APPM 4600 Lab 8

Playing with splines

1 Overview

Splines are piecewise approximations. In class, we derived linear and cubic splines. In this lab, you will start developing your code for creating these approximations. You will build it in small pieces strengthening your coding skills.

2 Before lab

1. The `numpy` command `where` tells you the indices of a vector that satisfy a condition. For example, let `x = np.linspace(0,10,100)`. The numpy array `ind1` defined by `ind = np.where(x<=1)` has the entries `0,1,2,3,4,5,6,7,8,9`. You will use this command to build a subroutine that finds the indices of the points that live in a set of subintervals.

   Specifically, let `xeval = np.linspace(0,10,1000)` denote the vector of points where you want to evaluate your piecewise approximation. Let `xint = np.linspace(0,10,11)` denote the intervals in which you will have piecewise defined polynomials. Write a subroutine that finds the point of `xeval` that lie in each of the 10 subintervals that make up `xint`.

2. Write a subroutine that constructs and evaluates a line that goes through the points $(x_0, f(x_0))$ and $(x_1, f(x_1))$.

3 Lab Day: Building splines

During lab, you will build both a linear and cubic spline codes.

3.1 Constructing piecewise linear approximations

In this section, you will take your pre-lab codes and build a linear spline code. Below is a Python code that is missing the two subroutines that you made previously. The calls to your previously created subroutines in the subroutine named `eval_lin_spline`. You can download the template as a .py file from the lab assignment.

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


def driver():

    f = lambda x: math.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 = np.zeros(Neval)
    for j in range(Neval):
      fex[j] = f(xeval[j])


    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  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'''
        '''let n denote the length of ind'''

        '''temporarily store your info for creating a line in the interval of
         interest'''
        a1= xint[jint]
        fa1 = f(a1)
        b1 = xint[jint+1]
        fb1 = f(b1)

        for kk in range(n):
           '''use your line evaluator to evaluate the lines at each of the points
           in the interval'''
           '''yeval(ind(kk)) = call your line evaluator at xeval(ind(kk)) with
           the points (a1,fa1) and (b1,fb1)'''


if __name__ == '__main__':
      # run the drivers only if this is called from the command line
      driver()
```

3.2 Exercise

Consider the function

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

on the interval $[-1, 1]$. Perform the same experiments as in Lab 7 but with your linear spline evaluator. How does this perform? Is it better or worse than global interpolation with uniform nodes?

3.3 Constructing cubic splines

1. Create the linear system that you need to solve in order to identify the $\{M_i\}_{i=0}^{n-1}$ coefficients. The subroutine will take as input the functions values at the interpolation nodes and the interpolation nodes. You can start with a free boundary condition code.

2. Use the linear algebra package in Numpy (`inv` loaded at the top of the Python code) to solve for these coefficients.
   Make sure to validate that your coefficients have been evaluated correctly.

3. Write a subroutine that evaluates a cubic polynomial on a subinterval where the cubic has the form

$$S_i(x) = \frac{(x_{i+1}-x)^3 M_i}{6h_i} + \frac{(x-x_i)^3 M_{i+1}}{6h_i} + C(x_{i+1}-1) + D(x-x_i)$$

   where

$$\begin{aligned} C &= \frac{f(x_i)}{h_i} - \frac{h_i}{6}M_i \\ D &= \frac{f(x_{i+1})}{h_i} - \frac{h_i M_{i+1}}{6} \\ h_i &= x_{i+1} - x_i. \end{aligned}$$

   The necessary input for this subroutine is M_i, M_{i+1}, x_i, x_{i+1}, $f(x_i)$ and $f(x_{i+1})$. It will return one function evaluation.

4. Create a new spline evaluation code by mimicing the linear spline code from earlier in the lab. The cubic spline code will require approximately 2 changes: First creating the coefficients M_i and evaluating a cubic instead of a line.

3.4 Exercise

Consider the function

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

on the interval $[-1, 1]$. Perform the same experiments as in Lab 7 but with your cubic spline evaluator. How does this perform compared to all the other methods.

4 Deliverables

Report your solutions to the questions in the exercise section on Canvas, including some plots. Push your codes to Git as usual.