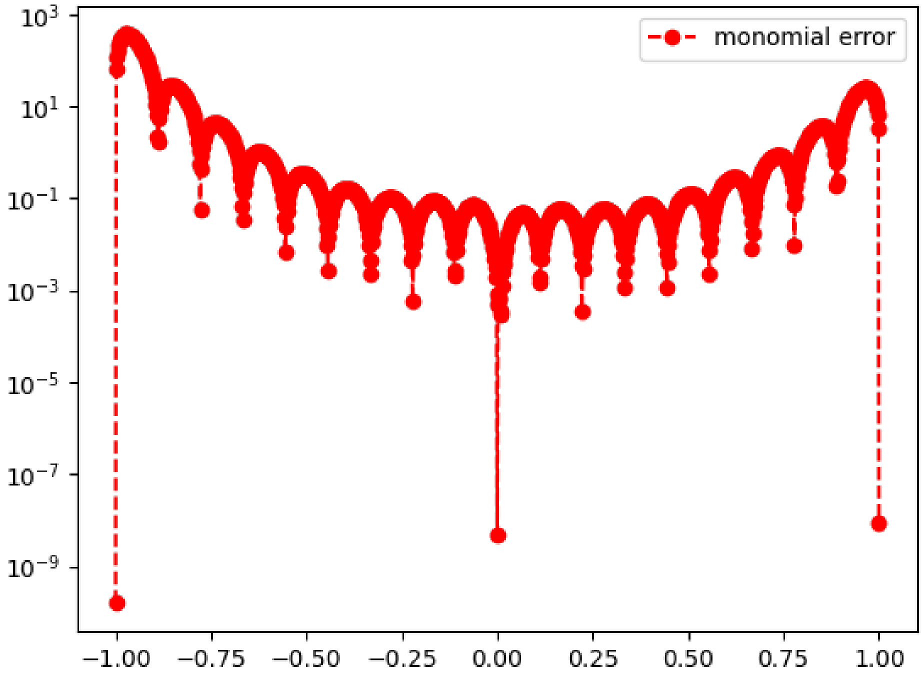
The following plot displays the monomial interpolation of the function $f(x) = \frac{1}{1+(10x)^2}$ for the interpolation points $x_i = -1 + (i-1)\frac{2}{N-1}$. Interpolations are plotted for $N = 2, 3, 19$. Once $N = 19$, the maximum value of $p(x) > 100$. I tested other values of N to discover this fact, but I have removed these values from the plot because it was too cluttered otherwise.

Figure 1: Monomial Interpolations for $N = 2, 3, 19$



Evidently, as N increases, the polynomial $p(x)$ approximates $f(x)$ more closely for the interpolation points. However, this comes at a tradeoff: the endpoints of polynomials with high N are very oscillatory due to their high degree. Thus, there is large error near the end points. This is exemplified in the plot below of the error for the polynomial estimation using $N = 19$:

Figure 2: $p(x)$ **Log Error for** $N = 19$



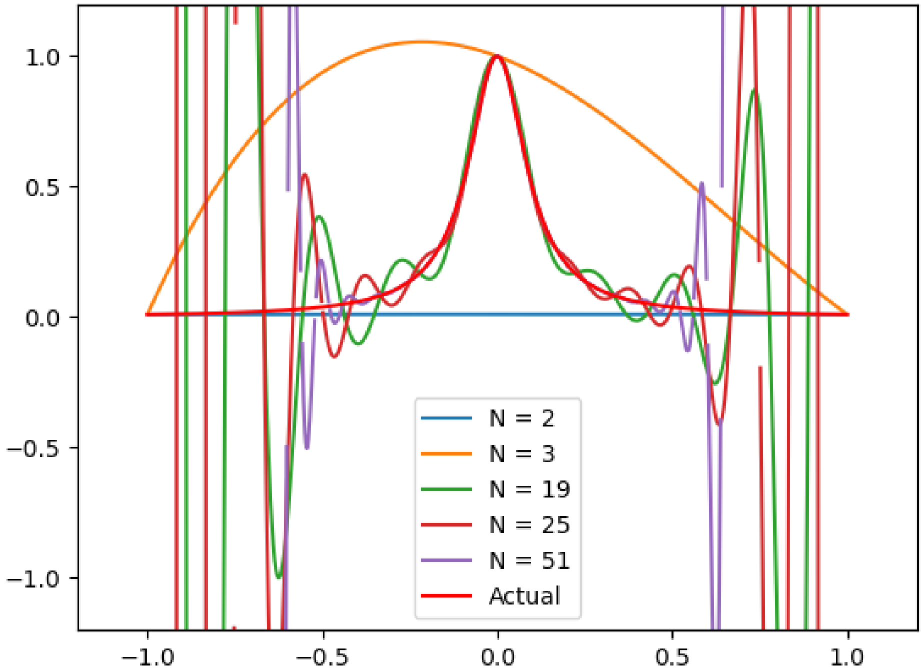
We see that the end points have very large errors, whereas this error decreases for values of x closer to zero. This is the behavior we expected.

2 Problem 2

2.a

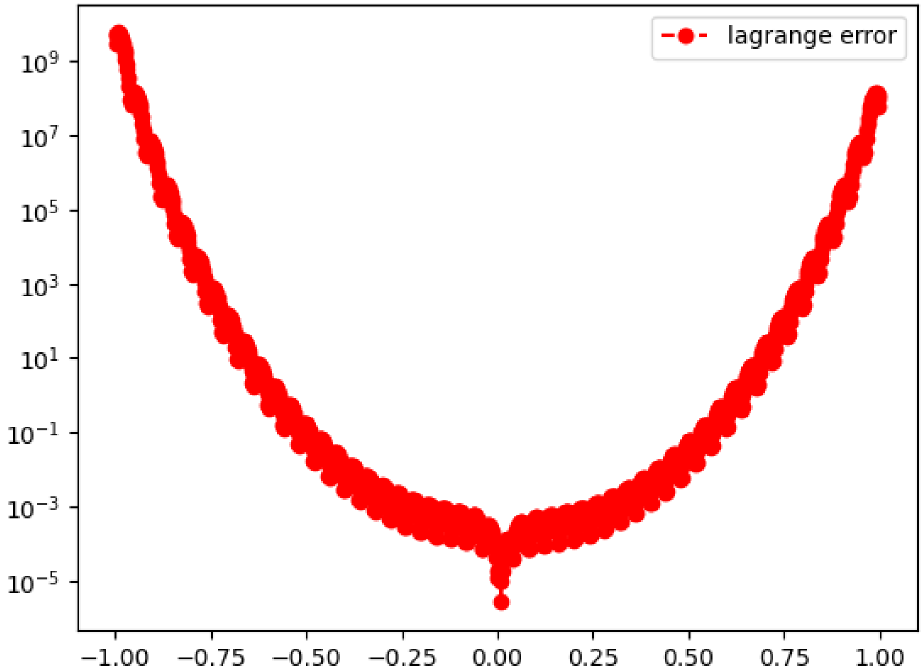
Next we will plot the interpolations of the same function $f(x)$ but using the Barycentric Lagrange method this time. Below are the plots of various $p(x)$ approximations for $N = 2, 3, 19, 25, 51$.

Figure 3: Barycentric Lagrange Interpolations for $N = 2, 3, 19, 25, 51$



The polynomials this time appear to be very strong for the points closer to zero, but wildly erroneous toward the endpoints. This is especially true for higher degree polynomials. Take a look at the log error plot for $N = 51$:

Figure 4: $p(x)$ **Log Error for** $N = 51$



The endpoints here have massive errors on the scale of $10e9$. However, the values of x closer to zero in the center perform much better, with absolute errors of around $10e5$. Again, this is the behavior we expected from the Barycentric Lagrange method.

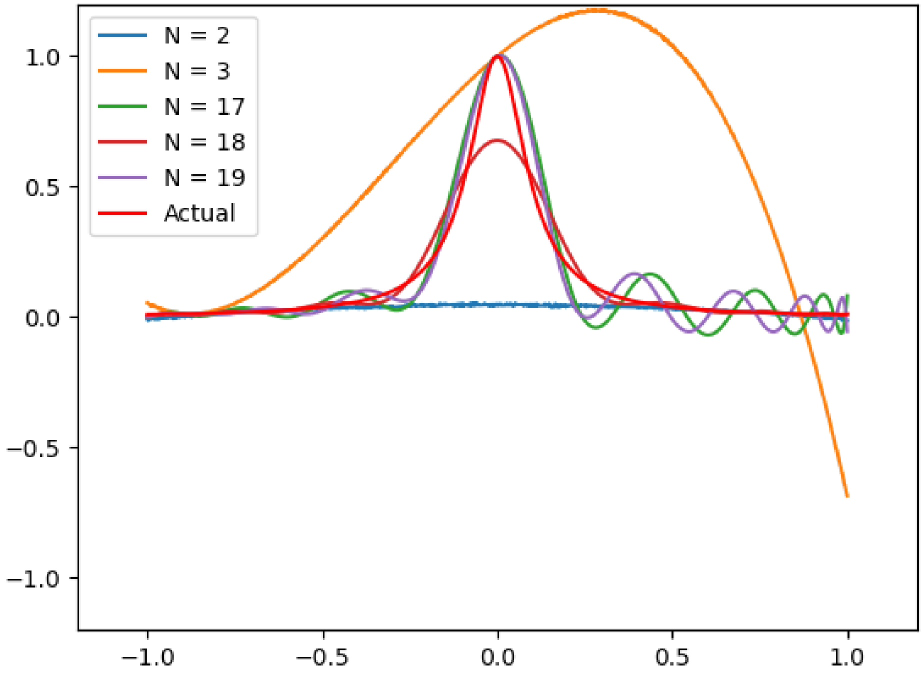
3 Problem 3

Next, we use the interpolation nodes $x_j = cos\frac{(2j-1)\pi}{2N}$. These are known as the Chebyshev points.

The first method of monomial interpolation fails for these points due to the fact that the Vandermond matrix is non-invertible. Therefore, we conclude that the monomial interpolation method is incompatible with the Chebyshev interpolation nodes in this instance.

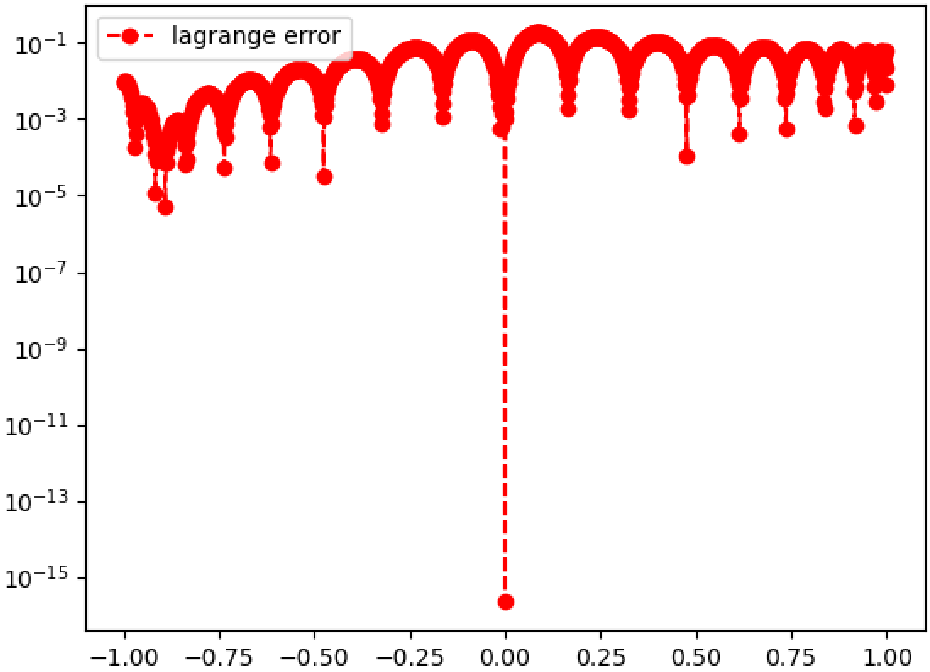
Next, we find that the Barycentric Lagrange method for interpolating the Chebyshev points works. Below is a plot of various $p(x)$ interpolations for $N = 2, 3, 17, 18, 19$:

Figure 5: Barycentric Lagrange Interpolations for $N = 2, 3, 17, 18, 19$



Evidently, the behavior near the endpoints is much better now! Although high degree polynomials still oscillate near the endpoints, these are much more controlled, and result in lower errors. Take a look at the log error plot for $N = 19$, for example:

Figure 6: $p(x)$ **Log Error for** $N = 19$



A noticible difference here is that the error is more even throughout the interval, as opposed to the "smile" shaped error curves we observed before. This is a known effect of the Chebyshev points. Furthermore, the overall error is lower on average. Therefore, the Chebyshev points do not fail, and in fact appear to work better for Barycentric Lagrange interpolation.

4 Appendix

4.1 Code

```
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt
from numpy.linalg import inv
from numpy.linalg import norm
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_bary(xeval,xint,yint,N,f):

    # 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]

    phi = 1

    for xi in xint:
        if xi != xeval:
            phi *= (xeval - xi)

    summation = 0

    for j in range(N+1):

        wj_denom = 1

        for i in range(N+1):
            if xint[j] != xint[i]:
                wj_denom *= (xint[j] - xint[i])

        wj = 1/wj_denom

        summation += (wj/(xeval-xint[j]))*f(xint[j])
```

```
    yeval = phi*summation


    return(yeval)


def question1():

    # Monomial

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

    N = 10
    a = -1
    b = 1

    ''' Create interpolation nodes'''
    xint = np.array([-1 + (j-1)*(2/(N-1)) for j in range(1,N+2)])

    '''Create interpolation data'''
    yint = f(xint)

    ''' Create the Vandermonde matrix'''
    V = Vandermonde(xint,N)

    ''' Invert the Vandermonde matrix'''
    Vinv = inv(V)

    # -----1.a-----

    c = Vinv@yint

    # -----1.b-----

    plt.figure(figsize=(8,6))

    Ns = [2,3,19]

    for N in Ns:
        a = -1
        b = 1

        ''' Create interpolation nodes'''
        xint = np.array([-1 + (j-1)*(2/(N-1)) for j in range(1,N+2)])
        yint = f(xint)

        ''' Create the Vandermonde matrix'''
        V = Vandermonde(xint,N)

        ''' Invert the Vandermonde matrix'''
        Vinv = inv(V)

        c = Vinv@yint

        xeval = np.linspace(-1,1,1001)
        # yeval = f(xeval)
        yeval = eval_monomial(xeval,c,N,1000)
```

```
        ymax = np.max(yeval)



        plt.plot(xint,yint,'o')
        plt.plot(xeval,yeval,label=f'N = {N}, max(p(x)={ymax}')



    plt.legend()
    plt.xlim(-1.2,1.2)
    plt.ylim(-1.2,1.2)
    plt.savefig("HW7.1.b.png")

    fex = f(xeval)
    plt.figure()
    err_m = abs(yeval-fex)
    plt.semilogy(xeval,err_m,'ro--',label='monomial error')
    plt.legend()
    plt.savefig("HW7.1.c.png")

    return

question1()

def question2():


    # Lagrange Barycentric

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

    plt.figure()

    Ns = [2,3,19,25,51]

    a = -1
    b = 1

    for N in Ns:

        ''' Create interpolation nodes'''
        xint = np.array([-1 + (j-1)*(2/(N-1)) for j in range(1,N+2)])

        '''Create interpolation data'''
        yint = f(xint)


        ''' create points for evaluating the Lagrange interpolating ...
           polynomial'''
        Neval = 1000
        xeval = np.linspace(a,b,Neval+1)
        yeval_l= np.zeros(Neval+1)

        ''' evaluate lagrange poly '''
        for kk in range(Neval+1):
            yeval_l[kk] = eval_lagrange_bary(xeval[kk],xint,yint,N,f)

```

```
        plt.plot(xeval,yeval_l,label=f'N = {N}')

    ''' create vector with exact values'''
    Neval = 1000
    xeval = np.linspace(a,b,Neval+1)
    fex = f(xeval)

    plt.plot(xeval,fex,'r-',label='Actual')

    plt.legend()
    plt.xlim(-1.2,1.2)
    plt.ylim(-1.2,1.2)
    plt.savefig("HW7.2.a.png")

    plt.figure()
    err_l = abs(yeval_l-fex)
    plt.semilogy(xeval,err_l,'ro--',label='lagrange error')
    plt.legend()
    plt.savefig("HW7.2.b.png")

    return

question2()

def question3a():

    # Monomial

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



    plt.figure(figsize=(8,6))

    Ns = [2,3,19]

    a = -1
    b = 1

    for N in Ns:

        ''' Create interpolation nodes'''
        xint = np.array([math.cos(((2*j -1)*math.pi)/(2*N)) for j in ...
            range(1,N+2)])
        yint = f(xint)

        ''' Create the Vandermonde matrix'''
        V = Vandermonde(xint,N)

        ''' Invert the Vandermonde matrix'''
        Vinv = inv(V)

        c = Vinv@yint

        xeval = np.linspace(-1,1,1001)
        yeval = f(xeval)
        yeval = eval_monomial(xeval,c,N,1000)
```

```
        fex = f(xeval)
        ymax = np.max(yeval)



        plt.plot(xint,yint,'o')
        plt.plot(xeval,yeval,label=f'N = {N}, max(p(x)={ymax}')



    plt.legend()
    plt.xlim(-1.2,1.2)
    plt.ylim(-1.2,1.2)
    plt.savefig("HW7.3.a.i.png")

    plt.figure()
    err = abs(yeval-fex)
    plt.semilogy(xeval,err,'ro--',label='monomial error')
    plt.legend()
    plt.savefig("HW7.3.a.ii.png")

# 3a fails due to singular matrix being non-invertivle

def question3b():

    # Lagrange Barycentric

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

    plt.figure()

    Ns = [2,3,17,18,19]

    a = -1
    b = 1

    for N in Ns:

        ''' Create interpolation nodes'''
        xint = np.array([math.cos(((2*j -1)*math.pi)/(2*N)) for j in ...
            range(1,N+2)])

        '''Create interpolation data'''
        yint = f(xint)


        ''' create points for evaluating the Lagrange interpolating ...
            polynomial'''
        Neval = 1000
        xeval = np.linspace(a,b,Neval+1)
        yeval_l= np.zeros(Neval+1)

        ''' evaluate lagrange poly '''
        for kk in range(Neval+1):
            yeval_l[kk] = eval_lagrange_bary(xeval[kk],xint,yint,N,f)


        plt.plot(xeval,yeval_l,label=f'N = {N}')
```

```python

    ''' create vector with exact values'''
    Neval = 1000
    xeval = np.linspace(a,b,Neval+1)
    fex = f(xeval)

    plt.plot(xeval,fex,'r-',label='Actual')

    plt.legend()
    plt.xlim(-1.2,1.2)
    plt.ylim(-1.2,1.2)
    plt.savefig("HW7.3.b.i.png")

    plt.figure()
    err = abs(yeval_l-fex)
    plt.semilogy(xeval,err,'ro--',label='lagrange error')
    plt.legend()
    plt.savefig("HW7.3.b.ii.png")

question3b()
```

NOTE: ALL CODE FOR THIS ASSIGNMENT IS ORIGINAL, WRITTEN BY JESSE HETTLEMAN