NumCSE

Autumn Semester 2017

Prof. Rima Alaifari

Exercise sheet 8
Splines

A. Dabrowski

Problem 8.1: Cubic splines

We implement interpolation of a discrete data set by a cubic spline.

Template: `CubicSplines.cpp`

Recall that the cubic spline s interpolating a given data set $(t_0, y_0), \ldots, (t_n, y_n)$ is a C^2 function on $[t_0, t_n]$ which is a polynomial of third degree on every subinterval $[t_j, t_{j+1}]$ for $j = 0, \ldots, n-1$, and such that $s(t_j) = y_j$ for every $j = 0, \ldots, n$. To ensure uniquenss we impose the additional boundary conditions $s''(t_0) = s''(t_n) = 0$.

Recall that since we can represent a polynomial of degree d as a vector of length $d+1$ which contains the polynomial's coefficients, a cubic spline on a data set of length $n+1$ can be represented as a $4 \times n$ matrix, where the column j specifies the coefficients of the interpolating polynomial on the interval $[t_j, t_j + 1]$.

(a) Implement a `C++` function `cubicSpline` which takes as input vectors T and Y, and returns the matrix representing the cubic spline which interpolates them.

Hint: implement the formulae from the tablet notes to calculate the second derivatives of the splines in the points t_j, then use them to build the matrix associated to the spline.

SOLUTION:

```
MatrixXd cubicSpline(const VectorXd &T, const VectorXd &Y) {
    // returns the matrix representing the spline interpolating the data
    // with abscissae T and ordinatae Y. Each column represents the coefficients
    // of the cubic polynomial on a subinterval.
    // Assumes T is sorted, has no repeated elements and T.size() == Y.size().

    int n = T.size() - 1; // T and Y have length n+1

    VectorXd h = T.tail(n) - T.head(n); // vector of lengths of subintervals

    // build the matrix of the linear system associated to the second derivatives
    MatrixXd A = MatrixXd::Zero(n-1, n-1);
    A.diagonal()   = (T.segment(2,n-1) - T.segment(0,n-1))/3;
    A.diagonal(1)  = h.segment(1,n-2)/6;
    A.diagonal(-1) = h.segment(1,n-2)/6;

    // build the vector of the finite differences of the data Y
    VectorXd slope = (Y.tail(n) - Y.head(n)).cwiseQuotient(h);

    // right hand side vector for the system with matrix A
    VectorXd r = slope.tail(n-1) - slope.head(n-1);

    // solve the system and fill vector of second derivatives
    VectorXd sigma(n+1);
    sigma.segment(1,n-1) = A.partialPivLu().solve(r);
    sigma(0) = 0; // "simple" boundary conditions
    sigma(n) = 0; // "simple" boundary conditions

    // build the spline matrix with polynomials' coefficients
    MatrixXd spline(4, n);
```

```
    spline.row(0) = Y.head(n);
    spline.row(1) = slope - h.cwiseProduct(2*sigma.head(n) + sigma.tail(n))/6;
    spline.row(2) = sigma.head(n)/2;
    spline.row(3) = (sigma.tail(n) - sigma.head(n)).cwiseQuotient(6*h);

    return spline;
}
```

(b) Implement a C++ function which given a cubic spline, its interpolation nodes and a vector of evaluation points, returns the value the spline takes on the evaluation points.

SOLUTION:

```
VectorXd evalCubicSpline(const MatrixXd &S, const VectorXd &T, const VectorXd
    &evalT) {
    // Returns the values of the spline S calculated in the points X.
    // Assumes T is sorted, with no repetetions.

    int n = evalT.size();
    VectorXd out(n);

    for (int i=0; i < n; i++) {
        for (int j=0; j < T.size()-1; j++) {
            if (evalT(i) < T(j+1) || j==T.size()-2) {
                double x = evalT(i) - T(j);
                out(i) = S(0,j) + x*(S(1,j) + x*(S(2,j) + x*S(3,j)));
                break;
            }
        }
    }

    return out;
}
```

(c) Run some tests of your spline evaluation function (see template).

Problem 8.2: Piecewise linear approximation on graded meshes

The quality of an interpolation depends heavily on the choice of the nodes: for instance if the function to be interpolated has very large derivatives on a part of the domain, more interpolation points will be required there. Commonly used tools to cope with this task are *graded meshes*, which are explored in this problem.

Given a mesh $\mathcal{T} = \{0 \le t_0 < t_1 < \cdots < t_n \le 1\}$ on the unit interval $I = [0, 1]$, we define the *piecewise linear interpolant*:

$$\mathsf{I}_{\mathcal{T}} : C^0(I) \to \mathcal{P}_{1,\mathcal{T}} = \{s \in C^0(I),\ s_{|[t_{j-1},t_j]} \in \mathcal{P}_1 \ \forall\, j\}, \quad \text{s.t.} \quad (\mathsf{I}_{\mathcal{T}} f)(t_j) = f(t_j), \quad j = 0, \ldots, n.$$

(a) If we choose the uniform mesh $\mathcal{T} = \{t_j\}_{j=0}^n$ with $t_j = j/n$, given a function $f \in C^2(I)$ what is the asymptotic behavior of the error $\max_{x \in I} |f(x) - \mathsf{I}_{\mathcal{T}} f(x)|$ when $n \to \infty$?

Hint: use the following property of the interpolating polynomial: for every j, there exists $\xi_j \in [t_j, t_{j+1}]$ such that

$$f(t) - p_j(t) = \frac{f''(\xi_j)}{6}(t - t_j)(t - t_{j+1}), \qquad \text{for } t \in [t_j, t_{j+1}],$$

where p_j is the linear interpolant of f in $[t_j, t_{j+1}]$.

SOLUTION:

From the previous identity we have

$$\|f - \mathsf{I}_{\mathcal{T}} f\|_{L^\infty(I)} \le \frac{1}{2n^2} \left\| f^{(2)} \right\|_{L^\infty(I)} \tag{8.1}$$

because the meshwidth is $h = 1/n$. Thus, the convergence is quadratic, i.e. algebraic with order 2.

(b) What is the regularity of the function $f : I \to \mathbb{R},\ f(t) = t^\alpha,\ 0 < \alpha < 2$? In other words, for which $k \in \mathbb{N}$ do we have $f \in C^k(I)$?

Hint: check the continuity of the derivatives in the endpoints of I.

SOLUTION:

If $\alpha = 1$, $f(t) = t$ clearly belongs to $C^\infty(I)$. If $0 < \alpha < 1$, $f'(t) = \alpha t^{\alpha-1}$ blows up to infinity for t going to 0, therefore $f \in C^0(I) \setminus C^1(I)$. If $1 < \alpha < 2$, f' is continuous but $f''(t) = \alpha(\alpha-1)t^{\alpha-2}$ blows up to infinity for t going to 0, therefore $f \in C^1(I) \setminus C^2(I)$.

More generally, for $\alpha \in \mathbb{N}$ we have $f(t) = t^\alpha \in C^\infty(I)$; on the other hand, if $\alpha > 0$ is not an integer, $f \in C^{\lfloor \alpha \rfloor}(I)$, where $\lfloor \alpha \rfloor = \mathrm{floor}(\alpha)$ is the largest integer not larger than α.

(c) Study with some numerical experiments the convergence of the piecewise linear approximation of $f(t) = t^\alpha$ (with $0 < \alpha < 2$) on uniform meshes.

SOLUTION:

```
VectorXd evalPiecewiseInterp(const VectorXd &T, const VectorXd &Y, const
    VectorXd &evalT) {
    // returns the values of the piecewise linear interpolant in evalT.

    int n = evalT.size();
    VectorXd out(n);

    for (int i=0; i < n; i++) {
        for (int j=0; j < T.size()-1; j++) {
            if (evalT(i) < T(j+1) || j==T.size()-2) {
                double slope = (Y(j+1) - Y(j)) / (T(j+1) - T(j));
                out(i) = Y(j) + slope * (evalT(i) - T(j));
                break;
            }
        }
    }

    return out;
}

double maxInterpError(double a, VectorXd T, VectorXd evalT) {

    int nInterpPts = T.size();
    VectorXd Y(nInterpPts);

    for (int i=0; i<nInterpPts; i++) {
        Y(i) = std::pow(T(i), a);
    }

    VectorXd evalInterp = evalPiecewiseInterp(T, Y, evalT);

    double maxError = 0;
    for (int i=0; i<evalT.size(); i++) {
        double error = std::abs(evalInterp(i) - std::pow(evalT(i), a));
        if (error > maxError)
            maxError = error;
    }

    return maxError;
}
```

```
    {// vary n, keep fixed alpha, uniform meshes
    int nTests = 10;
    int nEvalPts = 1 << 12;
    VectorXd evalT = VectorXd::LinSpaced(nEvalPts, 0, 1);
    VectorXd maxErrors(nTests);

    for (int n=0; n<nTests; n++) {
        int nInterpNodes = 1 << n;
        VectorXd T = VectorXd::LinSpaced(nInterpNodes, 0, 1);
        maxErrors(n) = std::log(maxInterpError(0.531, T, evalT));
    }

    mglData daty;
```

```
    daty.Link(maxErrors.tail(nTests).data(), nTests);
    mglGraph gr;
    gr.SetRanges(0, nTests, -6, 0);
    gr.Axis();
    gr.Plot(daty);
    gr.WriteFrame("uniformMesh_interpMaxErrorLog_varN.eps");
    }
```

Looking at the plots, the convergence is clearly algebraic: the rate is equal to α if it is smaller than 2, and equal to 2 otherwise. In brief, we can say that the order is $\min\{\alpha, 2\}$.

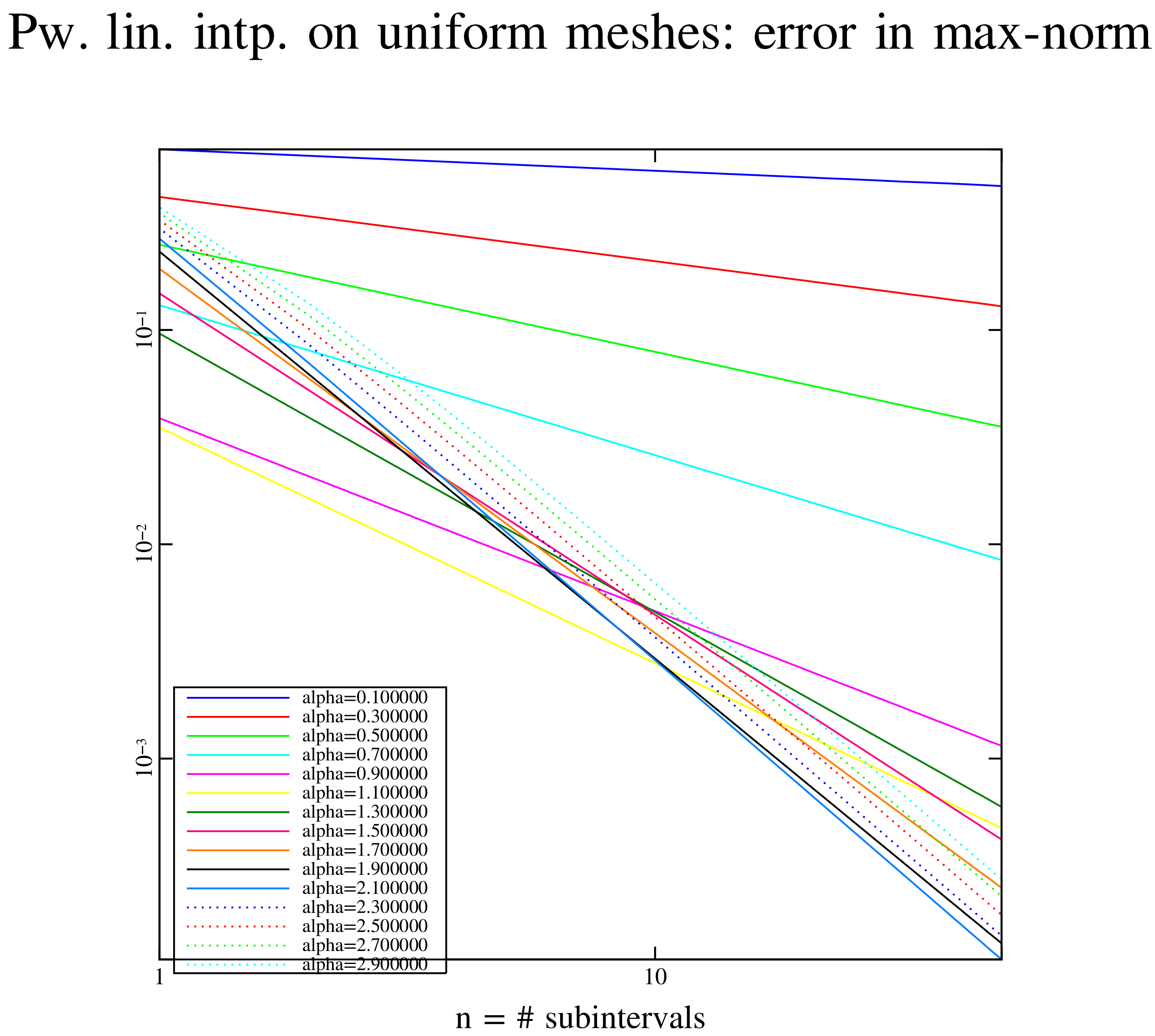

Fig. 1

(d) In which mesh interval do you expect $|f - \mathsf{I}_{\mathcal{T}} f|$ to attain its maximum?

SOLUTION:

The error representation in the linear case ($n = 1$) for some $\tau_t \in (t_j, t_{j+1})$ reads as:

$$\forall\, t \in (t_j, t_{j+1}) \qquad \left|f(t) - \big(\mathsf{I}_{\mathcal{T}} f\big)(t)\right| = \frac{1}{2}|f''(\tau_t)\ (t - t_j)(t - t_{j+1})|$$
$$\leq \frac{1}{8}|f''(\tau_t)|\ (t_{j+1} - t_j)^2 = \frac{1}{8n^2}|f''(\tau_t)|$$

Therefore the error can be large only in the subintervals where the second derivative of f is large. $|f''(t)| = \left|\alpha(\alpha - 1)t^{\alpha-2}\right|$ is monotonically decreasing for $0 < \alpha < 2$: therefore, we can expect a large error in the first subinterval, the one that is closer to 0.

(e) Compute by hand the exact value of $\|f - \mathsf{I}_{\mathcal{T}} f\|_{L^\infty(I)}$. Use the result of the Point (d) to simplify the problem. Compare the order of convergence obtained with the one observed numerically.

SOLUTION:

From Point (d) we expect that the maximum appears in the first subinterval. Then, for $t \in (0, 1/n)$ and $0 < \alpha < 2$ (with $\alpha \neq 1$), let us find this maximum by considering the error function φ:

$$\varphi(t) = f(t) - \big(\mathsf{I}_{\mathcal{T}} f\big)(t) = t^\alpha - t\,\frac{1}{n^{\alpha-1}}, \qquad\qquad (\varphi(0) = \varphi(1/n) = 0)$$
$$\varphi'(t) = \alpha t^{\alpha-1} - \frac{1}{n^{\alpha-1}}$$
$$\varphi'(t^*) = 0 \quad \text{if} \quad t^* = \frac{1}{n}\alpha^{-1/(1-\alpha)}$$
$$\max_{t\in(0,1/n)} |\varphi(t)| = |\varphi(t^*)| = \left|\frac{\alpha^{-\alpha/(1-\alpha)}}{n^\alpha} - \frac{\alpha^{-1/(1-\alpha)}}{n^\alpha}\right| = \frac{1}{n^\alpha}\left|\alpha^{-\alpha/(1-\alpha)} - \alpha^{-1/(1-\alpha)}\right| = \mathcal{O}(n^{-\alpha}) = \mathcal{O}(h^\alpha)$$

The order of convergence in $h = 1/n$ is equal to the parameter α, as in a certain way observed in Fig. 1. This plot is however skewed by the presence of measurements for $\alpha \sim 1$, given that the interpolant exactly captures $f(t) = t^\alpha$ for $\alpha = 1$.

(f) Since the interpolation error is concentrated in the left part of the domain, it seems reasonable to use a finer mesh only in this part. A common choice is an *algebraically graded mesh*, defined as $\mathcal{G} = \left\{t_j = \left(\frac{j}{n}\right)^\beta, \quad j = 0, \ldots, n\right\}$ for a parameter $\beta > 1$. An example is depicted in Fig. 2 for $\beta = 2$.

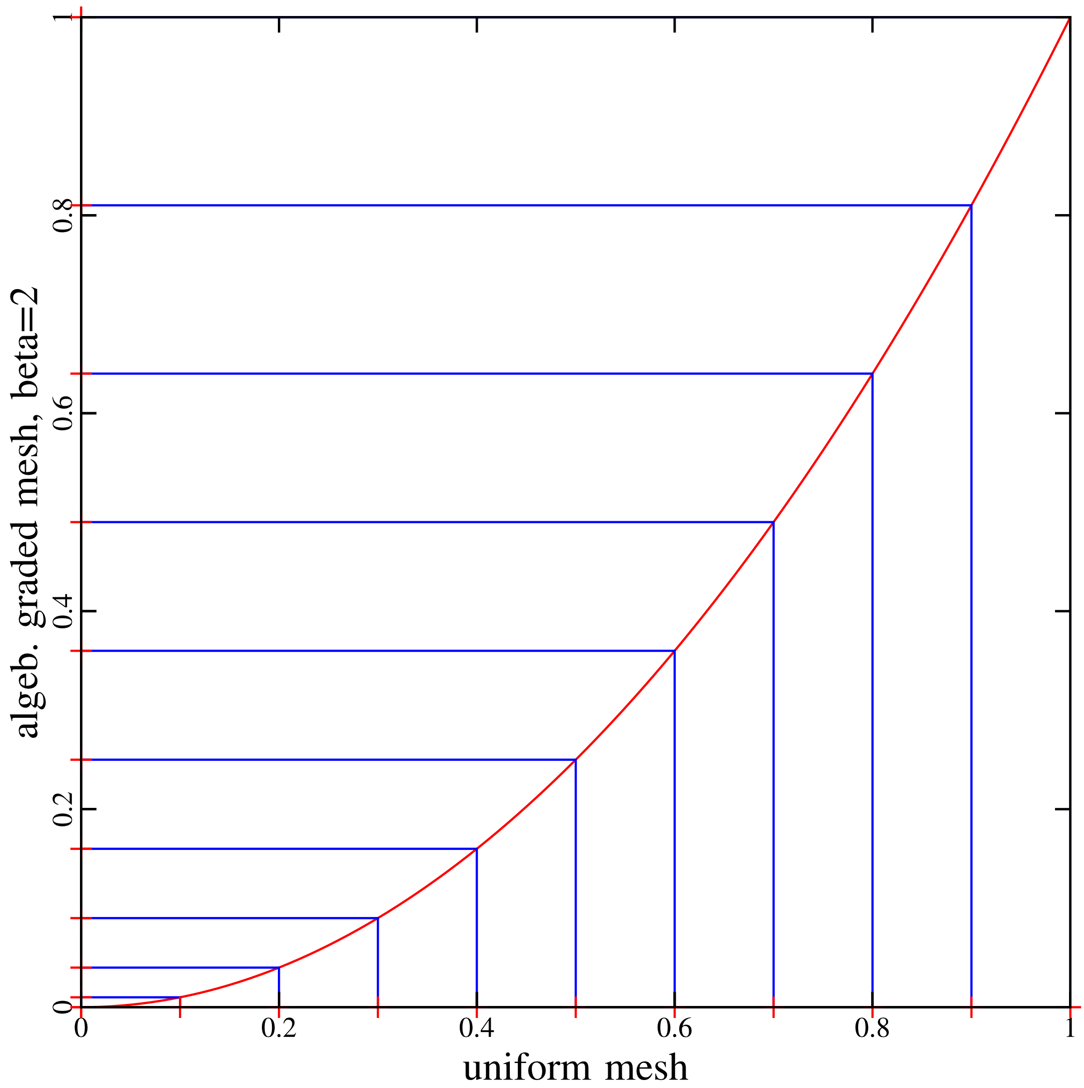

Fig. 2

For a fixed parameter α in the definition of f, determine with a numerical experiment the rate of convergence of the piecewise linear interpolant $\mathsf{I}_{\mathcal{G}}$ on the graded mesh $\mathcal{G}$ as a function of the parameter β. Try for instance $\alpha = 1/2$, $\alpha = 3/4$ or $\alpha = 4/3$.

How do you have to choose β in order to recover the optimal rate $\mathcal{O}(n^{-2})$ (if possible)?

SOLUTION:

```
    double a = 0.5; // varying this manually
    double b = 2/a; // varied this manually to find best value;
                    // better idea:  vary it automatically, using
                    // linear regression to estimate convergence rate.

    {// vary n, keep fixed alpha = 1/2, graded mesh
    int nTests = 10;
    int nEvalPts = 1 << 12;
    VectorXd evalT = VectorXd::LinSpaced(nEvalPts, 0, 1);
```

```
    VectorXd maxErrors(nTests);

    for (int n=0; n<nTests; n++) {
        int nInterpNodes = 1 << n;
        VectorXd T = VectorXd::LinSpaced(nInterpNodes, 0, 1);
        for (int i=0; i<nInterpNodes; i++) {
            T(i) = std::pow(T(i), b);
        }

        maxErrors(n) = std::log(maxInterpError(0.5, T, evalT));
    }

    mglData daty;
    daty.Link(maxErrors.tail(nTests).data(), nTests);
    mglGraph gr;
    gr.SetRanges(0, nTests, -16, 0);
    gr.Axis();
    gr.Plot(daty);
    gr.WriteFrame("gradedMesh_interpMaxErrorLog_varN.eps");
    }
```

The comparison of the plots for different values of α suggests that the choice of $\beta = 2/\alpha$ guarantees quadratic convergence.

Proceeding as in (d), we can see that the maximal error in the first subinterval $(0, t_1) = (0, 1/n^\beta)$ is equal to $1/n^{\alpha\beta}\,(\alpha^{-\alpha/(1-\alpha)} - \alpha^{-1/(1-\alpha)}) = \mathcal{O}(n^{-\alpha\beta})$ (replace the interval size $1/n$ with $1/n^\beta$ in those equations). This implies that a necessary condition to have quadratic convergence is $\beta \geq 2/\alpha$. In order to find an upper bound on the optimal β, we should control the error committed in every subinterval. If we were not satsfied by the numerical results, a more rigorous derivation of the optimal β could be obtained with the computation of $\varphi(t^*)$.