SCHOOL OF SOFTWARE ENGINEERING OF USTC

FIRST SEMESTER, 2012–2013
Campus: Hefei

DATA-MINING

Basic building blocks

(Time allowed: TWO hours)

The objective of this experiment class is to build some basic classes that you will reuse for further experiments. In previous years, students showed some troubles to properly organize their code : no structure, no use of object-oriented programming, same code copy-pasted several times, hard-coded limits . . . This work should be quite helpful,to have clean, reusable code, helping you to complete your future assignements faster. It is also some easy gained points, and a good time to sharpen your coding skills.

You can do your work in *C++* or *Java*. You can also use *C*, but it will be probably more work, maybe more than you can handle. *Python*, *R* or *Matlab* are **NOT** allowed. You can work on either Linux or Windows, and whatever compiler you wish, as long as your code is portable. I expect clean, readable code : each class in separate files, for example.

You have to work alone, one student per computer, this is non-negociable. It is to help you to be focused and practice by yourself. If you are stuck after trying various ways, you can call for help. You might have to patient while the teacher is busy helping someone else.

1. Implement the *Kahan summation* algorithm as a *KahanSum* class. The*KahanSum* class should have the following methods.

- *void reset()*, set the sum equal to 0.
- *void add(double value)*, add one value to the sum.
- *double get_sum()*, return the current value of the sum.

The constructor should set the sum to 0. The added values should be of *double* type, to have the greatest precision possible. Write a simple test to ensure that your implementation work properly.

2. Implement the *median* of a list algorithm as a *Median* class. The*Median* class should have the following methods.

- *void reset()*
- *void add(double value)*, add one value.
- *double get_median()*, return the current value of the median of all the added values.

The added values should be of *double* type. Write a simple test to ensure that your implementation work properly.

3. Implement a *Vector* class, that represents a vector of size N, where N is given as a parameter of the constructor. By *Vector*, I mean a mathematical vector, not a C++ *vector* container or a Java *Vector* container.

The class should have the following methods.

CONTINUED

- *void fill(double value)*, set all values of the vector to the same value.
- *double get(int i)*, returns the value at index i
- *void set(int i, double value)*, returns the value at index i ($A[i] = x$)
- *void add(const Vector& v)*, add to the current vector the vector v ($A = A + V$)
- *void sub(const Vector& v)*, substract to the current vector the vector v ($A = A - V$)
- *void mul(double k)*, multiply the vector by k ($A = kA$)
- *void inc_mul(const Vector& v, double k)*, add to the current vector the vector kV ($A = A + kV$)
- *double dot(const Vector& v)*, compute the dot product of the vector V ($A.V$). Note that a dot product is a sum, and that you just wrote a high precision sum algorithm at the previous question.
- *double square_norm()*, compute the squared norm of the vector ($||A||^2$)
- *double norm()*, compute the norm of the vector ($||A||$)
- *double sum()*, compute the sum of all the values of the vector.
- *double min()*, compute the minimum of all the values of the vector.
- *double max()*, compute the maximum of all the values of the vector.

Test properly your class. You might consider implementing a way to print a vector, to help you for the testing.

4. Implement a function *write_data*, that write to a file the values from a list of *Vector* instances. The list can have any size.

5. Implement a function *read_data*, that reads a file containing columns of numbers, and returns a list of *Vector* instances. This function should able to read what *write_data* wrote. The list can have any size.

6. Implement a function *vector_mean*, that gives the mean of a list of vectors. The list can have any size.

7. Implement a function *vector_median*, that gives the median of a list of vectors. The list can have any size.