

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 2-18: Gauss-Seidel Iteration for a Sparse Linear System of Equations

This problem studies a particular iterative method defined by a matrix and a vector, the so-called **Gauss-Seidel iteration**. We consider its algorithmic aspects when the matrix is given in a raw CRS format.

Familiarity with [Lecture → Section 2.7.1] is essential for this problem. Some concepts from [Lecture → Section 8.2.2] will also be used. Implementation in C++ based on EIGEN will be requested.

Given the **regular** square matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ and the vector $\mathbf{b} \in \mathbb{R}^n$, $n \in \mathbb{N}$, the **Gauss-Seidel iteration** $\mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)})$, $\Phi: \mathbb{R}^n \rightarrow \mathbb{R}^n$, is defined as

$$\left(\mathbf{x}^{(k+1)}\right)_i = (\mathbf{A})_{i,i}^{-1} \left((\mathbf{b})_i - \sum_{j=1}^{i-1} (\mathbf{A})_{i,j} \left(\mathbf{x}^{(k+1)}\right)_j - \sum_{j=i+1}^n (\mathbf{A})_{i,j} \left(\mathbf{x}^{(k)}\right)_j \right), \quad i = 1, 2, \dots, n \text{ (in this order!)}. \quad (2.18.1)$$

Here sums for which the lower index is larger than the upper are meant to evaluate to zero.

The matrix $\mathbf{A}$ is available in a raw compressed row-storage (CRS) format, encapsulated in the following data type:

```
struct CRSMatrix {
    unsigned int m;           // number of rows
    unsigned int n;           // number of columns
    std::vector<double> val;    // value array
    std::vector<unsigned int> col_ind; // same length as value array
    std::vector<unsigned int> row_ptr; // length m+1, row_ptr[m] ==
    val.size();
};
```

Refer to [Lecture → § 2.7.1.4] for further explanations on the meaning of the fields of **CRSMatrix**.

(2-18.a) 🕒 (10 min.) For what square matrices $\mathbf{A}$ and vectors $\mathbf{b}$ is the Gauss-Seidel iteration (2.18.1) well-defined?

Iteration (2.18.1) well-defined $\forall \mathbf{A} \in \mathbb{R}^{n,n}, \mathbf{b} \in \mathbb{R}^n$:

SOLUTION of (2-18.a):

The only obstruction to (2.18.1) being well-defined is a failure of the division due to $(\mathbf{A})_{i,i} = 0$. Therefore the Gauss-Seidel iteration is well defined for all matrices, for which all diagonal entries are non-zero.

iteration (2.18.1) well-defined $\forall \mathbf{A} \in \mathbb{R}^{n,n}, \mathbf{b} \in \mathbb{R}^n: (\mathbf{A})_{i,i} \neq 0 \quad \forall i \in \{1, \dots, n\}$.



(2-18.b) 📺 (45 min.)

In the file `gaussseidelcrs.hpp` implement the C++ function

```
bool GaussSeidelstep_crs (
    const CRSMatrix &A, const Eigen::VectorXd &b,
    Eigen::VectorXd &x);
```

that carries out a single step of the Gauss-Seidel iteration according to (2.18.1). The vector $\mathbf{x}^{(k)}$ is passed through `x` and it should return $\mathbf{x}^{(k+1)}$ also in `x`. **false** is returned in case the iteration is not well-defined, **true** otherwise.

SOLUTION of (2-18.b):

Let us first develop a code for the Gauss-Seidel iteration for a generic matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, assuming that we can access any entry:

C++ code 2.18.2: EIGEN-based implementation of `GaussSeidelstep_generic()`

```
2  template <typename MATRIX>
3  bool GaussSeidelstep_generic(const MATRIX &A, const Eigen::VectorXd &b,
4                                Eigen::VectorXd &x) {
5      const int n = A.cols();
6      assert(n == A.rows() && "Matrix must be square");
7      assert(n == b.size() && "Vector length mismatch");
8      assert(n == x.size() && "Vector b length mismatch");
9
10     for (int i = 0; i < n; ++i) {
11         if (A(i, i) == 0.0) return false;
12         double s = b[i];
13         for (int j = 0; j < i; ++j) s -= A(i, j) * x[j];
14         for (int j = i + 1; j < n; ++j) s -= A(i, j) * x[j];
15         x[i] = s / A(i, i);
16     }
17     return true;
18 }
```

Get it on 🐙 [GitLab](#) ([gaussseidelcrs.hpp](#)).

From this code we see that the entries of $\mathbf{A}$ are accessed *row-wise sequentially*. This is exactly their arrangement in the `val` field of `CRSMatrix`, which suggests an implementation based on an outer loop running through the `val` and `col_ind` vectors.

Also remember that the CRS-format for $\mathbf{A} \in \mathbb{R}^{n,n}$ is defined by the relationship ("C++ indexing")

$$\text{val}[k] = (\mathbf{A})_{ij} \Leftrightarrow \begin{cases} \text{col_ind}[k] = j, \\ \text{row_ptr}[i] \leq k < \text{row_ptr}[i+1], \end{cases} \quad 0 \leq k \leq \text{val.size()} ,$$

for $i, j \in \{0, \dots, n-1\}$. Therefore the index bounds stored in `row_ptr` can be used in an inner loop.

C++ code 2.18.3: EIGEN-based implementation of `GaussSeidelstep_crs()`

```
2  bool GaussSeidelstep_crs(const CRSMatrix &A, const Eigen::VectorXd &b,
3                                Eigen::VectorXd &x) {
4      assert(A.n == A.m && "Matrix must be square");
5      assert(A.n == b.size() && "Vector b length mismatch");
6      assert(A.n == x.size() && "Vector x length mismatch");
7  }
```

```

8 // START student code
9 // Outer loop over rows of the matrix
10 for (unsigned int i = 0; i < A.n; ++i) {
11     double Aii = 0.0;
12     double s = b[i];
13     // Inner summation loop over non-zero entries of i-th row.
14     // Skip diagonal entry in the summation and store it separately.
15     for (unsigned int l = A.row_ptr[i]; l < A.row_ptr[i + 1]; ++l) {
16         const unsigned int j = A.col_ind[l];
17         if (j != i) {
18             s -= A.val[l] * x[j];
19         } else {
20             // Fetch diagonal entry of A.
21             Aii = A.val[l];
22         }
23     }
24     if (Aii != 0.0)
25         x[i] = s / Aii;
26     else
27         return false;
28 }
29 // END student code
30 return true;
31 }

```

Get it on  GitLab (gaussseidelcrs.hpp).



(2-18.c)  (20 min.) For given regular $\mathbf{A} \in \mathbb{R}^{n,n}$ and $\mathbf{b} \in \mathbb{R}^n$ assume that the Gauss-Seidel iteration (2.18.1) is well-defined. Characterize the set of all **fixed points** of the Gauss-Seidel iteration (2.18.1) by a compact formula:

$$\{\mathbf{x}^* \in \mathbb{R}^n : \mathbf{x}^* \text{ is fixed point of (2.18.1)}\} = \boxed{\phantom{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} = \mathbf{b}}}.$$

A fixed point of an iteration $\mathbf{x}^{(k+1)} = F(\mathbf{x}^{(k)})$ is a vector $\mathbf{x}^*$ such that $\mathbf{x}^* = F(\mathbf{x}^*)$.

SOLUTION of (2-18.c):

$\mathbf{x}^*$ is a fixed point of (2.18.1), if $\mathbf{x}^{(k)} = \mathbf{x}^*$ implies $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$, which for (2.18.1) means

$$\begin{aligned}
 (\mathbf{x}^*)_i &= (\mathbf{A})_{i,i}^{-1} \left((\mathbf{b})_i - \sum_{j=1}^{i-1} (\mathbf{A})_{i,j} (\mathbf{x}^*)_j - \sum_{j=i+1}^n (\mathbf{A})_{i,j} (\mathbf{x}^*)_j \right), \quad i = 1, 2, \dots, n \\
 \Leftrightarrow (\mathbf{A})_{i,i} (\mathbf{x}^*)_i &= (\mathbf{b})_i - \sum_{j=1}^{i-1} (\mathbf{A})_{i,j} (\mathbf{x}^*)_j - \sum_{j=i+1}^n (\mathbf{A})_{i,j} (\mathbf{x}^*)_j, \quad i = 1, 2, \dots, n \\
 \Leftrightarrow \sum_{j=1}^n (\mathbf{A})_{i,j} (\mathbf{x}^*)_j &= (\mathbf{b})_i, \quad i = 1, 2, \dots, n \\
 \Leftrightarrow \mathbf{A}\mathbf{x}^* &= \mathbf{b} \quad !.
 \end{aligned}$$

Hence,

$$\{\mathbf{x}^* \in \mathbb{R}^n : \mathbf{x}^* \text{ is fixed point of (2.18.1)}\} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} = \mathbf{b}\},$$

or, equivalently, since $\mathbf{A}$ is regular

$$\{\mathbf{x}^* \in \mathbb{R}^n : \mathbf{x}^* \text{ is fixed point of (2.18.1)}\} = \{\mathbf{A}^{-1}\mathbf{b}\}. \quad (2.18.4)$$

We conclude that there is a unique fixed point of (2.18.1), which coincides with the solution of the linear system of equations $\mathbf{Ax} = \mathbf{b}$.



(2-18.d) 📄 (30 min.) [depends on Sub-problem (2-18.b)]

Develop a C++ function

```
bool GaussSeidel_iteration(
    const CRSMatrix &A, const Eigen::VectorXd &b, Eigen::VectorXd &x,
    double atol = 1.0E-8, double rtol = 1.0E-6,
    unsigned int maxit = 100);
```

which carries out Gauss-Seidel iterations for the data $\mathbf{A} \in \mathbb{R}^{n,n}$, $\mathbf{b} \in \mathbb{R}^n$ (arguments $\mathbf{A}$ and $\mathbf{b}$), and with initial guess $\mathbf{x}^{(0)}$ passed in $\mathbf{x}$. The same argument is used to return the final iterate. The iteration is to terminate after `maxit` steps. If this happens, **false** should be returned, **true** otherwise.

The function should rely on the **correction-based termination criterion** [Lecture → Eq. (8.5.3.2)] with relative and absolute tolerance passed in `rtol` and `atol`, respectively and based on the Euclidean vector norm.

SOLUTION of (2-18.d):

The correction-based termination criterion reads:

$$\text{STOP, when } \|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|_2 \leq \text{rtol} \|\mathbf{x}^{(k+1)}\|_2 \text{ or } \|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|_2 \leq \text{atol}. \quad (2.18.5)$$

Therefore we have to store the previous iterate temporarily. This is done in the following code, which relies on `GaussSeidelstep_crs` from Sub-problem (2-18.b).

C++ code 2.18.6: EIGEN-based implementation of `GaussSeidel_iteration()`

```
2 template <typename RECORDER = std::function<void(double, double)>>
3 bool GaussSeidel_iteration_II(
4     const CRSMatrix &A, const Eigen::VectorXd &b, Eigen::VectorXd &x,
5     double atol = 1.0E-8, double rtol = 1.0E-6, unsigned int maxit = 100,
6     RECORDER &&rec = [] (double, double) -> void {} ) {
7     assert(A.n == A.m && "Matrix must be square");
8     assert(A.n == b.size() && "Vector b length mismatch");
9     assert(A.n == x.size() && "Vector x length mismatch");
10
11     // Main loop for Gauss-Seidel iteration
12     Eigen::VectorXd x_old{x};
13     for (unsigned int k = 0; k < maxit; ++k) {
14         // START student code
15         if (!GaussSeidelstep_crs(A, b, x)) return false;
16         const double delta_x_norm = (x - x_old).norm();
17         const double x_norm = x.norm();
18         rec(x_norm, delta_x_norm); //
19         if ((delta_x_norm < atol) || (delta_x_norm < rtol * x_norm)) { //
```

```

20     return true;
21 }
22 x_old = x;
23 // END student code
24 }
25 // No termination
26 return false;
27 }

```

Get it on  [GitLab \(gaussseidelcrs.hpp\)](#).

The stopping rule (2.18.5) is implemented in Line 19.

This code also takes a recorder object as an argument to keep track of the norms of the iterates and of the updates. This is an add-on, which was not requested in this sub-problem.

A more efficient implementation that dispenses with temporary vectors is given next.

C++ code 2.18.7: EIGEN-based implementation of `GaussSeidel_iteration()`

```

2  bool GaussSeidel_iteration(const CRSMatrix &A, const Eigen::VectorXd &b,
3                          Eigen::VectorXd &x, double atol = 1.0E-8,
4                          double rtol = 1.0E-6, unsigned int maxit = 100) {
5      assert(A.n == A.m && "Matrix must be square");
6      assert(A.n == b.size() && "Vector b length mismatch");
7      assert(A.n == x.size() && "Vector x length mismatch");
8
9      // Main loop for Gauss-Seidel iteration
10     for (unsigned int k = 0; k < maxit; ++k) {
11         // START student code
12         double x_normsq = 0.0;
13         double delta_x_normsq = 0.0;
14         for (unsigned int i = 0; i < A.n; ++i) {
15             double Aii = 0.0;
16             double s = b[i];
17             for (unsigned int l = A.row_ptr[i]; l < A.row_ptr[i + 1]; ++l) {
18                 const unsigned int j = A.col_ind[l];
19                 if (j != i) {
20                     s -= A.val[l] * x[j];
21                 } else {
22                     // Fetch diagonal entry of A
23                     Aii = A.val[l];
24                 }
25             }
26             if (Aii != 0.0) {
27                 // Update iterate and compute Eucidean norm of current iterate
28                 // and of correction
29                 const double tmp = x[i];
30                 x[i] = s / Aii;
31                 delta_x_normsq += (x[i] - tmp) * (x[i] - tmp);
32                 x_normsq += x[i] * x[i];
33             } else
34                 return false;
35         }
36         // Correction based termination criterion
37         if ((delta_x_normsq < atol * atol) ||
38             (delta_x_normsq < rtol * rtol * x_normsq)) {
39             return true;
40         }
41         // END student code

```

```

42 }
43 // No termination
44 return false;
45 }

```

Get it on  [GitLab \(gaussseidelcrs.hpp\)](#).



(2-18.e)  (5 min.) [depends on Sub-problem (2-18.c)]

We apply the Gauss-Seidel iteration for

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 2 & 8 & 2 & 0 & 0 \\ 3 & 0 & 1 & 10 & 5 & 0 \\ 0 & 2 & 0 & 3 & 13 & 2 \\ 0 & 4 & 0 & 0 & 2 & 11 \end{bmatrix}, \quad \mathbf{b} = \mathbf{0}.$$

and initial guess $\mathbf{x}^{(0)} = [1, 2, 3, 4, 5, 6]^T$. The following table displays the progress of the iteration:

k	$\ \mathbf{x}^{(k)}\ _2$	$\ \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\ _2$	$\ \mathbf{x}^{(k)}\ _2 : \ \mathbf{x}^{(k-1)}\ _2$
1	3.863865e+00	1.127304e+01	
2	8.297690e-01	3.751824e+00	2.147510e-01
3	6.432366e-02	7.857529e-01	7.751996e-02
4	1.476504e-02	5.984439e-02	2.295429e-01
5	4.068577e-03	1.098126e-02	2.755548e-01
6	9.204751e-04	3.155445e-03	2.262401e-01
7	1.880926e-04	7.327086e-04	2.043429e-01
8	3.636497e-05	1.517466e-04	1.933355e-01
9	6.803531e-06	2.956273e-05	1.870902e-01
10	1.246867e-06	5.556761e-06	1.832676e-01
11	2.254491e-07	1.021425e-06	1.808125e-01
12	4.039721e-08	1.850525e-07	1.791855e-01
13	7.194116e-09	3.320315e-08	1.780844e-01
14	1.275722e-09	5.918398e-09	1.773286e-01

Describe qualitatively and quantitatively the observed *asymptotic* convergence of the iteration for this example.

SOLUTION of (2-18.e):

From the data we conclude that the iteration converges and that $\mathbf{x}^{(k)} \rightarrow \mathbf{0}$.

According to (2.18.4) for $\mathbf{b} = \mathbf{0}$ the only fixed point of the iteration will be $\mathbf{x}^* = \mathbf{0}$. Hence, also the iteration error $\mathbf{x}^{(k)} - \mathbf{x}^*$ is given by $\mathbf{x}^{(k)}$. From the last column of the table we see that for sufficiently large k (this is the meaning of “asymptotic” in the problem statement)

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\|_2 = \|\mathbf{x}^{(k+1)}\|_2 \leq 0.18 \|\mathbf{x}^{(k)} - \mathbf{x}^*\|_2 = 0.18 \|\mathbf{x}^{(k)}\|_2,$$

which means that the iteration *converges asymptotically linearly* with rate ≈ 0.18 .



End Problem 2-18 , 110 min.