STAT 598W Homework 4

Due by April 6, 2016

1 Matrix Class

Complete the following matrix class. There are several member functions in the derived class that are optional, though I suggest that you give them a try. All the required functions can be based on Gaussian Elimination and you do not need to consider numerical errors. Simply run the most basic algorithm is enough for this assignment.

```
template<class T, int m, int n>
class Matrix {
        vector<vector<T>> elements;
        int nrow;
        int ncol;
public:
        Matrix();
        Matrix(Matrix<T, m, n>&);

        T& operator() (int, int); //get elements in the matrix
        vector<T> row(int); //get ith row as a vector
        vector<T> col(int); //get jth colume as a vector

        //need to overload +, - and *
        //For *, consider both scalar and matrix multiplication

        Matrix<T, n, m> transpose(); //return the transpose of the matrix
        Matrix<T, m, n> GaussElimination(); //return the Gauss elimination
        int rank(); //return the rank of the matrix
        vector<vector<T>> Solve(const vector<T>&);
        //solve a linear equation Ax = b,
        //where A is the matrix and b is the input parameter.
        //The return value is a base of all solutions.

};

template<class T, int n>
class SquareMatrix :public Matrix<T, n, n> {
public:
        SquareMatrix();
        SquareMatrix(SquareMatrix<T, n>&);

        bool isNonSingular(); //return if the matrix is singular
        SquareMatrix<T, n> inverse(); //return the inverse of the matrix
```

```
        SquareMatrix<T, n> pow(int n); //calculate A^n

        vector<T> Eigenvalue();
        //(optional) return all the eigenvalues, in descending order
        vector<vector<T>> Eigenvector(T);
        //(optional) return the basis of eigenvectors of a certain eigenvalue

        SquareMatrix<T, n> cholesky(); //(optional) return the Cholesky decomposition

        SquareMatrix<T, n> Q(); //(optional) return the Q of QR decomposition
        SquareMatrix<T, n> R(); //(optional) return the R of QR decomposition
};
```

2 Stack

Complete the following stack class if you did not complete it in Homework 3.

```
template<class T>
struct stack_node {
        stack_node* next;
        T val;
        stack_node(stack_node* _next) : next(_next) {}
};
template<class T>
class stack {
        typedef stack_node<T> node;
        typedef stack_node<T>* iterator;
        typedef stack_node<T>& reference;
        //Some type define.

        iterator first;
        iterator last;
        size_t length;
        //Three variables we keep.
public:
        stack();
        //Constructor with no parameter.
        stack(T);
        stack(node);
        //Constructor with one node as parameter.
        stack(stack &);
        //Copy constructor.
        ~stack();
        //Destructor.

        size_t size();
        bool empty();
        //Return the size of the list  and whether it is empty.

        T& top();
        void push(const T&);
        void pop();
```

```
        //Get the top element, push new element to the top
        //and remove the top element.
};
```

3 Queue

Complete the following queue class if you did not complete it in Homework 3.

```
template<class T>
struct queue_node {
        queue_node* next;
        T val;
        queue_node(queue_node* _next) : next(_next) {}
};
template<class T>
class queue {
        typedef queue_node<T> node;
        typedef queue_node<T>* iterator;
        typedef queue_node<T>& reference;
        //Some type define.

        iterator first;
        iterator last;
        size_t length;
        //Three variables we keep.
public:
        queue();
        //Constructor with no parameter.
        queue(T);
        queue(node);
        //Constructor with one node as parameter.
        queue(queue &);
        //Copy constructor.
        ~queue();
        //Destructor.

        size_t size();
        bool empty();
        //Return the size of the list  and whether it is empty.

        T& front();
        T& back();
        //Get the next and last element.

        void push(const T&);
        void pop();
        //Push new element to the queue.
        //Remove the next element.
};
```

4 Simple Portfolio

In this exercise, we try to construct a class of a simple dynamic portfolio, which consists of stocks and options. Design of the portfolio is up to you, but you should include the following features.

a. A class for stock. The class should include: (1) a static variable for current time; (2) get the current time, as an integer that starts from zero; (3) update the current time by one unit; (4) get the current stock price; (5) update the current stock price to the next time according to the Black-Scholes model: $S_{t+1} = S_t \exp(\mu - \frac{1}{2}\sigma^2 + \sigma Z)$. Of course, μ and σ should be variables of the class.

b. The pool of stocks, which includes all the available stocks in the market.

c. A class for option. The class should have at least the following features: (1) a static variable for current time and interest rate; (2) variables for maturity time, strike price and whether the option is call or put; (3) a pointer to a stock object as the underlying asset; (4) a public method that returns the current option price, keeping in mind that after maturity, the price of the option simply changes by the interest rate; (5) a public method that updates the option price to the current time.

d. A class for portfolio. The class should contain several stocks and several options. Some important features should include: (1) for each type of asset, either option or stock, keep track of the number of shares held in the portfolio, keeping in mind that we might short sell; (2) keep track of the total value of the portfolio; (3) change position (by overloading operator + and -) for a certain asset, keeping in mind that this may include going long or going short, and also that it is possible to either add a new asset, or add to an existing asset, or remove completely an existing asset; (4) combine two portfolio (by overloading operator +); (5) update the time and meanwhile update the all the prices in the portfolio.