APPM 4600 Lab 7
Exploring Interpolation

1 Overview

In class we proved that the polynomial that goes through the points $\{(x_j, f(x_j))\}_{j=0}^n$ where $f(x) \in \mathcal{C}^{(n+1)}[a,b]$ and $x_j \in [a,b]$ is unique. There a variety of ways of constructing this polynomial. In this lab, you will explore some of the different ways of constructing the polynomial and determine where sources of error are being introduced in the process. In the end, you will determine which is the more stable method.

Next, you will explore the interpolation error. Recall that the error has the polynomial $\Psi(x) = (x-x_0)\cdots(x-x_n)$. This is the part of the error that you as a user have control over. You will investigate different placement of interpolation nodes and how this impacts the error.

2 Before lab

Probably the most naïve way of constructing an interpolation polynomial of $f(x) \in \mathcal{C}^{(n+1)}[a,b]$, with data $\{(x_j, f(x_j))\}_{j=0}^n$, $x_j \in [a,b]$ is to write the polynomial down in terms of monomials and solve for the coefficients; i.e. write

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

and find the coefficients so that it goes through the interpolation points. For pre-lab, write down the linear system that you need to solve in order to find the coefficient $\{a_j\}_{j=0}^n$. The matrix that you need to invert is called the *Vandermonde* matrix. Come up with a plan for how you write the code that would solve for the coefficients and then evaluate the polynomial.

In your pre-lab write up include a summary of the remainder of lab.

You should also download the Lagrange interpolation code and the code that creates the Newton-Divided Difference coefficients.

3 Lab Day: Exploring interpolation

At this point between class and the pre-lab, you know 3 different techniques for constructing and evaluating an interpolation polynomial: monomial expansion, Lagrange polynomials, and Newton-Divided Differences. In this section, we will consider the task of interpolating the function

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

on the interval $[-1,1]$. Our interpolation nodes are given by $x_j = -1 + (j-1)h$ where $h = \frac{2}{N-1}$ for $j = 0, \ldots, N$.

3.1 Exercises: Different evaluation techniques

1. Develop codes for constructing and evaluating the evaluating the polynomial at the 1000 points uniformly distributed in the interval $[-1,1]$ via

   - monomial expansion

- Lagrange polynomials
- Newton-Divided Differences

You can use codes developed from lectures.

For each of the experiments below, plot both the approximations and the (log) absolute error.

2. Use your code to evaluate the polynomials for $N = 2, 3, 4, 5, \ldots, 10$. Which method performs the best? Do you see any difference in performance for the different methods? If so, what is causing the difference?

3. Continue constructing and evaluating $p(x)$ for $N = 11, \ldots$ until the maximum value of $p(x)$ is about 100. (This should be for $N \sim 17 - 20$). What is happening? Where is the error the largest? Do you see any differences in the evaluations of the polynomials now? If so, what is causing the difference?

4. Compare the log absolute error for these three routines for the function $numpy.sinc(5x)$. What differences and similarities do you observe? What is the largest N you can use before you observe issues in the error plot?

3.2 Exercises: Improving the approximation

In the exercises in the previous section, you should have seen that the error grew quite large near the end points of the interval. This behavior is called the *Runge* phenomena. There is a mechanism to prevent it from happening. Essentially all you have to do to is place more interpolation nodes towards the end points of the interval.

By doing this we are controlling the polynomial

$$(x - x_0) \cdots (x - x_n)$$

in the expression for the error.

1. Repeat the experiments from the previous section with the most stable of the interpolation techniques and different interpolation nodes. Specifically, let your interpolation nodes be defined as
$$x_j = \cos\left(\frac{(2j-1)\pi}{2N}\right)$$
for $j = 1, \ldots, N$.

2. How has the behavior changed?

3. When you plot the absolute error do you notice anything different than from the experiments in the previous section?

4. Once again, plot the log absolute error for this method for the function $numpy.sinc(5x)$. What differences and similarities do you observe?

4 Deliverables

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