

Problem Sheet 7

Problem 1. Cubic Splines (core problem)

Since they mimic the behavior of an elastic rod pinned at fixed points, see [1, § 3.5.13], cubic splines are very popular for creating “aesthetically pleasing” interpolating functions. However, in this problem we look at a cubic spline from the perspective of its defining properties, see [1, Def. 3.5.1], in order to become more familiar with the concept of spline function and the consequences of the smoothness required by the definition.

For parameters $\alpha, \beta \in \mathbb{R}$ we define the function $s_{\alpha, \beta} : [-1, 2] \rightarrow \mathbb{R}$ by

$$s_{\alpha, \beta}(x) = \begin{cases} (x+1)^4 + \alpha(x-1)^4 + 1 & x \in [-1, 0] \\ -x^3 - 8\alpha x + 1 & x \in (0, 1] \\ \beta x^3 + 8x^2 + \frac{11}{3} & x \in (1, 2] \end{cases} \quad (12)$$

(1a) \square Determine α, β such that $s_{\alpha, \beta}$ is a cubic spline in $\mathcal{S}_{3, M}$ with respect to the node set $M = \{-1, 0, 1, 2\}$. Verify that you actually obtain a cubic spline.

(1b) \square Use MATLAB to create a plot of the function defined in (12) in dependence of α and β .

Problem 2. Quadratic Splines (core problem)

[1, Def. 3.5.1] introduces spline spaces $\mathcal{S}_{d, \mathcal{M}}$ of any degree $d \in \mathbb{N}_0$ on a node set $\mathcal{M} \subset \mathbb{R}$. [1, Section 3.5.1] discusses interpolation by means of cubic splines, which is the most important case. In this problem we practise spline interpolation for quadratic splines in order to understand the general principles.

Consider a 1-periodic function $f : \mathbb{R} \rightarrow \mathbb{R}$, that is, $f(t+1) = f(t)$ for all $t \in \mathbb{R}$, and a set of nodes

$$\mathcal{M} := \{0 = t_0 < t_1 < t_2 < \dots < t_{n-1} < t_n = 1\} \subset [0, 1] .$$

We want to approximate f using a I -periodic quadratic spline function $s \in \mathcal{S}_{2,\mathcal{M}}$, which interpolates f in the midpoints of the intervals $[t_{j-1}, t_j]$, $j = 0, \dots, n$.

In analogy to the local representation of a cubic spline function according to [1, Eq. (3.5.5)], we parametrize a quadratic spline function $s \in \mathcal{S}_{2,\mathcal{M}}$ according to

$$s|_{[t_{j-1}, t_j]}(t) = d_j \tau^2 + c_j 4 \tau(1 - \tau) + d_{j-1} (1 - \tau)^2, \quad \tau := \frac{t - t_{j-1}}{t_j - t_{j-1}}, \quad j = 1, \dots, n, \quad (13)$$

with $c_j, d_k \in \mathbb{R}$, $j = 1, \dots, n$, $k = 0, \dots, n$. Notice that the coefficients d_k are associated with the nodes t_k while the c_j are associated with the midpoints of the intervals $[t_{j-1}, t_j]$.

(2a) \square What is the dimension of the subspace of I -periodic spline functions in $\mathcal{S}_{2,\mathcal{M}}$?

(2b) \square What kind of continuity is already guaranteed by the use of the representation (13)?

(2c) \boxtimes Derive a linear system of equations (system matrix and right hand side) whose solution provides the coefficients c_j and d_j in (13) from the function values $y_j := f(\frac{1}{2}(t_{j-1} + t_j))$, $j = 1, \dots, n$.

HINT: By [1, Def. 3.5.1] we know $\mathcal{S}_{2,\mathcal{M}} \subset C^1([0, 1])$, which provides linear constraints at the nodes, analogous to [1, Eq. (3.5.6)] for cubic splines.

(2d) \boxtimes Implement an *efficient* MATLAB routine

```
function s=quadspline(t,y,x)
```

which takes as input a (sorted) node vector \mathbf{t} (of length $n-1$, because $t_0 = 0$ and $t_n = 1$ will be taken for granted), a n -vector \mathbf{y} containing the values of a function f at the midpoints $\frac{1}{2}(t_{j-1} + t_j)$, $j = 1, \dots, n$, and a *sorted* N -vector \mathbf{x} of evaluation points in $[0, 1]$.

The function is to return the values of the interpolating quadratic spline s at the positions \mathbf{x} .

You can test your code with the one provided by `quadspline_p.p` (available on the lecture website).

(2e) \square Plot f and the interpolating periodic quadratic spline s for $f(t) := \exp(\sin(2\pi t))$, $n = 10$ and $\mathcal{M} = \{\frac{j}{n}\}_{j=0}^{10}$, that is, the spline is to fulfill $s(t) = f(t)$ for all midpoints t of knot intervals.

(2f) ◻ What is the complexity of the algorithm in (2d) in dependance of n and N ?

Problem 3. Curve Interpolation (core problem)

The focus of [1, Chapter 3] was on the interpolation of data points by means of functions belonging to a certain linear space. A different task is to find a curve containing each point of a set $\{\mathbf{p}_0, \dots, \mathbf{p}_n\} \subset \mathbb{R}^2$.

This task can be translated in a standard interpolation problem after recalling that a curve in \mathbb{R}^2 can be described by a mapping (*parametrization*) $\gamma : [0, 1] \mapsto \mathbb{R}^2$, $\gamma(t) = \begin{pmatrix} s_1(t) \\ s_2(t) \end{pmatrix}$. Hence, given the nodes $0 = t_0 < t_1 < \dots < t_{n-1} < t_n = 1$, we aim at finding interpolating functions $s_1, s_2 : [0, 1] \rightarrow \mathbb{R}$, such that $s_i(t_j) = (\mathbf{p}_j)_i$, $i = 1, 2$, $j = 0, \dots, n$. This means that we separately interpolate the x and y coordinates of \mathbf{p}_j .

A crucial new aspect is that the nodes are not fixed, i.e., there are infinitely many parameterizations for a given curve: for any strictly monotonous and surjective $h : [0, 1] \rightarrow [0, 1]$ the mappings γ and $\tilde{\gamma} := \gamma \circ h$ describe exactly the same curve. On the other hand, the selection of nodes will affect the interpolants s_1 and s_2 and leads to different interpolating curves.

Concerning the choice of the nodes, we will consider two options:

$$\textcircled{1} \quad \text{equidistant parametrization: } t_k = k\Delta t, \Delta t = \frac{1}{n} \quad (14)$$

$$\textcircled{2} \quad \text{segment length parametrization: } t_k = \frac{\sum_{l=1}^k |\mathbf{p}_l - \mathbf{p}_{l-1}|}{\sum_{l=1}^n |\mathbf{p}_l - \mathbf{p}_{l-1}|}. \quad (15)$$


Point data will be generated by the MATLAB function `heart` that is available on the course webpage.


(3a) ◻ Write a MATLAB function

```
function pol = polycurveintp (xy,t,tt)
```

which uses global polynomial interpolation (using the `intpolyval` function, see [1, Code 3.2.28]) through the $n+1$ points $\mathbf{p}_i \in \mathbb{R}^2$, $i = 0, \dots, n$, whose coordinates are stored in the $2 \times (n+1)$ matrix `xy` and returns sampled values of the obtained curve in a $2 \times N$ matrix `pol`. Here, `t` passes the node vector $(t_0, t_1, \dots, t_n) \in \mathbb{R}^{n+1}$ in the parameter domain and N is the number of equidistant sampling points.

HINT: Code for `intpolyval` is available as `intpolyval.m`.

(3b)  Plot the curves obtained by global polynomial interpolation `polycurveintp` of the `heart` data set. The nodes for polynomial interpolation should be generated according to the two options (14) and (15)

(3c)  Extend your MATLAB function `pol = curveintp` to

```
function pch = pchcurveintp (xy,t,tt),
```

which has the same purpose, arguments and return values as `polycurveintp`, but now uses monotonicity preserving cubic Hermite interpolation (available through the MATLAB built-in function `pchip`, see also [1, Section 3.4.2]) instead of global polynomial interpolation.

Plot the obtained curves for the `heart` data set in the figure created in sub-problem (3b). Use both parameterizations (14) and (15).

(3d)  Finally, write yet another MATLAB function

```
function spl = splinecurveintp (xy,t,tt),
```

which has the same purpose, arguments and return values as `polycurveintp`, but now uses *complete* cubic spline interpolation.

The required derivatives $s'_1(0)$, $s'_2(0)$, $s'_1(1)$, and $s'_2(1)$ should be computed from the directions of the line segments connecting \mathbf{p}_0 and \mathbf{p}_1 , and \mathbf{p}_{n-1} and \mathbf{p}_n , respectively. You can use the MATLAB built-in function `spline`. Plot the obtained curves (`heart` data) in the same figure as before using both parameterizations (14) and (15).

HINT: read the MATLAB help page about the `spline` command and learn how to impose the derivatives at the endpoints.

Problem 4. Approximation of π

In [1, Section 3.2.3.3] we learned about the use of polynomial extrapolation (= interpolation outside the interval covered by the nodes) to compute inaccessible limits $\lim_{h \rightarrow 0} \Psi(h)$. In this problem we apply extrapolation to obtain the limit of a sequence $x^{(n)}$ for $n \rightarrow \infty$.

We consider a quantity of interest that is defined as a limit

$$x^* = \lim_{n \rightarrow \infty} T(n), \quad (16)$$

with a function $T : \{n, n+1, \dots\} \mapsto \mathbb{R}$. However, computing $T(n)$ for very large arguments k may not yield reliable results.

The idea of *extrapolation* is, firstly, to compute a few values $T(n_0), T(n_1), \dots, T(n_k)$, $k \in \mathbb{N}$, and to consider them as the values $g(1/n_0), g(1/n_1), \dots, g(1/n_k)$ of a continuous function $g :]0, 1/n_{\min}] \mapsto \mathbb{R}$, for which, obviously

$$x^* = \lim_{h \rightarrow 0} g(h) . \quad (17)$$

Thus we recover the usual setting for the application of polynomial extrapolation techniques. Secondly, according to the idea of extrapolation to zero, the function g is approximated by an interpolating polynomial $p \in \mathcal{P}_{k-1}$ with $p_{k-1}(n_j^{-1}) = T(n_j)$, $j = 1, \dots, k$. In many cases we can expect that $p_{k-1}(0)$ will provide a good approximation for x^* . In this problem we study the algorithmic realization of this extrapolation idea for a simple example.

The unit circle can be approximated by inscribed regular polygons with n edges. The length of half of the circumference of such an n -edged polygon can be calculated by elementary geometry:

n	2	3	4	5	6	8	10
$T(n) := \frac{U_n}{2}$	2	$\frac{3}{2}\sqrt{3}$	$2\sqrt{2}$	$\frac{5}{4}\sqrt{10-2\sqrt{5}}$	3	$4\sqrt{2-\sqrt{2}}$	$\frac{5}{2}(\sqrt{5}-1)$

Write a C++ function

```
double pi_approx(int k);
```

that uses the *Aitken-Neville scheme*, see [1, Code 3.2.31], to approximate π by extrapolation from the data in the above table, using the first k values, $k = 1, \dots, 7$.

Issue date: 29.10.2015

Hand-in: 05.11.2015 (in the boxes in front of HG G 53/54).

Version compiled on: October 29, 2015 (v. 1.0).

References

- [1] R. Hiptmair. *Lecture slides for course "Numerical Methods for CSE"*. <http://www.sam.math.ethz.ch/~hiptmair/tmp/NumCSE/NumCSE15.pdf>. 2015.