Exercises for the Lecture Series

"Object-Oriented Programming for Scientific Computing"

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

To be handed in on 30. 06. 2015 before the lecture

Exercise 1 Type Traits for Matrices

In a previous exercise we implemented the Frobenius norm for matrices. There the return type was always `double`.

We now want to extend the calculation of the norm in order to make it work for very different types of numbers. We intend to realize this using the concept of type traits. There are a few things to note:

- The return type depends primarily on the number type of the matrix.
- The calculation of the norm should work as well for `complex<T>` and `Rational` (see first exercise sheet).
- The return type is not always the number type of the matrix, e.g. for complex numbers it is the underlying real type.
- Changes may also be necessary for the calculation of the squared absolute value of the entries.

Answer the questions and program:

1. What kind of variability is necessary in the algorithm?
2. Which traits would you want to introduce because of this?
3. Design test problems for matrices with integer entries, real entries, rational entries and complex entries (3×3 matrices are sufficient).
4. Implement the corresponding traits classes and adapt the function `frobeniusnorm`. Write specializations of the traits, so that the calculation of the norm works for `double`, `float`, `int`, `Rational` and `complex<T>`.
5. Test your program on the basis of the test problems for the types `double`, `float`, `int`, `Rational`, `complex<double>`, `complex<float>`, `complex<int>` and `complex<Rational>`.

Base your algorithm on the one you handed in for the previous exercise if applicable. If the abstract matrix interface using iterators isn't available to you, you may use direct access to the entries of the matrices.

10 Points

Exercise 2 Matrix Norm via Policy

Depending on the application, you might want to choose between different matrix norms.

Let $A \in \mathbb{R}^{n \times n}$, and consider the following matrix norms:

- Frobenius norm

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

- Row sum norm

$$\|A\|_\infty = \max_i \sum_j |a_{ij}|,$$

- Total norm

$$\|A\|_G = n \cdot \max_{i,j} |a_{ij}|$$

Write a function

```
template<class M, class P=FrobeniusNorm>
MatrixTraits<M>::realType matrixNorm (const M& A);
```

which is parameterized with a policy class `P` and thus capable to calculate all three norms.

One can formulate these norms in a common way so that one iterates over all rows and for each row iterates over all of its entries. For each entry an operation `EntryOp` is executed and afterwards for the whole row an operation `RowOp`.
Pseudo code:

```
matrixNorm<P> (Matrix A) {
    result = 0;
    foreach (row r of A) {
        row_result = 0;
        foreach (entry x of r) {
            row_result = P::EntryOp(x, row_result);
        }
        result = P::RowOp(row_result, result);
    }
    return result;
}
```

The policy class `P` should supply all necessary operators and information that are needed to compute the norm.

1. What information / operators must be provided by the policy class for the three norms defined above?
2. Which template parameters (if any) do the policy classes need?
3. Implement the generic function `matrixNorm`.
4. Implement the policy classes for all three norms.
5. Test the classes with the following example:

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 3 & 2 \\ 0 & 1 & -4 \end{pmatrix}, \qquad \|A\|_F = 6, \qquad \|A\|_\infty = 6, \qquad \|A\|_G = 12$$

6. The algorithm above may be inefficient, e.g. in the case of the Frobenius norm. What change to the algorithm (and the policies) would allow more flexibility and a faster implementation? Please answer in words and pseudo code as above, an actual implementation is not required.

Try to reuse as much code as possible from previous exercises, so you don't need to reimplement everything.
10 Points