*Prof. R. Hiptmair*
G. Alberti,
F. Leonardi

AS 2015

ETH Zürich
D-MATH

# Numerical Methods for CSE

## Problem Sheet 8

### Problem 1. Natural cubic Splines (core problem)

In [1, Section 3.5.1] we learned about cubic spline interpolation and its variants, the complete, periodic, and natural cubic spline interpolation schemes.

**(1a)** ⊡ Given a knot set $\mathcal{T} = \{t_0 < t_1 < \cdots < t_n\}$, which also serves as the set of interpolation nodes, and values $y_j$, $j = 0, \ldots, n$, write down the linear system of equations that yields the slopes $s'(t_j)$ of the natural cubic spline interpolant $s$ of the data points $(t_j, y_j)$ at the knots.

**Solution:** Let $h_i := t_i - t_{i-1}$. Given the natural condition on the spline, one can remove the columns relative to $c_0 := s'(t_0)$ and $c_n := s'(t_n)$ from the system matrix, which becomes:

$$\mathbf{A} := \begin{pmatrix} 2/h_1 & 1/h_1 & 0 & 0 & \cdots & & 0 \\ b_0 & a_1 & b_1 & 0 & \cdots & & 0 \\ 0 & b_1 & a_2 & b_2 & \ddots & & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & b_{n-3} & a_{n-2} & b_{n-2} & 0 \\ 0 & & \cdots & 0 & b_{n-2} & a_{n-1} & b_{n-1} \\ 0 & & \cdots & \cdots & 0 & 1/h_n & 2/h_n \end{pmatrix}, a_i := \frac{2}{h_i} + \frac{2}{h_{i+1}}, b_i := \frac{1}{h_{i+1}} \tag{29}$$

$$\mathbf{c} := [c_0, c_1, \ldots, c_n] \tag{30}$$

$$\mathbf{b} := [r_0, \ldots, r_n] \tag{31}$$

Where $c_i := s'(t_i)$. We define $r_i := 3\left(\frac{y_i - y_{i-1}}{h_i^2} + \frac{y_{i+1} - y_i}{h_{i+1}^2}\right)$, $i = 1, \ldots, n-1$, and

$$r_0 = 3\frac{y_1 - y_0}{h_1^2}, r_n = 3\frac{y_n - y_{n-1}}{h_n^2}$$

The system becomes $\mathbf{Ac} = \mathbf{b}$.

**(1b)** ⚄ Argue why the linear system found in subsubsection (1a) has a unique solution.

HINT: Look up [1, Lemma 1.8.12] and apply its assertion.

**Solution:** Notice that $a_i := \frac{2}{h_i} + \frac{2}{h_{i+1}} > \frac{1}{h_{i+1}} + \frac{1}{h_i} =: b_i + b_{i-1}$. The matrix is (strictly) diagonally dominant and, therefore, invertible.

**(1c)** ⚄ Based on EIGEN devise an *efficient* implementation of a C++ class for the computation of a natural cubic spline interpolant with the following definition:

```
class NatCSI {
public:
    //! \brief Build the cubic spline interpolant with
       natural boundaries
    //! Setup the data structures you need.
    //! Pre-compute the coefficients of the spline
       (solve system)
    //! \param[in] t, nodes of the grid (for pairs (t_i,
       y_i)) (sorted!)
    //! \param[in] y, values y_i at t_i (for pairs (t_i,
       y_i))
    NatCSI(const const std::vector<double> & t, const
       const std::vector<double> & y);

    //! \brief Interpolant evaluation at x
    //! \param[in] x, value x where to evaluate the
       spline
    //! \return value of the spline at x
    double operator() (double x) const;

private:
    // TODO: store data for the spline
};
```

HINT: Assume that the input array of knots is sorted and perform binary searches for the evaluation of the interpolant.

**Solution:** See `natcsi.cpp`.

## Problem 2. Monotonicity preserving interpolation (core problem)

This problem is about monotonicity preserving interpolation. Before starting, you should revise [1, Def. 3.1.15], [1, § 3.3.2] and [1, Section 3.4.2] carefully.

**(2a)** ⚄ Prove [1, Thm. 3.4.17]:

> If, for fixed node set $\{t_j\}_{j=0}^n$, $n \geq 2$, an interpolation scheme $\mathsf{I} : \mathbb{R}^{n+1} \to C^1(I)$ is *linear* as a mapping from data values to continuous functions on the interval covered by the nodes (→ [1, Def. 3.1.15]), and *monotonicity preserving*, then $\mathsf{I}(\mathbf{y})'(t_j) = 0$ for all $\mathbf{y} \in \mathbb{R}^{n+1}$ and $j = 1, \ldots, n-1$.

HINT: Consider a suitable basis $\{\mathbf{s}^{(j)} : j = 0, \ldots, n\}$ of $\mathbb{R}^{n+1}$ that consists of monotonic vectors, namely such that $s_i^{(j)} \leq s_{i+1}^{(j)}$ for every $i = 0, \ldots, n-1$.

HINT: Exploit the phenomenon explained next to [1, Fig. 99].

**Solution:** Without loss of generality assume that $t_0 < t_1 < \cdots < t_n$. For $j = 0, \ldots, n$ let $\mathbf{s}^{(j)} \in \mathbb{R}^{n+1}$ be defined by

$$s_i^{(j)} = \begin{cases} 0 & i = 0, \ldots, j-1 \\ 1 & i = j, \ldots, n. \end{cases}$$

Clearly, $(t_i, s_i^{(j)})_i$ are monotonic increasing data, according to [1, Def. 3.3.3].

Take $j = 0, \ldots, n$. Note that $\mathbf{s}^{(j)}$ has a local extremum in $t_i$ for every $i = 1, \ldots, n-1$. Thus, since $\mathsf{I}$ is monotonicity preserving (see [1, § 3.3.6]), $\mathsf{I}(\mathbf{s}^{(j)})$ has to be flat in $t_i$ for every $i = 1, \ldots, n-1$ (see [1, Section 3.4.2]). As a consequence,

$$(\mathsf{I}(\mathbf{s}^{(j)}))'(t_i) = 0, \quad i = 1, \ldots, n-1. \tag{32}$$

Note now that $\{\mathbf{s}^{(j)} : j = 0, \ldots, n\}$ is a basis for $\mathbb{R}^{n+1}$ (indeed, they constitute a linearly independent set with cardinality equal to the dimension of the space). As a consequence, every $\mathbf{y} \in \mathbb{R}^{n+1}$ can be written as a linear combination of the $\mathbf{s}^{(j)}$s, namely

$$\mathbf{y} = \sum_{j=0}^{n} \alpha_j \mathbf{s}^{(j)}.$$

Therefore, by the linearity of $\mathsf{I}$ and using (32), for every $i = 1, \ldots, n-1$ we obtain

$$(\mathsf{I}(\mathbf{y}))'(t_i) = \left(\mathsf{I}\left(\sum_{j=0}^{n} \alpha_j \mathbf{s}^{(j)}\right)\right)'(t_i) = \left(\sum_{j=0}^{n} \alpha_j \mathsf{I}(\mathbf{s}^{(j)})\right)'(t_i) = \sum_{j=0}^{n} \alpha_j \mathsf{I}(\mathbf{s}^{(j)})'(t_i) = 0,$$

as desired.

## Problem 3. Local error estimate for cubic Hermite interpolation (core problem)

Consider the cubic Hermite interpolation operator $\mathcal{H}$ of a function defined on an interval $[a, b]$ to the space $\mathcal{P}_3$ polynomials of degree at most 3:

$$\mathcal{H} : C^1([a, b]) \to \mathcal{P}_3$$

defined by:

- $(\mathcal{H}f)(a) = f(a)$;
- $(\mathcal{H}f)(b) = f(b)$;
- $(\mathcal{H}f)'(a) = f'(a)$;
- $(\mathcal{H}f)'(b) = f'(b)$.

Assume $f \in C^4([a, b])$. Show that for every $x \in ]a, b[$ there exists $\tau \in [a, b]$ such that

$$(f - \mathcal{H}f)(x) = \frac{1}{24} f^{(4)}(\tau)(x - a)^2(x - b)^2. \tag{33}$$

HINT: Fix $x \in ]a, b[$. Use an auxiliary function:

$$\varphi(t) := f(t) - (\mathcal{H}f)(t) - C(t - a)^2(t - b)^2. \tag{34}$$

Find $C$ s.t. $\varphi(x) = 0$.

HINT: Use Rolle's theorem (together with the previous hint and the definition of $\mathcal{H}$) to find a lower bound for the number of zeros of $\varphi^{(k)}$ for $k = 1, 2, 3, 4$.

HINT: Use the fact that $(\mathcal{H}f) \in \mathcal{P}_3$ and for $p(t) := (t - a)^2(t - b)^2, p \in \mathcal{P}_4$.

HINT: Conclude showing that $\exists \tau \in ]a, b[, \varphi^{(4)}(\tau) = 0$. Use the definition of $\varphi$ to find an expression for $C$.

**Solution:** Fix $x \in ]a, b[$. Define:

$$\varphi(t) := f(t) - \mathcal{H}f(t) - C(t - a)^2(t - b)^2 \tag{35}$$

with

$$C := \frac{f(x) - \mathcal{H}f(x)}{(x - a)^2(x - b)^2}.$$

Then $\varphi(a) = \varphi(b) = \varphi'(a) = \varphi'(b) = 0$ (using the definition of $\mathcal{H}$). Moreover $\varphi(x) = 0$ (by construction). Therefore, by Rolle's theorem ($\varphi$ has at least two local extrema), $\exists \xi_1, \xi_2 \in ]a, b[, \xi_1 \neq \xi_2$ such that $\varphi'(\xi_1) = \varphi'(\xi_2) = 0$.
$\Rightarrow \varphi'$ has at least 4 zeros in $[a, b]$ ($a, b, \xi_1$ and $\xi_2$ are pairwise distinct).
$\Rightarrow \varphi''$ has at least 3 zeros in $[a, b]$ ($\varphi'$ has at least 3 local extrema).
$\Rightarrow \varphi^{(3)}$ has at least 2 zeros in $[a, b]$ ($\varphi''$ has at least 2 local extrema).
$\Rightarrow \varphi^{(4)}$ has at least 1 zero in $[a, b]$ ($\varphi^{(3)}$ has at least one local extrema), let $\tau$ be such zero.
$\Rightarrow 0 = \varphi^{(4)}(\tau) = f^{(4)}(\tau) - 24C$.
$\Rightarrow C = \frac{1}{24} f^{(4)}(\tau)$. $\square$

## Problem 4. Adaptive polynomial interpolation

In [1, Section 4.1.3] we have seen that the placement of interpolation nodes is key to a good approximation by a polynomial interpolant. The following *greedy algorithm* attempts to find the location of suitable nodes by its own:

> Given a function $f : [a, b] \mapsto \mathbb{R}$ one starts $\mathcal{T} := \{\frac{1}{2}(b + a)\}$. Based on a fixed finite set $\mathcal{S} \subset [a, b]$ of *sampling points* one augments the set of nodes according to
>
> $$\mathcal{T} = \mathcal{T} \cup \left\{ \operatorname*{argmax}_{t \in \mathcal{S}} |f(t) - \mathsf{I}_{\mathcal{T}}(t)| \right\} , \tag{36}$$
>
> where $\mathsf{I}_{\mathcal{T}}$ is the polynomial interpolation operator for the node set $\mathcal{T}$, until
>
> $$\max_{t \in \mathcal{S}} |f(t) - \mathsf{I}_{\mathcal{T}}(t)| \leq \texttt{tol} \cdot \max_{t \in \mathcal{S}} |f(t)| . \tag{37}$$

**(4a)** ⚄ Write a MATLAB function

```
function t = adaptivepolyintp(f,a,b,tol,N)
```

that implements the algorithm described above and takes as arguments the function handle `f`, the interval bounds `a`, `b`, the relative tolerance `tol`, and the number `N` of *equidistant* sampling points (in the interval $[a, b]$), that is,

$$\mathcal{S} := \{a + (b - a)\frac{j}{N},\ j = 0, \ldots, N\} .$$

HINT: The function `intpolyval` from [1, Code 3.2.28] is provided and may be used (though it may not be the most efficient way to implement the function).

**Solution:** See Listing 39.

**(4b)** ⚀ Extend the function from the previous sub-problem so that it reports the quantity

$$\max_{t \in \mathcal{S}} |f(t) - \mathsf{T}_{\mathcal{T}}(t)| \tag{38}$$

for each intermediate set $\mathcal{T}$.

**Solution:** See Listing 39.

Listing 39: Implementation of the function `adaptivepolyintp`

```
function [t,errs] = adaptivepolyintp(f,a,b,tol,N)
% Adaptive polynomial interpolation of function f:[a,b]
    -> R.
% argument 'f' must be a handle that allows vectorized
    evaluation,
% argument 'tol' specifies the relative tolerance,
% the optional argument 'N' passes the number of
    sampling points.
if (nargin < 5), N = 1000; end
sp = (a:(b-a)/N:b);       % N+1 equdistant sampling points
fvals = f(sp);            % evaluate function at sampling
    points
fmax  = max(abs(fvals)); % Maximum of f in sampling
    points
t = sp(floor(N/2));       % set of interpolation nodes
y = fvals(floor(N/2));    % function values at
    interpolation nods
errs = [];
for i=1:N
  err = abs(fvals - intpolyval(t,y,sp)); % modulus of
      interpolation error
  [mx,idx] = max(err); errs = [errs, mx];
  % finishes, once maximal pointwise interpolation error
      below threshold
  if (mx < tol*fmax), return; end
  t = [t,sp(idx)];
  y = [y,fvals(idx)];
end
error('Desired accuracy could not be reached');
```

**(4c)** ⚀ For $f_1(t) := \sin(e^{2t})$ and $f_2(t) = \frac{\sqrt{t}}{1+16t^2}$ plot the quantity from (38) versus the number of interpolation nodes. Choose plotting styles that reveal the qualitative decay of this error as the number of interpolation nodes is increased. Use interval $[a, b] = [0, 1]$, `N=1000` sampling points, tolerance `tol = 1e-6`.

**Solution:** See Listing 40 and Figure 16.

Listing 40: Implementation of the function `plot_adaptivepolyintp`

```
function plot_adaptivepolyintp ()
f1 = (t) sin(exp(2*t));f2 =
   (t) sqrt(t) ./ ( 1 + 16*t.^2 );
a = 0; b = 1; tol = 1e-6;
[¬,err1] = adaptivepolyintp (f1,a,b,tol);
[¬,err2] = adaptivepolyintp (f2,a,b,tol);

semilogy(1:length(err1), err1, 'r-', 1:length(err2),
   err2, 'b-');
legend('f_1', 'f_2');
title('{\bf Error decay for adaptive polynomial
   interpolation}');
xlabel('{\bf number of nodes}');
ylabel('{\bf error}');

print -depsc '../PICTURES/plot_adaptivepolyintp.eps'
```

Issue date: 05.11.2015

Hand-in: 12.11.2015 (in the boxes in front of HG G 53/54).

Version compiled on: November 12, 2015 (v. 1.0).