First name		Grade
Last name		
Department		
Computer Name		
Legi No.		
Date	12.08.2016	

1	2	3	4	5	Total

- Fill in this cover sheet first.
- Always keep your Legi visible on the table.
- Keep your phones, tablets and computers turned off in your bag.
- Start each handwritten problem on a new sheet.
- Put your name on each sheet.
- Do not write with red/green/pencil.
- Write your solutions clearly and work carefully.
- **Write all your solutions only in the folder `results`!**
- Any other location will not be backed-up and will be discarded.
- Files in `resources` may be overridden at any time.
- Make sure to regularly save your solutions.
- Time spent on restroom breaks is considered examination time.
- **Never turn off or log off from your computer!**

Final exam
Numerical Methods for CSE

R. Hiptmair, G. Alberti, F. Leonardi

August 12, 2016

A "`CMake`" file is provided with the templates. To generate a "`Makefile`" for all problems, type "`cmake .`" in the folder "`~/results`". Compile your programs with "`make`".

In order to compile and run the C++ code related to a single problem, e.g. Problem 3, type "`make problem3`". Execute the program using "`./problem3`".

If you want to manually compile your code, use:

```
g++ -I/usr/include/eigen3 -std=c++11
    -Wno-deprecated-declarations -Wno-ignored-attributes
    filename.cpp -o programname
```

or

```
clang++ -I/usr/include/eigen3 -std=c++11
    -Wno-deprecated-declarations -Wno-ignored-attributes
    filename.cpp -o programname
```

The flags `-Wno-deprecated-declarations -Wno-ignored-attributes` suppress some unwanted EIGEN warning.

For each problem requiring C++ implementation, a template file named `problemX.cpp` is provided. For your own convenience, there is a marker `TODO` in the places where you are supposed to write your own code. All templates should compile even if left unchanged.

Problem 1 Symmetric Gauss-Seidel iteration (20 pts.)

For a square matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, define $\mathbf{D_A}, \mathbf{L_A}, \mathbf{U_A} \in \mathbb{R}^{n,n}$ by:

$$(\mathbf{D_A})_{i,j} := \begin{cases} (\mathbf{A})_{i,j}, & i = j, \\ 0, & i \neq j \end{cases}, (\mathbf{L_A})_{i,j} := \begin{cases} (\mathbf{A})_{i,j}, & i > j, \\ 0, & i \leq j \end{cases}, (\mathbf{U_A})_{i,j} := \begin{cases} (\mathbf{A})_{i,j}, & i < j, \\ 0, & i \geq j \end{cases}. \tag{1}$$

The symmetric Gauss-Seidel iteration associated with the linear system of equations $\mathbf{Ax} = \mathbf{b}$ is defined as

$$\mathbf{x}^{(k+1)} = (\mathbf{U_A} + \mathbf{D_A})^{-1}\mathbf{b} - (\mathbf{U_A} + \mathbf{D_A})^{-1}\mathbf{L_A}(\mathbf{L_A} + \mathbf{D_A})^{-1}(\mathbf{b} - \mathbf{U_A}\mathbf{x}^{(k)}), \tag{2}$$

where $\mathbf{x}^{(k)}, \mathbf{x}, \mathbf{b} \in \mathbb{R}^n$ ($k \geq 0$) and $x^{(0)}$ is a given initial guess.

(1a) Give a necessary and sufficient condition on $\mathbf{A}$ such that the iteration (2) is well-defined.

(1b) Assume that (2) is well-defined. Show that a fixed point $\mathbf{x} \in \mathbb{R}^n$ of the iteration (2) is a solution of the linear system of equations $\mathbf{Ax} = \mathbf{b}$.

(1c) Implement a C++ function (in `problem1.cpp`)

```
using Matrix = Eigen::MatrixXd;
using Vector = Eigen::VectorXd;

void GSIt(const Matrix & A, const Vector & b, Vector & x,
   double rtol);
```

solving the linear system $\mathbf{Ax} = \mathbf{b}$ using the iterative scheme (2). To that end, apply the iterative scheme to an initial guess $\mathbf{x^{(0)}}$ provided in `x`. The approximated solution given by the final iterate is then stored in `x` as an output.

Use a correction based termination criterion with relative tolerance `rtol` using the Euclidean vector norm.

(1d) Test your implementation (use `n = 9` and `rtol = 10e-8`) with the linear system given by

$$\mathbf{A} = \begin{bmatrix} 3 & 1 & 0 & \dots & 0 \\ 2 & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \dots & 0 & 2 & 3 \end{bmatrix} \in \mathbb{R}^{n,n}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^n. \tag{3}$$

Output the l^2-norm of the *residual* of the approximated solution. Use $\mathbf{b}$ as your initial guess.

(1e) Using the same matrix $\mathbf{A}$ and the same r.h.s. vector $\mathbf{b}$ as above (1d), in Table 1 we have tabulated the quantity $\|\mathbf{x}^{(k)} - \mathbf{A}^{-1}\mathbf{b}\|_2$ for $k = 1, \ldots, 20$.

k	$\|\mathbf{x}^{(k)} - \mathbf{A}^{-1}\mathbf{b}\|_2$	k	$\|\mathbf{x}^{(k)} - \mathbf{A}^{-1}\mathbf{b}\|_2$
1	0.172631	11	0.00341563
2	0.0623049	12	0.00234255
3	0.0464605	13	0.00160173
4	0.0360226	14	0.00109288
5	0.0272568	15	0.000744595
6	0.0200715	16	0.000506784
7	0.0144525	17	0.000344682
8	0.0102313	18	0.000234316
9	0.00715417	19	0.000159234
10	0.00495865	20	0.000108185

Table 1: Euclidean norms of error vectors for 20 iterates

Describe qualitatively and quantitatively the convergence of the iterative scheme with respect to the number of iterations k.

Problem 2 Efficient sparse solver (24 pts.)

Fix $n \in \mathbb{N}$, $n \geq 2$, and let $1 \leq i_0, j_0 \leq n, i_0 > j_0$ and $c_i \in \mathbb{R}$ for $i = 1, \ldots, n-1$. We consider a $n \times n$ linear system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{n,n}$ is given by

$$(\mathbf{A})_{i,j} = \begin{cases} 1 & \text{if } i = j, \\ c_i & \text{if } i = j - 1, \\ 1 & \text{if } i = i_0,\ j = j_0, \\ 0 & \text{otherwise,} \end{cases}$$

for $1 \leq i, j \leq n$.

(2a) Efficiently construct the matrix $\mathbf{A}$ defined above as an EIGEN matrix of type `Eigen::SparseMatrix<double>` using an intermediate triplet format. To that end, implement a function

```
using Vector = Eigen::VectorXd;
```

```
using Matrix = Eigen::SparseMatrix<double>;

Matrix buildA(const Vector & c, unsigned int i0, unsigned int
    j0);
```

whose, given the coefficients c_i in `c` and the indices i_0 and j_0, returns the sparse matrix `A`.

(2b) Implement a function

```
Vector solveLSE(const Vector & c, const Vector & b,
          unsigned int i0, unsigned int j0);
```

that computes the solution $\mathbf{x} \in \mathbb{R}^n$ of the system $\mathbf{A}\mathbf{x} = \mathbf{b}$ with optimal asymptotic complexity $O(n)$.

HINT: If $(\mathbf{A})_{i_0,j_0}$ were zero, $\mathbf{A}$ would be a banded upper triangular matrix and the task would be easy. Regard $\mathbf{A}$ as a small modification of such a matrix.

Problem 3 Not-a-knot cubic spline interpolation (24 pts.)

We are given an interval $[a, b] \subset \mathbb{R}$ and a knot set

$$\mathcal{M} = \{a = t_0 < t_1 < \cdots < t_{n-1} < t_n = b\}$$

for some $n \in \mathbb{N} \setminus \{0\}$. Let $\mathcal{S}_{3,\mathcal{M}}$ denote the space of cubic spline functions on $\mathcal{M}$, and define

$$\tilde{\mathcal{S}}_{3,\mathcal{M}} = \{s \in \mathcal{S}_{3,\mathcal{M}} : s''' \text{ is continuous at } t_1 \text{ and at } t_{n-1}\}.$$

(3a) Derive a linear system of equations whose solution yields the slopes $\tilde{s}'(t_j)$, $j = 0, \ldots, n$, of $\tilde{s} \in \tilde{\mathcal{S}}_{3,\mathcal{M}}$ satisfying the interpolation conditions $\tilde{s}(t_j) = y_j$ for given $y_j \in \mathbb{R}$, $j = 0, \ldots, n$.

(3b) The provided C++ function (in `problem3.cpp`)

```
using Vector = Eigen::VectorXd;

void natsplineslopes (const Vector &t, const Vector &y, Vector
    &c)
```

computes the slopes $c_j := s'(t_j)$, $j = 0, \ldots, n$, of the *natural* cubic spline interpolant $s \in \mathcal{S}_{3,\mathcal{M}}$ through the data points (t_j, y_j), $j = 0, \ldots, n$.

Based on `natsplineslopes`, implement a corresponding function (in `problem3.cpp`)

```
void notknotslopes (const Vector &t, const Vector &y, Vector
  &c)
```

that computes the slopes $c_j := \tilde{s}'(t_j)$, $j = 0, \ldots, n$, of $\tilde{s} \in \tilde{\mathcal{S}}_{3,\mathcal{M}}$ satisfying $\tilde{s}(t_j) = y_j$ for given $y_j \in \mathbb{R}$, $j = 0, \ldots, n$.

Problem 4 On the Gauss points (24 pts.)

Given $f \in C^0([0,1])$, the integral

$$g(x) := \int_0^1 e^{|x-y|} f(y)\, dy$$

defines a function $g \in C^0([0,1])$. Throughout this problem, we approximate the integral by means of n-point Gauss quadrature, which yields a function $g_n(x)$.

(4a) Let $\{\xi_j^n\}_{j=1}^n$ be the nodes for the Gauss quadrature on the interval $[0,1]$ and write w_j^n for the corresponding weights. Assume that the nodes are ordered, namely $\xi_j^n < \xi_{j+1}^n$ for every $j = 1, \ldots, n-1$. We can write

$$(g_n(\xi_l^n))_{l=1}^n = \mathbf{M}\ (f(\xi_j^n))_{j=1}^n,$$

for a suitable matrix $\mathbf{M} \in \mathbb{R}^{n,n}$. Give a formula for the entries of $\mathbf{M}$.

(4b) Implement a function (in `problem4.cpp`)

```
using Vector = Eigen::VectorXd;

template<typename Function>
Vector comp_g_gausspts(Function f, unsigned int n)
```

that computes the n-vector $(g_n(\xi_l^n))_{l=1}^n$ with optimal complexity $O(n)$ (excluding the computation of the Gauss nodes and weights), where `f` is an object with an evaluation operator `double operator()(double x)`, e.g. a lambda function, that represents the function f.

HINT: You may use the provided function (in `problem4.cpp`)

```
void gaussrule(int n, Vector & w, Vector & xi)
```

that computes the weights `w` and the ordered nodes `xi` relative to the `n`-point Gauss quadrature on the interval $[-1,1]$.

(4c) Test your implementation by computing $g(\xi_{11}^{21})$ ($\xi_{11}^{21} = 1/2$) for $f(y) = e^{-|0.5-y|}$. What result do you expect?

Problem 5 Construction of an evolution operator (28 pts.)

Let Ψ^h define the discrete evolution of an order p Runge-Kutta single step method for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}), \mathbf{f} : D \subseteq \mathbb{R}^d \to \mathbb{R}^d$. We define a new evolution operator:

$$\tilde{\Psi}^h := \frac{1}{1-2^p}\left(\Psi^h - 2^p \cdot (\Psi^{h/2} \circ \Psi^{h/2})\right), \tag{4}$$

where $\circ$ denotes the composition of mappings.

(5a) Let Ψ^h be the evolution operator of the explicit Euler method. Give the explicit formula for $\tilde{\Psi}^h$.

(5b) Implement a C++ function (in `problem5.cpp`)

```
using Vector = Eigen::VectorXd;

template <class Operator>
Vector psitilde(const Operator &Psi, unsigned int p,
          double h, const Vector &y0);
```

that returns $\tilde{\Psi}^h \mathbf{y}_0$ when given the underlying Ψ.

Objects of type `Operator` (here and in the following) must provide an evaluation operator with signature:

```
Vector operator()(double h, const Vector &y);
```

providing the evaluation of $\Psi^h(\mathbf{y})$. A suitable C++ lambda function satisfies this requirement.

(5c) Implement a C++ function (in `problem5.cpp`)

```
template <class Operator>
std::vector<Vector> odeintequi(const Operator &Psi,
     double T, const Vector &y0, unsigned int N);
```

for the approximation of the solution of the initial value problem $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}), \mathbf{y}(0) = \mathbf{y}_0$. The function uses a single step method defined by the discrete evolution operator Ψ (given as `Psi`) on N equidistant timesteps with final time $T > 0$. The function returns the approximated value at each step (including y_0: $y_0, y_1, \ldots$) in a `std::vector<Vector>`.

HINT: You can use `std::vector<Vector>::push_back` to add elements to the end of a `std::vector<Vector>`.

(5d) In the case of the IVP

$$\dot{y} = 1 + y^2, \quad y(0) = 0, \tag{5}$$

with exact solution $y_{ex}(t) = \tan(t)$, determine empirically (using `odeintequi`) the order of the single step method induced by $\tilde{\Psi}^h$, when Ψ is the discrete evolution operator for the explicit Euler method. Monitor the error $|y_N(1) - y_{ex}(1)|$ at final time $T = 1$, where y_N is the solution approximated through $\tilde{\Psi}^h$ using N uniforms steps with $N = 2^q, q = 2, \ldots, 12$.

(5e) In general, the method defined by $\tilde{\Psi}^h$ has order $p + 1$. Thus, it can be used for adaptive timestep control and prediction.

Complete the implementation of a function (found in `problem5.cpp`)

```
template <class Operator>
std::vector<Vector> odeintssctrl(const Operator &Psi,
                        double T, const Vector &y0,
                        double h0, unsigned int p,
                        double reltol, double abstol,
                        double hmin);
```

for the approximation of the solution of the IVP by means of adaptive timestepping based on Ψ and $\tilde{\Psi}$, where Ψ is passed through the argument `Psi`. Step rejection and stepsize correction and prediction has to be employed. The argument `T` supplies the final time, `y0` the initial state, `h0` an initial stepsize, `p` the order to the discrete evolution Ψ, `reltol` and `abstol` the respective tolerances, and `hmin` a minimal stepsize that will trigger premature termination. Compute the solution obtained using this function applied to the IVP (5) up to time $T = 1$ with the following data: `h0 = 1/100`, `reltol = 10e-8`, `abstol = 10e-8`, `hmin = 10e-6`. Output the approximated solution at time $T = 1$.