APPM 4600 Homework 8

25 October 2024

The code used in this assignment is listed at the end.

1. We wish to interpolate

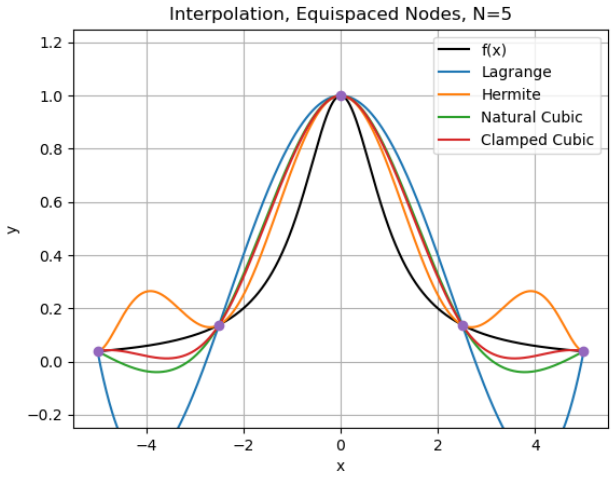
$$f(x) = \frac{1}{1+x^2},$$

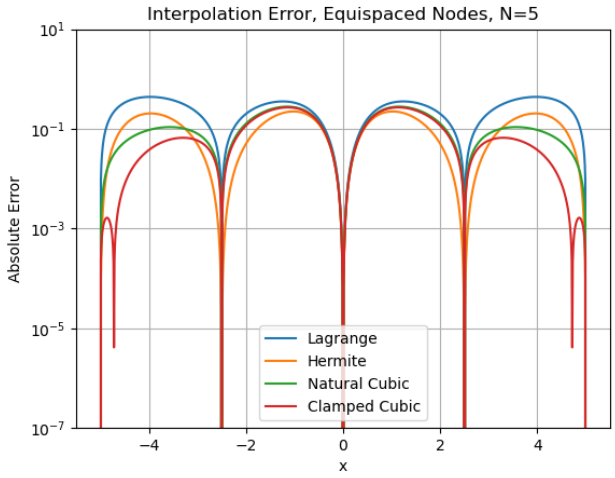
which has derivative

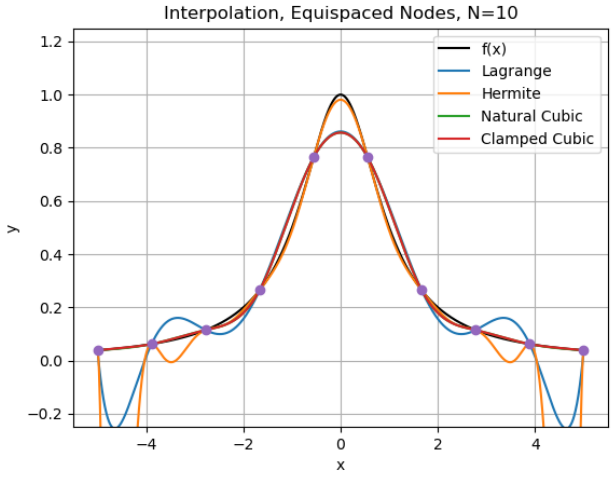
$$f'(x) = -\frac{2x}{(1+x^2)^2}.$$

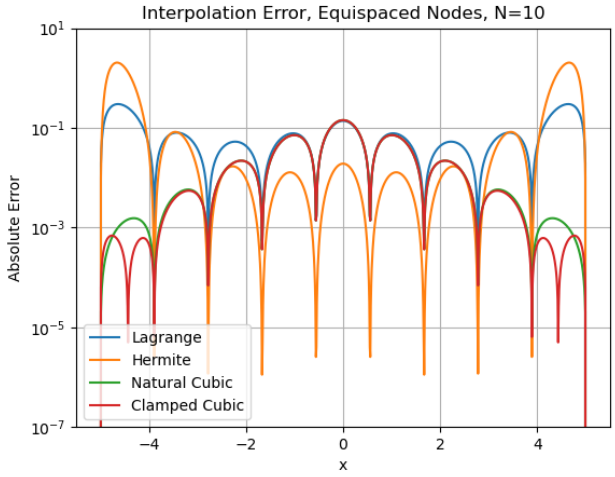
We perform Lagrange, Hermite, and cubic interpolations of f over $[-5,5]$ with equispaced nodes. The results are plotted below for $N = 5, 10, 15, 20$ nodes.

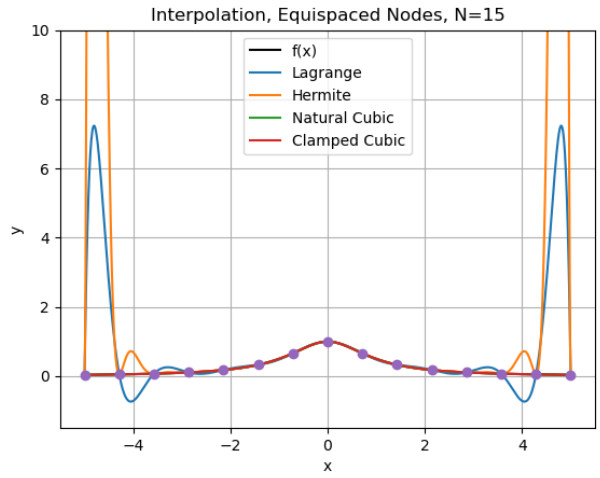
Notice that Lagrange and Hermite interpolation display the Runge phenomenon for large N, while the cubic interpolations do not. This is expected—the cubic interpolations are piecewise between points, so they don't suffer unbounded growth as N increases. In general, the cubic interpolations match the function best, with natural splines having smaller error as N grows.

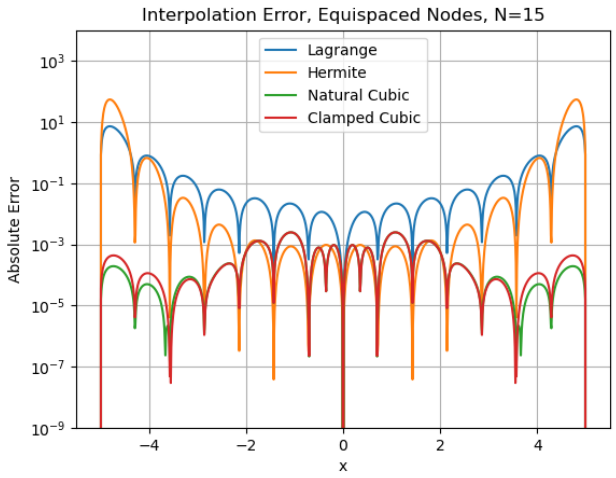


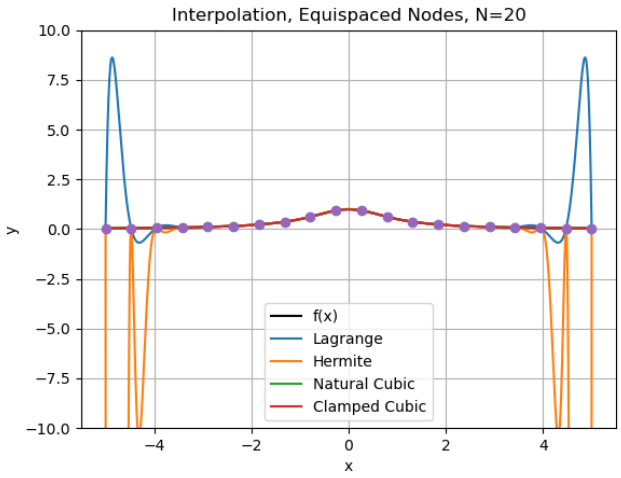


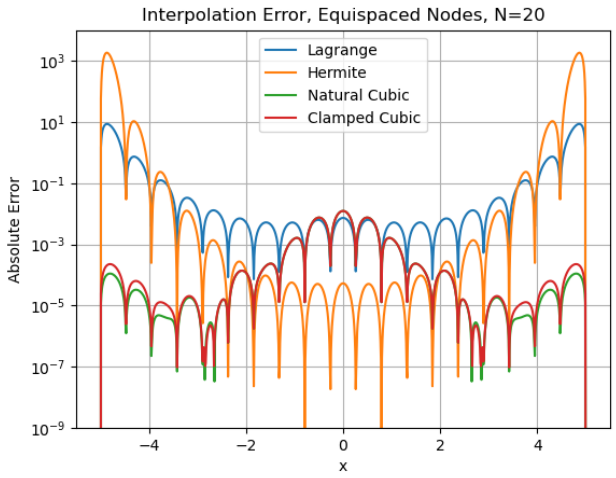





Interpolation, Equispaced Nodes, N=15
f(x)
Lagrange
Hermite
Natural Cubic
Clamped Cubic
x
y


Interpolation Error, Equispaced Nodes, N=15
Lagrange
Hermite
Natural Cubic
Clamped Cubic
Absolute Error
x


Interpolation, Equispaced Nodes, N=20
f(x)
Lagrange
Hermite
Natural Cubic
Clamped Cubic
x
y

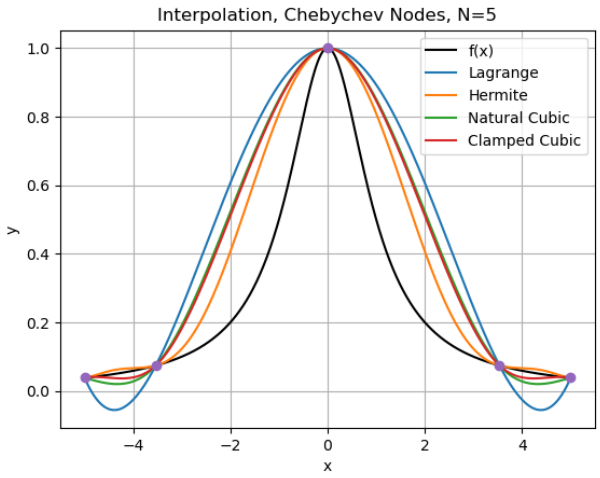

Interpolation Error, Equispaced Nodes, N=20
Lagrange
Hermite
Natural Cubic
Clamped Cubic
Absolute Error
x

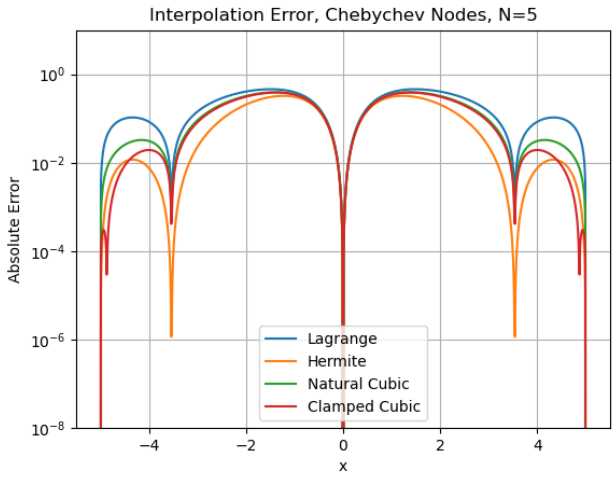
2. We construct the Chebychev nodes of the second kind,

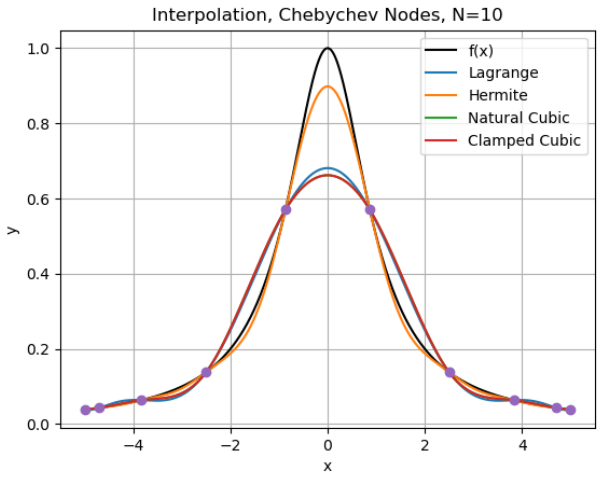
$$x_i = \cos\left(\frac{i}{N-1}\pi\right),\ n = 0, 1, \ldots, N-1.$$

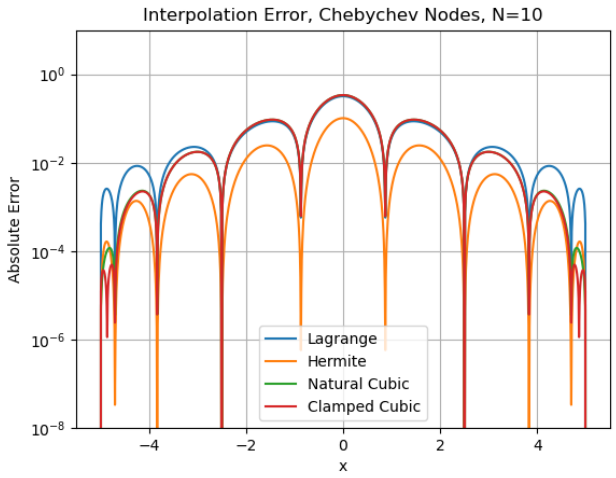
We perform the same interpolation as in the previous question over these nodes, yielding results plotted below.

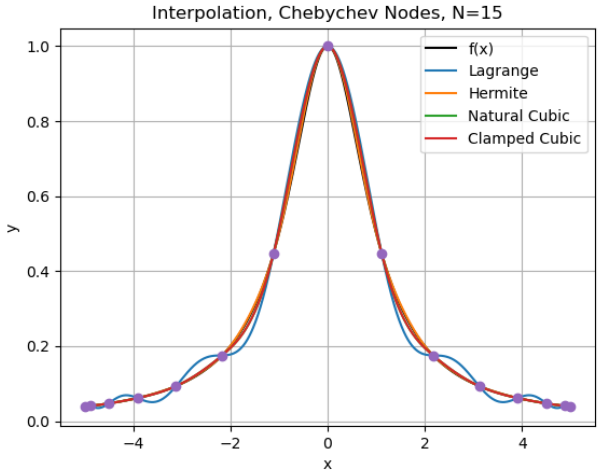
Notice that the Chebychev nodes are effective in eliminating the Runge phenomenon in the Lagrange and Hermite interpolations. All the interpolations are able to get reasonably close to the function. The cubic interpolations perform worse near $x = 0$ than they did with equispaced nodes, which is expected—the relative lack of nodes in this area will hurt their accuracy. In general, the Hermite interpolation has lowest error as N grows, although the cubic interpolations still perform better near the edges.

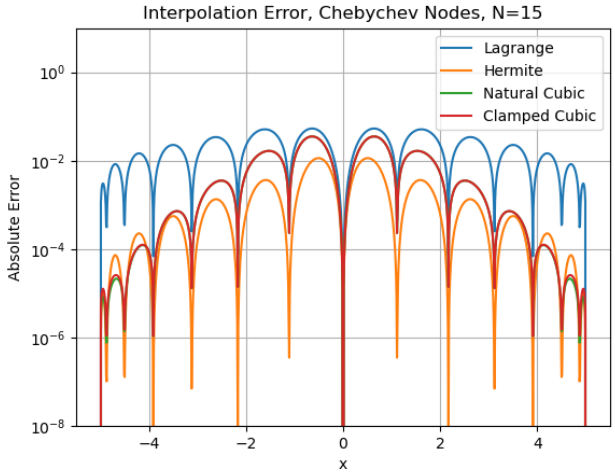


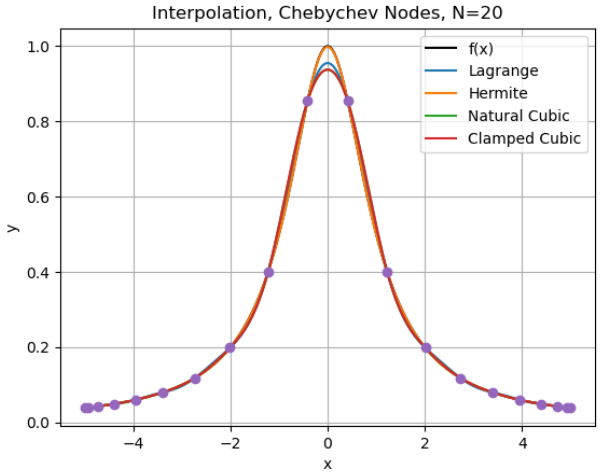


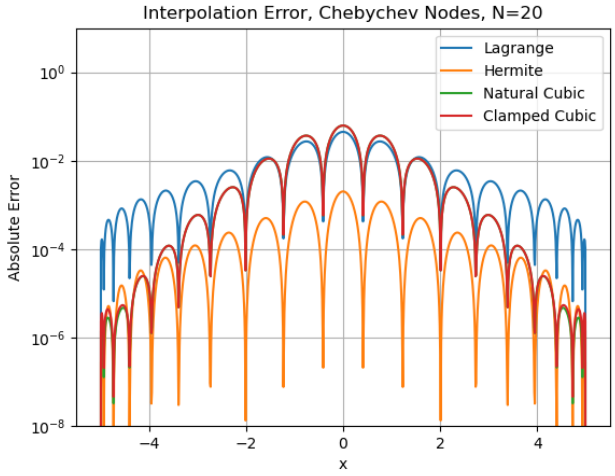





Interpolation, Chebychev Nodes, N=15
f(x)
Lagrange
Hermite
Natural Cubic
Clamped Cubic
x
y


Interpolation Error, Chebychev Nodes, N=15
Lagrange
Hermite
Natural Cubic
Clamped Cubic
Absolute Error
x


Interpolation, Chebychev Nodes, N=20
f(x)
Lagrange
Hermite
Natural Cubic
Clamped Cubic
x
y


Interpolation Error, Chebychev Nodes, N=20
Lagrange
Hermite
Natural Cubic
Clamped Cubic
Absolute Error
x

3. For a periodic spline, we want the periodic boundary conditions

$$S_0(x_0) = S_{n-1}(x_n) \tag{1}$$
$$S_0'(x_0) = S_{n-1}'(x_n) \tag{2}$$
$$S_0''(x_0) = S_{n-1}''(x_n). \tag{3}$$

The condition (1) is enforced by periodicity of the function being interpolated and the fact that we force the spline to match the function at the endpoints. The third condition (3) gives

$$\begin{aligned} 0 &= S_0''(x_0) - S_{n-1}''(x_n) \\ &= M_0 - M_n, \end{aligned}$$

that is, $M_0 = M_n$. The second condition (2) gives

$$\begin{aligned} 0 &= S_0'(x_0) - S_{n-1}'(x_n) \\ &= -\frac{h_0^2 M_0}{2h_0} + \frac{f(x_1) - f(x_0)}{h_0} - \frac{M_1 - M_0}{6} h_0 - \left(-\frac{h_{n-1}^2 M_n}{2h_{n-1}} + \frac{f(x_n) - f(x_{n-1})}{h_{n-1}} - \frac{M_n - M_{n-1}}{6} h_{n-1} \right). \end{aligned} \tag{4}$$

We have from periodicity that $f(x_0) = f(x_n)$, so (4) becomes

$$0 = -\frac{h_0 M_0}{2} + \frac{f(x_1) - f(x_0)}{h_0} - \frac{M_1 - M_0}{6} h_0 + \frac{h_{n-1} M_0}{2} - \frac{f(x_0) - f(x_{n-1})}{h_{n-1}} + \frac{M_0 - M_{n-1}}{6} h_{n-1},$$

which we can rearrange to yield

$$\frac{f(x_{n-1})}{h_{n-1}} + \frac{f(x_1)}{h_0} - f(x_0)\left(\frac{1}{h_0} + \frac{1}{h_{n-1}}\right) = \left(\frac{h_0}{2} - \frac{h_0}{6} - \frac{h_{n-1}}{2} - \frac{h_{n-1}}{6}\right) M_0 + \left(\frac{h_0}{6}\right) M_1 + \left(\frac{h_{n-1}}{6}\right) M_{n-1}.$$

Notice that this has added an M_{n-1} term into relationship we found previously for the natural splines. The matrix A becomes

$$A = \begin{bmatrix} \frac{h_0}{3} & \frac{h_0}{6} & 0 & \cdots & & \frac{h_{n-1}}{6} \\ \frac{h_0}{6} & \frac{h_1+h_0}{3} & \frac{h_1}{6} & 0 & \cdots & 0 \\ \cdots & & & & & \\ 0 & \cdots & & \frac{h_{n-2}}{6} & \frac{h_{n-1}+h_{n-2}}{3} & \frac{h_{n-1}}{6} \\ \frac{h_0}{6} & \cdots & 0 & 0 * \frac{h_{n-1}}{6} & \frac{h_{n-1}}{3} & \end{bmatrix}.$$

```
#! /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

# APPM 4600 Homework 8
# Edward Wawrzynek

# 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

def create_natural_spline(yint,xint,N):
    #    create the right  hand side for the linear system
    b = np.zeros(N+1)
    #  vector values
    h = np.zeros(N+1)
    h[0] = xint[1]-xint[0]
    for i in range(1,N):
       h[i] = xint[i+1] - xint[i]
       b[i] = (yint[i+1]-yint[i])/h[i] - (yint[i]-yint[i-1])/h[i-1]

    #  create the matrix A so you can solve for the M values
    A = np.zeros((N+1,N+1))
    A[0][0] = 1
    A[N][N] = 1
    for j in range(1, N):
```

```
        A[j][j-1] = h[j-1]/6
        A[j][j] = (h[j]+h[j-1]) / 3
        A[j][j+1] = h[j]/6

    #  Invert A
    Ainv = inv(A)

    # solver for M
    M  = Ainv @ b

    #  Create the linear coefficients
    C = np.zeros(N)
    D = np.zeros(N)
    for j in range(N):
       C[j] = yint[j] / h[j] - h[j] / 6 * M[j]
       D[j] = yint[j+1] / h[j] - h[j] / 6 * M[j+1]
    return(M,C,D)

def create_clamped_spline(yint,xint,N):
    #    create the right  hand side for the linear system
    b = np.zeros(N+1)
    #  vector values
    h = np.zeros(N+1)
    h[0] = xint[1]-xint[0]
    for i in range(1,N):
       h[i] = xint[i+1] - xint[i]
       b[i] = (yint[i+1]-yint[i])/h[i] - (yint[i]-yint[i-1])/h[i-1]

    #  create the matrix A so you can solve for the M values
    A = np.zeros((N+1,N+1))
    A[0][0] = h[0]/3
    A[0][1] = h[0]/6
    A[N][N-1] = h[N-1]/6
    A[N][N] = h[N-1]/3
    for j in range(1, N):
        A[j][j-1] = h[j-1]/6
        A[j][j] = (h[j]+h[j-1]) / 3
        A[j][j+1] = h[j]/6

    #  Invert A
    Ainv = inv(A)

    # solver for M
    M  = Ainv @ b

    #  Create the linear coefficients
    C = np.zeros(N)
    D = np.zeros(N)
    for j in range(N):
       C[j] = yint[j] / h[j] - h[j] / 6 * M[j]
       D[j] = yint[j+1] / h[j] - h[j] / 6 * M[j+1]
    return(M,C,D)

def eval_hermite(xeval,xint,yint,ypint,N):

    ''' Evaluate all Lagrange polynomials '''

    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])

    ''' Construct the l_j'(x_j) '''
    lpj = np.zeros(N+1)
#     lpj2 = np.ones(N+1)
    for count in range(N+1):
        for jj in range(N+1):
            if (jj != count):
#                 lpj2[count] = lpj2[count]*(xint[count] - xint[jj])
                lpj[count] = lpj[count]+ 1./(xint[count] - xint[jj])


    yeval = 0.

    for jj in range(N+1):
        Qj = (1.-2.*(xeval-xint[jj])*lpj[jj])*lj[jj]**2
        Rj = (xeval-xint[jj])*lj[jj]**2
        yeval = yeval + yint[jj]*Qj+ypint[jj]*Rj

    return(yeval)

def eval_hermite_poly(xeval, xj, yj, dyj):
    N = xj.size - 1

    yeval = np.zeros(xeval.size)

    for kk in range(xeval.size):
      yeval[kk] = eval_hermite(xeval[kk],xj,yj,dyj,N)

    return yeval

def eval_local_spline(xeval,xi,xip,Mi,Mip,C,D):
    # Evaluates the local spline as defined in class
    # xip = x_{i+1}; xi = x_i
    # Mip = M_{i+1}; Mi = M_i
    hi = xip-xi

    yeval = ((xip - xeval)**3 * Mi / (6*hi) + (xeval - xi)**3 * Mip /
        (6*hi) + C*(xip - xeval) + D*(xeval - xi))
    return yeval


def eval_cubic_spline(xeval,Neval,xint,Nint,M,C,D):

    yeval = np.zeros(Neval+1)

    for j in range(Nint):
        '''find indices of xeval in interval (xint(jint),xint(jint+1))'''
        '''let ind denote the indices in the intervals'''
        atmp = xint[j]
        btmp= xint[j+1]

        #   find indices of values of xeval in the interval
        ind= np.where((xeval >= atmp) & (xeval <= btmp))
        xloc = xeval[ind]

        # evaluate the spline
```

```
        yloc = eval_local_spline(xloc,atmp,btmp,M[j],M[j+1],C[j],D[j])
        #   copy into yeval
        yeval[ind] = yloc

    return(yeval)

def driver():


    f = lambda x: 1./(1.+x**2)
    fp = lambda x: -2*x/(1.+x**2)**2

    N = 15
    ''' interval '''
    a = -5
    b = 5

    ''' create equispaced interpolation nodes '''
    xint = np.linspace(a,b,N+1)

    ''' create interpolation data '''
    yint = np.zeros(N+1)
    ypint = np.zeros(N+1)
    for jj in range(N+1):
        yint[jj] = f(xint[jj])
        ypint[jj] = fp(xint[jj])

    ''' create points for evaluating the Lagrange interpolating polynomial '''
    Neval = 1000
    xeval = np.linspace(a,b,Neval+1)
    yevalL = np.zeros(Neval+1)
    yevalH = np.zeros(Neval+1)
    for kk in range(Neval+1):
      yevalL[kk] = eval_lagrange(xeval[kk],xint,yint,N)
      yevalH[kk] = eval_hermite(xeval[kk],xint,yint,ypint,N)

def question1(N):
    f = lambda x : 1/(1+x**2)
    df = lambda x: -2*x / ((1+x**2)**2)
    # construct linear nodes
    i = np.linspace(1, N, N)
    xj = 5*(-1 + (i - 1) * 2 / (N-1))

    yj = f(xj)
    dyj = df(xj)
    # plotting points
    Neval = 1000
    xeval = np.linspace(-5, 5, Neval + 1)
    yeval = f(xeval)

    # construct Lagrange interpolation
    yeval_lagrange = eval_bary_lagrange(xj, yj, xeval)

    # construct Hermite interpolation
    yeval_hermite = eval_hermite_poly(xeval, xj, yj, dyj)

    # construct natural spline
    (M,C,D) = create_natural_spline(yj, xj, N-1)
    yeval_cubic_nat = eval_cubic_spline(xeval,Neval,xj,N-1,M,C,D)
```

```
    # construct clamped spline
    (M,C,D) = create_clamped_spline(yj, xj, N-1)
    yeval_cubic_clamp = eval_cubic_spline(xeval,Neval,xj,N-1,M,C,D)

    # plot results
    fig1 = plt.figure(1)
    plt.plot(xeval, yeval, 'k', label="f(x)")
    plt.plot(xeval, yeval_lagrange, label="Lagrange")
    plt.plot(xeval, yeval_hermite, label="Hermite")
    plt.plot(xeval, yeval_cubic_nat, label="Natural Cubic")
    plt.plot(xeval, yeval_cubic_clamp, label="Clamped Cubic")
    plt.plot(xj, yj, 'o')
    plt.grid()
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("y")
    if N <= 10:
        plt.ylim((-0.25, 1.25))
    elif N == 15:
        plt.ylim((-1.5, 10))
    else:
        plt.ylim((-10, 10))
    plt.title("Interpolation, Equispaced Nodes, N=" + str(N))

    plt.savefig("equi_N" + str(N) + "_interp.png")
    fig1.clear()

    # plot error
    fig2 = plt.figure(2)
    plt.semilogy(xeval, np.abs(yeval - yeval_lagrange), label="Lagrange")
    plt.semilogy(xeval, np.abs(yeval - yeval_hermite), label="Hermite")
    plt.semilogy(xeval, np.abs(yeval - yeval_cubic_nat), label="Natural Cubic")
    plt.semilogy(xeval, np.abs(yeval - yeval_cubic_clamp), label="Clamped Cubic")
    if N <= 10:
        plt.ylim((1e-7, 1e1))
    else:
        plt.ylim((1e-9, 1e4))

    plt.grid()
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("Absolute Error")
    plt.title("Interpolation Error, Equispaced Nodes, N=" + str(N))
    plt.savefig("equi_N" + str(N) + "_error.png")
    fig2.clear()

def question2(N):
    f = lambda x : 1/(1+x**2)
    df = lambda x: -2*x / ((1+x**2)**2)
    # construct Chebychev nodes (of second kind)
    i = np.linspace(1, N, N)
    xj = 5*np.cos((i-1)/(N-1)*math.pi)
    xj = np.flip(xj)

    yj = f(xj)
    dyj = df(xj)
    # plotting points
    Neval = 1000
```

```
    xeval = np.linspace(-5, 5, Neval + 1)
    yeval = f(xeval)

    # construct Lagrange interpolation
    yeval_lagrange = eval_bary_lagrange(xj, yj, xeval)

    # construct Hermite interpolation
    yeval_hermite = eval_hermite_poly(xeval, xj, yj, dyj)

    # construct natural spline
    (M,C,D) = create_natural_spline(yj, xj, N-1)
    yeval_cubic_nat = eval_cubic_spline(xeval,Neval,xj,N-1,M,C,D)

    # construct clamped spline
    (M,C,D) = create_clamped_spline(yj, xj, N-1)
    yeval_cubic_clamp = eval_cubic_spline(xeval,Neval,xj,N-1,M,C,D)

    # plot results
    fig1 = plt.figure(1)
    plt.plot(xeval, yeval, 'k', label="f(x)")
    plt.plot(xeval, yeval_lagrange, label="Lagrange")
    plt.plot(xeval, yeval_hermite, label="Hermite")
    plt.plot(xeval, yeval_cubic_nat, label="Natural Cubic")
    plt.plot(xeval, yeval_cubic_clamp, label="Clamped Cubic")
    plt.plot(xj, yj, 'o')
    plt.grid()
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Interpolation, Chebychev Nodes, N=" + str(N))

    plt.savefig("cheb_N" + str(N) + "_interp.png")
    fig1.clear()

    # plot error
    fig2 = plt.figure(2)
    plt.semilogy(xeval, np.abs(yeval - yeval_lagrange), label="Lagrange")
    plt.semilogy(xeval, np.abs(yeval - yeval_hermite), label="Hermite")
    plt.semilogy(xeval, np.abs(yeval - yeval_cubic_nat), label="Natural Cubic")
    plt.semilogy(xeval, np.abs(yeval - yeval_cubic_clamp), label="Clamped Cubic")
    plt.ylim((1e-8, 1e1))

    plt.grid()
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("Absolute Error")
    plt.title("Interpolation Error, Chebychev Nodes, N=" + str(N))
    plt.savefig("cheb_N" + str(N) + "_error.png")
    fig2.clear()

for N in [5, 10, 15, 20]:
    question1(N)
    question2(N)
```