

## 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

$$s_{\alpha, \beta}(x) = \begin{cases} (x+1)^4 + \alpha(x-1)^4 + 1 & x \in [-1, 0] \\ -x^3 - 8\alpha x + 1 & x \in (0, 1] \\ \beta x^3 + 8x^2 + \frac{11}{3} & x \in (1, 2] \end{cases} \quad (19)$$

**(1a)**  $\square$  Determine  $\alpha, \beta$  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.

**Solution:** We immediately see that  $\alpha = -1$  is necessary to get a polynomial of 3<sup>rd</sup> degree.

Furthermore, from the condition

$$8 = s_{-1, \beta}(1^-) = s_{-1, \beta}(1^+) = \beta + 8 + \frac{11}{3} \quad \text{we get} \quad \beta = -\frac{11}{3}.$$

It remains to check the continuity of  $s, s', s''$  in the nodes 0 and 1. Indeed, we have

$$s'_{-1, \beta}(x) = \begin{cases} 4(x+1)^3 + 4\alpha(x-1)^3 & x \in [-1, 0] \\ -3x^2 - 8\alpha & x \in (0, 1] \\ 3\beta x^2 + 16x & x \in (1, 2] \end{cases} \quad s''_{-1, \beta}(x) = \begin{cases} 12(x+1)^2 + 12\alpha(x-1)^2 & -1 \leq x \leq 0, \\ -6x & 0 < x \leq 1, \\ 6\beta x + 16 & 1 < x < 2. \end{cases}$$

Therefore the values in the nodes are

	0 <sup>-</sup>	0 <sup>+</sup>	1 <sup>-</sup>	1 <sup>+</sup>			0 <sup>-</sup>	0 <sup>+</sup>	1 <sup>-</sup>	1 <sup>+</sup>
$s$	$2 + \alpha$	$1$	$-8\alpha$	$\beta + 8 + 11/3$	$\alpha = -1,$ $\beta = -11/3 \rightarrow$	$s$	$1$	$1$	$8$	$8$
$s'$	$4 - 4\alpha$	$-8\alpha$	$-3 - 8\alpha$	$3\beta + 16$		$s'$	$8$	$8$	$5$	$5$
$s''$	$12 + 12\alpha$	$0$	$-6$	$6\beta + 16$		$s''$	$0$	$0$	$-6$	$-6$

They agree for our choice of the parameters.

(1b) ☐ Use MATLAB to create a plot of the function defined in (19) in dependence of  $\alpha$  and  $\beta$ .

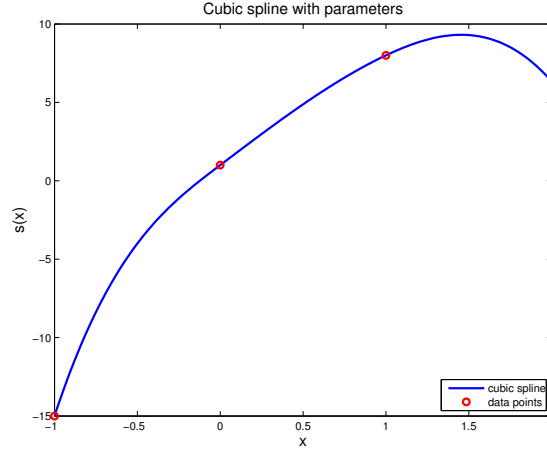
**Solution:**

Listing 34: Matlab Code for Cubic Spline

```

1 function ex_CubicSpline(alpha, beta)
2 if nargin==0; alpha=-1; beta=-11/3; end;
3
4 x1 = -1:0.01:0;
5 y1 = (x1+1).^4+alpha*(x1-1).^4+1;
6 x2 = 0:0.01:1;
7 y2 = -x2.^3-8*alpha*x2+1;
8 x3 = 1:0.01:2;
9 y3 = beta*x3.^3 + 8.*x3.^2+11/3;
10
11 x = [x1 x2 x3]; y = [y1 y2 y3];
12 nodes = [-1 0 1 2];
13 data = [y1(1) y1(end) y2(end) y3(end)];
14
15 close all;
16 plot(x,y,nodes,data,'ro','linewidth',2);
17 legend('cubic spline', 'data
    points','Location','SouthEast');
18 xlabel('x','fontsize',14); ylabel('s(x)','fontsize',14);
19 title('Cubic spline with parameters','fontsize',14)
20 print -depsc2 'ex_CubicSpline.eps'

```



## 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 1-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

$$s|_{[t_{j-1}, t_j]}(t) = d_j \tau^2 + c_j 4 \tau(1 - \tau) + d_{j-1} (1 - \tau)^2 , \quad \tau := \frac{t - t_{j-1}}{t_j - t_{j-1}} , \quad j = 1, \dots, n , \quad (20)$$

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 1-periodic spline functions in  $\mathcal{S}_{2,\mathcal{M}}$ ?

**Solution:** Counting argument, similar to that used to determine the dimensions of the spline spaces. We have  $n+1$  unknowns  $d_k$  and  $n$  unknowns  $c_j$ , the constraint  $d_0 = s(t_0) =$

$s(t_0 + 1) = s(t_n) = d_n$  leaves us with a total of  $2n$  unknowns. The continuity of the derivatives in the  $n$  nodes impose the same number of constraints, therefore the total dimension of the spline space is  $2n - n = n$ .

**(2b)**  $\square$  What kind of continuity is already guaranteed by the use of the representation (20)?

**Solution:** We observe that  $s(t_j^-) = s|_{[t_{j-1}, t_j]}(t_j) = d_j = s|_{[t_j, t_{j+1}]}(t_j) = s(t_j^+)$ , thus we get continuity for free. However the derivatives do not necessarily match.

**(2c)**  $\square$  Derive a linear system of equations (system matrix and right hand side) whose solution provides the coefficients  $c_j$  and  $d_j$  in (20) 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.

**Solution:** We can plug  $t = \frac{1}{2}(t_j + t_{j-1})$  into (20) and set the values equal to  $y_j$ . We obtain  $\tau = 1/2$  and the following conditions:

$$\frac{1}{4}d_j + c_j + \frac{1}{4}d_{j-1} = y_j, \quad j = 1, \dots, n. \quad (21)$$

We obtain conditions on  $d_j$  by matching the derivatives at the interfaces. The derivative of the quadratic spline can be computed from (20), after defining  $\Delta_j = t_j - t_{j-1}$ :

$$s'|_{[t_{j-1}, t_j]}(t) = \Delta_j^{-1} \frac{\partial}{\partial \tau} \left( \tau^2 d_j + 4\tau(1-\tau)c_j + (1-\tau)^2 d_{j-1} \right) = \Delta_j^{-1} \left( 2\tau d_j + 4(1-2\tau)c_j - 2(1-\tau)d_{j-1} \right).$$

Setting  $\tau = 1$  in  $[t_{j-1}, t_j]$  and  $\tau = 0$  in  $[t_j, t_{j+1}]$ , the continuity of the derivative in the node  $t_j$  enforces the condition

$$\frac{2d_j - 4c_j}{\Delta_j} = s'|_{[t_{j-1}, t_j]}(t_j) = s'(t_j^-) = s'(t_j^+) = s'|_{[t_j, t_{j+1}]}(t_j) = \frac{4c_{j+1} - 2d_j}{\Delta_{j+1}};$$

(this formula holds for  $j = 1, \dots, n$  if we define  $t_{n+1} = t_1 + 1$  and  $c_{n+1} = c_1$ ). Simplifying for  $d_j$  we obtain:

$$d_j = \frac{2\frac{c_j}{\Delta_j} + 2\frac{c_{j+1}}{\Delta_{j+1}}}{\frac{1}{\Delta_j} + \frac{1}{\Delta_{j+1}}} = 2 \frac{c_j \Delta_{j+1} + c_{j+1} \Delta_j}{\Delta_j + \Delta_{j+1}} = 2 \frac{c_j(t_{j+1} - t_j) + c_{j+1}(t_j - t_{j-1})}{t_{j+1} - t_{j-1}}, \quad j = 1, \dots, n.$$

Plugging this expression into (21), we get the following system of equations:

$$\frac{1}{2} \frac{c_{j+1}(t_j - t_{j-1}) + c_j(t_{j+1} - t_j)}{t_{j+1} - t_{j-1}} + c_j + \frac{1}{2} \frac{c_j(t_{j-1} - t_{j-2}) + c_{j-1}(t_j - t_{j-1})}{t_j - t_{j-2}} = y_j, \quad j = 1, \dots, n, \quad (22)$$

with the periodic definitions  $t_{-2} = t_{n-2} - 1$ ,  $c_0 = c_n$ ,  $c_{-1} = c_{n-1}$ . We collect the coefficients and finally we obtain


$$\underbrace{\left\{ \frac{1}{2} \frac{t_j - t_{j-1}}{t_j - t_{j-2}} \right\}}_{A_{j-1}} c_{j-1} + \underbrace{\left\{ \frac{1}{2} \frac{t_{j+1} - t_j}{t_{j+1} - t_{j-1}} + 1 + \frac{1}{2} \frac{t_{j-1} - t_{j-2}}{t_j - t_{j-2}} \right\}}_{B_j} c_j + \underbrace{\left\{ \frac{1}{2} \frac{t_j - t_{j-1}}{t_{j+1} - t_{j-1}} \right\}}_{C_{j+1}} c_{j+1} = y_j, \quad j = 1, \dots, n. \quad (23)$$

If we have an equidistant grid with  $t_j - t_{j-1} = 1/n$ , this simplifies to

$$\frac{1}{4}c_{j-1} + \frac{3}{2}c_j + \frac{1}{4}c_{j+1} = y_j, \quad j = 1, \dots, n.$$

The system of equations in matrix form looks as follows:

$$\begin{pmatrix} B_1 & C_2 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & A_0 \\ A_1 & B_2 & C_3 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & A_2 & B_3 & C_4 & 0 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & A_{n-2} & B_{n-1} & C_n \\ C_{n+1} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & A_{n-1} & B_n \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_{n-1} \\ c_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix}. \quad (24)$$

**(2d)**  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).

**Solution:** See Listing 35.

Listing 35: Construction of the quadratic spline and evaluation.

```

1 % part d of quadratic splines problem
2 % given: t = nodes (n-1 vect.),
3 %         y = data (n vect.),
4 %         x = evaluation pts (N vect.)
5 % create the interpolating quadratic spline and evaluate
   in x
6 function eval_x = quadspline_better(t,y,x)
7   % the number of nodes:
8   n = length(y);           % N = length(x);
9   % ensure nodes and data are line vectors:
10  t = t(:)'; y = y(:)';
11  % create (n+3) extended vectors using the periodicity:
12  ext_t = [t(end)-1,0,t,1,1+t(1)];
13  % increments in t:
14  de_t = diff(ext_t);           % (n+2)
15  dde_t = ext_t(3:end) - ext_t(1:end-2); % (n+1)
16
17  % build the three n-vectors that define the matrix
18  A = de_t(2:n+1) ./ (2*dde_t(1:n));
19  B = 1 + de_t(3:n+2) ./ (2*dde_t(2:n+1)) + ...
20     de_t(1:n) ./ (2*dde_t(1:n));
21  C = de_t(2:n+1) ./ (2*dde_t(2:n+1));
22  % Assembly of the matrix, be careful with spdiags and
   transpose!
23  M = spdiags([C; B; A].', (-1:1), n,n).';
24  %M(1,n) = A(1);
25  %M(n,1) = C(n);
26
27  %%%% Employ SMW formula for a rank 1 modification,
   which allows to
28  %%%% exploit the structure of the matrix
29  u = zeros(n,1);
30  v = zeros(n,1);
31  u(n,1) = 1;
32  u(1,1) = 1;
33  v(1,1) = C(n);
34  v(n,1) = A(1);

```

```

35
36 M(1,1) = M(1,1) - C(n);
37 M(n,n) = M(n,n) - A(1);
38
39 % solve the system for the coefficients c:
40 %c = (M\y(:)).';
41 Minvy = M \ y(:);
42
43 c = Minvy - (M\u*(v'*Minvy)) / (1 + v'*(M\u));
44 c = c';
45
46 % compute the coefficients d_1,...,d_n:
47 ext_c = [c,c(1)];
48 d = 2*( ext_c(2:end).*de_t(2:n+1)+ c.*de_t(3:n+2) ) ...
49     ./dde_t(2:n+1);
50 ext_d = [d(n),d];
51 % evaluate the interpolating spline in x:
52 eval_x = zeros(1,length(x));
53
54 % loop over the n intervals
55 % loop over the n intervals
56 ind_end = 0;
57 for i=1:n
58     left_t = ext_t(i+1);
59     right_t = ext_t(i+2);
60     % find the points in x in the interval [t(i-1),
61         t(i)):
62     ind_start = ind_end+1;
63     if x(ind_start) >= left_t
64         while ind_end < length(x) && x(ind_end+1) <
65             right_t
66                 ind_end = ind_end + 1;
67         end
68     end
69     ind_eval = ind_start:ind_end;
70     tau = (x(ind_eval) - left_t)./( right_t-left_t );
71     eval_x(ind_eval) = d(i)*tau.^2 + ...
72         c(i)*4*tau.*(1-tau) + ext_d(i)*(1-tau).^2;

```

```

71 end
72
73 end

```

It is important not to get lost in the indexing of the vectors. In this code they can be represented as:

```

t=(t1, t2, ..., t{n-1})    length = n - 1,
ext_t=(t{n-1}-1, t0=0, t1, t2,..., t{n-1}, tn=1, 1+t1)    length
= n + 3,
de_t=(1-t{n-1}, t1, t2-t1,..., t{n-1}-t{n-2}, 1-t{n-1}, t1) ,
dde_t=(t1+1-t{n-1}, t2, t3-t1,..., 1-t{n-2}, t1+1-t{n-1}) .

```

The vectors  $A, B, C, c, d$  have length  $n$  and correspond to the definitions given in (2c) (with the indexing from 1 to  $n$ ).

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

**Solution:** See code 36 and Figure 13.

Listing 36: Plot of the quadratic splines.

```

1 function ex_QuadraticSplinesPlot(f,n,N, func)
2
3 if nargin<3
4     f = @(t) exp(sin(2*pi*t));    % function to be
        interpolated
5     n = 10;                        % number of subintervals
6     N = 200;                       % number of evaluation
        points
7     func = @quadspline_better;
8 end
9
10 t = 1/n:1/n:1-1/n;                % n-1 equispaced nodes
        in (0,1)
11 x = linspace(0,1,N);               % N evaluation points
12 x = x(0 <= x & x < 1);             % be sure x belongs to
        [0,1)
13 middle_points = ([t,1]+[0,t])/2; % computes the n middle

```

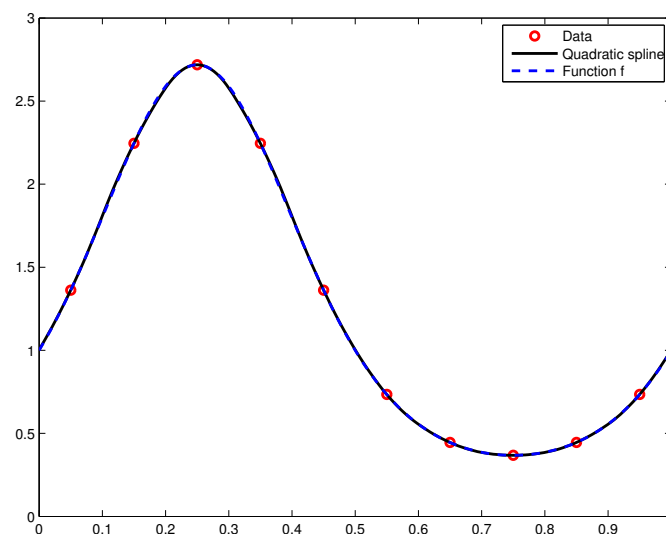


```

    points [0,t,1]
14 y = f(middle_points);           % evaluate f on the
    middle_points
15
16 % compute and evaluate the quadratic spline
17 eval_x = func(t,y,x);
18 %eval_x_better = quadspline(t,y,x);
19
20 close all; figure
21 plot(middle_points,y,'ro',x,eval_x,'k',x,f(x),'--','linewidth',2);
22 legend('Data','Quadratic spline','Function f');
23 print -depsc2 'interpol_quad_T2.eps'
24
25 end
26
27 % test:
28 % ex_QuadraticSplinesPlot(@ (t) sin(2*pi*t).^3, 10,500)

```

Figure 13: Quadratic spline interpolation of  $f$  in 10 equispaced nodes.

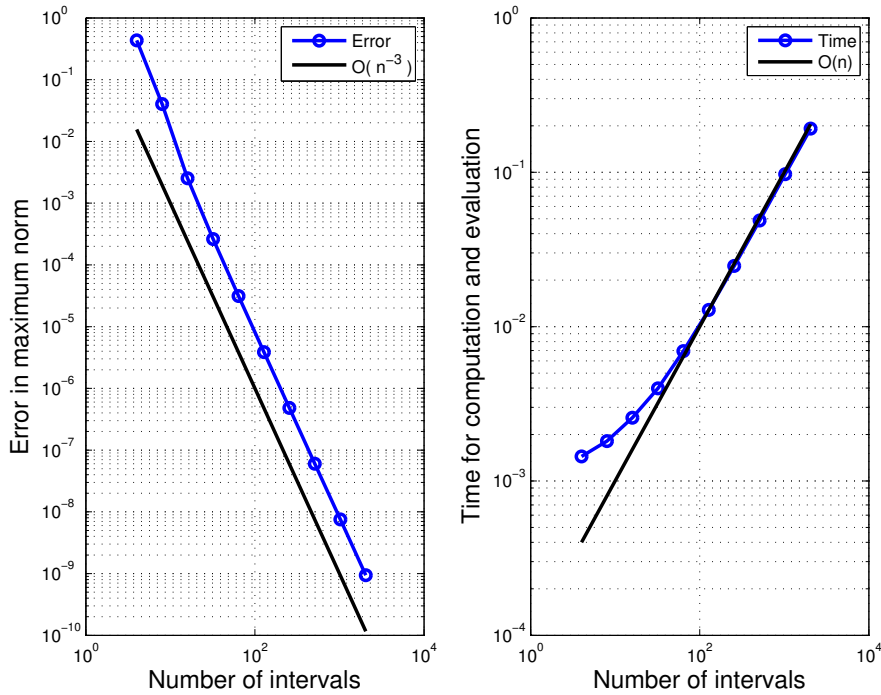


(2f) ☐ What is the complexity of the algorithm in (2d) in dependence of  $n$  and  $N$ ?

**Solution:** The complexity is linear both in  $N$  (sequential evaluation in different points)

and in  $n$ , provided one exploits the sparse structure of the system and uses a rank-one modification to reduce the system to the inversion of a (strictly diagonally dominant) tridiagonal matrix.

Figure 14: Error and timing for quadratic splines in equispaced nodes.



### 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 parametrizations for a given curve: for any strictly monotonous and surjective  $h : [0, 1] \rightarrow [0, 1]$


the mappings  $\gamma$  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:

$$\textcircled{1} \quad \text{equidistant parametrization: } t_k = k\Delta t, \Delta t = \frac{1}{n} \quad (25)$$

$$\textcircled{2} \quad \text{segment length parametrization: } t_k = \frac{\sum_{l=1}^k |\mathbf{p}_l - \mathbf{p}_{l-1}|}{\sum_{l=1}^n |\mathbf{p}_l - \mathbf{p}_{l-1}|}. \quad (26)$$

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`.

**Solution:** See Listing 37.

Listing 37: Matlab Code for `curveintp`

```
1 function [pol spl pch] = curveintp (xy,t,tt)
2
3 x = xy(1,:);
4 y = xy(2,:);
5
6 % compute the slopes at the extremes (necessary for
   complete spline):
7 x_start = (x(2)-x(1)) ./ (t(2)-t(1));
8 y_start = (y(2)-y(1)) ./ (t(2)-t(1));
9 x_end = (x(end)-x(end-1)) ./ (t(end)-t(end-1));
10 y_end = (y(end)-y(end-1)) ./ (t(end)-t(end-1));
```

```

11
12 % compute the interpolants
13 polx = intpolyval(t,x,tt);
14 poly = intpolyval(t,y,tt);
15
16 % complete splines, using the extended vector
17 splx = spline(t,[x_start x x_end],tt);
18 sply = spline(t,[y_start y y_end],tt);
19 pchx = pchip(t,x,tt);
20 pchy = pchip(t,y,tt);
21
22 pol = [polx; poly];
23 spl = [splx; sply];
24 pch = [pchx; pchy];
25
26 end

```

**(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 (25) and (26)

**Solution:** See Listing 38 and Figure 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 (25) and (26).

**Solution:** See Listing 37 and Figure 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 (25) and (26).

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

**Solution:** The code for the interpolation of the heart:

Listing 38: Matlab Code for heart\_Sol

```

1  function heart_Sol ()
2
3  xy = heart();
4  n = size(xy,2) - 1;
5
6  %evaluation points:
7  tt = 0:0.005:1;
8
9  figure;
10 hold on;
11
12 % equidistant parametrization of [0,1]:
13 t_eq = (0:n)/n;
14 [pol spl pch] = curveintp (xy,t_eq,tt);
15 subplot(1,2,1); xlabel('Equidistant Parametrization');
16 plot_interpolations (xy,pol,spl,pch);
17
18 % segment length parametrization of [0,1]:
19 t_seg = segment_param(xy);
20 [pol spl pch] = curveintp (xy,t_seg,tt);
21 subplot(1,2,2); xlabel('Segment Length Parametrization');
22 plot_interpolations (xy,pol,spl,pch);
23
24 hold off;
25 print -depsc2 '../PICTURES/ex_CurveIntp.eps'

```

```

26
27 end
28
29 % segment length parametrization of [0,1]:
30 function t_seg = segment_param (xy)
31     increments = sqrt(sum(diff(xy,1,2).^2));
32     t_seg = cumsum(increments);
33     t_seg = [0,t_seg/t_seg(end)];
34 end
35
36 % plotting function
37 function plot_interpolations (xy,pol,spl,pch)
38     plot(xy(1,:),xy(2,),'o', pol(1,:),pol(2,),'-.', ...
39         spl(1,:),spl(2,),'-', pch(1,:),pch(2,),'--',
40         'linewidth',2);
41     axis equal;
42     axis([-105 105 -105 105]);
43     legend('data','polynomial','spline','pchip','Location','Southoutside'
44 end

```

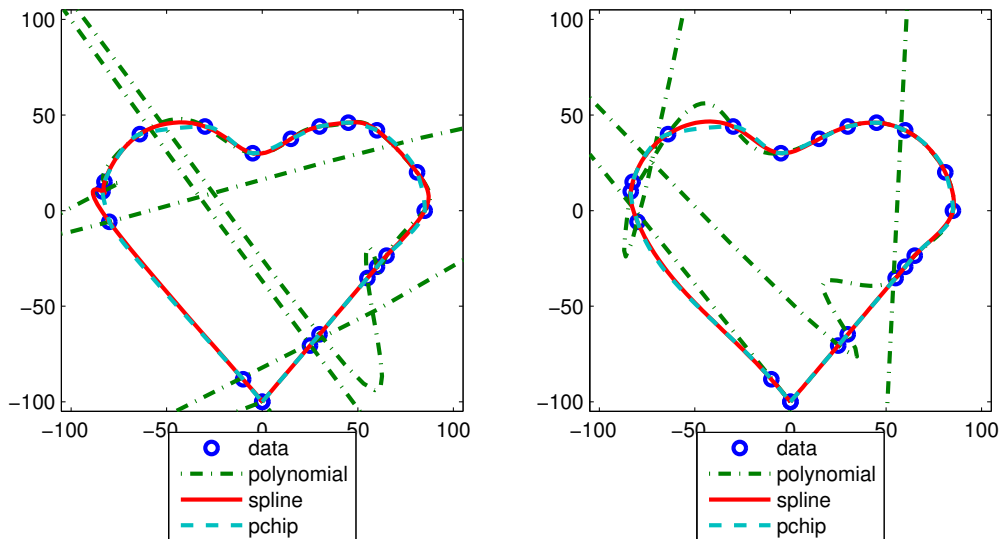


Figure 15: Plots for different parametrizations and interpolation schemes

**Remarks:**

- Note that `pchip` is a non-linear interpolation, so there is no linear mapping (matrix) from the nodes to the polynomial coefficients.
- Global polynomial interpolation fails. The degree of the polynomial is too high ( $n$ ), and the typical oscillations can be observed.
- In this case, when two nodes are too close, spline interpolation with equidistant parametrization introduces small spurious oscillations.

**Problem 4. Approximation of  $\pi$** 

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

$$x^* = \lim_{n \rightarrow \infty} T(n) , \quad (27)$$

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

$$x^* = \lim_{h \rightarrow 0} g(h) . \quad (28)$$

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  $\pi$  by extrapolation from the data in the above table, using the first  $k$  values,  $k = 1, \dots, 7$ .

**Solution:** See `pi_approx.cpp`.

Issue date: 29.10.2015

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

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