Lecture 4
Finance Project
Somesh Jha
Low-Level Design document

- We will call the Low-Level Design document LLD from now on.
- Be very detailed. Programming should be a very small step after this document is over.
- State the language upfront. We will be using JAVA and hence Object-Oriented Programming.
- Mention each object with its purpose and description of the constructors and the methods which can be called from outside (these will be the public methods in JAVA).
LLD (Contd)

- Order your objects by uses relation. For example, if $O_1$ is used by object $O_2$ describe $O_1$ first. In case of recursion, pick an order arbitrarily.

- Indicate if a class extends some other class or if a class is going to be abstract. Also, give a short rationale why did you choose to make a certain class abstract.

- Revisit the LLD while/after doing the implementation. You will be straying from the design a little bit. We will use LLD and additional information to understand and test your code.
Logistics

- Due Date: Feb 12, 1999.
- **Remember more detail the better.** This will make your job while implementation very easy.
- Also try to distribute work so that different people work on descriptions of different objects.
- Mention clearly who is responsible for which object. The person describing the design of an object will also implement it. In the implementation the author of each object should be mentioned very clearly in the header files.
Logistics (Contd)

- There should be one system integrator who will take descriptions/implementation of the objects and see whether they all fit together. The main loop of the program will be put together by the system integrator.
- Mention the roles of the team members in the document.
- After a team member implements an object, another team member should review the code. You will be surprised how many silly errors you will catch during review. Mention the reviewer in the header file with the object (along with the author of-course).
Implementation

• Due date: March 1, 1999.

• Make sure you have clear instructions describing how to use the system.

• State limiting assumptions you made while implementing.

• Please give a phone number of a person we can call in case we have difficulty running your system. This person should preferably be the system integrator because he/she has the overall idea about your system.
MortgagePool object

- **ObjectName**: MortgagePool object.
- **Extends**: Object.
- **Implements**: None.
- **Uses**: None.

**Constructor**
A single constructor which takes various parameters of the mortgage pool. Please see Lecture 2.

**Assumption**: I am going to assume a homogeneous mortgage pool.
MortgagePool object (Contd)

- **Method:**
  double[] cashFlows(double SMMS[])
  Takes as parameter array of SMMs for various times and returns array of cash-flows for each time up to the lifetime of the mortgage. Assume that array of SMMs has same size as the lifetime of the mortgage pool.

- **Method:**
  double next-cash-flow(double SMM)
  This allows the object to be used in an *iterative mode*. Whenever, this method is called the cash-flow in the current time period is returned and the current time period in the mortgage pool is incremented.
The parameter SMM determines the prepayment for this time period.
BondObject object

- **Name**: BondObject.
- **Extends**: Object.
- **Implements**: None.
- **Uses**: None.

**Constructor**
Name of the file/database (with the bond data) is passed to the constructor. We will assume that the bond data is in a file with all the required quantities.
BondObject object (Contd)

- **Methods:**
  - `double yield(t,T)`
    Yield at time $t$ of a zero-coupon bond maturing at time $T$.
  - `double price(t,T)`
    Price at time $t$ of a zero-coupon bond maturing at time $T$.
  - `double volatility(t,T)`
    Volatility at time $t$ of a zero-coupon bond maturing at time $T$.

- **Note:** This object will be used in pricing MBS with deterministic cash-flows. Please see Lecture 3 for a closed form expression.
NormalRandom object

- **Name**: NormalRandom.
- **Extends**: None.
- **Uses**: java.util.Random.
- **Constructor**: 
  Take the mean and variance as parameters and record it internally. We will generate two numbers with Normal distribution with mean $m$ and variance $v$. 
NormalRandom object (Contd)

- **Method:**
  ```java
double[] nextRandom()
```
Generate *two* random numbers with standard normal distribution. Method we will follow is due to Box-Muller-Marsaglia. Please see next slide for a description of the method. Before returning the random numbers apply the appropriate transform to match the mean and the variance given in the constructor. Using the following transformation

\[(x + m)\sqrt{v}\]
Box-Muller-Marsaglia method

- This algorithm is also called the polar method.

- **Step 1**
  Generate two random variables $U_1$ and $U_2$ (use the method `nextDouble`) in the class `java.util.Random`. Transform these variables according to the equations given below:
  \[
  V_1 = 2U_1 - 1 \\
  V_2 = 2U_2 - 1
  \]
Box-Muller-Marsagalia (Contd)

- **Step 2**
  Compute $S$ according to the equation given below:
  $$V_1^2 + V_2^2$$

- **Step 3**
  If $S \geq 1$, go to step 1.

- **Step 4**
  Return $X_1$ and $X_2$ given by the equations:
  $$X_1 = V_1 \sqrt{-\frac{2 \ln S}{S}}$$
  $$X_2 = V_2 \sqrt{-\frac{2 \ln S}{S}}$$

- **Note:** Steps 1 through 3 are executed 1.27 times on the average with standard deviation of 0.587. So we are not returning to Step 1 too many times.
InterestRate

- **Name**: InterestRate.
- **Extends**: None.
- **Implements**: None.
- **Uses**: NormalRandom.

**Constructor**

We will use the Cox-Ingersoll-Ross model. Constructor will take all the parameters as arguments. Please see the SDE given below:

\[ dr = \kappa (\mu - r) dt + \sigma \sqrt{r} dW \]

The parameters \( \kappa \), \( \mu \), \( \sigma \), and the initial short rate \( r_0 \) are passed to the constructor. **Assumption**: The model has already been calibrated.
InterestRate object (Contd)

- **Method**
  
  ```java
  void instantiate(double N, double T)
  ```
  
  Parameter `T` is the time horizon. `N` is the number of discrete time steps we will divide the time interval `[0, T]` into.

  **Assumption:** We assume that `T` is in months and `N` has granularity of at-least a month.

- **Method:**
  
  ```java
  double[] nextPath()
  ```
  
  Generates a random path where the `t`-th element in the array is the short rate at the `t`-th time. Let `h` be the step size given by
the following expression:

\[
\begin{align*}
\frac{T}{N}
\end{align*}
\]

Generate random path according to the following recurrence equation:

\[
r(i + 1) = \kappa(\mu - r(i))h + \sigma\sqrt{r(i)}N(0, h)
\]

where \(r(i)\) is the short rate at the discrete time step \(i\) and \(N(0, h)\) is a random number with normal distribution (mean 0 and variance \(h\)). Use method in object \texttt{NormalRandom} is used to generate this number.
PrePayment object

- **Name**: PrePayment.
- **Extends**: None.
- **Implements**: None.
- **Uses**: None.

**Constructor**
Pass a flag indication which option of prepayment function is going to be used (see Lecture 3 for explanation of the options). For option A pass a vector of PSAs and for option B pass the various parameters $\beta_1, \beta_2, \beta_3$. 
PrePayment (Contd)

- **Method:** double[] smmVector(int T)
  Use only with option A. Returns the vector of SMMs up to the time horizon $T$.

- **Method:** double smmRandom(double pi, double rf, double burnout, double season)
  Gives the random SMM given the required parameters. Please see Lecture 3 for the explanation. Need a random number with Poisson Distribution (Haven’t described it here).
PassThrough

- **Name:** PassThrough.
- **Extends:** None.
- **Implements:** None.
- **Uses:** MortgagePool, InterestRate, BondObject, and PrePayment.
- **Constructor**
  Pass an object of type MortgagePool, InterestRate, Prepayment, BondObject, and time horizon to the constructor. 
  **MortgagePool** is the underlying mortgage pool for the pass-through security.
PassThrough (Contd)

- **Method**
  double priceDeterministic()
  If the Prepayment generates deterministic SMMS, use the closed form expression given in Lecture 3.

- **Method**
  double price()
  Determine if the prepayment model is deterministic or random. If prepayment model is deterministic call method priceDeterministic(). If prepayment model is random (option B) use monte-carlo simulation. Generate a covariate based on the general technique described in Lecture 3. Price using monte-carlo simulation. Use
method `nextPath()` in the object of type `InterestRate`. 
CMOObject

• **Name**: CMOObject.
• **Extends**: None.
• **Uses**: MortgagePool, InterestRate, BondObject, PrePayment.
• **Implements**: None.

**Constructor**
Pass an object of type MortgagePool, InterestRate, BondObject, PrePayment and the time horizon to the constructor. **MortgagePool** is the underlying mortgage pool for the pass-through security. Also pass the number of tranches and par-value of each tranche to the constructor.

• **Method**
double[] priceDeterministic()
If the PrepaymentObject generates
deterministic SMMS, use the closed form
expression given in Lecture 3. Returns price
of each tranch.

• Method
double[] price()
Determine if the prepayment model is
deterministic or random. If prepayment
model is deterministic call method
priceDeterministic. If prepayment
model is random (option B) use monte-carlo
simulation. Generate a covariate based on
the general technique described in Lecture
Report price of each tranch.
StrippedMBS

- **Name**: StrippedMBS.
- **Extends**: None.
- **Uses**: InterestRate, MortgagePool, PrePayment, and BondObject.
- **Implements**: None.

**Constructor**
Pass an object of type MortgagePool, InterestRate, Prepayment, BondObject, and time horizon to the constructor. **MortgagePool** is the underlying mortgage pool for the stripped MBS.

**Method**
double[] priceDeterministic()
If the PrepaymentObject generates deterministic SMMS, use the closed form expression given in Lecture 3. Returns the price of PO and IO class.

**Method**

double[] price()

Determine if the prepayment model is deterministic or random. If prepayment model is deterministic call method priceDeterministic. If prepayment model is random (option B) use monte-carlo simulation. Generate a covariate based on the general technique described in Lecture 3. Price using monte-carlo simulation. Report price of PO and IO class.
Describe the flow

- **Step 1**: Ask the user for parameters of the underlying mortgage pool. Create a MortgagePool object.
- **Step 2**: Ask the user the prepayment option he/she wants to use. Instantiate an object of type PrePayment object.
- **Step 3**: Create objects of type PrePayment and BondObject.
- **Step 4**: Ask the user what MBS he/she wants to price. Instantiate a PassThrough, CMOObject, or StrippedMBS object based on this.
- **Step 5**: Call the method price() in the
required MBS object.

• *Step 6:* Return the result to the user.
Discretizing an SDE

• SDE stands for stochastic differential equation.

• Suppose $y_t$ follows the SDE given below:

$$dy_t = \mu(y, t)dt + \sigma(y, t)dW_t$$

• Drift term is $\mu(y, t)$ and the volatility term is $\sigma(y, t)$. 
Goal

- Our goal is to build a lattice structure corresponding to the process $y_t$.

- Suppose we are only interested in time-interval $[0, T]$.

- **Step 1:** Discretize File: slides-4.tex the interval into $N$ time steps. Each step-size is of the size $h = \frac{T}{N}$. The discrete time steps are $[0, h, 2h, \ldots, Nh]$. 
Goal (Contd)

• **Step 2:** Suppose we are at a node in the lattice where the process has value $y$. Successors of the lattice and the probability on edges is given by the following equations:

$$Y_{h^+} = y + \sqrt{h}\sigma(y, t)$$
$$Y_{h^-} = y - \sqrt{h}\sigma(y, t)$$
$$q_h = \frac{1}{2} + \sqrt{h} \frac{\mu(y, t)}{2\sigma(y, t)}$$

• $Y_{h^+}$ and $Y_{h^-}$ is the value of the process in the *up* and *down* nodes respectively.

• Probability of an *up-move* is $q_h$. 
Problem (no recombination)

- Total displacement for up-move followed by down-move is:
  \[ \sqrt{h}[-\sigma(y, t) + \sigma(Y_h^+, t + h)] \]

- Total displacement for down-move followed by up-move is:
  \[ \sqrt{h}[-\sigma(y, t) + \sigma(Y_h^+, t + h)] \]

- In general the two quantities are not equal.

- No recombination. We want a recombining lattice.

- When does the lattice recombine?
When does it recombine?

- When the volatility of $y$ (given by $\sigma(y, t)$) is constant.

**Basic idea:**
Transform $Y$ to a new process $X$ with constant volatility.

- Define $X(y, t)$ as follows:

$$
\int_0^y \frac{1}{\sigma(Z, t)} dZ
$$

- Assuming that $X(y, t)$ is twice differentiable in $y$ and once in $t$, we can use Ito’s lemma.
to show that $X(y, t)$ satisfies the following SDE:

$$dX(y_t, t) = \mu_X(y_t, t)dt + dW_t$$

Following equality should be easy to see:

$$\frac{\partial X}{\partial y} = \frac{1}{\sigma(y, t)}$$

The volatility term in the SDE for $X$ is given by the following formula:

$$\frac{\partial X}{\partial y} \sigma(y, t)$$
Basic Idea

• Also assume that we can invert $X$, i.e. there exists a function $Y(x, t)$ such that:

\[ Y(X(y, t), t) = y \]

• Most of the time we will only consider cases where we can find analytic expressions for $X$ and $Y$. This is the case in the example we will consider.
• **Warning:** I am glossing over lot of technical details. Will give reference at the end.

• Build a lattice for the transformed $X$ process. The lattice for $X$ is recombining.

• Using the inverse $Y$ function to derive a lattice for $y$ process. It is not as simple as this. There are few technicalities.
Recombining lattice for CIR

- The SDE for the short-rate in the Cox-Ingersoll-Ross model (known as CIR) from here on is given by the following SDE:

\[ dr = \kappa (\mu - r) dt + \sigma \sqrt{r} dW \]

- Initial value of the short rate is \( r_0 \).

- **Goal:** To build a recombining lattice for the CIR model.

- Transform the \( r \) process

\[
X(r) = \int_0^r \frac{1}{\sigma \sqrt{Z}} dZ
\]

\[
= \frac{2\sqrt{r}}{\sigma}
\]
CIR (Contd)

- If $\kappa \geq 0$, $\mu \geq 0$, and $r_0 > 0$, then 0 is lower boundary for $r$ (This can be proved formally). Interest rate can never go negative. This is an attractive feature of the CIR model.

- Inverse transform for the $X$ process is:

  $$R(x) = \begin{cases} 
  \frac{\sigma^2 x^2}{4} & \text{if } x > 0 \\
  0 & \text{otherwise}
  \end{cases}$$

- We never want $R(x)$ (which is the short rate) to go negative. Therefore we have the otherwise clause.

- Is there mean reversion in the model?
Algorithm

- Build the lattice for the $X$ process. Derive SDE for $X$ using Ito’s lemma. Use the simple construction.
- At each node in the $X$-lattice we have the value of $X$. We want to transform this lattice into the lattice for short-rate.
- Suppose we are at a node with the value of $X$ process $x$. The $r$ value corresponding to this is given by $\frac{\sigma^2 x^2}{4}$. 
Now we have to decide the successors of node with short rate $R(x)$ and also want to decide the probability of up-move. These quantities are given by the following equation:

$$
\begin{align*}
    x_h^+ &= x + J_h^+(x) \sqrt{h} \\
    x_h^- &= x + J_h^-(x) \sqrt{h} \\
    R_h^\pm &= R(x \pm J_h^\pm \sqrt{h})
\end{align*}
$$

The probability $q_h$ of making an up move is 0 if $R_h^+(x) = 0$ and if $R_h^+(x) > 0$ the expression for $q_h$ is given below:

$$
q_h = \frac{h \kappa (\mu - R(x)) + R(x) - R_h^-(x)}{R_h^+(x) - R_h^-(x)}
$$

We need to chose the jump sizes $J_h^\pm(x)$
such that the following constraints are satisfied:

\( (Legal\ probability): 0 \leq q_h(x) \leq 1 \)

\( Local\ drift\ convergence: \) As number of periods \( N \) tends to infinity, the local drift should converge to the drift in the SDE.
Constraints

- **Legal probability**
  Following equations have to hold:
  \[
  q_h \leq 1 \\
  h\kappa(\mu - R(x)) + R(x) \leq R_h^+ \\
  q_h \geq 0 \\
  R_h^- \leq h\kappa(\mu - R(x)) + R(x)
  \]

- **Local drift**
  Local drift of the short-rate has to match with the drift in the SDE. Following equation is trivially true:
  \[
  q_h R_h^+ + (1 - q_h) R_h^- - R(x) = h\kappa(\mu - R(x))
  \]
Jump sizes

- The argument to derive jump-sizes is quite technical. See the reference.

- $J_h^+(x)$ is given by:
  the smallest, odd, positive integer $j$ such that
  $$\frac{4h\kappa\mu}{\sigma^2} + x^2(1 - \kappa h) < (x + j\sqrt{h})^2$$

- $J_h^-(x)$ is given by:
  the smallest, odd, positive integer $j$ such that
  $$\frac{4h\kappa\mu}{\sigma^2} + x^2(1 - \kappa h) \geq (x - j\sqrt{h})^2$$
  or $x - j\sqrt{h} \leq 0$. 
Back to MBS

• Assume that we have built the lattice model for the CIR model. Want to price MBSs on it.

• In order to price MBSs need SMMs on all the nodes of the lattice. Once we have the SMMs we can use the Hull-White method (Paper 1) to price MBSs.

• Hull-White use a simple prepayment model where the SMM only depends on the interest rate at the node. The model is simple and unrealistic.

• We will use simulation to estimate SMMs at each node in the lattice.
MBSs on a lattice

- Generate $M$ random paths through the lattice.
- On each node in the path calculate the SMMs. This will be estimated from the prepayment model.
- Let us say a node $N$ is touched $k$ times during the simulation run. Let $SMM_1, \cdots, SMM_k$ be the SMMs at the node $N$ for these $k$ paths. $SMM$ at node $N$ is given by the following equation:

$$\frac{1}{k} \sum_{j=1}^{k} SMM_i$$
MBSs on a lattice (Contd)

- There are some nodes that will not be touched by the simulation runs. What do we do? Perform interpolation/extrapolation to find the SMMs at the node.
- Suppose there is a node $N$ that is not touched by the simulation runs. Find two nearest nodes $N_U$ and $N_L$ such that the following inequality holds:

\[ N_U(r) \leq N(r) \leq N_L(r) \]

Short rate at node $N$ is denoted by $N(r)$.

- SMM for node $N$ is given by the following equation:

\[
SMM(N_L) + \Delta(N(r))(SMM(N_U) - SMM(N_L))
\]
Where $\Delta(N(r))$ is given by the following equation:

$$\Delta(N(r)) = \frac{N(r) - N_L(r)}{N_U(r) - N_L(r)}$$

• What if we can’t find $N_L$ or $N_U$, use extrapolation.

• Rest of the method same as Paper 1.
Hull-White method

- At each node in the interest-rate lattice store $MB_{max}$ and $MB_{min}$.

- **Interpretation**
  $MB_{max}$ ($MB_{min}$) is the maximum (minimum) mortgage balance that can be realized at that node. Notice that there are many paths leading up to a node. On each of these paths the mortgage balance on a node can be different.

- **Forward Induction**
  At the root node $MB_{max}$ and $MB_{min}$ are both the same (equal to the Mortgage balance).
Forward Induction (Contd)

- Suppose we are going to compute $MB_{max}$ and $MB_{min}$ for a node $N$. Let $N_1$ and $N_2$ be the predecessors of this node.

- Let maximum and minimum mortgage balances at nodes $N_1$ and $N_2$ be given by the following quantities:
  
  $MB_{max,1}$, $MB_{min,1}$,
  
  $MB_{max,2}$, $MB_{min,2}$,

- $MB_{max}$ is given by the maximum of the following quantities:

  $$SMM(N)(MB_{max,1} - S_{max,1})$$
  
  $$SMM(N)(MB_{max,2} - S_{max,2})$$

- Similar explanation applies to $MB_{min}$.  

Hull-White method (Contd)

- At each node $N$ we have $MB_{max}$ and $MB_{min}$ (the maximum and minimum possible mortgage balance at that node).
- If $MB_{min} < MB_{min}$, split into $m$ equally spaced values.
- We will write a node as $(r, MB, SMM)$ where $r$ is the short rate, $MB$ the mortgage balance at that node, and $SMM$ determines the prepayment at that node.
- Let the successors of the node $(r, MB, SMM)$ be $(r^+, MB^+)$ and $(r^-, MB^-)$.
- The value of the MBS at node $(r, MB, SMM)$ is given by the backward
equation:

\[ V(r, MB, SMM) = \frac{1}{1+p} (CF + qV(r^+, MB^+) + (1-q)V(r^-, MB^-) \]

**Problem**

Nodes \((r^+, MB^+)\) and \((r^-, MB^-)\) might not exist.

**Solution**

Estimate \(MB^+\) and \(MB^-\) using interpolation. For example, find two nodes \((r_L, MB_L)\) and \((r_U, MB_U)\) such that \(r_L\) and \(r_U\) are nearest to \(r\) and the following equation is true:

\[ r_L \leq r \leq r_U \]

Estimate \(MB^+\) by interpolation.
Additional Information

- Formula given in Lecture 3 was correct.