MA/CS 321 : Intro to Numerical Methods

Problem Set 5

Due: Wednesday October 25, 2013

Note: Turn in all code that you have written and turn in all output generated by your functions/scripts. Your functions/scripts must be commented and output must be formatted nicely.

1. **Linear Spline interpolation and integration** For this problem you may adapt the code for linear splines that was generated in class.

   (a) (Adapted from E6.1.9) If the function \( f(x) = \exp(13x) \) is to be approximated on the interval \([0, 1]\) by an interpolating spline of degree 1, how many knots are needed to ensure that \(|S(x) - f(x)| \leq 10^{-8}\). (Use equally spaced knots). *Hint:* Use result from E6.1.7. For the number of knots you determine create a plot showing that \(|S(x) - f(x)|\) is indeed less than \(10^{-8}\) in \([0, \pi]\).

   (b) By hand determine \( \int_0^\pi f(x)dx \).

   (c) Approximating integrals by integrating linear splines corresponds to applying the composite trapezoid rule (See Section 5.1). Now if we wished to approximate the integral \(f(x)\) on the interval \([0, \pi]\), how many knots are needed to ensure that \(|\int_0^\pi S(x) - f(x)dx| \leq 10^{-8}\). *Hint:* Use the approach in Example 3 of Section 5.1. Using your calculation from part (b), for the number of knots you determined to show that \(|\int_0^\pi S(x) - f(x)dx|\) is indeed less than \(10^{-8}\).

   (d) Page 235 for the 7th edition and 225 of 6th edition of the book gives higher order rules for numerically integrating. The error for each rule is also given, note that the error for the quadrature must be used to derive the error for the corresponding composite quadrature rule. For each rule, repeat the previous problem to determine how many points are needed for each of the corresponding composite rules to approximately integrate \(f(x)\), so that the error is less than \(10^{-8}\). For each rule verify that the integral is approximated to the desired accuracy.

2. **Natural Cubic Spline interpolation**

   Equations (8) and (9) of section 6.2 in the book list the system of equations that must be solved in order to determine a natural cubic spline. Equation (5) of section 6.2 gives the form of each \(S_i\).

   (a) Write a program that determines the coefficients for each \(S_i\) polynomial using the system of equations mentioned above. The function header should look like

   ```matlab
   function [coeff] = ncubspline(x,y)
   % INPUTS
   % x and y contain the data (x(i),y(i))
   % the knots are located at the x(i)
   % OUTPUTS
   % coeff each row has the coefficients of a S_i
   % stored with highest order coefficient first, so that
   % each S_i(s) can be evaluated via polyval(coeff(i,:),s)
   ```
You may use your tridiagonal solver from homework 3 to solve the system.

(b) Create a jpeg image of your first name handwritten in cursive. Plot the image in mat-lab using

\[ A=\text{imread('yourimage'); imagesc(A).} \]

Generate \((x, y)\) data from your image using \texttt{ginput(N)}. (Note using \texttt{ginput} in octave may require that you hit enter after each click). Using your code from part (a) and the approach in the Space Curves subsection in Section 6.2 of the book, plot a natural cubic spline that passes through the points you collected. Submit a plot of the original image with spline plotted on top of it.

3. B-splines

(a) Write a function that evaluates a B-spline at a point \(x\). The function should implement the recursion in Equation 3 of Section 6.3. Note the base case for the recursion that \(B_0^0(x)\) is given in Equation 2. The header for the function should look like

```matlab
function y=Bspline(i,k,t,x)
% Evalutes ith Bspline of degree k determine by
% point \(t_{-k}, t_{-k+1}, \ldots t_{n-1}\)
% INPUTS
% i spline number
% k degree of spline
% t location of nodes
% total of \(n+k\) \(t = [t_{-k}, t_{-k+1} \ldots t_{n-1}]\)
% OUTPUT
% y value of \(B_i^k(x)\)
```

(b) Using your B-spline function, recreate the plot in figure 6.14, plotting \(B_k^0\) for \(k = 0, 1, 2, 3, 4\).

(c) Equation 8 section 6.3 gives a recursion for evaluating the first derivative of a B-spline. Explain how this expression could be used to evaluate higher derivatives.

(d) Write a function for evaluating derivatives of a B-spline \(\text{bsplineder}(i,k,t,x,d)\), where \(d\) is the desired derivative.

(e) To put your code from (a) and (d) to use to determine the natural cubic spline approximation to \(f(x) = \sin(x)\) in \([0, \pi]\) for thirty nodes using B-splines, Determine the natural cubic spline in the B-spline basis, by solving a linear system to determine the coefficients for \(S(x) = \sum_{i=-k}^{n-1} c_i B_i^k(x)\). There will be \(n+1\) equations for interpolating at the nodes, then there will be two equations for enforcing the natural cubic spline condition \(S''(t_0) = S''(t_n) = 0\). For the interpolating condition the entries in your matrix will correspond to \(B_j^k(t_i)\), that is each column correspond to a particular \(B_j^k\) and each row will correspond to a particular \(t_i\). For the natural cubic spline conditions you will need equations that involve \(\frac{d^2}{dx^2} B_i^k(t_0)\) or \(\frac{d^2}{dx^2} B_i^k(t_n)\) for the splines that have nonzero second derivatives near the boundary. Plot your computed natural cubic spline approximation to \(f\).