TP 2 : triangular matrices

Exercise 1 : templated sparse matrices

Change the code of your `MapMatrix` class so that it can model possibly non-real valued sparse matrices. You shall proceed by adding a template parameter `value_t` that models the type of the values stored in `data`. Modify both data members and member functions/operators accordingly.

Exercise 2 : reading/writing sparse matrices

Question 2.1 Write a function `MapMatrix<double> LoadMapMatrix(const std::string& filename)` that reads a file named `filename` containing data describing a real valued sparse matrix in coordinate format, assembles the corresponding `MapMatrix<double>` and returns it. We shall assume that the format of the file `filename` is the following

```
nr   nc
j1   k1   v1
j2   k2   v2
⋮    ⋮    ⋮
```

where `nr,nc` refer to the number of rows and columns of the matrix, and the triples j_l, k_l, v_l are the triples `(row position, column position, value)` involved in the coordinate format. Test the function `LoadMapMatrix` on toy examples of your invention.

Question 2.2 Use the routine `LoadMapMatrix` to load examples matrices from the archive `tp2_matrices.zip` available on the moodle page and use them to plot the computational time vs size of the matrices for the sparse matrix-matrix multiplication coded in exercise 3 of tutorial sheet 1. The plot can be achivied with e.g. `gnuplot` or `matplotlib`.

Question 2.3 Write a function `void Write(const std::string& filename, const MapMatrix<double>& m)` that writes an already existing matrix into a file named `filename` in the same format as the one described above. Again test this on toy examples.

Exercise 3 : triangular matrices

Write a class called `UpperTriangularMatrix` that models real valued upper triangular matrices. In this data structure, only the non-zero elements are stored in a single table named `data`. You shall choose by yourself how these nonzero entries are arranged (i.e. in which order) in `data`. This class will comply with the following specifications.

Data members :

- `int nr` : the number of rows
- `int nc` : the number of columns
- `std::vector<double> data` a vector storing the **non-zero** entries of the matrix.

Functions/member functions :

- a constructor `UpperTriangularMatrix(const int&, const int&)` intializing `nr,nc` with the input parameters and sizing `data` accordingly, setting the coefficients to 0.
- a copy constructor
- a copy assignment operator `=`
- member function `void push_back(const int& j, const int& k, const double& v)` that adds the value `v` in row-column position `(j,k)`. This routine shall check that the position `(j,k)` is located in the upper triangular part and returns an error if not.
- operator `( , )` that takes a pair of integers (j, k) and returns the (r-valued) coefficient located at the j-th row and k-th column.
- output stream operator $\ll$
- member-function `std::vector<double> Solve(const std::vector<double>& b)` that solves the corresponding upper triangular linear system with `b` as right-hand side, and returning the solution.
- friend function `int NbRow(const DenseMatrix&)` that returns the number of rows
- friend function `int NbCol(const DenseMatrix&)` that returns the number of columns