

ETH Lectures 401-0663-00L Numerical Methods for Computer Science
401-2673-00L Numerical Methods for CSE

Homework Problems and Projects

Prof. R. Hiptmair, SAM, ETH Zurich

Autumn Term 2021, Version of October 2, 2022
(C) Seminar für Angewandte Mathematik, ETH Zürich

[Link](#) to the current version of this homework collection

0.1 General Information

0.1.1 Weekly Homework Assignments

All problems will be published in this single “.pdf” file. Every week, we publish a list of problem numbers on the lecture Moodle page. These are the current assignments for that week. Details are given on the lecture Moodle page.

0.1.2 Importance of Homework

Homework assignments are **not** mandatory. However, it is **very important** that you constantly exercise with the material you learn. “Solving” the homework assignments one week before the main exam by looking at the solutions will likely result in failure.

We provide hints and solutions from the beginning. It is your responsibility to look at the solutions only if you are stuck with a difficult problem and you tried to find a solution for a sufficient amount of time.

Most homework problems come with an estimate for the time required for its solution. These times are what we expect a bright and well-prepared student to require for that problem during an exam.

0.1.3 Corrections and Grading of Assignments

Homework assignments will not be graded.

However, you may submit your handwritten solutions by handing them to your tutor directly or uploading them via the Moodle upload interface, see the lecture Moodle page for instructions. The assistants will

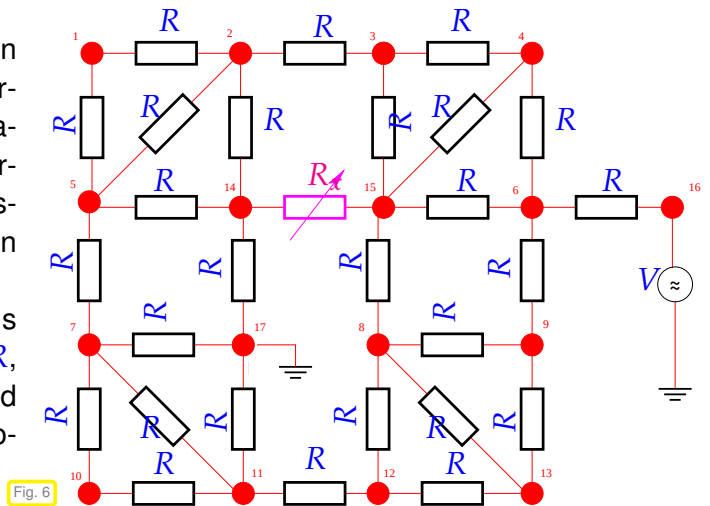
Problem 2-5: Resistance to impedance map

In [Lecture → § 2.6.0.12], we learned about the Sherman-Morrison-Woodbury update formula [Lecture → Lemma 2.6.0.21], which allows the efficient solution of a linear system of equations after a low-rank update according to [Lecture → Eq. (2.6.0.16)], provided that the setup phase of an elimination (→ [Lecture → § 2.3.2.15]) solver has already been done for the system matrix.

This problem is based on [Lecture → Ex. 2.1.0.3] and [Lecture → Ex. 2.6.0.24] and heavily draws on techniques introduced in [Lecture → Section 2.6].

In this problem, we examine the concrete application from [Lecture → Ex. 2.6.0.24], where the update formula is key to efficient implementation. This application is the computation of the impedance of the circuit drawn in Fig. 6 as a function of a variable resistance of a *single* circuit element, the resistor between nodes #14 and #15.

With the exception of that resistor, the circuit contains only identical linear resistors with the resistance R , that is, the relationship between branch currents and voltages is $I = R^{-1}U$ throughout. Excitation is provided by a voltage V imposed at node #16.



Here we consider DC operation (stationary setting), that is, all branch currents, voltages, and nodal potentials are real-valued. Moreover, all currents are given in 1A , all voltages in 1V , and all resistances in $1\Omega = \frac{\text{V}}{\text{A}}$. Thus we can drop the physical units in the following considerations and pretend that we deal with naked numbers even when computing physical quantities.

(2-5.a) ☐ (15 min.) Study [Lecture → Ex. 2.1.0.3] 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. ▲

(2-5.b) ☐ (20 min.) Use nodal analysis to derive the 15×17 linear system of equations $\hat{A}(R_x)\mathbf{x} = \mathbf{0}$ satisfied by the nodal potentials U_i of the circuit from Fig. 6. Here, $\mathbf{x}$ stands for the vector of unknown nodal potentials: $U_i = (\mathbf{x})_i$ is that at node # i , $i \in \{1, \dots, 17\}$. All resistors except for the controlled one (R_x , colored magenta in Fig. 6) have the same resistance R . Use the numbering of nodes indicated in Fig. 6.

SOLUTION of (2-5.b):

We write $I_{j,k}$ for the branch current flowing from node # i to node # k . Moreover, denote by I_G and I_V the currents flowing through the grounding contact and the current source, respectively. According to the

Kirchhoff current law (KCL) applied to every node we have

$$\begin{aligned}
 \text{Node \#1:} & I_{1,2} + I_{1,5} = 0 \\
 \text{Node \#2:} & I_{2,1} + I_{2,3} + I_{2,5} + I_{2,14} = 0 \\
 \text{Node \#3:} & I_{3,2} + I_{3,4} + I_{3,15} = 0 \\
 \text{Node \#4:} & I_{4,3} + I_{4,6} + I_{4,15} = 0 \\
 \text{Node \#5:} & I_{5,1} + I_{5,2} + I_{5,7} + I_{5,14} = 0 \\
 \text{Node \#6:} & I_{6,4} + I_{6,9} + I_{6,15} + I_{6,16} = 0 \\
 \text{Node \#7:} & I_{7,5} + I_{7,10} + I_{7,11} + I_{7,17} = 0 \\
 \text{Node \#8:} & I_{8,9} + I_{8,12} + I_{8,13} + I_{8,15} = 0 \\
 \text{Node \#9:} & I_{9,6} + I_{9,8} + I_{9,13} = 0 \\
 \text{Node \#10:} & I_{10,7} + I_{10,11} = 0 \\
 \text{Node \#11:} & I_{11,7} + I_{11,10} + I_{11,12} + I_{11,17} = 0 \\
 \text{Node \#12:} & I_{12,8} + I_{12,11} + I_{12,13} = 0 \\
 \text{Node \#13:} & I_{13,8} + I_{13,9} + I_{13,12} = 0 \\
 \text{Node \#14:} & I_{14,2} + I_{14,5} + I_{14,17} + I_{14,15} = 0 \\
 \text{Node \#15:} & I_{15,3} + I_{15,4} + I_{15,6} + I_{15,8} + I_{15,14} = 0 \\
 \text{Node \#16:} & I_{16,6} + I_V = 0 \\
 \text{Node \#17:} & I_{17,7} + I_{17,11} + I_{17,14} + I_G = 0.
 \end{aligned}$$

Next, we plug in the circuit element relation $I_{j,k} = R_{j,k}^{-1}(U_j - U_k)$:

$$\begin{aligned}
 \text{Node \#1:} & \frac{1}{R}(U_1 - U_2) + \frac{1}{R}(U_1 - U_5) = 0 \\
 \text{Node \#2:} & \frac{1}{R}(U_2 - U_1) + \frac{1}{R}(U_2 - U_3) + \frac{1}{R}(U_2 - U_5) + \frac{1}{R}(U_2 - U_{14}) = 0 \\
 \text{Node \#3:} & \frac{1}{R}(U_3 - U_2) + \frac{1}{R}(U_3 - U_4) + \frac{1}{R}(U_3 - U_{15}) = 0 \\
 \text{Node \#4:} & \frac{1}{R}(U_4 - U_3) + \frac{1}{R}(U_4 - U_6) + \frac{1}{R}(U_4 - U_{15}) = 0 \\
 \text{Node \#5:} & \frac{1}{R}(U_5 - U_1) + \frac{1}{R}(U_5 - U_2) + \frac{1}{R}(U_5 - U_7) + \frac{1}{R}(U_5 - U_{14}) = 0 \\
 \text{Node \#6:} & \frac{1}{R}(U_6 - U_4) + \frac{1}{R}(U_6 - U_9) + \frac{1}{R}(U_6 - U_{15}) + \frac{1}{R}(U_6 - U_{16}) = 0 \\
 \text{Node \#7:} & \frac{1}{R}(U_7 - U_5) + \frac{1}{R}(U_7 - U_{10}) + \frac{1}{R}(U_7 - U_{11}) + \frac{1}{R}(U_7 - U_{17}) = 0 \\
 \text{Node \#8:} & \frac{1}{R}(U_8 - U_9) + \frac{1}{R}(U_8 - U_{12}) + \frac{1}{R}(U_8 - U_{13}) + \frac{1}{R}(U_8 - U_{15}) = 0 \\
 \text{Node \#9:} & \frac{1}{R}(U_9 - U_6) + \frac{1}{R}(U_9 - U_8) + \frac{1}{R}(U_9 - U_{13}) = 0 \\
 \text{Node \#10:} & \frac{1}{R}(U_{10} - U_7) + \frac{1}{R}(U_{10} - U_{11}) = 0 \\
 \text{Node \#11:} & \frac{1}{R}(U_{11} - U_7) + \frac{1}{R}(U_{11} - U_{10}) + \frac{1}{R}(U_{11} - U_{12}) + \frac{1}{R}(U_{11} - U_{17}) = 0 \\
 \text{Node \#12:} & \frac{1}{R}(U_{12} - U_8) + \frac{1}{R}(U_{12} - U_{11}) + \frac{1}{R}(U_{12} - U_{13}) = 0 \\
 \text{Node \#13:} & \frac{1}{R}(U_{13} - U_8) + \frac{1}{R}(U_{13} - U_9) + \frac{1}{R}(U_{13} - U_{12}) = 0 \\
 \text{Node \#14:} & \frac{1}{R}(U_{14} - U_2) + \frac{1}{R}(U_{14} - U_5) + \frac{1}{R}(U_{14} - U_{17}) + \frac{1}{R_x}(U_{14} - U_{15}) = 0 \\
 \text{Node \#15:} & \frac{1}{R}(U_{15} - U_3) + \frac{1}{R}(U_{15} - U_4) + \frac{1}{R}(U_{15} - U_6) + \frac{1}{R}(U_{15} - U_8) + \frac{1}{R_x}(U_{15} - U_{14}) = 0 \\
 \text{Node \#16:} & \frac{1}{R}(U_{16} - U_6) + I_V = 0 \\
 \text{Node \#17:} & \frac{1}{R}(U_{17} - U_7) + \frac{1}{R}(U_{17} - U_{11}) + \frac{1}{R}(U_{17} - U_{14}) + I_G = 0.
 \end{aligned}$$

This is the requested linear system of equations. The last two equations featuring unknown currents

can be dropped. This yields the final linear system of equations, here given in matrix notation:

$$\begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ -1 & -1 & 0 & 0 & 4 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 4 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 4 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 3 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & -1 & 3 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 + \tilde{\zeta} & -\tilde{\zeta} & 0 & -1 \\ 0 & 0 & -1 & -1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -\tilde{\zeta} & 4 + \tilde{\zeta} & 0 & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \\ U_7 \\ U_8 \\ U_9 \\ U_{10} \\ U_{11} \\ U_{12} \\ U_{13} \\ U_{14} \\ U_{15} \\ U_{16} \\ U_{17} \end{bmatrix} = \mathbf{0}$$

where $\tilde{\zeta} := \frac{R}{R_x}$. Note that we have multiplied (“scaled”) the whole linear system with R .



(2-5.c) (15 min.) [depends on Sub-problem (2-5.b)]

The nodal potential of node #16 is fixed to the value V by connecting it to a grounded voltage source. In addition, node #17 is grounded, that is, its nodal potential is set to zero.

Taking into account these constraints we arrive at a 15×15 linear system of equations $\mathbf{A}(R_x)\mathbf{x} = \mathbf{b}$ for the unknown nodal potentials $U_1, \dots, U_{15}$, which are collected in the vector $\mathbf{x} := [U_1, \dots, U_{15}]^\top$. Determine the system matrix $\mathbf{A} \in \mathbb{R}^{15,15}$ and the right-hand side vector $\mathbf{b} \in \mathbb{R}^{15}$.

SOLUTION of (2-5.c):

In the 15×17 linear system obtained in Sub-problem (2-5.b) we multiply the columns corresponding to the known nodal potentials $U_{16} = V$ and $U_{17} = 0$ with those values and move them to the right-hand.

We end up with the 15×15 system matrix

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

with $\xi := \frac{R}{R_x}$. The right-hand side vector $\mathbf{b}$ is zero except in its sixth component:

$$(\mathbf{b})_i = \begin{cases} V & \text{for } i = 6, \\ 0 & \text{for } i \neq 6. \end{cases}$$

▲

(2-5.d) 🧩 (20 min.) [depends on Sub-problem (2-5.c)]

Provide an efficient implementation of the constructor and of the evaluation operator of the C++ class

```
class NodalPotentials {
public:
    NodalPotentials() = delete;
    NodalPotentials(const NodalPotentials&) = default;
    NodalPotentials(NodalPotentials&&) = default;
    NodalPotentials& operator=(const NodalPotentials&) = default;
    NodalPotentials& operator=(NodalPotentials&&) = default;
    ~NodalPotentials() = default;

    NodalPotentials(double R, double Rx);
    Eigen::VectorXd operator() (double V) const;
private:
    ...
};
```

This class is meant to simulate the circuit from Fig. 6. Its constructor accepts the resistances R and R_x (in Ω) as arguments. The evaluation operator should return the vector $[U_1, \dots, U_{15}]^\top \in \mathbb{R}^{15}$ of nodal voltages when the current source excites the circuit with a voltage V .

HINT 1 for (2-5.d): Use EIGEN's facilities for the direct solution of dense linear systems of equations [Lecture $\rightarrow$ § 2.5.0.4] to solve the LSE derived in Sub-problem (2-5.c). ┘

HINT 2 for (2-5.d): Trick: Note that all nodal voltages depend linearly on V . Thus the nodal voltages can be precomputed in the constructor for $V = 1$ and that vector, scaled with the provided value of V , can just be returned by the evaluation operator. ┘

SOLUTION of (2-5.d):

In the constructor we compute the solution of $\mathbf{A}(R_x)\mathbf{x} = \mathbf{b}$ for $V = 1$ and store it in data member of type **Eigen::VectorXd**. We use the **Topology** struct to build the matrix.

C++ code 2.5.1: Typedefs for **Topology**

```

2 // Models a resistance between node i, j with i < j
3 // (in principle, it can handle different resistances)
4 using resistor = std::pair<int, int>;
5 // Vector containing all resistances in
6 // the circuit (excluded the variable one, which is modelled
  afterwards)
7 using resistor_topology = std::vector<resistor>;
8 // Models a ground or source of voltage, indexes store the node
9 // connected to the ground/source through a resistance
10 using voltage = std::tuple<int, int, double>;
11 // Vector containing a voltage object for each node connected to sink
  or source.
12 using voltage_topology = std::vector<voltage>;

```

C++ code 2.5.2: Implementation of struct **Topology**

```

2 struct Topology {
3     /**
4      * brief Construct a new Topology object**
5      * param V Source voltage V at node 16 (in Volt), ground is set to 0V
6      * at node 17
7      */
8     Topology(double V) {
9         // We implement the topology of the resistances by pushing each
10        // Resistance between node i < j in a std::vector of
11        // pair (i,j) ∈ ℕ²
12        // This way we can automatically build a symmetric
13        // matrix, take care of indexing
14        // and avoid mistakes during the matrix filling
15        T.reserve(23);
16        T.push_back(resistor(1, 2));
17        T.push_back(resistor(1, 5));
18        T.push_back(resistor(2, 5));
19        T.push_back(resistor(2, 3));
20        T.push_back(resistor(2, 14));
21        T.push_back(resistor(3, 4));
22        T.push_back(resistor(3, 15));
23        T.push_back(resistor(4, 6));
24        T.push_back(resistor(4, 15));
25        T.push_back(resistor(5, 7));
26        T.push_back(resistor(5, 14));
27        T.push_back(resistor(6, 9));
28        T.push_back(resistor(6, 15));
29        T.push_back(resistor(7, 10));
30        T.push_back(resistor(7, 11));
31        T.push_back(resistor(8, 9));
32        T.push_back(resistor(8, 12));
33        T.push_back(resistor(8, 13));
34        T.push_back(resistor(8, 15));
35        T.push_back(resistor(9, 13));
36        T.push_back(resistor(10, 11));
37        T.push_back(resistor(11, 12));

```

```

37 T.push_back(resistor(12, 13));
38 // We omit the resistor between node 14 and 15 for an easier
   // matrix filling.
39
40 // We implement voltage ground and sources by
41 // specifying which node is connected to ground/source
42 // through a resistance. This will be also part of the r.h.s.
43 S.reserve(4);
44 S.push_back(voltage(6, 16, V));
45 S.push_back(voltage(7, 17, 0));
46 S.push_back(voltage(11, 17, 0));
47 S.push_back(voltage(14, 17, 0));
48 }
49
50 resistor_topology T;
51 voltage_topology S;
52 };

```

C++ code 2.5.3: Complete definition of class **NodalPotentials**

```

2  class NodalPotentials {
3  public:
4      NodalPotentials() = delete;
5      NodalPotentials(const NodalPotentials&) = default;
6      NodalPotentials(NodalPotentials&&) = default;
7      NodalPotentials& operator=(const NodalPotentials&) = default;
8      NodalPotentials& operator=(NodalPotentials&&) = default;
9      ~NodalPotentials() = default;
10
11     NodalPotentials(double R, double Rx);
12     Eigen::VectorXd operator()(double V) const;
13
14 private:
15     // TODO: (2-5.d) Specify which variables should be stored.
16     // START
17     Eigen::VectorXd nodal_voltage_;
18     double R_, Rx_, xi_;
19     // END
20 };

```

C++ code 2.5.4: Constructor of **NodalPotentials**

```

2  NodalPotentials::NodalPotentials(double R, double Rx)
3      // TODO: (2-5.d) initializer list.
4      // START
5      : R_(R),
6        Rx_(Rx)
7      // END
8  {
9      // TODO: (2-5.d) Build the matrix and the r.h.s. vector by either
   // using
10     // the topology struct or by hardcoding the matrix and vector.
11     // START
12     assert(Rx_ != 0. && "Rx should not be equal to zero!");
13     xi_ = R_ / Rx_;
14
15     // Build the topology for a source voltage of 1.
16     Topology top(1.);

```

```

17 // Automatically build A_Rx filling from topology
18 // A_Rx is the matrix  $A_R$  (i.e. with  $R_x = R$ )
19 Eigen::MatrixXd A_Rx = Eigen::MatrixXd::Zero(15, 15);
20 for (resistor& res : top.T) {
21     // Shift indices down by 1 (indices in Figure start from 1)
22     std::size_t i = res.first - 1;
23     std::size_t j = res.second - 1;
24     // Fill diagonal: each resistance contributes += R
25     // into the diagonal of both nodes
26     A_Rx(i, i) += 1;
27     A_Rx(j, j) += 1;
28     // Fill the off diagonal (negative part in  $\Delta V_{i,j}$ ,
29     // the matrix is kept symmetric
30     A_Rx(i, j) -= 1;
31     A_Rx(j, i) -= 1;
32 }
33
34 // Fill in the rest (source and ground), i.e.
35 // components with  $\Delta V_{i,j}$  with  $j > 15$ .
36 // Each node  $i$  connected to ground or source contributes
37 // to its own diagonal with R
38 for (voltage& volt : top.S) {
39     // Shift index down by 1 and get voltage in V2
40     int i = std::get<0>(volt) - 1;
41     // Add resistance to matrix diagonal: contribution of source
42     // current
43     // scaled by R.
44     A_Rx(i, i) += 1;
45 }
46
47 // Add resistance between node 14 and 15
48 A_Rx(13, 13) += xi_;
49 A_Rx(13, 14) -= xi_;
50 A_Rx(14, 13) -= xi_;
51 A_Rx(14, 14) += xi_;
52
53 // Set up r.h.s.
54 Eigen::VectorXd b = Eigen::VectorXd::Zero(15);
55 b(5) = 1.;
56
57 // Solve unscaled linear system and store in member.
58 nodal_voltage_ = A_Rx.lu().solve(b);
59 // END
60 }

```

C++ code 2.5.5: Evaluation operator of **NodalPotentials**

```

2 Eigen::VectorXd NodalPotentials::operator()(double V) const {
3     Eigen::VectorXd return_vector = Eigen::VectorXd::Zero(15);
4
5     // TODO: (2-5.d) Return the values of the potential at the nodes.
6     // START
7     return_vector = nodal_voltage_ * V;
8     // END
9
10    return return_vector;
11 }

```

Get it on  [GitLab](#) (circuitimpedance.hpp).



(2-5.e) 🕒 (20 min.) [depends on Sub-problem (2-5.c)]

Characterize the change in the circuit matrix $\mathbf{A}(R_x)$ derived in Sub-problem (2-5.c) induced by a change in the value of R_x as a **low-rank modification** ($\rightarrow$ [Lecture $\rightarrow$ § 2.6.0.12]) of the circuit matrix $\mathbf{A}(0)$ of the form

$$\mathbf{A}(R_x) = \mathbf{A}(0) + \mathbf{u}\mathbf{u}^\top \quad \text{with suitable } \mathbf{u} \in \mathbb{R}^{15}. \quad (2.5.6)$$

HINT 1 for (2-5.e): Four entries of the circuit matrix will change. This amounts to a rank-1-modification in the sense of [Lecture $\rightarrow$ Eq. (2.6.0.16)] with suitable vectors $\mathbf{u}$ and $\mathbf{v}$. ┘

SOLUTION of (2-5.e):

Define $\tilde{\xi} := R/R_x$. Define the vector

$$\mathbf{u} := \sqrt{\tilde{\xi}} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -1 \\ 1 \end{bmatrix} \in \mathbb{R}^{15}. \quad (2.5.7)$$

Then we can write

$$\mathbf{A}(R_x) = \mathbf{A}(0) + \mathbf{u}\mathbf{u}^\top.$$



(2-5.f) 🕒 (15 min.) [depends on Sub-problem (2-5.e)]

How can the solution of $\mathbf{A}(R_x) = \mathbf{b}$ be computed relying on

- a solver for linear systems of equations with system matrix $\mathbf{A}(0)$,
- simple vector operations with asymptotic cost proportional to the vector length?

SOLUTION of (2-5.f):

Assume that we have a LU-factorization of $\mathbf{A}(0)$ at our disposal. Then we can use the **Sherman-Morrison Woodbury formula** of [Lecture $\rightarrow$ Lemma 2.6.0.21] for $k = 1$ as given in [Lecture $\rightarrow$ Eq. (2.6.0.22)] to efficiently compute $\tilde{\mathbf{x}} := \mathbf{A}(R_x)^{-1}\mathbf{b}$, with $\mathbf{b} \in \mathbb{R}^{15}$:

$$\tilde{\mathbf{x}} = \mathbf{A}(0)^{-1}\mathbf{b} - \frac{\mathbf{A}(0)^{-1}\mathbf{u}(\mathbf{u}^\top(\mathbf{A}(0)^{-1}\mathbf{b}))}{1 + \mathbf{u}^\top(\mathbf{A}(0)^{-1}\mathbf{u})}. \quad (2.5.8)$$

The colors indicate the vectors that can be obtained by solving a LSE with system matrix $\mathbf{A}(0)$. These vectors can be regarded as “available”.

Supplement & trick: We can simplify formula (2.5.8) even further by examining (2.5.7) and noticing that

$$\mathbf{A}(R_x) = \mathbf{A}(0) + \zeta \mathbf{u}_0 \mathbf{u}_0^\top \quad \text{with} \quad \zeta = \frac{R}{R_x}, \quad \mathbf{u}_0 = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -1 \\ 1 \end{bmatrix} \in \mathbb{R}^{15}. \quad (2.5.9)$$

Thus (2.5.8) can be recast as

$$\tilde{\mathbf{x}} = \mathbf{A}(0)^{-1} \mathbf{b} - \zeta \frac{\mathbf{A}(0)^{-1} \mathbf{u}_0 (\mathbf{u}_0^\top (\mathbf{A}(0)^{-1} \mathbf{b}))}{1 + \zeta \mathbf{u}_0^\top (\mathbf{A}(0)^{-1} \mathbf{u}_0)}.$$

Setting $\mathbf{w} := \mathbf{A}(0)^{-1} \mathbf{b}$, $\mathbf{z} := \mathbf{A}(0)^{-1} \mathbf{u}_0$, we can state this in a more lucid way:

$$\tilde{\mathbf{x}} = \mathbf{w} - \zeta \frac{\mathbf{z} (\mathbf{u}_0^\top \mathbf{w})}{1 + \zeta \mathbf{u}_0^\top \mathbf{z}} = \mathbf{w} - \frac{\zeta \beta}{1 + \zeta \alpha} \mathbf{z}, \quad \beta := \mathbf{u}_0^\top \mathbf{w}, \quad \alpha := \mathbf{u}_0^\top \mathbf{z}. \quad (2.5.10)$$

Note that along with $\mathbf{u}_0$ also α and β are independent of R_x . Only ζ depends on the variable resistance.



(2-5.g) (45 min.) [depends on Sub-problem (2-5.f)]

Based on the EIGEN library, complete the implementation of the C++ class

```
class ImpedanceMap {
public:
    ImpedanceMap() = delete;
    ImpedanceMap(const ImpedanceMap&) = default;
    ImpedanceMap(ImpedanceMap&&) = default;
    ImpedanceMap& operator=(const ImpedanceMap&) = default;
    ImpedanceMap& operator=(ImpedanceMap&&) = default;
    ~ImpedanceMap() = default;

    ImpedanceMap(double R, double V);
    double operator() (double Rx) const;
private:
    ....
};
```

whose evaluation operator `operator()` returns the impedance R_* of the circuit from Fig. 6, when supplied with a concrete value for R_x . The impedance is defined as

$$R_* := \frac{V}{I_V}, \quad I_V := \text{current through voltage source.}$$

This function should be implemented efficiently using the result of Sub-problem (2-5.f). The setup phase for the direct solver should be carried out in the constructor, cf. [Lecture $\rightarrow$ § 2.3.2.15], [Lecture $\rightarrow$ Rem. 2.3.2.16].

HINT 1 for (2-5.g): Since I_V is proportional to V , we can just compute with $V = 1$. I_V itself is available as the current flowing between node #6 and the voltage source: According to the constitutive relation for the resistor between the two we have

$$I_V = R^{-1}(U_6 - V).$$

HINT 2 for (2-5.g): An EIGEN data type that can store the LU-decomposition of a real dense matrix is `Eigen::PartialPivLU<Eigen::MatrixXd>`. A class data member of that type can be used.

SOLUTION of (2-5.g):

“Standard” approach based on (2.5.8):

1. In the constructor we set up the system matrix $\mathbf{A}(0)$ and the right-hand side vector $\mathbf{b}$ (for $V = 1$) as found in Sub-problem (2-5.c). We let EIGEN compute the LU-decomposition of $\mathbf{A}(0)$ and store it. We can also precompute $\mathbf{A}(0)^{-1}\mathbf{b}$ and keep it in another data member of type `Eigen::VectorXd`.
2. In the evaluation operator we compute $\mathbf{A}^{-1}\mathbf{u}$ and evaluate the formula (2.5.8).

“Smart” approach exploiting (2.5.10):

1. In the constructor compute $\mathbf{w} := \mathbf{A}(0)^{-1}\mathbf{b}$, $\mathbf{z} := \mathbf{A}(0)^{-1}\mathbf{u}_0 \in \mathbb{R}^{15}$ and $\alpha := \mathbf{u}_0^\top \mathbf{z}$, $\beta := \mathbf{u}_0^\top \mathbf{w} \in \mathbb{R}$, notations as in (2.5.10). Store them all in appropriate data members.
2. The evaluation operator just has to compute $\zeta := \frac{R}{R_x}$ and for a linear combination of $\mathbf{w}$ and $\mathbf{z}$ according to (2.5.10).

Of course, it is good practice to check whether denominators are “nearly vanishing”, see [Lecture → Rem. 1.5.3.15].

C++ code 2.5.11: Complete definition of class `ImpedanceMap`

```

2  class ImpedanceMap {
3  public:
4      ImpedanceMap() = delete;
5      ImpedanceMap(const ImpedanceMap&) = default;
6      ImpedanceMap(ImpedanceMap&&) = default;
7      ImpedanceMap& operator=(const ImpedanceMap&) = default;
8      ImpedanceMap& operator=(ImpedanceMap&&) = default;
9      ~ImpedanceMap() = default;
10
11     ImpedanceMap(double R, double V);
12     double operator()(double Rx) const;
13
14 private:
15     // TODO: (2-5.g) Specify which variables should be stored.
16     // START
17     double R_, V_; //< Resistance R and source voltage V.
18     Eigen::VectorXd w_, z_;
19     double alpha_, beta_;
20     static const std::size_t nnodes_ = 15;
21     // END
22 };

```

C++ code 2.5.12: Constructor of `ImpedanceMap`

```

2  ImpedanceMap::ImpedanceMap(double R, double V)
3      // TODO: (2-5.g) initializer list.
4      // START
5      : R_(R),
6        V_(V)

```

```

7 // END
8 {
9 // TODO: (2-5.g) Build the matrix and the r.h.s. vector by either
10 // using
11 // the topology or by hardcoding the matrix and vector. Also, do
12 // the setup phase for the solver here or find an even smarter way
13 // for an efficient implementation
14 // START
15 // Build the topology with source voltage V_.
16 Topology top(V_);
17
18 // Automatically build A0 filling from topology
19 // A0 is the matrix  $A_R$  (i.e. with  $R_x = R$ )
20 Eigen::MatrixXd A0 = Eigen::MatrixXd::Zero(nnodes_, nnodes_);
21 for (resistor& res : top.T) {
22 // Shift indices down by 1 (indices in Figure start from 1)
23 std::size_t i = res.first - 1;
24 std::size_t j = res.second - 1;
25 // Fill diagonal: each resistance contributes += R
26 // into the diagonal of both nodes
27 A0(i, i) += 1;
28 A0(j, j) += 1;
29 // Fill the off diagonal (negative part in  $\Delta V_{i,j}$ ,
30 // the matrix is kept symmetric
31 A0(i, j) -= 1;
32 A0(j, i) -= 1;
33 }
34
35 // Fill in the rest (source and ground), i.e.
36 // components with  $\Delta V_{i,j}$  with  $j > 15$ .
37 // Each node  $i$  connected to ground or source contributes
38 // to the r.h.s  $b$  with  $R \cdot V$ 
39 // ( $R$  is the resistance between node  $i$  and ground/source
40 // node,  $V$  is voltage at sink or source)
41 // and to its own diagonal with  $R$ 
42 Eigen::VectorXd b = Eigen::VectorXd::Zero(nnodes_);
43 for (voltage& volt : top.S) {
44 // Shift index down by 1 and get voltage in  $V_2$ 
45 int i = std::get<0>(volt) - 1;
46 double source_voltage = std::get<2>(volt);
47 // Add voltage to r.h.s. (resistance assumed to be  $R$ )
48 b(i) += source_voltage;
49 // Add resistance to matrix diagonal: contribution of source
50 // current
51 // scaled by  $R$ .
52 A0(i, i) += 1;
53 }
54
55 // Following the "smart" approach for the SMW formula;
56 // we don't even have to store the LU-decomposition!
57 Eigen::PartialPivLU<Eigen::MatrixXd> lu(A0);
58 w_ = lu.solve(b);
59 Eigen::VectorXd u0 = Eigen::VectorXd::Zero(nnodes_);
60 u0(13) = -1.;
61 u0(14) = 1.;
62 z_ = lu.solve(u0);
63 alpha_ = u0.dot(z_);
64 beta_ = u0.dot(w_);
65 // END
66 }

```

C++ code 2.5.13: Evaluation operator of ImpedanceMap

```

2  double ImpedanceMap::operator()(double Rx) const {
3      double return_value = 0.;
4
5      // TODO: (2-5.g) Return the impedance of the circuit given a
        specific Rx
6      // START
7      assert(Rx != 0 && "Rx should not be equal to zero!");
8      // Store the scaled factor for convenience
9      double xi = R_ / Rx;
10
11     // Put the formula together, x is a column vector containing
        voltages
12     // at each node (except 16 and 17, which are prescribed)
13     Eigen::VectorXd x = w_ - xi * beta_ / (1. + xi * alpha_) * z_;
14
15     // Compute the current  $I = -\Delta V_{16,6}/R$ 
16     // and then impedance =  $V/I$ .
17     // Here  $\Delta V_{16,6} = V_{16} - x_6 = V - x_6$ .
18     return_value = V_ * R_ / (x(5) - V_);
19     // END
20
21     return return_value;
22 }

```

Get it on  GitLab (circuitimpedance.hpp).



End Problem 2-5, 150 min.