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

AS 2015

ETH Zürich
D-MATH

Numerical Methods for CSE

Problem Sheet 3

Problem 1   Rank-one perturbations (core problem)

This problem is another application of the Sherman-Morrison-Woodbury formula, see [1, Lemma 1.6.113]: please revise [1, § 1.6.104] of the lecture carefully.

Consider the MATLAB code given in Listing 25.

Listing 25: Matlab function `rankoneinvit`

```matlab
function lmin = rankoneinvit(d,tol)
if (nargin < 2), tol = 1E-6; end
ev = d;
lmin = 0.0;
lnew = min(abs(d));

while (abs(lnew-lmin) > tol*lmin)
    lmin = lnew;
    M = diag(d) + ev*ev';
    ev = M\ev;
    ev = ev/norm(ev);
    lnew = ev'*M*ev;
end
lmin = lnew;
```

(1a) ⌑ Write an equivalent implementation in EIGEN of the Matlab function `rankoneinvit`. The C++ code should use exactly the same operations.

Solution: See file `rankoneinvit.cpp`. Do not expect top understand what is the purpose of the function.

1

(1b) ⊡ What is the asymptotic complexity of the loop body of the function `rankoneinvit`? More precisely, you should look at the asymptotic complexity of the code in the lines 8-12 of Listing 25.

Solution: The total asymptotic complexity is dominated by the solution of the linear system with matrix M done in line 10, which has asymptotic complexity of $O(n^3)$.

(1c) ⊡ Write an efficient implementation in EIGEN of the loop body, possibly with optimal asymptotic complexity. Validate it by comparing the result with the other implementation in EIGEN.

HINT: Take the clue from [1, Code 1.6.114].

Solution: See file `rankoneinvit.cpp`.

(1d) ⊡ What is the asymptotic complexity of the new version of the loop body?

Solution: The loop body of the C++ function `rankoneinvit_fast` only consists in vector-vector multiplications, and so the asymptotic complexity is $O(n)$.

(1e) ⊡ Tabulate the runtimes of the two *inner loops* of the C++ implementations with different vector sizes $n = 2^k$, $k = 1, 2, 3, \ldots, 9$. Use, as test vector

```
Eigen::VectorXd::LinSpaced(n,1,2)
```

How can you read off the asymptotic complexity from these data?

HINT: Whenever you provide figure from runtime measurements, you have to specify the operating system and compiler (options) used.

Solution: See file `rankoneinvit.cpp`.

Problem 2   Approximating the Hyperbolic Sine

In this problem we study how Taylor expansions can be used to avoid cancellations errors in the approximation of the hyperbolic sine, *cf.* the discussion in [1, Ex. 1.5.60] carefully.

Consider the Matlab code given in Listing 26.

Listing 26: Matlab function `sinh_unstable`

```
1 function y = sinh_unstable(x)
2 t = exp(x);
3 y = 0.5*(t-1/t);
```

2

(2a) ⊙ Explain why the function given in Listing 26 may not give a good approximation of the hyperbolic sine for small values of x, and compute the relative error

$$\frac{\left|\texttt{sinh\_unstable}(x) - \sinh(x)\right|}{\left|\sinh(x)\right|}$$

with Matlab for $x = 10^{-k}$, $k = 1, 2, \ldots, 10$ using as "exact value" the result of the MATLAB built-in function `sinh`.

Solution: As $x \to 0$, the terms t and $1/t$ become close to each other, thereby creating cancellations errors in y. For $x = 10^{-3}$, the relative error computed with Matlab is $6.2 \cdot 10^{-14}$.

(2b) ⊙ Write the Taylor expansion of length m around $x = 0$ of the function e^x and also specify the remainder.

Solution: Given $m \in \mathbb{N}$ and $x \in \mathbb{R}$, there exists $\xi_x \in [0, x]$ such that

$$e^x = \sum_{k=0}^{m} \frac{x^k}{k!} + \frac{e^{\xi_x} x^{m+1}}{(m+1)!} \tag{8}$$

(2c) ⊙ Prove that for every $x \geq 0$ the following inequality holds true:

$$\sinh x \geq x. \tag{9}$$

Solution: The claim is equivalent to proving that $f(x) := e^x - e^{-x} - 2x \geq 0$ for every $x \geq 0$. This follows from the fact that $f(0) = 0$ and $f'(x) = e^x + e^{-x} - 2 \geq 0$ for every $x \geq 0$.

(2d) ⊙ Based on the Taylor expansion, find an approximation for $\sinh(x)$, with $0 \leq x \leq 10^{-3}$, so that the relative approximation error is smaller than 10^{-15}.

Solution: The idea is to use the Taylor expansion given in (8). Inserting this identity in the definition of the hyperbolic sine yields

$$\sinh(x) = \frac{e^x - e^{-x}}{2} = \frac{1}{2} \sum_{k=0}^{m} (1 - (-1)^k) \frac{x^k}{k!} + \frac{e^{\xi_x} x^{m+1} + e^{\xi_{-x}} (-x)^{m+1}}{2(m+1)!}.$$

The parameter m gives the precision of the approximation, since $(m+1)! \to 0$ as $m \to \infty$. We will choose it later to obtain the desired tolerance. Since $1 - (-1)^k = 0$ if k is even, we

3

set $m = 2n$ for some $n \in \mathbb{N}$ to be chosen later. From the above expression we obtain the new approximation given by

$$y_n = \frac{1}{2} \sum_{k=0}^{m} (1 - (-1)^k) \frac{x^k}{k!} = \sum_{j=0}^{n-1} \frac{x^{2j+1}}{(2j+1)!},$$

with remainder

$$y_n - \sinh(x) = \frac{e^{\xi_x} x^{2n+1} - e^{\xi_{-x}} x^{2n+1}}{2(2n+1)!} = \frac{(e^{\xi_x} - e^{\xi_{-x}}) x^{2n+1}}{2(2n+1)!}.$$

Therefore, by (9) and using the obvious inequalities $e^{\xi_x} \le e^x$ and $e^{\xi_{-x}} \le e^x$, the relative error can be bounded by

$$\frac{|y_n - \sinh(x)|}{\sinh(x)} \le \frac{e^x x^{2n}}{(2n+1)!}.$$

Calculating the right hand sides with MATLAB for $n = 1, 2, 3$ and $x = 10^{-3}$ we obtain $1.7 \cdot 10^{-7}$, $8.3 \cdot 10^{-15}$ and $2.0 \cdot 10^{-22}$, respectively.

In conclusion, y_3 gives a relative error below 10^{-15}, as required.

Problem 3   C++ project: triplet format to CRS format   (core problem)

This exercise deals with sparse matrices and their storage in memory. Before beginning, make sure you are prepared on the subject by reading section [1, Section 1.7] of the lecture notes. In particular, refresh yourself in the various *sparse storage formats* discussed in class (cf. [1, Section 1.7.1]). This problem will test your knowledge of algorithms and of advanced C++ features (i.e. structures and classes[1]). You do not need to use EIGEN to solve this problem.

The ultimate goal of this exercise is to devise a function that allows the conversion of a matrix given in *triplet list format* (COO, → [1, § 1.7.6]) to a matrix in *compressed row storage* (CRSm → [1, Ex. 1.7.9]) format. You do not have to follow the subproblems, provided you can devise a suitable conversion function and suitable data structures for you matrices.

(3a) ☺   In section [1, § 1.7.6] you saw how a matrix can be stored in triplet (or coordinate) list format. This format stores a collection of triplets (i, j, v) with $i, j \in \mathbb{N}, i, j \ge 0$ (the indices) and $v \in \mathbb{K}$ (the value at (i, j)). Repetitions of i, j are allowed, meaning that the values at the same indices i, j must be *summed* together in the final matrix.

---

[1]You may have a look at http://www.cplusplus.com/doc/tutorial/classes/.

Define a suitable structure:

```
template <class scalar>
struct TripletMatrix;
```

that stores a matrix of type `scalar` in COO format. You can store sizes and indices in `std::size_t` predefined type.

HINT: Store the rows and columns of the matrix inside the structure.

HINT: You can use a `std::vector<your_type>` to store the collection of triplets.

HINT: You can define an auxiliary structure `Triplet` containing the values i, j, v (with the appropriate types), but you may also use the type `Eigen::Triplet<double>`.

(3b) ⌑ Another format for storing a sparse matrix is the compressed row storage (CRS) format (have a look at [1, Ex. 1.7.9]).

Remark. Here, we are not particularly strict about the "compressed" attribute, meaning that you can store your data in `std::vector`. This may "waste" some memory, because the `std::vector` container adds a padding at the end of is data that allows for `push_back` with amortized $O(1)$ complexity.

Devise a suitable structure:

```
template <class scalar>
struct CRSMatrix;
```

holding the data of the matrix in CRS format.

HINT: Store the rows and columns of the matrix inside the structure. To store the data, you can use `std::vector`.

HINT: You can pair the column indices and value in a single structure `ColValPair`. This will become handy later.

HINT: Equivalently to store the array of pointers to the column indices you can use a nested `std::vector< std::vector<your_type> >`.

(3c) ☑ Optional: write member functions

```
Eigen::Matrix<scalar, Eigen::Dynamic, Eigen::Dynamic>
```

5

```
    TripletMatrix<scalar>::densify();
Eigen::Matrix<scalar, Eigen::Dynamic, Eigen::Dynamic>
    CRSMatrix<scalar>::densify();
```

for the structure `TripletMatrix` and `CRSMatrix` that convert your matrices structures to EIGEN dense matrices types. This can be helpful in debugging your code.

(3d) ⊡ Write a function:

```
template <class scalar>
void tripletToCRS(const TripletMatrix<scalar> & T,
  CRSMatrix<scalar> & C);
```

that converts a matrix $\mathbf{T}$ in COO format to a matrix $\mathbf{C}$ in CRS format. Try to be as efficient as possible.

HINT: The parts of the column indices vector in CRS format that correspond to indicdual rows of the matrix must be ordered and without repetitions, whilst the triplets in the input may be in arbitrary ordering and with repetitions. Take care of those aspects in you function definition.

HINT: If you use a `std::vector` container, have a look at the function `std::sort` or at the functions `std::lower_bound` and `std::insert` (both lead to a valid function with different complexities). Look up their precise definition and specification in a C++11 reference.

HINT: You may want to sort a vector containing a structure with multiple values using a particular ordering (i.e. define a custom ordering on your structure and sort according to this ordering). In C++, the standard function `std::sort` provides a way to sort a vector of type `std::vector<your_type>` by defining a `your_type` member operator:

```
bool your_type::operator<(const your_type & other) const;
```

that returns true if `*this` is less than `other` in your particular ordering. Sorting is then performed according to this ordering.

(3e) ⊡ What is the worse case complexity of your function (in the number of triplets)?

6

HINT: Make appropriate assumptions.

HINT: If you use the C++ standard library functions, look at the documentation: there you can find the complexity of basic container operations.

Solution: In `tripletToCTS.cpp`, we present two alteratives: pushing back the column/value pairs to an unsorted vector and inserting into a sorted vector at the right position. Let k be the number of triplets. The first method has k `push_back` amortized $O(1)$ operations and a sort of a vector of at most k entries (i.e. $O(k \log k)$ complexity). The second methods inserts k triplets using two $O(k)$ operation, for a complexity of $O(k^2)$. However, if there are many repeated triplets with the same index of if you are adding triplets to an already defined matrix, the second method could prove to be less expensive than the first one.

(3f) ⊡ Test the correctness and runtime of your function `tripletToCRS`.

Solution: See `tripletToCTS.cpp`.

Problem 4   MATLAB project: mesh smoothing

[1, Ex. 1.7.21] introduced you to the concept of a (planar) triangulation (→ [1, Def. 1.7.22]) and demonstrated how the node positions for smoothed triangulations (→ [1, Def. 1.7.26]) can be computed by solving sparse linear systems of equations with system matrices describing the combinatorial graph Laplacian of the triangulation.

In this problem we will develop a MATLAB code for mesh smoothing and refinement (→ [1, Def. 1.7.33]) of planar triangulations. Before you start make sure that you understand the definitions of planar triangulation ([1, Def. 1.7.22]), the terminology associated with it, and the notion of a smoothed triangulation ([1, Def. 1.7.26]).

(4a) ⊡ Learn about MATLAB's way of describing triangulations by two vectors and one so-called triangle-node incidence matrix from [1, Ex. 1.7.21] or the documentation of the MATLAB function `triplot`.

(4b) ⊡ (This problem is inspired by the dreary reality of software development, where one is regularly confronted with undocumented code written by somebody else who is no longer around.)

Listing 27 lists an uncommented MATLAB code, which takes the triangle-node incidence matrix of a planar triangulation as input.

Describe in detail what is the purpose of the function `processmesh` defined in the file and how exactly this is achieved. Comment the code accordingly.

Listing 27: An undocumented MATLAB function extracting some information from a triangulation given in MATLAB format

```
1  function [E,Eb] = processmesh(T)
2  N = max(max(T)); M = size(T,1);
3  T = sort(T')';
4  C = [T(:,1) T(:,2); T(:,2) T(:,3); T(:,1) T(:,3)];
5  % Wow! A creative way to use 'sparse'
6  A = sparse(C(:,1),C(:,2),ones(3*M,1),N,N);
7  [I,J] = find(A > 0); E = [I,J];
8  [I,J] = find(A == 1); Eb = [I,J];
```

HINT: Understand what find and sort do using MATLAB doc.

HINT: The MATLAB file meshtest.m demonstrates how to use the function processmesh.

Solution: See documented code processmesh.m.

(4c) ☑ Listing 28 displays another undocumented function getinfo. As arguments it expects the triangle-node incidence matrix of a planar triangulation (according to MATLAB's conventions) and the output E of processmesh. Describe in detail how the function works.

Listing 28: Another undocumented function for extracting specific information from a planar triangulation

```
1  function ET = getinfo(T,E)
2  % Another creative us of 'sparse'
3  L = size(E,1); A = sparse(E(:,1),E(:,2),(1:L)',L,L);
4  ET = [];
5  for tri=T'
6    Eloc = full(A(tri,tri)); Eloc = Eloc + Eloc';
7    ET = [ET; Eloc([8 7 4])];
8  end
```

Solution: See getinfo.m.

(4d) ⊡ In [1, Def. 1.7.33] you saw the definition of a regular refinement of a triangular mesh. Write a MATLAB-function:

```
function [x_ref, y_ref, T_ref] = refinemesh(x,y,T)
```

that takes as argument the data of a triangulation in MATLAB format and returns the corresponding data for the new, refined mesh.

Solution: See `refinemesh.m`.

(4e) ⊡ [1, Eq. (1.7.29)]–[1, Eq. (1.7.30)] describe the sparse linear system of equations satisfied by the coordinates of the interior nodes of a smoothed triangulation. Justify rigorously, why the linear system of equations [1, Eq. (1.7.32)] always has a unique solution. In other words, show that that the part $\mathbf{A}_{\mathrm{int}}$ of the matrix of the combinatorial graph Laplacian associated with the interior nodes is invertible for any planar triangulation.

HINT: Notice that $\mathbf{A}_{\mathrm{int}}$ is diagonally dominant ($\rightarrow$ [1, Def. 1.8.8]).

This observation paves the way for using the same arguments as for sub-problem (3b) of Problem 3 You may also appeal to [1, Lemma 1.8.12].

Solution: Consider $\ker \mathbf{A}_{\mathrm{int}}$. Notice $\mathbf{A}^{int}$ is (non strictly, i.e. weakly) diagonally dominant. However, since there is at least one boundary node, the solution vector cannot be constant (the boundary node is connected to at least one interior note, for which the corresponding row in the matrix does not sum to 0). Hence, the matrix is invertible.

(4f) ☑ A planar triangulation can always be transformed uniquely to a smooth triangulation under translation of the interior nodes (maintaining the same connectivity) (see the lecture notes and the previous subproblem).

Write a MATLAB-function

```
function [xs, ys] = smoothmesh(x,y,T);
```

that performs this transformation to the mesh defined by x, y, T. Return the column vectors xs, ys with the new position of the nodes.

HINT: Use the system of equations in [1, (1.7.32)].

Solution: See `smoothmesh.m`.

Problem 5   Resistance to impedance map

In [1, § 1.6.104], we learned about the Sherman-Morrison-Woodbury update formula [1, Lemma 1.6.113], which allows the efficient solution of a linear system of equations after

a low-rank update according to [1, Eq. (1.6.108)], provided that the setup phase of an elimination ($\rightarrow$ [1, § 1.6.42]) solver has already been done for the system matrix.

In this problem, we examine the concrete application from [1, Ex. 1.6.115], where the update formula is key to efficient implementation. This application is the computation of the impedance of the circuit drawn in Figure 7 as a function of a variable resistance of a *single* circuit element.
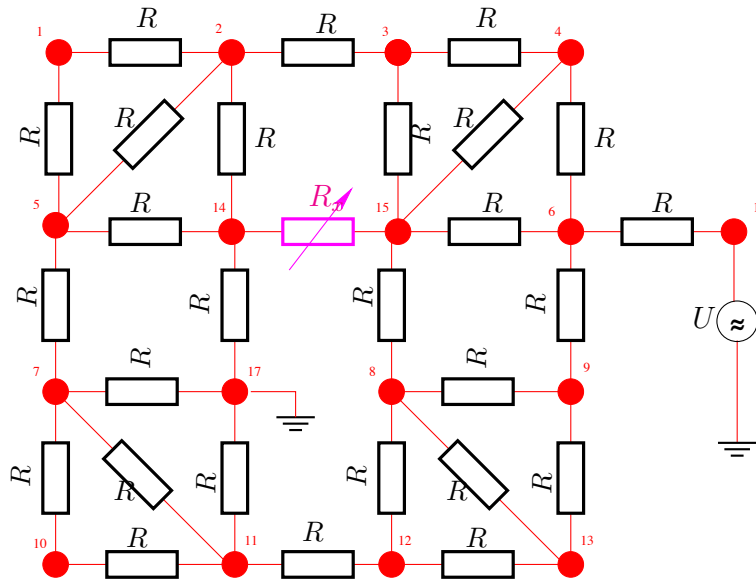

Figure 7: Resistor circuit with a single controlled resistance

(5a) ☐ Study [1, Ex. 1.6.3] that explains how to compute voltages and currents in a linear circuit by means of nodal analysis. Understand how this leads to a linear system of equations for the unknown nodal potentials. The fundamental laws of circuit analysis should be known from physics as well as the principles of nodal analysis.

(5b) ☐ Use nodal analysis to derive the linear system of equations satisfied by the nodal potentials of the circuit from Figure 7. The voltage W is applied to node #16 and node #17 is grounded. All resistors except for the controlled one (colored magenta) have the same resistance R. Use the numbering of nodes indicated in Figure 7.

HINT: Optionally, you can make the computer work for you and find a fast way to build a matrix providing only the essential data. This is less tedious, less error prone and more flexible than specifying each entry individually. For this you can use auxiliary data structures.

Solution: We use the Kirchhoff's first law (as in [1, Ex. 1.6.3]), stating that the sum the currents incident to a node is zero. Let $\mathbf{W} \in \mathbb{R}^{17}$ be the vector of voltages. Set $f(R_x) := R/R_x$. We rescale each sum multiplying by R. Let us denote by $\Delta W_{i,j} := (W_i - W_j)$. The system for each node $i = 1, \ldots, 15$ becomes:

$$
\begin{cases}
\Delta W_{1,2} + \Delta W_{1,5} & = 0 \\
\Delta W_{2,1} + \Delta W_{2,3} + \Delta W_{2,5} + \Delta W_{2,14} & = 0 \\
\Delta W_{3,2} + \Delta W_{3,4} + \Delta W_{3,15} & = 0 \\
\Delta W_{4,3} + \Delta W_{4,6} + \Delta W_{4,15} & = 0 \\
\Delta W_{5,1} + \Delta W_{5,2} + \Delta W_{5,7} + \Delta W_{5,14} & = 0 \\
\Delta W_{6,4} + \Delta W_{6,9} + \Delta W_{6,15} + \Delta W_{6,16} & = 0 \\
\Delta W_{7,5} + \Delta W_{7,10} + \Delta W_{7,11} + \Delta W_{7,17} & = 0 \\
\Delta W_{8,9} + \Delta W_{8,12} + \Delta W_{8,13} + \Delta W_{8,5} & = 0 \\
\Delta W_{9,6} + \Delta W_{9,8} + \Delta W_{9,13} & = 0 \\
\Delta W_{10,7} + \Delta W_{10,11} & = 0 \\
\Delta W_{11,7} + \Delta W_{11,10} + \Delta W_{11,12} + \Delta W_{11,17} & = 0 \\
\Delta W_{12,8} + \Delta W_{12,11} + \Delta W_{12,13} & = 0 \\
\Delta W_{13,8} + \Delta W_{13,9} + \Delta W_{13,12} & = 0 \\
\Delta W_{14,2} + \Delta W_{14,5} + \Delta W_{14,17} + f(R_x)\Delta W_{14,15} & = 0 \\
\Delta W_{15,3} + \Delta W_{15,4} + \Delta W_{15,6} + \Delta W_{15,8} + f(R_x)\Delta W_{15,14} & = 0
\end{cases}
\tag{10}
$$

with the extra condition $W_{16} := W, W_{17} = 0$. We now have to obtain the system matrix. The system is rewritten in the following matrix notation (with $\mathbf{C} \in \mathbb{R}^{15,17}$):

$$
\mathbf{C} = [\mathbf{A}(R_x), \mathbf{B}]\mathbf{W} = \mathbf{0} \iff \mathbf{A}(R_x)\tilde{\mathbf{W}} = -\mathbf{B} \cdot [W_{16}, \ W_{17}]^{\top} =: rhs
$$

with $\mathbf{W} = [\tilde{\mathbf{W}}, W_{16}, W_{17}]$, and with:

$$\mathbf{A}(R_x) = \begin{bmatrix}
2 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & -1 & 3 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
-1 & -1 & 0 & 0 & 4 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & -1 & 0 & 4 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & -1 & 0 & 4 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & -1 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 3 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 2 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & -1 & 3 & 0 & 0 \\
0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3+\frac{R}{R_x} & -\frac{R}{R_x} \\
0 & 0 & -1 & -1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -\frac{R}{R_x} & 4+\frac{R}{R_x}
\end{bmatrix}$$

a square 15×15 matrix and $\mathbf{B} \in \mathbb{R}^{15,2}$ zero except $B_{6,1} = -R$.

(For each term with the form $\beta \cdot \Delta W_{i,j}$ in (10) we have add an entry β in the diagonal i,i and an entry $-\beta$ in the cell i,j (if $j \leq 15$). Entries with $j > 15$ will produce a corresponding positive entry $\beta \cdot U_j$ in the r.h.s.)

(5c) ⊡ Characterize the change in the circuit matrix derived in sub-problem (5b) induced by a change in the value of R_x as a low-rank modification of the circuit matrix. Use as a base state $R = R_x$.

HINT: Four entries of the circuit matrix will change. This amounts to a rank-2-modification in the sense of [1, Eq. (1.6.108)] with suitable matrices $\mathbf{u}$ and $\mathbf{v}$.

Solution: The matrices

$$\mathbf{U} = \begin{bmatrix} 0 & 0 \\ \vdots & \vdots \\ -1 & 1 \\ 1 & -1 \end{bmatrix} = \mathbf{V}$$

are such that $\mathbf{A}(R)$ defined as $\mathbf{A}(R_x)$ allows to write:

$$\mathbf{A}(R_x) = \mathbf{A}(R) - \mathbf{U}\mathbf{V}^{\top}\left(1 - \frac{R}{R_x}\right)/2.$$

Therefore, if we already have the factorization of $\mathbf{A}_0$, we can use the SMW formula for the cheap inversion of $\mathbf{A}(R_x)$.

(5d) ☞ Based on the EIGEN library, implement a C++ class

```cpp
class ImpedanceMap {
public:
  ImpedanceMap(double R_, double W_) : R(R_), W(W_) {
      // TODO: build A0 = A(1), the rhs and factorize A_0
         with lu = A0.lu()
  };
  double operator() (double Rx) const {
      // TODO: compute the perturbation matrix U and
         solve (A+UU^T) x = rhs, from x, U and R compute
         the impedance
  };
private:
  Eigen::PartialPivLU<Eigen::MatrixXd>> lu;
  Eigen::VectorXd rhs;
  double R, W;
};
```

whose `()`-operator returns the impedance of the circuit from Figure 7 when supplied with a concrete value for R_x. Of course, this function should be implemented efficiently using [1, Lemma 1.6.113]. The setup phase of Gaussian elimination should be carried out in the constructor performing the LU-factorization of the circuit matrix.

Test your class using $R = 1, W = 1$ and $R_x = 1, 2, 4, \cdots, 1024$.

HINT: See the file `impedancemap.cpp`.

HINT: The impedance of the circuit is the quotient of the voltage at the input node #16 and the current through the voltage source.

Issue date: 01.10.2015

Hand-in: 08.10.2015 (in the boxes in front of HG G 53/54).