COMPUTATIONAL FINANCE

Lecture 2: Pricing Interest Derivatives
A Simple Binomial Interest Option Pricing Applet

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.
- We may want to value claims that are not simple combinations of puts and calls.

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

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

We choose $\delta = \sigma * \sqrt{\Delta t}$.

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 preference)

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

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

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 HTML File *Caplet.html*

```
<HTML>
<HEAD>
<TITLE>Binomial Cap Pricing Program</TITLE>
</HEAD>
<BODY>
<APPLET CODE=Caplet.class WIDTH=400 HEIGHT=100>
</APPLET>
</BODY>
</HTML>
```

The Program File *Caplet.java*

```
//
// Fixed income binomial cap pricing applet
//
import java.applet.*;
import java.awt.*;

public class Caplet extends Applet {
  F_I_bin c2;
  double caprate,rzero;
  Label capval;
  TextField interest_rate, capped_level;
  public Caplet() {
    setLayout(new GridLayout(3,2));
    add(new Label("Interest rate (%) ="));
    add(interest_rate = new TextField("5",10));
    add(new Label("Capped level (%) ="));
    add(capped_level = new TextField("5.5",10));
    add(new Label("Cap value (per $100 face) ="));
    add(capval = new Label("**********"));
    c2 = new F_I_bin((double) 2.0, (int) 24, (double) 0.01, (double) 0.05,
      (double) 0.125, (int) 5001);
    recalc();}
 public boolean action(Event ev, Object arg) {
    if(ev.target instanceof TextField) {
      recalc();
      return true;}
    return false;}
  double text2double(TextField tf) {
    return Double.valueOf(tf.getText()).doubleValue();}
  void recalc() {
    capval.setText(String.valueOf((float) (100 *
```

```
      c2.cap(text2double(capped_level)/100.0,
      text2double(interest_rate)/100.0))));}}
//
// Fixed-income binomial option pricing engine
//

class F_I_bin {
  int nper;
  double tinc,up,down,sigma,rbar,kappa,prfact;
  double [] r,val;

  public F_I_bin(double ttm,int nper,double sigma,double rbar,double kappa,
      int maxternodes) {
    this.nper=nper;
    tinc = ttm/(double) nper;
    this.sigma = sigma;
    up = sigma*Math.sqrt(tinc);
    this.rbar = rbar;
    this.kappa = kappa;
    prfact = kappa*Math.sqrt(tinc)/(2.0*sigma);
    val = new double[maxternodes];
    r = new double[maxternodes];}

  double 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 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 = Math.min((double) 1.0,Math.max((double) 0.0,prup));
        val[i] = (prup*val[i+1]+(1.0-prup)*val[i])*Math.exp(-r[i]*tinc);}}
    return(val[0]);}

  double 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 = Math.min((double) 1.0,Math.max((double) 0.0,prup));
        val[i] = (prup*val[i+1]+(1.0-prup)*val[i])*Math.exp(-r[i]*tinc)
          + Math.max((double) 0.0,(r[i]-level)*tinc);}}
    return(val[0]);}}
```