Graded Homework 3 Foundations of Computational Math 2 Spring 2022

The solutions must be submitted using the Canvas course page by 11:59PM on Sunday March 6, 2022.

Programming Exercise

Codes

1. **Interpolatory cubic spline**: In addition to the mesh points and the function $f(x)$ this code must accept as input the specification of two boundary conditions to be applied in the formation of $s(x)$. You must implement two spline codes. Spline Code 1 will use either the s_i' parameterization or the s_i'' parameterization. Spline Code 2 must use the parameterization in terms of the cubic B-spline basis.

2. Implement a code that evaluates the spline $s(x)$ and produced by your codes.

3. Implement the supporting code required to empirically validate the correct functioning of your codes and the tasks below.

Code Comments:

1. Your codes should be able to run in single or double precision (assumed to be IEEE standard FP).

2. Your codes must be efficient in time and space and make sure you discuss these aspects of your implementations.

3. You may use the parameterization of $s(x)$ in terms of the s_i'' or s_i' for Spline Code 1. This code must must accept data on a **nonuniform mesh**.

4. Spline Code 2 based on the B-spline basis may be restricted to a uniform mesh if you wish but you may implement the nonuniform mesh at your discretion. Note that if you implement a nonuniform mesh code then you must generalize the table in the notes that is used to set the coefficients in the equations.

5. Both spline codes should support both of the basic Hermite boundary conditions. That is, either the pair of values $(s''(x_0), s''(x_n))$ or the pair of values $(s'(x_0), s'(x_n))$ are specified. Depending on your choice of parameterization one of these cases may require the addition of two extra equations needed to specify boundary conditions in terms of the appropriate s_i'' or s_i' parameters. Your solution should include the derivation of these equations. For the Spline Code 2 based on the B-spline basis you should also explain how you handle the two types of boundary conditions.

Tasks

Empirically validate the correct functioning of your code.

- You must design experiments and describe the outcomes that provide evidence that your code is working. Evidence must also be given that you code works correctly for each of the required boundary conditions.

- This should include running your code to approximate carefully selected functions $g(x)$ for which the results are known. For example, if $g(x)$ is any polynomial of degree $d \leq 3$, the cubic spline, $s(x)$, should reproduce its value at any x. Similarly, if $g(x)$ is a piecewise cubic with intervals defined by the same mesh as $s(x)$ then $s(x)$ should match $g(x)$ for any x between x_0 and x_n.

- Compare the functioning of the two spline codes to each other.

- Once this has been done for specific functions, randomly generating similar functions and summarizing the results can produce good evidence of correct functioning.

- The rate of convergence can be estimated empirically from estimated values of $\|f(x) - s(x)\|_\infty$ as $h \to 0$. So if you have an error estimate $E(h)$ and $E(h/2)$ based on a finer grid with the intervals halved, then the rate of convergence can be estimated by

$$\log_2 \frac{|E(h)|}{|E(h/2)|}$$

  which should converge to 4 for the spline codes (see the posted papers on spline convergence and the summary in the notes). You can estimate the rate of convergence for the first and second derivative as well. See Table 8.5 on p. 361 of the textbook for an example of estimating a cubic spline convergence rate. You should be able to reproduce it and similar results. You are also encouraged to compare the convergence with that observed for you piecewise polynomial codes from the earlier graded homework.

- You can also check you spline results against other libraries. **Make sure however that you know exactly what the other library is choosing for boundary conditions etc. You must compare splines that are known to be the same in exact arithmetic.** Care should also be taken with respect to what you consider the "answer" for the other library's results. If you print them and then copy to your code you have modified the results so that they print in ASCII format. This is not the answer's full single or double precision representation. The most reliable way is to output to a file raw binary IEEE single or double precision results and read them into your code using the appropriate format in the high-level language of your code. This is a technique of input/output that you should develop skill in using. You are not required to use it here but make sure you compare an appropriately precise output value if you use file output to compare your results to a library code.

Solving the Linear Systems

Construction of the splines requires the solution of a linear system of equations that is tridiagonal or nearly tridiagonal. You are encouraged to implement or use a code of your own. However, you may apply any numerical linear algebra library you choose, e.g., LAPACK, to find a subroutine to solve such systems. For tridiagonal matrices Thomas' algorithm is described in the textbook. This algorithm is essentially an LU factorization. This could also be used as a piece of an algorithm to solve systems with matrices that are tridiagonal in all but a small number, $O(1)$, number of rows. You must document your approach and cite references appropriately.