Exercises for the Lecture Series

"Object-Oriented Programming for Scientific Computing"

Dr. Ole Klein
ole.klein@iwr.uni-heidelberg.de

(Bonus sheet)

Note: The following exercises are meant as preparation for the upcoming exam, i.e. additional tasks to practice programming, and as a source of bonus points. They don't count towards the limit needed for admission to the exam.

EXERCISE 1 STL ALGORITHMS: VECTOR NORM

Let $x \in \mathbb{R}^n$ be a given vector. We know different vector norms, e.g.:

$$||x||_2 = \sqrt{x \cdot x}, \qquad ||x||_1 = \sum_{i=0}^{n-1} |x_i|, \qquad ||x||_\infty = \max_{i=0}^{n-1} |x_i|$$

Assume the vector x is saved as an STL sequence container.

1. Implement the above vector norms using STL algorithms.
   - Which algorithms can be used here in order to save programming effort?
   - Would you change your mind of which algorithm to use, if the container may be modified? If so, how?
2. Rewrite the algorithms so that they share as much code as possible.
   - Identify the similarities of the various norm calculations.
   - Design an interface to take advantage of these similarities.
   - What kind of interchangeability would you choose for the parts that differ and why? (Template specialization, static polymorphism, dynamic polymorphism)

Hand in the final result of this iterative process and your answers to the questions.

6 Points

EXERCISE 2 ABSTRACT MATRIX ALGORITHMS: MATRIX NORM

In this exercise we program an application for the abstract matrix interface implemented on the previous exercise sheet. The resulting abstract algorithm will then also work with matrices based on other data structures. Program a function

```
template <class M>
double frobeniusnorm(const M& matrix);
```

which calculates the Frobenius norm of the matrix, and write a test program for your implementation. The Frobenius norm of a Matrix A is defined as

$$\|A\|_\mathrm{F} = \sqrt{\sum_{i,j} |a_{ij}|^2},$$

where a_{ij} refers to the entries of the matrix. *6 Points*

EXERCISE 3 ABSTRACT MATRIX ALGORITHMS: GAUSS-SEIDEL

Based on the abstract matrix interface, we now want to program an algorithm for the iterative solution of systems of equations.

The Gauss-Seidel algorithm provides an approximate solution to linear equation systems with matrices whose spectral radius is smaller than 1. Given such an $n \times n$ Matrix A and a righthand side b, we seek an approximate solution of the vector x, so that

$$Ax = b$$

holds. This corresponds to the n equations of a linear equation system.

To solve the system, the k-th equation is solved for x_k. The $(m+1)$-th iteration of the Gauss-Seidel algorithm then consists of solving n equations of the following form:

$$x_k^{(m+1)} = \frac{1}{a_{kk}} \left(b_k - \sum_{i=1}^{k-1} a_{ki} \cdot x_i^{(m+1)} - \sum_{i=k+1}^{n} a_{ki} \cdot x_i^{(m)} \right)$$

Implement the Gauss-Seidel algorithm based on the abstract iterator interface. Note that each of the equations that have to be solved corresponds to a row of the matrix, and take this into account when iterating over the contents of the matrix. Write your program so that a fixed number of iterations will be performed. The interface should look like this:

```
template<class M, class T>
void gaussSeidel (const M& A, const std::vector<T>& b,
                  std::vector<T>& x, int maxIter);
```

Note: Test your program using the following example:

$$\begin{pmatrix} 2 & 1 & 0 \\ 1 & 3 & 2 \\ 0 & 1 & -4 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ -0.5 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

If you did not implement the abstract matrix interface on the previous exercise sheet, you may either implement it for this exercise or directly access the internal container and describe in words what would be different when using iterators. You can at most reach 4 points when not using the abstract matrix interface.

8 Points