APPM 4600 Lab 8

17 October 2024

The code for this lab can be seen at the end of this document, or on github here (linear splines) and here (cubic splines).

1 Prelab

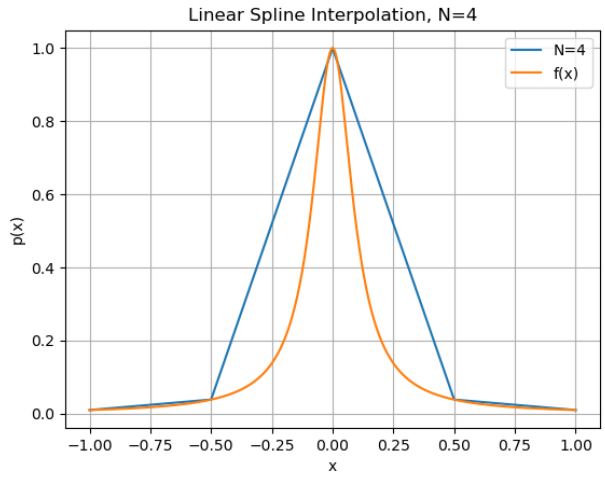
1. The code to evaluate a line through given points is included at the end of the document, in the function `eval_line`.

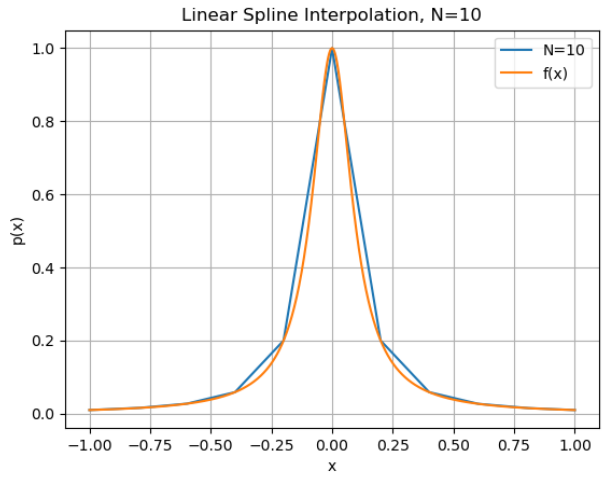
2 Linear Splines

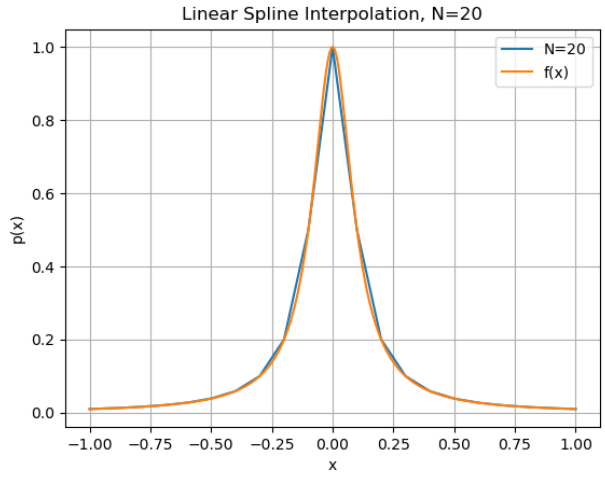
1. The code for the linear splines was implemented. Shown below is the interpolation of the function

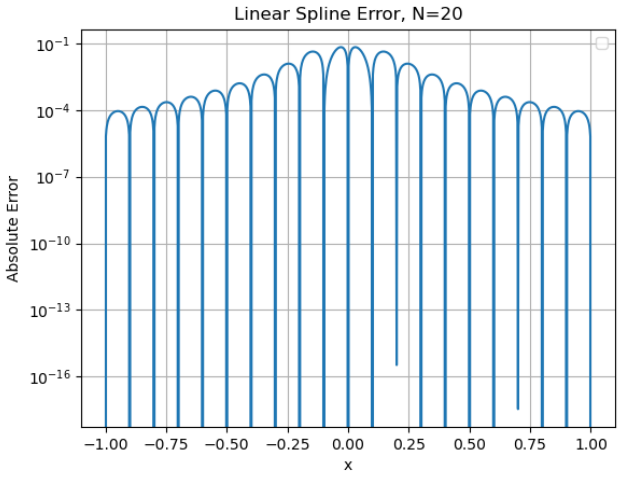
$$f(x) = \frac{1}{1 + (10 * x)^2}$$

with various numbers of interpolation points distributed uniformly over the interval $x \in [-1, 1]$.





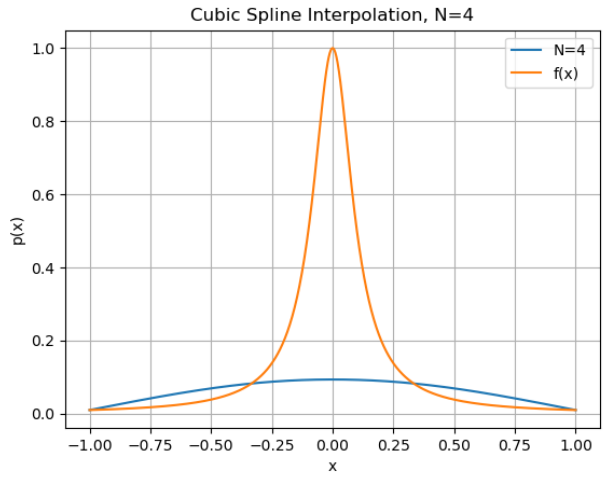


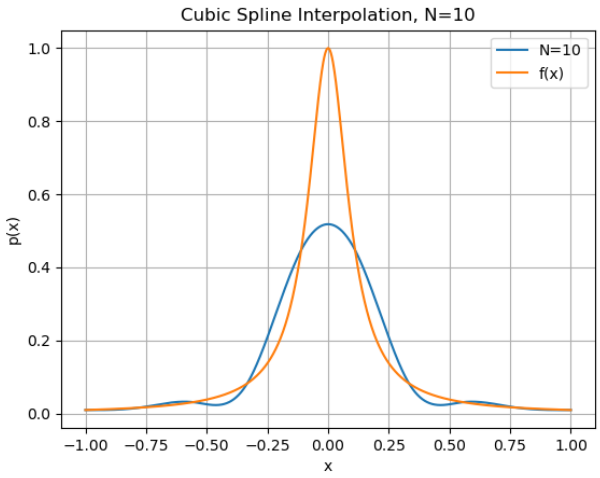


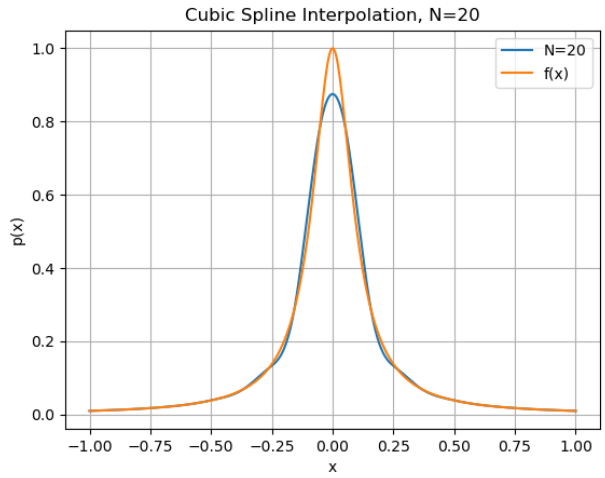
Notice that the spline interpolation does not display the Runge phenomenon.

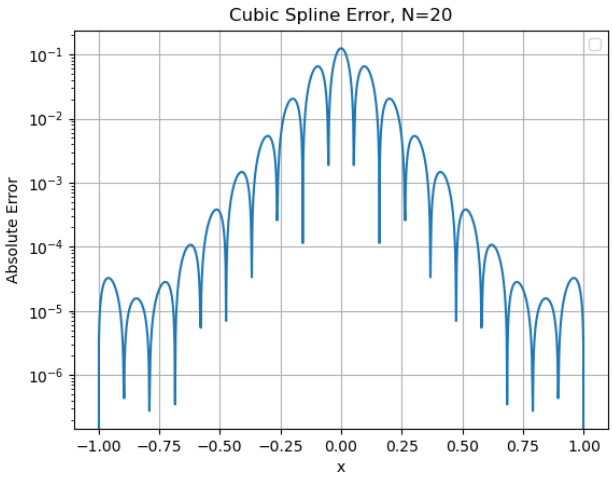
3 Cubic Splines

1. The code for the cubic splines was implemented. Shown below is the interpolation of the same function as the previous section.









Notice that the cubic interpolation displays similar error as the linear interpolation, through slightly lower for x near 1 or -1. As before, it does not display the Runge phenomenon.

4 Linear Splines Code

```
import matplotlib.pyplot as plt
import numpy as np
import math
from numpy.linalg import inv

def driver():
  f = lambda x: np.exp(x)
  a = 0
  b = 1
  # create points you want to evaluate at#
  Neval = 100
  xeval = np.linspace(a,b,Neval)
```

```
  # number of intervals#
  Nint = 10
  #evaluate the linear spline#
  yeval = eval_lin_spline(xeval,Neval,a,b,f,Nint)
  # evaluate f at the evaluation points#
  fex = f(xeval)
  plt.figure()
  plt.plot(xeval,fex,'ro-')
  plt.plot(xeval,yeval,'bs-')
  plt.legend()
  plt.show()
  err = abs(yeval-fex)
  plt.figure()
  plt.plot(xeval,err,'ro-')
  plt.show()

def question32(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_lin_spline(xeval, 1001, -1, 1, f, N)

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

    plt.plot(xeval, f(xeval), label="f(x)")
    plt.xlabel("x")
    plt.ylabel("p(x)")
    plt.title("Linear Spline Interpolation, N=" + str(N))
    plt.grid()
    plt.legend()

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

    plt.clf()
    plt.semilogy(xeval, np.abs(f(xeval) - yeval))
    plt.xlabel("x")
    plt.ylabel("Absolute Error")
    plt.title("Linear Spline Error, N=" + str(N))
    plt.grid()
    plt.legend()
    plt.savefig("lin_error" + str(N) + ".png")

def eval_line(x, a1, fa1, b1, fb1):
  return (x-a1) * (fb1-fa1) / (b1 - a1) + fa1

def eval_lin_spline(xeval,Neval,a,b,f,Nint):
  #create the intervals for piecewise approximations#
  xint = np.linspace(a,b,Nint+1)
  #create vector to store the evaluation of the linear splines#
  yeval = np.zeros(Neval)
  for jint in range(Nint):
    #find indices of xeval in interval (xint(jint),xint(jint+1))#
    #let ind denote the indices in the intervals#
    atmp = xint[jint]
```

```
    btmp= xint[jint+1]

    # find indices of values of xeval in the interval
    ind= np.where((xeval >= atmp) & (xeval <= btmp))
    xloc = xeval[ind]
    n = len(xloc)
    #temporarily store your info for creating a line in the interval of interest#
    fa = f(atmp)
    fb = f(btmp)
    yloc = np.zeros(len(xloc))

    for kk in range(n):
      #use your line evaluator to evaluate the spline at each location
      yloc[kk] = eval_line(xloc[kk], atmp, fa, btmp, fb)#Call your line evaluator with points (atmp,fa)

    # Copy yloc into the final vector
    yeval[ind] = yloc

  return yeval

question32(4)
question32(10)
question32(20)
```

5 Cubic Splines Code

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


def driver():

    f = lambda x: np.exp(x)
    a = 0
    b = 1

    ''' number of intervals '''
    Nint = 3
    xint = np.linspace(a,b,Nint+1)
    yint = f(xint)

    ''' create points you want to evaluate at '''
    Neval = 100
    xeval =  np.linspace(xint[0],xint[Nint],Neval+1)

#   Create the coefficients for the natural spline
    (M,C,D) = create_natural_spline(yint,xint,Nint)

#  evaluate the cubic spline
    yeval = eval_cubic_spline(xeval,Neval,xint,Nint,M,C,D)


    ''' evaluate f at the evaluation points '''
    fex = f(xeval)

    nerr = norm(fex-yeval)
```

```
    print('nerr = ', nerr)

    plt.figure()
    plt.plot(xeval,fex,'ro-',label='exact function')
    plt.plot(xeval,yeval,'bs--',label='natural spline')
    plt.legend()
    plt.show()

    err = abs(yeval-fex)
    plt.figure()
    plt.semilogy(xeval,err,'ro--',label='absolute error')
    plt.legend()
    plt.show()

def question34(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)
    (M,C,D) = create_natural_spline(yj, xj, N-1)

    #  evaluate the cubic spline
    yeval = eval_cubic_spline(xeval,1000,xj,N-1,M,C,D)

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

    plt.plot(xeval, f(xeval), label="f(x)")
    plt.xlabel("x")
    plt.ylabel("p(x)")
    plt.title("Cubic Spline Interpolation, N=" + str(N))
    plt.grid()
    plt.legend()

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

    plt.clf()
    plt.semilogy(xeval, np.abs(f(xeval) - yeval))
    plt.xlabel("x")
    plt.ylabel("Absolute Error")
    plt.title("Cubic Spline Error, N=" + str(N))
    plt.grid()
    plt.legend()
    plt.savefig("cubic_error" + str(N) + ".png")

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

    print(A)

#  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_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*(xeva
    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)

question34(4)
question34(10)
```

question34(20)