Prof. R. Hiptmair
G. Alberti,
F. Leonardi

AS 2015

Numerical Methods for CSE

ETH Zürich
D-MATH

Problem Sheet 2

You should try to your best to do the core problems. If time permits, please try to do the rest as well.

Problem 1. Lyapunov Equation (core problem)

Any linear system of equations with a finite number of unknowns can be written in the "canonical form" $\mathbf{A}\mathbf{x} = \mathbf{b}$ with a system matrix $\mathbf{A}$ and a right hand side vector $\mathbf{b}$. However, the LSE may be given in a different form and it may not be obvious how to extract the system matrix. This task gives an intriguing example and also presents an important *matrix equation*, the so-called Lyapunov Equation.

Given $\mathbf{A} \in \mathbb{R}^{n\times n}$, consider the equation

$$\mathbf{A}\mathbf{X} + \mathbf{X}\mathbf{A}^T = \mathbf{I} \tag{5}$$

with unknown $\mathbf{X} \in \mathbb{R}^{n\times n}$.

(1a) ⊡ Show that for a fixed matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ the mapping

$$L : \begin{cases} \mathbb{R}^{n,n} & \to & \mathbb{R}^{n,n} \\ \mathbf{X} & \mapsto & \mathbf{A}\mathbf{X} + \mathbf{X}\mathbf{A}^T \end{cases}$$

is linear.

HINT: Recall from linear algebra the definition of a linear mapping between two vector spaces.

Solution: Take $\alpha, \beta \in \mathbb{R}$ and $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{n,n}$. We readily compute

$$\begin{aligned} L(\alpha\mathbf{X} + \beta\mathbf{Y}) &= A(\alpha\mathbf{X} + \beta\mathbf{Y}) + (\alpha\mathbf{X} + \beta\mathbf{Y})A^T \\ &= \alpha A\mathbf{X} + \beta A\mathbf{Y} + \alpha\mathbf{X}A^T + \beta\mathbf{Y}A^T \\ &= \alpha(A\mathbf{X} + \mathbf{X}A^T) + \beta(A\mathbf{Y} + \mathbf{Y}A^T) \\ &= \alpha L(\mathbf{X}) + \beta L(\mathbf{Y}), \end{aligned}$$

as desired.

In the sequel let $\text{vec}(\mathbf{M}) \in \mathbb{R}^{n^2}$ denote the column vector obtained by reinterpreting the internal coefficient array of a matrix $M \in \mathbb{R}^{n,n}$ stored in column major format as the data array of a vector with n^2 components. In MATLAB, $\text{vec}(\mathbf{M})$ would be the column vector obtained by `reshape(M,n*n,1)` or by `M(:)`. See [1, Rem. 1.2.18] for the implementation with Eigen.

Problem (5) is equivalent to a linear system of equations

$$\mathbf{C}\text{vec}(\mathbf{X}) = \mathbf{b} \tag{6}$$

with system matrix $\mathbf{C} \in \mathbb{R}^{n^2,n^2}$ and right hand side vector $\mathbf{b} \in \mathbb{R}^{n^2}$.

(1b) ⊡ Refresh yourself on the notion of "sparse matrix", see [1, Section 1.7] and, in particular, [1, Notion 1.7.1], [1, Def. 1.7.3].

(1c) ⊡ Determine $\mathbf{C}$ and $\mathbf{b}$ from (6) for $n = 2$ and

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ -1 & 3 \end{bmatrix}.$$

Solution: Write $X = \left[\begin{smallmatrix} x_1 & x_3 \\ x_2 & x_4 \end{smallmatrix}\right]$, so that $\text{vec}(\mathbf{X}) = (x_i)_i$. A direct calculation shows that (5) is equivalent to (6) with

$$\mathbf{C} = \begin{bmatrix} 4 & 1 & 1 & 0 \\ -1 & 5 & 0 & 1 \\ -1 & 0 & 5 & 1 \\ 0 & -1 & -1 & 6 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

(1d) ⚁ Use the Kronecker product to find a general expression for $\mathbf{C}$ in terms of a general $\mathbf{A}$.

Solution: We have $\mathbf{C} = \mathbf{I} \otimes \mathbf{A} + \mathbf{A} \otimes \mathbf{I}$. The first term is related to $\mathbf{AX}$, the second to $\mathbf{XA}^T$.

(1e) ⚁ Write a MATLAB function

```
function C = buildC (A)
```

that returns the matrix $\mathbf{C}$ from (6) when given a square matrix $\mathbf{A}$. (The function `kron` may be used.)

Solution: See Listing 24.

Listing 24: Building the matrix $\mathbf{C}$ in (6) with MATLAB

```
% Create the matrix C

function C = buildC(A)

n = size(A);
I = eye(n);
C = kron(A,I) + kron(I,A);
```

(1f) Give an upper bound (as sharp as possible) for $\mathrm{nnz}(\mathbf{C})$ in terms of $\mathrm{nnz}(\mathbf{A})$. Can $\mathbf{C}$ be legitimately regarded as a sparse matrix for large n even if $\mathbf{A}$ is dense?

HINT: Run the following MATLAB code:

```
n=4;
A=sym('A',[n,n]);
I=eye(n);
C=buildC(A)
```

Solution: Note that, for general matrices $\mathbf{A}$ and $\mathbf{B}$ we have $\mathrm{nnz}(\mathbf{A}\otimes\mathbf{B}) = \mathrm{nnz}(\mathbf{A})\mathrm{nnz}(\mathbf{B})$. This follows from the fact that the block in position (i,j) of the matrix $\mathbf{A}\otimes\mathbf{B}$ is $a_{ij}\mathbf{B}$. In our case, we immediately obtain

$$\mathrm{nnz}(\mathbf{C}) = \mathrm{nnz}(\mathbf{I}\otimes\mathbf{A}+\mathbf{A}\otimes\mathbf{I}) \leq \mathrm{nnz}(\mathbf{I}\otimes\mathbf{A}) + \mathrm{nnz}(\mathbf{A}\otimes\mathbf{I}) \leq 2\mathrm{nnz}(\mathbf{I})\mathrm{nnz}(\mathbf{A}),$$

namely

$$\mathrm{nnz}(\mathbf{C}) \leq 2n\mathrm{nnz}(\mathbf{A}).$$

The optimality of this bound can be checked by taking the matrix $\mathbf{A} = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$.

This bound says that, in general, even if $\mathbf{A}$ is not sparse, we have $\mathrm{nnz}(\mathbf{A}) \leq 2n^3 \ll n^4$. Therefore, $\mathbf{C}$ can be regarded as a sparse matrix for any $\mathbf{A}$.

(1g) Implement a C++ function

```
Eigen::SparseMatrix<double> buildC(const MatrixXd &A)
```

that builds the Eigen matrix $\mathbf{C}$ from $\mathbf{A}$. Make sure that initialization is done efficiently using an intermediate triplet format. Read [1, Section 1.7.3] very carefully before starting.

Solution: See `solveLyapunov.cpp`.

(1h) ⚀ Validate the correctness of your C++ implementation of `buildC` by comparing with the equivalent Matlab function for $n = 5$ and

$$A = \begin{bmatrix} 10 & 2 & 3 & 4 & 5 \\ 6 & 20 & 8 & 9 & 1 \\ 1 & 2 & 30 & 4 & 5 \\ 6 & 7 & 8 & 20 & 0 \\ 1 & 2 & 3 & 4 & 10 \end{bmatrix}.$$

Solution: See `solveLyapunov.cpp` and `solveLyapunov.m`.

(1i) ⚁ Write a C++ function

```
void solveLyapunov(const MatrixXd & A, MatrixXd & X)
```

that returns the solution of (5) in the $n \times n$-matrix $\mathbf{X}$, if $A \in \mathbb{R}^{n,n}$.

Solution: See `solveLyapunov.cpp`.

Remark. Not every invertible matrix $\mathbf{A}$ allows a solution: if $\mathbf{A}$ and $-\mathbf{A}$ have a common eigenvalue the system $\mathbf{C}\mathbf{x} = \mathbf{b}$ is singular, try it with the matrix $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. For a more efficient solution of the task, see Chapter 15 of Higham's book.

(1j) ⚁ Test your C++ implementation of `solveLyapunov` by comparing with Matlab for the test case proposed in (1h).

Solution: See `solveLyapunov.cpp` and `solveLyapunov.m`.

Problem 2. Partitioned Matrix (core problem)

Based on the block view of matrix multiplication presented in [1, § 1.3.13], we looked a *block elimination* for the solution of block partitioned linear systems of equations in [1, § 1.6.92]. Also of interest are [1, Rem. 1.6.46] and [1, Rem. 1.6.44] where LU-factorization is viewed from a block perspective. Closely related to this problem is [1, Ex. 1.6.95], which you should study again as warm-up to this problem.

Let the matrix $\mathbf{A} \in \mathbb{R}^{n+1,n+1}$ be partitioned according to

$$\mathbf{A} = \begin{bmatrix} \mathbf{R} & \mathbf{v} \\ \mathbf{u}^T & 0 \end{bmatrix}, \tag{7}$$

where $\mathbf{v} \in \mathbb{R}^n$, $\mathbf{u} \in \mathbb{R}^n$, and $\mathbf{R} \in \mathbb{R}^{n \times n}$ is upper triangular and regular.

(2a) ⚀ Give a necessary and sufficient condition for the triangular matrix $\mathbf{R}$ to be invertible.

Solution: $\mathbf{R}$ being upper triangular $\det(\mathbf{R}) = \prod_{i=0}^{n}(\mathbf{R})_{i,i}$, means that all the diagonal elements must be non-zero for $\mathbf{R}$ to be invertible.

(2b) ⚁ Determine expressions for the subvectors $\mathbf{z} \in \mathbb{R}^n, \xi \in \mathbb{R}$ of the solution vector of the linear system of equations

$$\begin{bmatrix} \mathbf{R} & \mathbf{v} \\ \mathbf{u}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{z} \\ \xi \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \beta \end{bmatrix}$$

for arbitrary $\mathbf{b} \in \mathbb{R}^n, \beta \in \mathbb{R}$.

HINT: Use blockwise Gaussian elimination as presented in [1, § 1.6.92].

Solution: Applying the computation in [1, Rem. 1.6.30], we obtain:

$$\begin{bmatrix} \mathbf{1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{z} \\ \xi \end{bmatrix} = \begin{bmatrix} \mathbf{R}^{-1}(\mathbf{b} - \mathbf{v}s^{-1}b_s) \\ s^{-1}b_s \end{bmatrix}$$

with $s := -(\mathbf{u}^\top \mathbf{R}^{-1}\mathbf{v}), b_s := (\beta - \mathbf{u}^\top \mathbf{R}^{-1}\mathbf{b})$.

(2c) ⚁ Show that $\mathbf{A}$ is regular if and only if $\mathbf{u}^T\mathbf{R}^{-1}\mathbf{v} \neq 0$.

Solution: The square matrix $\mathbf{A}$ is regular, if the corresponding linear system has a solution for every right hand side vector. If $\mathbf{u}^T\mathbf{R}^{-1}\mathbf{v} \neq 0$ the expressions derived in the previous sub-problem show that a solution can be found for any $\mathbf{b}$ and β, because $\mathbf{R}$ is already known to be invertible.

(2d) ⚂ Implement the C++ function

```
template <class Matrix, class Vector>
void solvelse(const Matrix & R, const Vector & v, const
  Vector & u, const Vector & b, Vector & x);
```

for computing the solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ (with $\mathbf{A}$ as in (7)) efficiently. Perform size check on input matrices and vectors.

HINT: Use the decomposition from (2b).

HINT: you can rely on the `triangularView()` function to instruct EIGEN of the triangular structure of $\mathbf{R}$, see [1, Code 1.2.12].

HINT: using the construct:

```
typedef typename Matrix::Scalar Scalar;
```

you can obtain the scalar type of the `Matrix` type (e.g. `double` for `MatrixXd`). This can then be used as:

```
Scalar a = 5;
```

HINT: using `triangularView` and templates you may incur in weird compiling errors. If this happens to you, check http://eigen.tuxfamily.org/dox/TopicTemplateKeyword.html

HINT: sometimes the C++ keyword `auto` (only in std. C++11) can be used if you do not want to explicitly write the return type of a function, as in:

```
MatrixXd a;
auto b = 5*a;
```

Solution: See `block_lu_decomp.cpp`.

(2e) ⚁ Test your implementation by comparing with a standard LU-solver provided by EIGEN.

HINT: Check the page http://eigen.tuxfamily.org/dox/group__TutorialLinearAlgebra.html.

Solution: See `block_lu_decomp.cpp`.

(2f) ⚀ What is the asymptotic complexity of your implementation of `solvelse()` in terms of problem size parameter $n \to \infty$?

Solution: The complexity is $O(n^2)$. The backward substitution for $\mathbf{R}^{-1}\mathbf{x}$ is $O(n^2)$, vector dot product and subtraction is $O(n)$, so that the complexity is dominated by the backward substitution $O(n^2)$.

Problem 3. Banded matrix

For $n \in \mathbb{N}$ we consider the matrix

$$\mathbf{A} := \begin{bmatrix} 2 & a_1 & 0 & \dots & \dots & \dots & 0 \\ 0 & 2 & a_2 & 0 & \dots & \dots & 0 \\ b_1 & 0 & \ddots & \ddots & \ddots & & \vdots \\ 0 & b_2 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & a_{n-1} \\ 0 & 0 & \dots & 0 & b_{n-2} & 0 & 2 \end{bmatrix} \in \mathbb{R}^{n,n}$$

with $a_i, b_i \in \mathbb{R}$.

Remark. The matrix $\mathbf{A}$ is an instance of a banded matrix, see [1, Section 1.7.6] and, in particular, the examples after [1, Def. 1.7.53]. However, you need not know any of the content of this section for solving this problem.

(3a) ⚁ Implement an *efficient* C++ function:

```
template <class Vector>
void multAx(const Vector & a, const Vector & b, const
   Vector & x, Vector & y);
```

for the computation of $\mathbf{y} = \mathbf{A}\mathbf{x}$.

Solution: See `banded_matrix.cpp`.

(3b) ⚃ Show that $\mathbf{A}$ is invertible if $a_i, b_i \in [0, 1]$.

HINT: Give an indirect proof that $\ker \mathbf{A}$ is trivial, by looking at the largest (in modulus) component of an $\mathbf{x} \in \ker \mathbf{A}$.

Remark. That $\mathbf{A}$ is invertible can immediately be concluded from the general fact that kernel vectors of irreducible, diagonally dominant matrices (→ [1, Def. 1.8.8]) must be multiples of $[1, 1, \dots, 1]^\top$. Actually, the proof recommended in the hint shows this fact first before bumping into a contradiction.

Solution: Assume by contradiction that $\ker \mathbf{A} \neq \{0\}$. Pick $0 \neq \mathbf{x} \in \ker \mathbf{A}$ and consider $i = \operatorname{argmax}|x_j|, x_i \neq 0$. Since $2x_i + a_i x_{i+1} + b_{i-2}x_{i-2} = 0 \Rightarrow 2 \leq |\frac{x_{i+1}}{x_i} a_i + \frac{x_{i-2}}{x_i} b_{i-2}| <$

$a_i + b_{i-2} \le 2$, unless $\mathbf{x} = const.$ (in which case $\mathbf{A}\mathbf{x} \neq 0$, as we see from the first equation). By contradiction $\ker \mathbf{A} = \{0\}$.

(3c) ⚁ Fix $b_i = 0, \forall i = 1, \dots, n-2$. Implement an efficient C++ function

```
template <class Vector>
void solvelseAupper(const Vector & a, const Vector &
    r, Vector & x);
```

solving $\mathbf{A}\mathbf{x} = \mathbf{r}$.

Solution: See `banded_matrix.cpp`.

(3d) ⚁ For general $a_i, b_i \in [0,1]$ devise an efficient C++ function:

```
template <class Vector>
void solvelseA(const Vector & a, const Vector & b,
    const Vector & r, Vector & x);
```

that computes the solution of $\mathbf{A}\mathbf{x} = \mathbf{r}$ by means of Gaussian elimination. You cannot use any high level solver routines of EIGEN.

HINT: Thanks to the constraint $a_i, b_i \in [0,1]$, pivoting is not required in order to ensure stability of Gaussian elimination. This is asserted in [1, Lemma 1.8.9], but you may just use this fact here. Thus, you can perform a straightforward Gaussian elimination from top to bottom as you have learned it in your linear algebra course.

Solution: See `banded_matrix.cpp`.

(3e) ⚀ What is the asymptotic complexity of your implementation of `solvelseA` for $n \to \infty$.

Solution: To build the matrix we need at most $O(3n)$ insertions (3 per row). For the elimination stage we use three for loops, one of size n and two of size, at most, 3 (exploiting the banded structure of A), thus $O(9n)$ operations. For backward substitution we use two loops, one of size n and the other of size, at most, 3, for a total complexity of $O(3n)$. Therefore, the total complexity is $O(n)$.

(3f) ⚁ Implement `solvelseAEigen` as in (3d), this time using EIGEN's sparse elimination solver.

HINT: The standard way of initializing a sparse EIGEN-matrix efficiently, is via the triplet format as discussed in [1, Section 1.7.3]. You may also use direct initialization of a sparse matrix, provided that you `reserve()` enough space for the non-zero entries of each column, see documentation.

Solution: See `banded_matrix.cpp`.

Problem 4. Sequential linear systems

This problem is about a sequence of linear systems, please see [1, Rem. 1.6.86]. The idea is that if we solve several linear systems with the same matrix $\mathbf{A}$, the computational cost may be reduced by performing the LU decomposition only once.

Consider the following MATLAB function with input data $\mathbf{A} \in \mathbb{R}^{n,n}$ and $\mathbf{b} \in \mathbb{R}^n$.

```
function X = solvepermb(A,b)
[n,m] = size(A);
if ((n ≠ numel(b)) || (m ≠ numel(b))), error('Size mismatch'); end
X = [];
for l=1:n
    X = [X,A\b];
    b = [b(end);b(1:end-1)];
end
```

(4a) What is the asymptotic complexity of this function as $n \to \infty$?

Solution: The code consists of n solutions of a linear system, and so the asymptotic complexity is $O(n^4)$.

(4b) Port the MATLAB function `solvepermb` to C++ using EIGEN. (This means that the C++ code should perform exactly the same computations in exactly the same order.)

Solution: See file `solvepermb.cpp`.

(4c) Design an efficient implementation of this function with asymptotic complexity $O(n^3)$ in Eigen.

Solution: See file `solvepermb.cpp`.

Issue date: 24.09.2015
Hand-in: 01.10.2015 (in the boxes in front of HG G 53/54).

Version compiled on: October 5, 2015 (v. 1.0).