Projekt 6: Cholesky factorization of sparse matrices

Sparse matrices are characterized by the fact that only "few" entries (per row/column) are nonzero. Therefore, only the nonzero entries are stored. Several efficient data formats are commonly used. One of the most popular ones is the *CSC format* (*c*ompressed *s*parse *c*olumn). The following data structure represents this format:

```
class SCSMatrix
{
  unsigned int    mDim;
  unsigned int  * mColStarts;
  unsigned int  * mEntryPos;
  double        * mEntries;
};
```

Here:

- `mDim`: number of columns of the matrix
- `mColStarts`: an array with one more entry than the matrix has columns. The last entry is the number of non-zero entries. The entries `mColStarts`[i], $i = 0, \ldots, nDim - 1$, give the index in `mEntryPos` and `mEntries` of the first entry of column i.
- `mEntryPos` and `mEntries` contain, column by column, the row indices and the value of the non-zero entries. Row indices are assumed to be sorted in ascending order.

For example, the matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0.5 & 3 & 0 \\ 0 & \pi & 0 & 7 & -5 \\ -11 & 0 & 0 & 2 & 0 \\ 0 & 0 & 10 & 0 & 0 \end{pmatrix} \tag{1}$$

has the following CSC representation:

```
mDim       = 5;
mColStarts = {0, 2, 3, 5, 8, 10};
mEntryPos  = {0, 3, 2, 1, 4, 1, 2, 3, 0, 2};
mEntries   = {1, -11, 3.1415, 0.5, 10, 3, 7, 2, 4, -5};
```

`http://www.math.tuwien.ac.at/~melenk/teach/computernumerik_WS1920` contains pieces of code for reading and writing matrices in CSC format as well as example matrices. For symmetric matrices, it is common to store only the lower part.

1. Implement the matrix–vector–multiplikation for Matrizen stored in the CSC format. Make use of this format. That is, the number of operations should be proportional to the number of non-zero entries of the matrix and be independent of the size of the matrix.

Test and document your code.

2. The remainder of the project realizes the Cholesky factorization of symmetric matrices. For efficiency reasons, only the lower part of the matrix is assumed explicitly given.

a) Modify your code of Problem 1 such that it realizes the matrix-vector multiplication for symmetric matrices, where only the lower part is given.

b) Let A be a matrix. The Cholesky factor L of A is a lower triangular matrix such that $A = LL^\top$. If L is available, then solving $Ax = y$ can be reduced to solving the equations (forward and back substitution):

$$Lz = y \qquad \text{und } L^T x = z \tag{2}$$

Implement the forward and back substitutions for lower triangular matrices in CSC format.

Test and document your code. Check that the run-time of your code does not depend on the size of the matrix but only on the number of non-zero entries.

3. The standard storage scheme for symmetric matrices stores all entries of the lower part in a single long vector. Write a conversion routine from the CSC format to the standard format. Test and document your code.

Sparse matrices do not normally have sparse Cholesky factors. In class, we have provided the following fact: Let A be an SPD–matrix. Let the lower triangular matrix L be its Cholesky factor. Let

$$J_i(A) := \min\{j : A_{ij} \neq 0\} \tag{3}$$

be the first column index of row i such that $A_{i,J_i(A)} \neq 0$. Then:

$$L_{ij} = 0 \quad \text{for } j < J_i(A). \tag{4}$$

Hence, L preserves leading zeros in each row. The lecture introduced the Cholesky factorization as (LECT)

```
for $k = 1, \dots, n$
    $L_{kk} = (A_{kk} - \sum_{j=1}^{k-1} L_{kj}^2)^{1/2}$
    for $i = k+1, \dots, n$
        $L_{ik} = (A_{ik} - \sum_{j=1}^{k-1} L_{ij} L_{kj})/L_{kk}$
    end
end
```

It can be shown (the proof is not too complicated) that this is equivalent to the following variant (PROJ), which only acts on columns of A and overwrites A with its Cholesky factor:

```
for $k = 1, \dots, n$
    $A(k,k) = \sqrt{A(k,k)}$
    $A(k+1:n, k) = A(k+1:n, k)/A(k,k)$
    for $j = k+1, \dots, n$
        $A(j:n, j) = A(j:n, j) - A(j:n, k)A(j,k)$
    end
end
```

4. Select one of the above variants and implement the Cholesky factorization for full matrices (but with only the lower part being stored) in the standard column format. Document and test your code.

5. Implement the Cholesky factorization for matrices in CSC format using the variant (PROJ). Proceed in the following steps:

a) Implement a function that, given a matrix `A` in CSC format produces an "empty" Cholesky factor `L` with a sparsity pattern given by (3), (4). The entries of L are initialized with zero.

b) Copy `A` to `L`.

c) Compute the entries of `L` using variant (PROJ).

Document your code and your tests.

6. Download the example matrices or generate them using the MATLAB-Code `generate_matrices.m`. The goal is to check the complexity of the implementation. To that end, we test with two types of matrices:

- `tridiagonal_n.mtx`, where `n` is the dimension of the matrix. These are tridiagonal matrices.
- `poisson_n.mtx`, where `n` is the dimension. These are matrices that arise from the discretization of a Poisson problem in 2D. These are banded matrix where the band width is essentially $\sqrt{n}$.

These matrices can be read into MATLAB with `mmread.m`. The command `spy` visualizes the sparsity pattern. The matrix and Cholesky factors may be visualized in this way:

```
> A = mmread("poisson_256.mtx");
> spy(A);
```

Write programs (for full and sparse matrix arithmetic) that read in matrices, compute Cholesky factors and solve a linear system. Measure the time for the factorization and the solution. Which complexity do you expect for the full matrices and the sparse matrices? Document your results using appropriate MATLAB plots.