FM405: Building a Quantitative Mortgage Model

A. Deswal

March 2022

Contents

Introduction

In this assignment, we take the perspective of the back-office at 'University Building Society'(UBS) in January 2019. UBS provides mortgages to university students and employees all over UK, and issues mortgage-backed securities based on those mortgages. We are tasked with pricing a representative 10-year fixed rate mortgage, and mortgage-backed securities backed by an asset pool composed of such mortgages.

We will begin by calibrating interest rate trees using both the Ho-Lee and the Black-Derman-Toy model (***Part a***), and then pricing the mortgage using tree-based (***Part b***) and Monte Carlo (***Part d***) methods. We then incorporate additional prepayment assumptions and revalue the mortgage (***Part e***). Finally, we price a pass-through, principal only, and interest only MBS based on the mortgage priced in preceding parts (***Part f***).

This assignment has been completed on the programming language Python and the source code can be found in the file titled 'Summative 2 code'. This document is meant to be self-contained, but for the sake of brevity certain code chunks may be suppressed here. Whenever that is done, a reference (via the code cell number) would be provided to the source code for ease of verification.

The simulations run in this assignment are not seeded, and so running the codes again during verification will not yield exactly the same results. However, given the sample size, the relative deviation of any new runs will be negligible and the same conclusions will follow.

NOTE ABOUT INDEXING The data structures used to represent the trees of rates and prices will be NumPy matrices. While we denote the rate of period i and state j as $r_{i,j}$, it is stored in in row j and column i of this matrix tree. To avoid confusion, we index rows as m and columns as n in the code. So $j = m$, and $i = n$. That is

$$r_{i,j} = r[j, i] = r[m, n]$$

1 Part (a)

We begin by calibrating binomial trees under the Ho-Lee (HL) and the Black-Derman-Toy (BDT) model. We use the yield curve data (as provided by Bank of England) to calibrate a 20-period tree under each model, with semi-annual ($\Delta = 0.5$) continuously compounded short-rates. We take the historical volatility of the level of short interest rates ($\sigma^{HL} = 1.73\%$) and the log of rates ($\sigma^{BDT} = 21.42\%$), as given.

1.1 The Yield Curve

Following is the yield curve of the UK, as at 2nd January 2019. The column 'Price' denotes the price of a zero coupon bond (ZCB) of face value £100 maturing in $T\Delta$ years, as implied by the yield curve. We will use these prices to calibrate the interest rate trees.

Table 1: UK Yield Curve

Period (T)	**Maturity**	**Spot Rate**	**Price (£)**
1	0.5	0.93%	99.54
2	1.0	0.99%	99.01
3	1.5	1.07%	98.41
4	2.0	1.12%	97.79
5	2.5	1.14%	97.18
6	3.0	1.16%	96.57
7	3.5	1.18%	95.95
8	4.0	1.20%	95.31
9	4.5	1.22%	94.67
10	5.0	1.23%	94.02
11	5.5	1.25%	93.37
12	6.0	1.26%	92.71
13	6.5	1.28%	92.04
14	7.0	1.29%	91.37
15	7.5	1.30%	90.69
16	8.0	1.32%	90.01
17	8.5	1.33%	89.32
18	9.0	1.34%	88.63
19	9.5	1.35%	87.93
20	10.0	1.37%	87.23
21	10.5	1.38%	86.53

1.2 Calibrating the Ho-Lee Tree

The discrete-time version of the Ho-Lee model specifies the following behaviour of interest rates

$$r_{\text{PERIOD,STATE}} = r_{i,j}$$

$$r_{i,j} \begin{cases} r_{i+1,j} & \text{'up' state with risk-neutral probability } 1/2 \\ r_{i+1,j+1} & \text{'down' state with risk-neutral probability } 1/2 \end{cases}$$

$$r_{i+1,j} = r_{i,j} + \theta_i^{HL}\Delta + \sigma^{HL}\sqrt{\Delta}$$
$$r_{i+1,j+1} = r_{i,j} + \theta_i^{HL}\Delta - \sigma^{HL}\sqrt{\Delta}$$

Calibrating an interest rate tree using the Ho-Lee Model effectively amounts to finding the values of the period-contingent parameter, θ_i, such that the resulting tree 'matches' the yield curve. An interest rate tree that 'matches' the yield curve is one that produces the observed market prices of the benchmark securities (the ZCB's, in our case) when these securities are priced on the tree.

Recall that given an interest rate tree, the price of a T-period[1] ZCB in period i and state j, $V_{i,j}(T)$ is:-

$$V_{i,j}(T) = \begin{cases} 100 & \text{if } i = T \\ 0.5e^{-\Delta\ r_{i,j}}(V_{i+1,j}(T) + V_{i+1,j+1}(T)) & \text{otherwise} \end{cases} \tag{1}$$

i.e., the discounted risk-neutral expected value of the two possible states in the next period. The probabilities used in this expectation are $P(up) = P(down) = 1/2$, as specified by the model. The price of the ZCB according to the tree is $V_{0,0}(T)$, found by iterating backwards through the tree. In Python we declare this as[2]

```
def pricer(T,model):
    tree = np.zeros((T+1,T+1)) # data structure to hold the tree
    tree[:,T] = 100 # at maturity
    for n in range(T-1,-1,-1): # iterate back from the end of the
    ↪  tree
        for m in range(n+1): # iterate down each column (period)
            if model : # HL tree used
                tree[m,n] = 0.5*exp(-HL_tree[m,n]*delta)
                ↪  *(tree[m,n+1]+tree[m+1,n+1]) #formula above
            else :     # BDT tree used
```

[1] A 'period' is 6 months, for our purpose

[2] The function in this particular form is not used in the code, but forms the foundation of all subsequent tree pricing functions used

```
            tree[m,n] = 0.5*exp(-BDT_tree[m,n]*delta)
             ↪  *(tree[m,n+1]+tree[m+1,n+1]) # formula above
    return tree # returns the whole price tree
```

Note from Equation 1, it is clear that the tree price of the T-period ZCB, is a function of all the state rates in the preceding periods. Formally,

$$V_{0,0}(T) = f(r_{0,0}, \boldsymbol{r_1}, ..., \boldsymbol{r_{T-1}}|T) \qquad \text{for } T \geq 2 \tag{2}$$

where $\boldsymbol{r_i}$ is the vector of state rates in period i. Next, we observe that $r_{i,j}$ can be expressed as a function of i and j in the following manner[3]

$$r_{i,j} = r_{0,0} + \Delta \sum_{l=0}^{i-1} \theta_l^{HL} + (i-2j)\sigma^{HL}\sqrt{\Delta} \qquad \text{for } i \geq 1 \tag{3}$$

This alludes to the fact that we can express $r_{i,j}$ as a function of the θ^{HL}'s in the previous periods. That is,

$$\begin{aligned} r_{i,j} &= g_{i,j}(\theta_0^{HL}, \theta_1^{HL}, ..., \theta_{i-1}^{HL}) \\ &= g_{i,j}(\boldsymbol{\Theta_{i-1}^{HL}}) \qquad \text{for } i \geq 1 \end{aligned}$$

Where $\boldsymbol{\Theta_{i-1}^{HL}}$ is a vector of θ_l^{HL} of length $i-1$. So,

$$\boldsymbol{r_i} = \boldsymbol{g_i}(\boldsymbol{\Theta_{i-1}^{HL}}) \qquad \text{for } i \geq 1$$

This gives us

$$V_{0,0}(T) = f(r_{0,0}, \boldsymbol{g_1}(\boldsymbol{\Theta_0^{HL}}), ..., \boldsymbol{g_{T-1}}(\boldsymbol{\Theta_{T-2}^{HL}})|T) \qquad \text{for } T \geq 2 \tag{4}$$

A calibrated $\hat{T}$-period tree requires that

$$V_{0,0}(T) - Price(T) = 0 \quad \forall\ T \in 2, 3, ...\hat{T} \tag{5}$$

Where $Price(T)$ is the market-observed price of a T-period ZCB. We solve this problem sequentially, calibrating θ_0^{HL} using $Price(2)$, and then iterating forward through T to $\hat{T}$. These problems are generally not analytically solvable and we must resort to numerical methods to obtain solutions.

1.2.1 Developing the calibration algorithm for the Ho-Lee Tree

Here, we will make the calibration idea operational. To begin, we express Equation 3 as an explicit function of the particular θ we are aiming to calibrate in each iteration.

[3]See Appendix I 7.1 for derivation and Python code

$$r_{i,j} = \begin{cases} \hat{\theta}_0^{HL}\Delta + r_{0,0} + (i-2j)\sigma^{HL}\sqrt{\Delta} & \text{if } i = 1 \\ \hat{\theta}_{i-1}^{HL}\Delta + r_{0,0} + \Delta\sum_{l=0}^{i-1}\theta_{l-1}^{HL} + (i-2j)\sigma^{HL}\sqrt{\Delta} & \text{otherwise} \end{cases} \tag{6}$$

Equation 6 is a specific expression for the general $\boldsymbol{r_i} = \boldsymbol{g_i}(\boldsymbol{\Theta_{i-1}^{HL}})$ discussed before. In Python we declare this function in the following manner

```
def r_HL_c(theta_target,m,n):
    s = delta*theta_target+yields[0]+(n-2*m)*sigma_HL*sqrt(delta)
    if n != 1:
        s += delta*sum(theta_HL[0:n-1])
    return s
```

We do this so that we can used the express the pricing function in terms of the θ_{i-1}^{HL} that is the target of our calibration. We do this by declaring a 'calibrator function'. For a T-period ZCB, the calibrator produces the difference between the price implied by the tree and the market observed price. This will help us apply root-finding algorithms to calibrate the tree (in line with Equation 5). The Ho-Lee calibrator function in Python is declared as follows

```
def HL_calibrator(theta_target,T):
    tree = np.zeros((T+1,T+1))
    tree[:,T] = 100
    for n in range(T-1,-1,-1):
        for m in range(n+1):
            if n == T-1:
                tree[m,n] = 0.5*exp(-r_HL_c(theta_target,m,n)*
                ↪  delta)*(tree[m,n+1]+tree[m+1,n+1])
            else:
                tree[m,n] = 0.5*exp(-HL_tree[m,n]*delta)*
                ↪  (tree[m,n+1]+tree[m+1,n+1])

    return tree[0,0]-prices[T-1]
```

Below is the code for the iterative routine that calibrates θ_0^{HL} to θ_{19}^{HL} using the secant method[4]. This also generates the interest rate tree using the code for Equation 3.

```
HL_tree[0,0] = yields[0] # root node is observed rate currently
for l in range(2,len(yields)+1):
    theta_HL[l-2] = optimize.newton(HL_calibrator,0.01,args=[l])
    ↪  # generating thetas
```

[4]The numerical method is called by the `optimize.newton` function from the `scipy` library

```
for m in range(0,l):
    HL_tree[m,l-1] = r_HL(m,l-1) # generating the tree
```

Table 2 displays the calibrated Ho-Lee interest rate tree.

1.3 Calibrating the Black-Derman-Toy Tree

The Black-Derman-Toy model specifies the same law of motion as the Ho-Lee model, but rather for the logarithm of interest rates than interest rates themselves. This helps the Black-Derman-Toy model to overcome a major drawback of the Ho-Lee model, which is that negative rates occur with positive probability. By specifying dynamics for the log of rates, the Black-Derman-Toy model is able to enforce a zero lower bound on the interest rate. Formally,

$$z_{i,j} = ln(r_{i,j})$$

$$z_{i,j} \begin{cases} z_{i+1,j} & \text{'up' state with risk-neutral probability } 1/2 \\ z_{i+1,j+1} & \text{'down' state with risk-neutral probability } 1/2 \end{cases}$$

$$\begin{aligned} z_{i+1,j} &= z_{i,j} + \theta_i^{BDT}\Delta + \sigma^{BDT}\sqrt{\Delta} \\ z_{i+1,j+1} &= z_{i,j} + \theta_i^{BDT}\Delta - \sigma^{BDT}\sqrt{\Delta} \end{aligned}$$

Observing Equation 3, we can conclude that

$$z_{i,j} = z_{0,0} + \Delta\sum_{l=0}^{i-1}\theta_l^{BDT} + (i-2j)\sigma^{BDT}\sqrt{\Delta} \qquad \text{for } i \geq 1$$

So,

$$r_{i,j} = e^{(ln(r_{0,0})+\Delta\sum_{l=0}^{i-1}\theta_l^{BDT}+(i-2j)\sigma^{BDT}\sqrt{\Delta})} \qquad \text{for } i \geq 1 \tag{7}$$

Equation 7 is another specific expression for $\boldsymbol{r_i} = \boldsymbol{g_i}(\boldsymbol{\Theta_{i-1}^{BDT}})$. This allows us to again state the pricing function in terms of θ_i^{BDT}, which allows us to follow exactly the same procedure as Section 1.2 to calibrate the interest rate tree to the yield curve[5]. Table 3 displays the tree fitted using the Black-Derman-Toy model.

[5]Codes for this subsection are suppressed, as they are very similar to the previous section. They can be found in code cells 8-11 of the source code document

Table 2: The Calibrated Ho-Lee Tree

Maturity	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
Period	0	1	2	3	4	5	6	7	8	9	10
$100\theta_i^{HL}$	0.249	0.376	0.082	0.05	0.106	0.138	0.153	0.164	0.174	0.187	0.203
j											
0	0.93%	2.28%	3.69%	4.95%	6.20%	7.48%	8.77%	10.07%	11.38%	12.69%	14.00%
1		-0.17%	1.24%	2.51%	3.76%	5.03%	6.32%	7.62%	8.93%	10.24%	11.56%
2			-1.20%	0.06%	1.31%	2.59%	3.88%	5.18%	6.48%	7.79%	9.11%
3				-2.39%	-1.14%	0.14%	1.43%	2.73%	4.04%	5.35%	6.66%
4					-3.58%	-2.31%	-1.02%	0.28%	1.59%	2.90%	4.22%
5						-4.75%	-3.46%	-2.16%	-0.86%	0.45%	1.77%
6							-5.91%	-4.61%	-3.30%	-1.99%	-0.68%
7								-7.06%	-5.75%	-4.44%	-3.12%
8									-8.20%	-6.89%	-5.57%
9										-9.33%	-8.02%
10											-10.46%
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											

Maturity	5.0	5.5	6.0	6.5	7.0	7.5	8.0	8.5	9.0	9.5	10.0
Period	10	11	12	13	14	15	16	17	18	19	20
$100\theta_i^{HL}$	0.203	0.218	0.234	0.25	0.264	0.278	0.29	0.302	0.312	0.322	
j											
0	14.00%	15.33%	16.66%	18.00%	19.35%	20.70%	22.07%	23.43%	24.81%	26.19%	27.57%
1	11.56%	12.88%	14.21%	15.55%	16.90%	18.26%	19.62%	20.99%	22.36%	23.74%	25.13%
2	9.11%	10.43%	11.77%	13.11%	14.46%	15.81%	17.17%	18.54%	19.92%	21.30%	22.68%
3	6.66%	7.99%	9.32%	10.66%	12.01%	13.36%	14.73%	16.09%	17.47%	18.85%	20.23%
4	4.22%	5.54%	6.87%	8.21%	9.56%	10.92%	12.28%	13.65%	15.02%	16.40%	17.79%
5	1.77%	3.09%	4.43%	5.77%	7.12%	8.47%	9.83%	11.20%	12.58%	13.96%	15.34%
6	-0.68%	0.65%	1.98%	3.32%	4.67%	6.02%	7.39%	8.76%	10.13%	11.51%	12.89%
7	-3.12%	-1.80%	-0.47%	0.87%	2.22%	3.58%	4.94%	6.31%	7.68%	9.06%	10.45%
8	-5.57%	-4.24%	-2.91%	-1.57%	-0.22%	1.13%	2.49%	3.86%	5.24%	6.62%	8.00%
9	-8.02%	-6.69%	-5.36%	-4.02%	-2.67%	-1.32%	0.05%	1.42%	2.79%	4.17%	5.55%
10	-10.46%	-9.14%	-7.81%	-6.47%	-5.12%	-3.76%	-2.40%	-1.03%	0.34%	1.72%	3.11%
11		-11.58%	-10.25%	-8.91%	-7.56%	-6.21%	-4.85%	-3.48%	-2.10%	-0.72%	0.66%
12			-12.70%	-11.36%	-10.01%	-8.65%	-7.29%	-5.92%	-4.55%	-3.17%	-1.79%
13				-13.80%	-12.46%	-11.10%	-9.74%	-8.37%	-7.00%	-5.62%	-4.23%
14					-14.90%	-13.55%	-12.19%	-10.82%	-9.44%	-8.06%	-6.68%
15						-15.99%	-14.63%	-13.26%	-11.89%	-10.51%	-9.13%
16							-17.08%	-15.71%	-14.34%	-12.96%	-11.57%
17								-18.16%	-16.78%	-15.40%	-14.02%
18									-19.23%	-17.85%	-16.47%
19										-20.30%	-18.91%
20											-21.36%

Table 3: The Calibrated Black-Derman-Toy Tree

Maturity	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
Period	0	1	2	3	4	5	6	7	8	9	10
$100\delta_i^{BDT}$	22.077	28.794	1.367	-2.37	0.884	2.253	2.188	1.804	1.442	1.318	1.339
j											
0	0.93%	1.21%	1.63%	1.90%	2.19%	2.56%	3.01%	3.54%	4.16%	4.87%	5.71%
1		0.89%	1.20%	1.41%	1.62%	1.89%	2.22%	2.62%	3.07%	3.60%	4.22%
2			0.89%	1.04%	1.19%	1.40%	1.64%	1.93%	2.27%	2.66%	3.12%
3				0.77%	0.88%	1.03%	1.21%	1.43%	1.68%	1.96%	2.30%
4					0.65%	0.76%	0.90%	1.05%	1.24%	1.45%	1.70%
5						0.56%	0.66%	0.78%	0.91%	1.07%	1.26%
6							0.49%	0.58%	0.68%	0.79%	0.93%
7								0.43%	0.50%	0.58%	0.68%
8									0.37%	0.43%	0.51%
9										0.32%	0.37%
10											0.28%
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											

Maturity	5.0	5.5	6.0	6.5	7.0	7.5	8.0	8.5	9.0	9.5	10.0
Period	10	11	12	13	14	15	16	17	18	19	20
$100\delta_i^{BDT}$	1.339	1.395	1.45	1.477	1.459	1.382	1.248	1.061	0.824	0.544	
j											
0	5.71%	6.69%	7.84%	9.18%	10.76%	12.62%	14.78%	17.31%	20.24%	23.65%	27.59%
1	4.22%	4.94%	5.79%	6.78%	7.95%	9.32%	10.92%	12.78%	14.95%	17.47%	20.38%
2	3.12%	3.65%	4.28%	5.01%	5.87%	6.88%	8.06%	9.44%	11.05%	12.90%	15.06%
3	2.30%	2.70%	3.16%	3.70%	4.34%	5.08%	5.96%	6.97%	8.16%	9.53%	11.12%
4	1.70%	1.99%	2.33%	2.73%	3.20%	3.76%	4.40%	5.15%	6.03%	7.04%	8.21%
5	1.26%	1.47%	1.72%	2.02%	2.37%	2.77%	3.25%	3.81%	4.45%	5.20%	6.07%
6	0.93%	1.09%	1.27%	1.49%	1.75%	2.05%	2.40%	2.81%	3.29%	3.84%	4.48%
7	0.68%	0.80%	0.94%	1.10%	1.29%	1.51%	1.77%	2.08%	2.43%	2.84%	3.31%
8	0.51%	0.59%	0.69%	0.81%	0.95%	1.12%	1.31%	1.53%	1.79%	2.10%	2.45%
9	0.37%	0.44%	0.51%	0.60%	0.70%	0.83%	0.97%	1.13%	1.33%	1.55%	1.81%
10	0.28%	0.32%	0.38%	0.44%	0.52%	0.61%	0.71%	0.84%	0.98%	1.14%	1.33%
11		0.24%	0.28%	0.33%	0.38%	0.45%	0.53%	0.62%	0.72%	0.84%	0.99%
12			0.21%	0.24%	0.28%	0.33%	0.39%	0.46%	0.53%	0.62%	0.73%
13				0.18%	0.21%	0.25%	0.29%	0.34%	0.39%	0.46%	0.54%
14					0.15%	0.18%	0.21%	0.25%	0.29%	0.34%	0.40%
15						0.13%	0.16%	0.18%	0.22%	0.25%	0.29%
16							0.12%	0.14%	0.16%	0.19%	0.22%
17								0.10%	0.12%	0.14%	0.16%
18									0.09%	0.10%	0.12%
19										0.07%	0.09%
20											0.06%

2 Part (b)

As a benchmark, we assume that the buyers of the mortgage (of principal value £100,000) will prepay optimally and will not default. We proceed to find the fixed rates for which the value of the 10-year mortgage is (approximately) par under the Ho-Lee model and the Black-Derman-Toy model.

2.1 Valuing a Residential Mortgage

A homeowner with a fixed rate mortgage can be viewed as an issuer of a callable bond, where the straight bond component is the mortgage without any prepayment option ($V_{0,0}^{np}$), and the call option component is the prepayment option ($C_{0,0}$) which allows the homeowner to close their mortgage at any time by repaying the outstanding principal on the borrowing. The primary reason they may choose to do so is the availability of lower refinancing rate.

The prepayment option is undesirable for the lender as it causes reinvestment risk, i.e., they have to lend out the prepaid capital at a potentially lower rate (as that is the environment in which prepayments typically occur). So the value of the mortgage is

$$V_{0,0} = V_{0,0}^{np} - C_{0,0} \tag{8}$$

The par rate is that rate which ensures the mortgage value is equal to the principal. Using the par rate r_M we can find the 'coupon', which is the fixed periodic payment that the homeowner makes to fulfil their debt obligation. The coupon is comprised of both interest and scheduled principal repayment. It is calculated as[6]

$$\bar{C} = \frac{FV r_M \Delta}{(1 - \frac{1}{(1+r_M\Delta)^T})} \tag{9}$$

Where FV is the face value of the mortgage and T is the periods to maturity In Python, we define a function for the coupon given below

```
def coupon(r,FV,T):
    return ((FV*r*delta)/(1-(1+r*delta)**-T))
```

Each period, the coupon consists of

- interest payment I_i, which is $r_M \Delta OP_{i-1}$, where OP_{i-1} is the opening outstanding principal
- principal repayment L_i which is $\bar{C} - I_i$. This is deducted to from OP_{i-1} find the closing outstanding principal for period i

The evolution of interest payment, principal repayment and outstanding principal over time forms the repayment schedule. Since these are important inputs in pricing the mortgage, it is useful to generate a matrix that captures this

[6] Standard annuity formula

schedule. In Python, the code for the function that generates the repayment schedule is

```
# for the matrix, row 1,2, and 3 are interest paid, principal
↪  paid, and outstanding principal respectively
def os_principal(r,FV,T):
    matrix = np.zeros((3,T+1))
    matrix[2,0] = FV
    for n in range(1,T+1):
        matrix[0,n] = r*delta*matrix[2,n-1]
        matrix[1,n] =
        ↪  coupon(r,principal,len(yields)-1)-matrix[0,n]
        matrix[2,n] = matrix[2,n-1]-matrix[1,n]
    return matrix
```

2.1.1 Setting up pricing functions for the components of the mortgage

Here, we will set up the functions for pricing the components. This will help us express the mortgage value as a function of the par rate, which we then optimise for. Equation 1 and its code provides a basis for the building tree pricing functions here.

Mortgage value without prepayment is simply the risk-neutral expected present value of all future payments ($\bar{C}$) the mortgage promises to pay. On a tree,

$$V_{i,j}^{np}(T) = \begin{cases} 0 & \text{if } i = T \\ e^{-\Delta\ r_{i,j}}(0.5V_{i+1,j}^{np}(T) + 0.5V_{i+1,j+1}^{np}(T) + \bar{C}) & \text{otherwise} \end{cases} \tag{10}$$

The value is zero at $i = T$ as the mortgage has been fully paid down. In Python, this function is

```
def m_without_p(r,FV,T,model):
    tree = np.zeros((T+1,T+1))
    for n in range(T-1,-1,-1):
        for m in range(n+1):
            if model :
                tree[m,n] = exp(-HL_tree[m,n]*delta)*
                ↪  (0.5*(tree[m,n+1]+tree[m+1,n+1])
                ↪  +coupon(r,FV,T))
            else :
                tree[m,n] = exp(-BDT_tree[m,n]*delta)
                ↪  *(0.5*(tree[m,n+1]+tree[m+1,n+1])
                ↪  +coupon(r,FV,T))
```

```
    return tree
```

Value of the prepayment option A homeowner chooses to exercise the option when the value of their mortgage exceeds the outstanding principal on the borrowing (the strike of the option). That is, the payoff by prepaying is

$$C_{i,j}^{Ex} = max(V_{i,j}^{np}(T) - L_i(T), 0)$$

In Python, the following function generates the exercise payoff tree

```
def p_ex(r,FV,T,model):
    tree = np.zeros((T+1,T+1))
    for n in range(T-1,-1,-1):
        for m in range(n+1):
            tree[m,n] = max(m_without_p(r,FV,T,model)[m,n]-
            ↪  os_principal(r,FV,T)[2,n],0)
    return tree
```

However, for exercise to be optimal, the homeowner must also consider the value of waiting, which is

$$C_{i,j}^{Wait} = 0.5e^{-r_{i,j}\Delta}(C_{i+1,j} + C_{i+1,j+1})$$

So, it is optimal for the homeowner to prepay when $C_{i,j}^{Ex} > C_{i,j}^{Wait}$. This gives us the following rule to iterate backwards through the tree for the value of the prepayment option

$$C_{i,j} = max(C_{i,j}^{Ex}, C_{i,j}^{Wait})$$

We define the option value function in Python as

```
def prepayment_value(r,FV,T,model):
    tree = np.zeros((T+1,T+1))
    for n in range(T-1,-1,-1):
        for m in range(n+1):
            if model:
                tree[m,n] = max(p_ex(r,FV,T,model)[m,n],
                ↪  exp(-HL_tree[m,n]*delta)
                ↪  *(0.5*(tree[m,n+1]+tree[m+1,n+1])))
            else:
                tree[m,n] = max(p_ex(r,FV,T,model)[m,n],
                ↪  exp(-BDT_tree[m,n]*delta)
                ↪  *(0.5*(tree[m,n+1]+tree[m+1,n+1])))
    return tree
```

Note that the option value is zero at maturity, as there is no outstanding balance left to prepay.

2.1.2 Optimising for the par rate

Having defined these functions which take r_M as an input, we are now in a position to optimise for par rates by defining a function whose root is the solution we are looking for. In, Python we do it in the following manner

```
x = lambda r,FV,T,model : m_without_p(r,FV,T,model)[0,0] -
↪  prepayment_value(r,FV,T,model)[0,0] - FV
r_M_HL = optimize.newton(x,0.031,args =
↪  (principal,len(yields)-1,True),tol = 0.001)
r_M_BDT = optimize.newton(x,0.015,args =
↪  (principal,len(yields)-1,False),tol = 0.001)
```

Here, x sets Equation 8 equal to the principal value, and the subsequent lines apply the secant method to find r_M^{HL} and r_M^{BDT}, which are the par rates under the two models. Note that this optimisation problem is computationally intensive, since x is a function of several functions which may further be functions of other functions. This is why we choose a tolerance level of the deviation of 0.001. A smaller tolerance increases the run-time of the algorithm without a commensurate improvement in the estimate. We find the following estimates of the par rate[7]

Table 4: Par Rates under both Models

r_M^{HL}	r_M^{BDT}
3.138%	1.527%

2.2 Difference between Par Rates under both Models

We find that the par rate in the Ho-Lee model is almost double than what we find under the Black-Derman-Toy model. This is because the Black-Derman-Toy model assigns higher risk-neutral probabilities to larger interest rate realisations compared to the Ho-Lee model. We can see this from the trees in Tables 2 and 3. Note how the rate in the upmost state is similar for both models, whereas the lowest rate for the Ho-Lee model is -21.36% compared to a much higher 0.06% for the Black-Derman-Toy model. This means that the density for positive rates is one for Black-Derman-Toy model, but less than one for Ho-Lee model.

This matters because a higher likelihood of high interest rate environments means a lower likelihood of prepayment, since it would be difficult for homeowners to find attractive refinancing rates. This is desirable for the bank, the risk premium (captured in the rate) demanded by them would be much lower

[7]Please see Appendix II for repayment schedules and mortgage value trees under the two models

under the assumptions of the Black-Derman-Toy model, as compared to the Ho-Lee model.

3 Part (c)

Now, we switch to the Monte Carlo methodology. Using 100,000 simulations, we plot a histogram of the simulated interest rates in year 10 under each of the short-rate models.

3.1 Simulation Methodology

To simulate $N = 100,000$ interest rate paths over 20 periods, we generate a sequence of 20 i.i.d. discrete uniform random variables that take the value of either zero or one. A realisation of zero in period i corresponds to an up move while that of one corresponds to a down move. So, each sequence represents a realisation of a sample path of rates.

From the specification of both models, we see that the trees implied are recombining. This means that the interest rate realised at the terminal nodes only depends on the number of up and down moves, and not the order in which they are realised.

This fact is useful as it means that we can sum across this sequence of random variables to obtain what 10-year state rate we realise for a particular sample path s. More formally, let $\{X_i^s\}_{i=1}^{20}$ be a sequence of realisations, $X_i^s \sim unif\{0,1\}$, then

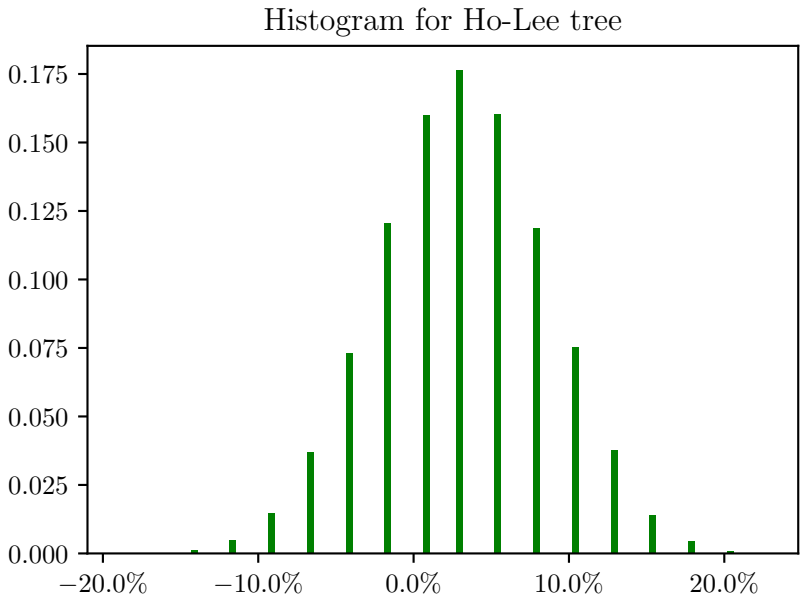
$$r_{20,j}^s = r_{20,\lambda^s}^s \tag{11}$$

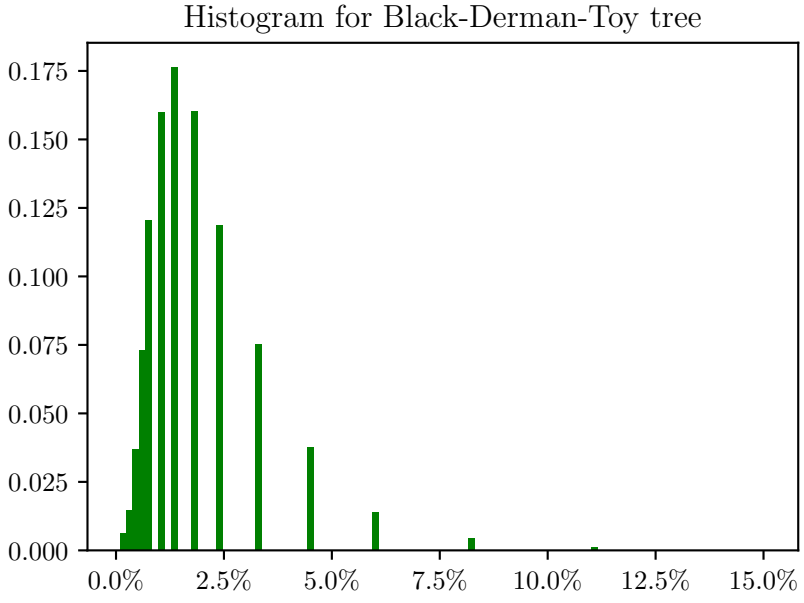
where $\lambda^s = \sum_{i=1}^{20} X_i^s$

3.2 Running the Simulations and producing the Histograms

Following is the code which runs the 100,000 simulations for the sample paths, below which are the histograms of the year 10 rate realisations from the simulation for the Ho-Lee and the Black-Derman-Toy model.

```
HL_realization = np.zeros(N) # vector of len 100,000 to hold rate
↪  realisations
BDT_realization = np.zeros(N)
for i in range(N):
    m = sum(np.random.randint(0,2,len(yields)-1)) # random vector
    ↪  of  20 0s and 1s (sequence)
    HL_realization[i] = HL_tree[m,HL_tree.shape[1]-1]
    ↪  #corresponding rate added to realisation vector
    BDT_realization[i] = BDT_tree[m,BDT_tree.shape[1]-1]
```


Histogram for Ho-Lee tree
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−20.0%
−10.0%
0.0%
10.0%
20.0%


Histogram for Black-Derman-Toy tree
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0%
2.5%
5.0%
7.5%
10.0%
12.5%
15.0%

3.3 Intuition for the Difference in the Histograms of both Models

We know that the Ho-Lee model outlined in section 1.2 is a discrete time approximation of an Îto process which is driven by an arithmetic Brownian motion. This means the stochastic process underlying the interest rate dynamics is Normal. Since our simulation generates a large sequence of year 10 rate realisations, the sequence converges in distribution[8] to the Normal distribution. This is evident from the shape of the histogram.

The Black-Derman-Toy model has the same dynamics for log of rate, and so the distribution of log rates converges to Normal. Consequently, the distribution of rates converges to a Log-normal distribution. The histogram of this model approximates the curve for the Log-normal distribution.

[8]Please see Appendix I 7.2 for the formal statement of this concept

4 Part (d)

We proceed with the Black-Derman-Toy model from here. Using the corresponding mortgage rate found in Part (b), we revalue the mortgage contract using a Monte Carlo methodology.

4.1 The Prepayment Decision in a Monte Carlo Setting

As discussed before, the homeowner choosing optimal exercise would prepay in a period if the (refinancing) rate is suitably low. In our context, we can express this as an exercise of the prepayment option if the interest rate realised is below a certain threshold rate r_i^*. This prepayment decision can be observed on the mortgage value tree[9]. For any period i, as we move down to lower rates, the value of the mortgage increases until it hits a constant value below which it does not move. This constant value is the outstanding principal[10] for that period, an indication that prepayment has occurred. The state rate corresponding to when this threshold value is first reached in a period is the trigger rate for this period. We use to following code to set up the vector of trigger rates

```
trigger = np.zeros(BDT_tree.shape[1])
for n in range(1,BDT_tree.shape[1]):
    for m in range(n+1):
        if mortgage_value_BDT[m,n] ==
        ↪  os_principal(r_M_BDT,principal,BDT_tree.shape[1]-1)[2,n]:
        ↪  # check for prepayment
            trigger[n] = BDT_tree[m,n] # if prepayment occurs,
            ↪  captures corresponding state rate
            break
```

4.2 Simulating the Sample Interest Rate Paths

We follow the same methodology outlined in Part (c) to generate path realisations. However, we are now interested in the full path of evolution of rates rather than the terminal value. Nonetheless, we can generalise the idea behind Equation 11 to find that given a sequence of random variables $\{X_i^s\}_{i=1}^{20}$, the interest rate path realised is $\{r^s_{i,\lambda_i^s}\}$, where $\lambda_{\hat{i}}^s = \sum_{i=1}^{\hat{i}} X_i^s$ in period $\hat{i}$. The code for generating 100,000 rate path realisations is

```
rate_path = np.zeros((N,len(yields)))
for m in range(rate_path.shape[0]):
    path = np.random.randint(0,2,20) # generating sequence of
    ↪  zeros and ones
```

[9] See trees in Appendix II
[10] See Table 8 in Appendix II for the repayment schedule

```
    for n in range(rate_path.shape[1]):
        rate_path[m,n] = BDT_tree[sum(path[0:n]),n] # generating
        ↪  rate path
```

4.3 Valuing the Mortgage through Monte Carlo

Given a realised interest rate path, the mortgage pays the fixed coupon, unless the trigger rate threshold is breached ($r_i^s \leq r_i^*$). If that happens, the outstanding principal is prepaid and cash flow in subsequent periods is zero. Finding the sum of the present value of all cash flows along this path gives us the value of the mortgage for this particular sample path. In Python, this is done for $100,000$ paths using the following code.

```
DCF_path = np.zeros((N,len(yields)))
for m in range(DCF_path.shape[0]):
    for n in range(1,DCF_path.shape[1]):
        DCF_path[m,n] = exp(-sum(rate_path[m,0:n])*delta)
        ↪  *coupon(r_M_BDT,principal,len(yields)-1) # finds the
        ↪  discounted value of CF occurring in period i of path
        ↪  s
        if rate_path[m,n] <= trigger[n]: # checks for prepayment
            DCF_path[m,n] += exp(-sum(rate_path[m,0:n])*delta)
            ↪  *repay_BDT[2,n] # prepays o/s principal and
            ↪  discounts it
            break
```

Averaging the estimates across paths gives us a point estimate of the value of the mortgage (V^{MC}). That is

$$V^{MC} = \frac{1}{N} \sum_{s=1}^{N} \sum_{i=1}^{20} DCF_i^s \tag{12}$$

where $N = 100,000$ and DCF_i^s is the discounted value of the cash flow occurring in period i of sample path s. Further, we can rely on the Central Limit Theorem[11], to assume that our price estimate is drawn from a Normal distribution (as it is an average). This allows us to compute confidence intervals for our price estimates. The 95% confidence intervals are computed as $(V^{MC} - 1.96SE, V^{MC} + 1.96SE)$, where 1.96 is the critical value of the Standard Normal distribution and SE is the standard error of our estimate.

4.4 Simulation Results

The results of the simulation are presented below. The column 'Tree Price' is the value of the mortgage from the tree with $r_M^{HL} = 1.527\%$ and optimal prepayment

[11] Please see Appendix I 7.3 for the formal statement of this theorem

assumption, and 'Price' is the simulated price. The lower and upper bounds of the 95% confidence interval are in the final 2 columns. We find that the results of our simulation are consistent with our answers in part (b). The simulated price is very close to the tree price, which is also within the 95% confidence interval.

Table 5: Simulation Results with Optimal Prepayment (£)

Simulations	Tree Price	Price	S.E.	Lower(95%)	Upper(95%)
100000	100000	100000.61	3.47	99993.82	100007.41

5 Part (e)

Continuing continue with the Black-Derman-Toy model and Monte Carlo methodology, we incorporate additional assumptions on prepayment.

- *We recognise that sometimes mortgages are prepaid even if it is not optimal. This typically happens when people move. Further, people tend to move more often during the summer months and tend to repay less often at the beginning of the contract. To find the probability p_i that a mortgage is prepaid in period i even when it is not optimal, we use the PSA standard prepayment model. In particular, we use 50% PSA probability and assume that it doubles during the summer. That is,*

$$p_i = season\ index(1 - (1 - 0.5CPR_i)^{\Delta}) \tag{13}$$

  where 'season index' is 1 or 2, depending whether the period is odd or even and $CPR_i = min\ (2.4\Delta i\%,\ 6\%)$ is the Conditional Prepayment Rate

- *We also recognise that that for various reasons, borrowers might not prepay in periods when it would be optimal to do so. They prepay with larger probability when interest rates are low, but probability may not be 1. Specifically, we assume the probability to prepay in period i where prepayment is optimal is*

$$q_i = ae^{-br_i} \tag{14}$$

  We assume $a = 0.8$ and $b = 20$

5.1 Functions for new Prepayment assumptions

We declare the following functions for p_i and q_i in Python as follows

```
def p(i): # Random Prepayment
    prob = (1-(1-0.5*min(12*i*delta*0.2,6)/100)**delta)
    if i%2 == 0: # season index
        prob *= 2
    return prob

def q(r): # Non-optimal Prepayment
    return 0.8*exp(-20*r)
```

5.2 Simulating Cash Flow Paths

We will use the same set of realised paths from part (d) and build on the methodology developed in section 4.3. A stochastic component has now been added to the prepayment decision. This is because,

- Prepayment occurs when non-optimal with probability $p_i > 0$

- Prepayment only occurs when optimal with probability $q_i < 1$

We incorporate these assumptions while generating cash flow paths in the following manner. For a given interest rate path realisation,

1. We generate a random variable from the continuous uniform distribution in each period (called p_i^{test}) which is compared to p_i. If $p_i^{test} < p_i$, a prepayment is deemed to have occurred.
2. For periods when the trigger threshold is breached (and $p_i^{test} > p_i$), another random variable is generated (called q_i^{test}) which is compared to q_i. Prepayment only occurs if $q_i^{test} < q_i$.
3. If neither conditions are satisfied, the coupon payments occur as usual.

We use the following routine in Python to generate a discounted cash flow realisation matrix with additional prepayment assumptions

```
DCF_path_2 = np.zeros((N,len(yields)))
for m in range(DCF_path_2.shape[0]):
    for n in range(1,DCF_path.shape[1]):
        p_test = np.random.random()
        DCF_path_2[m,n] = exp(-sum(rate_path[m,0:n])*delta)
        ↪  *coupon(r_M_BDT,principal,len(yields)-1)
        if p_test < p(n): # check random prepayment
            DCF_path_2[m,n] +=
            ↪  exp(-sum(rate_path[m,0:n])*delta)*repay_BDT[2,n]
            break
        else:
            if rate_path[m,n] <= trigger[n]:
                q_test = np.random.random()
                if q_test < q(rate_path[m,n]): #check non-optimal
                ↪  prepayment
                    DCF_path_2[m,n] +=
                    ↪  exp(-sum(rate_path[m,0:n])*delta)
                    ↪  *repay_BDT[2,n]
                    break
```

Results of this procedure are captured in Table 6.

5.3 Simulation Results

Table 6: Simulation Results with non-optimal Prepayment (£)

Simulations	Tree Price	Price	S.E.	Lower(95%)	Upper(95%)
100000	100000	100151.74	3.33	100145.21	100158.27

We find that the simulated price of the mortgage under these assumption is higher than that under optimal repayment. Further, this result is statistically significant, and the 95% confidence interval for the simulation does not contain the tree price.

We can understand this result in the following manner. By not choosing to exercise the prepayment option optimally, the homeowner decreases its value. Since the bank is 'short' this option implicit in the mortgage, the value of the mortgage on the bank's book goes up. More specifically, the random prepayment decreases the value of the mortgage, whereas the non-prepayment when optimal increases the value. In our context, the latter force dominates

5.4 Recommending a Mortgage Rate

Given the simulation results above, we can use the optimisation procedure outlined in part (b) to find an adjusted par rate consistent with the new simulated price estimate. The following table presents this rate, as well as a 95% confidence interval implied by the simulation results.

Table 7: Adjusted Par Rate

Adjusted Rate	Lower (95%)	Upper (95%)
1.5162%	1.5158%	1.5166%

As expected, using the higher simulated price with additional prepayment assumptions decreases the implied par rate. However, to recommend a mortgage rate considering these results, one must also consider the following factors:

1. **Pricing methodology and Risk Aversion:** While we use the assumption of risk-neutral pricing in our interest rate models, we build the additional assumptions with physical probabilities. Intuitively, risk-neutral probabilities adjust physical probabilities for risk aversion. That is, probabilities of undesirable states (such as random prepayment) are weighted up, while those of desirable states (such as non-prepayment) are weighted down. This means the price would be lower than the simulation currently predicts, and potentially even lower than the tree price if the market considers states where prepayment occurs as particularly undesirable. Pricing the mortgage must consider this.
2. **Structure of the Mortgage Market and Demand-Supply Dynamics:** If UBS is the only or one of the few providers of mortgages, it would enjoy a greater degree of pricing power and hence would not need to competitively price the mortgage, relative to if they operate in a more competitive market. Elasticity of demand for mortgages is another factor to consider, as a fairly elastic demand reduces pricing power.

3. **Information Asymmetry:** If we believe we must reprice the mortgage (lower) under these additional assumptions to ensure we have enough customers for the product, we implicitly assume the market is aware of the p and q probabilities defined above. In this case, they surmise that the value of the mortgage is less than fair (from the perspective of the borrower). We must consider the level of information asymmetry to evaluate the profit we can extract from the mortgage.

4. **Plausibility of Assumptions:** A critical assumption we make is that of no credit risk, which is rather unrealistic. Relaxing this assumption would mean the par rate would need to increase significantly to ensure fair compensation for the exposure to credit risk.

Considering the complexity of these factors, we recognise that simply reducing the rates after seeing a larger estimate of value would be rather naïve. Given the robustness of the tree pricing technique, and its validation via Monte Carlo leads us to recommend maintaining the mortgage rate at 1.527%.

6 Part (f)

Finally, we price a pass-through security, an interest rate only security and a principal only security backed by these mortgages. The maturity of these MBSs are the same, and are offered at a rate 50bps lower than the mortgage rate recommended by us in part (e) (to recover the cost of securitization). We will value them on the tree and then by Monte Carlo, under optimal prepayment and under assumptions from part (e).

6.1 Mortgage-Backed Securities

An investor of a pass-through security receives:

1. The complete principal component of the mortgage
2. The interest component of the mortgage less the servicing commission. In our context, the interest received would be 1.027%

A principal only (PO) and interest only (IO) security can be thought of as components of the pass-through, whereby a PO investor is entitled to the cash flow component 1 and the IO investor is entitled to the cash flow component 2.

As a benchmark, we produce the repayment schedule and value trees for all three MBSs.[12]

6.2 Simulation Methodology for Mortgage-Backed Securities

In this section, we will motivate the approach to finding the Monte Carlo estimates for the securities, under the prepayment assumptions of both part (b) and part (e). Since the cash flows of all 3 securities directly depend on the mortgage, we use the cash flow paths for the mortgage to generate a matrix of zeros and ones, which has ones where the mortgage was prepaid[13]. We use this to generate sample paths for the pass-through, PO and IO relying on the MBS repayment schedule. Under the assumption of optimal prepayment, the code is

```
PO_path = np.zeros_like(DCF_path)
IO_path = np.zeros_like(DCF_path)
for m in range(point_of_ex.shape[0]):
    for n in range(1,point_of_ex.shape[1]):
        PO_path[m,n] = exp(-sum(rate_path[m,0:n])*delta)
        ↪  *repay_MBS[1,n]
```

[12]Repayment schedule is Table 12 of Appendix II. Value trees are suppressed here, but found in code cell 40 of the source code document.

[13]Code for this matrix is in code cell 41 of the source code document

```
        IO_path[m,n] = exp(-sum(rate_path[m,0:n])*delta)
        ↪  *repay_MBS[0,n]
        if point_of_ex[m,n] == 1: # check for prepayment
            PO_path[m,n] += exp(-sum(rate_path[m,0:n])*delta)
            ↪  *repay_MBS[2,n]
            break

MBS_path = PO_path+IO_path # This is the path of a pass through
```

where `point_of_ex` is the indicator matrix and `MBS_path` is the cash flow path for the pass-through. We use the exact same methodology for the cash flow paths under the additional prepayment assumptions.[14]

6.3 Simulation Results

Below are the simulation results under both assumptions, providing price estimates along with their confidence intervals. Table 8 is under optimal prepayment while Table 9 is under the non-optimal prepayment assumptions

Table 8: Simulation Results under Optimal Prepayment (£)

MBS	Simulations	Tree Price	Price	S.E.	Lower (95%)	Upper (95%)
Pass-Through	100000	99030.1	99029.74	5.59	99018.78	99040.69
PO	100000	97038.87	97036.27	10.97	97014.77	97057.77
IO	100000	1991.23	1993.47	5.78	1982.14	2004.79

Table 9: Simulation Results under non-optimal Prepayment (£)

MBS	Simulations	Tree Price	Price	S.E.	Lower (95%)	Upper (95%)
Pass-Through	100000	99030.1	99063.11	5.2	99052.93	99073.3
PO	100000	97038.87	96827.87	10.25	96807.78	96847.95
IO	100000	1991.23	2235.24	5.53	2224.41	2246.08

The results we see here are consistent with parts (d) and (e). Under optimal prepayment, the simulated prices are very close to the tree prices of the three MBSs while under the additional prepayment assumptions, these prices diverge. It is interesting to note a diversification effect implicit in the pass-through. Considering the standard errors as a proportion of price, we see that the standard errors of the pass-through are much lower than those of the PO and the IO. This is because the PO has a positive duration while the IO has negative duration, which implies a negative correlation between the two components.

[14] Code cells 45-46 of the source code document

7 APPENDIX I: Results used for the Interest Rate Models

7.1 Deriving Equation 3

From the model specified in Subsection 1.2, we know that:

$$
\begin{aligned}
r_{0,0} &= r_{0,0} \text{ (observed from the yield curve)} \\
\\
r_{1,0} &= r_{0,0} + \theta_0^{HL}\Delta + \sigma^{HL}\sqrt{\Delta} \\
r_{1,1} &= r_{0,0} + \theta_0^{HL}\Delta - \sigma^{HL}\sqrt{\Delta} \\
&= r_{1,0} - 2\sigma^{HL}\sqrt{\Delta} \\
\\
r_{2,0} &= r_{1,0} + \theta_1^{HL}\Delta + \sigma^{HL}\sqrt{\Delta} \\
r_{2,1} &= r_{1,0} + \theta_1^{HL}\Delta - \sigma^{HL}\sqrt{\Delta} \\
&= r_{2,0} - 2\sigma^{HL}\sqrt{\Delta} \\
r_{2,2} &= r_{1,1} + \theta_1^{HL}\Delta - \sigma^{HL}\sqrt{\Delta} \\
&= (r_{1,0} - 2\sigma^{HL}\sqrt{\Delta}) + \theta_1^{HL}\Delta - \sigma^{HL}\sqrt{\Delta} \\
&= (r_{1,0} + \theta_1^{HL}\Delta - \sigma^{HL}\sqrt{\Delta}) - 2\sigma^{HL}\sqrt{\Delta} \\
&= r_{2,1} - 2\sigma^{HL}\sqrt{\Delta} \\
&= (r_{2,0} - 2\sigma^{HL}\sqrt{\Delta}) - 2\sigma^{HL}\sqrt{\Delta} \\
&= r_{2,0} - 4\sigma^{HL}\sqrt{\Delta} \\
&\vdots \\
r_{i,j} &= r_{i,0} - 2j\sigma^{HL}\sqrt{\Delta}
\end{aligned}
\tag{15}
$$

By solving for the rates in each state for all i periods, we can see a pattern emerge (see equations in bold) whereby the rate $r_{i,j}$ is a function of the rate in the 'upmost' state $r_{i,0}$, and the state j.

Further, focusing on node $j = 0 \; \forall \; i$:

$$r_{0,0} = r_{0,0} \text{ (observed from the yield curve)}$$

$$r_{1,0} = r_{0,0} + \theta_0^{HL}\Delta + \sigma^{HL}\sqrt{\Delta}$$

$$\begin{aligned} r_{2,0} &= r_{1,0} + \theta_1^{HL}\Delta + \sigma^{HL}\sqrt{\Delta} \\ &= (r_{0,0} + \theta_0^{HL}\Delta + \sigma^{HL}\sqrt{\Delta}) + \theta_1^{HL}\Delta + \sigma^{HL}\sqrt{\Delta} \\ &= r_{0,0} + (\theta_0^{HL} + \theta_1^{HL})\Delta + 2\sigma^{HL}\sqrt{\Delta} \end{aligned}$$

$$\begin{aligned} r_{3,0} &= r_{2,0} + \theta_2^{HL}\Delta + \sigma^{HL}\sqrt{\Delta} \\ &= (r_{0,0} + (\theta_0^{HL} + \theta_1^{HL})\Delta + 2\sigma^{HL}\sqrt{\Delta}) + \theta_2^{HL}\Delta + \sigma^{HL}\sqrt{\Delta} \\ &= r_{0,0} + (\theta_0^{HL} + \theta_1^{HL} + \theta_2^{HL})\Delta + 3\sigma^{HL}\sqrt{\Delta} \end{aligned}$$

$$\vdots$$

$$r_{i,0} = r_{0,0} + \Delta\sum_{l=0}^{i-1}\theta_l^{HL} + i\sigma^{HL}\sqrt{\Delta} \qquad \text{for } i \geq 1$$

Combining this with Equation 15, we get

$$\begin{aligned} r_{i,j} &= (r_{0,0} + \Delta\sum_{l=0}^{i-1}\theta_l^{HL} + i\sigma^{HL}\sqrt{\Delta}) - 2j\sigma^{HL}\sqrt{\Delta} \\ &= r_{0,0} + \Delta\sum_{l=0}^{i-1}\theta_l^{HL} + (i-2j)\sigma^{HL}\sqrt{\Delta} \qquad \text{for } i \geq 1 \end{aligned}$$

Which is Equation 3. In Python, we can declare this function as

```
def r_HL(m,n):
    return yields[0]+delta*sum(theta_HL[0:n])+(n-2*m)
    ↪  *sigma_HL*sqrt(delta)
```

7.2 Convergence in Distribution

Let $\{X_i\}_{i=1}^n$ be a sequence of random variables with CDFs $\{F_i\}_{i=1}^n$. We say that the sequence converges in distribution to X, i.e, $X_n \xrightarrow{d} X$ if

$$\lim_{x\to\infty} F_n(x) = F(x)$$

where F is the CDF of the random variable X

7.3 Lindeberg-Lévy Central Limit Theorem

Let $\{X_i\}_{i=1}^n$ be a sequence of random variables with mean μ and variance $\sigma^2 < \infty$. Then, as n approaches infinity, the random variable $\sqrt{n}(\bar{X}_n - \mu)$ converges in distribution to Normal. That is,

$$\sqrt{n}(\bar{X}_n - \mu) \xrightarrow{d} \mathcal{N}(0, \sigma^2)$$

or,

$$\bar{X}_n \xrightarrow{d} \mathcal{N}(\mu, \sigma^2/n)$$

8 APPENDIX II: Tables and Trees

Table 10: Repayment Schedule under the Ho-Lee Model (£)

Period	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Interest paid	0	1569	1502	1433	1364	1293	1221
Principal Paid	0	4295	4363	4431	4501	4571	4643
Outstanding Principal	100000	95705	91342	86911	82410	77839	73196

Period	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Interest paid	1149	1075	999	923	845	767	687
Principal Paid	4716	4790	4865	4941	5019	5098	5178
Outstanding Principal	68480	63691	58826	53884	48866	43768	38590

Period	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Interest paid	606	523	439	354	268	180	91
Principal Paid	5259	5341	5425	5510	5597	5685	5774
Outstanding Principal	33332	27990	22565	17055	11458	5774	0

Table 11: Repayment Schedule under the Black-Derman-Toy Model (£)

Period	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Interest paid	0	763	728	692	656	620	583
Principal Paid	0	4647	4683	4718	4754	4791	4827
Outstanding Principal	100000	95353	90670	85952	81198	76407	71580
Period	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Interest paid	546	509	472	434	396	358	319
Principal Paid	4864	4901	4939	4976	5014	5052	5091
Outstanding Principal	66716	61815	56876	51900	46886	41833	36742
Period	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Interest paid	280	241	202	162	122	82	41
Principal Paid	5130	5169	5209	5248	5288	5329	5369
Outstanding Principal	31612	26443	21235	15986	10698	5369	0

Table 12: Repayment Schedule for the MBSs (£)

Period	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Interest paid	0	513	489	465	441	417	392
Principal Paid	0	4647	4683	4718	4754	4791	4827
Outstanding Principal	100000	95353	90670	85952	81198	76407	71580
Period	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Interest paid	367	342	317	292	266	241	215
Principal Paid	4864	4901	4939	4976	5014	5052	5091
Outstanding Principal	66716	61815	56876	51900	46886	41833	36742
Period	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Interest paid	189	162	136	109	82	55	28
Principal Paid	5130	5169	5209	5248	5288	5329	5369
Outstanding Principal	31612	26443	21235	15986	10698	5369	0

Table 13: Mortgage Value Tree under the Ho-Lee Model (£)

Maturity Period	0.0 0	0.5 1	1.0 2	1.5 3	2.0 4	2.5 5	3.0 6	3.5 7	4.0 8	4.5 9	5.0 10
0	100000	93500	86165	78814	71807	65328	59423	54058	49159	44646	40441
1		95705	91249	84996	78046	71082	64481	58388	52811	47698	42969
2			91342	86911	82363	76240	69576	62989	56773	51019	45709
3				86911	82410	77839	73159	67159	60781	54539	48655
4					82410	77839	73196	68480	63655	57772	51650
5						77839	73196	68480	63691	58826	53852
6							73196	68480	63691	58826	53884
7								68480	63691	58826	53884
8									63691	58826	53884
9										58826	53884
10											53884
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											

Maturity Period	5.0 10	5.5 11	6.0 12	6.5 13	7.0 14	7.5 15	8.0 16	8.5 17	9.0 18	9.5 19	10.0 20
0	40441	36473	32677	28987	25342	21674	17910	13969	9753	5145	0
1	42969	38546	34351	30314	26364	22430	18437	14302	9930	5208	0
2	45709	40776	36140	31720	27439	23219	18982	14644	10110	5272	0
3	48655	43173	38051	33211	28570	24043	19548	14996	10294	5337	0
4	51650	45705	40086	34792	29761	24904	20134	15358	10482	5403	0
5	53852	48067	42164	36453	31013	25804	20741	15730	10674	5469	0
6	53884	48866	43741	38034	32298	26741	21370	16113	10870	5536	0
7	53884	48866	43768	38590	33314	27654	22018	16507	11070	5605	0
8	53884	48866	43768	38590	33332	27990	22559	16901	11274	5674	0
9	53884	48866	43768	38590	33332	27990	22565	17055	11458	5743	0
10	53884	48866	43768	38590	33332	27990	22565	17055	11458	5774	0
11		48866	43768	38590	33332	27990	22565	17055	11458	5774	0
12			43768	38590	33332	27990	22565	17055	11458	5774	0
13				38590	33332	27990	22565	17055	11458	5774	0
14					33332	27990	22565	17055	11458	5774	0
15						27990	22565	17055	11458	5774	0
16							22565	17055	11458	5774	0
17								17055	11458	5774	0
18									11458	5774	0
19										5774	0
20											0

Table 14: Mortgage Value Tree under the Black-Derman-Toy Model (£)

Maturity Period	0 0	0.5 1	1.0 2	1.5 3	2.0 4	2.5 5	3.0 6	3.5 7	4.0 8	4.5 9	5.0 10
0	100000	94759	89290	83784	78236	72683	67173	61739	56399	51166	46050
1		95353	90558	85433	80115	74691	69243	63825	58465	53182	47986
2			90670	85952	81137	76019	70736	65390	60045	54738	49489
3				85952	81198	76407	71547	66428	61184	55902	50632
4					81198	76407	71580	66716	61802	56675	51455
5						76407	71580	66716	61815	56876	51900
6							71580	66716	61815	56876	51900
7								66716	61815	56876	51900
8									61815	56876	51900
9										56876	51900
10											51900
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											

Maturity Period	5.0 10	5.5 11	6.0 12	6.5 13	7.0 14	7.5 15	8.0 16	8.5 17	9.0 18	9.5 19	10.0 20
0	46050	41058	36198	31471	26873	22392	18004	13664	9302	4807	0
1	47986	42887	37890	32997	28205	23505	18876	14285	9674	4958	0
2	49489	44310	39209	34187	29244	24372	19555	14765	9960	5072	0
3	50632	45400	40222	35103	30044	25039	20075	15133	10178	5159	0
4	51455	46215	40990	35800	30653	25546	20471	15411	10341	5223	0
5	51900	46754	41543	36320	31112	25929	20769	15621	10464	5272	0
6	51900	46886	41833	36664	31445	26215	20993	15778	10556	5307	0
7	51900	46886	41833	36742	31612	26405	21156	15895	10625	5334	0
8	51900	46886	41833	36742	31612	26443	21235	15974	10676	5354	0
9	51900	46886	41833	36742	31612	26443	21235	15986	10698	5369	0
10	51900	46886	41833	36742	31612	26443	21235	15986	10698	5369	0
11		46886	41833	36742	31612	26443	21235	15986	10698	5369	0
12			41833	36742	31612	26443	21235	15986	10698	5369	0
13				36742	31612	26443	21235	15986	10698	5369	0
14					31612	26443	21235	15986	10698	5369	0
15						26443	21235	15986	10698	5369	0
16							21235	15986	10698	5369	0
17								15986	10698	5369	0
18									10698	5369	0
19										5369	0
20											0