OCEN 689
Fall 2019

Homework 1, due Tue Sept 17

1. Consider a function $f(x)$ and its values at three uniformly spaced points x_{i-1}, x_i, and x_{i+1}. Using the Lagrange interpolating polynomial over the interval $[x_{i-1}, x_{i+1}]$, derive an approximate formula for the first and second derivatives of $f(x)$ at x_i.

2. We would like to efficiently evaluate the Lagrange polynomial that interpolates $N+1$ data pairs (x_j, y_j), $j = 1, \ldots, N$:

$$P_N(x) = \sum_{j=0}^{N} y_j L_j(x). \tag{1}$$

   To this end, define the following:

$$\rho_j = \prod_{\substack{i=0 \\ i \neq j}}^{N} (x_j - x_i), \;\; j = 0, 1, \ldots, N,$$

$$\psi(x) = \prod_{i=0}^{N} (x - x_i).$$

   (a) Show that $L_j(x) = \psi(x) \dfrac{1}{\rho_j (x - x_j)}$.

   (b) Show that $P_N(x)$ can be written as:

$$P_N(x) = \psi(x) \sum_{j=0}^{N} \frac{y_j}{(x - x_j)\rho_j}. \tag{2}$$

   (c) Using Equation 2, we can now formulate an efficient method for the evaluation of the Lagrange polynomial $P_N(x)$ at any arbitrary point $x = \hat{x}$ as follows:

   - **Construction step**: Compute ρ_j for $j = 0, \ldots, N$.
   - **Evaluation step**: Compute $\psi(\hat{x})$ and then $P(\hat{x})$ using the result from part (b).

   State using the big-O notation how many flops (floating point multiplications, divisions, additions and subtractions) are required for each step? Which step is independent of the point $\hat{x}$ at which the Lagrange polynomial is evaluated and thus can be precomputed? How many flops are required for computing $P(\hat{x})$ using the standard formula given in Equation 1?

(d) Suppose we want to add a new data pair (x_{N+1}, y_{N+1}) and update the Lagrange interpolant. Using Equation 2, describe an efficient way to compute $P(\hat{x})$ for any arbitrary point $\hat{x}$. How many flops are required for this computation? How many flops are required for computing $P(\hat{x})$ for any arbitrary point $\hat{x}$ using the standard formula given in Equation 1?

(e) Using Equation 2, show that $P_N(x)$ can also be written as follows:

$$P_N(x) = \frac{\sum_{j=0}^{N} \frac{y_j}{(x-x_j)\rho_j}}{\sum_{j=0}^{N} \frac{1}{(x-x_j)\rho_j}}.$$

This is sometimes called the "second barycentric form of the interpolating polynomial" which shows that Lagrange interpolation can be viewed as a weighted interpolation scheme.

3. Consider the data set $\{(-3,-1),(-2,-1),(-1,-1),(0,0),(1,1),(2,1),(3,1)\}$. Use Matlab to fit and plot a Lagrange interpolating polynomial, cubic spline, and a monotonic cubic spline for this data set. Explain what you observe. You may find the Matlab command `pchip` useful for monotonic cubic spline interpolation.

4. This problem deals with under water propagation of sound. The speed of sound in ocean water depends on pressure, temperature and salinity, all of which vary with depth. Let z denote the depth in feet under the ocean surface and $c(z)$ denote the speed of sound at depth z. Table 1 is typical data showing the speed of sound as a function of depth:

z	$c(z)$
0	5,042
500	4,995
1,000	4,948
1,500	4,887
2,000	4,868
2,500	4,863
3,000	4,865
3,500	4,869
4,000	4,875

Table 1: Speed of sound vs. depth

(a) Using the steps shown in Problem 2(c), implement a Matlab code to interpolate the data with the Lagrange polynomial. Plot the polynomial and the data points. Use the polynomial to predict the speed of sound at a depth of 5000 feet.

(b) Using Matlab's `spline` command, interpolate the data using cubic spline interpolation and plot the polynomial and data points. Which one (Lagrange/cubic spline) do you think interpolates the given data better?

(c) In order to compute the path traced by a sound ray, we need the first derivative of speed of sound $c'(z)$. Using Matlab's `spline` command, compute and plot $c'(z)$ vs. z.

Appendix

This tutorial teaches you how to use Matlab routines (`spline`, `ppval`) for spline calculations, and how to manipulate the coefficients to get the first (and higher order) derivative of a spline approximation. For example, consider interpolating the data given in Table 2 using cubic spline.

x	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
f(x)	0	0.0998	0.1987	0.2955	0.3894	0.4794	0.5646	0.6442	0.7174	0.7833	0.8415

Table 2:

Type the following commands:

```
x=0:0.1:1;
y=[0 0.0998 0.1987 0.2955 0.3894 0.4794 0.5646 0.6442 0.7174 0.7833 0.8415];
S=spline(x,y)
```

and press enter to get the following output:

```
S =

     form: 'pp'
   breaks: [0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000 1]
    coefs: [10x4 double]
   pieces: 10
    order: 4
      dim: 1
```

The structure `S` contains all information about the spline; the data points $(x_0, x_1, \dots, x_n)$ are contained in the sub-structure `S.breaks`, which is an array of length $n\,(= 11)$ and the coefficients for the spline in each sub-interval are contained in the sub-structure `S.coefs`, which is a matrix of size $(n-1) \times 4$. To illustrate this, if $g_3(x)$ is the cubic in the third sub-interval $[x_3, x_4]$, then:

$$
\begin{aligned}
g_3(x) &= a_1(x-x_3)^3 + a_2(x-x_3)^2 + a_3(x-x_3) + a_4, \text{ where} \\
x_3 &= \texttt{S.breaks}(3), \\
a_1 &= \texttt{S.coefs}(3,1), \\
a_2 &= \texttt{S.coefs}(3,2), \\
a_3 &= \texttt{S.coefs}(3,3), \\
a_4 &= \texttt{S.coefs}(3,4).
\end{aligned}
$$

To obtain the first derivative of $f(x)$ over each sub-interval, you will have to write a user–defined function `dspline(S)`, which takes `S` as the input argument, modifies the `S.coefs` matrix accordingly and returns `S`. The modified structure `S` will now contain all information about the first derivative of the spline. Finally, to evaluate the cubic spline (and derivatives) for a given x, type `ppval(S,x)`.