Numerical Methods for CSE
Final Exam HS2018

Prof. Rima Alaifari

ETH Zürich, D-MATH

January 29, 2019

Exercise 1. *Curve fitting* (26 pts) [*Template: 1.cpp*]

(a) (2 pts) Let $n \geq 3$ and $\mathbf{x} := (x_1, \ldots, x_n)^\top, \mathbf{y} := (y_1, \ldots, y_n)^\top \in \mathbb{R}^n$, where the entries of $\mathbf{x}$ are distinct. We want to determine the coefficients $\mathbf{c} := (c_0, c_1, c_2)^\top \in \mathbb{R}^3$ of a parabola

$$p_{\mathbf{c}}(x) := c_2 x^2 + c_1 x + c_0$$

such that

$$E(\mathbf{c}) := \sum_{i=1}^{n} |p_{\mathbf{c}}(x_i) - y_i|^2$$

is minimized (see Figure 1).

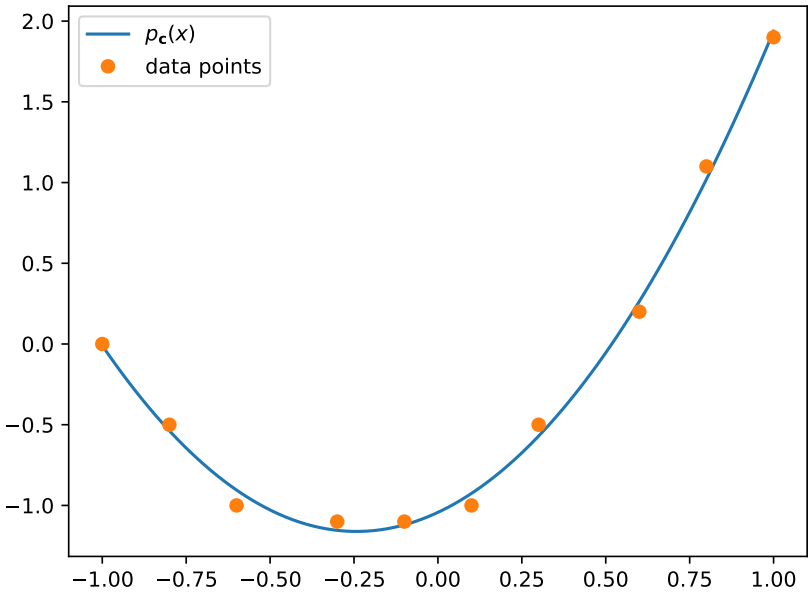

Figure 1: The parabola $p_{\mathbf{c}}(x)$ fitted to the data points $\{(x_i, y_i)\}_{i=1}^n$.

Formulate this as a linear *least squares problem*: For given $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ as above, find a matrix $\mathbf{A} \in \mathbb{R}^{n \times 3}$ and a vector $\mathbf{b} \in \mathbb{R}^n$ such that for all $\mathbf{c} \in \mathbb{R}^3$, we have

$$E(\mathbf{c}) = \|\mathbf{A}\mathbf{c} - \mathbf{b}\|_2^2. \tag{1}$$

(b) (4 pts) Implement a `C++/Eigen` function

```
Eigen::MatrixXd GetA(const Eigen::VectorXd &x);
```

that for given $\mathbf{x}$ as above returns the matrix $\mathbf{A}$ in (1).

(c) (4 pts) Implement a `C++/Eigen` function

```
Eigen::VectorXd LeastSquares(const Eigen::MatrixXd &A, const Eigen::VectorXd &b);
```

that for given $\mathbf{A}$ and $\mathbf{b}$ returns the least squares solution $\mathbf{c}$ of (1). You may use any `Eigen` solver of your choice.

(d) (8 pts) Let $k \in \mathbb{N}$ and $\mathbf{B} \in \mathbb{R}^{m \times n}$, where $m, n \geq k$. Implement a `C++/Eigen` function

```
Eigen::MatrixXd BestApprox(const Eigen::MatrixXd &B, int k);
```

that returns the $m \times n$ matrix that is the best rank-k approximation of $\mathbf{B}$, using the *singular value decomposition* of $\mathbf{B}$.

(e) (8 pts) Let $n \in \mathbb{N}$ and consider data points $(x_1, y_1), \ldots, (x_n, y_n) \in (0, \infty) \times (0, \infty)$. Let $\mathbf{B}_1 \in \mathbb{R}^{2 \times n}$ denote the best rank-1 approximation of

$$\mathbf{B} := \begin{pmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{pmatrix}. \tag{2}$$

Implement a `C++/Eigen` function

```
double FitLineThroughOrigin(const Eigen::MatrixXd &B);
```

that takes $\mathbf{B}$ as in (2) and computes $\mathbf{B}_1$ to extract the first principal component of $\mathbf{B}$. The line through the origin along this principal component will then fit the data points in $\mathbf{B}$. The function `FitLineThroughOrigin` shall return the slope of this line (see Figure 2).

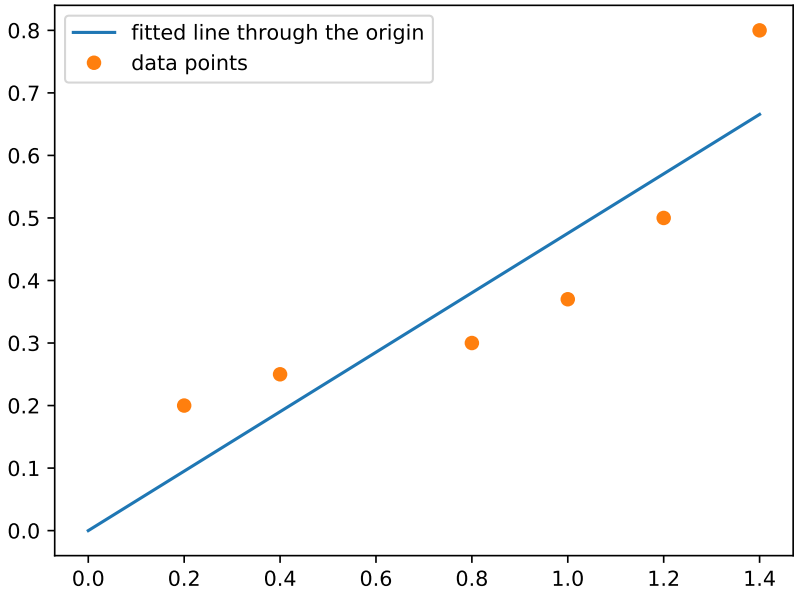

Figure 2: Line through the origin with slope computed by `FitLineThroughOrigin`.

Exercise 2. *Gauss-Chebyshev quadrature* (24 pts) [*Template: 2.cpp*]

The Chebyshev polynomial of the first kind $T_n(x)$ is a polynomial of exact degree n, defined by the relation

$$T_n(x) := \cos\left(n \arccos(x)\right), \quad \text{where } x \in [-1, 1]. \tag{3}$$

(a) (4 pts) The weighted inner product of two functions $f(x)$ and $g(x)$, with respect to a given continuous and non-negative weight function $w(x)$, can be defined as:

$$\langle f, g \rangle_w := \int_a^b w(x) f(x) g(x) \, \mathrm{d}x. \tag{4}$$

f and g are said to be orthogonal with respect to the weight function $w(x)$ if

$$\langle f, g \rangle_w = 0. \tag{5}$$

For the following interval and weight function:

$$[a, b] = [-1, 1] \,, \qquad w(x) = \frac{1}{\sqrt{1 - x^2}} \,,$$

show that the Chebyshev polynomials of the first kind satisfy

$$\langle T_i, T_j \rangle_w = 0 \quad \text{if } i \neq j. \tag{6}$$

Hint: Substitute $x = \cos\theta$ to simplify the computation of $\langle T_i, T_j \rangle_w$. The following trigonometric identity could also be useful:

$$2 \cos x \cos y = \cos(x + y) + \cos(x - y).$$

(b) (5 pts) Suppose that we now wish to calculate a definite integral of $f(x)$ with a general weight function $w(x)$, namely

$$I = \int_{-1}^{1} w(x) f(x) \, \mathrm{d}x \,. \tag{7}$$

I is to be approximated by an n-point quadrature formula of the form

$$Q_{n,w}(f) = \sum_{k=0}^{n-1} A_k f(x_k), \tag{8}$$

where A_k are the quadrature weights and x_k are the quadrature nodes in $[-1, 1]$.

Determine an *integral* expression for the quadrature weights A_k in terms of the quadrature nodes x_k and the general weight function $w(x)$, so that $Q_{n,w}(f)$ is guaranteed to have order $\geq n$.

Hint: In Eq.(7), substitute $f(x)$ with its polynomial Lagrange interpolant that is formed by interpolation through the quadrature nodes x_k.

(c) (3 pts) Show that

$$\int_{-1}^{1} w(x) T_n(x) q(x) \, \mathrm{d}x = 0 \tag{9}$$

for any polynomial function $q(x)$ of degree $n - 1$ or less, if

$$w(x) = \frac{1}{\sqrt{1 - x^2}}.$$

Hint: Use the orthogonality of Chebyshev polynomials as proved in (6).

(d) (5 pts) If the quadrature nodes x_k, $(k = 0, \ldots, n-1)$, are the known n zeros of the Chebyshev polynomial $T_n(x)$, and the quadrature weights A_k are those obtained in sub-problem (b), then $Q_{n,w}(f)$ corresponds to the Gauss-Chebyshev quadrature rule.

Derive the following statement: The Gauss-Chebyshev quadrature rule is of order $2n$.

Hint: Consider $f(x)$ to be a polynomial of degree $2n-1$ and then perform long polynomial division of $f(x)$ by $T_n(x)$, that is, write $f(x)$ as:

$$f(x) = T_n(x)q(x) + r(x),$$

and use the result derived in (9).

(e) (5 pts) For the Gauss-Chebyshev quadrature rule, it can be shown that the integral expression for the quadrature weights, as obtained in sub-problem (b), simplifies to

$$A_k = \frac{\pi}{n}, \quad \forall\, k = 0, \ldots, n-1. \tag{10}$$

Implement a `C++/Eigen` function

```
double Gauss_Chebyshev(const std::function<double(double)> &f, int n);
```

that performs the n-point Gauss-Chebyshev quadrature to approximate the integral

$$I = \int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}}\, \mathrm{d}x\ . \tag{11}$$

(f) (2 pts) Use the previous implementation to compute the quadrature error $E_{n,k}(f) := |I_k - Q_{n,w}(f)|$ for $n = 5,\ k = \{1, 2\}$, where

$$I_1 = \int_{-1}^{1} \frac{x^8}{\sqrt{1-x^2}}\, \mathrm{d}x\ ; \quad I_2 = \int_{-1}^{1} \frac{x^{10}}{\sqrt{1-x^2}}\, \mathrm{d}x\ .$$

Justify your results.

Note: The template '2.cpp' already contains the implementation for computing these errors.

Exercise 3. *Initial value problem* (26 pts) [*Template: 3.cpp*]

Consider the second order IVP

$$\ddot{y}(t) = 2y(t)\left(1 + y^2(t)\right), \quad y(0) = 0,\ \dot{y}(0) = 1, \tag{12}$$

where $t \in (-\frac{\pi}{2}, \frac{\pi}{2})$. We want to solve it using the *implicit Euler* scheme.

(a) (3 pts) Write down (on paper) a function $\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^2$ such that

$$\dot{\mathbf{z}}(t) = \mathbf{f}\left(\mathbf{z}(t)\right), \quad \mathbf{z}(0) = (0,1)^\top, \tag{13}$$

is equivalent to (12), where $\mathbf{z} : (-\frac{\pi}{2}, \frac{\pi}{2}) \to \mathbb{R}^2$ and $t \in (-\frac{\pi}{2}, \frac{\pi}{2})$. Moreover, implement a `C++/Eigen` function

```
Eigen::Vector2d f(const Eigen::Vector2d &x);
```

that takes $\mathbf{x} \in \mathbb{R}^2$ and returns the value $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^2$.

(b) (3 pts) The *implicit Euler* scheme with $N \in \mathbb{N}$ time steps of size $h > 0$ applied to (13) reads

$$\mathbf{z}^{(k+1)} = \mathbf{z}^{(k)} + h\mathbf{f}\left(\mathbf{z}^{(k+1)}\right), \quad \mathbf{z}^{(0)} = (0,1)^\top, \tag{14}$$

for all $k \in \{0, \dots, N-1\}$. To find $\mathbf{z}^{(k+1)}$ for given $\mathbf{z}^{(k)}$, we have to solve a *non-linear* equation

$$\mathbf{F}(\mathbf{x}) = 0, \tag{15}$$

where $\mathbf{x} \in \mathbb{R}^2$ and

$$\mathbf{F}(\mathbf{x}) = \mathbf{z}^{(k)} + h\mathbf{f}\left(\mathbf{x}\right) - \mathbf{x}. \tag{16}$$

Write down (on paper) the *Jacobian* $\mathrm{D}\,\mathbf{F}(\mathbf{x})$. Moreover, implement a `C++/Eigen` function

```
Eigen::Matrix2d DF(const Eigen::Vector2d &x, double h);
```

that takes a point $\mathbf{x} \in \mathbb{R}^2$ and the step size $h > 0$ and returns the value $\mathrm{D}\,\mathbf{F}(\mathbf{x})$.

(c) (6 pts) Implement a `C++/Eigen` function

```
Eigen::Vector2d Newton(Eigen::Vector2d x, double h, int n, double tol);
```

that returns the result of *Newton's* method applied to (15). The starting value is $\mathbf{x} \in \mathbb{R}^2$ and $h > 0$ is the step size in (16). Stop after $n \in \mathbb{N}$ iterations or when the *residual based* stopping criterion with tolerance `tol` is met.

(d) (8 pts) Implement a `C++/Eigen` function

```
Eigen::Vector2d QuasiNewton(Eigen::Vector2d x, double h, int n, double tol);
```

that performs the same task as `Newton`, but using the *Quasi-Newton* method via the *Sherman-Morrison-Woodbury* formula.

(e) (6 pts) Implement a `C++/Eigen` function

```
Eigen::Vector2d ImplicitEuler(Eigen::Vector2d z0, double h, int N);
```

that performs the *implicit Euler* scheme (14) to solve the IVP (13) with initial value `z0`, applying $N \in \mathbb{N}$ time steps of size $h > 0$. Equation (15) should be solved by `Newton` or `QuasiNewton` with $n = 10$ iterations and tolerance `tol = 1.0e-8`. The return value of `ImplicitEuler` is the approximate solution at time $N \cdot h$.