C++ Implementation of Haar systems for Black-Scholes model

Sebastian Ferrando and Jihong Cai,
Ryerson University.

November 3, 2004

Abstract

This is a financial application to assess the Haar Hedging(HH) and Black & Schole(BS) algorithm. The application simulates stock paths and then implements the algorithm for HH and BS, in addition it implements the value of the Holding portfolio along a sampled path. The application uses Object-Oriented design and programming, in particular, the stock, the path and the portfolio are all abstracted to classes. The assessing result is writen to some text files and plotted out using the g2 package.

Contents

1 Requirement Documentation

1.1 introduction and goal of the system

Options can be categorized into two basic types:calls and puts. A call option gives the holder the right to buy an asset by a certain date for a certain price. A put option gives the holder the right to sell an asset by a certain date for a certain price. The certain price in the option is called strike price, the certain date is called expiration date. An asset might be a kind of stock, currency or something else.

We only consider stock as asset in our system.Furthermore, we only compute European Call and European Put option. Both option are simple and easy to derive formular and algorithm. However, we design the application so that we can expand the kind of options in the future easily.

Hedging means to reduce the risk with forward contracts and options. We only consider hedging techniques using stock options.

A stock price depends on many random factors, such as no-risk interest rate, market volatility and time. Therefore, a stock path is random with certain regularity. A stock price has a lognormial distribution at an arbitrary time.

The goal of this system is to compute the payoff for different hedging techniques: Haar Hedging and Delta Hedging(Black & Schole Portfolio). The system assesses these techniques on an investment and output the visual result.

1.2 functional requirement

The system accepts input from 2 files. The file "parameter.txt" includes the following data:

1. Stock information: current price, market volatility and non-risk bank insterest. Stock information in clude interest only for simplicating the system.

2. Hedging information

   - j_H, R or threshold: For Haar Hedging only.
   - N and times: General hedging times along a path.

- N_H and Htimes: times for Haar Hedging, and it is a sub set of the general time. IN addition, its first and last time is same as the general time.
- N_BS and BStimes: times for Black and Schole Hedging, and it is a subset of the general time. In addition, its first and last time is same as the general time.
- M,seed: M is the number of paths to simulate. If M is equal to 1, it means we want to get the pathwise result. The path might be choose by defining different seed number.

3. CallNum,PutNum: define how many call and option are in the porfolio, and both must be consistent with the following option information.

4. Option information: This is a multilines input, and one line represent one option. The total line number is equal to the addition of callNUm and putNum above. Each line contains 3 numbers, which are amount, strike prices and expiration date. The first CallNum lines are for Call options. and the rests are for Put options.

The file "outputops.txt" defines outputs combinations that user wants. It includes the following data:

1. Hedging method: This information defines the hedging method to assess, 1 means only assess HH, -1 means assess BSH, and 0 means assess both hedging method.

2. ErrorOption: This difines the option to get sample values or error information.If it is 0, the payoff or values of Holding portfolio, HH payoff and/or BSP payoff will all be be calculated and the result is recorded by a ps file and a text file. Otherwise, the difference between HH payoff and holding portfolio and the difference between BSP payoff and the value of holding porfolio will be output.

3. assessTime: The result for different algorithms can be assessed at different asses time. Assess time might be T, which is the last element in the general time vector. It might be another time, but this assess time should be an element of threee time vectors.

The system simulates a stock price movement during a period defined by general times. A general path is generated during each simulation. The subpath for Haar Hedging and Black and schole Hedging are retrieved from the general path using Htimes and BStimes.

Each option has its own expiration time. This expiration time should belong to the intersection set of Htimes and BStimes. We only do Haar hedging along the subpath for Haar Hedging, wich means we Hedge at the time defined in Htimes from 0 to option's expiration time. The same rule applies to Black and Schole Hedging.

Holding portfolio will be computed along the general path. At least one option will expire at the end of the general time, otherwise the time after all options have expired is meaningless. Finally, we achieve three different payoffs from three different ways, holding portfolio, Haar Hedging and Black and Schole Hedging.

The simulation is done M times, so we will get 3 same-size vectors that contains M values of payoff computed by three different methods. The program will sort the vectors based on the simulation final stock price and plot out the three vectors.

There are some properties needed to be considered;

1. Time issues:
   - The vector times contains all the time(date) in a path, including the t0(0) and t_N(T). It contains N+1 elements.
   - The vector BStimes contains all the time(date) in a path for black & schole hedging, including the t0(=0), and the expiration time of an option. The vector has N_BS number of elements.
   - The vector HHtimes contains all the time(date) in a path for haar hedging, including the t0(=0), and the expiration time of an option. The vector has N_H number of elements.
   - At least one of the options has expiration time at the last date, means the last element in times vector.
   - At any time, there might be one call option, or one put option, or one call and one put options expire at a given time, but there weren't be 2 calls or 2 puts expiration at the same time.
2. Path issues:
   - The path simulated by a stock is the most general path. It contains N+1 spots from 0 to T(t_0 to t_T). The general path is used for computing value of holding portfolio.
   - The path for hedging is part of the general path. It can be subtracted from the general path using adequate algorithm described in section 2 on page 8. There are 2 kinds of subpath: Haar Hedging and Black and Schole Hedging path.
   - The hedging subpath obtained above is for general hedging path. An option's heding path actually is part of the general hedging path, but it is not computed explicitly. An individual path ends at its expiration time which might be less than T (final time).

2 Class design and Structure

2.1 General

1. Classes and class diagram

When some operations are strongly tight to an object, a class is created. There are 6 classes defined in the system, and they are named as Path, Stock, Option, EuCall, EuPut and Portfolio.

A path class has a vector of time, prices and jump numbers. Its operation includes generating a subpath and computing the index of a given time in the path.

A stock class has the ability to simulate a random path at each time in the time vector which is passed as parameter. A stock class has initial stock price,volatility, and the non-risk bank interest.

An option class has the ability to compute its expiration value and its value at any given time. Europan Call and Put otion class are the subclass of option class. As a subclass, both European Call and Put class inherite all the attributes that an general option has. They implement some virtual functions defined in parent class.

A portfolio object, which is consisted by European Call and Put options, has the ability to compute Haar Hedging payoffs, Black & Schole payoffs and holding portfolio.

A Haar class is able to run some haar algorithms, such as analysis, compression and synthesis. It contains a vector of items for haar operations and two integer J and j to identify the size of haar matrix and the current position of haar vector respectively. When a portfolio class is computing Haar Hedging payoff, it needs to construct a haar object to finish the above haar algorithms.

HHAssess class is used to read input from two input files. Then it constructs objects of the above classes to run hedging operations. Finally it writes output on text file and ps file.

The relationships between classes in this system is showed in Figure 1 below. The haar class, HHAssess class and some other libraries are not shown on the figure. Further considerations about designing of classes and their relationship will be discussed in the next session.

2. Libraries

   Some functionalities are very general regarding an finacial application, such as getIndex, cumulative. They should belong to an algorithm group. Like Math class in Java or C++, I group these special functions into a file called falgorithm.h, and it might be considerd as a tool.

   The similar idea is to realize the plotting functions; I just group the functions into plotting.h file. Plotting.h is an easy-to-use interface to draw lines and curves for applications with a lot of data to handle.

2.2 Class Definition

1. Stock:
   A stock class contains stock's current value, which is the price before

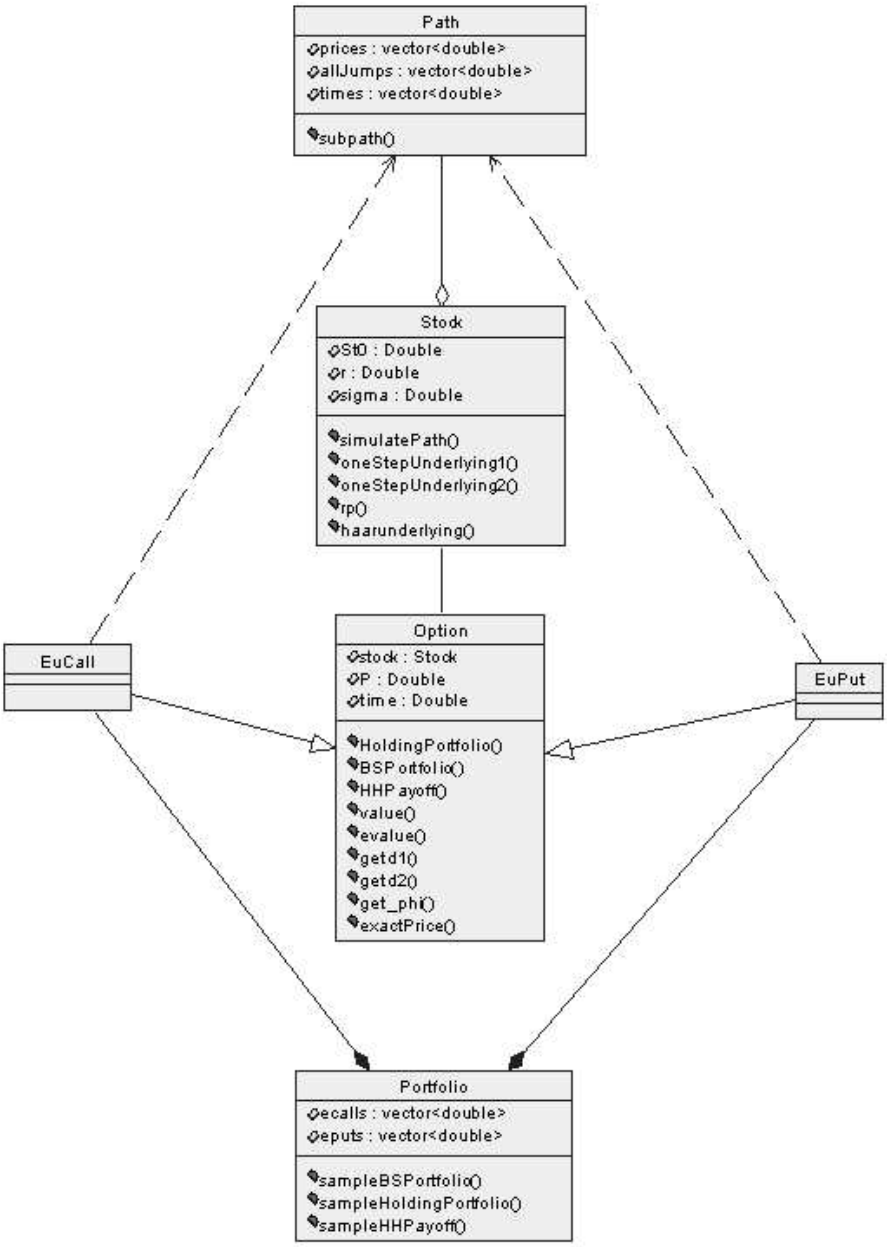

Figure 1: Class Diagram

we run any hedging algorithm. It also contains variable σ(sigma), which means the volatility of the stock. σ is assume to be a constant for one kind of stock.

Note:
Interest r is also included in the stock class due to simplification. Actually, it might be part of a bank object.

The definition of a stock object is shown below:

Listing 1: ../source/stock.h [Line 22 to 38]

```
class Stock{
 public:
  double St0;     //current price
  double r;       //interest
  double sigma;   //variarity

  //constructors
  Stock(double theSt0,double theR,double theSigma);
  Stock();

  Path simulatePath(vector<double> times);
  Path simulatePath(vector<double> times,int seed);
  vector<double> oneStepUnderlying1(int J_H,double currentPrice,double deltaT);
  vector<double> oneStepUnderlying2(int J_H,double currentPrice,double deltaT);
  vector<double> haarUnderlying(int J_H,vector<double> times);
  vector<double> rp(int J_H,double deltaT);
};
```

The operations that a stock can have include:

- Simulate a path given a vector of time
- Simulate a one step stock haar underlying given the current price of stock, the time period, and the partition size. The vector of underlying represent the posible stock price after a time period based on stock price's lognormial distribution at a fixed time. We have two ways of imlementation, one is based on lognormal distribution directly, another one is base on the abstract concept rp, which is described below. Currently, the system used the first way to ompute Haar Hedging payoff.
- Compute a rp vector reference to "HH Handout III". This vector doesn't use information of the current stock price, and it is like an abstrack tion of lognormal distribution. The vector is computed based on partition size and time period.
- Simulate a multi step stock haar underlying given the partition size and multi time period used the rp vector obtain above.

2. Path:

A path class is used to represent a simulated path for a stock, so it should have a price at a corresponding time. However, the price was simulated by generated a random number between 0 and 1, then an inverse cumulative number is calculated based on Normal Distribution. This inverse cumulative number is called jump numbers. Therefore, a path class contains 3 member variables, a vector of times, a vector of prices and a vector of all jumps.

A path is independent to a stock, but it is part of a stock. We only consider to simulate paths for a stock, so it's not neccessary and possible to keep these simulated paths in a stock object. Instead, we might define a real path element for a stock, but it is beyond this application. In addition, we will never record a real path for study. The information of a path is used to compute payoff of a stock, then this path will be discarded and another path will be simulated.

The definition of a path object is shown below:

Listing 2: ../source/stock.h [Line 10 to 18]

```
class Path{
 public:
  vector<double> prices;
  vector<double> allJumps;
  vector<double> times;
  Path(vector<double> thePrices,vector<double> theJumps,  vector<double> times);
  Path subPath(  vector<double> subtimes );
  int timeIndex(double time);
};
```

The operation that a path class has includes:

- subpath
  A general path is used to simulate the real path of a stock in a given time period, and the time period is divided in N intervals. However, either HH or BSP won't perform an operation at every time spot. This create the necessity to generate a subpath for HH or BSP. Time spots and corresponding prices are copied one by one the sub path. If the subpath skip several spots in the general path, the allJump values at these spots are summed up and divided by square root of the number of skippedspots. The value obtain in this way is the allJjump number for the subpath at this time spot.

  There are 4 kinds of paths used in this system. A general path is simulated by a stock. The subpath for computing the holding portfolio is same as the general path, i.e. we will compute holding portfolio at every time point. The path used for HH or BSP are all subpath of the general path, and these two paths are independent of

each other, and they might have a common set and a difference set. The subpaths at least contain the first and the last element of the general path.

- timeindex
  The payoff of HH or BSP might be checked at any time before the expiration time. The time to check the payoffs is called assesstime. Since we need to compare the hedging payof with holding portfolio, and the path for hedging might be a subpath of the path for holding portfolio. We need to know the relative position (index) in the time vector of the subpath given the assesstime.

3. Haar:
   A Haar class contains a vector of items and 2 integers j and J. J is used to identify the size of the Haar Matrix, which means the number of colums and rows of the matrix is 2^J and (J+1) respectively. The vector in a Haar object stores the element in one row, and j which identify in which row the vector contain the elements. So, the vector has 2^J elements and the domain for j is from 0 to J. See the definition below:

Listing 3: ../source/haar.h [Line 11 to 32]

```
class Haar
{
 public:
  vector<double> haarItems;//The vector for haar operation that contain 2^J elements
  int j,J;

  Haar(vector<double> theItems,int the_J, int the_j);
  Haar();

  Haar haarAnalysis(int numberOfCalls);
  Haar haarSynthesis(int numberOfCalls);
  Haar thresholdCompression(double threshold, int &Rt);
  Haar RCompression(int R);
  Haar SORCompression(int R);

  void printHaarVector();
  double energy();

 private:
  Haar oneStepHaarAnalysis();
  Haar oneStepHaarSynthesis();
};
```

The operations a Haar object has includes analysis, synthesis and compression.

The compression criteria might be the threshold for energy or R for the number of nonzero elements in the compressed haar vector. In our system,

we analysize the haar vector J times, compresse it and then synthetize it back to original position. For compression, we have 2 options: one is keeping the first element S0 unchanged, and another one is modifying S0 to 0. In our system, we will always set S0 = 0. The detailed explanation will be found in HH payoff algorithm in the next session.

4. Option:
   Option is a super abstract class, which is supposed to have all common properties of heterogeneous options. It contains four members: a stock, an expiration time, an expiration price (strike price) and an amount.

Listing 4: ../source/option.h [Line 13 to 34]

```
class Option{
 public:
  Stock stock;
  double etime; //expiration time
  double P;     //expiration (strike) price
  double amount;

  Option(Stock theStock,double theTime,double strikePrice,double amount);
  Option();

  virtual double exactPrice() {};
  virtual double value(double stockPrice,double time) {};
  virtual double evalue(double stockPrice) {};
  virtual double get_phi(double d1) {};
  virtual vector<double> BSPortfolio(Path path) {};
  virtual vector<double> HHPayoff(int j_H,double stockPrice,
                                  double time,double deltaT){};

  double get_d1(double stockPrice,double time);
  double get_d2(double d1,double time);
  vector<double> holdingPortfolio(Path path);
};
```

There are 3 public fuctions that can be inherited by subclass of option.

- get_d1 and get_d2: Both are basic functions used to compute holding value and BS portfolio. For a given option, the value of d1 and d2 depends on the current stock price and current time.
- holdingPortfolio: It returns the vector of option values along the path.

There are six virtual member functions. They are defined inside option class. These funcions must be implemented in subclass EuCall and EuPut.

- exact price: Compute the exact price for the option at final time in theory.

- value: Return the option's value at a given time.
- evalue: Compute the option's value at expiration time.
- get_phi: compute the stock investment for a portfolio. And it is based on d1's value. It has different formulars for Call and Put options.
- HHPayoff: compute the HH payoff at the current time for an option, the payoff depends on j_H, which identify how complexity we are doing the haar algorithm, and it is corresponding to the J in Haar class. The payoff also depends on the current stock price and the current time. And it also depends on the time interval for Haar Hedging. It return a vector of size 2^{j-H} at current time.
- BSPortfolio: Compute the BS portfolio for an option, it only depends on the path. It is different from HHPayoff from that it will return a vector along the path, and HHPayoff returns the vector for the current time. This function can't be used in portfolio class to compute BS payoff, since the value of ζ and ϕ are computed step by step. It considers the whole effect of all options, and we can't simply add the BS payoff for individual options together to get a total payoff for a portfolio.

5. EuCall and EuPut:
   Subclass of Option. They inherit four member variables from parent class and three public functions. They need to implement the six virtual functions.

Listing 5: ../source/option.h [Line 38 to 50]

```
class EuCall:public Option{
 public:
  EuCall(Stock theStock,double theTime,double strikePrice,double amount);
  EuCall();

  double exactPrice();
  double value(double stockPrice,double time);
  double evalue(double stockPrice);
  double get_phi(double d1);
  vector<double> BSPortfolio(Path path);
  vector<double> HHPayoff(int j_H,double stockPrice,double time,double deltaT);
  vector<double> HHPayoff(int j_H,vector<double> times);
};
```

Listing 6: ../source/option.h [Line 54 to 64]

```
class EuPut:public Option{
 public:
  EuPut(Stock theStock,double theTime,double strikePrice,double amount);
```

Listing 6: ../source/option.h [Line 54 to 64] (continued)

```
  EuPut();
  double exactPrice();
  double value(double stockPrice,double time);
  double evalue(double stockPrice);
  double get_phi(double d1);
  vector<double> BSPortfolio(Path path);
  vector<double> HHPayoff(int j_H,double stockPrice,double time,double deltaT);
};
```

6. Portfolio
   In this system, we only consider European call and put, so, a portfolio class contains a vector of European calls and a vetor of European puts.

Listing 7: ../source/portfolio.h [Line 12 to 23]

```
class Portfolio{
 public:
  vector<EuCall> ecs;
  vector<EuPut> eps;

  Portfolio(vector<EuCall> theEcs,vector<EuPut> theEps);
  Portfolio();

  vector<double> sampleBSPortfolio(Path path);
  vector<double> sampleHoldingPortfolio(Path path);
  vector<double> sampleHHPayoff(Path path,int j_H,int R);
};
```

The portfolio class has the ability to compute Black and Schole, Haar Hedging and Holding payoff along a simulated path. The paths for these 3 algorithm might be the same or not. The path for holding portfolio is the same as the general path, whish is simulated by a stock and is under the consideration. The path for BS and HH is subpath of the general path, but they must contain the first and last spot in the general path.

2.3 library definition

The system has two main libraries, The first is used for plotting, and the second is use to contain some general algorithm for financial.

1. Plotting
   I define a structure Point,which contains x and y coordinates of a point, to help pass the plotting information. The major plotting function is plottingCurve, which will plot a bare curve. Some fuctions can be consider as private functions,such as valuesComparing, defineColors. Users are not required to know these fuctions to plot curves. The 4 vectorPlotting functions are supposed to be interfaces for user.

Listing 8: ../source/plotting.h [Line 17 to 53]

```
#define PAPERSIZE g2_A4
#define MAX_PX 800
#define MAX_PY 600
#define EDGE 60
#define MOD1 15 //for non symetric X edge
#define MOD2 30 //for note area in Y
#define TE 25   //modification

struct Point{
double x;
double y;
};

void plottingCurve(vector<Point> vector, int id,double coeff_ax,double coeff_ay,
                   double coeff_bx,double coeff_by);
void valuesComparing(double& max_x, double& min_x,double& max_y ,double& min_y,double max_x1,
                     double min_x1,double max_y1 ,double min_y1);
void extremeValues(double& max_x, double& min_x,double& max_y ,double& min_y,
                   const vector<Point> vector1,const vector<Point> vector2);
void coefficient(double& coeff_ax, double& coeff_bx,double& coeff_ay ,double& coeff_by,
                 double max_x,double min_x,double max_y,double min_y);
void drawRectangle(int id,double max_x,double min_x,double max_y,double min_y);
void defineColors(int id,vector<int> &color);

void getPointVector(vector< vector<Point> > &pointVectors,
                    const vector< vector<double> > yvectors,const vector<double> xvector);
void readVector(vector<double> & approximationVector,
                vector<double> &MValues,ifstream &result,int length);
void vectorPlotting(const double exactValue,
                    const vector < vector<Point> > vectors,char *names[],int style);
//interface
void vectorPlotting(const vector< vector<Point> > vectors,char *names[]);
void vectorPlotting(const double exactValue,const vector < vector<Point> > vectors,char *names[]);
void vectorPlotting(const vector < vector<double> > yvectors,
                    const vector<double> xvector,char *names[]);
void vectorPlotting(const double exactValue,const vector < vector<double> > yvectors,
                    const vector<double> xvector,char *names[]);
```

In order to plot, user provide either a vector of type Point vectors, or a vector of x values and a vector of vectors of y values. If the function of the curves has a common exact value, this value can be put as the first parameter and the exact horizonal line for the curves will be plotted. The overloaded plotting functions are to provide convenience for plotting in different cases.

The functions that this library has are described below:

- plottingCurve: This function plots a curve given the vector of points for the curve. We use g2 package to plot curves, and g2 process

coordinates using its own coordination system. Therefore we need to transfer real the coordinates of the plotting curve into g2 coordinate system. The coefficient for this transformation are passed to this function.

- valuesComparing: This function compares 4 pairs of extreme values for 2 vectors, max for x, min for x, max for y, min for y. And it updates the real extreme values based on the comparation.
- extremeValues: This function computes the extreme values for several curves. The curves are represented by a vector of points. The purpose for computing the extreme values is to find out the coefficients so that all curves will be plotted in one screen.
- coefficient: This function is used to compute the coefficients for 4 known extreme values.
- drawRectangle: This function draws a rectangle to wrap the curves.
- defineColors: It defines the colors used to plot the curves.
- getPointVector: It transfers the curves represented in y vectors and x vector in point vectors.
- readVector: It reads from standard input device, and puts data into vector
- vectorPlotting: It plots out the curves represented by x vector and y vectors.

2. Financial Algorithms
   The functions defined in this library can be classified into 2 categories. The first category is about some simple algorithm that can be implemented in one function. The second is about sorting, and the difference between sorting in this system and the normal sorting is that we need to sort a vector but keep the orignal index for the elements in this vector.

   When we plot several vectors together, we need to do this kind of sorting based on the order of one vector. Normally, the basic vector contains the data for x dimension. The elements in these vectors are corelated to each other in this system. The element at the same position are obtained from the same condition. Only one vector will be sorted, and other vectors are reordered according the new order of the sorted vector. Therefore, the index of the sorted vector should be kept.

   The process for Haar compression is sorting the item vector first, then remove the elements from the smallest end, and place elements back to their original position. The information of index of each element need to be kept during sorting too.

Listing 9: ../source/falgorithm.h [Line 10 to 32]

```
double cumulative(double z);
```

Listing 9: ../source/falgorithm.h [Line 10 to 32] (continued)

```
double inverseCumulative(double u);
vector<double> partitiate(int J_H);
int getIndex(vector<double> spots,double value);
vector<double> matrixProduct(vector<double> rp1,vector<double> rp2);

struct indexItem
{
  double item;
  int index;
};
vector<indexItem> getIndexItems(vector<double> vector1);
vector<indexItem> mergeSort(vector<indexItem>  vectorCompress);

vector<indexItem> merge( vector<indexItem> vector1, vector<indexItem>  vector2);
void reorderVector(vector<double> &vector1,vector<int> indexes);
vector<int> sortVector(vector<double> &vector1);
void sortVectors(vector<double> &vector0,vector<double> &vector1);
void sortVectors(vector<double> &vector0,vector<double> &vector1, vector<double> &vector2);
void sortVectors(vector<double> &vector0,vector<double> &vector1, vector<double> &vector2,
                 vector<double> &vector3);
void sortVectors(vector<double> &vector0,vector<double> &vector1, vector<double> &vector2,
                 vector<double> &vector3,vector<double> &vector4);
```

The operation's defined in this library are explained in detail below:

- cumulative and inversecumulative: They are used in many places inside the system. I want to mention that inverse cumulative fuction was implemented by "Author" in his published book. We suppose that the probability distribution is normal. the cumulative of a double z means the area under the probability distribution curve from -∞ to z, and inversecumulative does the inverse operation.
- partitioning: It seperate the Y dimention line into 2^{j-H} intervals that has the same cumulative probability if the probability distribution is normal.
- getIndex: It returns the position index of the parameter value in the vector.

- matrixProduct: It retuns the product of 2 double vector.
- structure indexItem: It is defined for sorting a vector, which can be a member of a Haar object.
- getIndexItems: It returns a vector of indexItem based on the parameter, a vector of doubles.
- mergeSort: It sorts the vector of idexItem by comparing the values of items.
- merge: It merges 2 sorted vector of double.

- reorderVector: It reorder a vector based on the given order of an index vector.
- sortVector: It sorts one vector and returns the ondered index.
- sortVetors: It sorts several vectors based on the order of first vector.

3 Activity specification

I will summarize the formulars and algorithms used in computation. I separated these algorithms into three categories.

3.1 General Algorithm

1. Cumulative
   cumulative(double z), see handout "RA2003 III Page 3"

$$\Phi(z) = 1/2(1 + erf(z/\sqrt{2}))$$

   z - a double value on the y dimension line
   erf() - error function provided by the unix system
   The fuction returns the cumulative value from -∞ to z provided the distribution function is normal.

Listing 10: ../source/falgorithm.c [Line 12 to 15]

```
double cumulative(double z)
{
  return 0.5 * ( 1 + erf( z/sqrt(2.0) ) );
}
```

2. Partition
   participate(int j_H), see handout "RA2003 III Page 5"

$$C_k^{j\_H} = \Phi^{-1}(k/2^k)$$

   $C_k^{j\_H}$ - The participation spots along the Y dimension line
   j_H - Haar factor
   k - index, range from 1 to $2^{j\_H} - 1$
   The funtion participates y dimension line into 2^{j_H} parts and return a vector that contains $2^{j\_H} - 1$ values representing ordered spots in the dimension line.

Listing 11: ../source/falgorithm.c [Line 72 to 92]

```
vector<double> partitiate(int J_H)
{
```

Listing 11: ../source/falgorithm.c [Line 72 to 92] (continued)

```
  double spot;
  vector<double> spots;
  int length = 1<<J_H;

  for(int i=1; i < length/2; i++)
    {
      spot = inverseCumulative(1.0*i/length);
      spots.push_back(spot);
    }

  spots.push_back(0); //C_{2^{J-1}-1}^J

  for(int i = 1; i < length/2; i++)
    {
      spot = (-1) * spots[length/2 - 1 - i];
      spots.push_back(spot);
    }
  return spots;
}
```

3. Sorting
   Standard Temperate LIbrary (STL) in C++ doesn't define the sorting algorithm while keeping the index information, so we have to develop our own code. The core algorithm used here is merging.

   - getIndexItems: It saves the index for each element in the passed vector, and returns a new created vector contains indexItems.

Listing 12: ../source/falgorithm.c [Line 108 to 121]

```
vector<indexItem>  getIndexItems(vector<double> vector1){
  int length = vector1.size();

  indexItem aIndexItem;
  vector<indexItem> indexVector;

 for(int i=0; i < length; i++)
    {
      aIndexItem.item = vector1[i];
      aIndexItem.index = i;
      indexVector.push_back(aIndexItem);
    }
 return indexVector;
}
```

   - mergeSort:
     The merge function merges two sorted vectors into one sorted vector.

The standard algorithm for this case is used to do the operation. The first while loop is used to compare the items in two vectors. It puts the greater item in the temp vector, and it increase both iterators for temp vector and the vector that has the greater item. The second 2 while loop copies the rest elements in the vectorthat contains the greatest item. The return temp vector contains a descent sorted vector which includes all element in two input vectors.

Listing 13: ../source/falgorithm.c [Line 124 to 149]

```
vector<indexItem> merge(vector<indexItem> vector1, vector<indexItem>  vector2)
{
  vector<indexItem>::iterator first,first1,first2,last1,last2;
  vector<indexItem> temp(vector1.size() + vector2.size());

  first = temp.begin();
  first1 = vector1.begin();
  last1 = vector1.end();
  first2 = vector2.begin();
  last2 = vector2.end();

  while( (first1 < last1) && (first2 < last2))
    {
      if( fabs(first1->item) > fabs(first2->item))
        *(first++) = *(first1++);
      else
        *(first++) = *(first2++);
    }
  while(first1 < last1)
      *(first++) = *(first1++);

  while( first2 < last2 )
    *(first++) = *(first2++);

  return temp;
}
```

The mergeSort is used to sort a vector. It devides the vector into two vectors and sorts the subvectors. Then it merges both sorted vectors. This is a recursive function, so the division will continue until the subvector only contains one element.

Listing 14: ../source/falgorithm.c [Line 153 to 176]

```
vector<indexItem>  mergeSort(vector<indexItem>  vectorCompress)
{
  int i,half,whole;
  vector<indexItem> vector1, vector2;
  vector<indexItem> sortedVector1,sortedVector2;
                                                                    ➥
```

Listing 14: ../source/falgorithm.c [Line 153 to 176] (continued)

```
  // compressItem temp;
  if(vectorCompress.size() > 1)
     {
       whole = vectorCompress.size();
       half = whole/2;

       for(i=0;i<half;i++)
         vector1.push_back( vectorCompress[i] );
       for(i=half;i<whole;i++)
         vector2.push_back( vectorCompress[i] );

       sortedVector1 = mergeSort(vector1);
       sortedVector2 = mergeSort(vector2);
       return merge(sortedVector1,sortedVector2);
     }
   else
     return vectorCompress;
}
```

- sortVector: The vector is passed by reference. This function applies the mergesort fuction to sort the vector. The index vector, which records the index of the unsorted vector is returned.

Listing 15: ../source/falgorithm.c [Line 179 to 193]

```
vector<int> sortVector(vector<double> &vector1){
  vector<indexItem> indexedVector;
  vector<int> indexes;

  indexedVector = getIndexItems(vector1);
  indexedVector = mergeSort(indexedVector);

  int length = vector1.size();
  for(int i=0;i<length;i++){
    vector1[i] = indexedVector[length-1-i].item;
    indexes.push_back( indexedVector[length-1-i].index);
  }

  return indexes;
}
```

- reorderVector: The vector is passed by reference, and the index vector is passed too. The destination vector is ordered by the index vector.

Listing 16: ../source/falgorithm.c [Line 224 to 231] (continued)

Listing 16: ../source/falgorithm.c [Line 224 to 231]

```
void reorderVector(vector<double> &vector1,vector<int> indexes){
  vector<double> tempVector;
  int length = vector1.size();
  for(int i=0;i<length;i++)
    tempVector.push_back(vector1[i]);
  for(int i=0;i<length;i++)
    vector1[i]= tempVector[indexes[i]];
}
```

- sortVetors: It sorts more than one vector.

Listing 17: ../source/falgorithm.c [Line 214 to 221]

```
void sortVectors(vector<double> &vector0,vector<double> &vector1, vector<double> &vector2,
                 vector<double> &vector3,vector<double> &vector4){
   vector<int> indexes = sortVector(vector0);
   reorderVector(vector1,indexes);
   reorderVector(vector2,indexes);
   reorderVector(vector3,indexes);
   reorderVector(vector4,indexes);
}
```

4. Plotting
   g2 package is included to plot the output for this system. To plot a curve, I use g2_poly_lne from g2 library. I assume the g2 library will choose the proper points to represent the curve if too many points were passed to this function.

   - plottingCurve: Parameters inlude Vector of points in the curve, and 4 double coefficients to compute the pixel value. I pass device id, length and an array of points to g2_poly_line according to its signature.

Listing 18: ../source/plotting.c [Line 13 to 27]

```
void plottingCurve(vector<Point> vector, int id,double coeff_ax,
                   double coeff_ay,double coeff_bx,double coeff_by)
{
  int i,length;

  length = vector.size();
  double points[length*2];

  for(i=0; i < length; i++)
                                                                    ➥
```

Listing 18: ../source/plotting.c [Line 13 to 27] (continued)

```
    {
      points[2*i] = coeff_ax * vector[i].x + coeff_bx;
      points[2*i+1] = coeff_ay * vector[i].y + coeff_by;
    }
  g2_poly_line(id,length,points);
}
```

- extremeValues: It computes the extreme values for several vectors. These values include max_x,min_x, max_y,min_y. The algorithm is simple, which is getting extreme values for one vector, and then comparing with those of another, and keeping the greatest one. Repeat the process.

Listing 19: ../source/plotting.c [Line 64 to 80]

```
void extremeValues(double& max_x, double& min_x,double& max_y ,
                   double& min_y,const vector< vector<Point> > vectors)
{
  double max_y1,min_y1;

  max_x = vectors[0][vectors[0].size() - 1].x;
  min_x = vectors[0][0].x;
  extremeY(max_y1,min_y1, vectors[0]);
  max_y = max_y1;
  min_y = min_y1;

  for(int i=0; i < vectors.size(); i++){
    extremeY(max_y1,min_y1, vectors[i]);
    valuesComparing(max_x,min_x,max_y ,min_y,
                    vectors[i][vectors[i].size() - 1].x, vectors[i][0].x, max_y1, min_y1);
  }
}
```

- coefficient: It computes the coefficient based on extreme values. These coefficient are used to transfer the actual values for x and y coordinate into the x,y coordinate tha fit in the window.

Listing 20: ../source/plotting.c [Line 84 to 92]

```
void coefficient(double& coeff_ax, double& coeff_bx,double& coeff_ay ,
                 double& coeff_by,double max_x,double min_x,double max_y,double min_y)
{
  coeff_ax = ( MAX_PX - 2 * EDGE )/(max_x - min_x);
  coeff_bx = ( max_x * (EDGE+MOD1) - min_x * (MAX_PX - EDGE+MOD1) )/(max_x - min_x);

  coeff_ay = ( MAX_PY - 2 * EDGE - MOD2 )/(max_y - min_y);
  coeff_by = ( max_y * EDGE - min_y * (MAX_PY - EDGE - MOD2) )/(max_y - min_y);
```

➥

Listing 20: ../source/plotting.c [Line 84 to 92] (continued)

```
}
```

- vectorPlotting: This is the general vector plotting fuction. It is supposed to be used by other vector plotting fuctions,since it tries to cover all cases of plotting. A user has to decide the exact value and style in order to call this fuction. But it's hard to understand the meanings of these parameters and then to use it properly. Instead, user can use other vector plotting function which only require xvector and yvectors. The detailed names can be found in session 2.3 definition on page 12.

Listing 21: ../source/plotting.c [Line 151 to 220]

```
void vectorPlotting(const double exactValue,const vector < vector<Point> > vectors,
                    char *names[],int style)
{
  double max_x,max_y,min_x,min_y;
  double coeff_ax, coeff_bx,coeff_ay, coeff_by;

  int id,id_PS, id_X11;
  char *filename = "./Output/result.ps",str[20];

  double px_0,py_0;
  double px_1,py_1;

  vector<int> color;
  double dashes[3];

  extremeValues(max_x,min_x,max_y,min_y,vectors);
  //max_y += 0.15 * (max_y - min_y);

  coefficient(coeff_ax,coeff_bx,coeff_ay,coeff_by,max_x,min_x,max_y,min_y);

  //open devices
  id = g2_open_vd();
  id_X11 = g2_open_X11((int)MAX_PX,(int)MAX_PY);
  id_PS=g2_open_PS(filename,PAPERSIZE,g2_PS_land);
  //attach x11 and ps to visual device
  g2_attach(id,id_X11);
  g2_attach(id,id_PS);

   //draw rectangle
  drawRectangle(id,max_x,min_x,max_y,min_y);

  //draw the exact line
  if(style == 1){
    px_0 = coeff_ax * min_x + coeff_bx;
```

Listing 21: ../source/plotting.c [Line 151 to 220] (continued)

```
    px_1 = coeff_ax * max_x + coeff_bx;
    py_0 = coeff_ay * exactValue + coeff_by;
    py_1 = py_0;
    g2_line(id,px_0,py_0,px_1,py_1);
    if(exactValue == 0)
      g2_string(id,px_0 - 15,py_0,"0");
    else
      {
        sprintf(str,"exact value=%-8.6f",exactValue);
        g2_string(id,px_1 - 160,py_0+10,str);
      }
  }

  g2_set_line_width(id,2);

  defineColors(id_PS,color);
  defineColors(id_X11,color);

  dashes[0]=15.0;
  dashes[1]=5.0;
  dashes[2]=5.0;

  for(int i=0; i < vectors.size(); i++){
    g2_set_dash(id,i*5,dashes);
    g2_pen(id,color[i]);

    g2_line(id,(2 + i*4) * EDGE, MAX_PY-EDGE-MOD2*0.5,(3 + i*4) * EDGE, MAX_PY-EDGE-MOD2*0.5);
    g2_string(id,(3.2 + i*4) * EDGE, MAX_PY-EDGE-MOD2*0.7, names[i]);

    plottingCurve(vectors[i], id,coeff_ax,coeff_ay,coeff_bx,coeff_by);
  }

  g2_flush(id);
  getchar();
  g2_close(id);
}
```

3.2 Basic Algorithm

1. underlying
   oneStepUnderlying1(int J_H,double currentPrice,double deltaT), see handout "RA2003 III Page 6", I replace the T-T0 as δT, so the underlying just compute a small step.

   $$S_{J\_H}(i) = ae^{b^2/2}2^{J\_H}(\Phi(C_{i+1}^{J\_H} - b) - \Phi(C_i^{J\_H} - b))$$

   where
   $a = S_{T_0}e^{\gamma(T-T_0)}$

$b = \sigma\sqrt{(T - T_0)}$
$\gamma = r - \sigma^2/2$

haarUnderlying1(int J_H,vector $< double >$ times);

Listing 22: ../source/stock.c [Line 46 to 67]

```
vector<double> Stock::oneStepUnderlying1(int J_H,double currentPrice,double deltaT){
  vector<double> underlying;

  double nu = r - pow(sigma,2.0)/2.0;
  double a = currentPrice * exp( nu * deltaT );
  double b = sigma * sqrt(deltaT);
  vector<double> spots = partitiate(J_H);
  int length = spots.size()+1;
  double c[length+1];

  c[0]=0;
  c[length]=1;
  for(int i=1;i<length;i++)
    {
      c[i]=cumulative(spots[i-1] - b);
    }

  for(int i=0;i<length;i++)
    underlying.push_back( a * exp( pow(b,2)/2 ) * length * ( c[i+1] - c[i]));

  return underlying;
}
```

2. Haar analysis
   Haar Haar::oneStepHaarAnalysis(), see handout "RA2003 I P.3"
   Transform $S_j(i)(i = 0, 1, 2, ..., 2^j - 1)$ to
   $(S_{j-1}(i), d_{j-1}(i))i = (0, 1, 2, ..., 2^{j-1} - 1)$

   $$d_{j-1}(i) = 1/\sqrt{2}(S_j(2i + 1) - S_j(2i))S_{j-1}(i) = \sqrt{2}S_j(2i) - d_{j-1}(i)$$

   The fuction return a vector contains the next row in the haar matrix.

   Haar Haar::haarAnalysis(int numberOfCalls), see handout "RA2003 I P.3"

   The function analysize the haar vector numberOfCalls times. It calls the oneStepHaarAnalysis numberOfCalls times, and returns the vector contains the numberOfCalls row in the haar matrix for the original vector. The greatest value for numberOfCalls is J if the caller haar vector is original.

Listing 23: ../source/haar.c [Line 23 to 54]

```
Haar Haar::oneStepHaarAnalysis(){
  int p=1 << (j-1);
  Haar OV(haarItems,J,j-1);

  vector<double>::iterator OV_first_ptr = OV.haarItems.begin();
  vector<double>::iterator OV_second_ptr = OV_first_ptr + p;

  double d;
  for(int i=0; i < p; i++)
    {
      d = (1/sqrt(2.0)) * ( haarItems[2*i + 1] - haarItems[2*i] );
      *(OV_second_ptr++) = d;
      *(OV_first_ptr++)  = sqrt(2.0) *  haarItems[2*i] + d;
    }
  return OV;
}

//transform haar vector numberOfCalls steps by call oneStepAnalysis

Haar Haar::haarAnalysis(int numberOfCalls){
  if(numberOfCalls > j){
    cout<<"Too much times of Analysis"<<endl;
    exit(1);
  }

  Haar OV(haarItems,J,j);

  for(int i=0;i < numberOfCalls; i++)
      OV = OV.oneStepHaarAnalysis();

  return OV;
}
```

3. Haar compression
   Haar compression means replacing some of elements in the haar vector with 0.0 based on some cretiria. Normally, we only keep the greatest R elements which have higher absolute values. The last row of the matrix is compressed, and this vector is computed by haar sysnthesis.

   When we compute the sample haar hedging payoff for an option, we use the same input for Haar Hedging and δ Hedging, and this initial input is the holding portfolio at t0. The S0 in the last row of the haar matrix actually represent the total average of the original haar vector, so we set S0=0 if we want to use the holding portfolio at t0.

Listing 24: ../source/haar.c [Line 102 to 123] (continued)

Listing 24: ../source/haar.c [Line 102 to 123]

```
Haar Haar::RCompression(int R){
  vector<indexItem> indexVector = getIndexItems(haarItems);

  indexVector = mergeSort(indexVector);

  vector<double> v(1<<J,0.0);
  Haar OV(v,J,j);
  vector<double>::iterator OV_ptr;

  for(int i=0; i < R; i++)
    {
      OV_ptr = OV.haarItems.begin();
      OV_ptr += indexVector[i].index;
      *OV_ptr = indexVector[i].item;
    }
  return OV;
}

Haar Haar::SORCompression(int R){
  haarItems[0] = 0;
  return RCompression(R);
}
```

4. Haar synthesis
   Haar Haar::oneStepHaarSynthesis(), see handout "RA2003 II P.1"
   Transform $(S_j(i), d_j(i))(i = 0, 1, 2, ..., 2^j - 1)$ to
   $S_{j+1}(i)(i = 0, 1, 2, ..., 2^{j+1} - 1)$

   $$S_{j+1}(2i) = 1/\sqrt{2}(S_j(i) - d_j(i))S_{j+1}(2i + 1) = S_{j+1}(2i) + \sqrt{2}d_j(i))$$

   The fuction return a vector contains the next higher row in the haar matrix.

   Haar Haar::haarSynthesis(int numberOfCalls), see handout "RA2003 II Page 1"
   The function SYNTHESIZE the haar vector numberOfCalls times. It returns the vector contains the next higher numberOfCalls row to the original vector in the haar matrix. The greatest value for numberOfCalls is J if the caller haar vector is bottom one(j=0).

Listing 25: ../source/haar.c [Line 69 to 98]

```
Haar Haar::oneStepHaarSynthesis(){
  Haar OV(haarItems,J,j+1);
```

Listing 25: ../source/haar.c [Line 69 to 98] (continued)

```
  vector<double>::iterator OV_ptr = OV.haarItems.begin();

  int p = 1<<j;

  double d;
  for(int i=0; i < p; i++)
    {
      d = haarItems[p + i];
      *OV_ptr = ( 1/sqrt(2.0) ) * ( haarItems[i] - d );
      *(OV_ptr+1)  = *OV_ptr + sqrt(2.0) *  d;
      OV_ptr +=2;
    }
  return OV;
}

//transform vector numberOfCalls steps by call oneStepAnalysis
Haar Haar::haarSynthesis(int numberOfCalls){
  if(numberOfCalls > J-j){
    cout<<"Two much times of systhesis"<<endl;
    exit(1);
  }

  Haar OV(haarItems,J,j);

  for(int i=0;i < numberOfCalls; i++)
    OV = OV.oneStepHaarSynthesis();
  return OV;
}
```

5. path simulation
   simulatePath(), see Pathwise MRA handout I P.1

$$S_{h_{i+1}} = S_{h_i} * e^{\gamma*(h_{i+1}-h_i)+\sigma*\sqrt{(h_{i+1}-h_i)}*W_{h_i,h_{i+1}}}$$

   where
   $W_{h_i,h_{i+1}} \sim N(0,1)$
   $\gamma = r - \sigma^2/2$

   A vector of times is passed to this function, and it contains N+1 ordered time spots, i.e. including 0 as t0. A Path Object will be returned.

Listing 26: ../source/stock.c [Line 19 to 38]

```
Path Stock::simulatePath(vector<double> times){
  double drawNumber,NDNumber;
  double deltaT;
  double ST = St0;
```

Listing 26: ../source/stock.c [Line 19 to 38] (continued)

```
  vector<double> prices,allJumps;

  prices.push_back(ST);
  allJumps.push_back(1.0);
  for(int i=1;i<times.size();i++)
    {
      drawNumber=rand()/(double)(RAND_MAX);
      NDNumber = inverseCumulative(drawNumber);

      allJumps.push_back(NDNumber);
        deltaT = times[i] - times[i-1];
      ST = ST * exp( (r - pow(sigma,2.0)/2) * deltaT  +  NDNumber * sigma * sqrt(deltaT));
      prices.push_back(ST);
    }
  return Path(prices,allJumps,times);
}
```

6. subpath generation
   Subpath is a relatively simple algorithm. A vector of subtimes is passed to the function. The path object will generate a subpath object by extracting information from the general path at the time difined in subtime vector. It's very useful for computing Haar Hedging and Δ Hedging.

   Note: When several steps are skipped, skipped elements in allJumps, which is member vector of a path, are summed up and then divided by the number of elements skipped. The result jump number is push into the allJump vector of the generating subpath.

Listing 27: ../source/stock.c [Line 132 to 149]

```
Path Path::subPath(  vector<double> subtimes ){
  vector<double> thePrices,theJumps;
  double jump; int j,m=0;

  for(int i=0;i<subtimes.size();i++){
    jump = 0.0;
    for(j=m;j<times.size();j++){
      jump += allJumps[j];
      if(times[j] == subtimes[i]){
        thePrices.push_back(prices[j]);
        theJumps.push_back(jump/sqrt(j-m+1.0));
        break;
      }
    }
    m = j+1;
  }
  return Path(thePrices,theJumps,subtimes);
```

Listing 27: ../source/stock.c [Line 132 to 149] (continued)

```
}
```

7. d1

   get_d1(), see HH Handout II P10, and HH Handout V P.7

$$d1(S_{t_j}) = \frac{ln(S_{t_j}/P) + (r + \sigma^2/2)(T - t_j)}{\sigma\sqrt{(T - t_j)}}$$

   where

   j - index of time spots, j = 1 ... N(N_H or N_BS)

   S_{t_j} - sample stock price at time t_j

   P - strike price

   T - the whole time period

   t_j - the time period up to spot j

   σ - market volatility for the stock

   r - non-risk bank interest

   $(T - t_j)$ - the time from the spot j to expiration spot

Listing 28: ../source/option.c [Line 36 to 40]

```
double Option::get_d1(double stockPrice,double time)
{
  return (log(stockPrice/P) + (stock.r + pow(stock.sigma,2.0)/2.0) * (etime - time))
    / (stock.sigma * sqrt(etime - time));
}
```

8. d2

   get_d2(), see HH Handout II Page 10

$$d2(S_{t_j}) = d1(S_{t_j}) - \sigma\sqrt{(T - T_j)}$$

Listing 29: ../source/option.c [Line 44 to 47]

```
double Option::get_d2(double d1,double time)
{
  return d1 - stock.sigma * sqrt(etime - time);
}
```

9. option value

double value(double stockPrice,double time), double value(double stock-Price,double time), see B & S Approximation Detail P.4

$$V(Y(S_{ti}, ti)) = a * (S_{ti} * \Phi(d1(S_{ti}, ti)) - P * e^{-r(T-t0)} * \Phi(d2(S_{ti}, ti)))$$

$$V(Z(S_{ti}, ti)) = a * (P * e^{-r(T-t0)} * \Phi(-d2(S_{ti}, ti)) - S_{ti} * \Phi(-d1(S_{ti}, ti)))$$

where

$V(Y(S_{ti}, ti))$ - value for a European Call option

$V(Z(S_{ti}, ti))$ - value for a European Put option

i - index of time spots, i = 1 ... N(N_H or N_BS)

S_{ti} - sample stock price at time ti

P - strike price

t_i - the time period up to spot i

r - non-risk bank interest

a - The amount of an option

Listing 30: ../source/option.c [Line 74 to 80]

```
double EuCall::value(double stockPrice,double time){
  double d1 = get_d1(stockPrice,time);
  double d2 = get_d2(d1,time);
  double value = amount * (stockPrice * cumulative(d1)
                           - P * exp( -stock.r * (etime - time)) * cumulative(d2));
  return value;
}
```

Option value at a given time depends on non-risk bank interest and its expiration time and price. It also depends on the stock price at that given time and the given time itself. The value interval for the given time is from 0 to the time before its expiration time. When the given time is equal to its expiration time, the option will be realized, and its value is expiration value.

10. option expiration value

    double evalue(double stockPrice), double evalue(double stockPrice)

$$V(Y(S_T)) = a * max(0, S_T - P)$$

$$V(Z(S_T)) = a * max(0, P - S_T)$$

where

$V(Y(S_T))$ - expiration value for European Call option

$V(Z(S_T))$ - expiration value for European Put option

Listing 31: ../source/option.c [Line 84 to 86]

```
double EuCall::evalue(double stockPrice){
  return amount * max(0.0,(stockPrice - P));
}
```

The value of an option at expiration time should be computed by these 2 formulars. The formulars at the above section are used to compute premature option value.

3.3 Specific Algorithm

1. computing holding portfolio

   sampleHoldingPortfolio(Path path), see B & S Approximation Detail P.4,10

   Value of holding portfolio at ti:

$$V_{t0}(X^{ti}) = \sum_{j=1}^{C^{ti}} V(Y(S_{ti}, ti)) + \sum_{j=1}^{P^{ti}} V(Z(S_{ti}, ti))$$

   where

   $V(Y(S_{ti}, ti))$ - value of an European Call Option.

   $V(Z(S_{ti}, ti))$ - value of an European Call Option. Both value might be computed as a premature option or as an expired option.

   ti - the time when the option value is calculated. It ranges from t0 to T, which is the common final time.

   C^{ti} - the number of call options at time ti. It includes the premature options and the option expired at the time ti, but it dosen't include the option expired before time ti.

   P^{ti} - the number of Put options at time ti. It includes the premature options and the option expired at the time ti, but it dosen't include the option expired before time ti.

   The sum of values of options at each spots is called the holding value of the portfolio. The option that expired at the spot is computed by expiration value formular. The value of premature options are computed by option value formular. The computation implementation of value and expiration value for an option is inside the option class.

   This fucion will return a vector contains N values corresponding to the spots in the passed parameter path. We don't have a holding value for the last spot, at which spot time = T.

Listing 32: ../source/portfolio.c [Line 77 to 97]

```
vector<double> Portfolio::sampleHoldingPortfolio(Path path){
  vector< vector<double> > hdps;
  vector<double> HDP;
  HDP.assign(path.times.size(),0.0);

  if(ecs.size()==0 && eps.size()==0){
    cout<<"No options in the porfolio error"<<endl;
    exit(1);
  }

  for(int i = 0; i < ecs.size(); i++)
    hdps.push_back( ecs[i].holdingPortfolio(path) );

  for(int i = 0; i < eps.size(); i++)
    hdps.push_back( eps[i].holdingPortfolio(path) );

  for(int i = 0; i < hdps.size(); i++)
    for(int j = 0; j < hdps[i].size(); j++)
       HDP[j] += hdps[i][j];
  return HDP;
}
```

2. computing Black and Schole portfolio

   sampleBSPortfolio(Path path), see handout B & S Application for multi-time Porforliio.
   This function computes value of a portfolio along a simulated stock path using Black and Schole algorithm, whih is δ hedging. The parameter path is particularly for Black and Schole Hedging, and it can be achieved from the general path. We rebalance the portfolio at each spot on the path, and some options will expire at some of the spots on the path.

   The function returns a vector that contains the BS values at each spot. The vector also contain the initial value of the holding portfolio at the begining(t0). Then we compute the value of the portfolio at t1,t2,t3,...T(last point at the path). We rebalance the portfolio at each spot except T spot. When computing the value of the BS portfolio at t_i, we rebalance the portfolio at t_{i-1}. If any option expires at a spot t_{i-1}, the payoff of the option is subtracted from the the rebalance value ξ. Suppose we have $N_{BS}+1$ spots in the sample path(including the first spot st0), we will get N_{BS} values in the vector returned by this function.

   - Value of a call option at time spot t_j:
     $VCX(S_{t_j}) = a * (S_{t_j}\Phi(d1(S_{t_j})) - Pe^{-r(T-t_j)}\Phi(d2(S_{t_j})))$
   - Value of a put option at time spot t_j:
     $VPX(S_{t_j}) = a * (Pe^{-r(T-t_j)}\Phi(-d2(S_{t_j})) - S_{t_j}\Phi(-d1(S_{t_j})))$

a - amount of the option

- Total value of the call and put options:(value of the holding portfolio)
  $VX(S_{t_j}) = \sum_{i=1}^{I}(a_i * VX_i(S_{t_j}))$
  I - number of options
- Initial portfolio:
  Initialy, value of the holding portfolio equals to the value of the hedging portfilio.
  $BSP(S_{t0}) = VX_T(S_{t0})$
- Two coefficient ϕ and ξ:
  ϕ and ξ means the cash on the bank account and the number of shares of the stock respectively.
  $\phi c_{j-1} = \Phi(d1(S_{t_j}))$ (for a call option)
  $\phi p_{j-1} = -\Phi(-d1(S_{t_j}))$ (for a put option)
  $\phi_{j-1} = \sum_{i=1}^{I}(a_i * \phi_{i,j-1})$
  $\xi_{j-1} = BSP(S_{t_j}) - \phi_{j-1}S_{t_{j-1}}$
- Recursive computation of BSP:(rebalance update)
  $BSP(S_{t_j}) = \xi_{j-1}e^{r(t_j - t_{j-1})} + \phi_{j-1}S_{t_j}$

Listing 33: ../source/portfolio.c [Line 19 to 73]

```
vector<double> Portfolio::sampleBSPortfolio(Path path){
  vector<double> BSPs;
  double BSP=0;
  if(ecs.size()==0 && eps.size()==0){
    cout<<"No options in the porfolio error"<<endl;
    exit(1);
  }

  for(int i = 0; i < ecs.size(); i++){
    BSP += ecs[i].value(path.prices[0], 0.0);
  }
  for(int i = 0; i < eps.size(); i++){
    BSP += eps[i].value(path.prices[0], 0.0);
  }
  BSPs.push_back(BSP);   //black and schole value at t0

  double deltaT;
  double d1,phi,xi,epayoff; //expire payoff
  for(int i=1; i < path.times.size(); i++){
    deltaT = path.times[i] - path.times[i-1];
    phi = 0;
    epayoff = 0;

    for(int j = 0; j < ecs.size(); j++){
```

Listing 33: ../source/portfolio.c [Line 19 to 73] (continued)

```
      if(path.times[i-1] < ecs[j].etime){
        d1 = ecs[j].get_d1(path.prices[i-1],path.times[i-1]);
        phi += ecs[j].get_phi(d1);
      }
      //else if(path.times[i-1] == ecs[j].etime)
      //epayoff += ecs[j].evalue(path.prices[i-1]);

    }

    for(int j = 0; j < eps.size(); j++){
      if(path.times[i-1] < eps[j].etime){
        d1 = eps[j].get_d1(path.prices[i-1],path.times[i-1]);
        phi += eps[j].get_phi(d1);
      }
      //else if(path.times[i-1] == eps[j].etime)
      //epayoff += eps[j].evalue(path.prices[i-1]);
    }

    xi = BSP - phi * path.prices[i-1];
    //xi = BSP - phi * path.prices[i-1] - epayoff;
    double interest;
    if(ecs.size()>0) //at least one option
      interest = ecs[0].stock.r;
    else
      interest = eps[0].stock.r;
    BSP = xi * exp(interest * deltaT) + phi * path.prices[i];

    BSPs.push_back(BSP);
  }
  return BSPs;
}
```

3. computing Haar Hedging payoff

   sampleHHPayoff(Path path,int j_H,int R), see Handout "Draft for HH for multitime option"

   Haar Hedging will be performed along the HH path extracted from the general path, and the number of times is N_H. We will get N_H+1 values of HH payoffs (hpa in handout). Input at t0 is identical to the first holding portfoliio.

   Input at t1 for a Call option is computed by

$$Y_{t1}(i) = (S_{t1}(i) - K)_+$$

   where

   S_{t1} is the underlying vector of this stock at t1, and i is the index value corresponding the random jump value at t1. See basic algorithm "underlying".

k is the strike price for this option.

Bottom Script + means the result should be either 0 or a posive number.

Input at later time spots for a call option is computed by

$$Y_{tj}(i) = e^{-r(T-tj)} * (e^{a^2/2} B(S_{tj})(i))(1-\Phi(C(S_{tj}(i))-a)) - K(1-\Phi(C(S_{tj})(i))))$$

where

S_{tj} is the stock underlying at tj, i is th index value

a is $\sigma\sqrt{T-tj}$, and T is the option's expiration time.

$B(S_{tj})(i) = S_{tj}(i)e^{(r-\sigma^2/2)(T-tj)}$

$C(S_{tj})(i) = 1/a * ln(K/B(S_{tj})(i))$

The code for computing payoff for one European Call option is listed below:

Listing 34: ../source/option.c [Line 112 to 135]

```
vector<double> EuCall::HHPayoff(int j_H,double stockPrice,double time,double deltaT){
  if(time >= etime){
    cout<<"Hedging time should not greater or equal to option's expiration time"<<endl;
    exit(1);
  }

  vector<double> underlying = stock.oneStepUnderlying1(j_H,stockPrice,deltaT);
  vector<double> Xis;
  double Xi;
  // cout<< abs((time + deltaT) - etime) <<endl;
  if(  abs((time + deltaT) - etime) < 0.001 ){
    for(int j=0;j<underlying.size();j++){
      Xi = amount * max(0.0,underlying[j] - P);
      Xis.push_back(Xi);
    }
  }
  else{
    for(int j=0;j<underlying.size();j++){
       Xi = value(underlying[j],time+deltaT);
      Xis.push_back(Xi);
    }
  }
  return Xis;
}
```

For computation of HH payoff for a portfolio. I calculate the input for every option at each step using the above algorithm. I will get m vectors, and m is the number of options not been expired at the step. Then I add all m vectors together to get the total payoff vector.

The haar algorihm is applied on the total payoff vector. We perform haar

analysis, haar compression, and haar synthesis on the vector. For t0, We use Holding payoff as the Haar hedging payoff, and this initial payoff will be further iterated in the later haar operations. Therefore, we keep S0 = 0 while compression. Otherwise, the average will be cumulated. For haar algorithm, please see the basic algorithm section.

The corresponding index to the holding and B&S algorithm is computed based on the allJump numbers inside the generated path. In this system, We only simulate one general path. the paths for B&S and HH are all a subpath of the general path. HoW to compute a subpath for HH is explained in Basic Algorithm for subpath.

Listing 35: ../source/portfolio.c [Line 101 to 174]

```
vector<double> Portfolio::sampleHHPayoff(Path path,int j_H,int R){
  vector<double> HHPs;
  double HHP=0;
  double Xi;
  vector<double> Xis;
  vector< vector<double> > payoffs;

  double interest;
  if(ecs.size()>0) //at least one option
    interest = ecs[0].stock.r;
  else
    interest = eps[0].stock.r;

  if(ecs.size()==0 && eps.size()==0){
    cout<<"No options in the porfolio error"<<endl;
    exit(1);
  }

  for(int i = 0; i < ecs.size(); i++){
    HHP += ecs[i].value(path.prices[0], 0.0);
  }
  for(int i = 0; i < eps.size(); i++){
    HHP += eps[i].value(path.prices[0], 0.0);
  }
  HHPs.push_back(HHP);
  //at this point HHP contains V_{t_0}(X^{t_0})
  //with this initialization we should take s_0[0] =0

  //here we compute/load the input to Haar analysis
  // HHP =0; //FOR DEBUGGING
  for(int k=0; k<path.times.size()-1; k++){ //start of big loop
    payoffs.clear();
    for(int i = 0; i < ecs.size(); i++){
      Xis.clear();
      if(path.times[k] < ecs[i].etime){
        Xis = ecs[i].HHPayoff(j_H,path.prices[k],path.times[k],path.times[k+1]
                                                                          ➥
```

Listing 35: ../source/portfolio.c [Line 101 to 174] (continued)

```
                              - path.times[k]);
        payoffs.push_back(Xis);
      }
    }

    for(int i = 0; i < eps.size(); i++){
      Xis.clear();
      if(path.times[k] < eps[i].etime){
        Xis = eps[i].HHPayoff(j_H,path.prices[k],path.times[k],path.times[k+1]
                              - path.times[k]);
        payoffs.push_back(Xis);
      }
    }

    //sum up contributions of all options
    Xis.clear();
    for(int i = 0; i < payoffs[0].size(); i++){
      Xi = 0;
      for(int j = 0; j < payoffs.size(); j++)
        Xi += payoffs[j][i];
      Xis.push_back(Xi);
    }

    Haar h(Xis,j_H,j_H);

    h = h.haarAnalysis(j_H);
     h = h.SORCompression(R); //this compression sets s_0[0]=0
    //h = h.RCompression(R);//this compression DOES NOT force s_0[0]=0
    h = h.haarSynthesis(j_H);

    vector<double> spots = partitiate(j_H);
    int index = getIndex(spots,path.allJumps[k+1]);
    HHP = HHP * exp((path.times[k+1]-path.times[k]) * interest) +
      h.haarItems[index];
    HHPs.push_back(HHP);
  }//end of big loop
  return HHPs;
}
```

4 Test Plans

Testing is very important to make sure that each unit, module, and whole system work can work properly. If no bugs are found during some testing, it give us onfident of the software.

4.1 Unit Test

Considering the unit testing, I developed four independent testing codes. The plotting library has been used and tested by other software, therefore I didn't write specific code to test its functionality. However, it's a tool to show the computation result, so it functionality is tested each time when I run a test for other unit.

Here are the four test stubs:

1. Test Haar class The operations of haar analysis, symthesis, and compression are tested. The input data, which should be a vector, is from reference 1.

Listing 36: ../source/test/haarTest.c [Line 10 to 54]

```
int main(int argc, char **argv)
{
  vector<double> transformVector;

  transformVector.push_back(56);
  transformVector.push_back(40);
  transformVector.push_back(8);
  transformVector.push_back(24);
  transformVector.push_back(48);
  transformVector.push_back(48);
  transformVector.push_back(40);
  transformVector.push_back(16);

  Haar IV(transformVector,3,3);

  Haar OV1 = IV.haarAnalysis(3);

  int R=3,Rt=0;
  double threshold = 0.95;

  //Haar OV2 = OV1.RCompression(R);
  Haar OV2 = OV1.thresholdCompression(threshold,Rt);
  cout<<"Rt = "<<Rt<<endl;
  Haar OV3 = OV2.haarSynthesis(3);
  Haar OV4 = OV1.haarSynthesis(3);


  char *names[20];
  vector<double> xvector;
  vector< vector<double> > yvectors;

  for(int i=0; i<8;i++)
    xvector.push_back(i * 1.0);

  yvectors.push_back(IV.haarItems);
```

Listing 36: ../source/test/haarTest.c [Line 10 to 54] (continued)

```
  yvectors.push_back(OV3.haarItems);
  yvectors.push_back(OV4.haarItems);

  names[0] = "Original";
  names[1] = "Compressed";
  names[2] = "Non-compressed";

  vectorPlotting(yvectors,xvector,names);
  return 0;
}
```

2. Test Stock class The stock class is tested regarding its ability to generate a path and simulate an underlying at a fixed point.

Listing 37: ../source/test/stockTest.c [Line 11 to 59]

```
int main(int argc,char *argv[]){
  if(argc !=2){
    cout<<"stockTest 1(path)/2(underlying)/3(compare underlying algorithms)"<<endl;
    exit(1);
  }
  double St0=20.0,interest=0.2,sigma=0.2;
  Stock stock(St0, interest, sigma);

  vector<double> times;
  for(int i = 0; i<4;i++)
    times.push_back(i*0.1);


  char *names[20];
  vector<double> xvector;
  vector< vector<double> > yvectors;

  if(atoi(argv[1]) == 1){
    names[0] = "All jumps";
    names[1] = "All prices";
    Path path = stock.simulatePath(times);

    xvector = path.times;
    yvectors.push_back( path.allJumps );
    yvectors.push_back( path.prices );
  }
  else if(atoi(argv[1]) == 2){
    names[0] = "Haar underlying";
    vector<double> underlying = stock.haarUnderlying(4,times);

    for(int i=0; i<underlying.size();i++)
```

Listing 37: ../source/test/stockTest.c [Line 11 to 59] (continued)

```
      xvector.push_back(i * 1.0);
    yvectors.push_back( underlying );
  }
  else{
    names[0] = "underlying v1";
    names[1] = "underlying v2";
    vector<double> underlying1 = stock.oneStepUnderlying1(10,St0,times[1]-times[0]);
    vector<double> underlying2 = stock.oneStepUnderlying2(10,St0,times[1]-times[0]);

    for(int i=0; i<underlying1.size();i++)
      xvector.push_back(i * 1.0);
    yvectors.push_back( underlying1 );
    yvectors.push_back( underlying2 );
  }

  vectorPlotting(yvectors,xvector,names);
  return 0;
}
```

3. Test Option class The code below is used particularly to test the HHPayoff of an option. The result for a Call and a Put option are compared and plotting out.

Listing 38: ../source/test/optionTest.c [Line 13 to 60]

```
int main(int argc, char **argv){
  double St0=21.0,interest=0.2,sigma=0.2;
  double time1=0.9,time2=0.9,strikePrice=20.0,amount=1.0; int optionType=1;
  vector<double> times;
  for(int i = 0; i<10;i++)
    times.push_back(i*0.1);

  Stock stock(St0, interest, sigma);
  EuCall ec(stock, time1, strikePrice, amount);
  EuPut ep(stock, time2, strikePrice, amount);

  Path path = stock.simulatePath(times);


  cout<<"Times are:  ";
  for(int i =0; i<times.size();i++)
    cout<<times[i]<<" ";
  cout<<endl;

  cout<<"Path is:  ";
  for(int i =0; i<path.prices.size();i++)
    cout<<path.prices[i]<<" ";
```

Listing 38: ../source/test/optionTest.c [Line 13 to 60] (continued)

```
  cout<<endl;

  int k=7,j_H=4;
  vector<double>  hhpayoff1= ec.HHPayoff(j_H,path.prices[k],path.times[k],
                                         path.times[k+1] - path.times[k]);

  vector<double> hhpayoff2 =  ep.HHPayoff(j_H,path.prices[k],path.times[k],
                                          path.times[k+1] - path.times[k]);

  char *names[20];
  names[0] = "Call HH Payoff";
  names[1] = "Put HH Payoff";

  vector<double> xvector = times;
  vector< vector<double> > yvectors;
  yvectors.push_back(hhpayoff1);
  yvectors.push_back(hhpayoff2);

  cout<<hhpayoff1.size()<<" "<<hhpayoff2.size()<<endl;
  for(int i =0; i<hhpayoff1.size();i++)
    cout<<hhpayoff1[i]<<" "<<hhpayoff2[i]<<" "<<endl;

  vectorPlotting(yvectors,xvector,names);

  return 0;
}
```

4. Test Portfolio class I test different portfolio value along a path. The path is generated randomly, But the system gives us the same sequence of random numer each time we want a path. To solve this problem,I developed a member function with a random seed to be called for testing. This fuction won't be used by normal use of the system. Some typical input and output are described below.

Listing 39: ../source/test/portfTest.c [Line 14 to 95]

```
int main(int argc, char **argv){
  double St0=21.0,interest=0.2,sigma=0.2;
  double time1=1,time2=0.5,strikePrice=21.0,amount=1.0;
  int seed=3;
  vector<double> times;
  for(int i = 0; i<=10;i++)
    times.push_back(i*0.1);

  Stock stock(St0, interest, sigma);
  EuCall ec(stock, time1, strikePrice, amount);
  EuPut ep(stock, time2, strikePrice, amount);
```

Listing 39: ../source/test/portfTest.c [Line 14 to 95] (continued)

```
Path path = stock.simulatePath(times,seed);

cout<<"Times are:  ";
for(int i =0; i<times.size();i++)
  cout<<times[i]<<" ";
cout<<endl;

cout<<"Path is:  ";
for(int i =0; i<path.prices.size();i++)
  cout<<path.prices[i]<<" ";
cout<<endl;

vector<EuCall> ecs;
vector<EuPut> eps;

ecs.push_back(ec);
//eps.push_back(ep);

Portfolio p(ecs,eps);

vector<double> bsps = p.sampleBSPortfolio(path);
vector<double> hdps = p.sampleHoldingPortfolio(path);
vector<double> hhps = p.sampleHHPayoff(path,10,1);

vector<double> rebalanceTimes;
for(int i=0;i<times.size();i++)
  rebalanceTimes.push_back(times[i]);

cout<<bsps.size()<<" "<<hdps.size()<<" "<<hhps.size()<<endl;
for(int i =0; i<bsps.size();i++)
  cout<<bsps[i]<<" "<<hdps[i]<<" "<<hhps[i]<<endl;

vector<double> xvector = rebalanceTimes;
vector< vector<double> > yvectors;

int option=0;
if(argc==2)
  option=atoi(argv[1]);
if(option==0){
  char *names[20];
  names[0] = "BS portfolio";
  names[1] = "Holding Porfolio";
  names[2] = "HH Payoff";

  yvectors.push_back(bsps);
  yvectors.push_back(hdps);
  yvectors.push_back(hhps);
```

Listing 39: ../source/test/portfTest.c [Line 14 to 95] (continued)

```
    vectorPlotting(yvectors,xvector,names);
  }
  else {
    vector<double> v1,v2;
    double hherror=0,bserror=0;
    for(int i=0;i<hdps.size();i++){
      v1.push_back(bsps[i]-hdps[i]);
      v2.push_back(hhps[i]-hdps[i]);
      bserror += pow(bsps[i]-hdps[i],2.0);
      hherror += pow(hhps[i]-hdps[i],2.0);
    }
    cout<<"Average BSP Error = "<<sqrt(bserror/hdps.size())<<endl;
    cout<<"Average HHP Error = "<<sqrt(hherror/hdps.size())<<endl;
    char *names[20];
    names[0] = "BSP Error";
    names[1] = "HHP Error";
    yvectors.push_back(v1);
    yvectors.push_back(v2);
    vectorPlotting(0,yvectors,xvector,names);
  }
  return 0;
}
```

- Test the case that the portfoliio contains no options. It should give an error message
- Test the case that The porfolio contains only one option.
- Test the case that The porfolio contains two options,one is Eupean Call and another is European Put option.
- Test the case that The porfolio contains arbitrary nuber of options, whih might be Call or Put option.
- Test the case R = 0,1, 10, 2^{j_H-1} and $2^{j\_H} - 1$.

4.2 System Testing

The testing technique we used are Functional testing or called Black box testing. Since this application is developed for research, the expected result is not clear. Professor Ferrando contribute a lot of his time to test the functionality of the system with me.

5 User's guide

5.1 Installation and Compilation

The Software will be distributed by a tar file, name finance.tar. The platform required to run this soft ware include:

1. Hardware must at least Pentium III with 256 MB memory and 1G free hard Drive space
2. Operating system is Linux 7.0 up or Unix.
3. g2 must be installed in the platform. Information about g2 can be found on http:www.ap.univie.ac.at /users/ljubo/g2.shtml.

User should create a working directory for this software first, then type tar -xf finance.tar under this working directory. The Makefile, input files and source files will be generated. Under the working directory, then, user just type make command. The excuable file will be generated and is named HHAssess.

5.2 Running

In order to run the program properly, user need to have two parameter files named parameter.txt and outputops.txt. Both 2 files must be put in the working directory. The meaning of parameters for program are explained in the requirement session. The format of parameters must follow the sample input files supplied below.

1. File:parameter.txt

Listing 40: sampleOutput/parameter.txt [Line 2 to 25]

```
18 0.2 0.2
12 50
1
0  1
1
0  1
1
0 1
100 333
1 1
21.9852 1 1
21.9852 0 1

double St0,interest,sigma;
int j_H,R;
int N;
```

Listing 40: sampleOutput/parameter.txt [Line 2 to 25] (continued)

```
vector<double> times;
int N_H;
vector<double> Htimes;
int N_BS;
vector<double> BStimes;
int M,seed;
callNum putNum
vector<double> SPrice,amount,Stime;
```

2. File:outputops.txt

Listing 41: sampleOutput/outputops.txt [Line 2 to 15]

```
1 0  1

int hedgingMethod, errorOption, double assessTime
HedgingMethod:
0 -- coputing both HH and BS portfolio
1 -- only HH
-1 -- only BS portfolio

errorOption:
0 -- output sample values
1 -- output errors

assesstime:
output the value at the assesstime, greater than t0 and less than T
```

5.3 Concepts

1. holding portfolio: consist of call and put options
2. hedging portfolio: consist of cash on bank account and number of shares of stock
3. rebalance time: The time to operate a Haar hedging or δ Hedging. The rebalance times are listed in Htimes and BSPtimes, which are subsets of the general time.
4. HHAssess: Assesses the Haar Hedging algorithm, and compare its result with δ Hedging.

6 Sample output

We have a huge number of combinations of input, so we can't have the sample output for each possible input. Some typical input combinations are chose and they are supposed to give users visual perception of how the system works and what is the result to run the system.

Each time when HHAssess is runned, 6 files will be generated. The first one is result.ps, which visually shows the result. The second file result.txt contains all the detailed information for the plotting. The other 4 files contains part of the result.txt, such as data for Haar Hedging.

The first part of the result.txt contains the input parameters. I will present all this input data and the plotting ps file for the following typical input and output conditions. I don't present all the output data. There are two exceptions. For the first case, I put partial detail output data, and for the pathwise assessment, I present all the data along the path.

6.1 Final, one Eucall, one step

This means the final time output for one European Call with one step Hedging. This is the most typical and common case for this system. The result of this case can be compared with the work done before.

1. Input parameters:

Listing 42: sampleOutput/1/result.txt [Line 15 to 27]

```
18 0.2 0.2
12 1
1
0 1
1
0 1
1
0 1
300 1
1 1
21.9852 1 1
21.9852 0 1
0 0 1
```

2. Output data:

Listing 43: sampleOutput/1/result.txt [Line 31 to 34]

```
samplePrices  sampleHD  sampleBSP  sampleHH
12.499736    0    -3.369314094    4.897162607e-09
```

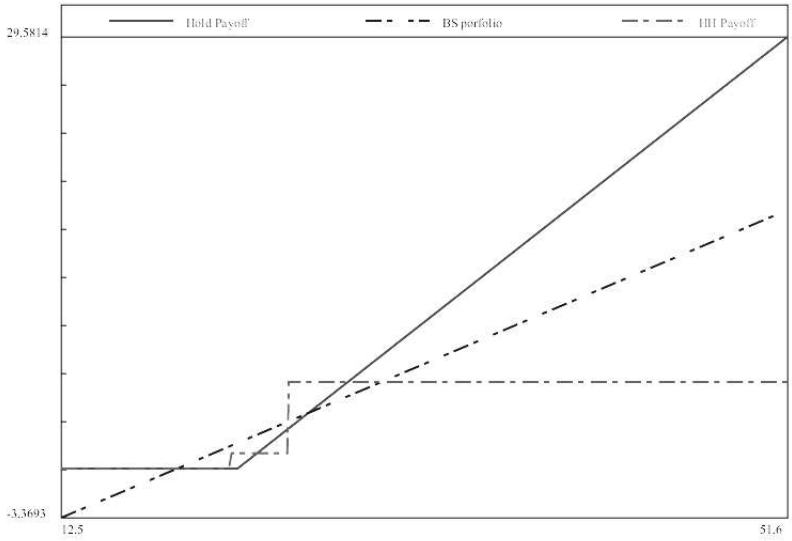

Figure 2: Final, one Eucall, one step

Listing 43: sampleOutput/1/result.txt [Line 31 to 34] (continued)

```
12.5847186    0    -3.323437739    4.897162607e-09
12.71316329    0    -3.254099144    4.897162607e-09
```

3. Continue...

Listing 44: sampleOutput/1/result.txt [Line 326 to 336]

```
31.65891236    9.673712358    6.973428514    5.942693648
32.42117203    10.43597203    7.38492092    5.942693648
33.024691    11.039491    7.710719967    5.942693648
33.15145343    11.16625343    7.779150424    5.942693648
33.21785623    11.23265623    7.814996803    5.942693648
34.96215062    12.97695062    8.756623284    5.942693648
38.08937872    16.10417872    10.44480208    5.942693648
39.80859523    17.82339523    11.37289072    5.942693648
51.56658394    29.58138394    17.72023303    5.942693648
Average BSP error = 1.473737084
Average HH  error = 2.085187971
```

4. Plottig ps file(Figure 2):

6.2 Final, one Eucall, several steps

This means the final time output for one European Call with several steps Hedging. This case is more advanced than the first case, and we have some previous knowledge about the result that is going to achieve. The only difference is that it adds a little bit complexity.

1. Input Parameters:

Listing 45: sampleOutput/2/result.txt [Line 14 to 26]

```
18 0.2 0.2
10 100
10
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
10
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
5
0 0.1 0.2 0.3 0.4 1
300 1
1 1
21.9852 1 1
21.9852 0 1
0 0 1
```

2. Plottig ps file(Figure 3):

6.3 Final, one Eucall, one Euput, several steps

This means the final time output for one European Call and one European Put with several steps Hedging. This case is more advanced than the second case, and we also have some previous knowledge about the result that is going to achieve.

1. Input Parameters:

Listing 46: sampleOutput/3/result.txt [Line 15 to 27]

```
18 0.2 0.2
10 100
10
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
10
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
5
0 0.1 0.2 0.3 0.4 1
300 1
```

Listing 46: sampleOutput/3/result.txt [Line 15 to 27] (continued)

```
1 1
21.9852 1 1
21.9852 1 1
0 0 1
```

2. Plottig ps file(Figure 4):

6.4 Final, three Eucall, four Euput, several steps

This means the final time output for three European Call and four European Put with several steps Hedging. This case is the genearl case for assissing the HH and BSP.

1. Input Parameters:

Listing 47: sampleOutput/4/result.txt [Line 15 to 32]

```
18 0.2 0.2
10 100
10
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
8
0 0.1 0.3 0.4 0.5 0.6 0.8 0.9 1
5
0 0.1 0.2 0.3 0.4 1
100 5
3 4
22 1 1
21 1 1
20 1 1
19 1 1
18 1 1
17 1 1
16 1 1
0 0 1
```

2. Plottig ps file(Figure 5):

6.5 Pathwise,one Eucall, several steps

This means Pathwise comparation for an European Call with several steps Hedging. If a user like to get the pathwise comparation, he can achieve his goal by setting the simulation variable M to 1. Now the parameter seed,right after the M, will be use to generate a special path to test.

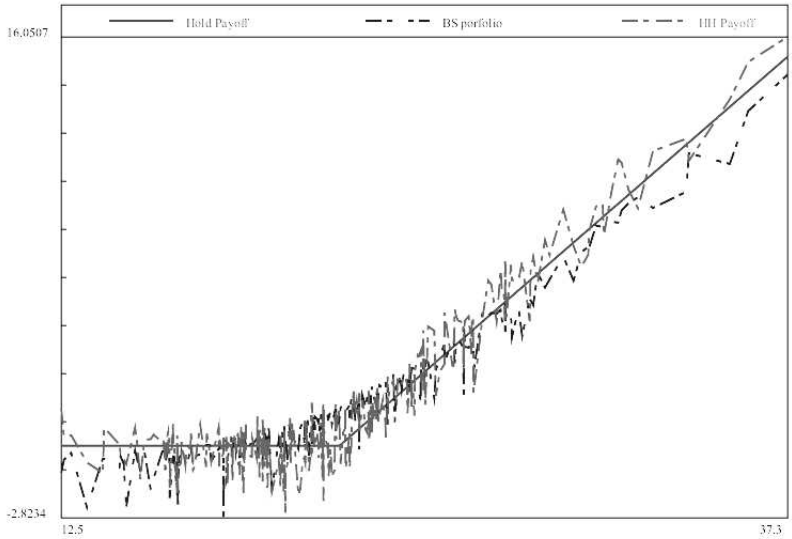

Figure 3: Final, one Eucall, several steps

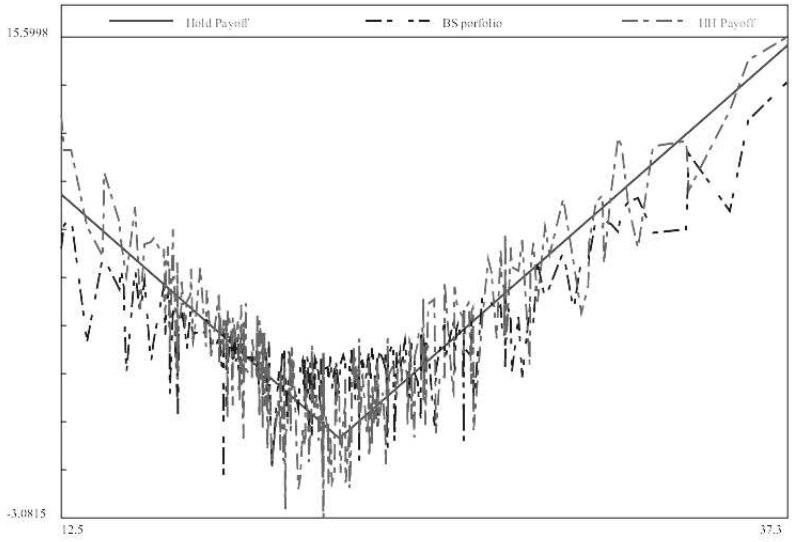

Figure 4: Final, one Eucall, one Euput, several steps

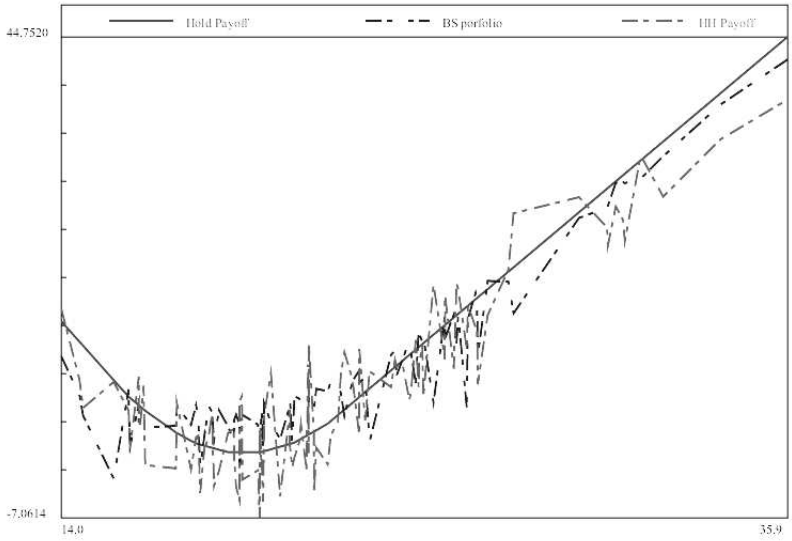

Figure 5: Sample output

1. Input Parameters:

Listing 48: sampleOutput/5/result.txt [Line 14 to 26]

```
18 0.2 0.2
10 100
10
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
8
0 0.1 0.3 0.4 0.5 0.6 0.8 0.9 1
5
0 0.1 0.2 0.3 0.4 1
1 5
1 1
21.9852 1 1
21.9852 0 1
0 0 1
```

2. Output data:

Listing 49: sampleOutput/5/result.txt [Line 30 to 43]

```
samplePrices  sampleHD  sampleBSP  sampleHH
0    1.433820847    1.433820847    1.433820847
0.1    1.031916312    1.075472393    1.461958605
0.2    0.3551616064    0.2582965476    1.729426426
0.3    1.321034317    0.8888572309    1.729426426
0.4    0.5219525353    -0.01868495885    0.7324208736
0.5    0.1810353356    -0.01868495885    0.2631587558
0.6    0.02250787266    -0.01868495885    -0.001526888825
0.7    0.00292128223    -0.01868495885    -0.001526888825
0.8    0.0002134226676    -0.01868495885    -0.1070220127
0.9    3.129062507e-07    -0.01868495885    -0.1129147203
1    0    -0.8165911525    -0.1153869787
Average BSP error = 0.3302788325
Average HH  error = 0.4601903078
```

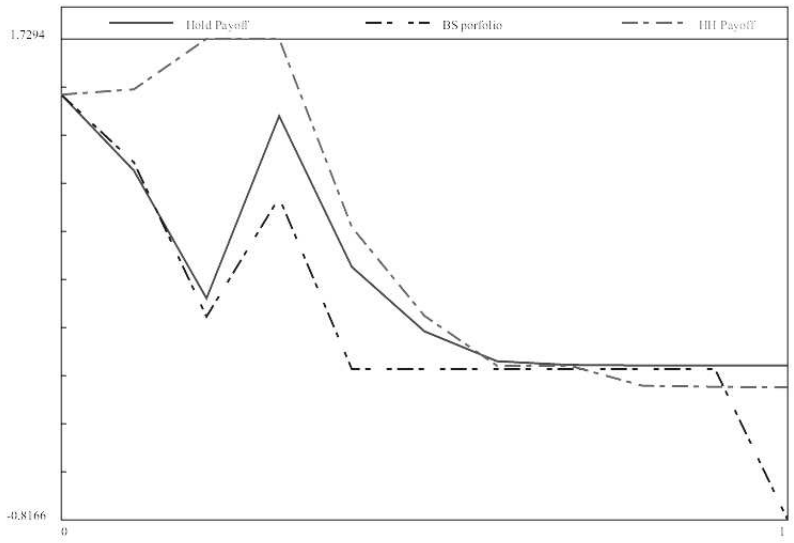

Figure 6: Final, three Eucall, four Euput, several steps

Listing 50: sampleOutput/6/result.txt [Line 14 to 26]

```
18 0.2 0.2
10 100
10
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
8
0 0.1 0.3 0.4 0.5 0.6 0.8 0.9 1
5
0 0.1 0.2 0.3 0.4 1
100 5
1 1
21.9852 1 1
21.9852 0 1
0 0 0.3
```

2. Plottig ps file(Figure 7):

6.7 Final,one Eucall, several steps,HH only,error only

This case is similar to the second case, but it only the error for Haar Hedging. This option can be achieved by put both the hedging method option and error option to 1 in the outputops.txt file. To compute Black & Schole portfolio only, the hedfging method option should be set to -1.

1. Input Parameters:

Listing 51: sampleOutput/7/result.txt [Line 14 to 26]

```
18 0.2 0.2
10 100
10
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
8
0 0.1 0.3 0.4 0.5 0.6 0.8 0.9 1
5
0 0.1 0.2 0.3 0.4 1
```

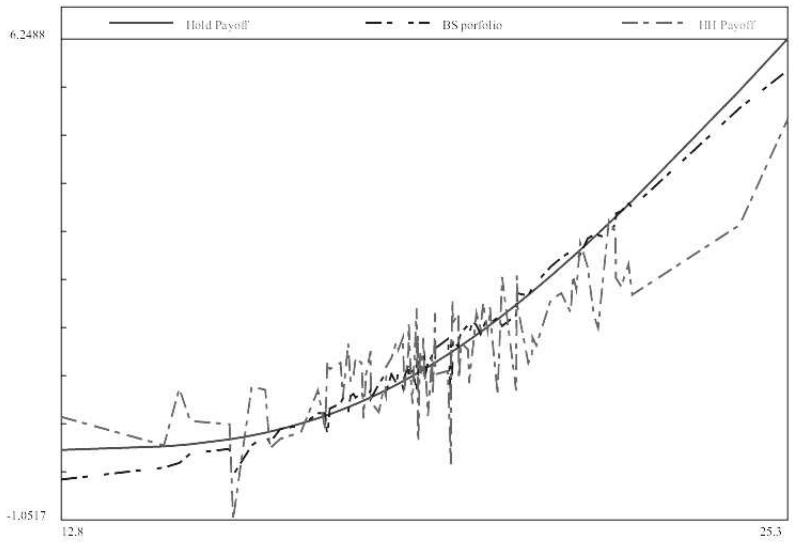

Figure 7: Fixed,one Eucall, several steps

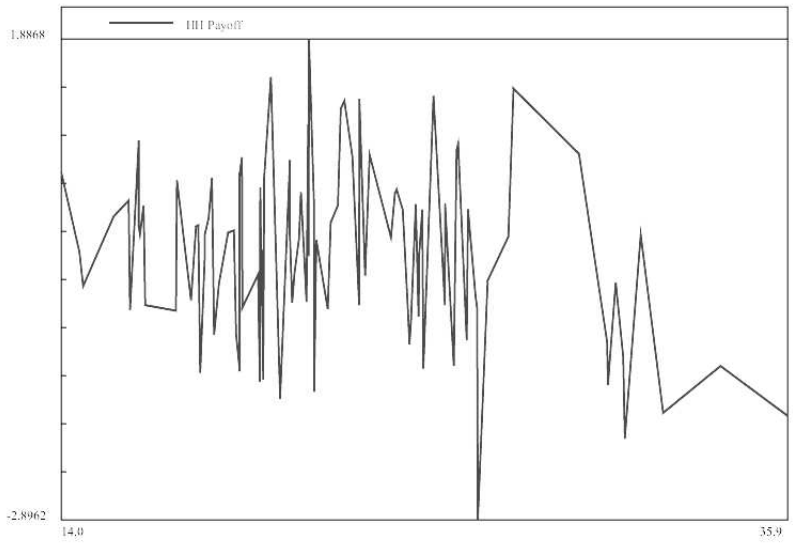

Figure 8: Final,one Eucall, several steps,HH only,error only

References

[1] John C. Hull: Futures and OPtions Markets, Prentice Hall, 2001