A General Computational Method for Calibration Based on Differential Trees

Yuh-Dauh Lyuu
Associate Professor
Dept. Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan
Tel: 886-2-2362-5336 ext. 429
Fax: 886-2-2362-8167
lyuu@csie.ntu.edu.tw
http://www.csie.ntu.edu.tw/~lyuu
July 23, 1999

Abstract

This paper presents a general computational paradigm for model calibration called the differential tree method. The idea is very simple, and the method is particularly applicable when the model under consideration has a tree (lattice) structure. We illustrate its wide applicability with three canonical problems: no-arbitrage interest rate model calibration, spread and option-adjusted spread of non-benchmark bonds, and implied volatility of American-style options. Comprehensive computer experiments show the differential tree method to be highly efficient in all three case studies.

1 Introduction

Many computational problems in derivative pricing involve trees. The binomial option pricing model and various discrete-time interest rate models are familiar examples.
Computation on such structures conceptually proceeds via backward induction on the tree with local calculation at each node taking inputs from its successor nodes. Efficient algorithms are generally obtainable for pricing purposes if the tree recombines.

Our main concern, model calibration, is the inverse of the pricing problem: Given price information, it finds parameter values on the tree that generate model prices consistent with the data. Calibration, therefore, can be perceived as a root-finding problem. In this paper, we propose a general iterative procedure, the differential tree method (Lyuu [1995]), for the calibration of models based on trees.

The basic idea behind the differential tree method is simple. It takes advantage of the inductive structure of the tree to calculate not only the model price but also its derivatives with respect to the parameters whose values we desire. This extra effort adds only a small amount of overhead to the overall computation, yet it pays off immensely in terms of faster convergence and clean software design.

We illustrate the wide-ranging applications of the method with three canonical problems: calibration of no-arbitrage interest rate models (in this paper, the Black-Derman-Toy model [1990]), the calculation of spread and option-adjusted spread (OAS) of non-benchmark bonds, and the computation of the implied volatility of American-style options (all listed stock options in the U.S. are American, for example). In each case, we obtain efficient algorithms in terms of speed, computer memory, and convergence rates. This claim will be substantiated with extensive computer experiments.

This paper is organized as follows. In Section 2, the differential tree method is presented in its full generality. In Section 3, the method is applied to the calibration of the Black-Derman-Toy model. Treatments of spread and option-adjusted spread then follow suit in Section 4. Section 5 applies the method to calibrating the Cox-Ross-Rubinstein binomial option pricing model with American option prices. In each of the three cases, extensive computer experiments are conducted to investigate the method’s performance and range of applicability. Section 6 ends the paper with conclusions.

2 The Differential Tree Method

Let \( P(x) \) be a function defined by evaluation via backward induction on a tree. In our context, \( P(x) \) denotes price. Such a framework captures derivative pricing un-
der discrete-time models. Its inverse function, the solution to $P(x) = p$ given $p$, defines the calibration problem. Pricing and calibration are therefore dual problems. In general, $P$ and $x$ can be vectored; for instance, $P(x)$ can be the discount function. To make matters simple, this paper focuses on binomial trees; generalization to multinomial trees is straightforward.

Backward induction on a tree can be seen as a cascading of function evaluations. Consider any node A with two successor nodes named B and C. In the chain of computations, node A computes a certain function $f_A(a, b, c)$ in which $a$ is locally computable, whereas $b$ and $c$ are supplied by similar computation at nodes B and C, respectively. Node A in turn will feed its two predecessor nodes with $f_A$, and so on down the tree. Function $P$ is simply the final result of these function evaluations at the root of the tree. If $a$, $b$, and $c$ are functions of some parameter $x$, then $f_A$ computed at A is also a function of $x$, and this holds inductively for $P$ as well. See Figure 1's top row for illustration.

To solve for $x$, one needs a fast root-finding scheme. The Newton-Raphson method is one such scheme. This popular iterative method uses a function and its derivative to improve upon the current approximation to the root until the error lies within some specified bounds. Being quadratically convergent, the Newton-Raphson method doubles the precision in each iteration.

To employ the Newton-Raphson method to solve $P(x) = p$, one needs $P'(x)$. In principle, we can write down $P(x)$ explicitly and then differentiate it term by term. This idea becomes infeasible, however, when the tree is large. Fortunately, the differential tree method takes advantage of the tree by computing $P'(x)$ along with $P(x)$ inductively, as described below.

The node A mentioned before computes $df_A/dx$ in addition to $f_A$ and then forwards both $f_A$ and $df_A/dx$ to its two predecessor nodes. This extra calculation is straightforward: With $db/dx$ and $dc/dx$ from nodes B and C, respectively (besides $b(x)$ and $c(x)$ generated for the computation of $P(x)$, of course), and with $da/dx$ available locally, we can calculate $df_A/dx$ thus,

$$\frac{df_A(a(x), b(x), c(x))}{dx} = \frac{\partial f_A}{\partial a} \frac{da(x)}{dx} + \frac{\partial f_A}{\partial b} \frac{db(x)}{dx} + \frac{\partial f_A}{\partial c} \frac{dc(x)}{dx}.$$ 

Backward induction now continues from node A to its two predecessor nodes and, inductively, toward the root, which produces $P(x)$ and $P'(x)$. The Newton-Raphson method then finds a new approximation to the root before the next iteration begins. This process repeats itself until the desired accuracy is achieved. The overall scheme is
simple and adds little overhead to the original backward induction algorithm. We will call a tree with these derivative values a differential tree (see Figure 1 for illustration).

In general, $x$ can be a vector. Take the two-dimensional case as an example, $\mathbf{x} \equiv (s, t)$. Then we need to calculate (Hildebrand [1976])

$$
\frac{\partial f_A(a(\mathbf{x}), b(\mathbf{x}), c(\mathbf{x}))}{\partial s} = \frac{\partial f_A \partial a(\mathbf{x})}{\partial a} \frac{\partial a}{\partial s} + \frac{\partial f_A \partial b(\mathbf{x})}{\partial b} \frac{\partial b}{\partial s} + \frac{\partial f_A \partial c(\mathbf{x})}{\partial c} \frac{\partial c}{\partial s}
$$

$$
\frac{\partial f_A(a(\mathbf{x}), b(\mathbf{x}), c(\mathbf{x}))}{\partial t} = \frac{\partial f_A \partial a(\mathbf{x})}{\partial a} \frac{\partial a}{\partial t} + \frac{\partial f_A \partial b(\mathbf{x})}{\partial b} \frac{\partial b}{\partial t} + \frac{\partial f_A \partial c(\mathbf{x})}{\partial c} \frac{\partial c}{\partial t}
$$

Again, inductively, $\partial b(\mathbf{x})/\partial s$, $\partial c(\mathbf{x})/\partial s$, $\partial b(\mathbf{x})/\partial t$, and $\partial c(\mathbf{x})/\partial t$ are available from nodes B and C, and the same computational principle applies. In the general case where $\mathbf{x}$ is $n$-dimensional, the differential tree has to calculate $n$ partial derivatives.

Let us illustrate the differential tree idea with an example: implied volatility under the binomial option model. Consider European puts. The standard CRR binomial pricing model implies the following backward induction formulation (Hull [1997]),

$$
f_A(\sigma, b, c) = \begin{cases} 
\frac{p^{b+1}(1-p)c}{e^{r\Delta t}} & \text{if } A \text{ is an internal node} \\
\max(X - S u^j, 0) & \text{if } A \text{ is a terminal node}
\end{cases}
$$

Here,

$$
p = \frac{e^{r\Delta t} - d}{u - d}
$$

is the probability of an up move, $r$ is the interest rate, $S$ is the current stock price, $u \equiv e^{\sqrt{\Delta t}}$, $d = 1/u$, $S u^j$ is the stock price at node A (so the number of up moves exceeds that of down moves by $j$ in order to reach A), $X$ is the strike price, and a time period has duration $\Delta t$. To apply the differential tree method, simply compute the following derivative as well,

$$
\frac{df_A}{d\sigma} = \begin{cases} 
\frac{p^{b+1}(1-p)c}{e^{r\Delta t}} & \text{if } A \text{ is an internal node} \\
0 & \text{if } A \text{ is a terminal node but not exercised} \\
-j\sqrt{\Delta t}S u^j & \text{if } A \text{ is a terminal node but exercised}
\end{cases}
$$

The sections to follow deal with three canonical calibration problems: interest rate model calibration, spread and option-adjusted spread of non-benchmark bonds, and implied volatility of American options (described above). All our programs are written in C. Following Patterson and Hennessy [1994], we take the position that the running time is the only valid performance measure. Other measures such as the number of iterations and/or the number of time partitions needed for convergence
will be mentioned but not given dominating emphasis since they lead to misleading conclusions (Lyuu [1998]).

Before closing this section, a few words about the computer setup and performance reports. All timing information will be in seconds. A variety of computing platforms were employed, including the Intel Pentium MMX processor, a 75MHz Sun SPARCstation 20, and a 200MHz Sun ULTRASPARC. (The 300MHz Sun ULTRA- SPARC IIi with a good C compiler from Sun was capable of cutting the reported running times based on the Sun SPARCstation 20 by about 90%.) Throughout this paper, \( n \) denotes the depth of the tree, i.e., the total number of time periods.

3 Calibration of Interest Rate Models: the Black-Derman-Toy Model

Our first application is concerned with the calibration of interest rate models. Calibration aims to set up a short rate tree consistent with the observed term structures, specifically the yields and, in some models, yield volatilities of zero-coupon bonds of all maturities as well. Although the differential tree method clearly applies to other models, we pick the Black-Derman-Toy (BDT) model for two reasons: analytical intractability and popularity. Calibration of the BDT model needs to fit both the bond prices and their yield volatilities.

For the BDT binomial tree, the logarithm of the future short rate obeys the binomial distribution. The limiting distribution for the short rate is hence lognormal. The BDT binomial tree of possible short rates for each future period is constructed as follows. Each short rate is followed by two short rates for the following time period of length \( \Delta t \). In Figure 2, node A coincides with the start of period \( j \) during which short rate \( r \) is in effect. At the conclusion of period \( j \), a new short rate goes into effect for period \( j + 1 \). This may take one of two possible values: \( r_l \), the “low” short rate outcome (up move, for the bond price), shown at node B, or \( r_h \), the “high” short rate outcome (down move, for the bond price), at node C. Each rate has a fifty percent chance of occurring in a risk-neutral economy. Percent volatility of the short rate, \( \Delta r / r \), is

\[
\sigma = \frac{1}{2\sqrt{\Delta t}} \times \ln(r_h/r_l)
\]
when the short rate follows a lognormal process in the limit. Note that
\[ \frac{r_h}{r_l} = e^{2\sigma \sqrt{\Delta t}}. \]  
(1)

To nail down the values of \( r_h \) and \( r_l \), we need information from the current term structures to establish the relation between \( r \) and its two successors.

As the binomial process unfolds, we make sure that the paths recombine. In general, there are \( j \) possible rates applicable for period \( j \). They are
\[ r_j, r_jv_j, r_jv_j^2, \ldots, r_jv_j^{j-1}, \]
where
\[ v_j = e^{2\sigma_j \sqrt{\Delta t}} \]
is the multiplicative ratio for the rates in period \( j \). The volatilities \( \sigma_j \) above are indexed by \( j \) because volatility is a function of time. We shall call \( r_j \) the baseline rate. Figure 3 depicts the tree structure.

With the abstract process in place, concrete numbers are needed to set it in motion: They are the annualized rates of return associated with the various riskless bonds making up the benchmark yield curve and their volatilities. For example, the on-the-run yield curve may be used as the benchmark curve. The term structure of volatilities (or volatility structure) may be estimated either from historical data or from interest rate derivatives. The binomial tree should be consistent with both term structures. From now on, for economy of expression, all numbers are measured by the period \( (\Delta t = 1) \) unless otherwise stated.

To store the whole tree, the space requirement would be proportional to \( n^2 \), which is prohibitively expensive for a large tree. For instance, modeling daily interest rate movements for thirty years amounts to keeping an array of roughly \((30 \times 365)^2/2 \approx 6 \times 10^7\) double-precision floating-point numbers. If each number takes up eight bytes, storage alone would consume nearly half a gigabyte! Fortunately, memory requirement can be made minimal. This is because only the baseline rates \( r_i \)'s and the multiplicative ratios \( v_i \)'s need to be stored in computer memory since the rest of the tree are known functions of them. The storage requirement is hence down to \( O(n) \).

Models based on such a paradigm are usually called *no-arbitrage models*, to which the BDT model belongs.

### 3.1 The differential tree method

To derive the actual values for $r_h$ and $r_l$, a naive approach starts with the implied one-period forward rates and their volatilities. It then dictates that the expected short rate equal the implied forward rate for the same period; in other words, the *unbiased expectations theory* holds. This method corresponds to the scheme “with no iteration” in Bjerksund and Stensland [1996]. However, Lyuu [1995] has proved that this approach overestimates the prices of benchmark securities as long as the yield volatilities are positive, independent of whether the volatility structure is fit.

We now present a correct approach using the differential tree method. Recall that there are $j$ possible short rates for period $j$. According to the binomial interest rate model, the $j$ rates are $r_j, r_jv_j, r_jv_j^2, \ldots, r_jv_j^{j-1}$, where $r_j$ is the baseline rate. Suppose the price of the $j$-period zero moves up to $P_u$ and down to $P_d$ one period from today. Obviously, $P_u$ and $P_d$ are functions of $r_j$ and $v_j$. In a risk-neutral world, it must hold that

$$
\frac{1}{2}P_u + \frac{1}{2}P_d = \frac{1}{(1 + y)^j},
$$

where $y$ is today’s yield of the $j$-period zero, which is known.

Viewed from now, the future $(j - 1)$-period yield at the beginning of period two is uncertain. Let $y_u$ represent the $(j - 1)$-period yield to maturity at the “up” node, $y_d$ the $(j - 1)$-period yield to maturity at the “down” node, and $\kappa^2$ the variance viewed from now of the $(j - 1)$-period yield to maturity. The variance of the $(j - 1)$-period yield is

$$
\kappa^2 = p(1 - p) \ln^2(y_u/y_d).
$$

Hence, for $p = 1/2$,

$$
\kappa = (1/2) \ln(y_u/y_d).
$$

As the bonds are zero-coupon bonds,

$$
y_u = P_u^{\frac{-1}{j-1}} - 1 \quad \text{and} \quad y_d = P_d^{\frac{-1}{j-1}} - 1.
$$

Substitute to get

$$
\kappa = (1/2) \ln \left( \frac{P_u^{\frac{-1}{j-1}} - 1}{P_d^{\frac{-1}{j-1}} - 1} \right). \quad (3)
$$
Finally, rearrange (2) and (3) as simultaneous equations,

\[ f(P_u, P_d) \equiv P_u + P_d - \frac{2(1 + r_1)}{(1 + y)^j} = 0 \]
\[ g(P_u, P_d) \equiv P_u^{-\frac{1}{j-1}} - 1 - e^{2k} \left( P_d^{-\frac{1}{j-1}} - 1 \right) = 0 \]

Since \( P_u \) and \( P_d \) are functions of \( r_j \) and \( v_j \), \( f(P_u, P_d) \) and \( g(P_u, P_d) \) are functions of \( r_j \) and \( v_j \) as well, denoted \( F(r_j, v_j) \) and \( G(r_j, v_j) \), respectively. For economy of expression, we use \( f(r, v) \) instead of \( f(r_j, v_j) \), \( g(r, v) \) instead of \( g(r_j, v_j) \), and so on. Given the \( k \)th approximation \( (r(k), v(k)) \), the Newton-Raphson method says the \( (k + 1) \)st approximation \( (r(k + 1), v(k + 1)) \) satisfies

\[
\begin{bmatrix}
\frac{\partial F(r(k), v(k))}{\partial r} & \frac{\partial F(r(k), v(k))}{\partial v} \\
\frac{\partial G(r(k), v(k))}{\partial r} & \frac{\partial G(r(k), v(k))}{\partial v}
\end{bmatrix}
\begin{bmatrix}
\Delta r(k + 1) \\
\Delta v(k + 1)
\end{bmatrix}
= -
\begin{bmatrix}
F(r(k), v(k)) \\
G(r(k), v(k))
\end{bmatrix}
\]

where \( \Delta r(k + 1) \equiv r(k + 1) - r(k) \) and \( \Delta v(k + 1) \equiv v(k + 1) - v(k) \).

We still need \( \partial F/\partial r, \partial F/\partial v, \partial G/\partial r, \) and \( \partial G/\partial v \) in the matrix above to solve for \( (r(k + 1), v(k + 1)) \). Obviously,

\[
\frac{\partial F}{\partial r} = \frac{\partial P_u}{\partial r} + \frac{\partial P_d}{\partial r}
\]
\[
\frac{\partial F}{\partial v} = \frac{\partial P_u}{\partial v} + \frac{\partial P_d}{\partial v}
\]

and, by the chain rule,

\[
\frac{\partial G}{\partial r} = \frac{\partial g}{\partial P_u} \frac{\partial P_u}{\partial r} + \frac{\partial g}{\partial P_d} \frac{\partial P_d}{\partial r}
\]
\[
\frac{\partial G}{\partial v} = \frac{\partial g}{\partial P_u} \frac{\partial P_u}{\partial v} + \frac{\partial g}{\partial P_d} \frac{\partial P_d}{\partial v}
\]

In the above four equations, the common terms requiring evaluations are \( \partial P_u/\partial r \), \( \partial P_d/\partial r \), \( \partial P_u/\partial v \), \( \partial P_d/\partial v \), \( \partial g/\partial P_u \), and \( \partial g/\partial P_d \). The differential tree method can compute them as follows. Working backward, the method finds \( P_u, P_d, \partial P_u/\partial r, \partial P_d/\partial r, \partial P_u/\partial v, \text{ and } \partial P_d/\partial v \). As for the two remaining terms \( \partial g/\partial P_u \) and \( \partial g/\partial P_d \), they can be computed directly from the definition of \( g \) as

\[
\frac{\partial g}{\partial P_u} = -\frac{1}{j-1} P_u^{-\frac{1}{j-1}}
\]
\[
\frac{\partial g}{\partial P_d} = e^{2k} \frac{1}{j-1} P_d^{-\frac{1}{j-1}}
\]

8
Thus we conclude the matrix can be set up.

The standard differential tree method leads to a backward induction implementation. Since each pass across the tree takes \( O(j^2) \) time to obtain the \( j \)th baseline rate and the \( j \)th multiplicative ratio, the total time is \( O(n^3) \), assuming the Newton-Raphson method takes a constant number of iterations to get to the desired accuracy. This assumption turns out to be reasonable in practice, as we will see later.

### 3.2 Forward induction

To reduce the computation time to \( O(n^2) \), forward induction can be employed (Jamshidian [1991]). This scheme inductively figures out, by moving forward in time, how much \$1 at a node contributes to the total price (see Figure 4). This number is called the state price since it is the price of a claim that pays \$1 at that particular state (node) and zero elsewhere. We call a tree with these state prices a binomial state price tree. As before, there is no need to explicitly store the whole tree.

Let us be more precise about the mechanism. Suppose we are at the end of period \( j \). So there are \( j + 1 \) nodes. Let the baseline rate for period \( j \) be \( r_j = r \), the multiplicative ratio be \( v_j = v \), and \( P_1, \ldots, P_j \) be the state prices at the nodes of the beginning of period \( j \). One dollar \( j \) periods from now has a known market value of \( 1/(1 + y)^j \), where \( y \) denotes the \( j \)-period spot rate. Alternatively, one dollar at the end of period \( j \) has a present value of

\[
f(r, v) \equiv \frac{P_1}{1 + r} + \frac{P_2}{1 + rv} + \frac{P_3}{1 + rv^2} + \cdots + \frac{P_j}{1 + rv^{j-1}}.
\]

To match the yield volatilities, we use the same logic as leads to (3) to obtain

\[
g(r, v) \equiv (1/2) \ln \left( \left( \frac{P_{u,1}}{1+r} + \frac{P_{u,2}}{1+rv} + \cdots + \frac{P_{u,j-1}}{1+rv^{j-2}} \right)^{\frac{1}{j-1}} - 1 \right) / \left( \frac{P_{d,1}}{1+r} + \frac{P_{d,2}}{1+rv} + \cdots + \frac{P_{d,j-1}}{1+rv^{j-2}} \right)^{\frac{1}{j-1}} - 1 \right) .
\]

In the above the \( P_{u,i} \)'s denote the state prices of the tree rooted at the up node, while the \( P_{d,i} \)'s denote the state prices of the tree rooted at the down node. Now, solve

\[
f(r, v) = \frac{1}{(1 + y)^j}
\]

\[
g(r, v) = \kappa
\]

Here, the Newton-Raphson method can be used to solve for \( r \) and \( v \) as the derivatives are easy to evaluate. The overall running time is now \( O(n^2) \). We emphasize
that forward induction is an efficient implementation of backward induction with
differential trees; it does not change the underlying computational logic a bit.

3.3 Experimental results

Consider the following term structures in Hull and White [1990],

\[ r^* + 0.05 \times \ln t \quad \text{for } t\text{-year zero-coupon bond yield} \]
\[ 1.4 \times (1 - e^{-0.1 \times t}) / t \quad \text{for } t\text{-year zero-coupon bond yield volatility} \]

3.3.1 Convergence of the differential tree method

The differential tree method converges very fast. Figure 5 reveals that, with one
partition per year, it takes an average of less than 3.5 iterations to achieve a relative
error of $10^{-13}$ with $r^* = 0.06$. With the Newton-Raphson method, it is critical to
get a suitable initial guess. Using the baseline rate and multiplicative ratio of the
previous period as the initial guesses for the current period has turned out to work
well.

3.3.2 Comparison between forward induction and backward induction

The running time of backward induction with the differential tree is $O(n^3)$ versus
$O(n^2)$ for forward induction. We benchmark backward induction with differential
trees and forward induction in Figure 6. The result is as expected: The forward
induction implementation is far more efficient. The running time is quadratic at
$0.000020 \times n^2$ seconds. Furthermore, we can calibrate the tree up to 270,000 years
(see Figure 7). From here on, the paper adopts a relative error of $10^{-6}$ since our
emphasis now shifts toward the growth rate of running time.

3.3.3 Aspects of forward induction

Figure 8 looks at the problem from a different perspective. It fixes the time span at
30 years and looks at how forward induction performs by increasing the number of
partitions. The conclusion is it can go as fine as 70 periods per year. The running
time is quadratic, at about $0.000098 \times n^2$ seconds. The experiment is then repeated
for 10-year trees with the results shown in Figure 9. The running time there grows
roughly as $0.000051 \times n^2$ seconds, and we can go as fine as 1,850 periods per year.

10
4 Spread Calculation

4.1 Spread of non-benchmark option-free bonds

Model prices calculated from trees calibrated off the benchmark bonds as a rule do not match market prices of non-benchmark bonds. To gauge the incremental return, or yield spread, over the benchmark bonds, one looks for the spread over the short rates in the tree that equates the model price and the market price (Fabozzi [1991]). When the underlying bond contains embedded options, the spread is called option-adjusted spread. We treat option-free bonds first and bonds with embedded options later. The techniques are identical save for the possibility of early exercise.

When a constant amount $s$ is added to every rate in the binomial interest rate tree, the model price will be a monotonically decreasing, convex function of $s$. Call this function $P(s)$. For a given market price $p$, we can employ a root-finding algorithm to solve $P(s) = p$.

Evaluating $P(s)$ directly by expansion looks intimidating. Fortunately, the differential tree method can be used to evaluate both $P(s)$ and $P'(s)$ in one pass. Here is the idea. Every node of the tree, call it node A, is the root of a subtree of the original tree and computes a price $p_A(s)$ during the process of computing the model price $P(s)$ via backward induction. Node A is also associated with a short rate $r$. Hence, prices computed at A’s two successor nodes, B and C, will be discounted by the factor $1/(1 + r + s)$ to obtain $p_A(s)$,

$$p_A(s) = c + \frac{p_B(s) + p_C(s)}{2(1 + r + s)}.$$

All this is standard. To compute $p'_A(s)$ as well, node A simply calculates

$$p'_A(s) = \frac{p'_B(s) + p'_C(s)}{2(1 + r + s)} - \frac{p_B(s) + p_C(s)}{2(1 + r + s)^2},$$

where $c$ denotes the cash flow at A. Now, computing $p'_A(s)$ is easy since $p'_B(s)$ and $p'_C(s)$ have been supplied by nodes B and C. Applying the above argument inductively will eventually lead to $P(s)$ and $P'(s)$. See Figure 10 for illustration.

The running time depends on the convergence rate of the Newton-Raphson method. In practice, only a small number of iterations is needed to bring the algorithm to convergence. Hence the empirical running time is $O(n^2)$. The memory requirement is again linear in $n$.  

11
The experiments apply the differential tree method to zero-coupon bonds under the BDT model and flat term structures,

- $8\%$ for $t$-year zero-coupon bond yield
- $10\%$ for $t$-year zero-coupon bond yield volatility

The differential tree method converges very fast. For example, it takes about 7.85 seconds for a tree with 500 periods on a 75MHz Sun SPARCstation 20 with 64MB of DRAM. In general, the running time is about $0.000031 \times n^2$ seconds.

### 4.2 Option-adjusted spread of non-benchmark bonds with embedded options

The calculation of OAS is the same as before except for the possibility of early exercise. The function $p_A(s)$ is slightly modified,

$$p_A(s) = c + \min \left( C, \frac{p_B(s) + p_C(s)}{2(1 + r + s)} \right), \quad (4)$$

where $C$ is the call price. The derivative $p_A'(s)$ is then

$$p_A'(s) = \begin{cases} 
0 & \text{if exercised} \\
\frac{p_B'(s) + p_C'(s)}{2(1 + r + s)} - \frac{p_B(s) + p_C(s)}{2(1 + r + s)^2} & \text{if not exercised}
\end{cases}$$

The experiments apply the differential tree method to 8% callable coupon bonds under the BDT model and flat term structures. The results show that the differential tree method converges very fast. For example, it takes about 8.11 seconds for a tree with 500 periods on a 75MHz Sun SPARCstation 20 with 64MB of DRAM. In general, the running time is quadratic, at about $0.0000065 \times n^2$ seconds per iteration.

### 5 Volatility Implied by American Options

The last case study is concerned with the implied volatility of American options. With early exercise, the option price as a function of volatility $\sigma$ is no longer differentiable everywhere. The backward induction formulation for American puts is

$$f_A(\sigma, b, c) = \begin{cases} 
\max \left( \frac{p^b + (1-p)c}{e^{\sigma\Delta t}}, X - S^j \right) & \text{if A is an internal node} \\
\max (X - S^j, 0) & \text{if A is a terminal node}
\end{cases}$$
(Compare it to European options in Section 2.) To apply the differential tree method, compute

\[
\frac{df_A}{d\sigma} = \begin{cases} 
\frac{u^b + pu^a - p\bar{c} + (1-p)c'}{e^{r\Delta t}} & \text{if } A \text{ is an internal node but not exercised} \\
0 & \text{if } A \text{ is a terminal node but not exercised} \\
-j\sqrt{\Delta t} Su^j & \text{if } A \text{ is exercised}
\end{cases}
\]

(5)

Note that the computational problem here is structurally similar to the OAS of (4). We solve the Black-Scholes formula for the implied volatility as the initial guess.

The fast convergence of the differential tree method is tabulated in Figure 11. The running time is quadratic: about 0.00000081 \times n^2 \text{ seconds for the American call and 0.00000045 \times n^2 \text{ seconds per iteration for the American put. Note that American calls will not be exercised early in our case of non-dividend-paying stock. Increasing the number of partitions may sometimes decrease the running time due to the reduction of the number of iterations. This is the case for the American put at } n = 600.

6 Conclusions

This paper presented the differential tree method, then applied it to representative computational problems in asset pricing. The method was furthermore shown to be extremely efficient. In terms of software development, the method results in a program structure which almost parallels that of its dual, the pricing module; only the derivatives calculation and adjustments to the approximate root at the end of each iteration need to be added.

Other interesting applications are easy to think of. Take the calibration of interest rate models again. For finite-dimensional models, the limited number of parameters makes matching exactly the term structures impossible. For them, one may aim to minimize the mean square error \(\| P(x) - p \|^2 \) between the market and the model-implied term structures. By providing the derivatives efficiently, the differential tree method can easily solve this optimization problem. We conclude that, as a general computing paradigm, the differential tree method has potential for efficient calibration of any tree-based models.
Acknowledgments

The author thanks George Wei-Tso Chan, Wei-Jui Chen, and Chih-Jen Lin for assistance and discussions.
References


Figure 1: The differential tree method. Computation at node A is driven by inputs from B and C.
Figure 2: Binomial interest rate process. From node A, there are two equally likely scenarios for the short rates. Rate \( r \) is applicable to node A for period \( j \). Rate \( r_i \) is applicable to node B and rate \( r_h \) to node C, both for period \( j + 1 \).
Figure 3: Binomial tree for the BDT model.
Figure 4: Sweeping a line across time forward to compute the binomial state price tree. The state price at a node is a weighted sum of the state prices of its two predecessors.
<table>
<thead>
<tr>
<th>Number of Years</th>
<th>Average number of iterations</th>
<th>Sample deviation</th>
<th>Number of Years</th>
<th>Average number of iterations</th>
<th>Sample deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>3.474747</td>
<td>0.519196</td>
<td>1100</td>
<td>2.926297</td>
<td>0.394491</td>
</tr>
<tr>
<td>200</td>
<td>3.236181</td>
<td>0.436405</td>
<td>1200</td>
<td>2.917431</td>
<td>0.394738</td>
</tr>
<tr>
<td>300</td>
<td>3.157192</td>
<td>0.373056</td>
<td>1300</td>
<td>2.923788</td>
<td>0.379878</td>
</tr>
<tr>
<td>400</td>
<td>3.085213</td>
<td>0.384873</td>
<td>1400</td>
<td>2.922802</td>
<td>0.373977</td>
</tr>
<tr>
<td>500</td>
<td>3.020040</td>
<td>0.414660</td>
<td>1500</td>
<td>2.893262</td>
<td>0.399234</td>
</tr>
<tr>
<td>600</td>
<td>2.973289</td>
<td>0.431584</td>
<td>1600</td>
<td>2.870544</td>
<td>0.415614</td>
</tr>
<tr>
<td>700</td>
<td>2.951359</td>
<td>0.428502</td>
<td>1700</td>
<td>2.847557</td>
<td>0.430938</td>
</tr>
<tr>
<td>800</td>
<td>2.929912</td>
<td>0.430495</td>
<td>1800</td>
<td>2.831573</td>
<td>0.439798</td>
</tr>
<tr>
<td>900</td>
<td>2.923248</td>
<td>0.421481</td>
<td>1900</td>
<td>2.817672</td>
<td>0.447092</td>
</tr>
<tr>
<td>1000</td>
<td>2.919920</td>
<td>0.412023</td>
<td>2000</td>
<td>2.806903</td>
<td>0.451480</td>
</tr>
</tbody>
</table>

Figure 5: **Convergence of the Differential Tree Method.** The zero-coupon bond yield is described by $0.06 + 0.05 \times \ln t$. The time partition is one period per year. Sample deviation refers to that of the number of iterations. The termination condition is $10^{-13}$ relative error.
<table>
<thead>
<tr>
<th>Number of Years</th>
<th>Backward induction</th>
<th>Forward induction</th>
<th>Number of Years</th>
<th>Backward induction</th>
<th>Forward induction</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1.8</td>
<td>0.2</td>
<td>1100</td>
<td>2237.1</td>
<td>24.5</td>
</tr>
<tr>
<td>200</td>
<td>14.2</td>
<td>0.8</td>
<td>1200</td>
<td>2887.5</td>
<td>29.5</td>
</tr>
<tr>
<td>300</td>
<td>47.5</td>
<td>1.8</td>
<td>1300</td>
<td>3703.7</td>
<td>35.2</td>
</tr>
<tr>
<td>400</td>
<td>109.2</td>
<td>3.2</td>
<td>1400</td>
<td>4632.5</td>
<td>41.1</td>
</tr>
<tr>
<td>500</td>
<td>207.3</td>
<td>5.0</td>
<td>1500</td>
<td>5557.0</td>
<td>46.6</td>
</tr>
<tr>
<td>600</td>
<td>352.2</td>
<td>7.1</td>
<td>1600</td>
<td>6682.1</td>
<td>52.2</td>
</tr>
<tr>
<td>700</td>
<td>560.0</td>
<td>9.6</td>
<td>1700</td>
<td>8099.5</td>
<td>58.7</td>
</tr>
<tr>
<td>800</td>
<td>844.7</td>
<td>12.6</td>
<td>1800</td>
<td>9286.6</td>
<td>65.3</td>
</tr>
<tr>
<td>900</td>
<td>1223.5</td>
<td>15.9</td>
<td>1900</td>
<td>11088.6</td>
<td>72.9</td>
</tr>
<tr>
<td>1000</td>
<td>1690.7</td>
<td>20.1</td>
<td>2000</td>
<td>12745.2</td>
<td>80.9</td>
</tr>
</tbody>
</table>

Figure 6: **Backward induction versus forward induction.** The setup is the same as Figure 5. All times are measured in seconds obtained on a 200MHz Sun Ultra-SPARC with 256MB of DRAM. The termination condition is $10^{-13}$ relative error. Data are from Chen [1997].
<table>
<thead>
<tr>
<th>Number of Years</th>
<th>Running time</th>
<th>Number of Years</th>
<th>Running time</th>
<th>Number of Years</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>398.88</td>
<td>39000</td>
<td>8562.640</td>
<td>75000</td>
<td>26182.080</td>
</tr>
<tr>
<td>6000</td>
<td>1697.680</td>
<td>42000</td>
<td>9579.780</td>
<td>78000</td>
<td>28138.140</td>
</tr>
<tr>
<td>9000</td>
<td>2539.040</td>
<td>45000</td>
<td>10785.850</td>
<td>81000</td>
<td>30230.260</td>
</tr>
<tr>
<td>12000</td>
<td>2803.890</td>
<td>48000</td>
<td>11905.290</td>
<td>84000</td>
<td>32317.050</td>
</tr>
<tr>
<td>15000</td>
<td>3149.330</td>
<td>51000</td>
<td>13199.470</td>
<td>87000</td>
<td>34487.320</td>
</tr>
<tr>
<td>18000</td>
<td>3549.100</td>
<td>54000</td>
<td>14411.790</td>
<td>90000</td>
<td>36795.430</td>
</tr>
<tr>
<td>21000</td>
<td>3990.050</td>
<td>57000</td>
<td>15932.370</td>
<td>120000</td>
<td>63767.690</td>
</tr>
<tr>
<td>24000</td>
<td>4470.320</td>
<td>60000</td>
<td>17360.670</td>
<td>150000</td>
<td>98339.710</td>
</tr>
<tr>
<td>27000</td>
<td>5211.830</td>
<td>63000</td>
<td>19037.910</td>
<td>180000</td>
<td>140484.180</td>
</tr>
<tr>
<td>30000</td>
<td>5944.330</td>
<td>66000</td>
<td>20751.100</td>
<td>210000</td>
<td>190557.420</td>
</tr>
<tr>
<td>33000</td>
<td>6639.480</td>
<td>69000</td>
<td>22435.050</td>
<td>240000</td>
<td>249138.210</td>
</tr>
<tr>
<td>36000</td>
<td>7611.630</td>
<td>72000</td>
<td>24292.740</td>
<td>270000</td>
<td>313480.390</td>
</tr>
</tbody>
</table>

Figure 7: **Forward Induction (continued).** We continue the benchmarking in Figure 6 for forward induction except that the termination condition is $10^{-6}$ relative error, and the times are measured on a slower 75MHz Sun SPARCstation 20 with 64MB of DRAM.
<table>
<thead>
<tr>
<th>Number of partitions</th>
<th>Running time</th>
<th>Number of partitions</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>7.310</td>
<td>1500</td>
<td>166.130</td>
</tr>
<tr>
<td>600</td>
<td>26.400</td>
<td>1800</td>
<td>291.120</td>
</tr>
<tr>
<td>900</td>
<td>59.480</td>
<td>2100</td>
<td>497.130</td>
</tr>
<tr>
<td>1200</td>
<td>106.130</td>
<td>2400</td>
<td>diverge</td>
</tr>
</tbody>
</table>

Figure 8: **Forward Induction: The 30-Year Case.** The setup is the same as Figure 7 except that \( r^n = 0.08 \), and the tree covers 30 years. The algorithm fails to terminate at 2,400.
<table>
<thead>
<tr>
<th>Number of partitions</th>
<th>Running time</th>
<th>Number of partitions</th>
<th>Running time</th>
<th>Number of partitions</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2100</td>
<td>242.110</td>
<td>4300</td>
<td>956.470</td>
<td>8500</td>
<td>3585.680</td>
</tr>
<tr>
<td>2300</td>
<td>291.850</td>
<td>4500</td>
<td>1037.540</td>
<td>9500</td>
<td>4500.570</td>
</tr>
<tr>
<td>2500</td>
<td>349.140</td>
<td>4700</td>
<td>1126.620</td>
<td>10500</td>
<td>5475.320</td>
</tr>
<tr>
<td>2700</td>
<td>407.370</td>
<td>4900</td>
<td>1202.000</td>
<td>11500</td>
<td>6579.760</td>
</tr>
<tr>
<td>2900</td>
<td>468.490</td>
<td>5100</td>
<td>1313.750</td>
<td>12500</td>
<td>7798.940</td>
</tr>
<tr>
<td>3100</td>
<td>527.100</td>
<td>5300</td>
<td>1394.370</td>
<td>13500</td>
<td>9191.240</td>
</tr>
<tr>
<td>3300</td>
<td>588.520</td>
<td>5500</td>
<td>1498.260</td>
<td>14500</td>
<td>10697.090</td>
</tr>
<tr>
<td>3500</td>
<td>651.360</td>
<td>5700</td>
<td>1609.560</td>
<td>15500</td>
<td>12167.020</td>
</tr>
<tr>
<td>3700</td>
<td>727.550</td>
<td>5900</td>
<td>1728.960</td>
<td>16500</td>
<td>13748.310</td>
</tr>
<tr>
<td>3900</td>
<td>804.130</td>
<td>6500</td>
<td>2067.470</td>
<td>17500</td>
<td>15732.760</td>
</tr>
<tr>
<td>4100</td>
<td>858.850</td>
<td>7500</td>
<td>2750.240</td>
<td>18500</td>
<td>17537.810</td>
</tr>
</tbody>
</table>

Figure 9: **Forward Induction: The 10-Year Case.** The setup is the same as Figure 8 except that the tree covers 10 years. The algorithm fails to terminate at 19,000.
Figure 10: The differential tree for spread. (a) The original binomial interest rate tree with short rates replaced by the discount factors. (b) The derivatives of the numbers in (a). (c) The simultaneous evaluation of a function and its derivative with the binomial tree structure.
<table>
<thead>
<tr>
<th>Number of partitions</th>
<th>American call</th>
<th>American put</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Running time</td>
<td>Number of iterations</td>
</tr>
<tr>
<td>100</td>
<td>0.008210</td>
<td>2</td>
</tr>
<tr>
<td>200</td>
<td>0.033310</td>
<td>2</td>
</tr>
<tr>
<td>300</td>
<td>0.072940</td>
<td>2</td>
</tr>
<tr>
<td>400</td>
<td>0.129180</td>
<td>2</td>
</tr>
<tr>
<td>500</td>
<td>0.201850</td>
<td>2</td>
</tr>
<tr>
<td>600</td>
<td>0.290480</td>
<td>2</td>
</tr>
<tr>
<td>700</td>
<td>0.394090</td>
<td>2</td>
</tr>
<tr>
<td>800</td>
<td>0.522040</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 11: IMPLIED VOLATILITY OF AMERICAN OPTIONS. The computer setup uses a PC equipped with an Intel 166MHz Pentium MMX and 32MB of DRAM, running Microsoft Windows 95. The program terminates when the volatility improves by less than $10^{-5}$. The parameters for the American options are 49 (stock price), 50 (exercise price), 2.39 (option price), 5% (interest rate), and 140 (days to maturity). The implied volatility is found to be 19.95% for the call and 18.55% for the put.