Cubic Spline Interpolation of Periodic Functions

A Project for MATH 5093

Cubic spline interpolation is an approximate representation of a function whose values are known at a finite set of points, by using cubic polynomials.

The setup is the following (for more details see Sec. 5.6 of the textbook, as well as Sec. 3.4 of the book *Numerical Analysis*, 8th edition, by R. L. Burden and J. D. Faires, pages of which I sent you). Let f be a function defined on the interval $[a, b]$, and let

$$a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b \tag{1}$$

be $n + 1$ distinct points at which the values of the function f are known. The points x_j divide the interval $[a, b]$ into n subintervals, referred to as a *partition* of $[a, b]$.

A *cubic spline interpolant* of f relative to the partition (1) is a function $S : [a, b] \to \mathbb{R}$ that satisfies the following properties:

(1) the restriction $S|_{[x_j, x_{j+1}]} : [x_j, x_{j+1}] \to \mathbb{R}$ of the interpolant S to the interval $[x_j, x_{j+1}]$ coincides with the cubic polynomial

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3 \ , \qquad j = 0, 1, \ldots, n - 1 \ ;$$

(2) the function S interpolates f at the points x_0, x_1, ..., x_n, i.e., $S_j(x_j) = f(x_j)$ and $S_j(x_{j+1}) = f(x_{j+1})$ for $j = 0, 1, \ldots, n - 1$;

(3) the function S is continuous, i.e.,

$$S_{j+1}(x_{j+1}) = S_j(x_{j+1}) \ , \qquad j = 0, 1, \ldots, n - 2 \ ;$$

(3) the derivative S' is continuous, i.e.,

$$S'_{j+1}(x_{j+1}) = S'_j(x_{j+1}) \ , \qquad j = 0, 1, \ldots, n - 2 \ ;$$

(3) the second derivative S'' is continuous, i.e.,

$$S''_{j+1}(x_{j+1}) = S''_j(x_{j+1}) \ , \qquad j = 0, 1, \ldots, n - 2 \ ;$$

(4) the interpolant S satisfies some boundary conditions, i.e., conditions at the ends of the interval $[a, b]$.

In this project you will develop cubic spline interpolation of periodic functions. Without loss of generality, you can assume that the period of a periodic function is 1, i.e., that

$$f(x + 1) = f(x) \quad \text{for all } x \in \mathbb{R} \ .$$

Because of the periodicity, the function f is completely defined by its values on the interval $[0,1]$, so below assume that $[a,b]=[0,1]$. We use the notation $h_j = x_{j+1} - x_j$.

(A) Formulate the conditions above in the case of a cubic spline of a periodic function. In this case the boundary conditions are provided by the condition of periodicity of f.

Hint: This case is in some sense easier than the cases of free, clamped, or not-a-knot splines because in the periodic case there are no boundary conditions, in the sense that the boundary points are just like the points inside the interval $[a,b]$. Because of the periodicity $f(0)=f(1)$, $f'(0)=f'(1)$, $f''(0)=f''(1)$. What do these equalities imply about the pairs of numbers $S(0+)$ and $S(1-)$, $S'(0+)$ and $S'(1-)$, $S''(0+)$ and $S''(1-)$? How about the pairs of numbers $S_0(0)$ and $S_{n-1}(1)$, $S_0'(0)$ and $S_{n-1}'(1)$, $S_0''(0)$ and $S_{n-1}''(1)$? Are this conditions similar to the matching conditions at the internal points x_1, …, x_{n-1}?

(B) Read the derivation of the equations for the coefficients a_j, b_j, c_j, and d_j of the cubic polynomial S_j in the case of free (natural) boundary conditions and clamped boundary conditions from Sec. 3.4 of Burden-Faires (namely, pages 140–142 and 145–146). In these two cases (as well as in the so-called not-a-knot boundary conditions, explained on pages 395 and 396 of Bradie), the coefficients a_j, b_j and d_j are expressed in terms of the coefficients c_j. In each case, the coefficients c_j satisfy a linear system with tridiagonal structure, so solving the system for the c_j's requires only $\mathrm{O}(n)$ operations (as shown on page 219 of Bradie). Once c_j are found, it is easy to compute the other spline coefficients.

Show that in the case of cubic spline interpolation of periodic functions, the vector of the coefficients $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})^T$ is the solution of the linear system

$$A\,\boldsymbol{\xi} = \boldsymbol{\delta}\ , \tag{2}$$

where

$$A = \begin{pmatrix} 2(h_{n-1}+h_0) & h_0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & h_{n-1} \\ h_0 & 2(h_0+h_1) & h_1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & h_1 & 2(h_1+h_2) & h_2 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & h_2 & 2(h_2+h_3) & h_3 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 2(h_{n-4}+h_{n-3}) & h_{n-3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ h_{n-1} & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{pmatrix}$$

$$\boldsymbol{\xi} = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_4 \\ \vdots \\ \xi_{n-2} \\ \xi_{n-1} \\ \xi_n \end{pmatrix}, \qquad \boldsymbol{\delta} = \begin{pmatrix} \frac{3}{h_0}(a_1-a_0) - \frac{3}{h_{n-1}}(a_0-a_{n-1}) \\ \frac{3}{h_1}(a_2-a_1) - \frac{3}{h_0}(a_1-a_0) \\ \frac{3}{h_2}(a_3-a_2) - \frac{3}{h_1}(a_2-a_1) \\ \frac{3}{h_3}(a_4-a_3) - \frac{3}{h_2}(a_3-a_2) \\ \vdots \\ \frac{3}{h_{n-3}}(a_{n-2}-a_{n-3}) - \frac{3}{h_{n-4}}(a_{n-3}-a_{n-4}) \\ \frac{3}{h_{n-2}}(a_{n-1}-a_{n-2}) - \frac{3}{h_{n-3}}(a_{n-2}-a_{n-3}) \\ \frac{3}{h_{n-1}}(a_0-a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1}-a_{n-2}) \end{pmatrix};$$

where $a_j = f(x_j)$, $j = 0, \ldots, n-1$ (following the notations of Bradie and Burden-Faires). You are allowed to use any intermediate results from the Bradie and Burden-Faires derivations, there is no need to derive everything from scratch. In fact, if you answered **(A)**, you can avoid doing *any* calculations, just look at the equations for the spline coefficient at the internal points for the case of the free or clamped splines.

(C) The $n \times n$ matrix A is "almost" tridiagonal – its only entries that violate the tridiagonal structure are the $(1, n)$ and $(n, 1)$ entries (both of which are equal to h_{n-1}). This prevents you from using a program that solves a tridiagonal system, but in this particular case there is a very efficient algorithm that allows solving a linear system with coefficient matrix with such structure by only $\mathrm{O}(n)$ operations.

An important fact about the matrix A from (2) is that it is *strictly diagonally dominant* (see the definition on page 211 on Bradie), which implies, in particular, that the system (2) has a unique solution, and that Gaussian elimination can be performed without row interchanges (see the theorem on page 211 of Bradie). Since A is a very structured and sparse matrix ("sparse" means that many of its entries are zero), the system (2) can be solved with only $\mathrm{O}(n)$ operations, and *without pivoting*!

In this part of the problem you have to write a MATLAB code called `tridiag_corners.m` that uses a simple and efficient algorithm for solving a linear system with the same structure as (2). Consider the linear system with augmented matrix

$$\left(\begin{array}{cccccccccc|c} \beta_1 & \gamma_1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \alpha_1 & \delta_1 \\ \alpha_2 & \beta_2 & \gamma_2 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \delta_2 \\ 0 & \alpha_3 & \beta_3 & \gamma_3 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \delta_3 \\ 0 & 0 & \alpha_4 & \beta_4 & \gamma_4 & 0 & \cdots & 0 & 0 & 0 & 0 & \delta_4 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & \alpha_{n-2} & \beta_{n-2} & \gamma_{n-2} & 0 & \delta_{n-2} \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & \alpha_{n-1} & \beta_{n-1} & \gamma_{n-1} & \delta_{n-1} \\ \gamma_n & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \alpha_n & \beta_n & \delta_n \end{array}\right), \tag{3}$$

where the coefficient matrix is strictly diagonally dominant (as in (2)).

Perform elementary row operations of type ERO_3 in the notations of Bradie (see page 150 of his book) to transform this augmented matrix (3) to the form

$$\left(\begin{array}{cccccccccc|c} \widetilde{\beta}_1 & \widetilde{\gamma}_1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \widetilde{\alpha}_1 & \widetilde{\delta}_1 \\ 0 & \widetilde{\beta}_2 & \widetilde{\gamma}_2 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \widetilde{\alpha}_2 & \widetilde{\delta}_2 \\ 0 & 0 & \widetilde{\beta}_3 & \widetilde{\gamma}_3 & 0 & 0 & \cdots & 0 & 0 & 0 & \widetilde{\alpha}_3 & \widetilde{\delta}_3 \\ 0 & 0 & 0 & \widetilde{\beta}_4 & \widetilde{\gamma}_4 & 0 & \cdots & 0 & 0 & 0 & \widetilde{\alpha}_4 & \widetilde{\delta}_4 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & \widetilde{\beta}_{n-2} & \widetilde{\gamma}_{n-2} & \widetilde{\alpha}_{n-2} & \widetilde{\delta}_{n-2} \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \widetilde{\beta}_{n-1} & \widetilde{\alpha}_{n-1} & \widetilde{\delta}_{n-1} \\ \gamma_n & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \alpha_n & \beta_n & \delta_n \end{array}\right). \tag{4}$$

The tildes signify only that these elements may differ from the elements without tildes. Note that we have not changed the nth row of the augmented matrix. To be memory-efficient, overwrite the elements α_j by $\widetilde{\alpha}_j$ (for $j = 1, \ldots, n-1$), etc. Then perform elementary row operations to transform (4) to the form

$$\left(\begin{array}{cccccccccccc|c} \bar{\beta}_1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \bar{\alpha}_1 & \bar{\delta}_1 \\ 0 & \bar{\beta}_2 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \bar{\alpha}_2 & \bar{\delta}_2 \\ 0 & 0 & \bar{\beta}_3 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \bar{\alpha}_3 & \bar{\delta}_3 \\ 0 & 0 & 0 & \bar{\beta}_4 & 0 & 0 & \cdots & 0 & 0 & 0 & \bar{\alpha}_4 & \bar{\delta}_4 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & \bar{\beta}_{n-2} & 0 & \bar{\alpha}_{n-2} & \bar{\delta}_{n-2} \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \bar{\beta}_{n-1} & \bar{\alpha}_{n-1} & \bar{\delta}_{n-1} \\ \gamma_n & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \alpha_n & \beta_n & \delta_n \end{array}\right) \tag{5}$$

(again, the bars simply mean that the corresponding elements may have changed their numerical values). Note that the nth row of (5) is still the same as in (3).

Finally, use elementary row operations to make the coefficient matrix in (5) upper-triangular, and then use back substitution to solve the system. Since there are many zeros in the transformed coefficient matrix, make sure that you do everything as efficiently as possible.

Your MATLAB code `tridiag_corners.m` should take n-dimensional arrays $\boldsymbol{\alpha} = (\alpha_j)$, $\boldsymbol{\beta} = (\beta_j)$, $\boldsymbol{\gamma} = (\gamma_j)$ and $\boldsymbol{\delta} = (\delta_j)$ as input variables, and should produce the solution $\mathbf{x} = (x_j)$ of the linear system with augmented matrix (3). As a model you can use the MATLAB code `tridiagonal.m` available at the class web-site. Note that, if you are testing your code that solves (3), the coefficient matrix in (3) should be strictly diagonally dominant (which in this case means that $|\beta_j| > |\alpha_j| + |\gamma_j|$).

(D) Write a MATLAB code `cubic_spline_periodic.m` that performs cubic interpolation of a periodic function of period 1. Your code should takes the values $0 = x_0$, x_1, …, x_{n-1} (as in see (1)) and the values $y_0 = f(x_0)$, $y_1 = f(x_1)$, …, $y_{n-1} = f(x_{n-1})$ of the periodic function f at each of these points. Because of the periodicity of f, you do not need to use the point $x_n = 1$.

Use that $a_j = f(x_j)$ (for $j = 0, \ldots, n-1$), $h_j = x_{j+1} - x_j$ (for $j = 0, \ldots, n-1$, with $x_n = 1$), then set up the linear system (2) and solve it by calling your code `tridiag_corners.m`, after which compute the values of the coefficients b_j and d_j (for $j = 0, \ldots, n-1$). Your code should return a matrix with five columns, containing the values of x_j, a_j, b_j, c_j, and d_j, respectively. Be careful with the indices because MATLAB starts with index 1!

In creating `cubic_spline_periodic.m` you can use as a model the code `cubic_clamped.m` (available at the class web-site). That code computes the coefficients of the cubic spline interpolant S in the case of clamped boundary conditions, i.e., when we know the derivatives of the function f at the endpoints a and b. The input are the vector $\mathbf{x} = (x_i)$ of the values of the argument at which the function f is known, the vector $\mathbf{y} = (y_i) = (f(x_i))$ of the values of the function f at the points x_i, and the values of the derivatives $f'(a)$ and $f'(b)$. The

output is a five-column matrix containing the information that defines the clamped cubic spline interpolant (namely, x_i in the first column, $a_i = y_i$ in the second, b_i in the third, c_i in the fourth, and d_i in the fifth). The code `cubic_clamped.m` calls the code `tridiagonal.m` to compute the spline coefficients c_j.

(E) Write a MATLAB code `spline_periodic_eval.m` that takes the matrix with the spline coefficients produced by `cubic_spline_periodic.m`, and a value $z \in [0,1]$ (or several values $\{z_k\} \subset [0,1]$), and returns the value $S(z)$ (respectively, the values of S at each of the points z_k). As a model you can use the MATLAB code `spline_eval.m` (available at the class web-site) which does the same, but for the case of a clamped cubic spline, taking the spline coefficients produced by the code `cubic_clamped.m`.

Here is an example of using the codes `cubic_clamped.m` (which calls `tridiagonal.m`) and `spline_eval.m`. In this example $f = \sin\frac{\pi x}{2}$, $[a,b] = [0,1]$, $f'(0) = \frac{\pi}{2}$, $f'(1) = 0$, and the cubic spline S interpolates the function f at the points 0, 0.25, 0.5, 0.75, and 1:

```
xx = linspace(0,1,5);     % creates the array xx = ( 0 0.25 0.5 0.75 1 )
yy = sin(pi/2*xx);        % computes the values of f(xx_i)
fpa = pi / 2.0;           % derivative f'(0) at the left end
fpb = 0.0;                % derivative f'(1) at the right end
csc = cubic_clamped( xx, yy, fpa, fpb);    % spline coefficients
y_exact = sin(pi/2*0.7)                    % exact value f(0.7)
y_approx = spline_eval (csc, 0.7)          % interpolated value S(0.7)
abs(y_exact - y_approx)                    % absolute error at 0.7
xx_dense = linspace(0,1,1001);             % array of argument values
yy_exact = sin(pi/2*xx_dense);             % f(xx_k)
yy_approx = spline_eval (csc, xx_dense);   % S(xx_k)
plot(xx_dense, yy_exact - yy_approx);      % plotting f(xx_k)-S(xx_k)
[xx_dense; yy_exact; yy_approx; yy_exact-yy_approx]'    % comparison
```

(F) Run the codes `cubic_spline_periodic.m` and `spline_periodic_eval.m` with $n = 10$, $n = 50$, and $n = 250$ points $(x_j, f(x_j))$ (where $x_j = j/n$ for $j = 0, \ldots, n-1$), for the periodic function $f(x) = \sin(2\pi x) - \frac{1}{2}\cos(6\pi x) - \frac{1}{3}\sin(10\pi x)$, and find empirically how the global absolute error depends on $h = \frac{1}{n}$. In each case, estimate the global absolute error as

$$\max_{m=0,\ldots,M-1} |f(\mu_m) - S(\mu_m)| \ ,$$

where M is some very large integer (say, $M = 10^6$ or 10^7), and $\mu_m = m/M$ for $m = 0, \ldots, M$. Compare your findings with the result about clamped cubic spline on page 402 of Bradie's book.