JOIM
www.joim.com

JOURNAL OF INVESTMENT MANAGEMENT, Vol. 7, No. 2, (2009), pp. 73–85

SURVEYS AND CROSSOVERS

DEALING WITH DIMENSION: OPTION PRICING ON FACTOR TREES

Sanjiv R. Das[a,*] *and Brian Granger*[b]

We present a scheme for pricing derivatives on M assets on K-factor recombining trees with N periods. The computational complexity of these trees is $O(N^{K+1})$, i.e. polynomial in N, making it possible to price a wide range of derivatives without resorting to Monte Carlo simulation. Numerical implementation examples are provided, along with a discussion of the issues that arise when these models are implemented on multicore processors. A calibration example is provided that shows how individual assets may be embedded on a multi-factor tree.

1 Introduction

The complexity of derivative securities has grown immensely since the advent of the single stock option pricing model of Black and Scholes (1973), and the corresponding discrete-time tree model of Cox *et al.* (1979). Derivatives are now being written on multiple underlying securities, and even when written on the same underlying, there may be several stochastic processes driving the evolution of the stochastic process of the security on which the derivative is written.

We present a methodology for pricing derivative securities on high-dimensional lattices based on an underlying factor structure. To fix ideas consider the case of pricing an option on a single stock. There are two approaches we might use. One, we directly model the stock price on a binomial tree and price options on it (a single factor approach, where the stock is the factor itself). Two, we may represent the movement of the stock as a function of a set of underlying factors plus an idiosyncratic factor (a multi-factor approach). The advantage of the latter approach is that we may model many stocks using the same lattice, whereas in the single factor model, a separate tree would be required for each stock. We would also then need to account for the correlations between stocks (across trees) which would make the model eventually more complicated than the multi-factor approach. In our approach, correlations among all the assets we may model on the tree are parsimoniously generated from the correlations

*Corresponding author. Leavey School of Business, Santa Clara University, Santa Clara, CA 95053, USA.

[a]Leavey School of Business, Santa Clara University, Santa Clara, CA 95053, USA. E-mail: srdas@scu.edu

[b]Physics Department, California Polytechnic State University, San Luis Obispo, CA 93407, USA. E-mail: bgranger@calpoly.edu

of the smaller factor set. The multi-factor approach also enables the pricing of options on portfolios.

In the ensuing exposition, we assume that the reader is familiar with option pricing trees used widely in the finance literature in papers by Cox *et al.* (1979), Nelson and Ramaswamy (1990), amongst others. The approach in this paper will result in a representation of a K-factor model on a $(K+1)$-dimensional lattice over N time periods. We impose two conditions on this lattice structure:

- The factors are orthogonal. This is achieved by the appropriate modeling of the factors. The numerical illustration in the paper will also make this aspect of the model clear.
- Each single factor is representable by a recombining tree, such that the number of nodes after N periods is linear in N. For example, in the case of single binomial trees, after N periods, we will have $(N+1)$ nodes.

The outcome of these conditions is that the K-factor tree is formed as the *product space* of the individual factor trees. Since each tree recombines, the product space is also recombining. From a computational standpoint, we get the following three results.

(1) The computational effort for each single tree is $O(N^2)$.
(2) The complexity for the K-factor tree will be $O(N^{K+1})$, i.e. the computational effort is polynomial in N. If the trees did not recombine, then complexity would be $O(2^{NK})$, i.e. exponential in $N \times K$.
(3) For $K = 3$ (and even $K = 4$), we are able to compute a 50-step tree in a few seconds, which is fast and covers most models, since even interest rate models are at most projectable onto a set of 3 to 4 factors.

We present an illustration of the implementation of the model using data on IBM and projecting the evolution of the stock onto a 3-factor tree. We show that the model returns values exactly the same as in the one-factor model and corresponds to Black–Scholes.

Further, we also begin to explore the issues that emerge when these K-factor tree algorithms are implemented on recent multi-CPU and multi-core hardware.

Finally, we present the algorithmic set up required for pricing options on portfolios of many assets. Next, we move on to elucidating the model notation and representation.

2 Model features

2.1 Virtual multi-factor trees

We assume a set of securities $S_i, i = 1, \ldots, M$. The factors are denoted $F_j, j = 1, \ldots, K$, and the times in the model are denoted $t = 0, 1, \ldots, N$. We create one recombining tree for each factor F_j, where we denote each node in the tree as $F_j(t,s)$, where t is the time and s is the index of the node level at each time t. A two-period tree would appear as shown in Figure 1. We may think of the factor

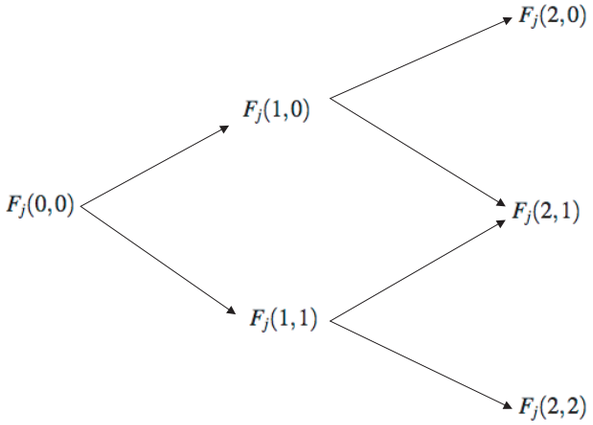

Figure 1 A tree for a single factor F_j. Nodes are denoted with notation $F(t,s)$, where t denotes time, and s denotes the level at time t, with $s = 0$ the topmost node, and $s = t$ the lowest node.

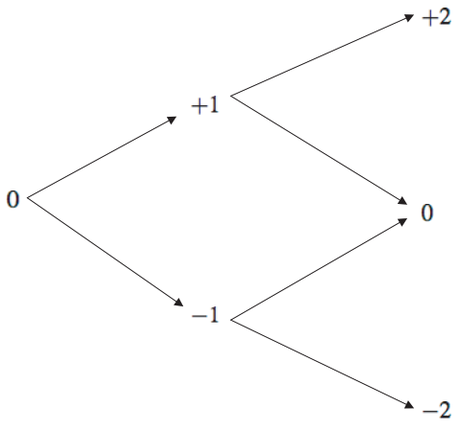

Figure 2 A tree for a single factor F_j, normalized.

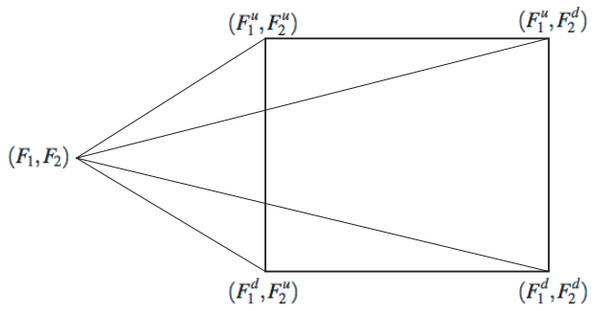

Figure 3 A typical node and branches on a tree for two factors ($K = 2$). If extended one more period, the lattice would have 9 nodes, i.e. $(N+1)^K$.

tree in each factor as the cumulative normalized factor deviations in the factor. This would result in the factor starting at value $F_j(0,0) = 0$ and then moving up $+1$ or down -1 each period. Hence, the normalized factor tree would be as portrayed in Figure 2.

The K-factor tree is the joint product of several trees. We never construct the joint tree in our program code; indeed, all we need to do is to develop the individual factor trees and then manage the joint tree carefully by accessing each individual factor tree as needed when running the option pricing algorithm. We call this a *virtual* joint tree approach and it will be elucidated later. Hence, substantial memory is saved in the computer implementation of the model. Since the memory required for each tree is $O(N)$, then the memory required across all trees is also $O(N)$. If the joint tree were actually constructed for the underlying factors, then the memory required would be $O(N^K)$. Thus, we proceed from memory usage of $O(2^{NK})$ (in the case of nonrecombining trees), to $O(N^K)$ (for recombining multi-factor trees), to $O(N)$ for recombining virtual multi-factor trees.

Keeping each tree separately is like having a virtual product tree, but we do not need to use up valuable memory resources in maintaining it. Conceptually however, we have a tree of order N^K. For example, if $K = 2$, then every node on the tree would have four branches, and would appear as in Figure 3.

Note however, that we do need to keep one tree of $O(N^K)$ for the actual derivative security that we are pricing, since that is derived from all the individual underlying factor trees. But we only keep individual trees for the factors, and no trees for the securities that are functions of the factors, as we describe next.

2.2 *Security trees*

Given we have constructed a tree in K factors, we then superimpose each security as a function of these factors to develop security trees on top of the factor tree. We assume that we have a function mapping the return r_i on security S_i to the movements in the factors. That is,

$$r_i = f(F_1, F_2, \ldots, F_K),$$

where the F_js are the normalized factors. We posit the following linear factor model for returns

$$\begin{aligned} r_i &= \delta_{i0} + \delta_{i1}F_1 + \cdots + \delta_{iK}F_K + e_i \\ &= \delta_{i0} + \sum_{j=1}^{K} \delta_{ij}F_j + e_i, \end{aligned}$$

where each F_j is orthogonalized. The coefficients $\delta_1 \cdots \delta_K$ are the shock terms to be applied on the lattice. These factor sensitivities may be determined by regression or other factor model technique

commonly used by portfolio managers. The δs allow us to map the risk for each security i to the factors. For instance, the variance of returns will be

$$\text{Var}(r_i) \equiv \sigma_i^2 = \sum_{j=1}^{K} \delta_{ij}^2 \sigma_j^2 + \sigma_{e_i}^2. \tag{1}$$

The exact application procedure is best clarified with an example.

Example: We present here a simplistic example using data. The pricing example deals with valuing an option on a single stock, and is deliberately simple so as to illustrate the approach. Of course, options on single stocks can be valued much faster by directly modeling the stock tree than a factor tree. But we note that once we understand how to embed a single stock on a factor tree, embedding multiple stocks is no different. Therefore, our example is with no loss of generality.

We downloaded the daily returns on IBM (from CRSP), and the Fama–French factors for 5 years, from January 2001 to December 2005. The mean daily return for the stock is 0.0173% and the standard deviation of returns is 1.852%. Hence, the annualized standard deviation of returns assuming 252 trading days in the year is 29.4%.

We use the most widely specified common three factors, excess market return (RMRF), small minus big stocks (SMB), and high minus low book-to-market (HML), respectively. In order to access the benefits of the virtual factor tree, the factors must be orthogonal. We orthogonalized the SMB factor by regressing it on the RMRF factor, and adding the residuals to the intercept of the regression. Likewise, we orthogonalized the HML factor by regressing it on RMRF and the orthogonalized SMB factor, and added the residuals to the intercept. We then regressed IBM's returns on these factors. The resulting regression is

$$r_{\text{IBM}} = 0.0412 + 1.1349\text{RMRF} - 0.1778\text{SMB} - 0.6391\text{HML}. \tag{2}$$

All coefficients are statistically significant, and the $R^2 = 0.50$. We also computed the annualized standard deviations for the orthogonalized factors, which happen to be $\sigma_{\text{RMRF}} = 0.1774$, $\sigma_{\text{SMB}} = 0.0868$, $\sigma_{\text{SMB}} = 0.0747$, and that of the residuals is $\sigma_i = 0.2083$. These values are then used to determine the cumulative return on IBM at any node on the lattice using the normalized factor trees. Suppose at node s at time $t = 2$, $F_{\text{RMRF}} = +2$, $F_{\text{SMB}} = 0$, $F_{\text{HML}} = -2$, and $F_i = -2$, then the cumulative return from the initial node on the tree to the current node will be

$$\begin{aligned} r_i(t,s) &= \sum_{j=1}^{K} (\delta_{ij}\sigma_{ij}\sqrt{h})F_j + \sigma_i\sqrt{h}F_i \\ &= \sum_{j=1}^{K} a_{ij}F_j + a_{i0}F_i, \end{aligned} \tag{3}$$

where we denote the idiosyncratic factor for the stock as F_i, and the time normalized loadings are denoted $\{a_{i1}, \ldots, a_{iK}, a_{i0}\}$. The time interval for the tree in the model is denoted h (in years). Note that we ignore the constant term δ_{i0} as the model will be risk-neutralized (discussed in the next subsection) to make the return of all assets risk free. For the same reason, the intercept in the factor regression is not required. More explicitly, for example, if $h = 1/12$, then based on Eq. (2), and the factor volatilities stated below that equation,

$$\begin{aligned} r_i = &(1.1349 \times 0.1774 \times \sqrt{1/12}) \times (+2) \\ &+ (-0.1778 \times 0.0868 \times \sqrt{1/12}) \times (0) \\ &+ (-0.6391 \times 0.0747 \times \sqrt{1/12}) \times (-2) \\ &+ (1 \times 0.2083 \times \sqrt{1/12}) \times (-2) \\ = &-0.0316. \end{aligned}$$

Hence, the stock price of IBM on the lattice at this node (after 2 months with an initial price of 90) will be: $S = S_0 e^R = 90e^{-0.0316} = 87.20$. In this way we can populate the four-dimensional lattice (three market factors and one idiosyncratic factor) for all

levels of stock prices and time on the tree. Given this, the run time of the algorithm on this virtual stock tree will be $O(N^{K+1})$.

2.3 *Obtaining the risk-neutral stock tree*

For a model with K factors, each of which is binomial, and an additional idiosyncratic factor, every node will branch to a set of 2^{K+1} nodes. Without loss of generality, we will assume that the movements up and down of all factors occur with equal probability. If the current security price is S, then any immediately ensuing node price will be of the form (dropping the subscript i to keep the notation simple):

$$S \exp\left[\pm a_1 \pm a_2 \pm \cdots \pm a_K \pm a_0\right],$$

where $a_j = \delta_j \sigma_j \sqrt{h}, \forall j$, and $a_0 = \sigma_i \sqrt{h}$. The notation above denotes that we are considering all possible binary combinations up and down of the factors. Each of the ensuing nodes occurs with equal probability, that is $1/(2^{K+1})$. For example if $K = 3$ (as we have in our example), and if the values of $\{F_1, F_2, F_3, F_i\} = \{+1, +1, +1, -1\}$, then the stock price on this branch will be $Se^{a_1+a_2+a_3-a_0}$ with probability 1/16. (As another example, Figure 4 shows the branching process for $K = 2$).

These dynamics capture the volatility of the process on the tree but do not ensure that the discounted (at risk free rate r_f) security price is a martingale under the assumed equiprobable probability measure. In order to transform the process to be risk-neutral we enhance the definition of the ensuing nodes with a drift term d, i.e. $S \exp\left[\pm a_1 \pm a_2 \pm \cdots \pm a_K \pm a_0 + d\right]$. In Figure 4, we present an explicit portrayal of nodes for a two-factor ($K = 2$) model. Note that the additional return drift d is added to all nodes irrespective of the other factors, hence it results in a mean shift with no impact on the variance of returns. The martingale condition at a node at time t requires that the $(t+h)$-forward price of the stock be equal to its expected value over all the ensuing nodes at the forward time $(t+h)$. The following

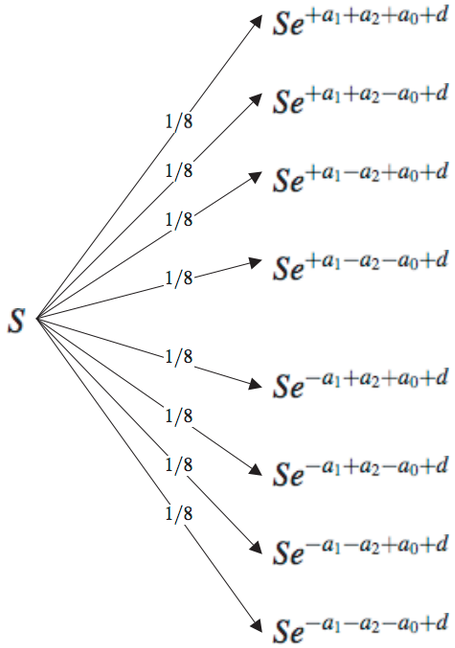

Figure 4 A typical node and branches on a tree for two factors ($K = 2$) plus an additional idiosyncratic factor specific to the security. An additional drift term is added to each node to ensure the process is risk-neutral. Each branch is equiprobable.

condition needs to be satisfied to make sure that the discounted security price is a martingale:

$$\begin{aligned} Se^{r_f h} &= \frac{1}{2^{K+1}} \sum_{\text{nodes}} S \exp\left[\pm a_1 \pm a_2 \right. \\ &\quad \left. \pm \cdots \pm a_K \pm a_0 + d\right] \\ &= \frac{S}{2^{K+1}} e^d \sum_{\text{nodes}} \exp\left[\pm a_1 \pm a_2 \right. \\ &\quad \left. \pm \cdots \pm a_K \pm a_0\right], \end{aligned}$$

where the term $[\pm a_1 \pm a_2 \pm \cdots \pm a_K \pm a_0]$ stands for combinations of $\{a_1, \ldots, a_0\}$ with positive and negative signs. The notation $\sum_{\text{nodes}}$ stands for a summation over all ensuing branches from a given node. Re-arranging the equation above, we get the solution for the risk-neutral drift term d as follows:

$$d = \ln\left[\frac{e^{r_f h}\, 2^{K+1}}{\sum_{\text{nodes}} \exp\left[\pm a_1 \pm a_2 \pm \cdots \pm a_K \pm a_0\right]}\right]. \quad (4)$$

In the Cox *et al.* (1979) approach, the tree is risk-neutralized by adjusting the probabilities on the up and down branches, rather than making a drift adjustment to the process itself. In our case, we prefer the drift adjustment, as it enables equiprobable branches, and also provides an analytic form for the drift which improves the speed of the model. Once the drift d is obtained the evolution of the tree is as depicted in Figure 4.

2.4 Pricing example

We continue with the example for IBM and price calls and puts using the Fama–French factor lattice and the estimated parameters from Subsection 2.2. The historical volatility of returns for IBM (annualized) is computed to be 0.294. Table 1 presents the inputs to the problem as well as the output values for varying time steps on the lattice.

Table 1 Option prices from the multi-factor model.

Factor	j	δ_j	σ_j (annualized)
Factor loadings and volatilities			
RMRF	1	1.1349	0.1774
SMB	2	−0.1778	0.0868
HML	3	−0.6391	0.0747
Idiosyncratic	0	1	0.2083

N	Call price	Put price	Run time (seconds)	Memory (MB)
10	12.2036	8.6745	0	0.61
20	12.1893	8.6605	0	15.58
30	12.1834	8.6545	3	109.21
40	12.1805	8.6513	8	441.96
50	12.1787	8.6494	26	1316.17
BS	12.1710	8.6423	0	—

BS: Black–Scholes

The table contains the input values and call and put option prices as N, the number of time steps is varied. The maturity of the options is $T = 1$ year, and the risk-free rate is $r = 0.04$, dividends are set to zero. The initial stock price is set to \$90 and the strike is at-the-money. The first panel in the table shows the factor loadings of IBM's return on the Fama–french factors, as well as the factor volatilities. The model was run on a Mac with a 2 Ghz Intel Core Duo processor and 1.5 GB 667 Mhz DDR2 SDRAM. Programs are written in the C programming language.

As a check, the value of the options was also computed using Black–Scholes and a volatility equal to the historical volatility of the stock. We also note that the historical volatility of the stock may be recovered from the factor loadings and volatilities, using Eq. (1) and a simple check will show that the variance is indeed equal to that directly computed from the time series of stock returns. The annual variance of the stock may be computed from the factor variance Eq. (1) given previously as follows:

$$\begin{aligned}\sigma_i^2 &= \sum_{j=1}^{K} \delta_j^2 \sigma_j^2 + \sigma_{e_i}^2 \\ &= 1.1349^2(0.0315) + 0.1778^2(0.0075) \\ &\quad + 0.6391^2(0.0056) + 0.0434 \\ &= 0.0864.\end{aligned}$$

Taking the square-root we get the annualized volatility as 29.4%. Note that $h = 1$ in the example above. Since the factors are orthogonal, there are no covariance terms. The annualized volatility is exactly that computed earlier in Section 2.2.

From Table 1, it is seen that as the number of time steps N increases, the option prices converge to the Black–Scholes value. In addition, the convergence is monotone, which suggests that even with a lower N, we may be able to extrapolate the value of the option to that for much larger N using any standard extrapolation scheme (such as Richardson (1910)'s extrapolation for example; see also Richardon (1927)). We also note the highly efficient run times achieved despite the massive tree sizes involved. For N periods and K factors, the number of nodes is $\sum_{i=1}^{N+1} i^K$. For $N = 50$, we compute a total of 1,758,276 nodes in under half a minute.

3 Computer implementation of the factor lattice model

On modern computer hardware, the one and two factor tree models have very short runtimes. For example, on a recent MacBook Pro with a 2.53 GHz Intel Core 2 Duo processor, the single factor model with $N = 1000$ runs in 3 milliseconds and the two factor model with $N = 1000$ runs in 4.4 seconds. However, because of the exponential scaling of the algorithm with K, the runtimes increase dramatically for models with 3 or more factors ($K > 2$). It is thus desirable to explore the possibility of implementing these models on parallel hardware such as multicore processors, graphics processors (GPUs) or clusters.

In this section we begin to explore the performance characteristics of the one and two factor tree models that are relevant in evaluating potential parallelization strategies. For these explorations, we have written a series of C programs that clarify the relative importance of floating point operations and memory bandwidth in these algorithms. All of these programs were run on an 2.53 GHz Intel Core 2 Duo processor with 6 MB of L2 cache.

Our experiments show that on current processors both the one and two factor tree models are not CPU bound, but rather, are memory bandwidth limited. To show this we have run two versions of the $K = 1$ and $K = 2$ models (see Appendix A). The first version does the full recursion on the K dimensional option tree. Because this version operates on the full tree, there are many memory operations. The second version does the same number of floating point operations as the first version, but no memory operations. If the algorithm were CPU bound, the runtimes of these two versions would be nearly identical. Instead, we find that the first version takes about twice as long to run as the second (see Appendix A). This shows that a significant amount of time is spent with the CPU waiting for the memory subsystem to provide the data needed for subsequent floating point operations.

Another way of seeing that these algorithms are memory bandwidth limited is to look at the number of floating point operations per memory access. For the general K-factor model, each node requires 2^K floating point operations (1 multiply and $2^K - 1$ adds) and 2^K (64 bit) memory accesses. Thus, there is exactly 1 floating point operation performed per memory access. This result is independent of the number of factors K or periods N and again shows that the bottleneck in these factor tree algorithms is the memory bandwidth.

How is this relevant for efforts to parallelize these algorithms using multicore processors? Because current multicore processors share the overall memory bandwidth, there will be little benefit to using multithreading across processor cores for these algorithms. However, this does suggest that it is worth pursuing parallel versions of these models on hardware that has higher total memory bandwidth, such as GPUs and clusters.

4 Options on multiple assets

The concepts from the prior sections make it easy to extend the model to pricing options on multiple assets. We consider two cases here. One, is the pricing of options on a portfolio. This may be used to model options on an index, or to model credit portfolios. These are essentially options on a weighted sum of the assets. Two, we consider options whose payoffs are a function of asset ratios or products.

4.1 Options on portfolios

Given a portfolio comprising $M \gg K$ assets, the factor lattice makes it easy to value options on the portfolio. Since each asset in the portfolio may be embedded on the factor tree, each node on the tree will have M assets implicitly, though implementation may create M virtual asset trees as seen

in the example earlier. Given portfolio weights $w_i, i = 1, \ldots, M$ in securities S_i, the current value of the portfolio at any node is simply

$$\sum_{i=1}^{M} w_i S_i, \quad \text{where} \sum_{i=1}^{M} w_i = 1.$$

This approach will be especially useful for arbitrage traders attempting to model S&P500 options using the entire set of stocks (a bottom-up approach) versus modeling the stochastic process for the S&P500 index directly (a top-down approach). The chosen factor structure may be either the Fama–French type of model we have already examined, or developed from a principal component decomposition of the time series of returns. In the latter case, the components are orthogonal by construction, and the principal components analysis (PCA) delivers the component time series and also the factor loadings. This makes implementation direct and simple. The value of an option on a portfolio depends not only on the volatility of each asset in the portfolio, but also on the correlations amongst them. Accounting for correlations would be difficult, but with the multi-factor approach this is seamlessly embedded into the method.

Being able to model options on portfolios is useful because they are not analytically tractable. Options on sums of lognormal variables pose problems as the underlying is no longer lognormal and does not fall into the standard Black–Scholes paradigm (see Curran (1994) for one solution approach).

4.2 Options on the ratio of two assets

Options may also be written on payoffs where the distinct returns of each asset are retained, not amalgamated into one common portfolio value as in the previous subsection. We illustrate the handling of these options by examining an option on the ratio of two assets. These options are known as outperformance options, or options to exchange one asset for another. The risk-neutral returns on two assets may be written as

$$r_1 = \sum_{j=1}^{K} \delta_{1j} + e_1 + d_1$$

$$r_2 = \sum_{j=1}^{K} \delta_{2j} + e_2 + d_2,$$

where $d_i, i = \{1, 2\}$ is the risk-neutral drift on the two assets. In a single period of interval h the growth in these assets will be

$$S_1(h) = S_1(0)e^{r_1 h}, \quad S_2(h) = S_2(0)e^{r_2 h}.$$

Consider a call on the ratio of these two assets at strike price X. The payoff function for this option is $\max[0, S_1/S_2 - X]$. We may write the ratio as

$$\frac{S_1}{S_2} = \exp\left[\sum_{j=1}^{K} \underbrace{(\delta_{1j} - \delta_{2j})}_{\delta_j} F_j + \underbrace{(e_1 - e_2)}_{e} + \underbrace{(d_1 - d_2)}_{d}\right]$$

$$= \exp\left[\sum_{j=1}^{K} \delta_j F_j + e + d\right].$$

We note that $E(e) = 0$ and $E(e_1 e_2) = 0$. The variance of e is $\sigma_e^2 = \sigma_1^2 + \sigma_2^2$.

Therefore, the problem reduces to one where we effectively have a single security with factor loadings δ_j and idiosyncratic variance σ_e^2. Implementation follows as usual.

5 Summary

Pricing options on multiple assets requires the modeling of multivariate stochastic processes. As the number of assets grows, the modeling of option pricing trees in the usual manner becomes computationally infeasible. The standard solution has

been to resort to Monte Carlo simulation. However, pricing American options via simulation poses many tricky issues and has not been found to be ideal in many situations. In this paper, we present an approach where a medium-dimensional factor tree may be used over any dimension in assets to achieve parsimonious and computationally efficient option pricing. We are able to compute option prices on trees that have more than 1.7 million nodes in under a half minute. We demonstrate how these tree models are calibrated, and begin to explore the issues that arise when the models are run on multi-core or multi-CPU hardware.

To summarize, we consider any claim with value $C(S_1, \ldots, S_M, t)$. We note that the same scheme could be implemented by finite-differencing as well. A lattice method can always be mapped into an explicit finite-difference scheme on a log-spaced grid. After constructing the PDE satisfied by the claim, and rotating the co-ordinate system to get rid of the cross-derivative term, we do a log change of variables, discretize explicitly, then we have an orthogonal, recombining tree. This requires $O(N^K)$ storage, with N being the number of nodes in each factor direction, and total work of $O(N^{K+1})$. We do not, of course, store all the nodes in the conceptual tree, only at each time-slice.

A deficiency of the approach as it currently stands is that imposing orthogonality is a tough restriction when correlations are state-dependent. Extending these approaches using a multivariate GARCH approach or using dyamic conditional correlations (DCC) is an important extension.

What next? Trading options on portfolios of assets has not become prevalent in practice for the lack of pricing algorithms such as the one presented in this paper. The arena in which the greatest promise exists is the valuation of basket options in the credit arena. Therefore extending these multi-factor tree models to embedding default is the next research step.

Acknowledgment

We are grateful for discussions of the model with Terry Marsh and Paul Pfleiderer. Jacob Sisk provided several neat ideas on improving the efficiency of the program code. Das also acknowledges the support of the Dean Witter Foundation and funding from a Breetwor Fellowship.

Appendix A. Performance issues in one and two factor models

In this Appendix, we present the C program code that we have used to study the relative importance of floating point operations and memory bandwidth in the one and two factor models. When this code was run on a 2.53 GHz Intel Core 2 Duo processor with 6 MB of L2 cache, we got the following results:

```
n = 10000
1 factor, full calculation time in seconds = 0.344381
1 factor, CPU bound calculation time in seconds = 0.159952
Ratio (cpu/full) = 0.464462

n = 1000
2 factor, full calculation time in seconds = 4.396845
2 factor, CPU bound calculation time in seconds = 2.302681
Ratio (cpu/full) = 0.523712
```

Thus, in the 1 factor model with $N = 10000$ periods the full algorithm took 2.15 times as long as the CPU bound version. Likewise, the 2 factor model with $N = 1000$ took 1.91 times as long as its CPU bound equivalent. We have repeated these tests for different numbers of periods and found similar numbers (the ratio is always approximately 2). We thus conclude that these algorithms are memory bandwidth limited for all K and all N. This is consistent with our calculation of the number of floating point operations per memory

A.1 C Program Code

```
// Das and Granger (2008)
// Explore the relative importance of floating point operations and
// memory bandwidth on 1 and 2 factor tree models. This code does the full
// recursion on the option tree, but with made up initial values on the
// leaves of the tree for simplicity.
//
// Compile with: gcc -o benchmark benchmark.c -lm
//
// Output on MacBook Pro (2.53 GHz Intel Core 2 Duo):
// n = 10000
// 1 factor, full calculation time in seconds = 3.443810e-01
// 1 factor, CPU bound calculation time in seconds = 1.599520e-01
// Ratio (cpu/full) = 0.464462
//
// n = 1000
// 2 factor, full calculation time in seconds = 4.396845e+00
// 2 factor, CPU bound calculation time in seconds = 2.302681e+00
// Ratio (cpu/full) = 0.523712
#include <stdio.h> #include <math.h> #include <stdlib.h> #include
<time.h>

// Recursion for 1 factor model
double ktree1full(double *opt, int n, double a) {
    int i, j;
    for (i=0;i<n;i++)
        for (j=0;j<n-i;j++)
            opt[j] = a*(opt[j] + opt[j+1]);
    return opt[0];
}

// Fake, CPU bound recursion for 1 factor model
// Same # of operations as ktree1full, but no memory transfer
double ktree1cpu(int n, double a) {
    int i, j;
    double p, q, r;
    p = 1.0; q = 2.0; // Arbitrary
    for (i=0;i<n;i++)
        for (j=0;j<n-i;j++)
            r = a*(p + q); // Same ops, no memory transfer
    return r;
}
```

```
// Recursion for 2 factor model
double ktree2full(double *opt, int n, double a) {
    int i, j, k;
    int base, offset;
    for (i=0;i<n;i++)
        for (j=0;j<n-i;j++)
            for (k=0;k<n-i;k++) {
                base = j*(n+1)+k;
                offset = (j+1)*(n+1)+k;
                opt[base] = a*(opt[base]+opt[offset]+ \
                    opt[base+1]+opt[offset+1]);
            }
    return opt[0];
}

// Fake, CPU bound recursion for 2 factor model
// Same # of operations as ktree2full, no memory transfer
double ktree2cpu(int n, double a) {
    int i, j, k;
    int base, offset;
    double p, q, r, s, t;
    p = 1.0; q = 2.0; s = 3.0; t = 4.0; // Arbitrary
    for (i=0;i<n;i++)
        for (j=0;j<n-i;j++)
            for (k=0;k<n-i;k++) {
                base = j*(n+1)+k;
                offset = (j+1)*(n+1)+k;
                r = a*(p+q+s+t); // Same ops, no memory transfer
            }
    return r;
}

// Run benchmarks for 1 factor model (n periods)
void benchmark1(int n) {
    clock_t start, end;
    double deltat1, deltat2;
    int i;
    double *opt;

    opt = (double *)malloc(sizeof(double)*(n+1));
```

```
    // Fake terminal nodes of the option tree
    for (i=0;i<n+1;i++) {
        opt[i] = 1.0;
    }

    printf("\nn = %i\n", n);

    start = clock();
    ktree1full(opt, n, 0.5);
    end = clock();
    deltat1 = (double)(end-start)/CLOCKS_PER_SEC;
    printf("1 factor, full calculation time in seconds = %e \n",
        deltat1);

    start = clock();
    ktree1cpu(n, 0.5);
    end = clock();
    deltat2 = (double)(end-start)/CLOCKS_PER_SEC;
    printf("1 factor, CPU bound calculation time in seconds = %e \n",
        deltat2);
    printf("Ratio (cpu/full) = %f\n", deltat2/deltat1);

    free(opt);
}

// Run benchmarks for 2 factor model (n periods)
void benchmark2(int n) {
    clock_t start, end;
    double deltat1, deltat2;
    int i;
    double *opt;

    opt = (double *)malloc(sizeof(double)*((n+1)*(n+1)));

    // Fake terminal nodes of the option tree
    for (i=0;i<(n+1)*(n+1)-1;i++) {
        opt[i] = 1.0;
    }
    printf("\nn = %i\n", n);

    start = clock();
    ktree2full(opt, n, 0.5);
    end = clock();
```

```
    deltat1 = (double)(end-start)/CLOCKS_PER_SEC;
    printf("2 factor, full calculation time in seconds = %e \n",
        deltat1);

    start = clock();
    ktree2cpu(n, 0.5);
    end = clock();
    deltat2 = (double)(end-start)/CLOCKS_PER_SEC;
    printf("2 factor, CPU bound calculation time in seconds = %e \n",
        deltat2);
    printf("Ratio (cpu/full) = %f\n", deltat2/deltat1);

    free(opt);
}

int main() {
    benchmark1(1000);
    benchmark2(1000);
    return 0;
}
```

accesses, which is unity regardless of K and N (see Section 3).

References

Black, F. and Scholes, M. (1973). "The Pricing of Options and Corporate Liabilities." *Journal of Political Economy*, **81**, 637–654.

Cox, J., Ross, S., and Rubinstein, M. (1979). "Option Pricing: A Simplified Approach." *Journal of Financial Economics* 7, 229–263.

Curran, M. (1994). "Valuing Asian and Portfolio Options by Conditioning on the Geometric Mean Price." *Management Science* **40**, 1705–1711.

Nelson, D. and Ramaswamy, K. (1990). "Simple Binomial Processes as Diffusion Approximations in Financial Models." *Review of Financial Studies* **3**, 393–430.

Nichols, B., Buttlar, D. and Farrell, J. (1996). "Pthreads Programming: A POSIX Standard for Better Multiprocessing," O'Reilly Publishing, California.

Pthreads Programming: A POSIX Standard for Better Multiprocessing. O'Reilly Publishing.

Richardson, L.F. (1910). "The Approximate Arithmetical Solution by Finite Differences of Physical Problems Including Differential Equations, with an Application to the Stresses in a Masonry Dam." *Philosophical Transactions of the Royal Society of London Series A* **210**, 307–357.

Richardson, L.F. (1927). "The Deferred Approach to the Limit." *Philosophical Transactions of the Royal Society of London Series A* **226**, 299–349.

Keywords: High-dimension; multi-factor trees; multi-threading