Aufgabe	A1	A2	A3	A4	A5	$\sum$
Punkte						

Aufgabe 1. Beh.: Seien n paarweise verschiedene Stützstellen $\{x_0, \dots, x_{n-1}\}$ gegeben und eine Permutation derselben $\{\tilde{x}_0, \dots, \tilde{x}_{n-1}\}$. Dann gilt

$$f[x_0, \dots, x_{n-1}] = f[\tilde{x}_0, \dots, \tilde{x}_{n-1}].$$

Beweis. Es gilt nach VL mit der Newtondarstellung für das Interpolationspolynom zu den Stützstellen $x_0, \dots, x_{n-1}$:

$$\begin{aligned}
p(x) &= \sum_{i=0}^{n-1} y[x_0, \dots, x_i] N_i(x) \\
&= \sum_{i=0}^{n-2} y[x_0, \dots, x_i] N_i(x) + y[x_0, \dots, x_{n-1}] N_{n-1}(x) \\
&= \mathcal{O}(x^{n-2}) + y[x_0, \dots, x_{n-1}] \prod_{i=0}^{n-2} (x - x_i) \\
&= \mathcal{O}(x^{n-2}) + y[x_0, \dots, x_{n-1}] \left(x^{n-1} + \mathcal{O}(x^{n-2})\right) \\
&= y[x_0, \dots, x_{n-1}] x^{n-1} + \mathcal{O}(x^{n-2}).
\end{aligned}$$

Der Leitkoeffizient des Interpolationspolynoms in der Monombasis ist also $y[x_0, \dots, x_{n-1}]$. Dieser ist unabhängig von der Reihenfolge der Stützstellen. Damit folgt die Behauptung. □

Aufgabe 2. Beh.: Für $N \geq 2\pi 10^3$ gilt $\max\limits_{0 \leq x \leq 1} |f(x) - s(x)| < 10^{-12}$.

Beweis. Es ist $f \in C^4([0,1])$. Dann gilt nach VL

$$\delta := \max_{0 \leq x \leq 1} |f(x) - s(x)| \leq h^4 \max_{0 \leq x \leq 1} |f^{(4)}(x)|.$$

Mit $f(x) = \sin(2\pi x)$ folgt sofort

$$f^{(4)}(x) = 16\pi^4 \sin(2\pi x) \quad \text{also} \quad \max_{0 \leq x \leq 1} |f^{(4)}(x)| = 16\pi^4.$$

Mit $h = \frac{1}{N}$ ergibt sich

$$\delta \leq 16 \frac{\pi^4}{N^4} \implies N \geq 2\pi \sqrt[4]{\delta} = 2\pi 10^3.$$

□

Aufgabe 3. Beh.: Das komplexe trigonometrische Interpolationspolynom ist gegeben als

$$t^*(x) = \frac{1}{2} - \frac{1}{4} e^{ix} - \frac{1}{4} e^{3ix}.$$

Beweis. Die Stützstellen sind als $x_j = \frac{2\pi j}{4}$, $j = 0, \dots, 3$ gegeben. Damit folgt

$$\begin{aligned}
f(x_0) &= f(0) = \min\{0, 2\} = 0 \\
f(x_1) &= f\left(\frac{\pi}{2}\right) = \min\left\{\frac{1}{2}, \frac{3}{2}\right\} = \frac{1}{2} \\
f(x_2) &= f(\pi) = \min\{1, 1\} = 1 \\
f(x_3) &= f\left(\frac{3}{2}\pi\right) = \min\left\{\frac{3}{2}, \frac{1}{2}\right\} = \frac{1}{2}.
\end{aligned}$$

Die Interpolationsbedingung ist erfüllt, denn

$$\begin{aligned}
t^*(x_0) &= \frac{1}{2} - \frac{1}{4} - \frac{1}{4} = 0 = f(x_0) \\
t^*(x_1) &= \frac{1}{2} - \frac{1}{4}\underbrace{e^{i\frac{\pi}{2}}}_{=i} - \frac{1}{4}\underbrace{e^{i\frac{3}{2}\pi}}_{=-i} = \frac{1}{2} = f(x_1) \\
t^*(x_2) &= \frac{1}{2} - \frac{1}{4}\underbrace{e^{i\pi}}_{=-1} - \frac{1}{4}\underbrace{e^{i\pi}}_{=-1} = 1 = f(x_2) \\
t^*(x_3) &= \frac{1}{2} - \frac{1}{4}\underbrace{e^{i\frac{3}{2}\pi}}_{=i} - \frac{1}{4}\underbrace{e^{i\frac{3}{2}\pi}}_{=-i} = \frac{1}{2} = f(x_3).
\end{aligned}$$

Aus der Eindeutigkeit des komplexen trigonometrischen Interpolationspolynoms folgt die Behauptung. □

Aufgabe 4. a) (1) Es ist

$$f_1''(x) = 6x - 14 \implies f_1''(0) = -14 \neq 0.$$

Also erfüllt f_1 nicht die natürlichen Randbedingungen, also $f_1 \notin S(X)$.

(2) Es gilt für $0 \leq x < 1$:

$$\begin{aligned}
f_2(x) &= -\frac{1}{2}x^3 \xrightarrow{x\to 1} -\frac{1}{2} \\
f_2'(x) &= -\frac{3}{2}x^2 \xrightarrow{x\to 1} -\frac{3}{2} \\
f_2''(x) &= -3x \xrightarrow{x\to 1} -3 \text{ und } f_2''(0) = 0
\end{aligned}$$

Für $1 \leq x \leq 2$ gilt:

$$\begin{aligned}
f_2(x) &= (x-1)^3 - \frac{1}{2}x^3 \implies f_2(1) = -\frac{1}{2} \\
f_2'(x) &= 3(x-1)^2 - \frac{3}{2}x^2 \implies f_2'(1) = -\frac{3}{2} \\
f_2''(x) &= 3x - 6 \implies f_2''(1) = -3 \text{ und } f_2''(2) = 0.
\end{aligned}$$

f_2 ist auf beiden Teilintervallen ein Polynom von Grad 3 und damit auf den Teilintervallen beliebig oft stetig differenzierbar. Außerdem ist f_2 2 mal stetig differenzierbar an der Stelle 1, also insgesamt $f_2 \in C^2([0,2])$. Die natürlichen Randbedingungen sind außerdem erfüllt, also folgt $f_2 \in S(X)$.

(3) Es ist

$$f_3''(x) = 6x - 2 \implies f_3''(0) = -2 \neq 0.$$

Also erfüllt f_3 nicht die natürlichen Randbedingungen, also $f_3 \notin S(X)$.

b) Der interpolierende Spline s von $f(x) = x^3$ folgt mit der Darstellung der VL direkt als

$$s(x) = \begin{cases} 1 + 4x + 4,5x^2 + 1,5x^3 & x \in [0,1) \\ 8 + 8,5(x-1) - 1,5(x-1)^3 & x \in [1,2] \end{cases}.$$

Aufgabe 5. Auszüge aus *splines.cc*:

```
1  template<typename REAL>
2  void solveTriDiag(hdnum::DenseMatrix<REAL> &A, std::vector<REAL> &x, std::vector<REAL>
       &b) {
3      int N = b.size();
4      x[0] = A[0][0];
5      // LU Zerlegung
6      REAL l;
7      for (int j=1; j<N; j++) {
8          // berechne l faktor
9          l = A[j][j-1] / x[j-1];
10         // modifiziere diagonalelemente und rechte seite
11         x[j] = A[j][j] - l * A[j-1][j];
12         b[j] = b[j] - l * b[j-1];
13     }
14     // Loesen durch Rueckwaertseinsetzen
```

```
    x[N-1] = b[N-1] / x[N-1];
    for (int j=N-2; j>=0; j--) {
        x[j] = (b[j] - A[j][j+1]*x[j+1])/x[j];
    }
}
```

Schneller Löser für tridiagonale Matrizen

Implementation in einer Klasse `CubicSpline`. Die Funktion `getCubicSpline` entspricht dem ersten Konstruktor.

```
template<typename REAL>
class CubicSpline {
    public:
        // erstelle einen kubischen spline mit vorgegebenen stuetzstellen
        // und werten
        CubicSpline(std::vector<REAL> xs, std::vector<REAL> ys) {
            calculateCoefficients(xs, ys);
        }

        // erstelle einen kubischen spline mit vorgegebenen stuetzstellen
        // und einer zu interpolierenden funktion
        CubicSpline(std::vector<REAL> xs, REAL(*f)(REAL)) {
            int N = xs.size();
            std::vector<double> ys(N);
            for (int i=0; i<N; i++) {
                ys[i] = f(xs[i]);
            }
            calculateCoefficients(xs, ys);
        }

        // erstelle einen kubischen spline an aequidistanten stuetzstellen
        // und einer zu interpolierenden funktion
        CubicSpline(REAL a, REAL b, int N, REAL(*f)(REAL)) {
            std::vector<double> xs(N+1);
            std::vector<double> ys(N+1);
            for (int i=0; i<=N; i++) {
                xs[i] = a + (1.0*i)/N*(b-a);
                ys[i] = f(xs[i]);
            }
            calculateCoefficients(xs, ys);
        }

        // werte kubischen spline an vorgegebener stelle aus
        REAL evaluate(REAL x) {
            for (int i=1; i<x_s.size(); i++) {
                if (x > x_s[i] && i < x_s.size() - 1) {
                    continue;
                } else {
                    return a_0[i-1] + a_1[i-1] *(x - x_s[i]) + a_2[i-1]*std::pow(x -
    x_s[i], 2) + a_3[i-1]*std::pow(x-x_s[i], 3);
                }
            }
            return 0;
        }

        // gebe alle interpolations polynome aus
        void print() {
            for(int i=0; i<x_s.size()-1; i++) {
                printf("p_%d(x) = %4.2f + %4.2f(x - %4.2f) + %4.2f(x - %4.2f)^2 + %4.2
    f(x - %4.2f)^3\n",
                        i, a_0[i], a_1[i], x_s[i], a_2[i], x_s[i], a_3[i], x_s[i]);
            }
        }

    private:
        // stuetzstellen
        std::vector<REAL> x_s;
        // koeffizienten
        std::vector<REAL> a_0;
        std::vector<REAL> a_1;
        std::vector<REAL> a_2;
        std::vector<REAL> a_3;

        void calculateCoefficients(std::vector<REAL> xs, std::vector<REAL> ys) {
            // stuetzstellen from x_0 ... to x_n
```

```
64              int n = xs.size()-1;
65              // copy stuetzstellen
66              x_s = std::vector<double>(n+1);
67              x_s = xs;
68              // setup (n-1)x(n-1) matrix for a_2
69              hdnum::DenseMatrix<REAL> A(n-1,n-1);
70              std::vector<REAL> b(n-1);
71              std::vector<REAL> x(n-1);
72              REAL h; // h_i
73              REAL h1; // h_{i+1}
74              // setup LGS for a_2
75              // A has tridiagonal structure
76              for (int i = 1; i<n; i++) {
77                  h = xs[i] - xs[i-1];
78                  h1 = xs[i+1] - xs[i];
79                  b[i-1] = 3 * ((ys[i+1] - ys[i])/h1 - (ys[i] - ys[i-1])/h);
80                  if (i > 1) {
81                      A[i-1][i-2] = h;
82                  } if (i < n-1) {
83                      A[i-1][i] = h1;
84                  }
85                  A[i-1][i-1] = 2 * (h + h1);
86              }
87              // initialize vectors
88              a_0 = std::vector<double>(n);
89              a_1 = std::vector<double>(n);
90              a_2 = std::vector<double>(n);
91              a_3 = std::vector<double>(n);
92              solveTriDiag(A,a_2,b);
93              // natuerliche randbedingung
94              a_2[n-1] = 0;
95              // berechne restliche koeffizienten
96              for (int i = 1; i<=n; i++) {
97                  h = xs[i] - xs[i-1]; // h_i
98                  h1 = xs[i+1] - xs[i]; // h_{i+1}
99                  a_0[i-1] = ys[i];
100                 if (i == 1) { // a_2[-1] = 0
101                     a_1[i-1] = (ys[i] - ys[i-1])/h + (h/3)*(2*a_2[i-1]);
102                     a_3[i-1] = (a_2[i-1])/(3*h);
103                 } else {
104                     a_1[i-1] = (ys[i] - ys[i-1])/h + h/3*(2*a_2[i-1] + a_2[i-2]);
105                     a_3[i-1] = (a_2[i-1] - a_2[i-2])/(3 * h);
106                 }
107             }
108         }
109 };
```

Konstruktion und Auswertung eines kubischen Splines

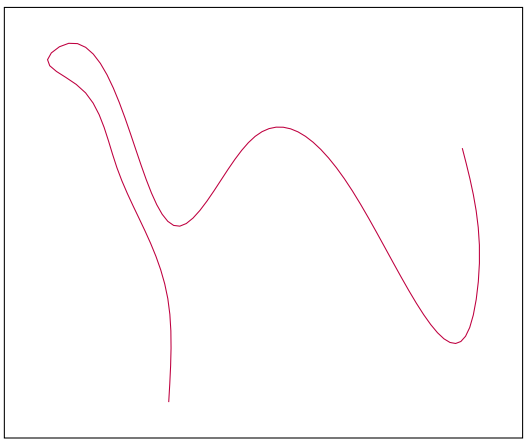

Abbildung 1: Rekonstruktion des Sauriers