ISyE 6767 Project 1 Report

Yuliang Li GTID# 903012703

Project Goal

The project supposes a security with an initial investment cost \$1000 whose value at maturity is, \$1000 when the S&P 500 index is equal or greater than $\frac{1}{2}S_0$ (where S_0 is the initial index), otherwise is $S_M/(\frac{1}{2}S_0)\times\$500$ (where S_M is the index at maturity). The security can be replicated by the combination of 3 kinds of security: 500 long digital call option on the index with strike 840, a short put on the index with strike 840 and a bond which pay \$500 at maturity. The security also pays interests quarterly, which construct a 15-year's cash flow, and is based on a fixed rate in the first year and a float rate after the first year.

Suppose the initial investment is \$1000, in the project, the portfolio is created, and the prices of the digital call option, the put option and the bond. The left cash that is not invested in the above portfolio is also calculated in the project. The current value of cash flow of the security is also calculated in project.

Project Structure

The project includes 7 head files and 8 .cpp files. The structure of the project is like,

Main.cpp: includes the main function of the project and runs the program.

- NormDistIntegral.h and NormDistIntegral.cpp: NormDistIntegral.h is the head file of the NormDistIntegral.cpp. In these files, a class can generate a normal distribution with the specified mean and standard deviation. The member functions of the class can generate the probability density function of the normal distribution. The cumulative distribution function can also be generated. One of the functions calculates the integral from a lower bound to up bound on the generated distribution.

- BSE.h and BSE.cpp: these files prices the European call and put option, as well as the digital call and put option based on the Black-Scholes Model. The function object in the class realizes option pricing.

- BondPrice.h and BondPrice.cpp: the risk-free zero-coupon bond is pricing in these files.

- LinearInterpolation.h and LinearInterpolation.cpp: a function that realizes the linear interpolation for data type of double is included in these files.

- getCMT.h and getCMT..cpp: in these files, the data of CMT, which is downloaded from the website[1] is input into the program. In this project, we need CMT rates of quarter year to 15 years, so the other rates, which are not provided by the Fed, are generated by linear interpolation in these files.

- CashFlow.h and CashFlow.cpp: these files are key files in the project. The current value of the discounted cash flows the security offers is calculated in these files.

- IO.h and IO.cpp: they holds functions used to get input and produce output, include function which gets the data from the form which is downloaded, function which gets the parameters to generate the portfolios, and the functions print vectors to a file.

Input and Output

All the parameters about option and other portfolio are input from InputParameters.txt, include the birth S&P 500 index, the strike index, the barrier index, the risk free rate, the dividend yield and two volatilities.

The monthly risk-free interest rate (CMT) are downloaded from Fed's website http://www.federalreserve.gov/releases/h15/data.htm. We only can get CMT of 3 month, 6 month, 1 year, 2 year, 5 year, 7 year, 10 year and 20 year. So CMT on other date should be interpolated with the former data to get.

30 year swap rate and 2 year swap rate are also downloaded from the Fed's website, and they are used to estimate the floating rate.

The program output 6 .csv files and a .txt file. The .csv files record the value of portfolio when index ranges. The .txt file record the price of the portfolio, the left cash and the current value of cash flow.

Pricing the Portfolios

The solution of Black-Scholes Model provides the methods pricing the European put option and the digital call option. Set p_E is the price of European put option, and

[1] http://www.federalreserve.gov/releases/h15/data.htm Historic data of rates provided by the Fed.

c_D is the price of digital call option. The prices of them are,

$$p_E = Ke^{-r(T-t)}N(-d_2) - S_t e^{-r(T-t)}N(-d_1) \qquad (1)$$

$$c_D = e^{-r(T-t)}N(d_2) \qquad (2)$$

and S_t is the stock price at time *t*, *K* is the strike price, *T* is the expire date, *r* is the risk-free rate. $N(x)$ is the cumulative distribution function of standard normal distribution.

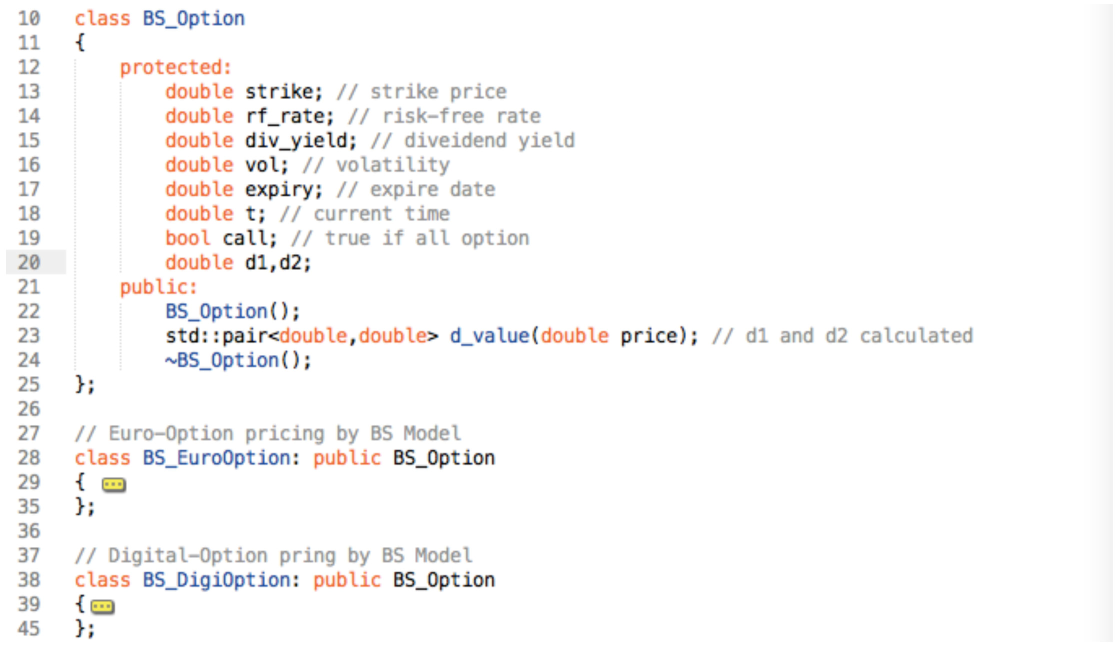
The value of d_1 and d_2 are

$$d_1 = \frac{\ln(S_t / K) + (r - q + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}} \qquad (3)$$

$$d_2 = d_1 - \sigma\sqrt{T-t} \qquad (4)$$

and σ is the volatility, *q* is the dividend yield .

In the project, because the digital call option and European put option share the same parameters like, strike price, risk-free rate, volatility, expire date, d_1 and d_2, etc. So in BSE.h and BSE.cpp, a base class BS_Option is defined, and two derived classes, which generate European option and digital option dividedly are also defined, like the code part from BSE.h below.

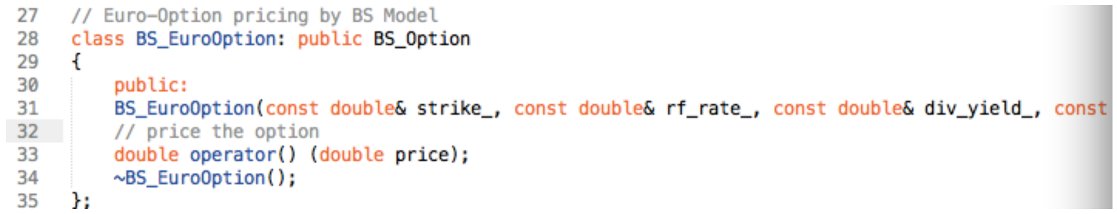


```cpp
class BS_Option
{
    protected:
        double strike; // strike price
        double rf_rate; // risk-free rate
        double div_yield; // diveidend yield
        double vol; // volatility
        double expiry; // expire date
        double t; // current time
        bool call; // true if all option
        double d1,d2;
    public:
        BS_Option();
        std::pair<double,double> d_value(double price); // d1 and d2 calculated
        ~BS_Option();
};

// Euro-Option pricing by BS Model
class BS_EuroOption: public BS_Option
{ ...
};

// Digital-Option pring by BS Model
class BS_DigiOption: public BS_Option
{...
};
```

Function object implements the function that prices the option in each derived class, like the code part from BSE.cpp. Because of the function object, when we call function to price, the only member variable of the function is the spot price of index.

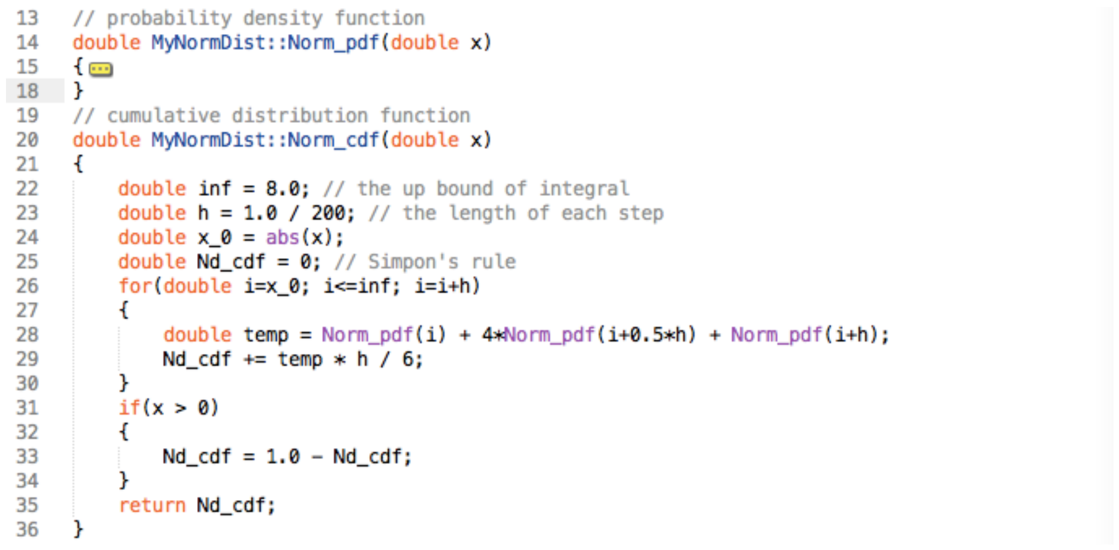


```cpp
// Euro-Option pricing by BS Model
class BS_EuroOption: public BS_Option
{
    public:
    BS_EuroOption(const double& strike_, const double& rf_rate_, const double& div_yield_, const
    // price the option
    double operator() (double price);
    ~BS_EuroOption();
};
```

To obtain the cumulative distribution function of the normal distribution in the Black-Scholes Model solution, in the NormDistIntegral.cpp, Simpson's rule to integral. By Simpson's rule, the integral is the following approximation:

$$\int_a^b f(x)dx \approx \frac{b-a}{6}[f(a)+4(\frac{a+b}{2})+f(b)] \qquad (5)$$

Divide the interval of integral into small parts, and let *a* and *b* be the two endpoints of each part. Apply Simpson's rule in each part, and the sum of integral results will be the integral result on the whole interval. For calculate $N(x)$, based on the character of normal distribution, we can just integral from the abstraction of *x* to the up bound. If *x* is greater or equal to 0, $N(x)$ will be 1 minus the integral result; otherwise, the integral result will be $N(x)$. Like the code part below:



```
// probability density function
double MyNormDist::Norm_pdf(double x)
{
}
// cumulative distribution function
double MyNormDist::Norm_cdf(double x)
{
    double inf = 8.0; // the up bound of integral
    double h = 1.0 / 200; // the length of each step
    double x_0 = abs(x);
    double Nd_cdf = 0; // Simpon's rule
    for(double i=x_0; i<=inf; i=i+h)
    {
        double temp = Norm_pdf(i) + 4*Norm_pdf(i+0.5*h) + Norm_pdf(i+h);
        Nd_cdf += temp * h / 6;
    }
    if(x > 0)
    {
        Nd_cdf = 1.0 - Nd_cdf;
    }
    return Nd_cdf;
}
```

The price of risk-free zero-coupon bond, which can be formulated by

$$B = Pe^{-rT} \tag{6}$$

P is the payment of bond at maturity.

When the index is below $\frac{1}{2}S_0$, the slope of the line of security's value is $\frac{500}{840}$. But the slope of a short put option when it strikes is 1, and this part of line constructs the part of security's value line mentioned formerly. To construct the security, the short put option should multiply a coefficient of $\frac{500}{840}$.

Thus, the left cash would be formulated like,

$$CashLeft = 1000 - (-\frac{500}{840}p_E + 500c_D + B_{500}) \tag{7}$$

and p_E is the unit price of Euro put option, c_D is the unit price of digital call option and B_{500} is the price of risk-free zero-coupon bond which pay \$500 at maturity.

So the prices of the unit portfolio and the left cash that will be invested in some high interest rate security are displayed in the Table. 1[2].

[2] The data can get from Portfolio_price_CF.txt

Table 1 The unit price of portfolio and cash left

Volatility	31%	40%
Unit Price of European Short Option ($)	121.244	193.947
Unit Price of Digital Call Option ($)	0.343	0.260
Price of Risk-free Zero-coupon Bond ($)	313.127	313.127
Cash Left ($)	587.44	672.51

Value of Combined Portfolio

The portfolio consists of 500 digital call option, a short put option and the zero coupon bond. Valuing the portfolio on 5 years and 10 years after portfolio formation needs the forward rates of the 10-year's bond rate on 5th year and the 5-year's bond rate on 10th year.

The forward rate are given by,

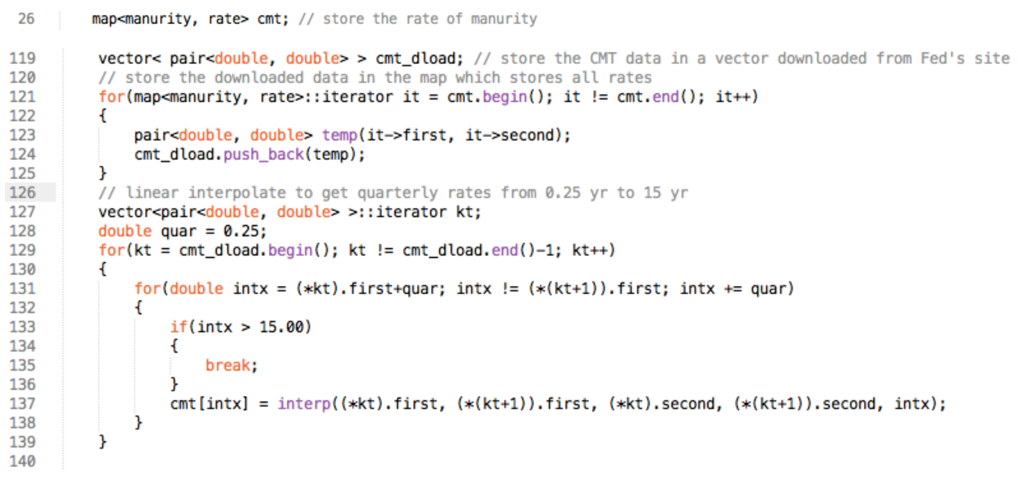
$$ForwardRate = \frac{r_L t_1 - r_s t_2}{t_1 - t_2} 6 \qquad \textbf{(8)}$$

and r_L is the t_1-year's bond's interest rate, r_s is the t_2-year's bond's interest rate. All the bonds' interest rates are risk-free rate.

For the Fed's website has not provided the risk-free rate of 15 years, program generates the risk-free rate of 15 years by linear interpolation. The formulation of interpolation is

$$f(t) = \frac{f(x_1) - f(x_0)}{x_1 - x_0} t + f(x_0) 6 \qquad \textbf{(9)}$$

The program stores all the risk-free interests rates in a map. For there is no operator '+' on map for interpolation, the program generates a vector<pair<double, double> > to store the downloaded interest rates. The rates calculated by interpolation are still stored in the same map mentioned above. The code part is



```
map<manurity, rate> cmt; // store the rate of manurity

vector< pair<double, double> > cmt_dload; // store the CMT data in a vector downloaded from Fed's site
// store the downloaded data in the map which stores all rates
for(map<manurity, rate>::iterator it = cmt.begin(); it != cmt.end(); it++)
{
    pair<double, double> temp(it->first, it->second);
    cmt_dload.push_back(temp);
}
// linear interpolate to get quarterly rates from 0.25 yr to 15 yr
vector<pair<double, double> >::iterator kt;
double quar = 0.25;
for(kt = cmt_dload.begin(); kt != cmt_dload.end()-1; kt++)
{
    for(double intx = (*kt).first+quar; intx != (*(kt+1)).first; intx += quar)
    {
        if(intx > 15.00)
        {
            break;
        }
        cmt[intx] = interp((*kt).first, (*(kt+1)).first, (*kt).second, (*(kt+1)).second, intx);
    }
}
```

After interpolation, the program utilizes the risk-free interest rate (CMT) stored in the map to value the portfolio. Formulation (1), (2) and (6) is still used for price. However, the rate in them has changed. On the date of portfolio, the rate changes to be the CMT of 15 years. 5 and 10 years after formulation, the rate is the forward rate to the 15th year on current year.

Chart. 1 and Chart. 2 show the value of portfolio when the S&P 500 index *S* in the range 700<*S*<1000.

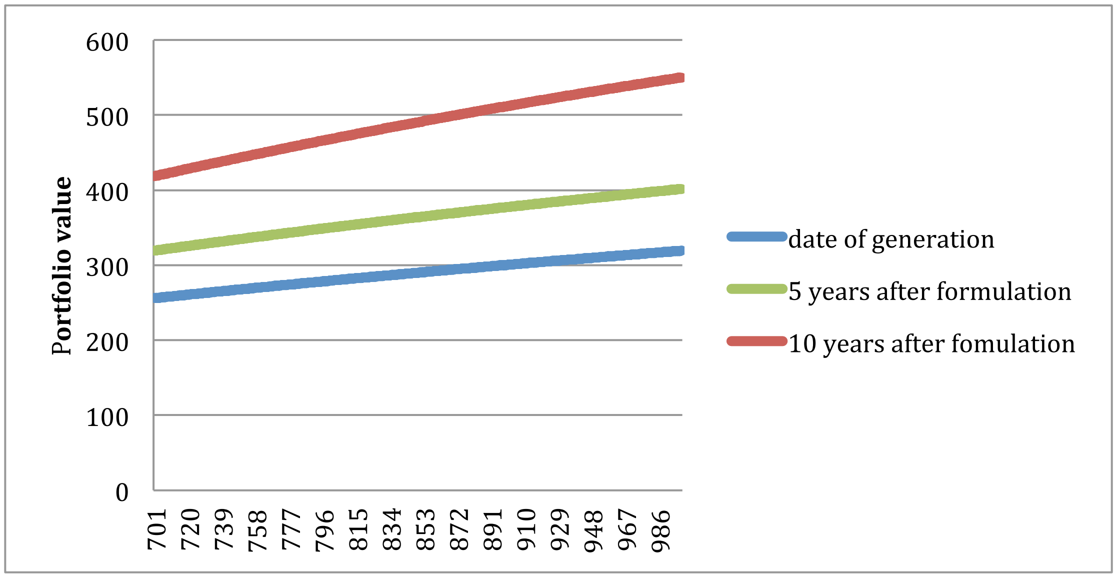


Chart. 1 Portfolio value when volatility is 31% (index ranges from 701 to 999)

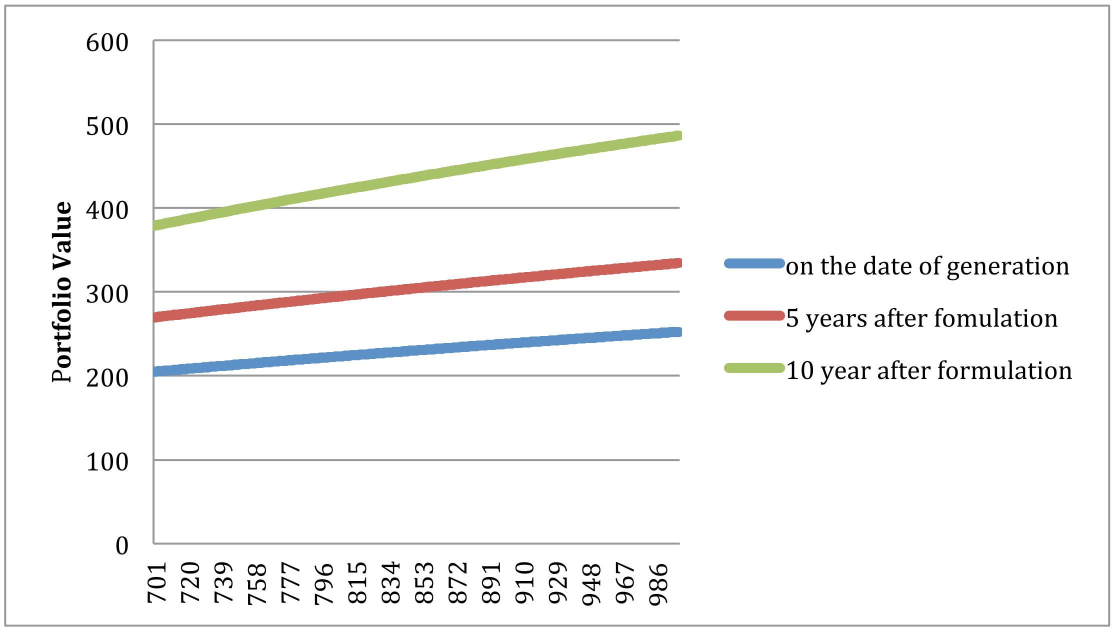


Chart. 2 Portfolio value when volatility is 40% (index ranges from 701 to 999)

Concluded from both charts, as the index increases, the value of portfolio also increases. That is because as the index increases, the short option has less probability to strike while the call option has more probability to strike. The value of a short put and a long call will both increase. As the volatility increases, the value decreases. And as the date approaches the expire date, the value is growing too.

Current Value of Cash Flow

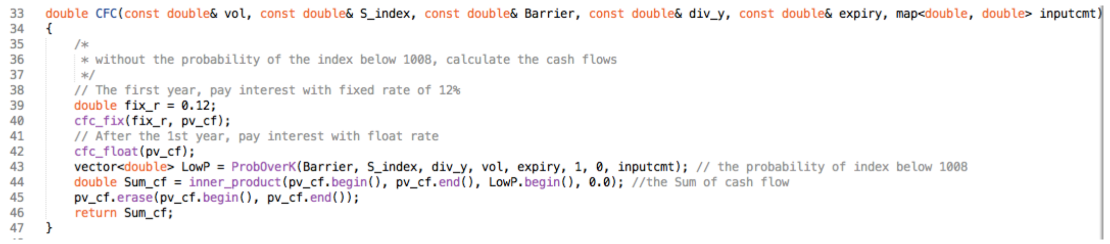
The security pays interests quarterly, and each payment can be discounted to the present by the factor $e^{-0.0544t}$. In the first year, the annual rate is 12%, which is fixed. However, after the 1st year, the rate is float. It is equal to $5(Sw_{30} - Sw_2)$, where Sw_{30} is the 30 year swap rate and Sw_2 is the 2 year swap rate. Because there will be no payment if the index is below the barrier 1008, the actual payment in each quarter of year will be the product of interest payment and the probability that index above 1008. Thus, the current value of cash flow is

$$CurrentCashFlow = 1000\left(\sum_{i=1}^{4}(e^{3\%}-1)e^{-0.0544i/4} + \sum_{i=5}^{60}(e^{5(Sw_{30}-Sw_2)}-1)P_{below}e^{-0.0544i/4}\right) \quad (10)$$

Payment with float rate could be regarded as a call option: when the index is below 1008, there is no payment; otherwise pay the interest. So the probability that the index above 1008 P_{below} can be calculated by $N(d_2)$ based on Black Scholes Model. To get d_2 by formulation (3) and (4), the rate should be the CMT at time payment occurs.

$5(Sw_{30} - Sw_2)$ is the average of the historic data from Fed's website.

In the program, the payment and the probability are stored in two different vectors. To get the current value of payment, the program used the function inner_product in library <numeric>. The code part is



```
double CFC(const double& vol, const double& S_index, const double& Barrier, const double& div_y, const double& expiry, map<double, double> inputcmt)
{
    /*
     * without the probability of the index below 1008, calculate the cash flows
     */
    // The first year, pay interest with fixed rate of 12%
    double fix_r = 0.12;
    cfc_fix(fix_r, pv_cf);
    // After the 1st year, pay interest with float rate
    cfc_float(pv_cf);
    vector<double> LowP = ProbOverK(Barrier, S_index, div_y, vol, expiry, 1, 0, inputcmt); // the probability of index below 1008
    double Sum_cf = inner_product(pv_cf.begin(), pv_cf.end(), LowP.begin(), 0.0); //the Sum of cash flow
    pv_cf.erase(pv_cf.begin(), pv_cf.end());
    return Sum_cf;
}
```

So the current values of cash flows are displayed in the Table 2.

Table 2 Current value of cash flow

Volatility	31%	40%
Current Value of Cash Flow	698.423	601.1