Synthetic Portfolios and Currency Options

Stuart L. Pook

Department of Econometrics

ABSTRACT

This thesis is submitted in partial fulfilment of the requirements for a Bachelor of Economics (Honours) degree in Operations Research. The work has been done in association with the Macquarie Bank.

1. Introduction

At the moment the market for options in Australia is very thin. This is due to a lack of grantors or the reluctance of grantors to write options except at very significant premiums, usually above the theoretical economic price of the option (Das, 1986). This thin market would give a medium sized bank, such as Macquarie Bank, extra income earning possibilities should it be able to offer currency options at a price closer to their theoretical value. Due to the lack of competition in the market place at the moment, it may be possible to sell these options with a considerable margin over the fair price, until the number of grantors increases. This margin will provide a safety buffer should the cost of providing an option be higher than first calculated.

When an option is written the seller of the option is faced with an effective short position in the underlying asset in the case of a call option, or an effective long position in the case of a put option. However, the position of the grantor is not exactly the same as that of a holder of the asset as the grantor's upside and downside risk distributions are not symmetric (Das, 1986). In fact, the gain from writing an option is limited whilst the possible loss is very large or even unlimited. For this reason the writer of an option will often wish to manage the risk from adverse currency movements in some way. Some methods of managing this risk are:

— Purchase an identical option.

— Simultaneously purchase other options which lead to a reduced time or price risk.

— Grant options against existing currency positions.

— Use a synthetic option to match the granted option.

A synthetic option is created from a portfolio of existing traded instruments which with proper management over time can replicate the return characteristics of an option. The synthetic option is created by using a portfolio consisting of two instruments: the underlying asset of the option and a risk free asset. The key to creating a synthetic option is to determine the proportion to maintain between the two instruments. This proportion is adjusted through time in a very specific way to replicate the price behaviour of an option (Das, 1986: 13).

A bank should be able to offer options at a price lower than a customer can provide internally with synthetic options because a bank has lower overheads involved in the management of a synthetic option portfolio. The areas that a bank would have lower costs would include: management costs (from larger scale operations and more highly skilled personnel), and lower financial market costs (resulting from lower transaction costs on the currency markets and a greater access to funds).

The aim of this thesis is to test the robustness of the option pricing and synthetic portfolio models, to be discussed in the literature review, to relaxation of their underlying assumptions. Should the models be not significantly affected by the relaxation of their assumptions that occurs in the real world they will be be useful to an option trader in the exchange markets.

2. Literature Review

2.1 Black and Scholes (1973)

The paper by Black and Scholes entitled ‘‘*The Pricing of Options and Corporate Liabilities*’’ was the first to successfully describe a satisfactory method of pricing options. Previous work on the valuation of options was expressed in terms of warrants[1]. However, none of the option valuation formulas produced were complete, since they all involved one or more arbitrary parameters (p 639). The valuation method proposed by Black and Scholes derives from the realization that if options are correctly priced in the market it should not be possible to make sure profits by creating portfolios of long and short positions in options and their underlying stock. In deriving their formula for the value of an option some assumptions were made and as these are important for this thesis they are listed below.

a. The short-term interest rate is known and is constant through time.

b. The stock price follows a random walk in continuous time with a variance rate proportional to the square of the stock price. Thus, the distribution of possible stock prices at the end of any finite interval is log-normal. The variance rate of return on the stock is constant.

c. The stock pays no dividends or other distributions.

---

1. ‘‘A warrant is similar to a call. Its owner has the right to buy a fixed number of shares of a specified common stock at a specified price at any time until a given date. However, they are not exactly the same. Warrants are issued by corporations rather than by individuals. When a warrant is exercised, new shares are created, and the exercise price paid for them becomes part of the assets of the firm’’ (Cox and Rubinstein, 1985: 392).

d. The option is ‘‘European’’.

e. There are no transaction costs in buying or selling the stock or the option.

f. It is possible to borrow any fraction of the price of a security to buy or hold it, at the short-term interest rate.

g. There are no penalties to short selling.

Using the realization that sure profits should not be available if options are correctly priced, Black and Scholes derive the following differential equation and boundary conditions which describe the value of a European call option:

$$w_2 = rw - rxw_1 - \tfrac{1}{2} v^2 x^2 w_{11}$$

where:

$$w(x, 0) = \max(0, x - c)$$

Equation 1.

The notation used in this section is:

w value of a foreign exchange call option (domestic units per foreign unit)

x spot price of the deliverable currency (domestic units per foreign unit)

f futures price of the currency, deliverable at option maturity (domestic units per foreign unit)

f_t futures price of the currency at time t, deliverable at option maturity (domestic units per foreign unit)

c exercise price of the option (domestic units per foreign unit)

t time remaining until maturity of the option (as a fraction of a year)

v volatility of the spot currency price

r_d continuously compounding version of the domestic (riskless) interest rate (as a fraction per annum)

r_f continuously compounding version of the foreign (riskless) interest rate (as a fraction per annum)

$N(.)$ cumulative normal distribution

The equation can be solved to get the following closed form formula for the value of a European call option:

$$w(x,t) = xN(d_1) - e^{-r_d t} cN(d_2)$$

where:

$$d_1 = \frac{\ln(x/c) + (r_d + \frac{1}{2} v^2)t}{v\sqrt{t}}$$

$$d_2 = d_1 - v\sqrt{t}$$

Equation 2.

2.2 Black (1976)

In his paper titled *"The Pricing of Commodity Contracts"* Black modified the original Black and Scholes model to change the underlying stock of the option to a futures contract[2] on the stock. The result was the following differential equation for the value of a futures option (which is missing a term because the value of a futures contract is zero):

---

2. A futures contract is similar to a forward contract except that as the price of the commodity of the contract changes the party in whose favour the price change occurred must immediately be paid the full amount of the change by the losing party (Cox and Rubinstein, 1985: 62). Note: in a forward contract the settlement is made once only at the expiry time of the contract.

$$w_2 = rw - \tfrac{1}{2} v^2 f^2 w_{11}$$

where:

$$w(f, 0) = \max(0, f - c)$$

Equation 3.

This differential equation gives the following formula for the value of a futures call option on a non-dividend paying stock:

$$w(f, t) = e^{-r_d t}[fN(d_1) - cN(d_2)]$$

where:

$$d_1 = \frac{\ln(f/c) + \tfrac{1}{2} v^2 t}{v\sqrt{t}}$$

$$d_2 = d_1 - v\sqrt{t}$$

Equation 4.

2.3 Cox, Ross and Rubinstein (1979)

The article by Cox, Ross and Rubinstein entitled ''*Option Pricing: A Simplified Approach*'' presents a discrete-time model for valuing options. As this model allows the behaviour of the holder of an option to be examined at many time points throughout the life of the option it is possible to account for more option types than the original Black and Scholes model. One extra possibility is ''American'' options. Using the discrete-time model the possibility of early exercise can be taken into account by checking to see at each time point whether the option is worth more dead[3] than alive.

2.4 Rubinstein and Leland (1981)

In the paper entitled ''*Replicating Options with Positions in Stock and Cash*'' Rubinstein and Leland state that ''In most situations of practical relevance, the price behaviour of a call option is very similar to a combined position involving the underlying stock and borrowing.'' (p 63). The paper then demonstrates how to create an option

---

3. exercised

position[4] in a stock that has all the characteristics of an option on the stock. This option position replicates the returns available from owning the option using only positions in the underlying stock of the option and cash. The replication is achieved for a purchased call by a strategy of buying and selling shares. As the stock price rises (falls) buy (sell) shares and increase (decrease) the borrowings in the portfolio. The paper also discusses the factors that can affect the accuracy of the replicating strategy, these factors are:

a. As the strategy can involve frequent trading transaction costs must be low.

b. It must be possible to borrow any cash required to buy shares and to be able to short the stock to the extent required.

c. There must not be any jump movements in the stock price that prevent adjustment of the option position to the changing price.

d. Future interest rates, the stock's volatility and dividends must be known.

2.5 Garman and Kohlhagen (1983)

The formulations of Black and Scholes do not apply well to foreign exchange options since multiple interest rates are involved, both foreign and domestic. So Garman and Kohlhagen in their paper entitled ''*Foreign Currency Option Values*'' modified the Black and Scholes approach to handle the differing interest rates. The differential equation they obtained for the value of a foreign exchange option on spot prices is:

---

4. synthetic option

$$w_2 = r_d w - (r_d - r_f)w_1 - \tfrac{1}{2} v^2 x^2 w_{11}$$

where:

$$w(x, 0) = \max(0, x - c)$$

Equation 5.

The formula they obtained was:

$$w(x, t) = e^{-r_f t} xN(d_1) - e^{-r_d t} cN(d_2)$$

where:

$$d_1 = \frac{\ln(x/c) + (r_d - r_f + \tfrac{1}{2} v^2)t}{v\sqrt{t}}$$

$$d_2 = d_1 - v\sqrt{t}$$

Equation 6.

Equation 6 can be differentiated with respect to the currency price, x, to get the hedge ratio for the option. This ratio is given in Equation 7.

$$\Delta(x, t) = e^{-r_f t} N(d_1)$$

Equation 7.

The results for $w(f, t)$ were as for Black (1976). However the hedge ratios were different as the forward price needed to be discounted by the factor $e^{(r_f - r_d)t}$, (Garman and Kohlhagen, 1983: 235).

2.6 Hoag and McKay (1984)

The article by Hoag and McKay entitled *''Foreign Exchange Risk Exposure: Managing Through Synthetic Options''* details the use of synthetic option portfolios in reducing the risk that an Australian borrower assumes when borrowing money offshore (in this case borrowing Swiss francs).

An Australian borrower would like to be able to purchase a foreign exchange currency call option on the Swiss franc to reduce the possible losses from adverse currency movements. However it is not possible to purchase traded call options on the

Swiss franc in Australia, so the paper describes how a synthetic call option is created from a portfolio of existing traded instruments. The characteristics of this portfolio are:

a. The initial investment in the portfolio should equal the value of a traded call.

b. The intermediate cash flow from the option should be zero.

c. The ending value of the portfolio should equal the value of a call option on Swiss francs at maturity —

    i. if the Australian dollar/Swiss franc exchange rate increases the portfolio will have value equal to the excess Australian dollars needed to purchase the required number of Swiss francs, and

    ii. if the Australian dollar/Swiss franc exchange rate decreases then the portfolio will expire worthless.

The method of creating the synthetic portfolio is described by calculating the value of the portfolio and the distribution of the value between Swiss francs and Australian dollar assets. The hedge ratio derived is the same as that given in the paper by Garman and Kohlhagen (1983), ie the partial derivative of the value of the option with respect to the exchange rate.

3. The Experiment

The option pricing and synthetic portfolio theories were developed under the assumptions listed in the Literature Review. When the assumptions are not violated the returns from granting options both with and without a corresponding synthetic portfolio are theoretically zero. The objective is to design and implement a simulation experiment to assess the robustness of the option valuation theory and the synthetic portfolio management strategy to relaxation of some of the underlying assumptions. The assumptions to be relaxed in the experiment are described below.

3.1 Relaxing The Assumptions

3.1.1 Assumption 1

The currency price follows a random walk in continuous time with a variance rate proportional to the square of the currency price. Thus the distribution of possible currency prices at the end of any finite interval is log-normal and the variance rate of return on the currency is constant.

To assess the robustness of the theory to relaxation of this assumption, options are written using both

a. **historical exchange rates**: the exchange rates as provided in the data file are used in the valuation of the option as well as in determining the losses and gains from exchange rate movements over the life of the option, and,

b. **random walk exchange rates**: the volatility used in the option valuation formula is used to produce a series of exchange rate movements which follow a random

walk.

3.1.2 Assumption 2

Short-term interest rates are known and constant through time.

To assess the robustness of the theory to relaxation of this assumption, options are written using both

a. **constant interest rates**: generate a data set in which the interest rates are constant throughout the life of the option. In this case the interest rates used to value the option are applied to the currency holdings on each day, and,

b. **historical interest rates**: use the domestic and foreign interest rates as supplied in the data file to calculate both the value of the options and the interest received from, or paid for, holding the currencies on each day of the option's life.

3.1.3 Assumption 3

The volatility of the price movements of the currency is stationary.

To assess the robustness of the theory to relaxation of this assumption, options are written using both

a. **short volatility measure**: calculate the volatility of the currency using the fluctuations occurring in the time period, of length equal to that of the option, preceding the starting date, and,

b. **long volatility measure**: calculate the volatility of the currency on the assumption that the volatility is constant throughout time. In this method all the data available is used, from the beginning of the data set up to starting date of the option.

3.1.4 Assumption 4

It is possible to instantaneously adjust the composition of the synthetic portfolio in response to currency price movements.

To assess the robustness of the theory to relaxation of this assumption, options are written using both

a. **daily adjustment**: calculate the theoretical delta of the synthetic portfolio each business day and by buying or selling the foreign currency bring the holdings back into agreement with the delta, and,

b. **weekly adjustment**: similar to the above except that adjustments occur once every five business days.

3.1.5 Summary

The first three assumptions are important to both the valuation and management theories while the last affects only the management strategy.

Once the tests on the assumptions of the model have been done a further set of experiments was carried out to determine if it is profitable to use a synthetic portfolio when granting options.

3.2 Testing For Robustness

The method used to perform each of the robustness tests is discussed below.

3.2.1 Exchange Rates

The effect of the violation of the random walk exchange rate assumption is isolated

by comparing two set of simulation runs, the first set using exchange rates generated in such a way that they obey the random walk restriction[5] and the second set using historical exchange rates. Each of these sets contains four simulations in which a synthetic portfolio is used. There are four simulations as there are two portfolio adjustment frequencies considered and two volatility calculation formulas. When a synthetic portfolio is not used there are two simulations, one for each of the two volatility measures. All the simulations use generated constant interest rates.

The six simulations in each set are compared like against like giving a total of six comparisons for this part of the experiment. Each of the comparisons will involve computing the difference in the means of the two runs. The differences will be tested for a significant deviation from zero.[6] A significant result will indicate that the violation of the tested assumption is important to the returns available from granting options.

3.2.2 Interest Rates

The effect of having unknown and variable interest rates on the returns from option trading is examined in a similar manner to that for exchange rates. Two sets of simulations are compared, one set using constant interest rates and the other using historical rates. All the simulations use generated exchange rates. Each set of simulations contains six elements, as in the exchange rate case, requiring a further six comparisons. The difference in the means of each pair is then tested for a significant deviation from zero.

---

5. The method used to produce all the generated data series will be described in the Data section.

6. The significance level to be used for the tests will be calculated once the total number of comparisons is known.

3.2.3 Exchange Rates and Interest Rates

Once the importance of the exchange rate and interest assumptions has been determined individually they are then tested simultaneously. The test is done as before, by comparing two sets of simulations. The first set is produced using generated exchange and interest rates. The second set is simulated using historical exchange and interest rates. There are six simulations in each set, thus requiring six more simultaneous comparisons to be performed. The mean difference in performance of each of the policies is calculated and compared with zero.

If any of the mean differences between policies deviate significantly from zero then it will be possible to conclude that the violation of the assumptions of the theoretical model is important to the returns from writing options.

3.2.4 Volatility Measure

There are two different measures of volatility used in this paper. The tests performed in this sub-section will evaluate their relative performance and decide if the choice of volatility measure makes a difference to the returns from granting options.

As in the preceding comparisons there are two sets of simulations whose results are compared. The two sets both contain the three simulation runs. Two of these are performed using a synthetic portfolio, one with daily portfolio adjustment and the other with weekly adjustment. The other simulation does not use a synthetic portfolio.

In contrast to the preceding experiments this experiment does not use both generated and historical data series. Only historical data is used, as in the generated

exchange rate data the same volatility is used for valuing the option and for the generation of the exchange rate data. This double use of the calculated volatility will result in the volatility used to value the option being reflected in the actual currency movements during the life of the option. The conclusion is that when using generated data it will not be possible to distinguish between the two volatility measures. So this experiment adds three comparisons to the total to date.

Should there be a significant difference in performance under either of the different policies the sign of the comparison will allow the best policy to be identified, both for the synthetic portfolio cases and the non-portfolio case.

3.2.5 Portfolio Adjustment Frequency

A synthetic portfolio has to be adjusted as the exchange rate varies and as the time to the option maturity decreases. Two different portfolio adjustment frequencies are examined. The comparisons are done as before, two sets of simulations are carried out and the significance of the mean difference in performances is examined. Each set contains four simulations, the four combinations of short and long volatility measure, and actual and generated data. The pairs of comparisons are between daily and weekly portfolio adjustments.

A significant result in one of the comparisons would indicate that the frequency of adjustment is an important variable to consider when managing a synthetic portfolio.

3.2.6 Synthetic Portfolio

In order to come to a conclusion about the desirability of using a synthetic

portfolio, a further set of experiments was performed. The experiments involved the comparison of the profits from options written and managed using a synthetic portfolio to the profits from an identical option not managed with a synthetic portfolio.

The performance from not using a portfolio is compared to the performance from a portfolio adjusted daily and to a portfolio adjusted weekly. This results in three sets of simulations. Each set of simulations contains four simulations, being all the combinations of volatility measure and actual versus generated data. The comparison of the set of no portfolio simulations against the other two sets requires eight more comparisons to be made.

If any of the above comparisons show a significant mean difference in profits then it would indicate that having a synthetic portfolio can influence the profits from granting options.

3.3 Option Type

In order to keep the problem within manageable limits attention was restricted mainly to options with the following characteristics:

— Call option on the United States dollar

— Duration 28 days

— European type

— Written at the money, ie the exercise price is equal to the exchange rate at the time of granting the option

3.4 Significance Level

There are 33 separate comparisons, listed in the description of the tests, to be performed in this experiment. Each of the comparisons must be regarded as being done simultaneously. The significance level used needs to take into account the simultaneous nature of the 33 comparisons.

The approach used is based on the Bonferroni inequality:

$$P[\textit{at least one of } c \textit{ possibly-dependent statements is false}] \leq \sum_{i=1}^{c} P[\textit{statement } i \textit{ is false}]$$

Using this inequality Bratley et al (1983: 81) derive the following method for the construction of c individual confidence intervals that cover all c parameters with probability at least α. Let

c = *the number of individual confidence intervals*
N = *the number of observations of each policy*
X_{qr} = *output of run* r *for policy* q

Then:

$$\bar{X}_i = \frac{1}{N}\sum_{r=1}^{N} X_{ir}$$

$$s_{ij}^2 = \frac{1}{N-1}\sum_{r=1}^{N}\left[X_i - X_j - (\bar{X}_i - \bar{X}_j)\right]^2$$

$$Z = \sqrt{N}\,\frac{\left[(\bar{X}_i - \bar{X}_j) - E(\bar{X}_i - \bar{X}_j)\right]}{s_{ij}}$$

$$\theta = \frac{1-\alpha}{2c} \textit{ percentile of } t_{N-1}$$

Equation 8.

Assuming Z has a t-distribution with $N-1$ degrees of freedom, then the individual confidence intervals of $E(\bar{X}_i - \bar{X}_j)$ have the form:

$$\left(\bar{X}_i - \bar{X}_j - \theta\,\frac{s_{ij}}{\sqrt{N}}\,,\ \bar{X}_i - \bar{X}_j + \theta\,\frac{s_{ij}}{\sqrt{N}}\right)$$

This method is used with a modification to the method used to calculate s_{ij} to take account of the correlation between the observations on X. The approach used to calculate s_{ij} is described in the section titled ‘‘Variance Estimation’’.

For experiments carried out $c = 33$ and $\alpha = 0.95$ (a 95% confidence interval), so

$$\theta = \frac{1 - 0.95}{2 \times 33} \textit{ percentile of } t_{N-1}$$
$$= 0.000758 \textit{ percentile of } t_{N-1}$$

Tables for $t_{0.000758}$ are not commonly available, so the tables for $t_{0.005}$ and $t_{0.0005}$ are used. Should the t value obtained for any test lie between the value given by these two tables a more accurate estimate of $t_{0.000758}$ will be required.

Once a value has been obtained for θ, the difference between two means can be tested for significance by calculating

$$t = \frac{|\bar{X}_i - \bar{X}_j|\sqrt{N}}{s_{ij}}$$

and comparing this with θ. If $t > \theta$ then the difference between the means is significantly non-zero.

The experiment were performed using only a small number of different sample sizes. The sample sizes used and their significance levels are given in Table 1.

N	$t_{0.005,N-1}$	$t_{0.0005,N-1}$
32	2.750	3.646
64	2.660	3.460
128	2.617	3.373
≥ 256	2.576	3.291

TABLE 1. Significance Levels

3.5 Sample Sizes

The number of replications required for each of the tests to be performed will be determined by doing a dummy run with 32 replications. The variance of the statistic under investigation will be used to calculate the number of replications needed to get significant results or to show that the statistic is not significantly different from zero.

For an initial sample size of 32 the value for θ is between $t_{0.0005,31} = 3.646$ and $t_{0.005,31} = 2.750$. Using the larger value, so as not to get an under estimate, the number of replications required to distinguish between policies i and j is given in Equation 9[7].

$$\begin{aligned} N &= \left(\frac{t_{0.0005,31} s_{ij}}{\bar{X}_i - \bar{X}_j} \right)^2 \\ &= \left(\frac{3.646 s_{ij}}{\bar{X}_i - \bar{X}_j} \right)^2 \\ &= \left(\frac{3.646 \sqrt{N}}{t} \right)^2 \end{aligned}$$

Equation 9.

The values for $\bar{X}_i$, $\bar{X}_j$, and s_{ij} are calculated from the dummy run of 32 replications.

3.6 Overall Method

In the section outlining the tests to be carried out a number of policies were described. To simulate the granting of options under these policies a program was designed which simulates a given number of options for each policy and analyses the results of the options separately and then in pairs. For example, when asked to simulate n options it chooses[8] n different starting dates randomly from amongst those available[9] and

---

7. In Equation 9, t is the value calculated in the preceding sub-section on significance levels.

8. Using a method outlined later.

simulates an option starting from each date.

For each option the following steps are gone through:

— The volatility is calculated using the method described later in this section.

— The option is valued by the Black and Scholes (1973) formula as modified by Garman and Kohlhagen (1983) using the interest and exchange rates as at the close of trading the day before the option starts.

— If the option is to be simulated on generated data this is produced as described later.

— The option is simulated using the synthetic option algorithm.

— The resulting profit or loss is then recorded.

Once the results have been collected the mean and the t-statistic of the mean are calculated for each policy. Then each pair of policies is compared with the mean of the differences and its t-statistic produced. These results are presented and discussed in the results section.

3.7 Normal Curve Integral

The option valuation formula as used in this paper requires the calculation of $N(x)$ for various values of x. $N(x)$ is the area under the cumulative normal density function, ie:

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}t^2} dt$$

The first attempt to calculate $N(x)$ used a polynomial approximation as described in

---

9. A date is available if an option written from that date matures before the end of the data set. With the provided data and 28 day options there are 619 possible starting days.

M. Abramovitz and I. A. Stegun (1964: 932). This approximation is given below along with its error bound.

$$N(x) = 1 - \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2}(b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5) + \varepsilon(x)$$

$$t = \frac{1}{1 + px}$$

$$|\varepsilon(x)| < 7.5 \times 10^{-8}$$

$$p = 0.231\,641\,9$$

$$b_1 = 0.319\,381\,530$$

$$b_2 = -0.356\,563\,782$$

$$b_3 = 1.781\,477\,937$$

$$b_4 = -1.821\,255\,978$$

$$b_5 = 1.330\,274\,429$$

However as $N(-6) \approx 9.866 \times 10^{-10}$ the above approximation for $N(-6)$ does not result in an answer with any significant digits at all. The same problem happens for all values of $x < -5.5$ and thus as the project requires values of $N(x)$ for x in this range a better approximation was required.

The method used came from the Collected Algorithms from Communications of the Association of Computing Machinery (CACM). The algorithm used is number 304 originally developed by I. D. Hill and S. A. Joyce (1967). However it was necessary to use two of the modifications suggested in later editions of the CACM. The first modification was by Arthur G. Adams (1969) which resulted in a speed improvement and another described by Bo Holmgrem (1970) which was necessary to avoid overflows occurring for some values of x. As described the algorithm uses a convergent power series for $N(x)$ if x lies in the central area of the curve and a continued fraction if x lies in one of the tails, see I. D. Hill and S. A. Joyce (1967) for details.

Hill and Joyce (1967) state ‘‘the procedure works virtually to the accuracy of the machine (provided that the constant $1/\sqrt{2\pi}$ is given to this accuracy) for $x \leq 7$ but to 1

decimal place less than the accuracy of the machine for $x > 7$''.[10] The limited testing that was done appeared to support the claim. The program as used is called *Sim/normal.c* and is in listed in Appendix B.

3.8 Volatility

Because it needs to be estimated in some way from historical data the currency volatility is probably the input which is most difficult to calculate. It appears that the most realistic way of calculating the volatility is from previous currency movements over a time period comparable in length to that of the option being estimated. Thus to calculate the volatility for a three month option, the data from the three months preceding the start of the option should be used.

The formula to be used (Cox and Rubinstein, 1985: 256) is given by Equation 10.

---

10. This quote refers to upper tail values: for lower tail values as used in this paper the direction of the inequalities and the sign of the 7 should be reversed.

$$
\begin{aligned}
v &= \frac{(\frac{n}{2})^{\frac{1}{2}}(\frac{n}{2}-\frac{3}{2})!}{(\frac{n}{2}-1)!}\left(\frac{1}{n}\sum_{k=1}^{n}(\log R_k - u)^2\right)^{\frac{1}{2}} \\
&= \left(\frac{(\frac{n}{2})^{\frac{1}{2}}(\frac{n}{2}-\frac{3}{2})!}{(\frac{n}{2}-1)!}\right)\frac{1}{n}\left(n\sum_{k=1}^{n}(\log R_k)^2 - \left(\sum_{k=1}^{n}\log R_k\right)^2\right)^{\frac{1}{2}} \\
&\approx \left(\frac{1}{n-1}\sum_{k=1}^{n}(\log R_k - u)^2\right)^{\frac{1}{2}} \\
&= \frac{1}{[n(n-1)]^{\frac{1}{2}}}\left(n\sum_{k=1}^{n}(\log R_k)^2 - \left(\sum_{k=1}^{n}\log R_k\right)^2\right)^{\frac{1}{2}}
\end{aligned}
$$

where:

$$u = \frac{1}{n}\sum_{k=1}^{n}\log R_k$$

$$R_j = \frac{S_j}{S_{j-1}}$$

$$x! = \int_0^\infty e^{-v}v^x dv = \Gamma(x+1)$$

Equation 10.

In this formula n is the number of price relatives, R_j's, which is one less than the number of spot price observations, S_j's. The formula gives the volatility of the currency over a length of time equal to the period of the observations. To get an annual volatility multiply v by the square root of the number of observations in a year. For example, for daily price data v, the annualized volatility, is given by Equation 11[11].

$$v = 251^{\frac{1}{2}}v$$

Equation 11.

3.9 The Delta

The delta (Δ) of an option is the sensitivity of the value of the option to a change in the currency price, ie the partial derivative of the value of the option to the currency price.

---

11. The equation is correct if the daily volatilities are independent or, as is assumed here, constant. Also, there are about 251 working days in a year.

For foreign exchange currency options the delta is as given in Equation 12 (Garman and Kohlhagen, 1983, and Hoag and McKay, 1984).

$$\Delta = e^{-ft} N(d)$$

where:

$$d = \frac{\ln(\frac{s}{c}) + (r - f + \frac{1}{2} v^2)t}{v\sqrt{t}}$$

Equation 12.

3.10 Random Number Generators

This project needs two good sources of random numbers, one set to generate a random walk for the simulated exchange rate data and the other to choose the starting date for the simulated options. The later use became less important as every possible starting date was used in later simulation runs.

3.10.1 Uniform Random Numbers

Uniform random numbers are needed to generate both the above two series and if the uniform random numbers used are reliable the generators can be proved correct. The only problem is to generate independent numbers uniform on the range [0, 1). The first attempt to do this used a Multiplicative Generator with Prime Modulus (Bratley et al 1983: 199), ie:

$$X_i = 16807 X_{i-1} \bmod (2^{31} - 1)$$

$$U_i = \frac{X_i}{2^{31} - 1}$$

The generator is a special case of the Linear Congruential class of generators where:

$$X_i = (aX_{i-1} + c) \bmod m$$

The above generator for U_i was checked with the χ^2 and Maximum-of-t (Knuth, 1981: 68) tests. Appendix C contains the program *Test/uniftest.c* which was used for these tests.

The χ^2 test was carried out with 1360 observations and $\left\lfloor 4 \times 1360^{2/5} \right\rfloor = 71$ equal sized cells (Bratley et al, 1983: 205). For the initial seed 1234567890, $\chi^2 = 50.0809$ which is less than the critical value $\chi^2_{0.050} = 90.5312$ with 70 degrees of freedom. Thus it is not possible to reject the null hypothesis that the observations came from a uniform $[0, 1)$ distribution.

The Maximum-of-t test is as follows:

1. Compute:

$$V_j = \max(U_{tj}, U_{tj+1}, \ldots, U_{tj+t-1}) \quad \text{for } 0 \le j < n$$

2. Apply the Kolmogorov-Smirnov test to the sequence $V_0, V_1, \ldots, V_{n-1}$, with the distribution function $F(x) = x^t, 0 \le x \le 1$. The Kolmogorov-Smirnov test requires n observations $X_1, X_2, \ldots, X_n$, ordered so that $X_{i+1} \ge X_i$ for $i = 1, \ldots, n-1$.

3. Compute:

$$K_n^+ = \sqrt{n} \max_{j=1,\ldots,n} \left[\frac{j}{n} - F(X_j) \right]$$
$$K_n^- = \sqrt{n} \max_{j=1,\ldots,n} \left[F(X_j) - \frac{j-1}{n} \right]$$

4. Compare K_n^+ and K_n^- to the critical values as listed in Knuth (1981: 48).

In this case $n = 30$ and $t = 45$ resulting in $K_{30}^+ = 0.3648$ and $K_{30}^- = 0.6960$. The critical value at the 95% level is 0.8036 so the null hypothesis is accepted.

Once the generator had passed the above two tests it was used to generate random walks to simulate currency options. However, it was discovered that call options had a significantly positive profit and put options a significant loss. This behaviour was

unexpected so the above generator was further checked. With reference to Bratley et al (1983: 210) it was found that when the Box-Muller method, as discussed later, is used to generate normal variates from these U_i's the resulting normals are poor. So another method of generating random variables was obtained — the Tausworthe generator which is as follows:

$$h(x) = x^k + x^q + 1$$
$$X_i = h(X_{i-1})$$

such that $h(x)$ is a primitive polynomial, k is equal to the number of non-sign bits in the computer word and $k \geq 2q$. For a PDP-11 using **long**s, $k = 31$ and $q = 13$.

This generator was tested as above with initial seed 524287, resulting in a χ^2 statistic of 82.2397, $K_{30}^{+} = 0.2410$ and $K_{30}^{-} = 0.8001$. All these values are less than the critical values, thus the generator was accepted. When the models mentioned above were rerun both the profits and losses became insignificant. The generator was therefore used for the rest of the project.

3.10.2 Random Sampling

When the starting dates for option simulations are being chosen it is necessary to have an unbiased choice of n dates at random from the N possible starting dates. Knuth (1981: 136) gives the following method:

select the $(t+1)$st record with probability $(n-m)/(N-t)$, if m items have already been selected.

This method has the following desirable properties:

— *n* records are always selected,

— The sample is completely unbiased: the probability that any record is selected is n/N.

3.11 Synthetic Option Algorithm

The synthetic option algorithm is used to simulate the granting of an option, the receipt of the premium[12], managing the portfolio and the possible delivery of foreign currency should the option be exercised. To perform this algorithm the following information is needed:

c option exercise price

v the currency volatility, annualised

s an array of the t exchange rates during the life of the option, Australian dollars per United States dollar, closing prices

r an array of the t annual domestic interest rates during the life of the option

f an array of the t annual foreign interest rates during the life of the option

T(x) a function which gives the transaction costs involved in buying or selling x units of foreign currency, in the experiments performed in this paper T(x) is assumed to be zero

Δ(.) a function giving the hedge ratio (delta), the functional form for this value is as derived by Garman and Kohlhagen and is given in the literature review

---

12. The premium is the price of the option, paid to the grantor by the purchaser.

w(.) the premium required to purchase the option, the functional form for this value is as derived by Garman and Kohlhagen and is given in the literature review

n the length of the option in units

m the number of units per year

It is assumed that the option is granted at the beginning of the first day of the option period and may be exercised only at the close of trading on the maturity date. The option is valued using the closing prices from the day previous to its granting. All portfolio adjustments are done at market closing time.

The algorithm used to calculate the cost of using a synthetic option[13] is as follows:

13. The granted option is assumed to be for the delivery of one United States dollar at a price of c Australian dollars at the end of t periods. This simplification is only correct if T(x) is linearly homogeneous in the region of interest.

```
a = w(s[0], n / m, r[0], f[0], c, v)
u = 0
i = 1
y[0] = y[1] − 1

while n > y[i] − y[0]
        u = u + f[i] × u / m × (y[i] − y[i − 1])
        a = a + r[i] × a / m × (y[i] − y[i − 1])
        d = Δ(s[i], (n − y[i] + y[0]) / m, r[i], f[i], c, v) − u
        u = u + d
        a = a − d × s[i] − T(d)
        i = i + 1

if n != y[i] − y[0] then error

u = u + f[i] × u / m × (y[i] − y[i − 1])
a = a + r[i] × a / m × (y[i] − y[i − 1])

if s[i] > c then
        profit = u × s[i] + a − s[i] + c
else
        profit = u × s[i] + a
```

Where:

a is the quantity of Australian dollars held at any time

u is the quantity of United States dollars held at any time

and assuming T(0)=0. At all time points the value of the portfolio should equal or exceed the current value of the option. Thus if and when the option is exercised the required United States dollar can be purchased from the exercise price, c, and the value of the portfolio. Any excess or deficit is the profit or loss resulting from such a transaction.

3.12 Variance Estimation

An unbiased estimate of the variance of the profits from each replication is required so as to be able to calculate t-statistics and confidence intervals. The naive method of calculating the variance assuming independent observations $x_1, \ldots, x_n$ viz,

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 ,$$

is not usable in this experiment as the observations may be seriously correlated. Correlation may come about through two sources:

— Correlation in the currency price movements over time causing correlation in the profits from options.

— As option starting values are chosen randomly and there is only room for 32 non-overlapping 28 day options in the data set, some of the options will overlap and use the same data points for parts of the simulation. This problem will get worse as the number of replications increases to the maximum of 619 at which time many data points will occur in about 20 replications.

If the correlation is ignored in the calculation of the variance then the estimate obtained may be seriously biased. To avoid this problem the method of Fishman (1967) was used. This method is also described in Fishman (1968), Kleijnen (1975: 454-468) and to a superficial extent in Wagner (1969: 912).

Fishman's method of estimating σ^2 corrects for the correlation in the observations by explicitly including the covariance terms (r_i) with lag i in the estimation procedure for σ^2. His formula is:

$$\sigma^2 = \frac{1}{1 - k/n}\left[r_0 + 2\sum_{i=1}^{k}(1 - \frac{i}{k})r_i \right]$$

where:

$$r_i = \frac{1}{n}\sum_{j=1}^{n-i}\left[(x_i - \bar{x})(x_{j+i} - \bar{x}) \right]$$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Equation 13.

and

n number of observations

k number of lag coefficients taken into account

When $k = 0$ or $r_i = 0$ for *all* $i > 0$ Fishman's equation collapses to the naive approach given earlier. The assumptions underlying Fishman's method are:

i. The joint probability function of the observations $x_1, x_2, \ldots, x_i, \ldots, x_n$ is not a function of i, or more specifically, the covariance between x_i and x_{i+j} does not vary with i but only with j.

ii. n is large.

iii. The lag coefficients (r_i) vanish after k lags, ie $r_i = 0$ for $i > k$.

If these assumptions are correct, as is supposed throughout this paper, the only problem is to find a value for k.

The following discussion of the choice of k comes from Fishman (1968: 290-291). In the choice of k there are two opposing factors to consider. These are:

i. Mathematical convergence: as $k \to \infty$ the value of the formula for σ^2 tends towards the desired mathematical limit.

ii. Convergence in probability: as k becomes large the variance of the estimate of σ^2 increases, ie the estimate becomes less statistically reliable.

The two factors indicate that for good resolution k should be large, but for reliability k should be small. Fishman gives the following subjective method for choosing k[14]:

> It is suggested that k not exceed $n/4$ and in general be kept much smaller. One may easily compute σ^2 for several values of k, say $n/32$, $n/16$, $n/8$, $3n/16$, and $n/4$. Doing so permits the experimenter to decide, albeit subjectively, when σ^2 is well resolved.

The effect of using Fishman's procedure in this application is given in the results section of the paper.

---

14. The notation in the quote has been altered to be consistent with the rest of the paper.

4. Data

This section describes the data as supplied and the transformations applied to it to put it into a form suitable for use in this project. Also presented are some plots of the data and a discussion of these plots and the trends evident in them.

4.1 Data Transformations

Data was provided by Macquarie Bank on a magnetic tape and consisted of three sets of data.

The first set was information on the United States versus Australian dollar exchange rate. The data was provided on a daily basis from 12/12/83 to 1/9/86, some 681 records in all. Each day's information consisted of the date, opening rate, daily highest value, daily lowest value and the closing price. All exchange rates were provided to 4 significant digits accuracy.

The second data set was the United States dollar 24 hour call interest rate and consisted of 851 records from 1/6/83 to 2/9/86. These interest rates were accurate to 3 significant digits.

The final data set was the Australian dollar 24 hour call interest rate from 15/2/84 to 2/9/86, some 662 records. These interest rates were also provided to 3 significant digits accuracy.

The information was then transformed into a more usable form. Of the currency data, only the closing price was required for each day and rather than the United States versus Australian dollar rate the Australian versus United States exchange rate was

required. Thus, all but the last item of each data record was deleted and the remaining one was inverted to give the required rate.

The three data sets were then combined and trimmed to give a single data set covering the period from 15/2/84 to 1/9/86. It was found that some days had exchange rates and not one or other of the interest rates and visa versa. This was due to the differing holiday periods in the different countries and other collection problems of the bank. To solve these difficulties and get a consistent data set, all days without exchange rate information were deleted and for those days with exchange rate data but a missing interest rate the interest rate for the previous business day was inserted. The justification for this procedure was that days without exchange rate data were assumed to be days in which the currency markets were closed and thus no trading possible. Also, when days were missing interest rates it was most likely the bank, for some reason, was unable to record that day's interest rate, in which case the previous day's interest is a reasonable estimate of the missing value. However, the adjustment was only required in a few cases so the overall effect should be quite small.

Once these corrections had been carried out the data consisted of 638 records covering the period from 15/2/84 to 1/9/86. For the year 1985 (the only year for which full data was available) there were 251 records (ie. business days).

In undertaking this project it was found that use could be made of the currency price data excluded above, for example in the calculation of exchange rate volatilities. So the excluded currency data was reintroduced and combined with dummy (zero) interest rates resulting in a data set with exchange rates covering the period from 12/12/83 to

1/9/86 and interest rate information from 15/2/84 to 1/9/86. This provided a final data set of 678 records, a sample of which is included in Table 2 below.

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
08/02/84	1.0817	0	0
09/02/84	1.0755	0	0
10/02/84	1.0748	0	0
13/02/84	1.0735	0	0
14/02/84	1.0727	0	0
15/02/84	1.0661	9.75	9.69
16/02/84	1.0569	8	9.56
17/02/84	1.0588	8	9.56
20/02/84	1.0588	10	9.69
21/02/84	1.0650	10.5	9.63
22/02/84	1.0650	11.25	9.5
23/02/84	1.0588	11.5	9.75
24/02/84	1.0610	11.5	9.75
27/02/84	1.0607	11	9.88
28/02/84	1.0601	11	9.88
29/02/84	1.0609	12	9.75
01/03/84	1.0600	11.5	9.81
02/03/84	1.0599	12	9.81
05/03/84	1.0499	11	9.88
06/03/84	1.0466	11.5	9.88
07/03/84	1.0449	12	9.88

TABLE 2. A Sample of Data

A listing of the complete set of data as well as a graph of each of the three data series is supplied in Appendix A. These graphs show up some data points whose values are somewhat questionable. For example, the United States interest rate for 8/7/86 is 7%, for 9/7/86 it jumps to 9.94% and then drops back to 6.88% on 10/7/86. No attempt has been made to investigate or substantiate these and several other unexpected changes in the United States interest rate series.

4.2 Discussion

The plot of the exchange rate data appears to indicate a definite upward trend. However, upon closer inspection the graph consists of three horizontal segments joined by steep vertical rises. The first jump was in February 1985 when the exchange rate increased from 1.2320 at the start of the month to 1.4002 at the end. The next rise was in June-July 1986 when the rate increased from 1.4110 to 1.6756. Other than in those two periods the exchange rate moved constantly but without definite direction. This implies for most of the period of the simulations performed in this paper the random walk assumption supposed to be underlying currency price movements is not obviously violated. Although the random walk assumption is not directly tested, simulations are done both on the actual data and some generated data which does obey the random walk criterion. These simulations are compared and any difference will indicate that the data does not obey the random walk restriction.

The partial derivative of the price of a call option with respect to the exchange rate is positive, so a rising exchange rate will cause currency call options to be systematically undervalued by the theoretical formula. However, most of the options simulated are of very short duration (ie. 28 days) and so only a few options studied will include the sudden price jumps evident in the data. Options written in January-February 1985 and in May to July 1986 will include these steep price rises.

The graph of the Australian interest rate shows a small upward trend in some sections but the effect is not as obvious as in the exchange rate series. The theoretical model assumes constant interest rates so the upward movement violates the assumption

of constant and known interest rates. Later in this paper tests will be done which, along with other things, check to see whether this violation affects the profits received from writing options.

The United States interest rate has a quite definite downward movement over the 32 months of observations. For example, from 17/9/84 to 7/11/84 the United States interest rate dropped from 11.69% to 9.25% with hardly any upward movements. The partial derivative of the value of a call option with respect to the foreign interest rate is negative so the interest rates used will cause call options written during the periods of downward trend to be undervalued. The tests mentioned above will attempt to establish the importance of these violations of the underlying assumptions of the model.

4.3 Generated Data

Some of the experiments performed require data that satisfies the assumptions of the option pricing and synthetic portfolio models. The method of generating this data is described in this sub-section.

The models assume that the interest rates throughout the life of the option are known and constant. Interest rates that satisfy this assumption are generated by using the interest rate that was used to value the option being simulated, throughout the life of the option.

Exchange rates are assumed to follow a random walk with constant variance. An exchange rate series that obeys this assumption is generated by starting with the currency price used to value option and generating the successive price levels by the following formula:

$x_{i+1} = e^y$
where:

> y is a normally distributed random variable with mean $\log(x_i)$ and variance the square of the calculated volatility of the currency.

4.3.1 Normal Random Numbers

The Box-Muller method is used to generate a standard normal variate X from two independent uniform random variables U_1 and U_2 by:

$$X = \cos(2\pi U_1)\sqrt{-2\log(U_2)}$$

As discussed above this method does not work well if the U_i's are generated from a Linear Congruential Generator.

5. Results

A large number of simulations runs were performed using the method outlined in the experiment description section. Once the number of correlation terms to use in the variance calculations has been determined, the results of the simulations will be analysed in terms of the stated objectives.

5.1 Variance Estimation Results

The standard errors used in the experiments are calculated by the method of Fishman (1968) as outlined in the section titled ‘‘The Experiment’’. This sub-section will present the standard errors obtained using Fishman’s formula for four of the simulations performed. Consideration of the various values of s_{ij} listed will indicate when the estimates are ‘‘well resolved’’.

The experiments chosen to for use in the determination of k were picked on the grounds that they were typical of the complete set of 33 tests. The experiments are:

A. Daily portfolio adjustment and short volatility measure on actual data minus daily portfolio adjustment and short volatility measure on simulated data.

B. Weekly adjustment and long volatility measure on actual data minus weekly adjustment and long volatility measure on simulated exchange rate data.

C. Daily adjustment and short volatility measure on actual data minus no portfolio and short volatility measure on actual data.

D. No portfolio and short volatility measure on simulated data minus no portfolio and long volatility on simulated data.

The following tables list the standard deviations obtained using six different values of *k* on the four experiments selected.

k	**Experiment**			
	A	B	C	D
0	0.00205185	0.00387344	0.00824791	0.00265111
n/32	0.00208468	0.00391542	0.00837988	0.00269353
n/16	0.00235764	0.00481509	0.00983296	0.00220890
n/8	0.00288892	0.00598816	0.0103735	0.00300821
3*n*/16	0.00312749	0.00670975	0.00999308	0.00296079
n/4	0.00317173	0.00708697	0.00966055	0.00282733

TABLE 3. Standard Deviations For Various Values Of *k* With *n* = 32

k	**Experiment**			
	A	B	C	D
0	0.000921929	0.00104065	0.00278288	0.00125043
n/32	0.00162947	0.00205683	0.00648379	0.00117436
n/16	0.00149967	0.00261315	0.00739321	0.00113711
n/8	0.00137110	0.00330185	0.00828630	0.00131019
3*n*/16	0.00132065	0.00359756	0.00849226	0.00148495
n/4	0.00119427	0.00373178	0.00846397	0.00158344

TABLE 4. Standard Deviations For Various Values Of *k* With *n* = 256

k	**Experiment**			
	A	B	C	D
0	0.000639784	0.000698716	0.00162885	0.000817972
n/32	0.00140184	0.00182965	0.00580680	0.000710943
n/16	0.00124201	0.00237934	0.00669492	0.000796458
n/8	0.00134592	0.00304908	0.00757188	0.000867254
3*n*/16	0.00139816	0.00330627	0.00748008	0.000835030
n/4	0.00144059	0.00341220	0.00724107	0.000818568

TABLE 5. Standard Deviations For Various Values Of *k* With *n* = 619

The results listed show that as *k* is increased from zero towards *n*/8 the standard deviations increase. Once *k* reaches *n*/8 the standard deviations, in some cases, start to drop again. The peak at *n*/8 is most noticeable in experiments C and D.

Fishman (1986) recommends that in general k be much less than $n/4$. And as the values of the standard errors appear to level off at $k = n/8$, this is the value of k used in all the following experiments.

5.2 Robustness Tests Results

This section presents the results of the experiments outlined in the ‘‘Testing For Robustness’’ sub-section. Each set of tests described there has a corresponding sub-section here, were the results of the tests are presented and discussed. Conclusions are then drawn in terms of the objective of the test as specified in the earlier section.

5.2.1 Exchange Rates

The results from the dummy run of 32 replications of the six comparisons designed to test the effect of the relaxation of the exchange rate movements assumption are given in Table 6.

Volatility Measure	Portfolio Adjustment Frequency	Difference in Mean Profit (t-statistic)	Replications Required for Significance
short	daily	0.000639333 (0.226)	8328
	weekly	0.00119347 (0.277)	5544
	no portfolio	0.0209414 (1.45)	202
long	daily	0.00506905 (1.12)	339
	weekly	0.00626555 (1.05)	386
	no portfolio	0.0285879 (2.27)	83

TABLE 6. Exchange Rate Assumption Tests: Difference in Mean Profit Between Simulated Data and Data With Actual Exchange Rates, 32 Replications

With the limited data available it is not possible to use a sample size larger than 619. In Table 6 the comparisons using the short volatility measure and a synthetic portfolio both require more than 619 replications for a definite conclusion to be drawn. All that can be done in this situation is to repeat the simulation using the maximum number of replications possible. This will allow definite conclusions to to be reached for those four comparisons whose replication requirement is satisfied. For the other two comparisons, the direction and relative magnitude of the change in mean profit difference and t-statistic will allow a tentative conclusion about the significance of the statistics to be reached.

Volatility Measure	**Portfolio Adjustment Frequency**	**Difference in Mean Profit (t-statistic)**	**Replications Required for Significance**
short	daily	−0.000138955 (0.102)	644386
	weekly	0.000489463 (0.473)	29966
	no portfolio	0.0110290 (1.48)	3061
long	daily	0.00309832 (1.20)	4656
	weekly	0.00417919 (1.37)	3572
	no portfolio	0.0146620 (1.72)	2266

TABLE 7. Exchange Rate Assumption Tests: Difference in Mean Profit Between Actual Data and Data With Simulated Exchange Rates, 619 Replications

Table 7 shows that, for the four comparisons whose replication requirement was satisfied, the difference in profits for options written using the two types of data are insignificant.

For the short volatility measure and daily adjustment frequency comparison case (the first comparison in the above two tables) increasing the number of replications lead to a change in the sign of the difference in profits and a lowering of the significance of this difference from 0.226 to 0.102. The number of replications now required to get a significant comparison has increased to 644386. These observations result in the conclusion that the difference in policies is collapsing towards zero.

For the short volatility measure and weekly portfolio adjustment case (the second comparison in the tables) the result of increasing the the number of comparisons is that mean difference dropped from 0.001 down to 0.0005 and the t-statistic increased from 0.277 to 0.473. However, the value of the t-statistic did not increase as much as the increased sample size would require if the performances did significantly differ. This is because the required sample size rose from 5544 to 29966 once the new replications were taken into account in the calculation of the number of replications required. These factors all tend to indicate that there is no difference in the performance of the option strategy under the two types of data (simulated exchange rates and historical exchange rates).

The results in this sub-section in four cases indicate conclusively that the relaxation of the the random walk exchange rate assumption does not affect the returns from granting options. The rising number of replications required for significance in the other two case also appears to support this supposition. Thus, all the available evidence leads to the conclusion that the relaxation of the assumption of random walk exchange rates does not lead to a significant change in the returns from granting options. This conclusion holds for both constant known interest rates and historical interest rates.

5.2.2 Interest Rates

The results for the dummy run of 32 replications for the six comparisons involved in the test of the importance of the assumption that interest rates are known and constant throughout the life of an option are listed in Table 8.

Volatility Measure	**Portfolio Adjustment Frequency**	**Difference in Mean Profit (t-statistic)**	**Replications Required for Significance**
short	daily	0.0000906839 (0.897)	529
	weekly	0.0000623215 (0.497)	1722
	no portfolio	−0.00000399173 (1.24)	277
long	daily	0.000102943 (0.981)	442
	weekly	0.0000759622 (0.589)	1226
	no portfolio	−0.00000134421 (0.585)	1243

TABLE 8. Interest Rate Assumption Tests: Difference in Mean Profit Between Simulated Data and Data With Historical Interest Rates, 32 Replications

The results for the dummy run to test the interest rate assumption indicate, as for the exchange rate tests, that some of the comparisons require more replications than are possible. As before, the solution is to use the maximum sample size possible and examine the changes in the resulting statistics.

Volatility Measure	Portfolio Adjustment Frequency	Difference in Mean Profit (t-statistic)	Replications Required for Significance
short	daily	0.0000799787 (1.08)	5748
	weekly	0.0000814578 (1.09)	5643
	no portfolio	−0.00000298483 (1.05)	6081
long	daily	0.0000856485 (1.14)	5159
	weekly	0.0000876108 (1.15)	5069
	no portfolio	−0.00000123754 (0.536)	23335

TABLE 9. Interest Rate Assumption Tests: Difference in Mean Profit Between Simulated Data and Data With Historical Interest Rates, 619 Replications

All the mean profit differences in the simulation run with 619 replications are highly insignificant compared with the required significance level of between 2.576 and 3.291. This allows the conclusion that, for the daily adjusted and no portfolio options using the short volatility measure and the daily adjusted portfolio using the short portfolio measure, the relaxation of the interest rate assumption does not change the profits available from granting options.

In the comparison between the profits from options written using the long volatility measure and no synthetic portfolio the magnitude of the mean profit dropped from 0.0000013 down to 0.0000012 and the significance of this value declined from 0.585 to 0.536. It is therefore possible to conclude that the mean profit difference is tending towards zero as the sample size increases.

For the comparisons using the weekly portfolio adjustment frequency the

conclusion is not so clear cut. After the sample size is increased from 32 to 619 the two means both increase and the corresponding significance levels also rise, but not to anywhere near the minimum significance level of 2.576. However the updated number of replications required for significance also increased from under 2000 to over 5000 in both cases. This increase in the number of replications gives some evidence that the mean difference is moving towards zero.

The mean profit differences for the different interest reate series are not significantly different from zero and except for the weekly adjusted portfolios there is strong evidence that the relaxation of the constant interest rate assumption does not affect the returns from granting options.

5.2.3 Exchange Rates and Interest Rates

The results for the dummy run of 32 replications for the six comparisons involved in the test of the importance of the joint assumption that exchange rates follow a random walk with constant volatility and that interest rates are known and constant throughout the life of an option are listed in Table 10.

Volatility Measure	Portfolio Adjustment Frequency	Difference in Mean Profit (t-statistic)	Replications Required for Significance
short	daily	−0.000764796 (0.265)	6057
	weekly	−0.00131007 (0.302)	4664
	no portfolio	−0.0209374 (1.45)	202
long	daily	−0.00522632 (1.13)	333
	weekly	−0.00642350 (1.06)	379
	no portfolio	−0.0285866 (2.27)	83

TABLE 10. Exchange Rate and Interest Rate Assumption Tests: Difference in Mean Profit Between Actual Data and Simulated Data, 32 Replications

The results for the dummy run in Table 10 give required numbers of replications greater the maximum possible in the first two cases. As before, the approach taken in this situation is to use the maximum sample size of 619 and examine the changes in the resulting statistics.

Volatility Measure	Portfolio Adjustment Frequency	Difference in Mean Profit (t-statistic)	Replications Required for Significance
short	daily	0.0000174789 (0.013)	39669772
	weekly	−0.000579021 (0.562)	21226
	no portfolio	−0.0110260 (1.48)	3061
long	daily	−0.00322707 (1.23)	4431
	weekly	−0.00427169 (1.38)	3520
	no portfolio	−0.0146607 (1.72)	2266

TABLE 11. Exchange Rate and Interest Rate Assumption Tests: Difference in Mean Profit Between Actual Data and Simulated Data, 619 Replications

All the difference in mean profit in Table 11 are insignificant compared to the minimum significance level of 2.576. The last four of the comparisons in the table have had the required number of replications performed, thus as the results are not significantly different for zero it is possible to conclude that in these case the joint relaxation of the exchange rate and interest rate assumptions does not affect the returns from granting options.

For the mean profit difference between the data series using the short volatility measure and a synthetic portfolio daily the sign of the difference changed as the number of replications increased. As well the absolute value of the differences and its t-statistic also decreased. The large reductions of the magnitude of the statistic and the corresponding drop in significance leads to the conclusion that the mean profit difference for this options type is tending towards zero.

The results of the comparison using the short volatility measure and a portfolio adjusted weekly are not so easy to interpret. The absolute value of the mean profit difference decreased as the number of replications increased and the significance of the statistic increased marginally from 0.302 to 0.562. However, the number of replications required for significance rose from 4664 to 21226 indicating that it is unlikely that the statistic will become significant.

The results in this sub-section in four cases indicate conclusively that the relaxation of the exchange rate and interest rate assumptions does not affect the profits from granting options. The other two sets of results also appear to support this conclusion but not as strongly. However, there is no evidence to suggest that the relaxation of the assumptions does make a difference to the profits obtained.

5.2.4 Volatility Measure

The results for the dummy run of 32 replications for the three comparisons involved in the test of the two different volatility measures are listed in Table 12.

Portfolio Adjustment Frequency	**Difference in Mean Profit (t-statistic)**	**Replications Required for Significance**
daily	0.00460441 (1.77)	136
weekly	0.00511800 (1.72)	144
no portfolio	0.00486182 (1.72)	144

TABLE 12. Volatility Measure Tests: Difference in Mean Profit Between Short Volatility Measure and Long Volatility Measure on Actual Data, 32 Replications

The results in Table 12 indicate that a simulation run with 256 replications will be

more than sufficient to make conclusions about the preferred (most profitable) volatility measure. The simulation run with 256 replications is given below.

Portfolio Adjustment Frequency	**Difference in Mean Profit (t-statistic)**	**Replications Required for Significance**
daily	0.00355398 (1.13)	2171
weekly	0.00357083 (1.14)	2133
no portfolio	0.00325569 (1.11)	2250

TABLE 13. Volatility Measure Tests: Difference in Mean Profit Between Short Volatility Measure and Long Volatility Measure on Actual Data, 256 Replications

All the mean profit differences in Table 13 are insignificant compared with the minimum significance level of 2.576. Is is thus possible to conclude that the choice of volatility measure does not make a difference to the profits from writting options over the period of the data set.

5.2.5 Portfolio Adjustment Frequency

The results for the dummy run of 32 replications for the four comparisons involved in the test of the two different portfolio adjustment frequencies are listed in Table 14.

Data Used To Simulate The Option	Volatility Measure	Difference In Mean Profit (t-statistic)	Replications Required For Significance
actual	short	0.00195958 (1.99)	107
	long	0.00247318 (1.76)	137
generated	short	0.00141431 (0.641)	1035
	long	0.00127601 (0.719)	823

TABLE 14. Portfolio Adjustment Frequency Tests: Difference in Mean Profit Between Daily Adjusted Portfolios and Weekly Adjusted Portfolios, 32 Replications

The results for the dummy run indicate that for two of the comparisons that more than the maximum number of comparisons are required, thus Table 15 contains the results for a run with the maximum of 619 replications.

Data Used To Simulate The Option	Volatility Measure	Difference In Mean Profit (t-statistic)	Replications Required For Significance
actual	short	0.000702090 (0.815)	10093
	long	0.00101859 (1.08)	5748
generated	short	0.000105590 (0.544)	22654
	long	−0.0000260221 (0.128)	409191

TABLE 15. Portfolio Adjustment Frequency Tests: Difference in Mean Profit Between Daily Adjusted Portfolios and Weekly Adjusted Portfolios, 619 Replications

The comparisons using actual data for which the required number of replications were performed are insignificant. This allows the conclusion that when actual data over the period of the data set is used the choice of volatility measure does not make a difference to the profits from writting options.

For the generated data set it was not possible to carry out the required number of replications. However, once the number of replications was increased to the maximum the mean performance difference for both the long and short volatility measure dropped by at least an order of magnitude. The significance of the mean difference in performance also dropped. These changes indicate that the mean profit difference is tending towards zero.

The low significance levels discussed above lead to the conclusion that the choice of portfolio adjustment frequency does not significantly affect the profits gained from granting options.

5.2.6 Synthetic Portfolio

The results for the dummy run of 32 replications for the eight comparisons involved in the test of the the desirablility of using a synthetic option portfolio are listed in Table 16.

Data Used To Simulate The Option	Volatility Measure	Portfolio Adjustment Frequency	Difference In Mean Profit (t-statistic)	Replications Required For Significance
actual	short	daily	0.0182654 (1.76)	137
		weekly	0.0163059 (1.58)	170
	long	daily	0.0185229 (1.79)	133
		weekly	0.0160497 (1.57)	173
generated	short	daily	−0.00190713 (0.316)	4260
		weekly	−0.00332144 (0.535)	1486
	long	daily	−0.00483742 (1.32)	244
		weekly	−0.00611343 (1.66)	154

TABLE 16. Synthetic Portfolio Tests: Difference in Mean Profit Between Synthetic Portfolio and No Portfolio Options, 32 Replications

The results in Table 16 show that a simulation run with the maximum of 619 replications will be sufficient for all but two of the comparisons. The two comparisons on generated data using the short volatility measure require more than the maximum so the changes in the mean profit difference and t-statistic will have to guide any conclusions made. The results for a run with 619 replications are in Table 17.

Data Used To Simulate The Option	Volatility Measure	Portfolio Adjustment Frequency	Difference In Mean Profit (t-statistic)	Replications Required For Significance
actual	short	daily	0.00878764 (1.16)	4982
		weekly	0.00808555 (1.08)	5748
	long	daily	0.00868155 (1.17)	4898
		weekly	0.00766296 (1.06)	5967
generated	short	daily	−0.00225583 (1.18)	4815
		weekly	−0.00236142 (1.20)	4656
	long	daily	−0.00275212 (1.71)	2293
		weekly	−0.00272609 (1.61)	2586

TABLE 17. Synthetic Portfolio Tests: Difference in Mean Profit Between Synthetic Portfolio and No Portfolio Options, 619 Replications

All the results in Table 17 are insignificant compared to the minimum significance level of 2.576. Thus the results allow the conclusion that for the actual data series and the long volatility measure options having a synthetic portfolio does not significantly affect the return gained from granting options.

For the options using the long volatility measure on the generated a data the situation is not as clear. The mean profit differences for both these comparisons do not change by much when the number of replications is increased and the t-statistics increase. Also for one of the cases (the daily adjusted portfolio) the number of replications required does not rise dramatically. This forces the conclusion that that least for the daily adjusted case there is not enough data to allow a conclusion about the desirability of using a

synthetic portfolio.

The final conclusion of this sub-section is that for most of the option types studied there is no advantage or disadvantage in using a synthetic option portfolio. In the two comparisons using generated data and the short volatility measure there is not enough data available to decide conclusively about the desirability of using a synthetic portfolio.

5.3 Extensions

The experiments carried out in this thesis so far have all used a particular type of option as described in the section on the experimental design. To access the applicability of these results to other types of options the experiments described previously were repeated using

a. put options,

b. call options with a duration of 61 days, and,

c. put options with a duration of 61 days.

The results for these extra sets of experiments are the same as those for the experiments already reported. None of the comparisons between any pair of policies gave significant mean profit differences and in many cases there is sufficient data available to conclude that there is no difference between policies. As the results do not give any additional information and due to time and space restrictions the results of the extra experiments are not presented here.

6. Conclusion

The first aim of this thesis was to examine the profits gained from granting options and managing synthetic portfolios under less restrictive assumptions than those used to develop the option pricing and synthetic portfolio theories. This aim was achieved by simulating the behaviour of various different option granting policies using generated data that obeys the assumptions of the model and actual data for the period 15/2/84 to 1/9/86. The results from simulating the different policies on the two types of data were then compared. In all cases the difference in profits were not significantly different from zero. And for most of the policies there was sufficient data available to enable enough replications to be performed to conclude definitely that there is no significant performance difference on the two types of data. This result allows the grantor of options to use the theoretical option pricing and synthetic portfolio management theories without having to worry about the violation of the assumptions of the models that occur in the real world.

The second aim was to determine if the different portfolio adjustment frequencies and currency volatility measures affected the returns gained from granting options, both in the real world and when generated data is used. The mean profit differences between the different adjustment frequencies and volatility measures were all insignificant and there were sufficient replications in all cases to conclude that the different policies do not affect the returns from granting options.

The final object of the thesis was to determine if the use of a synthetic portfolio affects the returns from granting options. The results for this section indicated that for

most of the policies studied the use of a synthetic portfolio does not affect the returns from granting options. When generated data and the short volatility measure was used it was not possible to conclude definitely about the effect of a synthetic portfolio on profits.

This thesis has shown that the option pricing and synthetic portfolio theories are still correct when used in the real world on actual currency prices and interest rate movements. In other words, the the two theories are robust to the relaxation of some of their underlying assumptions. This robustness means that an option trader can use the option pricing and synthetic portfolio theories in the foreign exchange markets with some confidence.

7. Bibliography

Abramovitz, M. and I. A. Stegun, 1964, *Handbook of Mathematical Functions*, National Bureau of Standards, Washington D.C.

Adams, Arthur A., 1970, 'Remark on Algorithm 304' *Communications of the Association of Computing Machinery*.

Black, Fisher, 1976, 'The Pricing of Commodity Contracts' *Journal of Financial Economics*, 3 167-179.

Black, Fisher and Myron Scholes, 1973, 'The Pricing of Options and Corporate Liabilities' *Journal of Political Economy*, 81 637-654.

Bratley, Paul, Bennett L. Fox and Linus E. Schrage, 1983, *A Guide to Simulation*, (Springer-Verlag).

Cox, John C., Stephen A. Ross and Mark Rubinstein, 1979, 'Option Pricing: A Simplified Approach' *Journal of Financial Economics*, 7 229-263.

Cox, John C. and Mark Rubinstein, 1985, *Options Markets*, (Prentice-Hall, New Jersey).

Das, Satyajit, 1986, 'The Granting of Options: The Risk Management Process' *Option Strategies in Active Financial Management*, Conference April 18.

Fishman, George S., 1967, 'Problems in the Statistical Analysis of Simulation Experiments: The Comparison of Means and the Length of Sample Records' *Communications of the Association of Computing Machinery*, 10 2 94-99.

Fishman, George S., 1968, 'The Allocation of Computer Time in Comparing Simulation Experiments' *Operations Research*, 16 280-295.

Garman, Mark B. and Steven W. Kohlhagen, 1983, 'Foreign Currency Option Values' *Journal of International Money and Finance*, 2 231-237.

Geske, Robert and Kuldeep Shastri, 1985, 'Valuation by Approximation: A Comparison of Alternative Option Valuation Techniques' *Journal of Financial and Quantitative Analysis*, 20 1 45-71.

Hill, I. D. and S. A. Joyce, 1967, 'Normal Curve Integral' *Communications of the Association of Computing Machinery*, 10 (June), 374.

Hoag, James W., and Ralph McKay, 1984, 'Foreign Exchange Risk Exposure: Managing Through Synthetic Options' *The Securities Institute Journal*, 4 (December) 17-26.

Holmgrem, Bo, 1971, 'Remark on Algorithm 304' *Communications of the Association of Computing Machinery*.

Kleijnen, Jack P. C., 1975, *Statistical Techniques in Simulation*, part 2, (Marcel Dekker).

Knuth, Donald E., 1981, *The Art of Computer Programming*, Volume 2, *Seminumerical Algorithms*, (Addison-Wesley).

Rubinstein, Mark and Hayne E. Leland, 1981, 'Replicating Options with Positions in Stock and Cash' *Financial Analysts Journal*, July-August 63-72.

Wagner, Harvey M., 1969, *Principles of Operations Research, With Applications to Managerial Decisions*, (Prentice-Hall).

Whaley, Robert E., 1986, 'Valuation of American Futures Options: Theory and Empirical Tests' *The Journal of Finance*, 41 127-150.

8. Appendix A — Data

The following 3 pages are a plot of the data used in this thesis. These plots are not meant to be be highly accurate but are just for illustrative purposes. A listing of the data follows the plots.

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
12/12/83	1.0941	0	0
13/12/83	1.1062	0	0
14/12/83	1.1105	0	0
15/12/83	1.1167	0	0
16/12/83	1.1186	0	0
19/12/83	1.1416	0	0
20/12/83	1.1331	0	0
21/12/83	1.1198	0	0
22/12/83	1.1074	0	0
28/12/83	1.1161	0	0
29/12/83	1.1161	0	0
03/01/84	1.1068	0	0
04/01/84	1.1109	0	0
05/01/84	1.1123	0	0
06/01/84	1.1074	0	0
09/01/84	1.1001	0	0
10/01/84	1.1025	0	0
11/01/84	1.1007	0	0
12/01/84	1.1044	0	0
13/01/84	1.1044	0	0
16/01/84	1.1056	0	0
17/01/84	1.1067	0	0
18/01/84	1.1091	0	0
19/01/84	1.1117	0	0
20/01/84	1.1068	0	0
23/01/84	1.1075	0	0
24/01/84	1.1035	0	0
25/01/84	1.1011	0	0
26/01/84	1.1007	0	0
31/01/84	1.0893	0	0
01/02/84	1.0875	0	0
02/02/84	1.0848	0	0
03/02/84	1.0828	0	0
06/02/84	1.0841	0	0
07/02/84	1.0853	0	0
08/02/84	1.0817	0	0
09/02/84	1.0755	0	0
10/02/84	1.0748	0	0
13/02/84	1.0735	0	0
14/02/84	1.0727	0	0
15/02/84	1.0661	9.75	9.69
16/02/84	1.0569	8	9.56

17/02/84	1.0588	8	9.56
20/02/84	1.0588	10	9.69
21/02/84	1.0650	10.5	9.63
22/02/84	1.0650	11.25	9.5

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
23/02/84	1.0588	11.5	9.75
24/02/84	1.0610	11.5	9.75
27/02/84	1.0607	11	9.88
28/02/84	1.0601	11	9.88
29/02/84	1.0609	12	9.75
01/03/84	1.0600	11.5	9.81
02/03/84	1.0599	12	9.81
05/03/84	1.0499	11	9.88
06/03/84	1.0466	11.5	9.88
07/03/84	1.0449	12	9.88
08/03/84	1.0495	12.75	9.88
09/03/84	1.0488	13	9.88
12/03/84	1.0474	12.5	9.88
13/03/84	1.0411	12.75	10
14/03/84	1.0378	13	10.06
15/03/84	1.0368	13	10.13
16/03/84	1.0368	12.75	10.13
19/03/84	1.0499	12.25	10.13
20/03/84	1.0433	12.25	10.13
21/03/84	1.0482	13.5	10.38
22/03/84	1.0616	12.75	10.63
23/03/84	1.0661	14	11
26/03/84	1.0529	13	11
27/03/84	1.0582	13.5	9.88
28/03/84	1.0684	14.75	9.88
29/03/84	1.0638	14	10.38
30/03/84	1.0689	14.25	10.5
02/04/84	1.0616	14	10.38
03/04/84	1.0661	15	10.5
04/04/84	1.0638	15.75	10.88
05/04/84	1.0707	15.5	10.94
06/04/84	1.0858	15	10.75
09/04/84	1.0953	15.75	10.5
10/04/84	1.0921	15.75	10.5
11/04/84	1.0858	15.5	10.38
12/04/84	1.0864	15.75	10.56
13/04/84	1.0897	15.75	10.56

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
16/04/84	1.0887	14.5	10.69
17/04/84	1.0854	15	10.75
18/04/84	1.0828	15.5	10.75
19/04/84	1.0834	14.5	10.75
24/04/84	1.0864	13.5	10.63
26/04/84	1.0864	14.25	10.63
27/04/84	1.0858	14.5	10.63
30/04/84	1.0874	14.5	10.63
01/05/84	1.0847	15	10.75
02/05/84	1.0799	15	10.88
03/05/84	1.0746	16	10.75
04/05/84	1.0778	15.5	10.56
07/05/84	1.0899	15.5	10.56
08/05/84	1.0935	15.25	11
09/05/84	1.0991	15.25	11
10/05/84	1.1040	15.25	11
11/05/84	1.1062	14.25	11
14/05/84	1.1151	14.5	10.94
15/05/84	1.1123	14	10.88
16/05/84	1.1077	13.5	10.75
17/05/84	1.1117	14	10.63
18/05/84	1.1206	14.4	9.56
21/05/84	1.1201	14	10.25
22/05/84	1.1084	14	10.5
23/05/84	1.1077	14	10.75
24/05/84	1.1086	14	10.75
25/05/84	1.1105	14.25	10.5
28/05/84	1.1105	14.25	10.5
29/05/84	1.1136	14.25	10.75
30/05/84	1.1137	14.5	10.75
31/05/84	1.1121	13.5	10.81
01/06/84	1.1094	13.5	10.75
04/06/84	1.1051	13	10.81
05/06/84	1.1111	12.5	10.75
06/06/84	1.1167	13	10.81
07/06/84	1.1138	12	10.88
08/06/84	1.1145	12.5	10.81
12/06/84	1.1159	13	11.06
13/06/84	1.1173	13	11
14/06/84	1.1211	12.5	11
15/06/84	1.1261	11.5	11.31

18/06/84	1.1333	10.75	11.38
19/06/84	1.1319	11.5	11.56
20/06/84	1.1308	12	11.69
21/06/84	1.1455	11.75	12.31
22/06/84	1.1537	13	12.19
25/06/84	1.1730	14.25	12.25
26/06/84	1.1663	14.75	12
27/06/84	1.1541	16	12.88
28/06/84	1.1601	14.5	11.56
29/06/84	1.1601	13.75	11.5
02/07/84	1.1674	13	11.5
03/07/84	1.1666	11.5	11.63
04/07/84	1.1669	12.5	11.63
05/07/84	1.1792	12.5	11.44

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
06/07/84	1.1869	10	11.56
09/07/84	1.1983	10.5	11.63
10/07/84	1.2026	11.75	11.44
11/07/84	1.2188	11.5	11.44
12/07/84	1.2095	12.5	11.31
13/07/84	1.1905	13	11.5
16/07/84	1.1905	12	11.44
17/07/84	1.2024	12	11.5
18/07/84	1.2085	12	11.69
19/07/84	1.2069	12.5	11.38
20/07/84	1.2048	12.5	11.38
23/07/84	1.2158	12.5	11.44
24/07/84	1.2121	12.5	11.38
25/07/84	1.2048	13	11.38
27/07/84	1.1943	13.5	11.31
30/07/84	1.2077	13	11.44
31/07/84	1.2048	12.75	11.56
01/08/84	1.2037	13.25	11.56
02/08/84	1.1983	12.75	11.5
03/08/84	1.1919	12.25	11.5
06/08/84	1.1827	12.25	11.56
07/08/84	1.1915	11.5	11.75
08/08/84	1.1876	12	11.56
09/08/84	1.1858	12	11.63
10/08/84	1.1848	12	11.63
13/08/84	1.1898	12	11.81
14/08/84	1.1901	12	11.69

15/08/84	1.1848	12.25	11.69
16/08/84	1.1848	11.75	11.69
17/08/84	1.1802	12.25	11.75
20/08/84	1.1688	12.3	11.81
21/08/84	1.1710	12.5	11.88
22/08/84	1.1696	12.5	11.69
23/08/84	1.1682	12.5	11.75
24/08/84	1.1680	12	11.69
27/08/84	1.1744	11.25	11.69
28/08/84	1.1765	11	11.75
29/08/84	1.1730	12	11.75
30/08/84	1.1788	12	11.75
31/08/84	1.1788	12	11.69
03/09/84	1.1869	12	11.81
04/09/84	1.1891	11	11.81
05/09/84	1.2034	10	11.69
06/09/84	1.2012	10.5	11.88
07/09/84	1.2041	10	11.75
10/09/84	1.2063	10.5	11.69

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
11/09/84	1.1998	10.5	11.56
12/09/84	1.1955	11.75	11.56
13/09/84	1.2026	11	11.56
14/09/84	1.2026	10.75	11.5
17/09/84	1.2089	10.25	11.69
18/09/84	1.2092	10	11.63
19/09/84	1.2063	10	11.5
20/09/84	1.2026	10	11.38
21/09/84	1.2092	10.5	11.25
24/09/84	1.2015	11.25	11.19
25/09/84	1.2055	11.25	11.19
26/09/84	1.2077	11.75	11.13
27/09/84	1.1998	11.75	11.06
28/09/84	1.1999	12	11.06
02/10/84	1.2034	11.25	11.31
03/10/84	1.1998	11.5	11.13
04/10/84	1.2012	11.5	10.94
05/10/84	1.1983	11	10.81
08/10/84	1.1996	11.25	10.75
10/10/84	1.2032	11.5	10.63
11/10/84	1.2041	11.25	10.56
12/10/84	1.2034	11.5	10.44

15/10/84	1.2054	10.5	10.44
16/10/84	1.2051	10.5	10.5
17/10/84	1.2060	10.5	10.44
18/10/84	1.2034	10.75	10.25
19/10/84	1.1990	11	10
22/10/84	1.1975	11	10.06
23/10/84	1.1976	11.5	9.94
24/10/84	1.1843	11.5	9.56
25/10/84	1.1806	11.75	9.38
26/10/84	1.1781	11.75	9.63
29/10/84	1.1884	11.35	9.88
30/10/84	1.1848	11.3	9.94
31/10/84	1.1772	11.5	9.75
01/11/84	1.1719	11.5	9.81
02/11/84	1.1655	11	9.81
05/11/84	1.1628	10.5	9.75
06/11/84	1.1601	10.5	9.63
07/11/84	1.1594	10.25	9.25
08/11/84	1.1561	10.5	9.56
09/11/84	1.1601	10.9	9.63
12/11/84	1.1605	11.25	9.44
13/11/84	1.1586	11.25	9.44
14/11/84	1.1614	11.75	9.44
15/11/84	1.1658	11.75	9.56

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
16/11/84	1.1637	12	9.56
19/11/84	1.1632	12	9.44
20/11/84	1.1648	12.5	9.38
21/11/84	1.1688	12.25	9.25
22/11/84	1.1655	11.85	9
23/11/84	1.1648	12	8.88
26/11/84	1.1700	12	8.94
27/11/84	1.1692	11.8	8.94
28/11/84	1.1660	11.75	8.88
29/11/84	1.1617	11.75	8.63
30/11/84	1.1641	11.65	8.75
03/12/84	1.1707	11.35	9.06
04/12/84	1.1721	11.25	9.06
05/12/84	1.1696	11.8	9.81
06/12/84	1.1765	12.25	8.88
07/12/84	1.1751	12.5	8.88
10/12/84	1.1847	12.5	8.88
11/12/84	1.1905	11.75	8.88
12/12/84	1.1795	12.25	8.81
13/12/84	1.1837	12.75	8.69
14/12/84	1.1919	12	8.69
17/12/84	1.1947	11.5	8.56
18/12/84	1.1969	11.75	8.06
19/12/84	1.1965	12	7.81
20/12/84	1.2012	12	8.25
21/12/84	1.2005	12.5	8.63
24/12/84	1.1976	12.5	8.69
27/12/84	1.2034	12.2	9.06
28/12/84	1.2085	11.75	8.5
02/01/85	1.2225	12.5	8.44
03/01/85	1.2308	12	8.5
04/01/85	1.2232	11.75	8.44
07/01/85	1.2361	12	8.31
08/01/85	1.2297	12	8.25
09/01/85	1.2346	12.25	8.19
10/01/85	1.2225	12.25	8.13
11/01/85	1.2151	12	8.19
14/01/85	1.2188	11.75	8.31
15/01/85	1.2258	11.25	8.19
16/01/85	1.2213	11.25	8.19
17/01/85	1.2206	11	8.25
18/01/85	1.2255	11.25	8.19

21/01/85	1.2252	11.5	8.19
22/01/85	1.2323	11.5	8.19
23/01/85	1.2293	12	8.19
24/01/85	1.2288	12.25	8.25

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
25/01/85	1.2276	12	8.25
29/01/85	1.2240	12.25	8.44
30/01/85	1.2258	12	8.38
31/01/85	1.2270	11.85	8.44
01/02/85	1.2320	12	8.75
04/02/85	1.2594	11.75	8.75
05/02/85	1.2762	10.5	8.63
06/02/85	1.2796	9.5	8.44
07/02/85	1.3004	10.25	8.44
08/02/85	1.2887	10.5	8.75
11/02/85	1.3098	11.75	8.56
12/02/85	1.3333	12.25	8.56
13/02/85	1.3396	12.5	8.63
14/02/85	1.3495	12.5	8.63
15/02/85	1.3477	12	8.56
18/02/85	1.3432	11.8	8.63
19/02/85	1.4184	12	8.38
20/02/85	1.4815	11.75	8.69
21/02/85	1.4085	13.25	8.63
22/02/85	1.3996	14	8.75
25/02/85	1.4225	13.5	8.63
26/02/85	1.4347	13.8	8.56
27/02/85	1.4347	14	8.56
28/02/85	1.4002	14	8.75
01/03/85	1.4114	13.75	8.75
04/03/85	1.4100	13.5	8.75
05/03/85	1.4391	13.5	8.75
06/03/85	1.4667	13.75	8.56
07/03/85	1.4514	14	8.63
08/03/85	1.4552	14.25	8.69
11/03/85	1.4347	14.5	8.5
12/03/85	1.4267	14.75	8.56
13/03/85	1.4333	15.25	8.69
14/03/85	1.4461	15	8.81
15/03/85	1.4596	14.6	8.88
18/03/85	1.4535	14.5	8.88
19/03/85	1.4430	14.5	8.75

20/03/85	1.4255	14.5	8.75
21/03/85	1.4482	14.5	8.44
22/03/85	1.4255	13.75	8.88
25/03/85	1.4382	14	8.94
26/03/85	1.4353	14.25	8.94
27/03/85	1.4211	14.25	8.75
28/03/85	1.4164	14.5	8.75
29/03/85	1.4198	14.5	8.69
01/04/85	1.4430	14.5	8.75

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
02/04/85	1.4859	15.25	8.88
03/04/85	1.5088	15	8.69
04/04/85	1.5244	15	8.81
09/04/85	1.4993	15.75	8.81
10/04/85	1.5026	16.25	8.75
11/04/85	1.4986	16.25	8.81
12/04/85	1.4870	15.75	8.69
15/04/85	1.5060	15.5	8.69
16/04/85	1.4948	16	8.56
17/04/85	1.5140	15.75	8.44
18/04/85	1.5798	15.75	8.44
19/04/85	1.5625	15.25	8.25
22/04/85	1.5848	15.5	8.13
23/04/85	1.5576	15.6	8.19
24/04/85	1.5437	15.5	8.25
26/04/85	1.5221	15.5	8.31
29/04/85	1.5267	15.75	8.31
30/04/85	1.5302	15.75	8.44
01/05/85	1.5456	16.25	8.44
02/05/85	1.5373	16	8.5
03/05/85	1.5198	16	8.44
06/05/85	1.5049	16.25	8.44
07/05/85	1.5049	16	8.25
08/05/85	1.4556	16	8.19
09/05/85	1.4535	16	8.19
10/05/85	1.4535	16	8.19
13/05/85	1.4378	16	8.19
14/05/85	1.4225	16.25	8.13
15/05/85	1.4524	16	8.06
16/05/85	1.4684	16.1	8.13
17/05/85	1.4760	16	8.13
20/05/85	1.4440	16	7.88

21/05/85	1.4430	16	7.81
22/05/85	1.4451	16.2	7.75
23/05/85	1.4663	16.1	7.81
24/05/85	1.4695	16	7.75
27/05/85	1.5049	16	7.75
28/05/85	1.5198	16	7.75
29/05/85	1.5072	15.5	7.75
30/05/85	1.5038	15.5	7.75
31/05/85	1.5209	15.25	7.56
03/06/85	1.5094	15.25	7.56
04/06/85	1.5106	15.5	7.69
05/06/85	1.5205	15.25	7.63
06/06/85	1.5129	15.75	7.63
07/06/85	1.5060	16	7.63

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
11/06/85	1.5072	16.5	7.81
12/06/85	1.4937	17	7.63
13/06/85	1.5060	19	7.69
14/06/85	1.5060	20	7.69
17/06/85	1.5083	17	7.5
18/06/85	1.5083	17	7.31
19/06/85	1.4903	16.75	7.13
20/06/85	1.4970	16.25	7.31
21/06/85	1.4981	16	7.5
24/06/85	1.5088	20	7.69
25/06/85	1.5015	20	7.81
26/06/85	1.4981	25	7.81
27/06/85	1.4970	23	7.75
28/06/85	1.5015	21	7.81
01/07/85	1.4981	18	7.81
02/07/85	1.5038	15.5	7.88
03/07/85	1.4993	14.75	7.81
04/07/85	1.4925	14	7.88
05/07/85	1.4892	14	7.88
08/07/85	1.4749	14.5	7.88
09/07/85	1.4706	15	7.88
10/07/85	1.4609	15.5	7.88
11/07/85	1.4347	16	7.81
12/07/85	1.4306	15.75	7.81
15/07/85	1.4205	15.5	7.81
16/07/85	1.4134	16	7.81
17/07/85	1.4168	16	7.75

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
18/07/85	1.3957	16.5	7.81
19/07/85	1.4045	16	7.94
22/07/85	1.4065	16.25	8.06
23/07/85	1.3957	16.5	8.06
24/07/85	1.4021	16.5	8.88
25/07/85	1.4164	15.75	7.88
26/07/85	1.4144	15.75	7.81
29/07/85	1.3957	15.5	7.88
30/07/85	1.3908	15.5	7.88
31/07/85	1.3746	15.5	7.94
01/08/85	1.3822	15.5	7.94
02/08/85	1.4035	16	8
06/08/85	1.4031	15.75	7.94
07/08/85	1.4296	16	7.94
08/08/85	1.4174	16	7.88
09/08/85	1.4174	15.75	7.88
12/08/85	1.3966	16.5	7.88
13/08/85	1.4025	16.25	7.94
14/08/85	1.4124	16.3	8.06

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
15/08/85	1.4184	16.25	8
16/08/85	1.4286	16.5	8.06
19/08/85	1.4184	16.75	8.06
20/08/85	1.4194	17.5	7.94
21/08/85	1.4174	17.5	7.88
22/08/85	1.4154	16.5	7.88
23/08/85	1.4184	16.8	7.88
26/08/85	1.4239	16.5	7.88
27/08/85	1.4255	17.25	8.06
28/08/85	1.4255	18	8.13
29/08/85	1.4209	17.5	8
30/08/85	1.4221	17	8
02/09/85	1.4351	16.75	8.06
03/09/85	1.4535	17	8.13
04/09/85	1.4545	17.5	8.13
05/09/85	1.4461	17.5	8
06/09/85	1.4556	17	8
09/09/85	1.4826	17.5	8.06
10/09/85	1.4848	17.5	8.13
11/09/85	1.4832	17.75	8.19
12/09/85	1.4870	17	8.13
13/09/85	1.4848	16.5	8.06

16/09/85	1.4738	16.75	8.06
17/09/85	1.4663	16.9	8.13
18/09/85	1.4674	17	8.13
19/09/85	1.4756	16.75	8
20/09/85	1.4674	16.75	8.13
23/09/85	1.4245	16.5	8
24/09/85	1.4045	16.5	8
25/09/85	1.4134	16.5	8
26/09/85	1.3966	16.75	7.94
27/09/85	1.4006	16.5	7.94
30/09/85	1.4154	16	8
01/10/85	1.4209	16.75	8.06
02/10/85	1.4049	16	8
03/10/85	1.4124	16	7.94
04/10/85	1.3966	16.25	7.94
07/10/85	1.4225	16.25	8.06
08/10/85	1.4201	16.25	8.06
09/10/85	1.4306	16.25	8
10/10/85	1.4231	16.5	8.13
11/10/85	1.4225	16.5	8.06
14/10/85	1.4235	16.5	8.06
15/10/85	1.4259	16.4	8.13
16/10/85	1.4296	16.25	8.13
17/10/85	1.4306	16.25	8.06

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
18/10/85	1.4221	16.25	8.06
21/10/85	1.4198	16.35	8.13
22/10/85	1.4245	16.35	8.13
23/10/85	1.4255	16.35	8.19
24/10/85	1.4296	16.25	8.19
25/10/85	1.4302	16.35	8.13
28/10/85	1.4286	16.25	8.06
29/10/85	1.4276	16.45	8.06
30/10/85	1.4249	16.25	7.88
31/10/85	1.4286	16.3	8
01/11/85	1.4345	16.35	8
04/11/85	1.4556	16.35	8.13
05/11/85	1.4910	16.5	8
06/11/85	1.5117	16.35	8.19
07/11/85	1.4937	16.35	8.19
08/11/85	1.4892	16.25	8.13
11/11/85	1.4948	17.15	8.06
12/11/85	1.5232	17.75	8.06
13/11/85	1.5256	18.25	8.13
15/11/85	1.4870	18.5	8.38
18/11/85	1.4881	18.25	8.25
19/11/85	1.4826	19	8.13
20/11/85	1.4728	19	8
21/11/85	1.4669	19.4	8.19
22/11/85	1.4584	19	8.06
25/11/85	1.4409	19.1	8.13
26/11/85	1.4493	19.25	8.13
27/11/85	1.4503	19.25	8.19
28/11/85	1.4594	19.25	8.19
29/11/85	1.4599	19.25	8.19
02/12/85	1.4648	19	8.25
03/12/85	1.4760	19.25	8.38
04/12/85	1.4717	19.25	8.31
05/12/85	1.4663	19.25	8.25
06/12/85	1.4706	19.25	8.25
09/12/85	1.4771	19.25	8.25
10/12/85	1.4689	19.75	8.07
11/12/85	1.4620	19.25	8.19
12/12/85	1.4641	19.5	8.06
13/12/85	1.4609	19.75	8.06
16/12/85	1.4674	19.75	8.06
17/12/85	1.4745	19.75	8.06

18/12/85	1.4717	20	8
19/12/85	1.4684	19.75	8.5
20/12/85	1.4674	19.75	8.81
23/12/85	1.4684	19.75	8.38

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
24/12/85	1.4684	19.5	8.63
27/12/85	1.4706	19.5	9.19
30/12/85	1.4661	19.25	8.31
31/12/85	1.4684	19.25	8.31
02/01/86	1.4652	19	8.25
03/01/86	1.4635	18.75	8.31
06/01/86	1.4684	18.9	8.19
07/01/86	1.4663	19.25	8.13
08/01/86	1.4535	19	8
09/01/86	1.4434	19	8.13
10/01/86	1.4420	19.25	8.13
13/01/86	1.4426	19.25	8.19
14/01/86	1.4545	19.4	8.25
15/01/86	1.4440	19.3	8.25
16/01/86	1.4337	19.3	8.06
17/01/86	1.4327	19.25	8.13
20/01/86	1.4358	19.25	8.19
21/01/86	1.4292	19.25	8.06
22/01/86	1.4164	19.25	8.13
23/01/86	1.4059	19.25	8.19
24/01/86	1.4098	19	8.13
28/01/86	1.4021	19	8.06
29/01/86	1.4051	18.75	8
30/01/86	1.4019	18.75	8
31/01/86	1.3992	18.7	8.06
03/02/86	1.4128	18.6	8.06
04/02/86	1.4461	18.5	7.94
05/02/86	1.4468	18.75	7.88
06/02/86	1.4327	18.75	7.88
07/02/86	1.4420	18.9	7.88
10/02/86	1.4428	18.75	7.94
11/02/86	1.4503	19.1	8
12/02/86	1.4545	19.15	8.06
13/02/86	1.4545	19.25	7.94
14/02/86	1.4225	19.15	8
17/02/86	1.4327	19	7.94
18/02/86	1.4368	18.9	8

19/02/86	1.4124	18.9	8
20/02/86	1.4124	19	7.94
21/02/86	1.4245	19	7.94
24/02/86	1.4205	18.5	7.94
25/02/86	1.4144	18.55	8
26/02/86	1.4114	18.5	8
27/02/86	1.4134	17.75	7.94
28/02/86	1.4306	17.35	7.94
03/03/86	1.4388	17.25	7.94

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
04/03/86	1.4316	16.5	7.94
05/03/86	1.4225	17	7.81
06/03/86	1.4286	17.3	7.81
07/03/86	1.4265	17	7.56
10/03/86	1.4215	17	7.5
11/03/86	1.4194	17	7.56
12/03/86	1.4194	17.35	7.56
13/03/86	1.4198	16.75	7.56
14/03/86	1.4217	17	7.56
17/03/86	1.4184	17	7.88
18/03/86	1.4006	17.45	7.69
19/03/86	1.4065	17.25	8.06
20/03/86	1.4045	17	7.94
21/03/86	1.4035	16.5	7.81
24/03/86	1.3924	16.25	7.94
25/03/86	1.3841	16.25	7.94
26/03/86	1.3947	16.25	7.56
27/03/86	1.4045	16.75	7.5
01/04/86	1.3947	16.5	7.5
02/04/86	1.3889	16.65	7.5
03/04/86	1.3947	16.75	7.5
04/04/86	1.3899	16.5	7.5
07/04/86	1.3918	17	7.31
08/04/86	1.3899	17	7.25
09/04/86	1.3908	16.75	6.94
10/04/86	1.4006	16.75	7.06
11/04/86	1.3986	16.75	7.06
14/04/86	1.3966	17	7.06
15/04/86	1.4023	17.25	7.06
16/04/86	1.3976	16.5	7.06
17/04/86	1.3947	15.25	6.75
18/04/86	1.3986	15.85	6.69

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
21/04/86	1.4035	15.75	6.88
22/04/86	1.3721	16.5	7
23/04/86	1.3686	16.5	7.06
24/04/86	1.3721	16.75	7.06
28/04/86	1.3550	17.25	7
29/04/86	1.3541	16.25	7
30/04/86	1.3550	15.5	7
01/05/86	1.3572	15.8	7.06
02/05/86	1.3633	15.25	7.06
05/05/86	1.3559	15.5	7.06
06/05/86	1.3514	16.75	6.94
07/05/86	1.3585	16.5	6.94
08/05/86	1.3495	16	6.94
09/05/86	1.3477	15.75	6.94
12/05/86	1.3378	15.5	7
13/05/86	1.3541	15.5	7
14/05/86	1.4265	15.25	7
15/05/86	1.3966	14.5	7
16/05/86	1.3928	14	7.06
19/05/86	1.3784	15	7.06
20/05/86	1.3784	14.75	7.06
21/05/86	1.3864	14.75	7.06
22/05/86	1.3996	14.75	7.06
23/05/86	1.3918	15.25	6.94
26/05/86	1.3908	15	6.94
27/05/86	1.3870	15	7
28/05/86	1.3850	14.5	7.06
29/05/86	1.3908	14.5	7
30/05/86	1.3947	14.5	7.06
02/06/86	1.4110	14.75	7.06
03/06/86	1.4205	14.5	7.13
04/06/86	1.4296	14.5	7.13
05/06/86	1.4286	14	7.06
06/06/86	1.4451	14.75	7.07
10/06/86	1.4472	14.75	7.06
11/06/86	1.4399	15.25	7.06
12/06/86	1.4550	15.75	7.06
13/06/86	1.4440	15.75	7.06
16/06/86	1.4368	15.75	7.06
17/06/86	1.4347	15.25	7.06
18/06/86	1.4409	14.75	7.06

19/06/86	1.4413	14.75	7.06
20/06/86	1.4430	14.75	7
23/06/86	1.4674	14.75	7.06
24/06/86	1.4674	15	7.13
25/06/86	1.5069	15	7.13
26/06/86	1.4885	15	7.13
27/06/86	1.4896	16	7.06
30/06/86	1.4771	15.5	7.06
01/07/86	1.5106	15	7.13
02/07/86	1.5267	14.75	7.13
03/07/86	1.5686	14.25	7.06
04/07/86	1.5480	14	7.06
07/07/86	1.5564	14	7.06
08/07/86	1.5949	14.3	7
09/07/86	1.5873	14.75	9.94
10/07/86	1.5625	15.75	6.88
11/07/86	1.5699	15.75	6.63
14/07/86	1.5552	15	6.69
15/07/86	1.5569	14.75	6.69

Date	$A/$US Exchange Rate	Domestic Interest Rate	Foreign Interest Rate
16/07/86	1.5637	14.5	6.56
17/07/86	1.5738	14.5	6.56
18/07/86	1.5649	14.5	6.56
21/07/86	1.5581	14.25	6.56
22/07/86	1.5674	14	6.56
23/07/86	1.5949	14	6.56
24/07/86	1.6327	14.25	6.56
25/07/86	1.6393	14.25	6.56
28/07/86	1.6194	14.5	6.56
29/07/86	1.6420	14.5	6.56
30/07/86	1.6359	14.85	6.5
31/07/86	1.6756	14.85	6.5
01/08/86	1.6617	15.5	6.44
04/08/86	1.6611	15.5	6.5
05/08/86	1.6287	17	6.5
06/08/86	1.6116	17	6.5
07/08/86	1.6090	17	6.5
08/08/86	1.6420	17.5	6.5
11/08/86	1.6529	17.5	6.5
12/08/86	1.6469	17.5	6.44
13/08/86	1.6250	17.25	6.38
14/08/86	1.6281	17.25	6.25
15/08/86	1.5964	17.25	6.31
18/08/86	1.5990	17.5	6.5
19/08/86	1.5876	17.5	6.44
20/08/86	1.6388	17.75	6.38
21/08/86	1.6276	18	6
22/08/86	1.6420	17.75	6.06
25/08/86	1.6523	18	6.06
26/08/86	1.6466	18.5	6
27/08/86	1.6447	18	6
28/08/86	1.6407	18	5.94
29/08/86	1.6428	18	6
01/09/86	1.6420	18	6

9. Appendix B — Sim Program Source

Appendix B contains the source code for the main simulation program *sim*. This program was written in C on a PDP-11/34 under the UNIX level 7 operating system. If it is desired to transport *sim* to another computer running UNIX the main requirement is that **long**s should have at least 32 bits so that the random number generator *unif()* will work correctly. For non-UNIX systems another problem could be the random access that is performed on the temporary file by the routines in *Sim/swap.c*, if this is a problem they can be removed all together if the target machine has sufficient memory to store all the temporary arrays in memory. The only other specific requirement is that the standard mathematical library must be loaded with these files.

```
/*
 *      File:    assert.h
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 *
 *      @(#)assert.h    1.2     86/10/31
 */

#ifndef NDEBUG
#define _assert(ex)     ((ex) || (fprintf(stderr,\
"Assertion failed: file %s, line %d\n", __FILE__, __LINE__), exit(1)))

#define assert(ex)      ((ex) || (fprintf(stderr,\
"Assertion failed: file %s, line %d\n", __FILE__, __LINE__), exit(1)))

#else
#define _assert(ex)     ((ex), 1)
#define assert(ex)      ((ex), 1)
#endif
```

```
/*
 *      File:    defs.h
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 *
 *      @(#)defs.h      1.2       86/10/23
 */

typedef enum
{
        false    = 0,
        true     = 1
}
        bool;

extern double  Gamma();
extern double  normal();
```

```
/*
 *      File:    sim.h
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 *
 *      @(#)sim.h       1.5       86/10/30
 */

/*
 *      The type used to store temporary data in.  May be reduced to float
 *      to save space.
 */
typedef doublereal;

/*
 *      the type of a seed for unif (a pointer to one of these must
 *      passed to it, initialized to some value).
 */
typedef long    seed_t;

/*
 *      a data point, contains the day number, exchange rate,
 *      domestic interest rate and foreign interest rate.
 */
typedef struct
{
        int     d_daynum;
        float   d_exrate;
        float   d_austr;
        float   d_usr;
}
        d_data;

/*
 *      A structure to hold all the data in, just to keep to
 *      pointer to the items and the number of items together.
 */
typedef struct
{
        int     dt_count;
        d_data *dt_info;
}
        dt_data;
```

```
/*
 *      A type used to hold the results of analysis in.
 */
typedef struct
{
        double an_mean;       /* mean of results */
        double an_sdev;       /* s. dev. of mean */
}
        an_anal;

/*
 *      A type used to form the array of different policies.
 */
typedef struct
{
        double (*py_syn)();   /* synthesis procedure */
        double (*py_vol)();   /* volatility function */
        double (*py_gen)();   /* data generation function */
        char    *py_name;     /* name of this policy */
}
        py_policy;

void    couldnot();
void    fcouldnot();
char    *salloc();
char    *numday();

#define         DAYS  (365.0)
#define         PERCENT     100

/*
 *      Starting values for random number generation from
 *      Bratley, Fox and Schrage (1983) p203.
 */
#define         GEN_SEED    ((seed_t)524287L)
#define         START_SEED          ((seed_t)2050954260L)
```

```
/*
 *      File:   Gamma.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)Gamma.c      1.1     86/10/22";

/*
 *      A real gamma function, as compared to the maths library
 *      gamma function which returns the log of the absolute value
 *      of the gamma function.
 *
 *      This is adapted from the gamma(3m) manual entry.
 *
 *      Author:
 *              Stuart Pook
 *              September 1986.
 */

#include        <math.h>

extern double  gamma();
extern int      signgam;

double
Gamma(x)
double x;
{
        register doubley;

        y = gamma(x);
        if (y > 88.0)
                error("Gamma argument too big");
        return exp(y) * signgam;
}
```

```
/*
 *      File:   analyse.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)analyse.c      1.6     86/10/22";

#include       <stdio.h>
#include       <math.h>
#include       <sys/types.h>

#include       "defs.h"
#include       "sim.h"

/*
 *      Number of significant digits to print out for the t−statistic
 */
#define        TDIGITS                 3

extern double  fabs();

char    *salloc();
void    swapin();

static double   sdev();
static double   autocorr();
static void     compare();

void
analyse(reps, fd, policy, M)
int             reps;
int             fd;
py_policy       policy[];
int             M;
{
        register int    i;
        register real   *array1;
        register real   *array2;
        register int    n;
        an_anal                 anal;

        /*
         *      count the number of policys
```

```
 */
for (n = 0; policy[n].py_vol != (double (*)())0; n++)
        ;

array1 = (real *)salloc(reps * sizeof (real));
/*
 *      analyse each policy by itself
 */
for (i = 0; i < n; i++)
{
        swapin(i, reps, array1, fd);
        compare(reps, array1, &anal, M);
        printf
        (
                "%s\t%g (%g) (%.*g)\n",
                policy[i].py_name,
                anal.an_mean,
                anal.an_sdev,
                TDIGITS,
                fabs(anal.an_mean / anal.an_sdev)
        );
}

array2 = (real *)salloc(reps * sizeof (real));

/*
 *      analyse the difference between each policy
 */
for (i = 0; i < n; i++)
{
        register int      j;

        swapin(i, reps, array1, fd);
        for (j = i + 1; j < n; j++)
        {
                register int      k;

                swapin(j, reps, array2, fd);
                for (k = 0; k < reps; k++)
                        array2[k] = array1[k] − array2[k];

                compare(reps, array2, &anal, M);
                printf
                (
                        "%s − %s\t%g (%g) (%.*g)\n",
```

```
                                policy[i].py_name,
                                policy[j].py_name,
                                anal.an_mean,
                                anal.an_sdev,
                                TDIGITS,
                                fabs(anal.an_mean / anal.an_sdev)
                        );
                }
        }

        (void)free((char *)array1);
        (void)free((char *)array2);
}

static void
compare(n, output, anp, M)
int             n;
register real   output[];
an_anal                 *anp;
int             M;
{
        register int    i;
        register double x;

        if (n < 2)
                error("too few output points (n = %d)", n);

        /*
         *      Calculate the mean using the method given in Knuth v2 p216.
         */
        x = output[0];
        for (i = 1; i < n; i++)
                x = x + (output[i] − x) / (i + 1);

        anp−>an_mean = x;

        anp−>an_sdev = sdev(n, output, x, M);
}

/*
 *      Calculate the standard deviation (or variance) of the n output
 *      points given.
 *      M is the number of autocorrelation values to use.
 */
static double
```

```
sdev(n, output, mean, M)
int     n;
real    output[];
double mean;
int     M;
{
        register doublev;
        register int    i;
        extern bool     O_variance;

        v = autocorr(0, n, output, mean);

        for (i = 1; i <= M; i++)
        {
                v += 2.0 * (1.0 − (double)i / M) * autocorr(i, n, output, mean);
        }

        v /= 1.0 − (double)M / n;
        /*
         *      v is now the same as m hat from equation 25a in Fishman
         *      1967 in Operations Research 16 pp. 280−295.
         *      Now need to divide v by n to get an estimate of var(xbar).
         *      Ref. Fishman p 281.
         */
        v /= n;

        /*
         *      If `−v´ option produce an estimate of the variance
         *      else an estimate of the standard deviation.
         */
        if (O_variance == false)
                v = sqrt(v);

        return v;
}

static double
autocorr(k, n, x, xbar)
register int    k;      /* autocorrelation with lag k */
int             n;      /* number of observations */
real            x[];    /* observations */
double          xbar;   /* mean of the observations */
{
        register int    i;
        double          a;
```

```
        a = 0.0;
        for (i = 0; i < n − k; i++)
                a += (x[i] − xbar) * (x[i + k] − xbar);
        a /= n;

        debug("autocorr: k = %d n = %d xbar = %g a = %g", k, n, xbar, a);
        return a;
}
```

```
/*
 *      File:   daynum.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)daynum.c      1.3      86/10/22";

#include        <stdio.h>
#include        <sys/types.h>

#include        "assert.h"

#define         FIRSTYEAR  70
#define         YEARSZ              365
#define         LEAPSZ              (YEARSZ + 1)

static void     check();
static int      yearsz();
static int      monthsz();

/*
 *      Return the day number of date in id, 1 January FIRSTYEAR is day 1.
 */
int
daynum(id)
char    *id;
{
        register int    num;
        int             day;
        int             month;
        int             year;

        if (sscanf(id, "%2d/%2d/%2d", &day, &month, &year) != 3)
                error("daynum: bad format on date (%s)", id);

        check(day, month, year);

        num = day;
        for (month--; month > 0; month--)
                num += monthsz(month, year);

        for (year--; year >= FIRSTYEAR; year--)
                num += yearsz(year);
```

```
	return num;
}

char	*
numday(day)
register int	day;
{
	register int	year;
	register int	month;
	int		input;
	static char	buf[16];

	input = day;

	year = FIRSTYEAR;
	while (day > 0)
		day −= yearsz(year++);
	day += yearsz(−−year);

	month = 1;
	while (day > 0)
		day −= monthsz(month++, year);
	day += monthsz(−−month, year);

	sprintf(buf, "%02d/%02d/%02d", day, month, year);
	if (daynum(buf) != input)	/* paranoia */
	{
		error
		(
			"buf wrong (%s) input %d day %d month %d year %d daynum %d",
			buf,
			input,
			day,
			month,
			year,
			daynum(buf)
		);
	}
	return buf;
}

static int
yearsz(year)
int	year;
```

```
{
        if
        (
                year % 4 == 0
                &&
                (year % 100 != 0 || year % 400 == 0)
        )
                return LEAPSZ;
        else
                return YEARSZ;
}

static int
monthsz(month, year)
int     month;
int     year;
{
        static int      months[]        =
        {
                31,
                28,
                31,
                30,
                31,
                30,
                31,
                31,
                30,
                31,
                30,
                31
        };

        if (month == 2 && yearsz(year) == LEAPSZ)
                return months[month − 1] + 1;
        return months[month − 1];
}

static void
check(day, month, year)
int     day;
int     month;
int     year;
{
        if (day <= 0 || month <= 0 || year < FIRSTYEAR)
```

```
		error("daynum: illegal date");
	if (month > 12)
		error("daynum: month too large");
	if (day > monthsz(month, year))
		error("daynum: day too large");
}
```

```
/*
 *      File:   debug.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)debug.c        1.2     86/10/22";

#include        <stdio.h>
#include        <sys/types.h>

#include        "defs.h"

extern char     *myname;

extern bool     O_debug;

/*VARARGS*/
void
debug(s, e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12, e13,
        f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13)
char    *s;
long    e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12, e13;
long    f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13;
{
        if (O_debug)
        {
                printf(s, e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12, e13,
                        f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13);
                putchar('\n');

                fflush(stdout);
        }
}
```

```
/*
 *      File:   error.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)error.c 1.2      86/10/22";

#include        <stdio.h>
#include        <sys/types.h>

#include        "defs.h"

extern char     *myname;

extern bool     O_debug;

/*VARARGS*/
void
error(s, e1, e2, e3, e4, e5, e6, e7, e8, e9, e10)
char    *s;
long    e1, e2, e3, e4, e5, e6, e7, e8, e9, e10;
{
        fflush(stdout);

        fprintf(stderr, "%s: ", myname);
        fprintf(stderr, s, e1, e2, e3, e4, e5, e6, e7, e8, e9, e10);
        putc(´\n´, stderr);

        if (O_debug)
        {
                fflush(stderr);
                abort();
        }
        exit(1);
}
```

```
/*
        C program for floating point log gamma function

        gamma(x) computes the log of the absolute
        value of the gamma function.
        The sign of the gamma function is returned in the
        external quantity signgam.

        The coefficients for expansion around zero
        are #5243 from Hart & Cheney; for expansion
        around infinity they are #5404.

        Calls log and sin.
*/

#include <errno.h>
#include <math.h>

int     errno;
int     signgam = 0;
static double goobie    = 0.91893853320467274178032973;
static double pi        = 3.14159265358979323846264338;

#define M 6
#define N 8
static double p1[] = {
        0.83333333333333101837e-1,
        -.27777777735865004e-2,
        0.793650576493454e-3,
        -.5951896861197e-3,
        0.83645878922e-3,
        -.1633436431e-2,
};
static double p2[] = {
        -.42353689509744089647e5,
        -.20886861789269887364e5,
        -.87627102978521489560e4,
        -.20085274013072791214e4,
        -.43933044406002567613e3,
        -.50108693752970953015e2,
        -.67449507245925289918e1,
        0.0,
};
static double q2[] = {
        -.42353689509744090010e5,
```

```
        −.29803853309256649932e4,
        0.99403074150827709015e4,
        −.15286072737795220248e4,
        −.49902852662143904834e3,
        0.18949823415702801641e3,
        −.23081551524580124562e2,
        0.10000000000000000000e1,
};

double
gamma(arg)
double arg;
{
        double log(), pos(), neg(), asym();

        signgam = 1.;
        if(arg <= 0.) return(neg(arg));
        if(arg > 8.) return(asym(arg));
        return(log(pos(arg)));
}

static double
asym(arg)
double arg;
{
        double log();
        double n, argsq;
        int i;

        argsq = 1./(arg*arg);
        for(n=0,i=M−1; i>=0; i−−){
                n = n*argsq + p1[i];
        }
        return((arg−.5)*log(arg) − arg + goobie + n/arg);
}

static double
neg(arg)
double arg;
{
        double temp;
        double log(), sin(), pos();

        arg = −arg;
        temp = sin(pi*arg);
```

```
	if(temp == 0.) {
		errno = EDOM;
		return(HUGE);
	}
	if(temp < 0.) temp = −temp;
	else signgam = −1;
	return(−log(arg*pos(arg)*temp/pi));
}

static double
pos(arg)
double arg;
{
	double n, d, s;
	register i;

	if(arg < 2.) return(pos(arg+1.)/arg);
	if(arg > 3.) return((arg−1.)*pos(arg−1.));

	s = arg − 2.;
	for(n=0,d=0,i=N−1; i>=0; i−−){
		n = n*s + p2[i];
		d = d*s + q2[i];
	}
	return(n/d);
}
```

```
/*
 *      File:   gen.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)gen.c  1.3      86/10/29";

#include       <math.h>
#include       <sys/types.h>

#include       "defs.h"
#include       "sim.h"

/*
 *      A set of masks to describe the type of generation to do.
 *              G_INTRATE ->      generate constant interest rates
 *              G_EXRATE ->       generate exchange rates
 */
#define        G_INTRATE  0x01
#define        G_EXRATE   0x02

static void    makespace();
static double  gennext();
static double  gen();

double
actual(data, start, length, v, w, func, put, seed)
dt_data*data;
int    start;
int    length;
double v;
double w;
double (*func)();
bool   put;
seed_t *seed;          /* not used */
{
        return (*func)(data, start, length, v, w, put);
}

double
genexrate(data, start, length, v, w, func, put, seed)
dt_data*data;
int    start;
```

```
int     length;
double v;
double w;
double (*func)();
bool    put;
seed_t  *seed;
{
        return gen(data, start, length, v, w, func, put, seed, G_EXRATE);
}

double
genintrate(data, start, length, v, w, func, put, seed)
dt_data*data;
int     start;
int     length;
double v;
double w;
double (*func)();
bool    put;
seed_t  *seed;
{
        return gen(data, start, length, v, w, func, put, seed, G_INTRATE);
}

double
genboth(data, start, length, v, w, func, put, seed)
dt_data*data;
int     start;
int     length;
double v;
double w;
double (*func)();
bool    put;
seed_t  *seed;
{
        return gen(data, start, length, v, w, func, put, seed, G_INTRATE | G_EXRATE);
}

static double
gen(data, start, length, v, w, func, put, seed, type)
dt_data*data;
int     start;          /* the index of the first day of the option */
int     length;         /* length of the option in days */
double v;               /* the annualized volatility of the currency */
double w;               /* option value */
```

```
double (*func)();       /* function to call to do the simulation */
bool    put;
seed_t  *seed;          /* seed for random number generation */
int     type;           /* what type of generation */
{
        register d_data *np;
        register d_data *ip;
        register int    n;
        dt_data         new;
        double          var;    /* annualized variance of the option */
        double          s;
        double          r;
        double          f;

        /*
         *      Skip gives the number of data points to move from the
         *      beginning of the option to the end.  So need to generate
         *      1 more than this.  However, also need to allocate space
         *      for the data point just before the start of the option.
         *
         *      Need to subtract 1 from length because the option length
         *      given includes both the end points, so need to include 1
         *      day less to get the correct number of points.
         */
        n = skip(data, start, length − 1) + 1;

        makespace(&new, n + 1);

        ip = &data−>dt_info[start − 1];
        np = &new.dt_info[0];

        s = ip−>d_exrate;
        r = ip−>d_austr;
        f = ip−>d_usr;
        *np++ = *ip++;

        var = v * v;

        while (n−− > 0)
        {
                *np = *ip;
                if (type & G_EXRATE)
                {
                        np−>d_exrate = s = gennext(s, var, seed);
                }
```

```
                if (type & G_INTRATE)
                {
                        np−>d_austr = r;
                        np−>d_usr = f;
                }
                np++;
                ip++;
        }

        return (*func)(&new, 1, length, v, w, put);
}

/*
 *      Set data up to contain n data points.
 *      Uses a staticly allocated buffer for efficiency.
 */
static void
makespace(data, n)
dt_data*data;
int     n;
{
        static int      length  = −1;
        static char     *space;
        extern char     *salloc();

        /*
         *      Is this the first time or do we need a bigger buffer?
         */
        if (length == −1 || n > length)
        {
                if (length != −1)
                        (void)free(space);

                space = salloc(n * sizeof data−>dt_info[0]);
                length = n;
        }

        data−>dt_info = (d_data *)space;
        data−>dt_count = n;
}

/*
 *      Generate the next currency price given:
 *              S       the current price
 *              var     the variance rate of return on the currency
```

```
 */
static double
gennext(S, var, seed)
double S;
double var;
seed_t  *seed;
{
        extern double  normrv();
        extern double  O_mult;

        return exp(normrv(log(S), var / O_mult, seed));
}
```

```
/*
 *      File:    main.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)main.c          1.8     86/10/23";

/*
 *      sim [−dgvP] [−l<length>] [−M<n>{,<n>}] [−c<reps>] [−n<data_points>] [<file> ...]
 *
 *      Written by −
 *                      Stuart Pook,
 *                      October, 1986.
 */

#include        <stdio.h>
#include        <math.h>
#include        <sys/types.h>

#include        "defs.h"
#include        "sim.h"

#define         talloc(type)    ((type *)salloc(sizeof (type)))

/*
 *      A structure to hold the list of different autocorrelation
 *      fractions in.
 */
typedef struct ml_list  ml_list;
typedef struct ml_list
{
        double ml_value;
        ml_list *ml_next;
};

extern double  atof();

void    analyse();
void    sim();
char    *salloc();

static int      options();
static void     readlist();
```

```
static void	usage();

char	*myname;
/*
 *	Maximum of data points to be read from the input files.
 *	It is fatal error if this is too small.
 */
int	O_dpoints	= 680;
/*
 *	Length of options (in days)
 */
int		O_length	= 28;
static bool	O_Put		= false;
bool		O_variance	= false;
bool		O_debug			= false;
bool		O_nogamma	= false;
/*
 *	Write out the results of each option?
 */
bool		O_option	= false;
/*
 *	Multipication factor for volatility calculations,
 *	should be the number of working days in a year.
 */
double O_mult			= 251.0;
/*
 *	Number of replications of each policy.
 */
int	O_count			= 320;
/*
 *	Fraction of autocorrelation terms to use in the calculation
 *	of the variance of the mean of the result of a run.
 *	Expressed as a fraction of O_count.
 *	Should be bigger than 4.0.
 */
ml_list deflist  =
{
	8.0,
	NULL
};
ml_list *O_M		= NULL;

main(argc, argv)
int	argc;
char	*argv[];
```

```
{
        register ml_list        *p;
        int                     tmpfile;
        dt_data                 data;
        real                    **results;
        extern py_policy        policy[];
        extern char             *strrchr();

        if ((myname = strrchr(argv[0], ´/´)) == NULL || *++myname == ´\0´)
                myname = argv[0];
        argv += options(argv) + 1;

        tmpfile = swapopen();

        readdata(argv, &data);
        checkdata(&data);
        invertdata(&data);

        sim(&data, O_count, policy, tmpfile, O_length, O_Put);

        if ((p = O_M) == NULL)
                p = &deflist;

        while (p != NULL)
        {
                printf("M = %g\n", p−>ml_value);
                analyse(O_count, tmpfile, policy, (int)(O_count / p−>ml_value));

                if ((p = p−>ml_next) != NULL)
                        putchar(´\f´);
        }

        exit(0);
}

static int
options(argv)
register char    *argv[];
{
        register int      i;
        register int      j;

        for (i = 1; argv[i] != NULL && argv[i][0] == ´−´; i++)
        {
                for (j = 1; argv[i][j] != ´\0´; j++)
```

```
switch (argv[i][j])
{
case ´M´:
        readlist(&O_M, argv[i] + j + 1);
        goto break2;

case ´c´:
        if ((O_count = atoi(argv[i] + j + 1)) <= 0)
                error("count (%d) too small", O_count);
        goto break2;

case ´l´:
        if ((O_length = atoi(argv[i] + j + 1)) <= 0)
                error
                (
                        "length (%d) too small",
                        O_length
                );
        goto break2;

case ´P´:
        O_Put = true;
        break;

case ´d´:
        O_debug = true;
        break;

case ´g´:
        O_nogamma = true;
        break;

case ´o´:
        O_option = true;
        break;

case ´v´:
        O_variance = true;
        break;

case ´n´:
        if ((O_dpoints = atoi(argv[i] + j + 1)) <= 0)
                error("maximum number of data points (%d) too small", O_dpoints);
        goto break2;
```

```
                        default:
                                usage();
                        }
        break2:
                ;
        }
        return i − 1;
}

/*
 *      Add the comma seperated list of doubles in s to the list list.
 */
static void
readlist(list, s)
ml_list **list;
char    *s;
{
        register ml_list        *p;
        extern double           atof();

        while ((p = *list) != NULL)
                list = &p−>ml_next;

        while (*s != ´\0´)
        {
                *list = p = talloc(ml_list);
                if ((p−>ml_value = atof(s)) <= 0.0)
                        error("bad value for M (%g)", p−>ml_value);
                list = &p−>ml_next;

                while (*s != ´\0´ && *s != ´,´)
                        s++;
                if (*s == ´,´)
                        s++;
        }
        *list = NULL;
}

static void
usage()
{
        fprintf
        (
                stderr,
                "usage: %s [−dgvP] [−l<length>] [−M<num>{,<num>}] [−c<reps>] [−n<data_points>] [<file
```

```
                myname
        );
        exit(1);
}
```

```
/*
 *      File:   misc.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)misc.c 1.2       86/10/22";

#include        <sys/types.h>

#include        "defs.h"
#include        "sim.h"

static void     doday();

/*
 *      return the number of data points needed to skip
 *      to make the current business day greater than or
 *      equal to number more than today.
 */
int
skip(data, start, number)
register dt_data        *data;
int                     start;
int                     number;
{
        register int    i;
        register int    last;

        if (start < 0 || start >= data->dt_count)
                error("skip: bad value of start (%d)", start);

        i = start;
        last = data->dt_info[i].d_daynum + number;

        while (last > data->dt_info[i].d_daynum)
                if (++i >= data->dt_count)
                        error("skip: ran off end of data");

        return i - start;
}

/*
 *      Return the index (array offset) of the day daynum in the
```

```
 *      data array.
 */
int
index(data, daynum)
register dt_data        *data;
register int            daynum;
{
        register int    i;
        register d_data *ip;

        for (i = 0, ip = data->dt_info; i < data->dt_count; i++, ip++)
                if (daynum == ip->d_daynum)
                        return i;

        error("index: daynum %d not in data set", daynum);
        /*NOTREACHED*/
}

/*
 *      return the index of the day count before the one at start or
 *      the one just previous if this does not exist.
 */
int
skipback(data, start, count)
dt_data         *data;
register int    start;
int             count;
{
        register d_data *ip;
        register int    day;

        ip = &data->dt_info[start];

        for
        (
                day = ip->d_daynum - count;
                start >= 0 && day < ip->d_daynum;
                ip--, start--
        )
                ;

        if (start < 0)
                error("skipback: run off front of data");

        return start;
```

```
}

/*
 *      Check that the day numbers are strictly monotonically increasing
 *      and the exchange rates and interest rates non−negative.
 */
void
checkdata(data)
register dt_data        *data;
{
        register int    i;
        register d_data *ip;

        ip = &data−>dt_info[0];
        doday(ip);

        for (i = 1, ip++; i < data−>dt_count; i++, ip++)
        {
                if (data−>dt_info[i − 1].d_daynum >= ip−>d_daynum)
                {
                        error
                        (
                                "date sequence error, date number %d",
                                ip−>d_daynum
                        );
                }
                doday(ip);
        }
}

static void
doday(ip)
d_data *ip;
{
        if (ip−>d_exrate < 0.0)
        {
                error
                (
                        "negative exchange rate (%f) day %d",
                        ip−>d_exrate,
                        ip−>d_daynum
                );
        }
        if (ip−>d_austr < 0.0)
        {
```

```
                error
                (
                        "negative domestic interest rate (%f) day %d",
                        ip->d_austr,
                        ip->d_daynum
                );
        }
        if (ip->d_usr < 0.0)
        {
                error
                (
                        "negative foreign interest rate (%f) day %d",
                        ip->d_usr,
                        ip->d_daynum
                );
        }
}

void
invertdata(data)
register dt_data        *data;
{
        register d_data *ip;
        register int    i;

        for (i = data->dt_count, ip = &data->dt_info[0]; i > 0; i--, ip++)
                ip->d_exrate = 1.0 / ip->d_exrate;
}

/*
 *      Return the number of the next record be selected given that:
 *              m is the number of records selected so far
 *              t is the current record pointer
 *              n is the total number of records to be selected
 *              N is the total number of records
 *              seed is the address of the random number seed
 *
 *      Reference: Knuth (1981) pp 136-137, Algorithm S.
 */
int
select(m, t, n, N, seed)
int     m;
int     t;
int     n;
int     N;
```

```
seed_t  *seed;
{
        extern double  unif();

        while ((N − t) * unif(seed) >= n − m)
                t++;
        return t;
}
```

```
/*
 *      File:    normal.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)normal.c       1.1     86/10/22";

/*
 *      Collected Algorithms from CACM
 *
 *      Algorithm 304
 *      Normal Curve Integral [S15]
 *      I. D. Hill and S. A. Joyce, Comm. ACM 10 (June 1967), 374
 *
 *      Calculates the tail area of the standarized normal curve
 *      from minus infinity to x.
 *      This routine should give the answer to the accuracy of the machine
 *      being used, except for |x| > 7 where 1 significant digit
 *      is lost.
 */

#include        <math.h>

/*      1/sqrt(2 * pi)   */
#define         RSQRT2PI     0.39894228040143267793994605

double
normal(x)
double x;
{
        int     upper   = 0;
        if (x == 0) return 0.5; else
        {
                double n, x2, y;
                upper = upper == x > 0;
                x = fabs(x); x2 = x * x;
                y = RSQRT2PI * exp(−0.5 * x2);
                n = y / x;
                if (!upper && 1.0 − n == 1.0) return 1.0; else
                if (upper && n == 0) return 0.0; else
                {
                        double s, t;
                        if (x > (upper ? 2.32 : 3.5))
```

```
                {
                        double p1, p2, q1, q2, a1, a2, m;
                        a1 = 2.0; a2 = 0.0;
                        n = x2 + 3.0;
                        p1 = y; q1 = x;
                        p2 = (n − 1.0) * y; q2 = n * x;
                        m = p1 / q1; t = p2 / q2;
                        if (!upper)
                        {
                                m = 1.0 − m; t = 1.0 − t;
                        }
                        for (n = n + 4.0; m != t && s != t; n = n + 4.0)
                        {
                                a1 = a1 − 8.0; a2 = a1 + a2;
                                s = a2 * p1 + n * p2;
                                p1 = p2; p2 = s;
                                s = a2 * q1 + n * q2;
                                q1 = q2; q2 = s;
                                s = m; m = t;
                                if (q2 > 1e30)
                                {
                                        p1 = p1 * 1e−30; p2 = p2 * 1e−30;
                                        q1 = q1 * 1e−30; q2 = q2 * 1e−30;
                                }
                                t = upper ? p2 / q2 : 1.0 − p2 / q2;
                        }
                        return t;
                } else
                {
                        s = x = y * x; n = 1.0; t = 0;
                        for (n = n + 2.0; s != t; n = n + 2.0)
                        {
                                t = s; x = x * x2 / n;
                                s = s + x;
                        }
                        return upper ? 0.5 − s : 0.5 + s;
                }
        }
    }
}
```

```
/*
 *      File:    normrv.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)normrv.c       1.2     86/10/23";

/*
 *      Generate an observation from the normal distribution with
 *      mean u and variance s.
 *      Uses the Box−Muller Method from A Guide to Simulation by
 *      Bratley, Fox and Schrage.
 */

#include       <sys/types.h>
#include       <math.h>

#include       "sim.h"

#define        PI      3.14159265358979323846264 3

double         unif();

double
normrv(u, s, seed)
double u;
double s;
seed_t  *seed;
{
        double u1;
        double u2;

        u1 = unif(seed);
        u2 = unif(seed);

        return u + cos(2 * PI * u1) * sqrt(−2 * log(u2) * s);
}
```

```
/*
 *      File:   option.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)option.c       1.2     86/10/23";

#include       <math.h>

extern double  normal();

static double   d();

double
calldelta(s, t, r, f, c, v)
double s;       /* spot currency price */
double t;       /* option duration */
double r;       /* domestic interest rates */
double f;       /* foreign interest rates */
double c;       /* exercise price */
double v;       /* volatility */
{
        return exp(−f * t) * normal(d(s, t, r, f, c, v));
}

double
callvalue(s, t, r, f, c, v)
double s;       /* spot currency price */
double t;       /* option duration */
double r;       /* domestic interest rates */
double f;       /* foreign interest rates */
double c;       /* exercise price */
double v;       /* volatility */
{
        double d1;

        d1 = d(s, t, r, f, c, v);
        return exp(−f * t) * s * normal(d1) − exp(−r * t) * c * normal(d1 − v * sqrt(t));
}

/*
 *      Return the delta of a put option, ie the partial derivative of P wrt S.
 *      Garman and Kohlhagen 1983.
```

```
 */
double
putdelta(s, t, r, f, c, v)
double s;       /* spot currency price */
double t;       /* option duration */
double r;       /* domestic interest rates */
double f;       /* foreign interest rates */
double c;       /* exercise price */
double v;       /* volatility */
{
        return −exp(−f * t) * normal(−d(s, t, r, f, c, v));
}

/*
 *      return the value of a european put option from
 *      Garman and Kohlhagen 1983 p234.
 */
double
putvalue(s, t, r, f, c, v)
double s;       /* spot currency price */
double t;       /* option duration */
double r;       /* domestic interest rates */
double f;       /* foreign interest rates */
double c;       /* exercise price */
double v;       /* volatility */
{
        double d1;

        d1 = d(s, t, r, f, c, v);
        return −exp(−f * t) * s * normal(−d1) + exp(−r * t) * c * normal(−d1 + v * sqrt(t));
}

static double
d(s, t, r, f, c, v)
double s;       /* spot currency price */
double t;       /* option duration */
double r;       /* domestic interest rates */
double f;       /* foreign interest rates */
double c;       /* exercise price */
double v;       /* volatility */
{
        return (log(s / c) + (r − f + 0.5 * v * v) * t) / (v * sqrt(t));
}
```

```
/*
 *      File:   policy.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)policy.c       1.9     86/10/29";

#include       <sys/types.h>

#include       "sim.h"

double syn0();
double syn1();
double syn4();
double nosyn();
double volshort();
double vollong();
double actual();
double genboth();
double genintrate();
double genexrate();

py_policy      policy[]        =
{
        {
                syn0,
                volshort,
                actual,
                "daily adj (short vol) (actual)"
        },
        {
                syn1,
                volshort,
                actual,
                "bi−daily adj (short vol) (actual)"
        },
        {
                syn4,
                volshort,
                actual,
                "weekly adj (short vol) (actual)"
        },
        {
```

```
		nosyn,
		volshort,
		actual,
		"no portfolio (short vol) (actual)"
	},
	{
		syn0,
		vollong,
		actual,
		"daily adj (long vol) (actual)"
	},
	{
		syn1,
		vollong,
		actual,
		"bi−daily adj (long vol) (actual)"
	},
	{
		syn4,
		vollong,
		actual,
		"weekly adj (long vol) (actual)"
	},
	{
		nosyn,
		vollong,
		actual,
		"no portfolio (long vol) (actual)"
	},
	{
		syn0,
		volshort,
		genboth,
		"daily adj (short vol) (both simulated)"
	},
	{
		syn1,
		volshort,
		genboth,
		"bi−daily adj (short vol) (both simulated)"
	},
	{
		syn4,
		volshort,
		genboth,
```

```
		"weekly adj (short vol) (both simulated)"
	},
	{
		nosyn,
		volshort,
		genboth,
		"no portfolio (short vol) (both simulated)"
	},
	{
		syn0,
		vollong,
		genboth,
		"daily adj (long vol) (both simulated)"
	},
	{
		syn1,
		vollong,
		genboth,
		"bi−daily adj (long vol) (both simulated)"
	},
	{
		syn4,
		vollong,
		genboth,
		"weekly adj (long vol) (both simulated)"
	},
	{
		nosyn,
		vollong,
		genboth,
		"no portfolio (long vol) (both simulated)"
	},
	{
		syn0,
		volshort,
		genintrate,
		"daily adj (short vol) (intrate simulated)"
	},
	{
		syn1,
		volshort,
		genintrate,
		"bi−daily adj (short vol) (intrate simulated)"
	},
	{
```

```
		syn4,
		volshort,
		genintrate,
		"weekly adj (short vol) (intrate simulated)"
	},
	{
		nosyn,
		volshort,
		genintrate,
		"no portfolio (short vol) (intrate simulated)"
	},
	{
		syn0,
		vollong,
		genintrate,
		"daily adj (long vol) (intrate simulated)"
	},
	{
		syn1,
		vollong,
		genintrate,
		"bi−daily adj (long vol) (intrate simulated)"
	},
	{
		syn4,
		vollong,
		genintrate,
		"weekly adj (long vol) (intrate simulated)"
	},
	{
		nosyn,
		vollong,
		genintrate,
		"no portfolio (long vol) (intrate simulated)"
	},
	{
		syn0,
		volshort,
		genexrate,
		"daily adj (short vol) (exrate simulated)"
	},
	{
		syn1,
		volshort,
		genexrate,
```

```
		"bi−daily adj (short vol) (exrate simulated)"
	},
	{
		syn4,
		volshort,
		genexrate,
		"weekly adj (short vol) (exrate simulated)"
	},
	{
		nosyn,
		volshort,
		genexrate,
		"no portfolio (short vol) (exrate simulated)"
	},
	{
		syn0,
		vollong,
		genexrate,
		"daily adj (long vol) (exrate simulated)"
	},
	{
		syn1,
		vollong,
		genexrate,
		"bi−daily adj (long vol) (exrate simulated)"
	},
	{
		syn4,
		vollong,
		genexrate,
		"weekly adj (long vol) (exrate simulated)"
	},
	{
		nosyn,
		vollong,
		genexrate,
		"no portfolio (long vol) (exrate simulated)"
	},
	{
		(double (*)())0,
		(double (*)())0,
		(double (*)())0,
		(char *)0
	},
};
```

```
/*
 *      File:   read.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)read.c 1.2       86/10/23";

#include        <stdio.h>
#include        <sys/types.h>

#include        "sim.h"

static int      readfile();

void
readdata(argv, data)
char    **argv;
dt_data*data;
{
        extern char     *salloc();
        extern int      O_dpoints;

        data->dt_info = (d_data *)salloc(O_dpoints * sizeof data->dt_info[0]);

        if (*argv == NULL)
        {
                data->dt_count = readfile
                (
                        stdin,
                        &data->dt_info[0],
                        &data->dt_info[O_dpoints],
                        "<stdin>"
                );
        }
        else
        {
                data->dt_count = 0;
                for (; *argv != NULL; argv++)
                {
                        register FILE   *fp;

                        if ((fp = fopen(*argv, "r")) == NULL)
                                fcouldnot("open", *argv);
```

```
                        data−>dt_count += readfile
                        (
                                fp,
                                &data−>dt_info[data−>dt_count],
                                &data−>dt_info[O_dpoints],
                                *argv
                        );
                        if (fclose(fp) == EOF)
                                fcouldnot("close", *argv);
                }
        }
}

static int
readfile(fp, data, maxdata, file)
FILE            *fp;
register d_data *data;
register d_data *maxdata;
char            *file;
{
        double x;
        double a;
        double u;
        char   id[64];
        d_data *p;
        int    i;

        p = data;
        while ((i = fscanf(fp, " %64s %lf %lf %lf", id, &x, &a, &u)) == 4)
        {
                if (data >= maxdata)
                        error("too much data on file \"%s\"", file);
                data−>d_daynum = daynum(id);
                data−>d_exrate = x;
                data−>d_austr = a;
                data−>d_usr = u;
                data++;
        }

        if (i != EOF)
                error("bad input found on \"%s\"", file);
        return data − p;
}
```

```
/*
 *      File:   salloc.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)salloc.c         1.3     86/10/23";

#include        <stdio.h>
#include        <sys/types.h>

extern char     *myname;

char    *
salloc(size)
register int    size;
{
        register char   *p;
        extern char     *malloc();
        extern char     *sysmess();

        if ((p = malloc((unsigned)size)) == NULL)
        {
                fflush(stdout);
                fprintf
                (
                        stderr,
                        "%s: Ran out of memory: %s.\n",
                        myname,
                        sysmess()
                );
                exit(1);
        }
        return p;
}
```

```
/*
 *      File:   sim.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)sim.c  1.9      86/10/26";

#include        <stdio.h>
#include        <math.h>
#include        <sys/types.h>

#include        "defs.h"
#include        "sim.h"

/*
 *      Date of start of first simulation
 */
#define         FIRSTDAY    "15/02/84"

double BSputval();
double BScallval();
void    swapout();

void
sim(data, reps, policy, fd, olength, put)
dt_data         *data;
int             reps;   /* number of replications of each policy */
py_policy       policy[];
int             fd;
int             olength;        /* option length */
bool            put;
{
        register py_policy      *p;
        real                    *results;
        int                     firstday;
        int                     lastday;
        int                     dpoints;
        double                  (*value)();
        extern bool             O_option;

        if (put)
                value = BSputval;
        else
```

```
		value = BScallval;

	/*
	 *	Find the index of the FIRSTDAY in the data.
	 */
	firstday = index(data, daynum(FIRSTDAY));
	/*
	 *	find the index of the last day on which can start a simulation
	 */
	lastday = skipback(data, data->dt_count - 1, olength - 1);
	dpoints = lastday - firstday + 1;
	if (reps > dpoints)
		error
		(
			"sim: more replications (%d) than data points (%d)",
			reps,
			dpoints
		);

	results = (real *)salloc(reps * sizeof (real));

	for (p = &policy[0]; p->py_vol != (double (*)())0; p++)
	{
		int	i;
		/*
		 *	how far through the list of possible starting dates
		 */
		int	t;
		double mean;
		seed_t	gen_seed;
		seed_t	start_seed;

		mean = 0.0;
		gen_seed = GEN_SEED;
		start_seed = START_SEED;

		if (O_option)
			printf("sim: policy %s\n", p->py_name);
		t = 0;
		for (i = 0; i < reps; i++)
		{
			double v;
			double w;
			double r;
			int	start;	/* start of this simulation */
```

```
                int     length; /* length of this option */

                t = select(i, t, reps, dpoints, &start_seed);
                start = firstday + t;
                length =
                        data->dt_info
                        [
                                start
                                +
                                skip(data, start, olength - 1)
                        ].d_daynum
                        -
                        data->dt_info[start].d_daynum
                        +
                        1;

                v = (*p->py_vol)(data, start, length);
                w = (*value)(data, start, length, v);
                r = (*p->py_gen)(data, start, length, v, w, p->py_syn, put, &gen_seed);
                if (O_option)
                        printf("sim: start %s length %d v %g, w %g, r %g\n", numday(data->dt_info[s
                results[i] = r;

                mean += w;
                t++;
        }
        swapout(p - policy, reps, results, fd);
        printf("%s mean option value %g\n", p->py_name, mean / reps);
    }
}
```

```
/*
 *      File:   swap.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)swap.c         1.4     86/10/22";

#include        <local−system>
#include        <sys/types.h>
#include        <signal.h>
#include        <fcntl.h>

#include        "defs.h"
#include        "sim.h"

extern long     lseek();

static char     tmpname[]       = "/tmp/SwapXXXXXX";

/*
 *      Create and remove a temporary file, return a file descriptor
 *      to it.  The file will be empty.
 */
int
swapopen()
{
        register int    (*sigint)();
        register int    (*sigterm)();
        register int    fd;

        (void)mktemp(tmpname);

        if ((int)(sigint = signal(SIGINT, SIG_IGN)) == SYSERROR)
                fcouldnot("signal", "SIGINT");
        if ((int)(sigterm = signal(SIGTERM, SIG_IGN)) == SYSERROR)
                fcouldnot("signal", "SIGTERM");

        if ((fd = open(tmpname, O_RDWR | O_EXCL | O_CREAT, 0)) == SYSERROR)
                fcouldnot("open", tmpname);
        if ((int)unlink(tmpname) == SYSERROR)
                fcouldnot("unlink", tmpname);

        if ((int)signal(SIGINT, sigint) == SYSERROR)
```

```
		fcouldnot("signal", "SIGINT");
	if ((int)signal(SIGTERM, sigterm) == SYSERROR)
		fcouldnot("signal", "SIGTERM");

	return fd;
}

/*
 *	Write the array containing count reals into the index´th position
 *	on the file fd.
 */
void
swapout(index, count, array, fd)
int	index;
int	count;
real	*array;
int	fd;
{
	count *= sizeof array[0];

	if ((int)lseek(fd, (long)index * count, 0) == SYSERROR)
		fcouldnot("lseek", tmpname);

	if (write(fd, (char *)array, count) != count)
		fcouldnot("write", tmpname);
}

/*
 *	Read the array containing count reals from the index´th position
 *	on the file fd into array.
 */
void
swapin(index, count, array, fd)
int	index;
int	count;
real	*array;
int	fd;
{
	count *= sizeof array[0];

	if ((int)lseek(fd, (long)index * count, 0) == SYSERROR)
		fcouldnot("lseek", tmpname);

	if (read(fd, (char *)array, count) != count)
		fcouldnot("read", tmpname);
```

```
}
```

```
/*
 *      File:   syn.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)syn.c  1.6      86/10/23";

#include        <stdio.h>
#include        <sys/types.h>

#include        "defs.h"
#include        "assert.h"
#include        "sim.h"

char    *numday();
double putdelta();
double calldelta();

static double
syn(data, start, length, v, w, put, skipnum)
dt_data*data;
int     start;
int     length;         /* option duration in days */
double v;               /* volatility */
double w;               /* option value */
bool    put;
int     skipnum;        /* number of days to skip between adjustments */
{
        register d_data*ip;
        register int    n;
        double          (*delta)();     /* delta calculation function */
        double          u;      /* current quantity of US dollars */
        double          a;      /* current quantity of Aust dollars */
        double          c;      /* exercise price */
        int             firstday;

        if (put)
                delta = putdelta;
        else
                delta = calldelta;
        if (start <= 0)
                error("syn: start too small (%d)", start);
```

```
	ip = &data->dt_info[start - 1];
	c = ip->d_exrate;

	debug
	(
		"syn: start %s length = %d c = %f v = %f",
		numday(data->dt_info[start].d_daynum),
		length,
		c,
		v
	);
	/*
	 *	set n to the number of data points to be processed
	 *	in the life on the option (not including the last days
	 *	data point).
	 */
	n = skip(data, start, length - 1);
	a = w;
	u = 0.0;
	debug
	(
		"syn: %s s = %f r = %f f = %f a = %f",
		numday(ip->d_daynum), ip->d_exrate, ip->d_austr, ip->d_usr, a
	);

	ip++;
	firstday = ip->d_daynum - 1;
	u *= 1.0 + ip->d_usr / DAYS / PERCENT;
	a *= 1.0 + ip->d_austr / DAYS / PERCENT;

	while (n > 0)
	{
		register int	i;
		double		d;

		d = (*delta)
		(
			ip->d_exrate,
			(double)(length - ip->d_daynum + firstday) / DAYS,
			ip->d_austr / PERCENT,
			ip->d_usr / PERCENT,
			c,
			v
		) - u;
		u += d;
```

```
		a −= d * ip−>d_exrate;
		debug
		(
			"syn: %s u = %f a = %f d = %f s = %f r = %f f = %f today = %d n = %d",
			numday(ip−>d_daynum),
			u, a, d, ip−>d_exrate,
			ip−>d_austr, ip−>d_usr, ip−>d_daynum, n
		);
		/*
		 *	skip over skipnum data points but don´t
		 *	run off the end of the option period.
		 */
		for (i = 0; i <= skipnum && n > 0; i++, n−−)
		{
			register int	days;

			days = ip−>d_daynum;
			ip++;
			days = ip−>d_daynum − days;
			u *= 1.0 + ip−>d_usr / DAYS / PERCENT * days;
			a *= 1.0 + ip−>d_austr / DAYS / PERCENT * days;
		}
	}

	if (length != ip−>d_daynum − firstday)
		error("option finished on non business day");

	debug
	(
		"syn: %s u = %f a = %f s = %f r = %f f = %f",
		numday(ip−>d_daynum), u, a, ip−>d_exrate, ip−>d_austr, ip−>d_usr
	);

	if (put && c > ip−>d_exrate)
		return u * ip−>d_exrate + a + ip−>d_exrate − c;

	if (!put && ip−>d_exrate > c)
		return u * ip−>d_exrate + a − ip−>d_exrate + c;

	return u * ip−>d_exrate + a;
}

double
syn0(data, start, length, v, w, put)
dt_data*data;
```

```
int	start;
int	length; /* option duration in days */
double v;	/* volatility */
double w;	/* option value */
bool	put;
{
	return syn(data, start, length, v, w, put, 0);
}

double
syn1(data, start, length, v, w, put)
dt_data*data;
int	start;
int	length; /* option duration in days */
double v;	/* volatility */
double w;	/* option value */
bool	put;
{
	return syn(data, start, length, v, w, put, 1);
}

double
syn4(data, start, length, v, w, put)
dt_data*data;
int	start;
int	length; /* option duration in days */
double v;	/* volatility */
double w;	/* option value */
bool	put;
{
	return syn(data, start, length, v, w, put, 4);
}

double
nosyn(data, start, length, v, w, put)
dt_data*data;
int	start;
int	length; /* option duration in days */
double v;	/* volatility */
double w;	/* option value */
bool	put;
{
	register d_data*ip;
	register int	n;
	int		firstday;
```

```
int             yesterday;
double          a;      /* current Australian dollars */
double          r;      /* profit */
double          c;      /* exercise price */
char            startdate[32];

assert(start > 0);
ip = &data->dt_info[start - 1];
c = ip->d_exrate;

n = skip(data, start, length - 1);

a = w;
ip++;
firstday = yesterday = ip->d_daynum - 1;
while (n-- > 0)
{
        a *= 1.0 + ip->d_austr / DAYS / PERCENT * (ip->d_daynum - yesterday);
        yesterday = ip->d_daynum;
        ip++;
}

if (length != ip->d_daynum - firstday)
        error("option finished on non business day");

a *= 1.0 + ip->d_austr / DAYS / PERCENT * (ip->d_daynum - yesterday);

if (put && c > ip->d_exrate)
        r = a + ip->d_exrate - c;
else if (!put && ip->d_exrate > c)
        r = a - ip->d_exrate + c;
else
        r = a;
strcpy(startdate, numday(data->dt_info[start].d_daynum));
debug
(
        "nosyn: start %s finish %s length %d vol %g val %g c %g a %g ex %g r %g",
        startdate,
        numday(ip->d_daynum),
        length,
        v,
        w,
        c,
        a,
        ip->d_exrate,
```

```
                r
        );
        return r;
}
```

```
/*
 *      File:   sys.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)sys.c   1.1      86/10/22";

#include       <stdio.h>
#include       <sys/types.h>

extern char     *myname;

void
couldnot(what, which)
char    *what;
char    *which;
{
        extern char     *sysmess();

        fprintf
        (
                stderr,
                "%s: could not %s \"%s\": %s\n",
                myname,
                what,
                which,
                sysmess()
        );
}

void
fcouldnot(what, which)
char    *what;
char    *which;
{
        couldnot(what, which);
        exit(1);
}

/*
 *      Resolve ´errno´ into an ascii message.
 */
char    *
```

```
sysmess()
{
        extern int      errno;
        extern int      sys_nerr;
        extern char     *sys_errlist[];

        if (errno < 0 || errno > sys_nerr)
                return "Unknown error";
        return sys_errlist[errno];
}
```

```
/*
 *      File:    unif.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)unif.c  1.4      86/10/30";

/*
 *      The Tausworthe (1965) random nummber generator as given in
 *      Bratley, Fox and Schrage (1983) p189 and p203.
 */
#include        <stdio.h>
#include        <sys/types.h>

#include        "assert.h"
#include        "sim.h"

double
unif(x)
seed_t  *x;
{
        long    y;

        y = *x;
        y = (y & 0x7fffffff) >> 13;     /* right shift Y by q bits */

        *x = y = y ^ *x;                /* low−order bits now updated */

        y <<= 18;                       /* left shift Y by k − q bits */

        *x = y ^ *x;
        *x = *x & 0x7fffffff;           /* set sign bit positive */

        return *x / 2147483647.0;
}
```

```
/*
 *      File:   val.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)val.c   1.4      86/10/23";

#include        <sys/types.h>

#include        "defs.h"
#include        "sim.h"

double callvalue();
double putvalue();

double
BScallval(data, start, length, vol)
register dt_data        *data;
register int            start;
int                     length;
double                  vol;
{
        if (--start < 0)
                error("b_and_s: run off front of data");

        return callvalue
        (
                data->dt_info[start].d_exrate,
                length / 365.0,
                data->dt_info[start].d_austr / PERCENT,
                data->dt_info[start].d_usr / PERCENT,
                data->dt_info[start].d_exrate,
                vol
        );
}

double
BSputval(data, start, length, vol)
register dt_data        *data;
register int            start;
int                     length;
double                  vol;
{
```

```
	if (−−start < 0)
		error("b_and_s: run off front of data");

	return putvalue
	(
		data−>dt_info[start].d_exrate,
		length / 365.0,
		data−>dt_info[start].d_austr / PERCENT,
		data−>dt_info[start].d_usr / PERCENT,
		data−>dt_info[start].d_exrate,
		vol
	);
}

double
badb_and_s(data, start, length, vol)
register dt_data	*data;
register int		start;
int			length;
double			vol;
{
	if (−−start < 0)
		error("b_and_s: run off front of data");

	return callvalue
	(
		data−>dt_info[start].d_exrate,
		length / 365.0,
		data−>dt_info[start].d_austr / PERCENT,
		data−>dt_info[start].d_usr / PERCENT,
		data−>dt_info[start].d_exrate,
		vol
	) / 10;
}
```

```
/*
 *      File:   vol.c
 *      Author:         Stuart Pook
 *
 *      Operations Research Honours 1986
 */

static char     SccsId[]        = "@(#)vol.c   1.3      86/10/22";

#include        <stdio.h>
#include        <math.h>
#include        <sys/types.h>

#include        "defs.h"
#include        "sim.h"

static double   volatility();

double
volshort(data, start, length)
dt_data*data;
int     start;
int     length;
{
        return volatility(data, skipback(data, start, length), length);
}

double
vollong(data, start, length)
dt_data*data;
int     start;
int     length;
{
        return volatility(data, 0, start − 1);
}

/*
 *      calculate the volatility of the currency in data from the
 *      day start for a period of length days.
 */
static double
volatility(data, start, length)
register dt_data        *data;
int                     start;
int                     length;
```

```
{
        register int      i;
        int               lastday;
        double            sum;
        double            sum2;
        int               n;
        extern double     O_mult;
        extern bool       O_nogamma;
        extern bool       O_debug;

        debug
        (
                "volatility: start %s length %d",
                numday(data->dt_info[start].d_daynum),
                length
        );
        lastday = data->dt_info[start].d_daynum + length;
        sum = 0.0;
        sum2 = 0.0;
        n = 0;

        for (i = start + 1; i < data->dt_count; i++)
        {
                register doubleR;

                if (data->dt_info[i].d_daynum >= lastday)
                        break;

                R = log(data->dt_info[i].d_exrate / data->dt_info[i - 1].d_exrate);
                sum += R;
                sum2 += R * R;
                n++;
        }

        if (i == data->dt_count)
                error("volatility: ran off end of data set");

        if (n < 2)
                error("not enough data points included");

        if (O_nogamma == false)
        {
                return
                        sqrt(n / 2.0)
                        * Gamma((n - 3.0) / 2.0 + 1.0)
```

```
                        / Gamma(n / 2.0)
                        / n
                        * sqrt(O_mult * (n * sum2 − sum * sum));
        }
        else
        {
                double          var;

                var = (n * sum2 − sum * sum) / (n * (n − 1.0));
                return sqrt(O_mult * var);
        }

        /*NOTREACHED*/
}
```

10. Appendix C — Other Program Source

Appendix C contains the source for the miscellaneous computer programs used in this project.

```
/*
 *      uniftest [−M<num>] [−t<num>] [−n<num>]
 *
 *      Operations Research Honours 1986
 *
 *      A program to test the randomness of the random number generator
 *      unif().   Does the tests for each element of seeds[].
 *
 *      Options:
 *              −M      gives the number of observations to use in the chi
 *                      squared test.
 *              −t      gives the value of t to use in the Maximum−of−t test,
 *                      Knuth (1981) p68 Test H.
 *              −n      gives the value of n to use in the Maximum−of−t test.
 *
 *      Written by −
 *                      Stuart Pook,
 *                      October, 1986.
 */

#include      <local−system>
#include      <stdio.h>
#include      <sys/types.h>
#include      <math.h>

#include      "../sim.h"
#include      "../defs.h"

void    couldnot();
void    fcouldnot();
void    usage();
char    *salloc();
void    chi2();
void    ks();

/*
 *      The function, whose properties as a uniform [0, 1) random number
 *      generator, we are testing.
 */
double unif();

char    *myname;
bool    O_debug       = false;
int     O_M   = 1360;
int     O_t     = 45;
```

```
int	O_n	= 30;

seed_t	seeds[] =
{
	524287,
	GEN_SEED,
	START_SEED,
	1933985544,
	918807827,
};
#define		SEEDSSZ		(sizeof seeds / sizeof seeds[0])

main(argc, argv)
int	argc;
char	*argv[];
{
	int		i;
	int		error;
	extern char	*strrchr();

	if ((myname = strrchr(argv[0], ´/´)) == NULL || *++myname == ´\0´)
		myname = argv[0];
	argv += options(argv) + 1;
	if (*argv != NULL)
		usage();

	for (i = 0; i < SEEDSSZ; i++)
	{
		printf("seed = %ld", (long)seeds[i]);
		chi2(O_M, seeds[i]);
		ks(O_t, O_n, seeds[i]);
		putchar(´\n´);
	}
	exit(0);
}

/*
 *	Do a chi−squared test on the M random numbers starting with a initial
 *	seed of seed.  Calculate the number of partitions as 4*Mˆ(2/5).
 *	Reference: Bratley, Fox and Schrage (1983) pp 204−205 (Section 6.6.1).
 */
void
chi2(M, seed)
int		M;
seed_t		seed;
```

```
{
        register int    i;
        register int    n;      /* number of partitions */
        /*
         *      expected number of elements in each partition
         */
        double          f;
        double          chi;
        /*
         *      actual number of elements in each partition
         */
        int             *cells;

        n = 4.0 * pow((double)M, 2.0 / 5.0);
        cells = (int *)salloc(n * sizeof(int));

        for (i = 0; i < n; i++)
                cells[i] = 0;

        for (i = 0; i < M; i++)
        {
                register int    j;
                double          x;

                x = unif(&seed);
                for (j = 1; j <= n; j++)
                {
                        if (x <= (double)j / n)
                        {
                                cells[j - 1]++;
                                break;
                        }
                }
        }

        chi = 0.0;
        f = (double)M / n;
        for (i = 0; i < n; i++)
        {
                if (O_debug)
                        printf("cells[i] %d\tf %g\n", cells[i], f);
                chi += (cells[i] - f) * (cells[i] - f) / f;
        }

        (void)free((char *)cells);
```

```
	printf("\tchi2(%d) = %g", n, chi);
}

/*
 *	Do a Maximum−of−t test on the n * t random numbers starting with
 *	initial seed seed.
 *	Reference: Knuth (1981) p 68 Test H.
 */
void
ks(t, n, seed)
int	t;
int	n;
seed_t	seed;
{
	register int	i;
	register int	j;
	double		kplus;
	double		kminus;
	double		*V;
	int		dcomp();

	V = (double *)salloc(n * sizeof V[0]);

	for (j = 0; j < n; j++)
	{
		register int	i;
		double		x;

		V[j] = unif(&seed);
		for (i = 1; i < t; i++)
			if ((x = unif(&seed)) > V[j])
				V[j] = x;
	}

	qsort((char *)V, n, sizeof V[0], dcomp);

	kplus = 1.0 / n − pow(V[0], (double)n);
	kminus = pow(V[0], (double)n);

	if (O_debug)
		printf("V[0] =\%g\n", V[0]);
	for (j = 1; j < n; j++)
	{
		double tmp;
		double f;
```

```
                f = pow(V[j], (double)t);
                if (O_debug)
                        printf("V[%d] =\%g\n", j, V[j]);
                if ((tmp = (double)j / n − f) > kplus)
                        kplus = tmp;
                if ((tmp = f − (double)(j − 1) / n) > kminus)
                        kminus = tmp;
        }

        kplus *= sqrt((double)n);
        kminus *= sqrt((double)n);
        printf("\tk(%d)+ = %g\tk(%d)− = %g", n, kplus, n, kminus);

        free((char *)V);
}


/*
 *      Compare 2 floating point numbers, used by qsort.
 */
int
dcomp(p1, p2)
double *p1;
double *p2;
{
        if (*p1 < *p2)
                return −1;
        if (*p1 > *p2)
                return 1;
        return 0;
}

void
couldnot(what, which)
char    *what;
char    *which;
{
        extern char     *sysmess();

        fprintf
        (
                stderr,
                "%s: could not %s \"%s\": %s\n",
                myname,
```

```
                what,
                which,
                sysmess()
        );
}

void
fcouldnot(what, which)
char    *what;
char    *which;
{
        couldnot(what, which);
        exit(1);
}

char    *
salloc(size)
register int      size;
{
        register char    *p;
        extern char      *malloc();

        if ((p = malloc((unsigned)size)) == NULL)
        {
                fprintf(stderr, "%s: Ran out of memory.\n", myname);
                exit(1);
        }
        return p;
}

int
options(argv)
register char    *argv[];
{
        register int      i;
        register int      j;

        for (i = 1; argv[i] != NULL && argv[i][0] == ´−´; i++)
        {
                for (j = 1; argv[i][j] != ´\0´; j++)
                        switch (argv[i][j])
                        {
                        case ´d´:
                                O_debug = true;
                                break;
```

```
                        case ´n´:
                                if ((O_n = atoi(&argv[i][j + 1])) <= 0)
                                        usage();
                                goto break2;

                        case ´t´:
                                if ((O_t = atoi(&argv[i][j + 1])) <= 0)
                                        usage();
                                goto break2;

                        case ´M´:
                                if ((O_M = atoi(&argv[i][j + 1])) <= 0)
                                        usage();
                                goto break2;

                        default:
                                usage();
                        }
        break2:
                ;
        }
        return i − 1;
}

void
usage()
{
        fprintf
        (
                stderr,
                "usage: %s [−M<num>] [−n<num>] [−t<num>]\n",
                myname
        );
        exit(1);
}

/*
 *      Resolve ´errno´ into an ascii message.
 */
char    *
sysmess()
{
        extern int      errno;
        extern int      sys_nerr;
```

```
        extern char      *sys_errlist[];

        if (errno < 0 || errno > sys_nerr)
                return "Unknown error";
        return sys_errlist[errno];
}
```

CONTENTS

LIST OF TABLES