COMPUTATIONAL FINANCE

Lecture 1: Pricing Interest Derivatives
A Simple Binomial Interest Option Pricing Program

Philip H. Dybvig
Washington University
Saint Louis, Missouri

Some Simple Interest Derivatives

- Riskless Bonds
- Bond Options
  - Puts, Calls, Straddles, etc.
  - American, European, Down-and-Out, etc.
- Bond Futures and Futures Options
- Caps Floors, Collars
- Riskless Inverse Floaters

Some Complex Interest Derivatives

- Mortgages and CMOs
- Structured Loans
- Risky Corporate Bonds
- Callable and/or Convertible Bonds
- Foreign Exchange Futures Options
- Hybrid Securities, e.g. a binary option paying off if at maturity 3-mo LIBOR $> 12\%$ and the dollar is stronger against the yen than it was at the start of the contract

Why Not Simply Use Black-Scholes?

- The interest rate is not constant.
- The volatility is not constant.
- "Today's price" is not an asset price.

A very clever (or lucky) application of Black-Scholes may give a reasonable approximation, but it is simpler and more reliable to price a claim directly with a model designed to price interest derivatives.

Binomial Pricing of Interest Derivatives

$$r \begin{array}{l} \nearrow r+\delta \\ \searrow r-\delta \end{array}$$

The interest rate is not an asset! Therefore, we can't use the formula from the previous lecture to compute the risk-neutral probabilities or state prices. There are several approaches:

- Make an assumption about the price process for some asset (e.g. a perpetuity).
- Make an assumption about the nature of supply and demand in the economy and compute equilibrium prices.
- Make an assumption about the interest rate process in the risk-neutral probabilities, e.g.
  - Random walk
  - Modest mean reversion (my favorite)

A Random Walk or Modest Mean Reversion

We choose the risk-neutral probabilities to induce a modest amount of mean reversion, say 12-15% per year. If we want

$$E[\Delta r] = k(r^* - r)\Delta t,$$

then

$$\pi_u \delta + (1 - \pi_u)(-\delta) = k(r^* - r)\Delta t$$

or

$$\pi_u = \frac{1}{2} + \frac{k(r^* - r)\Delta t}{\delta}.$$

A random walk (good for short maturities and non-critical applications) corresponds to $k = 0$.

Another issue: use uneven spacing to make interest rate volatility a function of the interest rate. Fudge factors can fit today's yield curve. *Stochastic volatility* and additional factors are harder.

Two Observations

About Timing The short riskless rate is known at the *beginning* of the period, so the riskless rate we learn now affects the riskless return (and therefore the discounting) over the time period starting now. Therefore, the pricing of a riskless bond involves computations using interest rates up until one period before maturity.

About Intermediate Cash Flows When a claim includes intermediate cash flows (as for a coupon bond or a cap), the claim is simply added in at the appropriate time. For example, if V indicates the ex-cashflow value and C the cashflow, we have, at some node at time t,

$$V(t,r) = \frac{\pi_u^*(V(t,r+\delta)+C(t,r+\delta))+\pi_d^*(V(t,r-\delta)+C(t,r-\delta))}{(1+r)}$$

In-class Exercise: Bond Prices

Consider a two-period binomial model. The short riskless interest rate starts at 20% and moves up or down by 10% each period (i.e., up to 30% or down to 10% at the first change). The risk neutral probability of each of the two states is 1/2. What is the price (after the coupon is paid) at each node of a discount bond with face value of \$100 maturing two periods from the start? (Hint: solve back one period at a time. Be sure to use the appropriate discount factor at each node!) What is the price at each node of a bond with a face value of \$100 and a coupon of 10% per period?

In-Class Exercise: Bond Option Evaluation

For the coupon bond in the previous In-Class Exercise, compute the initial value of a European call option on the coupon bond. The call option matures in the middle period and has an exercise price of \$90. (Exercise of the option does not give you a claim to the coupon in the middle period.)

The Header File cap.h

```
// cap.h
//
// Fixed-income binomial option pricing include file
//
#ifndef MAXTERNODES
#define MAXTERNODES 200
#endif
class f_i_bin {
public:
  f_i_bin(double ttm=.25,int npers=4,double sigma=.02,
    double rrbar=.07,double k=.125);
  double bprice(double r0);
  double cap(double level,double r0);
private:
  int nper;
  double tinc,up,down,sig,rbar,kappa,prfact;
  double r[MAXTERNODES],val[MAXTERNODES];};
```

The Test File captest.cc

```
// captest.cc
//
// Fixed income binomial option pricing model: test
//
#include <iostream.h>
#include "cap.h"
main() {
  f_i_bin c2;
  double caprate,rzero;
  cout << "\nType the cap rate, a space, the initial"
    << " rate, and then ENTER." << "\n"
    << "Make the cap rate negative to terminate." << "\n\n";
  cout << "State rates in percent per annum (without the "
    << "percent sign).\n\n";
  while(1) {
    cout << "cap-rate initial-rate: ";
    cin >> caprate >> rzero;
```

```
if(!cin) {
  cout << "\nError: expected cap-rate initial-rate\n"
    << "Terminating\n" << endl;
  return(1);}
if(caprate<0.0) {
  cout << endl;
  return(0);}
cout << "cap value = " << c2.cap(caprate/100.0,rzero/100.0)
  << "\n\n";}}
```

The Implementation File cap.cc: Part 1

```
// cap.cc
//
// Fixed-income binomial option pricing engine
//
#include <math.h>
#include <iostream.h>
#include "cap.h"
#define MAX(a,b) (((a) > (b)) ? (a) : (b))
#define MIN(a,b) (((a) < (b)) ? (a) : (b))
```

The Implementation File cap.cc: Part 2

```
f_i_bin::f_i_bin(double ttm,int npers,double sigma,double rrbar,double k) {
  nper=npers;
  tinc = ttm/(double) nper;
  sig = sigma;
  up = sigma*sqrt(tinc);
  rbar = rrbar;
  kappa = k;
  prfact = kappa*sqrt(tinc)/(2.0*sig);}
```

The Implementation File cap.cc: Part 3

```
double f_i_bin::bprice(double r0) {
  int i,j;
  double prup;
//initialize terminal payoffs
//i is the number of up moves
  for(i=0;i<=nper;i++) {
//  r[i] = r0 + up * (double)(2*i-nper);   not needed for this claim
    val[i] = 1.0;}
//compute prices back through the tree
//j+1 is the number of periods from the end
//i is the number of up moves from the start
  for(j=0;j<nper;j++) { for(i=0;i<nper-j;i++) {
      r[i] = r0 + up * (double) (2*i-nper + j);
      prup = 0.5 + prfact*(rbar-r[i]);
      prup = MIN(1.0,MAX(0.0,prup));
      val[i] = (prup*val[i+1]+(1.0-prup)*val[i])*exp(-r[i]*tinc);}}
  return(val[0]);}
```

The Implementation File cap.cc: Part 4

```
double f_i_bin::cap(double level,double r0) {
  int i,j;
  double prup;
//initialize terminal payoffs:  i is the number of up moves
  for(i=0;i<=nper;i++) {
//  r[i] = r0 + up * (double)(2*i-nper);   not needed for this claim
    val[i] = 0.0;}
//compute prices back through the tree
//j is the number of periods from the end
//i is the number of up moves from the start
  for(j=1;j<=nper;j++) {for(i=0;i<=nper-j;i++) {
      r[i] = r0 + up * (double) (2*i-nper + j);
      prup = 0.5 + prfact*(rbar-r[i]);
      prup = MIN(1.0,MAX(0.0,prup));
      val[i] = (prup*val[i+1]+(1.0-prup)*val[i])*exp(-r[i]*tinc)
        + MAX(0.0,(r[i]-level)*tinc);}}
  return(val[0]);}
```