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. On the module for lab codes you can find a Python code (demo_linspline.py) that is missing the two subroutines that you made previously. Use your subroutine to write the evaluation subroutine `eval_lin_spline` called in this code.

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 mimicking 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.