University of Southampton

School of Mathematics

American Options - Valuation by Analytical Approximations

By:

Robert F. Byström

A Project Report Submitted in the Course "Undergraduate Project" MATH3031

Saturday, 28 May 2005

Supervisor: Dr Gerard Kennedy

Table of Contents

Abstract

American options are one of the largest traded derivatives on the financial market. A correct and efficient valuation is of great importance to the market participants. However, valuing most types of American options traded is a complicated task. It is complicated because a stock price, at which an early exercise would be optimal, has to be found as part of the solution to the valuation problem. This is known as a free boundary problem.

This dissertation examines how American options can be valued using an analytical approximation model introduced by G. Barone-Adesi & R. E. Whaley. The model, for simplicity called BAW, value options on assets with a continuous dividend yield or holding cost. The derivation of the model is presented and a Java program is written to perform calculations of option values. A comparison between BAW and a numerical valuation method is presented along with an empirical study of how the model performs against real markets.

1. Introduction

Freedom is a luxury that is very hard to value. How much one would pay for a freedom depends certainly on the sort of freedom, the amount of freedom and perhaps on how much relative freedom one already possesses.

Mainly there are two types of option contracts, European and American. An American option has both a specific sort of freedom and a certain amount of it, a European does not. It is not an easy task to value this extra feature, freedom, as a simple or closed-form solution for this value does not exist.

One could easily argue that the market itself could, or actually does, value these options through supply and demand. But as a mathematician and economist one seeks to find different ways of modeling the real world so that in the future one could try to forecast, in this case, economic variables, financial values and of course option prices.

American vanilla options written on dividend-paying assets have two major concerns that one has to identify and solve in order to be able to value them correctly. First the "extra" characteristic feature of the early exercise privilege. This is commonly referred to as a "free-boundary" problem. Secondly the shift, or jump, in the underlying value due to dividends. There exist several models that approximate these options, some more well known than others, as for example, the Roll model [9], Pseudo-American model [6], and the bi- and trinomial models [6].

In June 1987 such a model was introduced by Giovanni Barone-Adesi & Robert E Whaley [1] for pricing American put and call options not only stocks which could pay dividends, but also contracts such as futures. This model, which will be referred to as the BAW model, or just BAW for simplicity, is a quadratic approximation based on MacMillan's [7] earlier approximation of the American put option on a non-dividend-paying asset. The BAW model is adjusted for a constant dividend yield and lets the user find the "optimal" exercise price of the underlying asset.

The approach used by Giovanni Barone-Adesi & Robert E Whaley has since been further modified. Models that are more efficient and/or produce a more satisfying result on certain types of derivatives and length of maturities have been developed. Nevertheless BAW, with its computational advantage and result accuracy, has made significant contributions to the problem of option valuation. It is claimed to be a very good estimator on options with short- and long-range maturities and can be applied to a variety of derivatives.

It is very important that models such as BAW give results that are satisfactory in both accuracy and efficiency. That is, if a model gives a completely accurate result but takes too large an amount of time to compute, then the result may very well be worthless since the opportunity to use it may have passed. At the extreme, if the amount of time to compute the result is very small but the result is too inaccurate, one would prefer not to use it since one would have little confidence in the result.

This dissertation begins with an in-depth study of how the BAW model is derived, how it works and why. The model is implemented as an option calculator programmed in Java to be used to calculate values efficiently. BAW is run against numerical valuation method CRR [4] to compare results, deviations and efficiencies. Finally an empirical study is undertaken, the objective being to see how well the BAW model performs in context of real markets.

2. The Definition of an Option

An option is a financial derivative contract between two parties to exchange an asset. For a European call option, one party has the option to buy an asset at an agreed strike price, X, at some future expiry date, T. The other party of the contract has the obligation to sell the asset at the agreed time at the agreed price. The buying party is said to have a long position and the selling party is said to have a short position. The selling (short) party, who has the obligation to sell the asset, is exposed to a market risk, i.e. at the predetermined time T the asset might be worth more then the agreed price X. The seller (short) prices this risk and offers the buyer (long) the opportunity to enter into the option contract at this price. This is called the risk premium, ε.

For the buyer the value of this contract at maturity is the option's payoff. The payoff is the assets present spot price, S, minus the agreed price X, or zero, whichever is greatest. This can be shown as

$$V_{call}^{long} = max\left[S - X, 0\right].$$

The buyer's profit from this contract at expiry depends on the asset's spot price, the strike price and the premium paid for the contract. The profit relationships for a long call are described as

$$\Pi_{call}^{long} = \begin{cases} S - X - \varepsilon & if \quad S > X \\ -\varepsilon & if \quad S \le X \end{cases}.$$

Note that S has to be larger than $X + \varepsilon$ in order to profit from the call option.

The specific option described above is known as a European call option. A call is an option contract where the holder of the contract has the right to buy the underlying asset. European has in this context no geographical meaning but is simply referring to the fact that the option can only be exercised at maturity.

There is also another option type, the European put option. Here the buyer (long) of the contract has the right to sell the underlying asset at a specific price at maturity. The party that sells (shorts) this contract then has the obligation to buy the asset at that price.

For the buyer, the value of this contract at maturity is the strike price, X, minus the assets present spot price, S, minus the agreed price X or zero, whichever is greatest. This is defined to be

$$V_{put}^{long} = max\left[X - S, 0\right].$$

The buyer’s profit from this contract at expiry depends on the asset’s spot price, strike price and the premium paid for the contract. The profit relationships for a long put are described as

$$\Pi_{put}^{long} = \begin{cases} X - S - \varepsilon & if \quad S < X \\ -\varepsilon & if \quad S \geq X \end{cases}.$$

Note that X has to be larger than $S + \varepsilon$ in order to profit from the put option. The profit to seller of a call option is described by

$$\Pi_{call}^{short} = \begin{cases} \varepsilon + X - S & if \quad S > X \\ \varepsilon & if \quad S \leq X \end{cases}.$$

and the profit to the seller of a put option is just the reverse, that is

$$\Pi_{put}^{short} = \begin{cases} \varepsilon + S - X & if \quad S < X \\ \varepsilon & if \quad S \geq X \end{cases}.$$

The holder of the option always has the right not to exercise the option and the maximum loss is therefore the premium paid for the contract. The seller of the option, on the other hand, has a maximum loss of $\varepsilon + S - X$ on a put option, as the value of the underlying asset never can be less then zero. For the call option, the seller has a risk of an infinite loss, i.e. as ε and X are constant and as $S \to \infty$, $\Pi_{call}^{short} \to -\infty$. So, the seller has a much riskier position, which would justify the logic behind the premium. (Note that a holder of a put option actually could take on an infinite loss if he was stupid enough or really wanted too.)

As an option gives the holder the right to buy or sell a stock at a certain, predefined price, the value of the option must clearly depend on the value of that stock. It is commonly assumed that stock price undergoes a Markov process and follows a lognormal probability distribution. It is also assumed that trading takes place continuously in time and that interest rates are constant. Markets are supposed to be frictionless, that is, there are no bid/ask spreads, taxes and other limitations on sales, margin requirements, and so on. Markets are competitive, that is, nothing can affect the markets pricing of the asset (for example buying large blocks of stock). Further assumptions are that there are no counterparty risks and no arbitrage opportunities.

Plain vanilla options, such as those described above, have a closed-form solution and are not a problem to value. The model most used for valuing these options is the Black-Scholes equation [3] (BS). Since its introduction in 1973 the model has been modified in several directions but one particular modification was made by Merton [8]. He included parameters for constant dividend yields and this is mostly known as the Constant-Dividend-Yield model (CDY).

The model presented below is the CDY with one adjustment, the cost of carry, b, has been defined as the risk-free interest rate, r, minus the dividend yield, d, i.e. $b = r - d$.

$$V_{call}^{CDY} = \text{Sexp}((b - r)T)\,N(d_1) - \text{Xexp} - (rT)N(d_2) \tag{1}$$

$$V_{put}^{CDY} = \text{Xexp}(-rT)\,N(-d_2) - \text{Sexp}((b - r)T)N(-d_1) \tag{2}$$

Where

$$d_1 = \frac{ln\left(\frac{S}{K}\right) + (b + \sigma^2 / 2)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}\ .$$

With the variables being:

S = stock spot price
X = strike price
T = time remaining until maturity, expressed in years
r = risk-free interest rate per annum
d = annualized dividend yield
b = r-d
σ = annual volatility of stock price (the standard deviation of the short-term returns over one year).
exp = natural logarithm
N(·) = standard normal cumulative distribution function
exp = the exponential function

There are other, perhaps more straightforward approaches to value European options, such as the binomial method introduced by Cox, Ross & Rubenstein [4], but the choice of introducing the CDY and stating it is simple. The CDY gives some fundamental ideas of option pricing equations and is closely related to the BAW model, which will be studied closely later on.

2.2 American options

An American option, in contrast to its European equivalent, can be exercised at any time during its life. This extra characteristic causes the option to be at least as valuable as the European, i.e. an American option equals a European plus an early exercise premium. The problem here is how much this extra freedom is worth compared with the amount of freedom in a European option. This is often referred to as the free-boundary problem.

For the American vanilla call written on a non-dividend-paying asset, it can be shown that the optimal strategy is not to exercise the option early. One should instead keep the option alive by reselling it and here the European price formula (1) will correctly value the American option. For an American vanilla call written on an asset that does pay dividends the same argument can not be made and the problem becomes at once non-trivial. There is always a possibility of early exercise.

In order to value the American option written on a dividend-paying asset one must first identify the characteristics of this type of option.

At each instance of time during the option's life there is a stock price that serves as a borderline. On one side of that border it would be optimal to exercise the option early and on the other side it would be optimal to keep the option alive.

The valuation problem of American vanilla options has no closed-form solution (accept for the case of an American vanilla option written on an underlying asset that pays one discrete dividend). So what one does is to use models that approximate the option's value. Such a model is the BAW model.

3. The BAW model

Now that the basics have been stated, one must take a closer look at the model, piece by piece. In this section the BAW model is presented followed by a derivation of it with explanations of the different steps.

3.1 *The Barone-Adesi & Whaley model*

The model (BAW) introduced by Barone-Adesi & Whaley is

$$V_{call}^{BAW} = \begin{cases} c(S,T) + (A_1(S/S^*)^{q_2} & if \quad S < S^* \\ S - X & if \quad S \geq S^* \end{cases}$$

With

$$M = \frac{2r}{\sigma^2}, \qquad N = \frac{2b}{\sigma^2}, \qquad K(T) = 1 - exp(-rT)$$

$$A_2 = \frac{S^*}{S}\left(1 - exp((b-r)T)N(d_1(S^*))\right)$$

$$q_2 = \frac{1}{2}\left[-(N-1) + \sqrt{(N-1)^2 + \frac{4M}{K}}\right]$$

S^* is the asset price which solves:

$$S^* - X = c(S^*, T) + \frac{1 - \exp((b-r)T)N(d_1(S^*))S^*}{q_2}$$

3.2 *Derivation of the BAW model*

To start from the beginning we state the well known partial differential equation that describes the value of an option:

$$\frac{1}{2}\sigma^2 S^2 V_{SS} + bSV_S - rV + V_t = 0 \tag{3}$$

This is just the ordinary BS equation with some adjustments to the basic assumptions of the model. Basically we are allowing dividends on the underlying asset and assume they are constant yields (d). A dividend yield is being defined as the dividend per share divided by the stock price. An asset's cost-of-carry (b) is the risk-free rate-of-return minus the annualised dividend yield (b = r-d). When d = 0 and b = r this is the ordinary BS formula and when r = d and b = 0 this is Black's [2] model for options on futures.

The early exercise premium is defined to be

$$\varepsilon_c(S,T) = C(S,T) - c(S,T)\,, \tag{4}$$

where C(S,T) is the American option value and c(S,T) is the European option value as in (1). The fundamental point here is that the value of an American option must be equal to a European plus a premium for the extra feature, an open strike-date. Now, letting time evolve backwards from the time of the maturity date, t^*, to a present time t. Then the time to maturity T is defined as $T = t^* - t$, and the premium's rate of change with respect to time has the equality $\varepsilon_T = -\varepsilon_t$. Applying this fact on (3) gives the partial differential equation for the early exercise premium

$$\frac{1}{2}\sigma^2 S^2 \varepsilon_{SS} + bS\varepsilon_S - r\varepsilon + \varepsilon_t = 0\,. \tag{5}$$

Now multiplying this last equation by $\frac{2}{\sigma^2}$ results in

$$S^2\varepsilon_{SS} + \frac{2bS\varepsilon_S}{\sigma^2} - \frac{2r\varepsilon}{\sigma^2} + \frac{2\varepsilon_t}{\sigma^2} = 0\,.$$

Then gathering terms and defining them as $M = \frac{2r}{\sigma^2}$, $N = \frac{2b}{\sigma^2}$ shows

$$S^2\varepsilon_{SS} - M_\varepsilon + NS\varepsilon_S - \left(\frac{M}{r}\right)\varepsilon_T = 0\,. \tag{6}$$

Then Barone-Adesi & Whaley rewrite the early exercise premium as $\varepsilon_c(S,K) = K(T)\,f(S,K)$, which express this as a function of the time to maturity and the stock price. It follows that $\varepsilon_{SS} = Kf_{SS}$ and $\varepsilon_T = K_T\,f + KK_T\,f_K$. Substituting these into above equation (6), and by gathering terms and factorisation it shows that

$$S^2 f_{SS} + NSf_S - Mf\left[1 + \left(\frac{K_T}{rK}\right)\left(1 + \frac{Kf_K}{f}\right)\right] = 0\,. \tag{7}$$

Then choosing $K(T) = 1 - \exp(-rT)$ and simplifying (7) gives a nicer equation,

$$S^2 f_{SS} + NSf_S - \frac{M}{K}f - (1-K)Mf_K = 0\,. \tag{8}$$

So far, there has not been any approximation made and therefore this is still an exact analysis. But this will end here. Now look at the last part of the LHS of equation (8),

$(1-K)Mf_K$. By letting T tend to zero, f_K tends to zero, and if T tends to infinity, K tends to one. The two approaches should justify that an approximation can be made here, and hence by eliminating this last term, the remaining equation is a neat-looking, second-order, ordinary differential equation,

$$S^2 f_{SS} + NSf_S - \frac{M}{K} f = 0 . \tag{9}$$

Equation (9) looks to be an Euler equation and has linear solutions of the form aS^q which we can find by letting $f = aS^q$ and substituting into (9) shows

$$aS^q \left[q^2 + (N-1)q - \frac{M}{K} \right] = 0 . \tag{10}$$

By isolating q we can find its two roots,

$$q_1 = \frac{-(N\text{-}1) - \sqrt{(N\text{-}1)^2 + \frac{4M}{K}}}{2}$$

and

$$q_2 = \frac{-(N\text{-}1) + \sqrt{(N\text{-}1)^2 + \frac{4M}{K}}}{2} .$$

Note here that as $\frac{M}{K} > 0$, $q_1 < 0$ and $q_2 > 0$. The general solution to (10) is

$$f(S) = a_1 S^{q_1} + a_2 S^{q_2} . \tag{11}$$

Now, to some tricky business, that is recognising and setting boundary conditions (constraints) on the formula. Since $q_1 < 0$, it follows that $f(S) \to \pm\infty$ as $S \to 0$. This is not very reasonable as, logically, it is not worth paying something extra for something that is not worth anything and never can be anything worth. Generally, if S tends to zero so must the early exercise premium. Therefore the constraint $a_1 = 0$ is imposed so that the part $a_2 S^{q2}$ never can approach $\pm\infty$. Now the formula can be rewritten as

$$C(S,T) = c(S,T) + Ka_2 S^{q_2} . \tag{12}$$

The third and last constraint is imposed on a_2, the more tricky “free-boundary” problem. We begin by examining equation (12). If $S = 0$ so must $C(S,T)$ since both parts on the RHS are equal to zero. Assuming a_2 to be larger then zero, then as S grows, so does $C(S,T)$ which should approach S-X. Substitute $C(S,T)$ with its conditioning, S^*, which is the new, higher stock price,

$$S^* - X = c(S^*,T) + Ka_2 S^{*q_2} . \tag{13}$$

S^* can be explained as an optimal stock price that, if breached, would trigger an early exercise of the option. But this is not enough; below the “optimal” stock price S^* the value of the American call is represented by (12) but should never intersect the boundary S-X. This is done by differentiating the above equation (13) so that the slopes are equal (point of tangency), we obtain;

$$1 = \exp((b-r)T)N(d_1(S^*)) + Kq_2 a_2 S^{*q_2 - 1} \tag{14}$$

with

$$\exp((b-r)T)N(d_1(S^*)) \tag{15}$$

being the partial derivative of $c(S^*,T)$ with respect to S^* and where

$$d_1(S^*) = \frac{\ln\frac{S^*}{X} + \left(b + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} . \tag{16}$$

This leaves us with two equations, (13) and (14), and two unknowns, a_2 and S^*. Rearranging equation (14) with respect to a_2 it can be expressed as

$$a_2 = \frac{1 - \exp((b-r)T)N(d_1(S^*))}{Kq_2 S^{*q_2 - 1}} . \tag{17}$$

Substituting (17) into (13) and simplifying for S^* shows that

$$S^* - X = c(S^*,T) + \left[1 - \exp((b-r)T)N(d_1(S^*))\right] \frac{S^*}{q_2} . \tag{18}$$

Here S^* is the only unknown but its value has to be found through iterations. (One way to do this is included in the source files of the Java applet.)

With S^* known, equation (13) provides the value of a_2. Then by substituting equation (17) back into (12) results in;

$$V_{Call}^{BAW} = \begin{cases} c(\mathrm{S,T}) + A_2(\mathrm{S/S}^*)^{q_2} & if \quad S < S^* \\ S - X & if \quad S \geq S^* \end{cases}, \tag{19}$$

where

$$A_2 = \frac{S^*}{S}\left(1 - exp((\mathrm{b - r})T)N(d_1(S^*))\right) \tag{20}$$

3.3 ***BAW for put options***

The derivation for puts follows the same logic pattern as for calls. The general equation (3) holds to be true also for puts, so it also applies to the early exercise premium and can hence be defined as

$$\varepsilon_p(S,T) = P(S,T) - p(S,T)\,. \tag{21}$$

The implementations and deductions made from equations (4) to (11) is exactly the same for the put. Recall that equation (11) is

$$f(S) = a_1 S^{q_1} + a_2 S^{q_2}\,.$$

For puts it must be true that as the asset's spot price (S) tends to zero, so must the early exercise premium (ε_p). The second part of the RHS of equation (11) does not hold for this argument, so by imposing that $a_2 = 0$, the last term vanishes and the remainders are

$$P(S,T) = p(S,T) + a_1 S^{q_1}\,. \tag{22}$$

Once again there is only two unknowns remaining, a_1 and the critical asset price, which for puts is denoted by S^{**}. Following the same steps as for calls it is concluded that

$$a_1 = -\frac{1 - \exp((b-r)T)N(-d_1(S^{**}))}{Kq_1 S^{**q_1 - 1}}, \tag{23}$$

Where

$$-\exp((b-r)T)N(-d_1(S^{**}))$$

is the partial derivative of $p(S^{**}, T)$ with respect to the critical asset price S^{**}. This is determined by solving

$$X - S^{**} = p(S,T) - \left[1 - \exp((b-r)T) N(-d_1(S^{**}))\right] \frac{S^{**}}{q_1}. \tag{24}$$

Now, with S^{**} as the only unknown, the approximate value of an American option can be written as;

$$V_{Put}^{BAW} = \begin{cases} p(\mathrm{S,T}) + A_1 (\mathrm{S/S}^{**})^{q_1} & if \quad S > S^{**} \\ S - X & if \quad S \le S^{**} \end{cases}, \tag{25}$$

where

$$A_1 = -\frac{S^{**}}{q_1} \left[1 - \exp((b-r)T) N(-d_1(S^{**}))\right]. \tag{26}$$

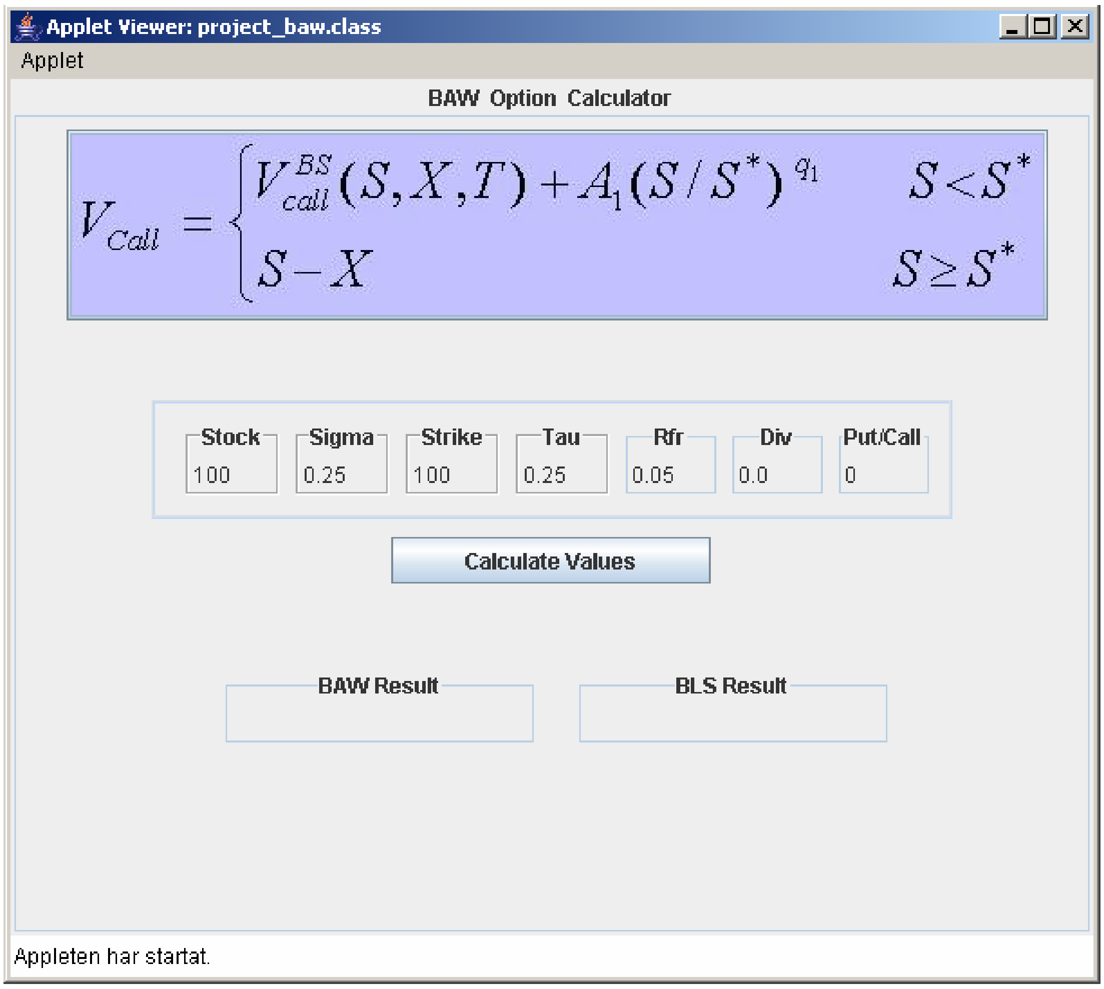
This ends the derivation of the BAW-formula. A Java program has been written to find the critical values for the optimal early exercise price. The source code to this program is attached in the appendix and a brief description of the user interface follows in this dissertation.

4. BAW Option Calculator

The BAW model has been implemented in Java using the freeware RealJ and realised as an applet. This applet, called the BAW Option Calculator, gives its user an efficient way to value American vanilla type options. This section only briefly describes how to operate the BAW Option Calculator.

The BAW Option Calculator can be operated on any platform such as Windows, Linux or Apple. It can be run as an internet application or as a standalone program.

This is how it is displayed in Windows once started.



$$V_{Call} = \begin{cases} V_{call}^{BS}(S,X,T) + A_1(S/S^*)^{q_1} & S < S^* \\ S - X & S \geq S^* \end{cases}$$

Stock	Sigma	Strike	Tau	Rfr	Div	Put/Call
100	0.25	100	0.25	0.05	0.0	0

The calculator has three important parts; the input area, the execution area and the output area.

The input area lets the user set the preferred parameter values. The parameters that could be used in the calculator are:

Stock:	the stock price of the underlying asset.
Sigma:	the annual volatility of the asset
Strike:	the options strike price
Tau:	the time to maturity expressed in years
Rfr:	the interest rate per annum
Div:	the annualised dividend yield
Put/Call:	decides whether the option is a put or a call, 0 = call, 1 = put.

Once these values have been set, the user should press the "Calculate Values"-button. The resulting values are displayed beneath this button. To the left is the value of an option calculated using the BAW method and to the right the value of its European equivalent is displayed.

The results are displayed with an unlimited amount of characters. If not all can be seen the user can put the marker inside the field and move to the left to reveal all characters.

Every field and function in the program has help-text available and is shown when the mouse cursor is placed over a specific field.

The BAW Option Calculator has other features built in to it. It comes with API documentation and for further information regarding the program this documentation or the source code should be consulted.

5. Studies

A series of studies has been made to give an idea of how the BAW model is performing. First BAW is compared with data computed with the binomial model with 200 steps (BIN200). Then option values computed with the BAW Option Calculator are compared with real market data.

5.1 BAW against Binomial model – a comparison

The idea here is to make it possible to detect deviations and unexpected behaviour of BAW. In order to do so a reliable model to compare with is needed. For this purpose, the Binomial model seems to be a good model to use since it is a rather simple model to understand and that its accuracy is satisfying. The Binomial model converges toward the BS model, which fairly value an option, after about 150 time increments (steps). The only negative part is the computational time needed. For the comparison with BAW the binomial model with 200 time increments are used. The notation for this method will be BIN200.

The variables are chosen so that a satisfying data range could be obtained even though the amount of parameters is small. The BIN200 values have been computed using the Matlab 6.5 function binprice and for computation of Baw values the BAW Option Calculator was used. The source code to the BAW Option Calculator and the Matlab m-file for BIN200 are omitted in the appendix.

The total amount of option values simulated was 3072 (768 values respectively per model and option type) which gives a total of 1536 data sets. BAW and BIN200 are both computed with the following set of parameters:

Parameter	Values
S	230, 240, 250, 260
R	1%, 2%, 3%, 4%
T	10 Days, 90 Days, 360 Days
Sigma	15%, 20%, 25%, 30%
Div	0%, 0,01%, 1%, 3%
Flag	Call, Put
Strike	240

For comparison purposes the RMS (Root Mean Squared) has been used and are defined as

$$RMS = \sqrt{\frac{1}{m}\sum_{i=1}^{m} e_i^2}$$

where e_i is the relative error of each observation calculated as

$$e_i = \frac{BAW_i - BIN200_i}{BIN200_i}.$$

There are perhaps other methods that can be used but this method is known to the author and is suitable since it doesn't over or under penalizes deviations. A summary of calculated RMS values are presented here below.

	Total	Calls	Puts
RMS:	4,096960824	0,008037585	5,793971987

For call options BAW is doing rather well in comparison with BIN200. The RMS is 0,008037585 over a sample size of 1536 values and the relative errors are scattered somewhat equally over the sample space. The author should therefore not determine any behaviour nor make any statements without closer studying the data.

Below is a summary of statistics from the data:

BIN200 Calls		*BAW Calls*	
Mean value	16,73199388	Mean value	16,79659159
Standard error	0,387263132	Standard error	0,389541577
Median value	16,6208	Median value	16,6870394
Mode	20	Mode	20
Standard deviation	10,73215072	Standard deviation	10,79529284
Sample Variance	115,1790591	Sample Variance	116,5383474
Kurtosis	-0,540961948	Kurtosis	-0,541203731
Skewness	0,352009836	Skewness	0,356942978
Range	46,1359	Range	46,13871666
Minimum	0,1002	Minimum	0,100918691
Maximum	46,2361	Maximum	46,23963535
Sum	12850,1713	Sum	12899,78234
Amount	768	Amount	768
Confidence level (95,0%)	0,760221716	Confidence level (95,0%)	0,764694445

When the comparison is done for put options the findings are quite different. Now the RMS is 5,793971987, which is an unexpected and a very large value. It should be noted that no manipulations have been done to the data. If extreme data points (4 values) were to be discarded the total RMS value would decline to 0,3816950, which still is a high value. Overlooking all the extreme data points it can be seen that BAW generally is overvaluing the option and especially for options that are near maturity and deep out-of-the-money.

An odd behaviour was discovered during the comparison for puts. With the parameters set to;

Parameter	Stock	Strike	T	Sigma	r	Div	Flag
Value	260	240	90 Days	20%	4%	0%	Put

the option's value according to BAW is 17,62216138. If the value of r is increased or decreased with only a fraction (say to 3,9999999999 or 4,0000000001) the option's value is about 2,1235, a value very close to the corresponding BIN200 value. The same phenomena is found for one more set of parameters and together the two abbreviations causes a relative error of 166,066425 which then squared causes the RMS to go through the roof.

As for the call option more in-depth studies o the results are necessary in order to draw further conclusions. Below is a chart of statistics from the data obtained for puts.

BIN200 Puts		*BAW Puts*	
Mean value	10,37387375	Mean value	11,15366454
Standard error	0,294912084	Standard error	0,311384018
Median value	10,0122	Median value	10,83542262
Mode	0,0011	Mode	6,825077707
Standard deviation	8,178162533	Standard deviation	8,634943264
Sample Variance	66,88234242	Sample Variance	74,56224517
Kurtosis	-0,351225539	Kurtosis	-0,358333188
Skew ness	0,62587818	Skew ness	0,588352653
Range	35,23406188	Range	38,2930968
Minimum	0,00093812	Minimum	0,003355083
Maximum	35,235	Maximum	38,29645188
Sum	7977,508915	Sum	8577,168028
Amount	769	Amount	769
Confidence level (95,0%)	0,578929499	Confidence level (95,0%)	0,61126486

Computational times have not been measured. The main two reasons are that the methods have been run in different programs, which make a time measurement hard to do, and that the author is not good enough as a programmer. Even so it would seem undisputable that the analytical approximations by BAW would need just a fraction of computation effort compared with BIN200. BIN200 loops the calculations 200 times and within each loop boundary conditions must be applied to each node in the binomial tree. BAW has a different approach and the only process that could slow calculations down is the loop that finds the critical stock prices, even if this usually is done within five to ten iterations.

5.2 BAW Compared with Real Market Data

Options were chosen from Stockholm exchange (Stockholmsbörsen), a Swedish exchange located in Stockholm. The options in hand are written on stocks that have been somewhat stable over time and are paying dividends at a level common for large mature companies in the market. The options are further written on stocks of companies that are well known, which perhaps help getting a feeling of what one is trying to value.

The following results were made through the study:

(H&M) Hennes & Mauritz AB, serie B			Stock Spot Price as of 20/5-2005:		257,5
(All values are in SEK)			**Dividend yield:**		3,50%
	Strike Price:	**Middle Price:**	**Implied Volatility:**	**Time to maturity:**	**BAW value:**
July 05 Calls	240	22,375	29,520%	63 Days	22,394
	250	13,125	22,120%	63 Days	13,136
	260	7,25	20,320%	63 Days	7,26
	270	3,35	18,980%	63 Days	3,35
	280	1,325	18,280%	63 Days	1,326
July 05 Puts	240	1,1	29,520%	63 Days	6,1
	250	2,975	22,120%	63 Days	9,192
	260	6,75	20,320%	63 Days	10,39
	270	12,875	18,980%	63 Days	16,454
	280	21,5	18,290%	63 Days	24,4

	Call	Put	Total
RMS:	0,000882675	2,254527475	1,594191788

Volvo AB, series B			Stock Spot Price as of 20/5-2005:		295
(All values are in SEK)			**Dividend yield:**		4,22%
	Strike Price:	**Middle Price:**	**Implied Volatility:**	**Time to maturity:**	**BAW value:**
July 05 Calls	270	27,625	26,070%	63 Days	27,625
	280	19,192	22,870%	63 Days	19,337
	290	12,125	20,750%	63 Days	12,135
	300	6,875	19,480%	63 Days	6,88
	310	3,5	18,800%	63 Days	3,51
July 05 Puts	270	1,125	26,070%	63 Days	4,147
	280	1,675	22,870%	63 Days	5,766
	290	5,75	20,750%	63 Days	15,1
	300	10,5	19,480%	63 Days	12,98
	310	17,25	18,800%	63 Days	19,56

	Call	Put	Total
RMS:	0,003645653	1,783195261	1,260912096

Autoliv SDB INC			**Stock Spot Price** as of 20/5-2005:		342,5
(All values are in SEK)			**Dividend yield:**		1,244%
	Strike Price:	**Middle Price:**	**Implied Volatility:**	**Time to maturity:**	**BAW value:**
June 05Calls	310	34,625	28,71%	35 Days	34,627
	330	17,375	23,21%	35 Days	17,394
	350	6,125	21,60%	35 Days	6,126
	370	1,525	21,56%	35 Days	1,525
June 05 Puts	310	0,6	28,71%	35 Days	2,128
	330	3,375	23,21%	35 Days	5,0485
	350	12	21,60%	35 Days	13,325
	270	27,625	21,56%	35 Days	28,684

	Call	Put	Total
RMS:	0,000553577	1,298560893	0,918221297

For this comparison most of the data have been data been collected from the Stockholm exchange and are closing prices and rates as of May 20, 2005. The only data obtained elsewhere is the dividend yields which are retrieved from the Swedish financial newspaper Veckans Affärer's webpage. Options on the Stockholm Exchange expire on the third Friday in the settlement month. To express time to maturity in years a year has been assumed to have 360 days. Interest rates used are 1 and 2 month annualized fixed STIBOR rates. Implied volatility has been calculated with the BS model using a maximum of 2000 iterations and a tolerance factor for convergence of 10^{-6}. Baw values have been found using the BAW Option Calculator. The options are of American style and middle prices have been calculated as $(Bid + Ask)/2$.

It is worth pointing out that the fair price of an option lies within the bid-ask spread but typically not in the middle. There is often a skewed relationship between the bid-ask spread and the fair value which can be hard to find. Nevertheless, the average of them both creates a reliable tool for comparison purposes.

Though this study it is found that BAW correctly values American style call options (at least with a time maturity around 35 and 60 days). The values found for put options are, on the contrary, not at all satisfying. The BAW Option Calculator value these options in a way that makes the author suspicious. Remembering that in the comparison between BAW and BIN200 there were discrepancies for the put option, it would now be legitimate to say that these probably depend on a programming error.

6 *Reflections and Comments*

Throughout the derivation of the BAW model some thoughts and concerns arises to the author. To begin with, it should be said that the BAW model handles options on dividend-paying assets as well as non-dividend-paying assets. As there is no dividend yield, the cost-of carry equals the annual interest rate, and a call option will be valued using Merton's formula (1), and a put will be valued by MacMillan's formula [7] which the BAW model is built upon.

The models main concern is to find the critical value of the stock price where an early exercise would be optimal. It does so in a fast and rather accurate manner. But the BAW model does not consider any of the rules set by the different exchanges regarding when the underlying asset has to be held in order to get the accruals from dividends paid out. In fact for most options on dividend-paying assets it would be optimal with an early exercise at a time very close to the dividend date. The problem is that the so called cum-dividend date often lies several days, or even weeks, before the ex-dividend date. So even if a model like BAW finds a theoretical optimal exercise price it may not be possible to take advantage of it due to trading and other regulations. This is a major concern not only for the BAW model but for all models as the author has not yet seen one that brings this issue into the light.

Further, by making the approximation and transforming equation (8) into (9) by letting T tend to zero or infinity, one would assume the BAW model to behave rather well on options with short or long maturities. In fact most American options are short-dated. Options on futures often have long maturities and it can be shown that the BAW model can be used to value these as well. But the justification for this approximation becomes harder when it comes to middle-ranged derivatives.

It is common in the world of finance to annualise yields as it provides uniform comparisons. Making the assumption that dividend yields are annualised and valuing options written on assets with dividends that in reality are not annualised, may cause some problems. One could of course argue that interest rate undergoes the same logics and that the model is not that sensitive to the level of interest rate, the cost-of-carry and hence the dividend yields. This argument seems vague and one should consider this when using the BAW model.

Dividends coming from indexes consisting of stocks that pay dividends could be considered as annualised dividends. This is often inaccurately used to further support the use of annualised dividend yields in the BAW model. In order to be a valid statement, options on indexes have to be American. This is certainly not the case, as, so far, the author has not yet seen an American index option.

It has been shown that BAW does not have the same accuracy in its valuation as BIN200. Through the empirical study it is shown (for calls) that the results are reflecting the market values and that BAW could be used in a real environment.

References

[1] G. Barone-Adesi & R. E. Whaley, *Efficient Analytic Approximation of American Option Values*, The Journal of Finance, No 2 (June 1987), 301-320

[2] F. Black, *Valuation of American Futures Options: Theory and Empirical Tests, Journal of Finance*, 41 (March 1986), 127-150

[3] F. Black & M. Scholes, *The Pricing of Options and Corporate Liabilities*, Journal of Political Economy, 81 (May-June 1973), 637-659

[4] J. Cox, S. Ross and M. Rubinstein, *Option Pricing: A simplified approach*, Journal of Financial Economics, 7 (1979), 229-263

[5] Dr C. Hoffman, *Valuation of American Options*, University of Oxford, 2000

[6] Jarrow and Turnbull, *Derivative Securities*, 2nd Edition, South-Western College, Ohio 2000

[7] L. MacMillan, *Analytical Approximation for the American Put Option*, Advances in Futures and Options Research, 5 (1985), 119-139

[8] R. C. Merton, *Theory of Rational Option Pricing*, Bell Journal of Economics and Management Science 4 (Spring 1973), 141-183

[9] R. Roll, *An Analytic Valuation Formula for Unprotected American Call Option on Stocks with Known Dividends*, Journal of Financial Economies, 5 (1977), 251-258

[10] http://www.stockholmsborsen.se, May 21 2005

[11] http://java.sun.com, April 25 2005

[12] http://finance.bi.no, April 28 2005

Affärsvärlden's hemsida !! etc

Appendix

A. Java Source Code

```
/**
 * @(#) project_baw.java  Version: 1.0 05/05/23
 *
 * This program has been written by
 * Robert Byström, Högsätrav 30, 181 58  Lidingö.
 *
 * The program is written as a part in the course
 * "Mathematics Project" (MATH3031) at the University of Southampton
 * with Dr Gerard Kennedy as mentor.
 *
 * All Rights Reserved.
 */

/**
 * The project class is the programs main class whitch provides
methods
 * and instances to the programs interface.
 *
 * @version 1.0 23 May 2005
 * @author  Robert Byström
 */

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.lang.Object.*;
import java.awt.Cursor.*;
import java.text.*;
import java.applet.*;
import javax.swing.*;
import javax.swing.AbstractButton.*;
import javax.swing.border.*;
import javax.swing.JTextField;
import javax.swing.JTextField.*;


public class project_baw extends JApplet implements ActionListener {

/** This declares the varible s0, the initial value of the
* underlying asset. */

   private double s0 = 100;

/** This declares the varible strike, the options strike price
(negative for puts)?? */

   private double strike = 100;

/** This declares the varible tau, the time to expiration in years.
*/

   private double tau = 0.25;

/** This declares the varible rfr, the risk-free rate. */
```

```
   private double rfr = 0.05;

/** This declares the varible div. */

   private double div = 0.01;

/** This declares the variable sigma, the volatility. */

   private double sigma = 0.25;

/** Decleration of option type, i.e. if it's a put or call,
* if flag = 0, option = call; if flag = 1, option = put.*/

   private double flag = 0;

/** Declares of variables used within methods. */

   baw_function sim_baw = null;
   bls_function sim_bls = null;

/** Declaration of Strings used in the Interface.  */

   static String generate = "        Calculate Values        ";
   static String title = "BAW  Option  Calculator";

/** Declaration of Panels used in the Interface */

   private JPanel mainPanel              = null;
   private JPanel inputPanel             = null;
   private JPanel variablePanel          = null;
   private JPanel resultPanel            = null;
   private JPanel formulaPanel           = null;
   private JPanel generatePanel          = null;

/** Declaration of Textfields used in the Interface */

   private JTextField inputtau           = null; //double
   private JTextField inputstrike        = null; //int
   private JTextField inputrfr           = null; //double[]
   private JTextField inputs0            = null; //double
   private JTextField inputdiv           = null; //double
   private JTextField inputsigma         = null; //double
   private JTextField inputflag          = null; //returns value...
   private JTextField empty              = null; //some empty space
   private JTextField outputbaw          = null; //returns value...
   private JTextField outputbls          = null; //returns value...


/** Declaration of different buttons and checkboxes that
* are used in the Interface */

   private JButton generateButton        = null;
   private JButton formulaButton         = null;


/** Declare Labels used in the interface */

   private JLabel formula                = null;


/** Creates colors used in the interface*/
```

```
   private Color blue = new Color(191,216,249);
   private Color grey = new Color(222,222,222);


/** Initializes the applet AutomatonInterface and initializes
surface that
* will contain everything, except the toolbaar. */

      public void init() {

         Container contentPane = getContentPane();


/** Creates a panel to contain other objects */

         mainPanel = new JPanel();
         mainPanel.setLayout(new GridLayout(3,0,5,5));
         mainPanel.setBorder(new javax.swing.border.TitledBorder(
            null, title, javax.swing.border.TitledBorder.CENTER, +
            javax.swing.border.TitledBorder.ABOVE_TOP));
         contentPane.add(mainPanel);


/** Creates panel for input data and action buttons  */

         inputPanel = new JPanel();
         inputPanel.setLayout(new FlowLayout());


/** Creates a first panel to contain input variables
* s0, tau, strike, rfr, sigma */

         variablePanel = new JPanel();
         variablePanel.setLayout(new FlowLayout());
         variablePanel.add(Box.createRigidArea(new Dimension(1,1)));
         /** Uses two border layouts; EtchedBorder & TitledBorder */
         Border etta;
         etta = BorderFactory.createEtchedBorder(
            EtchedBorder.RAISED, grey, blue);
         variablePanel.setBorder(new
javax.swing.border.TitledBorder(
            etta, "", javax.swing.border.TitledBorder.CENTER,
            javax.swing.border.TitledBorder.ABOVE_TOP));


/** Creates area for commands and actions */

         resultPanel = new JPanel();
         resultPanel.setLayout(new FlowLayout());
         resultPanel.setBorder(BorderFactory.createEmptyBorder());
         resultPanel.add(Box.createRigidArea(new Dimension(1,1)));


/** Creates area for generate button */

         generatePanel = new JPanel();
         generatePanel.setLayout(new FlowLayout());
         generatePanel.setBorder(BorderFactory.createEmptyBorder());
```

```
/** Creates area for formulas */

         formulaPanel = new JPanel();
         formulaPanel.setLayout(new FlowLayout());
         formulaPanel.add(Box.createRigidArea(new Dimension(1,1)));


/** Creates empty Space */

         empty = new JTextField("",1);
         empty.setEditable(false);
         empty.setBorder(BorderFactory.createEmptyBorder());


/** Creates an input field for the variable s0 */

         inputs0 = new JTextField("100",4);
         inputs0.setEditable(true);
         inputs0.setBackground(

   javax.swing.UIManager.getDefaults().getColor("Button.background")
);
         /** Uses two border layouts; EtchedBorder & TitledBorder */
         Border one;
         one =
BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);
         inputs0.setBorder(new javax.swing.border.TitledBorder(
            one, "Stock", javax.swing.border.TitledBorder.CENTER,
            javax.swing.border.TitledBorder.DEFAULT_POSITION));
         variablePanel.add(inputs0);
         inputs0.setToolTipText("Please input the variable \"Stock\"
here."+
          " This is the price of the underlying asset.");
         inputs0.addActionListener(this);


/** Creates an input field for the variable Sigma*/

         inputsigma = new JTextField("0.25",4);
         inputsigma.setEditable(true);
         inputsigma.setBackground(

   javax.swing.UIManager.getDefaults().getColor("Button.background")
);
         /** Uses two border layouts; EtchedBorder & TitledBorder */
         Border two;
         two =
BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);
         inputsigma.setBorder(new javax.swing.border.TitledBorder(
            two, "Sigma", javax.swing.border.TitledBorder.CENTER,
            javax.swing.border.TitledBorder.DEFAULT_POSITION));
         variablePanel.add(inputsigma);
         inputsigma.setToolTipText("Please input the variable
\"Sigma\" here."+
            " This is the annual volatility of the underlying
asset.");
         inputsigma.addActionListener(this);


/** Creates an input field for the variable strike */
```

```
         inputstrike = new JTextField("100",4);
         inputstrike.setEditable(true);
         inputstrike.setBackground(

   javax.swing.UIManager.getDefaults().getColor("Button.background")
);
         /** Uses two border layouts; EtchedBorder & TitledBorder
*/
         Border eight;
         eight =
BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);
         inputstrike.setBorder(new javax.swing.border.TitledBorder(
            eight, "Strike", javax.swing.border.TitledBorder.CENTER,
            javax.swing.border.TitledBorder.DEFAULT_POSITION));
         variablePanel.add(inputstrike);
         inputstrike.setToolTipText("Please input the variable
\"Strike\" here."+
            " The contracts strike/settlement price.");
         inputstrike.addActionListener(this);


/** Creates an input field for the variable tau */

         inputtau = new JTextField("0.25",4);
         inputtau.setEditable(true);
         inputtau.setBackground(

   javax.swing.UIManager.getDefaults().getColor("Button.background")
);          /** Uses two border layouts; EtchedBorder & TitledBorder
*/
         Border nine;
         nine =
BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);
         inputtau.setBorder(new javax.swing.border.TitledBorder(
            nine, "Tau", javax.swing.border.TitledBorder.CENTER,
            javax.swing.border.TitledBorder.DEFAULT_POSITION));
         variablePanel.add(inputtau);
         inputtau.setToolTipText("Please enter the variable \" Tau
\" here."+ " Tau represents the options life to maturity expressed
in years.");
         inputtau.addActionListener(this);


/** Creates an input field for the variable rfr */

         inputrfr = new JTextField("0.05",4);
         inputrfr.setEditable(true);
         inputrfr.setBackground(

   javax.swing.UIManager.getDefaults().getColor("Button.background")
);
         inputrfr.setBorder(new javax.swing.border.TitledBorder(
            null, "Rfr",javax.swing.border.TitledBorder.CENTER,
            javax.swing.border.TitledBorder.DEFAULT_POSITION));
         variablePanel.add(inputrfr);
         inputrfr.setToolTipText("Please input the variable \"Rfr\"
here."+ " The risk-free interest rate per annum." );
         inputrfr.addActionListener(this);


/** Creates an input field for the variable div */
```

```
         inputdiv = new JTextField("0.0",4);
         inputdiv.setEditable(true);
         inputdiv.setBackground(

   javax.swing.UIManager.getDefaults().getColor("Button.background")
);
         inputdiv.setBorder(new javax.swing.border.TitledBorder(
            null, "Div",javax.swing.border.TitledBorder.CENTER,
            javax.swing.border.TitledBorder.DEFAULT_POSITION));
         variablePanel.add(inputdiv);
         inputdiv.setToolTipText("Please enter the variable \"Div\"
here."+
            " The annualized dividend yield of the underlying
asset.");
         inputdiv.addActionListener(this);

/** Creates an input field for the variable flag */

         inputflag = new JTextField("0",4);
         inputflag.setEditable(true);
         inputflag.setBackground(

   javax.swing.UIManager.getDefaults().getColor("Button.background")
);
         inputflag.setBorder(new javax.swing.border.TitledBorder(
            null, "Put/Call",javax.swing.border.TitledBorder.CENTER,
            javax.swing.border.TitledBorder.DEFAULT_POSITION));
         variablePanel.add(inputflag);
         inputflag.setToolTipText("Please enter \"0\" if the option
is a Call"+
            " and \"1\" if it is a put.");
         inputflag.addActionListener(this);

/** Creates button for the formula to be displayed in */

         Icon icon = new ImageIcon("formel.jpg");
         formulaButton = new JButton(icon);
         formulaButton.setMargin(new Insets(0,0,0,0));
         formulaPanel.add(formulaButton);
         formulaButton.setMnemonic(KeyEvent.VK_I);


/** Creates an output field for the calculated BAW */

         outputbaw = new JTextField("",15);
         outputbaw.setEditable(true);
         outputbaw.setBackground(

   javax.swing.UIManager.getDefaults().getColor("Button.background")
);
         outputbaw.setBorder(new javax.swing.border.TitledBorder(
            null, "BAW Result",
javax.swing.border.TitledBorder.CENTER,
            javax.swing.border.TitledBorder.DEFAULT_POSITION));
         outputbaw.setToolTipText("Results from BAW calculation is
shown here.");
         outputbaw.addActionListener(this);


/** Creates an output field for the calculated BLS */
```

```
         outputbls = new JTextField("",15);
         outputbls.setEditable(true);
         outputbls.setBackground(

   javax.swing.UIManager.getDefaults().getColor("Button.background")
);
         outputbls.setBorder(new javax.swing.border.TitledBorder(
            null, "BLS Result",
javax.swing.border.TitledBorder.CENTER,
            javax.swing.border.TitledBorder.DEFAULT_POSITION));
         outputbls.setToolTipText("Results from BLS calculation is
shown here.");
         outputbls.addActionListener(this);


/** Generate button */

         generateButton = new JButton(generate);
         generateButton.setMinimumSize(new Dimension(5,10));
         generateButton.setMnemonic(KeyEvent.VK_G);

/** Shows information about the function for the user */

         generateButton.setToolTipText("Calculate values of
option");

/** Desides what happens when the generateButton is beeing pressed,
* through a anomymous actionlistener.*/

         generateButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {

/** Sets the boolean variable errors' value to to false before
* starting check of erors */

            boolean errors = false;

/** Tells the program to start if the generateButton is pressed */

            if (event.getActionCommand() == generate) {

/** Displays a message at the bottom of the interface */

               showStatus("Generating values.    Please wait...");

/** Gets and checks the values of users entered inputs */

               try {
                  tau =
Double.valueOf(inputtau.getText()).doubleValue();
                  if (tau<0) {
                  JOptionPane.showMessageDialog(null,"Please give a
positive float"+
                     "number", "ERROR tau"
,JOptionPane.ERROR_MESSAGE); }
                  System.out.println("tau:" +tau);
               }
               catch (NumberFormatException e) {
                  JOptionPane.showMessageDialog(null,"Please give a
positive float"+
                  "number", "ERROR tau" ,JOptionPane.ERROR_MESSAGE);
                  errors = true;
               }
               try {
                  sigma = 
Double.valueOf(inputsigma.getText()).doubleValue();
                  System.out.println("sigma:" +sigma);
               }
               catch (NumberFormatException e) {
                  JOptionPane.showMessageDialog(null,"Please give a 
float number",
                  "ERROR sigma" ,JOptionPane.ERROR_MESSAGE);
                  errors = true;
               }
               try {
                  div = Double.parseDouble(inputdiv.getText());
                  System.out.println("div:" +div);
               }
               catch (NumberFormatException e) {
                  JOptionPane.showMessageDialog(null,"Please give a 
float number",
                  "ERROR div" ,JOptionPane.ERROR_MESSAGE);
                  errors = true;
               }
               try {
                  rfr = Double.parseDouble(inputrfr.getText());
                  System.out.println("rfr:" +rfr);
               }
               catch (NumberFormatException e) {
                  JOptionPane.showMessageDialog(null,"Please give a 
float number",
                  "ERROR rfr" ,JOptionPane.ERROR_MESSAGE);
                  errors = true;
               }
               try {
                  s0 = Double.parseDouble(inputs0.getText());
                  System.out.println("s0:" +s0);
               }
               catch (NumberFormatException e) {
                  JOptionPane.showMessageDialog(null,"Please give a 
float number",
                  "ERROR s0" ,JOptionPane.ERROR_MESSAGE);
                  errors = true;
               }
               try {
                  strike = 
Double.parseDouble(inputstrike.getText());
                  System.out.println("strike:" +strike);
               }
               catch (NumberFormatException e) {
                  JOptionPane.showMessageDialog(null,"Please give a 
float number",
                  "ERROR strike" ,JOptionPane.ERROR_MESSAGE);
                  errors = true;
               }
               try {
                  flag = Double.parseDouble(inputflag.getText());
                  System.out.println("Flag:" +flag);
                  if (flag<0 || flag>1) {
                     JOptionPane.showMessageDialog(null,"Option type 
must be 0 or 1"+
                        "number", "ERROR flag"
,JOptionPane.ERROR_MESSAGE); }
                        System.out.println("Flag:" +flag);
                        errors = true;
               }
               catch (NumberFormatException e) {
                  JOptionPane.showMessageDialog(null,"Please give a
float number",
                  "ERROR Flag" ,JOptionPane.ERROR_MESSAGE);
                  errors = true;
               }


/** Performs calculations if errors = false */

               if (flag == 0) {     // Call
                  baw_function sim_baw = new
baw_function(s0,strike,rfr,div,sigma,tau,flag);
                  outputbaw.setEditable(false);

   outputbaw.setText(Double.toString(sim_baw.getbaw()));
               }
               else if (flag == 1) {   // Put
                  baw_function_put sim_baw = new
baw_function_put(s0,strike,rfr,div,sigma,tau,flag);
                  outputbaw.setEditable(false);

   outputbaw.setText(Double.toString(sim_baw.getbaw()));
               }

               bls_function sim_bls = new
bls_function(s0,strike,rfr,div,sigma,tau,flag);
               outputbls.setEditable(false);
               outputbls.setText(Double.toString(sim_bls.getbls()));
            }
         showStatus("Values Calculated.");
         }
      });

/** Orientation and orders of panels */

      mainPanel.add(formulaPanel);
      mainPanel.add(inputPanel);
      inputPanel.add(variablePanel);
      inputPanel.add(generatePanel);
      generatePanel.add(empty);
      generatePanel.add(generateButton);
      resultPanel.add(outputbaw);
      resultPanel.add(empty);
      resultPanel.add(outputbls);
      mainPanel.add(resultPanel);
   }

/** Listens for values to calculate on and when button changes
status. */

   public void actionPerformed(ActionEvent e) {
   }
}
```

```
/**
 * @(#) Baw_function.java  Version: 1.0 05/05/23
 *
 * This program has been written by
 * Robert Byström, Högsätrav 30, 181 58  Lidingö.
 *
 * The program is written as a part in the course
 * "Mathematics Project" (MATH3031) at the University of Southampton
 * with Dr Gerard Kennedy as mentor.
 *
 * All Rights Reserved.
 */

/**
 * The baw_function class provides methods for calculations due to
the model of
 * Barone-Adesi & Whaley.
 * and instances to the program interfacec.
 *
 * @version 1.0 23 May 2005
 * @author  Robert Byström
 */

public class baw_function extends formulas {
   double baw;

   public baw_function(double s0,
                       double strike,
                       double rfr,
                       double div,
                       double sigma,
                       double tau,
                       double flag) {

//    double flag;
      double coc = rfr-div;
      double sigma_sqr = (sigma*sigma);

      double N = 2*div/sigma_sqr;
      double M = 2*rfr/sigma_sqr;
      double K = 1-Math.exp(-rfr*tau);

      double q2 = (-(N-1)+Math.sqrt(((N-1)*(N-1))+(4*M/K)))*0.5;
      double q2_inf = 0.5*((-N-1)+Math.sqrt(((N-1)*(N-1))+4*M));
      double S_star_inf = strike/(1-1/q2_inf);
      double h2 = -
(coc*tau+2*sigma*Math.sqrt(tau))*(strike/(S_star_inf-strike));
      double S_seed = strike+(S_star_inf-strike)*(1-Math.exp(h2));

      final double ACC = Math.pow(10,-6);

      int no_iterations = 0; // iterate on S to find S_star, using
Newton steps
      double Si = S_seed;
      double g = 1;
      double g_prime = 1.0;

      double d1;
      double d2;
      double bls;
      while ((Math.abs(g)>ACC) && (Math.abs(g_prime)>ACC)
         && (no_iterations++<100) && (Si>0.0) && (coc<rfr)) {

         d1 =
(Math.log(s0/strike)+(coc+sigma_sqr/2)*tau)/(sigma*Math.sqrt(tau));
         d2 = d1-(sigma*Math.sqrt(tau));
         bls = Si*Math.exp((coc-rfr)*tau)*calcN(d2)-
strike*Math.exp(-(rfr*tau))*calcN(d2);
         d1 =
(Math.log(Si/strike)+(coc+sigma_sqr/2)*tau)/(sigma*Math.sqrt(tau));

         g = (1-1/q2)*Si-strike-bls+(1/q2)*Si*Math.exp((div-
rfr)*tau)*calcN(d1);
         g_prime = (1-1/q2)*(1-Math.exp((coc-rfr)*tau)*calcN(d1))+
                     (1/q2)*(Math.exp((coc-
rfr)*tau)*calc_n(d1))*(1/(sigma*Math.sqrt(tau)));

         Si = Si-(g/g_prime);
         System.out.println("number of iterations_call : " +
no_iterations);
      };

      double S_star = 0;

      if (Math.abs(g)>ACC) {         S_star = S_seed;
      }
      else {
         S_star = Si;
      }

      double C = 0;

         d1 =
(Math.log(s0/strike)+(coc+sigma_sqr/2)*tau)/(sigma*Math.sqrt(tau));
         d2 = d1-(sigma*Math.sqrt(tau));
         bls = s0*Math.exp((coc-rfr)*tau)*calcN(d1)-
strike*Math.exp(-(rfr*tau))*calcN(d2);

      if (s0 >= S_star) {
         C = s0-strike;
         System.out.println("C : " + C);
      }
      else {
         d1 =
(strike*Math.exp(S_star)+(coc+sigma_sqr/2)*tau)/(sigma*Math.sqrt(tau
));
         double A2 = (1-Math.exp((coc-
rfr)*tau)*calcN(d1))*(S_star/q2);
         C = bls+A2*Math.pow((s0/S_star),q2);
         System.out.println("C=bls+ : " + C);
      };

      if (C > bls) {
         baw = C;
         System.out.println("baw=C : " + baw);
      }
      if (coc >= rfr) {
         baw = bls;
         System.out.println("coc=>rfr; baw=bls : " + baw);
      }
      else if (bls > C) {
         baw = bls;
         System.out.println("bls>C+ : " + C);
      }
      baw = Math.max(C,bls);
      System.out.println("baw-Max : " + C);
   }

   public final double getbaw() {
      System.out.println("getbaw() : " + baw);
      return baw;
   }
}


/**
 * @(#) Baw_function_put.java  Version: 1.0 05/05/23
 *
 * This program has been written by
 * Robert Byström, Högsätrav 30, 181 58  Lidingö.
 *
 * The program is written as a part in the course
 * "Mathematics Project" (MATH3031) at the University of Southampton
 * with Dr Gerard Kennedy as mentor.
 *
 * All Rights Reserved.
 */

/**
 * The baw_function_put class provides methods for calculations due
to the model of
 * Barone-Adesi & Whaley.
 * and instances to the program interfacec.
 *
 * @version 1.0 23 May 2005
 * @author  Robert Byström
 */

public class baw_function_put extends formulas {
   double baw;

   public baw_function_put(double s0,
                           double strike,
                           double rfr,
                           double div,
                           double sigma,
                           double tau,
                           double flag) {

      double coc = rfr-div;
      double sigma_sqr = (sigma*sigma);

      double M = 2*rfr/sigma_sqr;
      double N = 2*div/sigma_sqr;
      double K = 1-Math.exp(-rfr*tau);

      double q1 = (-(N-1)-Math.sqrt(((N-1)*(N-1))+(4*M/K)))*0.5;
      double q1_inf = ((-N-1)+Math.sqrt(((N-1)*(N-1))+4*M))*0.5;
      double S_star_inf = strike/(1-1/q1_inf);
      double h1 = (coc*tau-2*sigma*Math.sqrt(tau))*(strike/(strike-
S_star_inf));
      double S_seed = S_star_inf+(strike-S_star_inf)*Math.exp(h1);
      final double ACC = Math.pow(10,-6);

      int no_iterations = 0;  // iterate on S to find S_star, using
Newton steps
      double Si = S_seed;
      double g = 1;
      double g_prime = 1.0;

      double d1;
      double d2;
      double bls;

      while ((Math.abs(g)>ACC) && (Math.abs(g_prime)>ACC)
         && (no_iterations++<100) && (Si>0.0) && (coc<rfr)) {

         d1 =
(Math.log(s0/strike)+(coc+sigma_sqr/2)*tau)/(sigma*Math.sqrt(tau));
         d2 = d1-(sigma*Math.sqrt(tau));
         d1 = -1*d1;
         d2 = -1*d2;
         bls = Si*Math.exp((coc-rfr)*tau)*calcN(d2)-
strike*Math.exp(-(rfr*tau))*calcN(d2);
         d1 =
(Math.log(Si/strike)+(coc+sigma_sqr/2)*tau)/(sigma*Math.sqrt(tau));
         d1 = -1*d1;

         g = strike-s0-bls+(s0/q1)*(1-Math.exp((coc-rfr)*tau));
         System.out.println("g : " + g);
         g_prime = ((1/q1)-1)*(1-Math.exp((coc-rfr)*tau)*calcN(d1))+
            (1/q1)*Math.exp((coc-
rfr)*tau)*(1/(sigma*Math.sqrt(tau)))*calc_n(d1);
         System.out.println("g_prime : " + g_prime);

//       g = -((1-calcN(d1))*Si)/q1;
//       g_prime = 1-(1-calcN(d1))*(1-
1/q1)+(calcN(d1)/(sigma*q1*Math.sqrt(tau)));

         Si = Si-(g/g_prime);
         System.out.println("number of iterations_put : " +
no_iterations);
      };

      double S_star = 0;

      if (Math.abs(g)>ACC) {
         S_star = S_seed;
      }
      else {
         S_star = Si;
      }

      double C = 0;

      d1 =
(Math.log(s0/strike)+(coc+sigma_sqr/2)*tau)/(sigma*Math.sqrt(tau));
      d2 = d1-(sigma*Math.sqrt(tau));
      d1 = -1*d1;
      d2 = -1*d2;
      bls = strike*Math.exp(-rfr*tau)*calcN(d2)-s0*Math.exp((coc-
rfr)*tau)*calcN(d1);
```

```
      if (s0 <= S_star) {
         C = strike-s0;
         System.out.println("C : " + C);
      }
      else {
         d1 =
(strike*Math.exp(S_star)+(coc+sigma_sqr/2)*tau)/(sigma*Math.sqrt(tau
));
         d1 = -1*d1;
         double A1 = -(S_star/q1)*(1-Math.exp((coc-
rfr)*tau)*calcN(d1));
         C = bls+A1*Math.pow((s0/S_star),q1);
         System.out.println("C=bls+ : " + C);
      };

      if (C > bls) {
         baw = C;
         System.out.println("baw=C : " + baw);
      }
      if (coc >= rfr) {
         baw = bls;
         System.out.println("coc=>rfr; baw=bls : " + baw);
      }
      else if (bls > C) {
         baw = bls;
         System.out.println("bls>C+ : " + C);
      }
      baw = Math.max(C,bls);
      System.out.println("baw-Max : " + C);
   }

   public final double getbaw() {
      System.out.println("getbaw_put() : " + baw);
      return baw;
   }
}


/**
 * @(#) bls_function.java  Version: 1.0 05/05/23
 *
 * This program has been written by
 * Robert Byström, Högsätravägen 30, 181 58 Lidingö.
 *
 * The program is written as a part in the course
 * "Mathematics Project" (MATH3031) at the
 * University of Southampton with Dr Gerard Kennedy as mentor.
 *
 * All Rights Reserved.
 *
 */

/**
 * The bls_function class provides methods to calculate Black &
Scholes values
 * for European call options.
 *
 * @version 1.0 23 May 2005
 * @author  Robert Byström
 */
```

```
public class bls_function extends formulas {

   double bls;


/** Creates a method to calculate Logarithmic Returns, which will be
called
* from the parent class AutomatonInterface */

   public bls_function(double s0,
                       double strike,
                       double rfr,
                       double div,
                       double sigma,
                       double tau,
                       double flag) {
   double coc = rfr-div;
   double d1;
   double d2;

   d1 =
(Math.log(s0/strike)+(coc+(sigma*sigma)/2)*tau)/(sigma*Math.sqrt(tau
));
   d2 = d1-(sigma*Math.sqrt(tau));

   if (flag == 0) {            // Call
      bls = s0*Math.exp((coc-rfr)*tau)*calcN(d1)-strike*Math.exp(-
(rfr*tau))*calcN(d2);
   }
   else if (flag == 1) {      //Put
      d1 = -1*d1;
      d2 = -1*d2;
      bls = strike*Math.exp(-rfr*tau)*calcN(d2)-s0*Math.exp((coc-
rfr)*tau)*calcN(d1);
   }
}

   public final double getbls() {
      System.out.println("getbls() : " + bls);
      return bls;
   }
}


/**
 * @(#) formulas.java  Version: 1.0 05/05/23
 *
 * This program has been written by
 * Robert Byström, Högsätravägen 30, 181 58 Lidingö.
 *
 * The program is written as a part in the course
 * "Mathematics Project" (MATH3031) at the
 * University of Southampton with Dr Gerard Kennedy as mentor.
 *
 * All Rights Reserved.
 *
 */

/**
 * The class formulas provides methods and functions needed in other
 * classes of this package.
 *
 * @version 1.0 23 May 2005
 * @author  Robert Byström
 */

public class formulas {
   /** Stock spot price */
   protected double s0;

   /** Contracts strike price */
   protected double strike;

   /** Risk-free-rate */
   protected double rfr;

   /** Cost-of-carry (b=r-d)*/
   protected double coc;

   /** Volatility, sigma */
   protected double sigma;

   /** Time in years */
   protected double tau;

   protected double x;
   protected double n;
   protected double bls;
   protected double baw;

   /**
    * Constructor by default
    */
   protected formulas() {
      // Initialise variables
      s0 = strike = rfr = coc = sigma = tau =n=x;
   }

   /**
   * Method calc_n
   * Returns univariate normal density function, n(x)
   *
   * @return univariate double normal density
   */
   double calc_n(double x) {
      double result;
      //  Calculates univariate normal density funftion n(x)
      double n = (1/(Math.sqrt(2*Math.PI)))*Math.exp(-
Math.pow(x,2)/2);
      return n;
   }

   /**
   * Method calcN
   * Returns cumulative normal probability distribution [0,1], N(x)
   *
   * @return cumulative normal probability
   */
   double calcN(double n){
      // Calculates cumulative normal probability distribution for
      // variable with mean of zero and standard deviation of one
```

```
        double result;

        double x = Math.abs(n);

        double l  = 0.33267d;
        double a1 = 0.4361836d;
        double a2 = -0.1201676d;
        double a3 = 0.9372980d;
        double k  = 1/(1+(x*l));

        double N = (1/Math.sqrt(Math.PI*2));
        N *= Math.exp(-(Math.pow(x,2)/2));

        result = N*((a1*k)+(a2*Math.pow(k,2))+(a3*Math.pow(k,3)));
        result = 1-result;

        if (n >= 0) {
           return result;
           }
        else {
           return (1-result);
           }
    }
}
```

B. Matlab m-file

C:\Program\Matlabb\toolbox\finance\finance\binprice.m

```
function [pr,opt] = binprice(so,x,r,t,dt,sig,flag,q,div,exdiv)
%BINPRICE Binomial put and call pricing.
% [PR,OPT] = BINPRICE(SO,X,R,T,DT,SIG,FLAG,Q,DIV,EXDIV) prices an
option
% using a binomial pricing model. SO is the underlying asset price,
X is the
% option exercise price, R is the risk-free interest rate, T is the
option's
% time until maturity in years and DT is the time increment within
T.
% DT will be adjusted so that the length of each interval is
consistent with
% the maturity time of the option. SIG is the asset's volatility,
FLAG
% specifies whether the option is a call (flag = 1) or a put (flag =
0),
% Q is the dividend rate, DIV is the dividend payment at an ex-
dividend date,
% EXDIV. EXDIV is specified in number of periods. All inputs to this
% function are scalar values except DIV and EXDIV which are 1-by-n
vectors.
% For each dividend payment, there must be a corresponding ex-
dividend date.
% By default q,div, and EXDIV equal 0. If a value is entered for the
% dividend rate q, DIV and EXDIV should equal 0 or not be entered.
If values
% are entered for DIV and EXDIV, set Q = 0.
%
% [P,O] = binprice(52,50,.1,5/12,1/12,.4,0,0,2.06,3.5) returns the
asset
```

```
% price and option value at each node of the binary tree.
%
% P =
%
% 52.0000 58.1367 65.0226 72.7494 79.3515 89.0642
% 0 46.5642 52.0336 58.1706 62.9882 70.6980
% 0 0 41.7231 46.5981 49.9992 56.1192
% 0 0 0 37.4120 39.6887 44.5467
% 0 0 0 0 31.5044 35.3606
% 0 0 0 0 0 28.0688
%
% O =
%
% 4.4404 2.1627 0.6361 0 0 0
% 0 6.8611 3.7715 1.3018 0 0
% 0 0 10.1591 6.3785 2.6645 0
% 0 0 0 14.2245 10.3113 5.4533
% 0 0 0 0 18.4956 14.6394
% 0 0 0 0 0 21.9312
%
% See also BLSPRICE.
%
% Reference: Options, Futures, and Other Derivative Securities,
% 2nd Edition, Hull, Chapter 14.
% Author(s): C.F. Garvin, 5-10-95
% Copyright 1995-2002 The MathWorks, Inc.
% $Revision: 1.9 $ $Date: 2002/03/11 19:19:02 $
if nargin < 8
q = 0;
end
if nargin < 9
div = 0;
exdiv = 0;
end
if nargin < 7
error('Missing one of SO, X, R, T, DT, SIG, and FLAG.')
end
if flag ~= 0 & flag ~= 1
error('FLAG must be 1 for call option or 0 put option.')
end
if q ~= 0 & div ~= 0
error('DIV and EXDIV must be zero for non-zero dividend rate, Q.')
end
% Calculate the number of periods, nper, and the length of each
interval, dt.
% Make sure that the length of each interval is consistent with
% the maturity time of the option.
nper = round(t/dt); % Number of periods after time zero
dt = t/nper;
npp = nper+1; % Number of periods including time zero
% Calculate the probability of an upward price movement
u = exp(sig.*sqrt(dt));
d = 1./u;
a = exp((r-q).*dt);
p = (a-d)./(u-d);
jspan = -fix(nper*.5); % j-th node offset number
ispan = rem(round(t/dt),2); % i-th node offset number
i = ispan:(nper+ispan); % i-th node numbers
j = (jspan:(nper+jspan))'; % j-th node numbers
jex = j(:,ones(size(i'))); % expand i and j to eliminate for loop
iex = i(ones(size(j)),:);
```

```
pvdiv = div.*exp(-exdiv.*dt.*r); % Find present value of all
dividends
so = so-sum(pvdiv(:)); % Find current price - div present values
% Asset price at nodes, matrix is flipped so tree appears correct
visually
pr = triu(fliplr(flipud(so.*u.^jex.*d.^(iex-jex))));
if div ~= 0 % Present value of future dividends at nodes
lendiv = length(div(:));
lenexdiv = length(exdiv(:));
if lendiv ~= lenexdiv
error('Number of dividend and ex-dividend entries must be equal.')
end
dpvtot = zeros(npp); % Preallocate matrix
for y = 1:lenexdiv
z = (exdiv(y):-1:0); % Create vector from 0 to ex-div date
dpv = div(y)*exp(-z*dt*r); % Discount dividends nodes
dpvmat = [dpv(ones(npp,1),:) zeros(npp,npp-length(dpv))]; % Expand
matrix
dpvtot = dpvtot + dpvmat; % Add next discounted dividend to total
end
m = find(pr~=0); % Find nodes where option will have value
pr(m) = pr(m)+dpvtot(m); % combine div pv's and prices to get new
prices
end
opt = zeros(size(pr));
if flag == 1 % Option is a call
opt(:,npp) = max(pr(:,npp)-x,0); % Determine option values from
prices
for n = nper:-1:1
k = 1:n;
% Probable option values discounted back one time step
discopt = (p*opt(k,n+1)+(1-p)*opt(k+1,n+1))*exp(-r*dt);
% Option value is max of current price - X or discopt
opt(:,n) = [max(pr(1:n,n)-x,discopt);zeros(npp-n,1)];
end
elseif flag == 0 % Option is a put
opt(:,npp) = max(x-pr(:,npp),0); % Determine option values from
prices
for n = nper:-1:1
k = 1:n;
% Probable option values discounted back one time step
discopt = (p*opt(k,n+1)+(1-p)*opt(k+1,n+1))*exp(-r*dt);
% Option value is max of X - current price or discopt
opt(:,n) = [max(x-pr(1:n,n),discopt);zeros(npp-n,1)];
end
end
```