Answers to Homework 6: Interpolation: Spline Interpolation

---

1. In class, we interpolated the function $f(x) = \frac{1}{x}$ at the points $x = 2, 4, 5$ with the cubic spline that satisfied the *natural* boundary conditions

$$S''(a) = 0\,; \tag{1}$$
$$S''(b) = 0 \tag{2}$$

for $a = 2$ and $b = 5$.

(a) Change conditions (1-2) to the *clamped* boundary conditions

$$S'(a) = f'(a)\,; \tag{3}$$
$$S'(b) = f'(b)\,, \tag{4}$$

find the corresponding cubic spline and evaluate it at $x = 3$. Is the result more accurate than the one of the natural cubic spline interpolation?

Note: No programming is necessary, but a calculator might help.

Solution: Let the cubic spline in the interval from $x = 2$ to $x = 4$ be the polynomial

$$S_1(x) = 0.5 + b_1\,(x-2) + c_1\,(x-2)^2 + d_1\,(x-2)^3$$

and the spline in the interval from $x = 4$ to $x = 5$ be the polynomial

$$S_2(x) = 0.25 + b_2\,(x-4) + c_2\,(x-4)^2 + d_2\,(x-4)^3\,.$$

The six coefficients $b_1, c_1, d_1, b_2, c_2, d_2$ are the unknowns that we need to determine. From the interpolation conditions, we get

$$\begin{aligned} S_1(4) &= 0.5 + 2\,b_1 + 4\,c_1 + 8\,d_1 = f(4) = 0.25\,; \\ S_2(5) &= 0.25 + b_2 + c_2 + d_2 = f(5) = 0.2\,. \end{aligned}$$

From the smoothness conditions at the internal point, we get

$$\begin{aligned} S_1'(4) = b_1 + 2\,c_1\,(4-2) + 3\,d_1\,(4-2)^2 &= S_2'(4) = b_2\,; \\ S_1''(4) = 2\,c_1 + 6\,d_1\,(4-2) &= S_2''(4) = 2\,c_2\,. \end{aligned}$$

Finally, from the boundary conditions, we get

$$\begin{aligned} S_1'(2) = b_1 &= f'(2) = -0.25\,; \\ S_2'(5) = b_2 + 2\,c_2 + 3\,d_2 &= f'(5) = -0.04\,. \end{aligned}$$

Thus, we have six linear equations to determine the six unknowns. In the matrix form, the equations are

$$\begin{bmatrix} 2 & 4 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 4 & 12 & -1 & 0 & 0 \\ 0 & 2 & 12 & 0 & -2 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} b_1 \\ c_1 \\ d_1 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} -0.25 \\ -0.05 \\ 0 \\ 0 \\ -0.25 \\ -0.04 \end{bmatrix} .$$

The equations can be solved, for example, by successive elimination of unknowns. We get $b_1 = -0.25$, then

$$\begin{bmatrix} 4 & 8 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 4 & 12 & -1 & 0 & 0 \\ 2 & 12 & 0 & -2 & 0 \\ 0 & 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} c_1 \\ d_1 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} 0.25 \\ -0.05 \\ 0.25 \\ 0 \\ -0.04 \end{bmatrix} .$$

Take $c_1 = 0.0625 - 2\,d_1$, then

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 4 & -1 & 0 & 0 \\ 8 & 0 & -2 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} d_1 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} -0.05 \\ 0 \\ -0.125 \\ -0.04 \end{bmatrix} .$$

Take $d_1 = 0.25\,b_2$, then

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & -2 & 0 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} b_2 \\ c_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} -0.05 \\ -0.125 \\ -0.04 \end{bmatrix} .$$

Take $b_2 = -0.0625 + c_2$, then

$$\begin{bmatrix} 2 & 1 \\ 3 & 3 \end{bmatrix} \begin{bmatrix} c_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} 0.0125 \\ 0.0225 \end{bmatrix} .$$

Finally, take $c_2 = 0.00625 - 0.5\,d_2$ and get $d_2 = 0.0025$. The final answer is

$$\begin{aligned} d_2 &= 0.0025 \\ c_2 &= 0.005 \\ b_2 &= -0.0575 \\ d_1 &= -0.014375 \\ c_1 &= 0.09125 \\ b_1 &= -0.25 \end{aligned}$$

Evaluating the spline at $x = 3$, we get

$$S(3) = S_1(3) = 0.5 + b1 + c_1 + d_1 = 0.326875 .$$

This is closer to the exact result $f(3) = 0.3333\ldots$ than the result of the natural spline interpolation ($S(3) = 0.35625$).

(b) Prove that if $S(x)$ is a cubic spline that interpolates a function $f(x) \in C^2[a,b]$ at the knots $a = x_1 < x_2 < \cdots < x_n = b$ and satisfies the clamped boundary conditions (3-4), then

$$\int_a^b \left[S''(x)\right]^2 dx \le \int_a^b \left[f''(x)\right]^2 dx \; . \tag{5}$$

Hint: Divide the interval $[a,b]$ into subintervals and use integration by parts in each subinterval.

Solution: Let us form the difference $D(x) = f(x) - S(x)$. From the integral equality

$$\int_a^b \left[f''(x)\right]^2 dx = \int_a^b \left[S''(x)\right]^2 dx + \int_a^b \left[D''(x)\right]^2 dx + 2\int_a^b S''(x)\,D''(x)\,dx \; ,$$

we can see that the theorem will be proved if we can prove that

$$\int_a^b S''(x)\,D''(x)\,dx = 0 \; .$$

Indeed, the integral

$$\int_a^b \left[f''(x)\right]^2 dx$$

in this case will be equal to the integral

$$\int_a^b \left[S''(x)\right]^2 dx$$

plus some non-negative quantity

$$\int_a^b \left[D''(x)\right]^2 dx \; .$$

Applying integration by parts, we get

$$\int_a^b S''(x)\,D''(x)\,dx = S''(x)D'(x)\Big|_a^b - \int_a^b S'''(x)\,D'(x)\,dx \; .$$

The first term is zero because of the clamped boundary conditions:

$$\begin{aligned} D'(a) &= f'(a) - S'(a) = 0 \; ; \\ D'(b) &= f'(b) - S'(b) = 0 \; . \end{aligned}$$

The integral in the second term can be divided into subintervals, as follows:

$$-\int_a^b S'''(x)\,D'(x)\,dx = -\sum_{k=1}^{n-1} \int_{x_k}^{x_{k+1}} S'''(x)\,D'(x)\,dx \; .$$

Integration by parts in each subinterval produces

$$\int_{x_k}^{x_{k+1}} S'''(x)\,D'(x)\,dx = S'''(x)\,D(x)\Big|_{x_k}^{x^{k+1}} - \int_{x_k}^{x_{k+1}} S^{(4)}(x)\,D(x)\,dx\ .$$

The first term in the expression above is zero because of the interpolation condition

$$D(x_k) = f(x_k) - S(x_k) = 0\ , \quad k = 1,2,\ldots,n\ .$$

The second term is zero because the spline $S(x)$ in each subinterval is a cubic polynomial and has zero fourth derivative.

We have proved that

$$\int_a^b S''(x)\,D''(x)\,dx = 0\ ,$$

which proves the theorem.

2. The natural boundary conditions for a cubic spline lead to a system of linear equations with the tridiagonal matrix

$$\begin{bmatrix} 2\,(h_1+h_2) & h_2 & 0 & \cdots & 0 \\ h_2 & 2\,(h_2+h_3) & h_3 & \ddots & \vdots \\ 0 & h_3 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & h_{n-2} \\ 0 & \cdots & 0 & h_{n-2} & 2\,(h_{n-2}+h_{n-1}) \end{bmatrix}, \tag{6}$$

where $h_k = x_{k+1} - x_k$. The textbook shows that the clamped boundary conditions lead to the matrix

$$\begin{bmatrix} 2\,h_1 & h_1 & 0 & \cdots & \cdots & 0 \\ h_1 & 2\,(h_1+h_2) & h_2 & 0 & & \vdots \\ 0 & h_2 & 2\,(h_2+h_3) & & & \vdots \\ \vdots & 0 & & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & h_{n-2} & 0 \\ \vdots & & \ddots & h_{n-2} & 2\,(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & \cdots & \cdots & 0 & h_{n-1} & 2\,h_{n-1} \end{bmatrix}. \tag{7}$$

Find the form of matrices that correspond to two other popular types of boundary conditions:

(a) "not a knot" conditions:

$$S_1'''(x_2) = S_2'''(x_2)\,; \tag{8}$$
$$S_{n-2}'''(x_{n-1}) = S_{n-1}'''(x_{n-1})\,. \tag{9}$$

(b) periodic conditions:

$$S_1'(x_1) = S_{n-1}'(x_n)\,; \tag{10}$$
$$S_1''(x_1) = S_{n-1}''(x_n)\,. \tag{11}$$

Here $S_k(x)$ represent the spline function on the interval from x_k to x_{k+1}, $k = 1,2,\dots,n-1$. The periodic conditions are applied when $S(x_1) = S(x_n)$.

Solution: The central part of the matrix will always have the same tridiagonal structure, which results from the recursive relationship

$$c_{k-1}\,h_{k-1} + 2\,c_k\,(h_{k-1}+h_k) + c_{k+1}\,h_k = 3\,(\,f[x_k,x_{k+1}] - f[x_{k-1},x_k])\ ,$$

where c_k is the second-order coefficient in the spline expression

$$S_k(x) = f_k + b_k\,(x-x_k) + c_k\,(x-x_k)^2 + d_k\,(x-x_k)^3\ , \quad x_k \le x \le x_{k+1}\ , k = 1,2,\dots,n-1$$

The boundary conditions will only affect the first and the last rows of the matrix.

(a) The “not a knot” conditions transform into the equations

$$\begin{aligned} d_1 &= d_2\ ; \\ d_{n-2} &= d_{n-1}\ . \end{aligned}$$

Using the recursive relationship

$$d_k = \frac{c_{k+1}-c_k}{3\,h^k}\ ,$$

where $h_k = x_{k+1} - x_k$, the conditions further transform to

$$\begin{aligned} \frac{c_2-c_1}{h_1} &= \frac{c_3-c_2}{h_2}\ ; \\ \frac{c_{n-1}-c_{n-2}}{h_{n-2}} &= \frac{c_n-c_{n-1}}{h_{n-1}}\ , \end{aligned}$$

where $c_n = S''_{n-1}(x_n)/2$. Using this two conditions, we can eliminate c_1 and c_n from the system with the help of the expressions

$$\begin{aligned} c_1 &= c_2\left(1+\frac{h_1}{h_2}\right) - c_3\,\frac{h_1}{h_2}\ ; \\ c_n &= c_{n-1}\left(1+\frac{h_{n-1}}{h_{n-2}}\right) - c_{n-2}\,\frac{h_{n-1}}{h_{n-2}}\ . \end{aligned}$$

The first equation in the system is then

$$c_1 h_1 + 2\,c_2\,(h_1+h_2) + c_3\,h_2 = c_2\left(3\,h_1 + 2\,h_2 + \frac{h_1^2}{h_2}\right) + c_3\left(h_2 - \frac{h_1^2}{h_2}\right) = 3\,(\,f[x_2,x_3] - f[x_1,x_2])\ ,$$

and the last equation is

$$\begin{aligned} c_{n-2}h_{n-2} + 2\,c_{n-1}\,(h_{n-2}+h_{n-1}) + c_n\,h_{n-1} &= \\ c_{n-2}\left(h_{n-2} - \frac{h_{n-1}^2}{h_{n-2}}\right) + c_{n-1}\left(3\,h_{n-1} + 2\,h_{n-2} + \frac{h_{n-1}^2}{h_{n-2}}\right) &= 3\,(\,f[x_{n-1},x_n] - f[x_{n-2},x_{n-1}])\ . \end{aligned}$$

The matrix takes the form

$$\begin{bmatrix} 3h_1+2h_2+\frac{h_1^2}{h_2} & h_2-\frac{h_1^2}{h_2} & 0 & \cdots & 0 \\ h_2 & 2(h_2+h_3) & h_3 & \ddots & \vdots \\ 0 & h_3 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & h_{n-2} \\ 0 & \cdots & 0 & h_{n-2}-\frac{h_{n-1}^2}{h_{n-2}} & 3h_{n-1}+2h_{n-2}+\frac{h_{n-1}^2}{h_{n-2}} \end{bmatrix}.$$

Alternative forms are possible.

(b) The periodic boundary conditions lead to the equations

$$\begin{aligned} b_1 &= b_{n-1}+2c_{n-1}h_{n-1}+3d_{n-1}h_{n-1}^2 \\ c_1 &= c_n \end{aligned}$$

After eliminating c_n from the system, the first equation transforms to

$$f[x_1,x_2]-\frac{2c_1+c_2}{3}h_1 = f\left[x_{n-1},x_n\right]+\frac{2c_{n-1}+c_1}{3}h_{n-1}$$

or

$$2c_1(h_1+h_{n-1})+c_2h_1+c_{n-1}h_{n-1} = 3\left(f[x_1,x_2]-f\left[x_{n-1},x_n\right]\right).$$

The system matrix is

$$\begin{bmatrix} 2(h_1+h_{n-1}) & h_1 & 0 & \cdots & 0 & h_{n-1} \\ h_1 & 2(h_1+h_2) & h_2 & \ddots & & 0 \\ 0 & h_2 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & & \ddots & \ddots & \ddots & h_{n-2} \\ h_{n-1} & 0 & \cdots & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{bmatrix}.$$

Alternative forms are possible.

3. The algorithm for solving tridiagonal symmetric systems, presented in class, decomposes a symmetric tridiagonal matrix into a product of lower and upper bidiagonal matrices, as follows:

$$\begin{bmatrix} a_1 & b_1 & 0 & \cdots & 0 \\ b_1 & a_2 & b_2 & \ddots & \vdots \\ 0 & b_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & b_{n-1} \\ 0 & \cdots & 0 & b_{n-1} & a_n \end{bmatrix} = \begin{bmatrix} \alpha_1 & 0 & \cdots & \cdots & 0 \\ b_1 & \alpha_2 & \ddots & & \vdots \\ 0 & b_2 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & b_{n-1} & \alpha_n \end{bmatrix} \begin{bmatrix} 1 & \beta_1 & 0 & \cdots & 0 \\ 0 & 1 & \beta_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \beta_{n-1} \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix}$$

The algorithm for solving the linear system

$$\begin{bmatrix} a_1 & b_1 & 0 & \cdots & 0 \\ b_1 & a_2 & b_2 & \ddots & \vdots \\ 0 & b_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & b_{n-1} \\ 0 & \cdots & 0 & b_{n-1} & a_n \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ \vdots \\ g_n \end{bmatrix}$$

is summarized below.

TRIDIAGONAL$(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_{n-1}, g_1, g_2, \ldots, g_n)$

$\alpha_1 \leftarrow a_1$
for $k \leftarrow 1, 2, \ldots, n-1$
do
  $\beta_k \leftarrow b_k / \alpha_k$
  $\alpha_{k+1} \leftarrow a_{k+1} - b_k \, \beta_k$
$c_1 \leftarrow g_1$
for $k \leftarrow 2, 3, \ldots, n$
do
  $c_k \leftarrow g_k - \beta_{k-1} \, c_{k-1}$
$c_n \leftarrow c_n / \alpha_n$
for $k \leftarrow n-1, n-2, \ldots, 1$
do
  $c_k \leftarrow c_k / \alpha_k - \beta_k \, c_{k+1}$
return $c_1, c_2, \ldots, c_n$

(a) The algorithm will fail (with division by zero) if any α_k is zero. Prove that, in the case of cubic spline interpolation with the natural boundary conditions,

$$\alpha_k > b_k > 0 \,, \quad k = 1, 2, \ldots, n \,.$$

Hint: Start with $k = 1$ and use the method of mathematical induction.

Solution: In the case of the natural boundary conditions,

$$a_k = 2\,(h_k + h_{k+1}) \,, \quad k = 1, 2, \ldots, n-2$$

and

$$b_k = h_{k+1} > 0 \,, \quad k = 1, 2, \ldots, n-3$$

Let us first check the case $k = 1$:

$$\alpha_1 = a_1 = 2\,(h_1 + h_2) > h_2 = b_1 \,.$$

The theorem is satisfied. Using the method of mathematical induction, let us assume that

$$\alpha_k > b_k$$

for some k and prove that the analogous inequality is true for $k+1$. Indeed, the algorithm shows that

$$\alpha_{k+1} = a_{k+1} - \frac{b_k^2}{\alpha_k} \,.$$

The assumed inequality implies that

$$\frac{b_k^2}{\alpha_k} < b_k \,.$$

Therefore,

$$\alpha_{k+1} > a_{k+1} - b_k = 2\,(h_k + h_{k+1}) - h_{k+1} = h_{k+1} + 2\,h_k > 0 \,.$$

QED.

(b) Design an alternative algorithm, where the tridiagonal matrix is factored into the product of upper and lower bidiagonal matrices, as follows:

$$\begin{bmatrix} a_1 & b_1 & 0 & \cdots & 0 \\ b_1 & a_2 & b_2 & \ddots & \vdots \\ 0 & b_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & b_{n-1} \\ 0 & \cdots & 0 & b_{n-1} & a_n \end{bmatrix} = \begin{bmatrix} \hat{\alpha}_1 & b_1 & 0 & \cdots & 0 \\ 0 & \hat{\alpha}_2 & b_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & b_{n-1} \\ 0 & \cdots & \cdots & 0 & \hat{\alpha}_n \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ \hat{\beta}_1 & 1 & \ddots & & \vdots \\ 0 & \hat{\beta}_2 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \hat{\beta}_{n-1} & 1 \end{bmatrix}$$

Solution: Matching the diagonal elements, we arrive at the system of equations

$$\begin{aligned} \hat{\alpha}_1 + b_1\,\hat{\beta}_1 &= a_1 \\ \cdots & \quad \cdots \\ \hat{\alpha}_{n-1} + b_{n-1}\,\hat{\beta}_{n-1} &= a_{n-1} \\ \hat{\alpha}_n &= a_n \end{aligned}$$

Matching the off-diagonal elements leads to the system

$$\begin{aligned} \hat{\alpha}_2\,\hat{\beta}_1 &= b_1 \\ \cdots & \quad \cdots \\ \hat{\alpha}_n\,\hat{\beta}_{n-1} &= b_{n-1} \end{aligned}$$

Together, the two systems define the *backward* recursion

$$\hat{\alpha}_n = a_n\,;$$

$$\left\{ \begin{aligned} \hat{\beta}_k &= b_k/\hat{\alpha}_{k+1} \\ \hat{\alpha}_k &= a_k - b_k\,\hat{\beta}_k \end{aligned} \right. \quad , \quad k = n-1, n-2, \ldots, 1\,.$$

After the decomposition, the upper and lower bidiagonal matrices are inverted using recursion in the opposite directions. The final algorithm is

```
TRIDIAGONAL2(a_1, a_2, ..., a_n, b_1, b_2, ..., b_{n-1}, g_1, g_2, ..., g_n)
α̂_n ← a_n
for k ← n−1, n−2, ..., 1
do
    β̂_k ← b_k / α̂_{k+1}
    α̂_k ← a_k − b_k β̂_k
c_n ← g_n
for k ← n−1, n−2, ..., 1
do
    c_k ← g_k − β̂_k c_{k+1}
c_1 ← c_1 / α̂_1
for k ← 2, 3, ..., n
do
    c_k ← c_k / α̂_k − β̂_{k−1} c_{k−1}
return c_1, c_2, ..., c_n
```

4. (Programming) In this assignment, you can use your own implementation of the natural cubic spline algorithm or a library function. For your convenience, here is the algorithm summary:

NATURAL SPLINE COEFFICIENTS($x_1, x_2, \ldots, x_n, f_1, f_2, \ldots, f_n$)

for $k \leftarrow 1, 2, \ldots, n-1$
do
    $h_k \leftarrow x_{k+1} - x_k$
    $b_k \leftarrow (f_{k+1} - f_k) / h_k$
for $k \leftarrow 2, 3, \ldots, n-1$
do
    $a_k \leftarrow 2\,(h_k + h_{k-1})$
    $g_k \leftarrow b_k - b_{k-1}$
$c_1 \leftarrow 0$
$c_n \leftarrow 0$
$c_2, c_3, \ldots, c_{n-1} \leftarrow$ TRIDIAGONAL($a_2, a_3, \ldots, a_{n-1}, h_2, h_3, \ldots, h_{n-2}, g_2, g_3, \ldots, g_{n-1}$)
for $k \leftarrow 1, 2, \ldots, n-1$
do
    $d_k \leftarrow (c_{k+1} - c_k) / h_k$
    $b_k \leftarrow b_k - (2\,c_k + c_{k+1})\,h_k$
    $c_k \leftarrow 3\,c_k$
return $b_1, b_2, \ldots, b_{n-1}, c_1, c_2, \ldots, c_{n-1}, d_1, d_2, \ldots, d_{n-1}$

SPLINE EVALUATION($x, x_1, x_2, \ldots, x_n, f_1, f_2, \ldots, f_n, b_1, b_2, \ldots, b_{n-1}, c_1, c_2, \ldots, c_{n-1}, d_1, d_2, \ldots, d_{n-1}$)

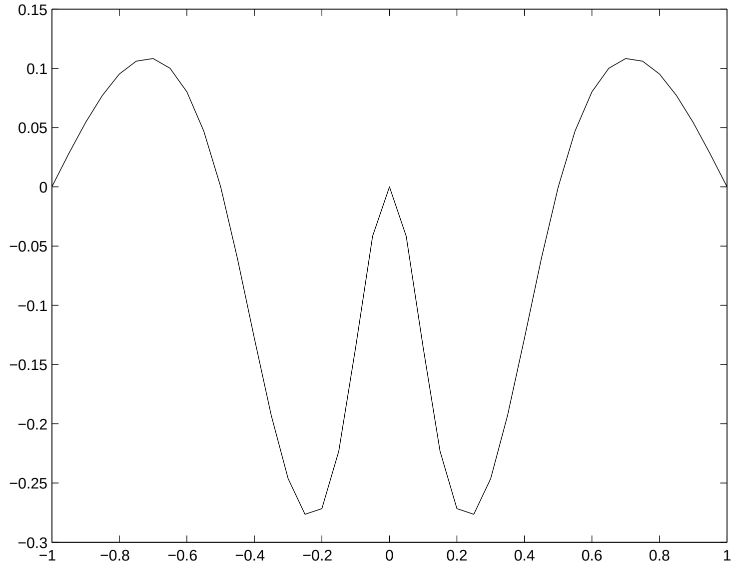
for $k \leftarrow n-1, n-2, \ldots, 1$
do
    $h \leftarrow x - x_k$
    if $h > 0$
        then exit loop
$S \leftarrow f_k + h\,(b_k + h\,(c_k + h\,d_k))$
return S

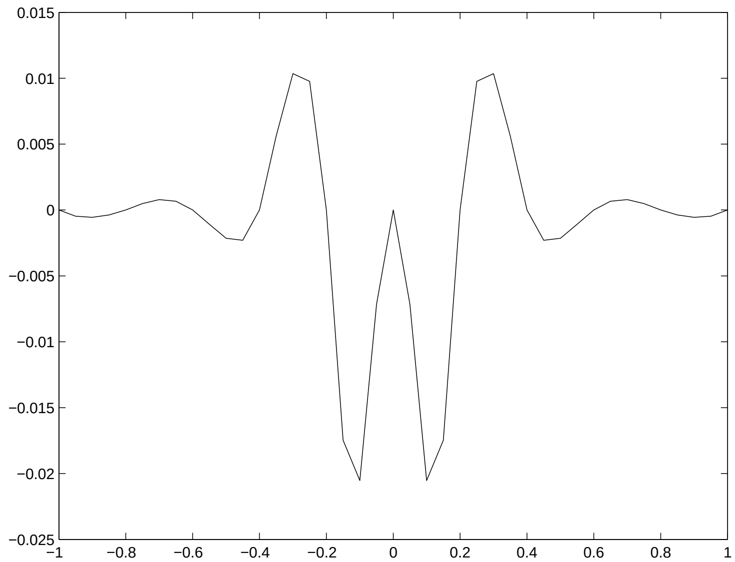
Using your program, interpolate Runge's function $f(x) = \frac{1}{1+25x^2}$ on a set of n regularly spaced spline knots

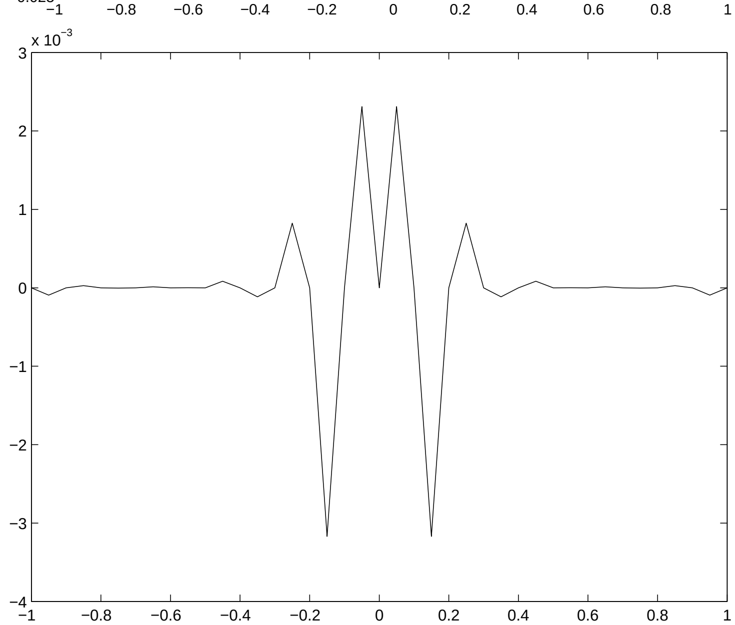
$$x_k = -1 + \frac{2\,(k-1)}{n-1}\,, \quad k = 1, 2, \ldots, n\,.$$

Take $n = 5, 11, 21$ and compute the interpolation spline $S(x)$ and the error $f(x) - S(x)$ at 41 regularly spaced points. You can either plot the error or output it in a table. Does the interpolation accuracy increase with the number of knots?

Answer:







The accuracy does increase with the number of knots.

Solution:

C program:

```
#include <stdlib.h> /* for allocation */
#include <assert.h> /* for assertion */

#include "spline.h"

/* Function: tridiagonal
   ---------------------
   Symmetric tridiagonal system solver
   n         - data length
   diag[n]   - diagonal
   offd[n-1] - off-diagonal
   x[n]      - in: right-hand side, out: solution
*/
void tridiagonal(int n, const double* diag, const double* offd, double* x)
{
    double *a, *b;
    int k;

    /* allocate storage */
    a = (double*) malloc(n*sizeof(double));
    b = (double*) malloc((n-1)*sizeof(double));
    assert (a != NULL && b != NULL);

    /* LU decomposition */
    a[0] = diag[0];
    for (k=0; k < n-1; k++) {
b[k] = offd[k]/a[k];
a[k+1] = diag[k+1] - b[k]*offd[k];
    }
    /* inverting L */
    for (k=1; k < n; k++) {
x[k] = x[k] - b[k-1]*x[k-1];
    }
    /* inverting U */
    x[n-1] /= a[n-1];
    for (k=n-2; k >=0; k--) {
x[k] = x[k]/a[k] - b[k]*x[k+1];
    }

    free (a);
    free (b);
}

/* Function: tridiagonal2
   ----------------------
   Alternative form of symmetric tridiagonal system solver
   n         - data length
   diag[n]   - diagonal
   offd[n-1] - off-diagonal
   x[n]      - in: right-hand side, out: solution
*/
void tridiagonal2(int n, const double* diag, const double* offd, double* x)
{
    double *a, *b;
    int k;
```

```
    /* allocate storage */
    a = (double*) malloc(n*sizeof(double));
    b = (double*) malloc((n-1)*sizeof(double));
    assert (a != NULL && b != NULL);

    /* UL decomposition */
    a[n-1] = diag[n-1];
    for (k=n-2; k >= 0; k--) {
b[k] = offd[k]/a[k+1];
a[k] = diag[k] - b[k]*offd[k];
    }
    /* inverting U */
    for (k=n-2; k >= 0; k--) {
x[k] = x[k] - b[k]*x[k+1];
    }
    /* inverting L */
    x[0] /= a[0];
    for (k=1; k < n-1; k++) {
x[k] = x[k]/a[k] - b[k-1]*x[k-1];
    }

    free (a);
    free (b);
}

/* Function: spline_coeffs
   -----------------------
   Compute spline coefficients for interpolating natural cubic spline
   n - number of knots
   x[n] - knots
   f[n] - function values
   coeff[4][n] - coefficients
*/
void spline_coeffs(int n, const double* x, const double* f, double* coeff[])
{
    double *a, *h, *b, *c, *d;
    int k;

    h = (double*) malloc((n-1)*sizeof(double));
    a = (double*) malloc((n-2)*sizeof(double));
    assert (h != NULL);

    /* rename for convenience */
    b = coeff[1];
    c = coeff[2];
    d = coeff[3];

    for (k=0; k < n-1; k++) {
h[k] = x[k+1] - x[k];       /* interval length */
coeff[0][k] = f[k];
b[k] = (f[k+1]-f[k])/h[k];  /* divided difference */
    }
    for (k=0; k < n-2; k++) {
a[k] = 2.*(h[k+1] + h[k]);  /* diagonal */
c[k+1] = b[k+1] - b[k];     /* right-hand side */
    }
    c[0] = 0;
    /* solve the tridiagonal system */
```

```
    tridiagonal(n-2, a, h, c+1);
    for (k=0; k < n-1; k++) {
if (k < n-2) {
    d[k] = (c[k+1]-c[k])/h[k];
    b[k] -= (c[k+1]+2.*c[k])*h[k];
} else {
    d[k] = -c[k]/h[k];
    b[k] -= 2.*c[k]*h[k];
}
c[k] *= 3.;
    }
}

/* Function: spline_eval
   ---------------------
   Evaluate a cubic spline
   n           - number of knots
   y           - where to evaluate
   x[n]        - knots
   coeff[4][n] - spline coefficients
*/
double spline_eval(int n, double y, const double* x, double* coeff[])
{
    double h, s;
    int i, k;

    /* find the interval for x */
    for (k=n-2; k >=0; k--) {
h = y - x[k];
if (h >= 0.) break;
    }
    if (k < 0) k = 0;

    /* evaluate cubic by Horner's rule */
    s = coeff[3][k];
    for (i=2; i >=0; i--) {
s = s*h + coeff[i][k];
    }
    return s;
}

#include <stdlib.h> /* for allocation */
#include <stdio.h>  /* for output */
#include <assert.h> /* for assertion */

#include "spline.h"

/* Runge's function */
double runge (double x)
{
    return (1./(1.+25.*x*x));
}

int main (void)
{
    int i, k, n[]={5,11,21}, nx, ny=41;
    double *x, *f, *coeff[4], *y, xk, s, e;

    y = (double*) malloc (ny*sizeof(double));
```

```
    assert (y != NULL);

    /* regular grid for plotting */
    for (k=0; k < ny; k++) {
y[k] = -1. + 2.*k/(ny-1.);
    }

    /* three cases */
    for (i=0; i < 3; i++) {
nx = n[i];

/* allocate space for table */
x = (double*) malloc (nx*sizeof(double));
f = (double*) malloc (nx*sizeof(double));
assert (x != NULL && f != NULL);

/* allocate coefficients */
for (k=0; k < 4; k++) {
    coeff[k] = (double*) malloc ((nx-1)*sizeof(double));
    assert (coeff[k] != NULL);
}

/* build the table */
for (k=0; k < nx; k++) {
    xk = -1. + 2.*k/(nx-1.);
    f[k] = runge(xk);
    x[k] = xk;
}

/* compute coefficients */
spline_coeffs(nx, x, f, coeff);

/* evaluate the spline function */
for (k=0; k < ny; k++) {
    xk = y[k];
    s = spline_eval(nx, xk, x, coeff); /* spline */
    e = runge(xk)-s;                   /* error  */
    /* print out the table */
    printf("%d %f %f %g\n", k, xk, s, e);
}

free (x);
free (f);
for (k=0; k < 4; k++) {
    free (coeff[k]);
}
    }

    exit(0);
}
```

5. (Programming)

The values in the table specify $\{x, y\}$ points on a curve $\{x(t), y(t)\}$.

x	2.5	1.3	-0.25	0.	0.25	-1.3	-2.5	-1.3	0.25	0.	-0.25	1.3	2.5
y	0.	-0.25	1.3	2.5	1.3	-0.25	0.	0.25	-1.3	-2.5	-1.3	0.25	0.

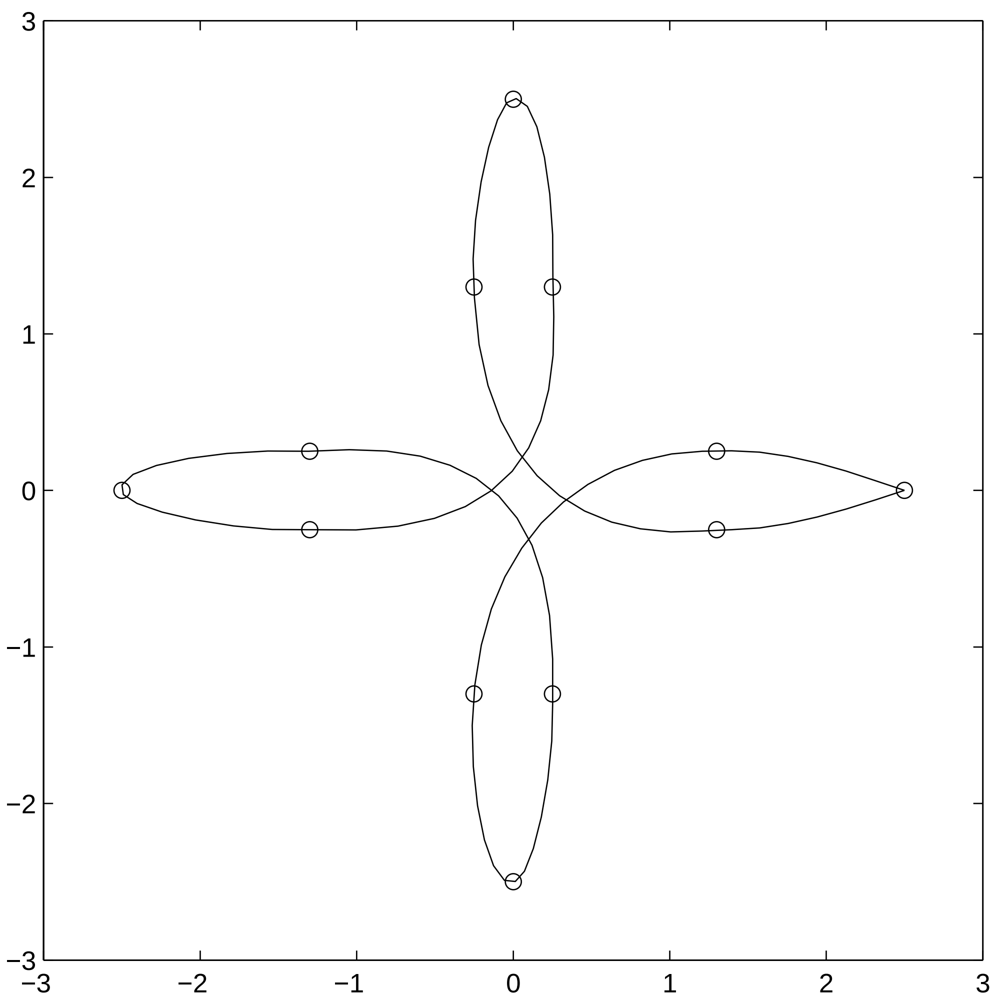
In this assignment, you will reconstruct the curve using cubic splines and interpolating independently $x(t)$ and $y(t)$. We don't know the values of t at the spline knots but can approximate them. For example, we can take t to represent the length along the curve and approximate it by the length of the linear segments:

$$
\begin{aligned}
t_1 &= 0\,; \\
t_k &= t_{k-1} + \sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2}\,, \quad k = 2, 3, \ldots, n\,.
\end{aligned}
$$

Calculate spline coefficients for the natural cubic splines interpolating $x(t)$ and $y(t)$, then evaluate the splines at 100 regularly spaced points in the interval between t_1 and t_n and plot the curve.

What other boundary conditions would be appropriate in this example?

Answer:



The periodic boundary conditions would be more appropriate in this case.

Solution:

C program:

```
#include <stdlib.h> /* for allocation */
#include <stdio.h>  /* for output */
#include <math.h>   /* for math functions */
#include <assert.h> /* for assertion */

#include "spline.h"

int main (void)
{
    const int nt=13, nt1=100;
    int k;
    double x[] = {2.5,1.3,-0.25,0.,0.25,-1.3,-2.5,-1.3,0.25,0.,-0.25,1.3,2.5};
    double y[] = {0.,-0.25,1.3,2.5,1.3,-0.25,0.,0.25,-1.3,-2.5,-1.3,0.25,0.};
    double t[13], *xc[4], *yc[4], t1[100], tk, x1, y1;

    for (k=0; k < 4; k++) {
xc[k] = (double*) malloc ((nt-1)*sizeof(double));
yc[k] = (double*) malloc ((nt-1)*sizeof(double));
assert (xc[k] != NULL && yc[k] != NULL);
    }

    /* find the knots */
    t[0] = 0.;
    for (k=1; k < nt; k++) {
t[k] = t[k-1] + hypot(x[k]-x[k-1],y[k]-y[k-1]);
    }

    /* regular grid for plotting */
    for (k=0; k < nt1; k++) {
t1[k] = k*t[nt-1]/(nt1-1.);
    }

    /* spline coefficients for x(t) and y(t) */
    spline_coeffs(nt, t, x, xc);
    spline_coeffs(nt, t, y, yc);

    /* evaluate the spline function */
    for (k=0; k < nt1; k++) {
tk = t1[k];
x1 = spline_eval(nt, tk, t, xc);
y1 = spline_eval(nt, tk, t, yc);

/* print out the table */
printf("%d %f %f %g\n", k, tk, x1, y1);
    }

    exit(0);
}
```