A SKYLINE SPARSE MATRIX SOLVER IN RUST

1. Theory

1.1. **Notations.** We want to solve a linear system

$$Ax = b, \tag{1.1}$$

where A is a $n \times n$ real (for instance) matrix. We use the following notations:

- The range $i \dots j$ is the list of integers $[i, i+1, \dots, j-1]$. Please note that j is not included in the range (this is similar to the C, Python and Rust notations) and that if if $i \geq j$, then $i \dots j = []$ (the range is empty).
- The elements of the matrix A are noted a^i_j: the row index i is noted with a superscript and the column index j with a subscript. We use the convention that the row and column indices are in the range $0 \dots n$.
- $a^{i \dots k}_{j \dots l}$ is the sub-matrix of A constructed from the given rows and columns ranges. If one range is empty, the sub-matrix is empty . The scalar product of two empty vectors is assumed to be zero.

1.2. *LU* **decomposition.** It is generally possible to write A under the form

$$A = LU, \tag{1.2}$$

where L is a lower triangular matrix with a unit diagonal and U is an upper triangular matrix. The LU decomposition allows to reduce (1.1) to the resolution of two triangular linear systems.

Traditionally, in the computer algorithm, the decomposition is stored in the same place as the initial matrix A. The following algorithm (Doolittle algorithm) replaces A by $L - I + U$. For an easier reading, we distinguish between L, U and A, but in practice we can replace L and U by A in the Doolittle algorithm.

```
• Initialization: $L - I + U = A$.
• for $k$ in $1 \dots n$ (diagonal loop on the pivots)
    – for $j$ in $0 \dots k$
          $$L^k_j \leftarrow \frac{1}{U^j_j}\left(L^k_j - \sum_{0 \leq p < j} L^k_p U^p_j\right)$$
      (update row of $L$ left to the pivot).
    – for $i$ in $0 \dots k+1$
          $$U^i_k \leftarrow U^i_k - \sum_{0 \leq p < i} L^i_p \cdot U^p_k$$
      (update the column of $U$ above the pivot).
```

This algorithm works if the pivot U^k_k never vanishes. It can be rewritten with scalar products of sub-rows of L with sub-columns of U:

```
• Initialization: $L - I + U = A$.
• for $k$ in $1 \dots n$
```

– for j in $0 \dots k$

$$L^k_j \leftarrow L^k_j - L^k_{0\dots j} \cdot U^{0\dots j}_j$$

$$L^k_j \leftarrow \frac{L^k_j}{U^j_j}$$

– for i in $0 \dots k+1$

$$U^i_k \leftarrow U^i_k - L^i_{0\dots i} \cdot U^{0\dots i}_k$$

1.3. **Triangular solve.** Once the decomposition is computed, for solving

$$Ax = b,$$

we solve two triangular systems, first

$$Ly = b,$$

and then

$$Ux = y.$$

The solution of $Ly = b$ is given by the following algorithm

- Initialization: $y^0 = b^0$
- for i in $1 \dots n$

$$y^i = b^i - \sum_{0 \le j < i} L^i_j y^j$$

or

$$y^i = b^i - L^i_{0\dots i} \cdot y^{0\dots i}$$

The solution of $Ux = y$ is given by the following algorithm

- Initialization: $x^{n-1} = y^{n-1}/U^{n-1}_{n-1}$
- for i in $[n-2, n-3, \dots, 0]$

$$x^i = \frac{1}{U^i_i}\left(y^i - \sum_{i+1 \le j < n} U^i_j x^j\right)$$

or

$$x^i = \frac{1}{U^i_i}\left(y^i - U^i_{i+1\dots n} x^{i+1\dots n}\right)$$

2. Implémentation for sparse matrices

2.1. **Sparse skyline matrix.** The matrix A is said to be sparse if it contains many zeros. For this kind of matrix it is interesting to find data structures that avoid the storage of the zeros and unnecessary operations with these zeros. A nice case is when the pattern of the zeros is the same for A and $L - I + U$. It is indeed possible to find such patterns. The skyline storage is an example of such a pattern. We need some definitions:

- An array prof$[0 \dots n]$ is called a profile of A if $A^i_j = 0$ when $j <$ prof$[i]$. A couple of indices (i, j) is said to be in the profile of A if $j \ge$ prof$[i]$.
- An array sky$[0 \dots n]$ is called a skyline of A if $A^i_j = 0$ when $i <$ sky$[j]$. A couple of indices (i, j) is said to be in the skyline of A if $i \ge$ sky$[j]$.

Remark: if (i, j) correspond to a non-zero value $a^i_j \neq 0$ then (i, j) is both in the profile and in the skyline of A. We can have zeros in the profile and in the skyline, they are simply treated as non-zero terms. The limit cases correspond to prof$[i] = 0$, sky$[j] = 0$ (full matrix storage) and prof$[i] = i$, sky$[j] = j$ (diagonal matrix).

It is possible to prove the following theorem.

Theorem 1. *If prof is a profile of A and sky a skyline of A then prof remains a profile and sky a skyline during all the steps of the Gauss algorithm and in the end prof is a profile of $L - I + U$ and sky a skyline of $L - I + U$.*

The most efficient method for storing a sparse matrix in term of memory is the coordinate format. Assume that the matrix has nnz non-zero entries. We store it in an array coo$[0 \dots nnz]$ of triplets (i, j, v) such that

$$\text{coo}[k] = (i, j, v) \Rightarrow a^i_j = v.$$

We will store the lines of L (without the diagonal terms) in the array ltab, the columns of U (with the diagonal terms) in utab. ltab and utab are two vectors of vectors of 64 bits double precision numbers. Because we cannot know a priori the number of non-zero terms in the lines of L and in the columns of U we need first to construct the skyline and the profile. This is done by the following algorithm:

```
• Initialization: $\text{prof}[i] = i$, $\text{sky}[j] = j$.
• for $k$ in $0 \dots nnz$
    – $(i, j, v) = \text{coo}[k]$
    – if $i > j$ and $v \neq 0$ then $\text{prof}[i] = \min(\text{prof}[i], j)$
    – if $i < j$ and $v \neq 0$ then $\text{sky}[j] = \min(\text{sky}[j], i)$
```

We can now construct ltab[][] and utab[][] with the following algorithm

```
• Initialization: $\text{ltab}[i] = [0; i - \text{prof}[i]]$, $\text{utab}[j] = [0; j - \text{sky}[j] + 1]$
• for $k$ in $0 \dots nnz$
    – $(i, j, v) = \text{coo}[k]$
    – if $i > j$ then $\text{ltab}[i][j - \text{prof}[i]] {+}{=} v$
    – if $i \leq j$ then $\text{utab}[j][i - \text{sky}[j]] {+}{=} v$
```

2.2. **Optimisation of the Doolittle algorithm.** The Doolittle elimination algorithm given above can yet be optimized for the skyline representation. This gives:

```
• for $k$ in $1 \dots n$
    – for $j$ in $\text{prof}[k] \dots k$
        $$L^k_j \leftarrow \frac{1}{U^j_j}\left(L^k_j - \sum_p L^k_p U^p_j\right), \quad p \in \max(\text{prof}[k], \text{sky}[j]) \dots j$$
    – for $i$ in $\text{sky}[k] \dots k + 1$
        $$U^i_k \leftarrow U^i_k - \sum_p L^i_p \cdot U^p_k, \quad p \in \max(\text{prof}[i], \text{sky}[k]) \dots i$$
```

2.3. **Optimisation of the triangular solvers.** The optimisation of the L solver is easy

```
• for $i$ in $0 \dots n$
        $$y^i = b^i - L^i_{\text{prof}[i] \dots i} \cdot y^{0 \dots i}$$
```

The optimisation of the U solver is less simple (because the rows of U are less natural to access than the columns in the skyline storage). The naive algorithm would be to write:

- for i in $[n-1, n-2, \dots, 0]$

$$x^i = \frac{1}{U_i^i}\left(y^i - \sum_{i<j<n} U_j^i x^j\right), \quad \max(i, \mathrm{sky}[j]) < j < n$$

But this is not the good method, because one would have to loop on all j and test if $\max(i, \mathrm{sky}[j]) < j$. A better way is to loop on the columns of U starting from the last one and add the corresponding contribution to the x vector. When the contributions of column j are treated we can divide $x^{j-1}$by $U_{j-1}^{j-1}$and continue. We obtain the following algorithm

- Initialization $x^{n-1} = y^{n-1}/U_{n-1}^{n-1}$
- for j in $[n-2, n-3, \dots, 0]$
  - for i in $0..j+1$

$$x^i \leftarrow x^i - U_{j+1}^i x^{j+1}$$

  - $x^j \leftarrow x^j/U_j^j$

In this form, we can use the skyline structure, we obtain

- Initialization $x^{n-1} = y^{n-1}/U_{n-1}^{n-1}$
- for j in $[n-2, n-3, \dots, 0]$
  - for i in $\mathrm{sky}[j+1]..j+1$

$$x^i \leftarrow x^i - U_{j+1}^i x^{j+1}$$

  - $x^j \leftarrow x^j/U_j^j$

Final trick: if the vector b is not needed anymore, the computations can be done in place with $x = y = b$. At the end b is replaced by the solution x.