APPM 4600 Homework 7

17 October 2024

1. The code used to answer this question is listed at the end of the question.

   (a) We have the polynomial

   $$p_n(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n,$$

   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

   $$\mathbf{y} = V\mathbf{c},$$

   where V is the matrix

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

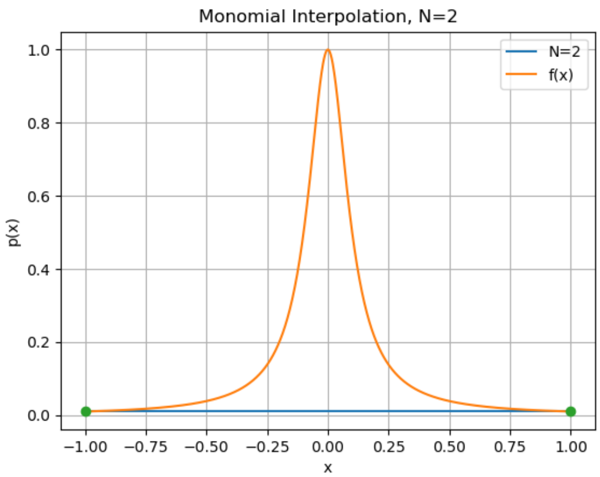
   Thus, the coefficients $\mathbf{c}$ are given by

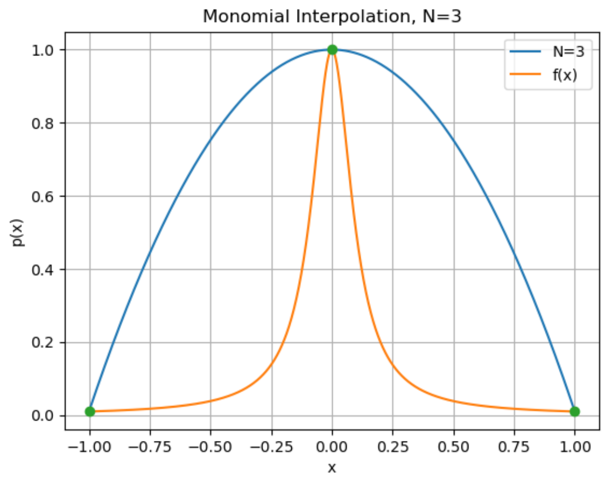
   $$\mathbf{c} = V^{-1}\mathbf{y}.$$

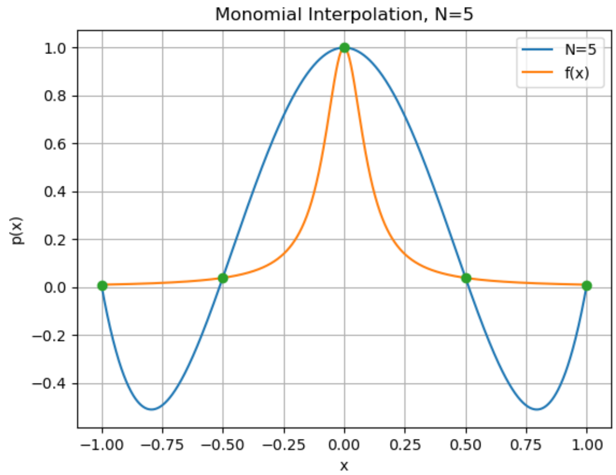
   (b) We perform monomial interpolation on

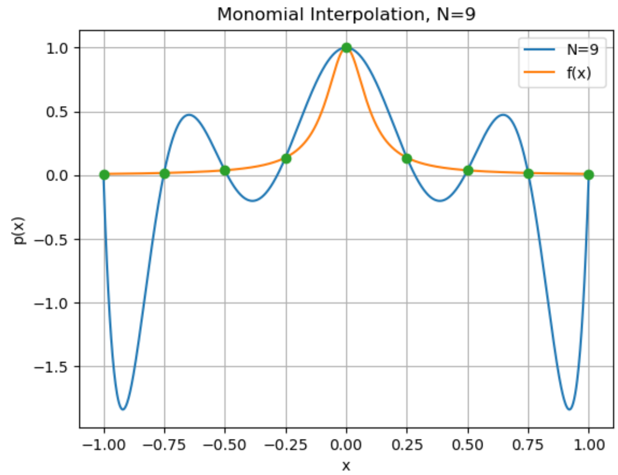
   $$f(x) = \frac{1}{1 + (10x)^2}$$

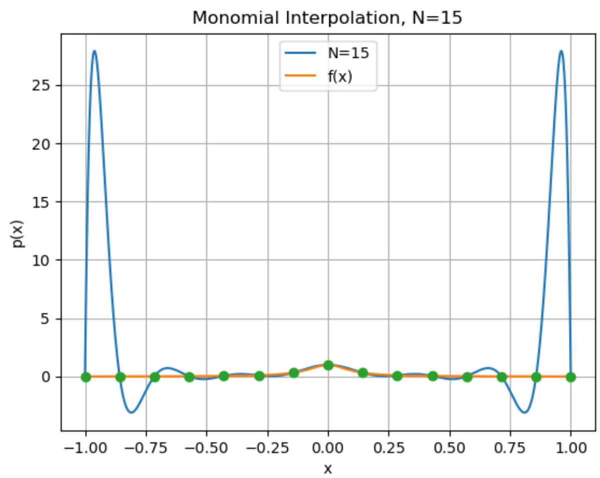
   with an evenly spaced grid of interpolation points over $x \in [-1, 1]$. Resulting polynomials are shown below for various number of points N.

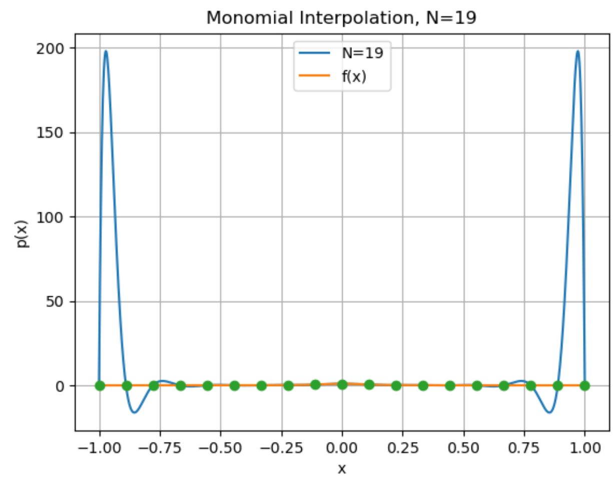












With increasing N, the polynomial is poorly behaved around the endpoints of the region, demonstrating the Runge phenomenon.

```
#! /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

# construct the vandermonde matrix for the given data points
def construct_vandermonde(xj):
    N = xj.size-1
    V = np.zeros((N+1, N+1))
    for i in range(N+1):
        for j in range(N+1):
            V[j][i] = xj[j]**i

    return V

# perform monomial interpolation of (xj, yj) on xeval
def eval_monomial(xj, yj, xeval):
    V = construct_vandermonde(xj)
```

```
    coeff = inv(V) @ yj

    yeval = coeff[0]*np.ones(xeval.shape)

    for j in range(1, xj.size):
        for i in range(xeval.size):
            yeval[i] = yeval[i] + coeff[j]* (xeval[i]**j)

    return yeval

def question1b(N):
    i = np.linspace(1, N, N)
    xj = -1 + (i - 1) * 2 / (N-1)
    # evaluate f(xj)
    f = lambda xj: 1/(1 + (10*xj)**2)
    yj = f(xj)

    xeval = np.linspace(-1, 1, 1001)
    yeval = eval_monomial(xj, yj, xeval)

    plt.clf()
    plt.plot(xeval, yeval, label="N="+str(N))

    plt.plot(xeval, f(xeval), label="f(x)")
    plt.plot(xj, yj, 'o')
    plt.xlabel("x")
    plt.ylabel("p(x)")
    plt.title("Monomial Interpolation, N=" + str(N))
    plt.grid()
    plt.legend()

    plt.savefig("mono" + str(N) + ".png")

for N in range(2, 20):
    question1b(N)
```

2. The code used in this question is listed at the end of the question.

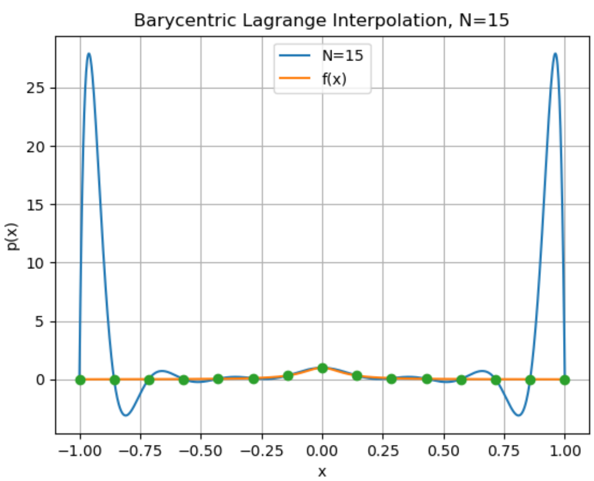
   We use the second Barycentric Lagrange formula,

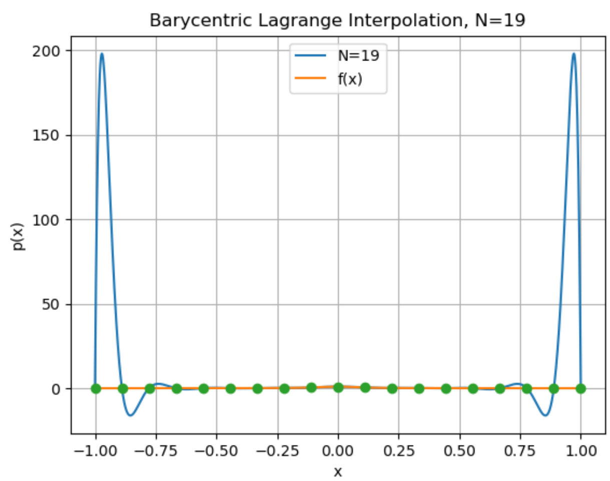
$$p(x) = \frac{\sum_{j=0}^{n} \frac{w_j}{x-x_j} f(x_j)}{\sum_{j=0}^{n} \frac{w_j}{x-x_j}},$$

   where

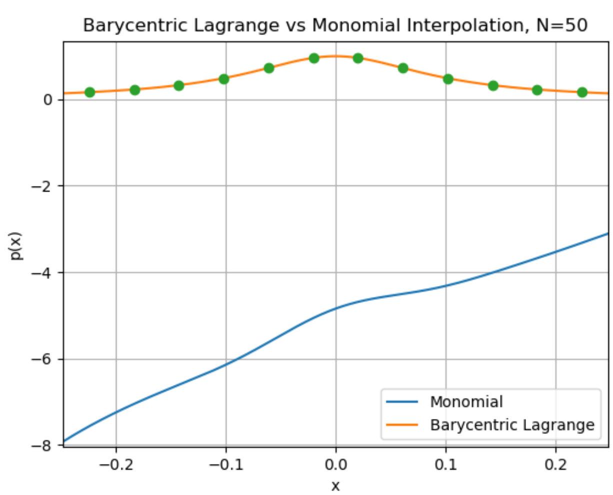
$$w_j = \frac{1}{\prod_{i=0, i\neq j}^{n} (x_j - x_i)}.$$

   The interpolation shows the same Runge phenomenon as before, shown below.





   However, the barycentric interpolation is much more stable than the monomial interpolation. For $N = 50$ interpolation points, both barycentric lagrange and monomial are plotted below. Barycentric Lagrange closely matches the interpolation points, whereas the monomial interpolation misses these points as a consequence of instability.




```
#! /usr/bin/env python3
import numpy as np
import numpy.linalg as la
from numpy.linalg import inv
```

```python
from numpy.linalg import norm
import matplotlib.pyplot as plt
import math

# evaluate w_j for xj
def eval_wj(xj):
    wj = np.zeros(xj.size)
    for j in range(xj.size):
        w = 1
        for i in range(xj.size):
            if i != j:
                w *= (xj[j] - xj[i])

        wj[j] = 1/w

    return wj

def bary_lagrange(xj, yj, wj, x):
    if x in xj:
        i = np.where(xj == x)
        return yj[i[0]]

    n = xj.size
    num = np.sum( wj / (x*np.ones(n) - xj) * yj)
    denom = np.sum(wj / (x*np.ones(n) - xj))

    return num / denom

def eval_bary_lagrange(xj, yj, xeval):
    wj = eval_wj(xj)

    yeval = np.zeros(xeval.size)
    for n in range(xeval.size):
        yeval[n] = bary_lagrange(xj, yj, wj, xeval[n])

    return yeval

# construct the vandermonde matrix for the given data points
def construct_vandermonde(xj):
    N = xj.size-1
    V = np.zeros((N+1, N+1))
    for i in range(N+1):
        for j in range(N+1):
            V[j][i] = xj[j]**i

    return V

# perform monomial interpolation of (xj, yj) on xeval
def eval_monomial(xj, yj, xeval):
    V = construct_vandermonde(xj)
    coeff = inv(V) @ yj

    yeval = coeff[0]*np.ones(xeval.shape)

    for j in range(1, xj.size):
        for i in range(xeval.size):
            yeval[i] = yeval[i] + coeff[j]* (xeval[i]**j)

    return yeval
```

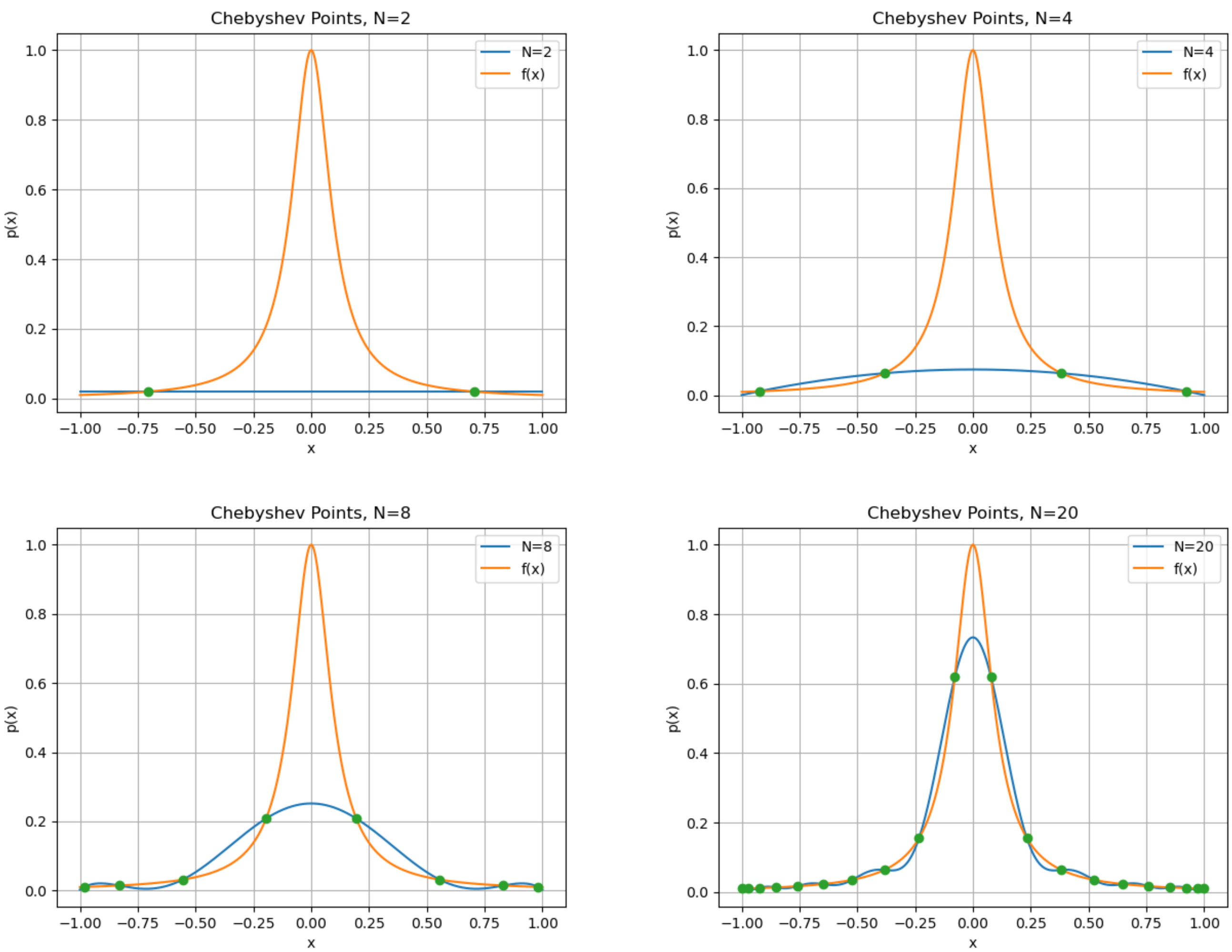
```python
def question1b(N):
    i = np.linspace(1, N, N)
    xj = -1 + (i - 1) * 2 / (N-1)
    # evaluate f(xj)
    f = lambda xj: 1/(1 + (10*xj)**2)
    yj = f(xj)

    xeval = np.linspace(-0.25, 0.25, 1001)
    yeval_mono = eval_monomial(xj, yj, xeval)
    yeval_bary = eval_bary_lagrange(xj, yj, xeval)

    plt.clf()
    plt.plot(xeval, yeval_mono, label="Monomial")
    plt.plot(xeval, yeval_bary, label="Barycentric Lagrange")

    #plt.plot(xeval, f(xeval), label="f(x)")
    plt.plot(xj, yj, 'o')
    plt.xlabel("x")
    plt.ylabel("p(x)")
    plt.title("Barycentric Lagrange vs Monomial Interpolation, N=" + str(N))
    plt.grid()
    plt.legend()

    plt.show()

question1b(50)
```

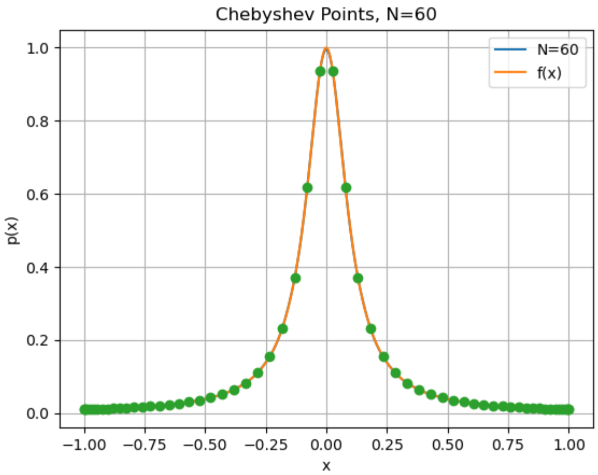
3. The code used in this question is listed at the end of the question.

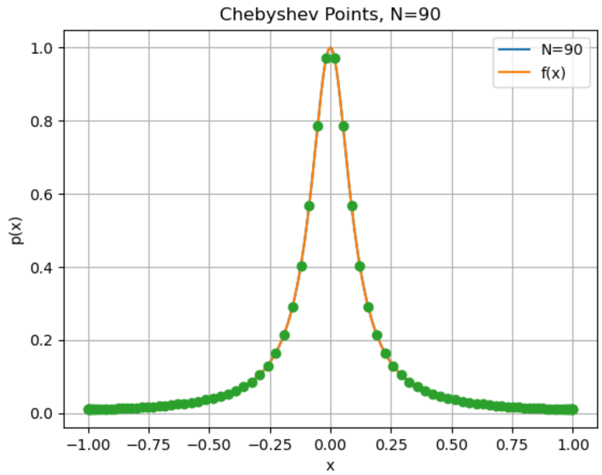
   We apply Barycnetric Lagrange interpolation over the function with Chebyshev points,

$$x_j = \cos\frac{(2j-1)\pi}{2N},\ i = 1, \ldots, N.$$

   The interpolation is shown below for various numbers of points N. Notice that the polynomial remains bounded near the ends of the interval, even for large N.








```
#! /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

# evaluate w_j for xj
def eval_wj(xj):
    wj = np.zeros(xj.size)
    for j in range(xj.size):
        w = 1
        for i in range(xj.size):
            if i != j:
                w *= (xj[j] - xj[i])

        wj[j] = 1/w

    return wj

def bary_lagrange(xj, yj, wj, x):
    if x in xj:
        i = np.where(xj == x)
        return yj[i[0]]

    n = xj.size
    num = np.sum( wj / (x*np.ones(n) - xj) * yj)
    denom = np.sum(wj / (x*np.ones(n) - xj))

    return num / denom

def eval_bary_lagrange(xj, yj, xeval):
    wj = eval_wj(xj)

    yeval = np.zeros(xeval.size)
    for n in range(xeval.size):
        yeval[n] = bary_lagrange(xj, yj, wj, xeval[n])

    return yeval
```

```python
def question1b(N):
    i = np.linspace(1, N, N)

    xj = np.cos((2*i-1)*math.pi / (2*N))

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

    xeval = np.linspace(-1, 1, 1001)
    yeval = eval_bary_lagrange(xj, yj, xeval)

    plt.clf()
    plt.plot(xeval, yeval, label="N="+str(N))

    plt.plot(xeval, f(xeval), label="f(x)")
    plt.plot(xj, yj, 'o')
    plt.xlabel("x")
    plt.ylabel("p(x)")
    plt.title("Chebyshev Points, N=" + str(N))
    plt.grid()
    plt.legend()

    plt.savefig("cheb" + str(N) + ".png")

for N in range(2, 93, 2):
    question1b(N)
```