MFE 237J – Fall 2012
Professor Torous

Homework 2

The typical steps in pricing bonds with term structure consistent models include:

1) Selecting a suitable term structure model;

2) Calibrating the model to the current term and volatility structure of interest rates;

3) Applying Monte Carlo simulation and variance reduction methods to generate a large number of interest rate paths for the model;

4) Enumerating the cash flow structure of the subject bonds; and

5) Computing the bond prices. Our first exercise requires you to implement each of these steps.

For this exercise use the data provided to you in the spreadsheet 'HW2_Data.xlsx'.

a. (5 points) Calibrate a Hull and White term structure consistent model. Please consult the document "Hull-White-Theta-Calibration" to understand how to fit the Theta function. You will be using this model to price a three hundred and sixty month bond
b. (5 points) Use Monte Carlo simulation and suitable variance reduction methods to generate one thousand interest rate paths.
c. (5 points) Compute the cash flows for the following bonds all with a face amount of $100,000:
   - A three hundred and sixty month zero coupon bond.
   - A callable thirty year annuity, paying monthly, with a coupon of 2.12%. The strike is the face amount of the annuity at the end of each month.
   - An IO strip and a PO strip from the annuity above.
d. (5 points) Use your sample of interest rate draws to price the four bonds in part c.
e. (5 points) Discuss the strengths and weaknesses of your valuation methodology.
f. (5 points) Solve for the par coupon of the annuity.

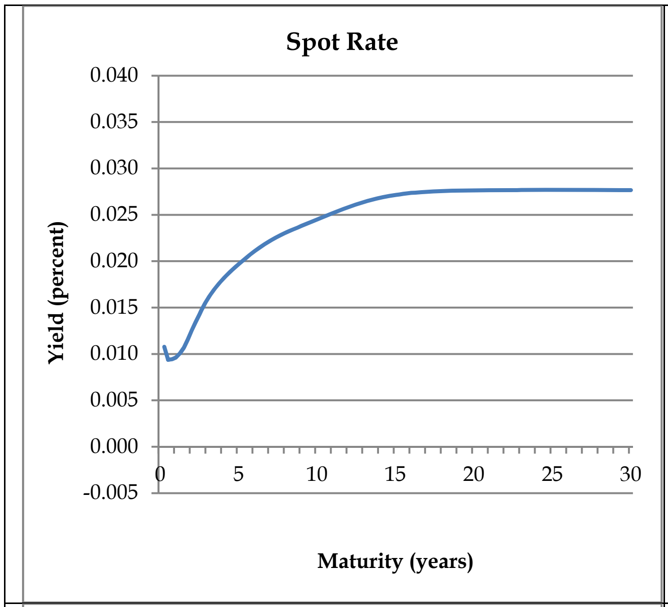
Mgt 237m – Fall 2012

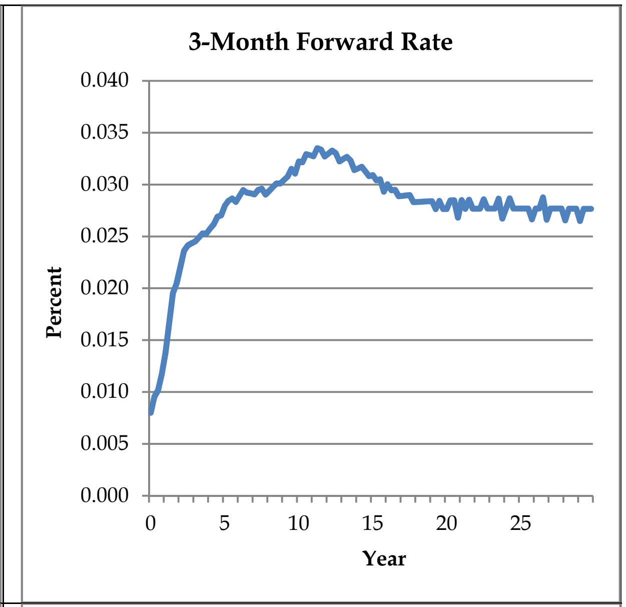
Homework 2 Solution

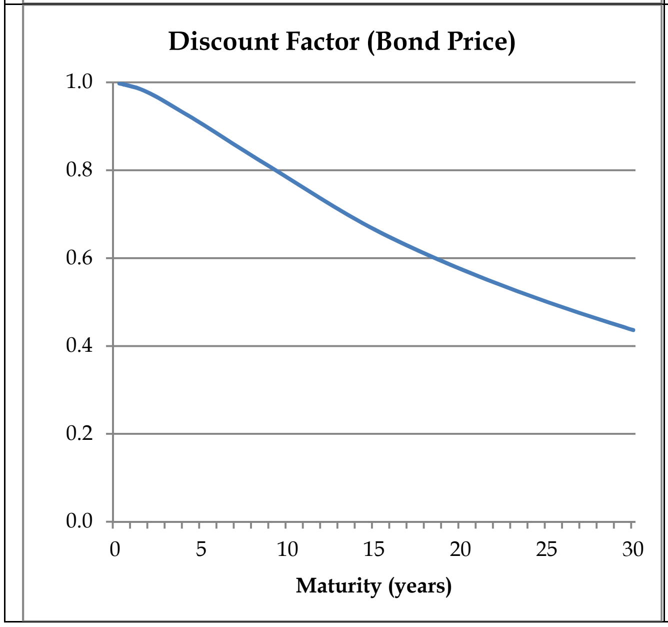
Part a)

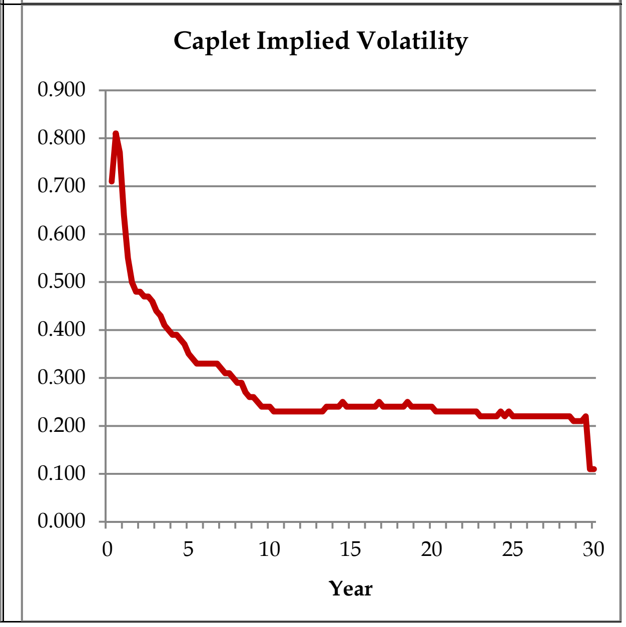
Use the swap rates from the Excel datasheet to construct a series of pure discount bond prices. From the discount factors, calculate the spot rate and forward rate for each maturity.

The following graphs show the spot rate curve, the forward rate curve and the discount rate curve obtained from swap data. Also shown are the caplet volatilities.









For the calibration, we use the Black volatilities from the series of caplets, the corresponding maturities in years, and the series of discount factors calculated in part a). The cap rate was given as 2.75%.

Caplets act like options with the forward rate underlying the option and a strike price equal to the cap rate. Since the volatilities reported in the spreadsheet are the implied black volatilities, we can use Black's formula to infer the cash price of the option.

Solve for the volatility parameters of the Hull-White model. These parameters are estimated by minimizing the sum of squared residual between the cash price and the price obtained from the Hull White Model. The Matlab function "lsqnonlin" performs nonlinear least squares on the specified objective function. It takes as an argument a function that calculates the residuals. "lsqnonlin" calculates the squared sum internally. The Matlab macro "`CalibrateHW.m`" calibrates the Hull-White model to a given series of caplets as just described.

The Matlab function "`HW_cap_resid.m`" calculates the residual between the price implied by Hull-White and the quoted cash prices for a series of caplets which serves as an input to the calibration function. We calculate the caplet price implied by the Hull White model. A caplet is equivalent to a put option with maturity t on a pure discount bond with maturity s with the face value of the bond L(1+cap_rate*dT) and strike price of L, where L is the principal underlying each cap. The formula for a put option under the Hull White model given in the lecture notes calculates the price per dollar of the underlying bond. We adjust so that the corresponding strike price is L/L(1+cap_rate*dT) = 1/(1+cap_rate*dT). The final price is readjusted prior to calculating the residual.

Finally, my calibrated Hull-White parameters are: $\alpha = 0.1005$, $\sigma = 0.0001$

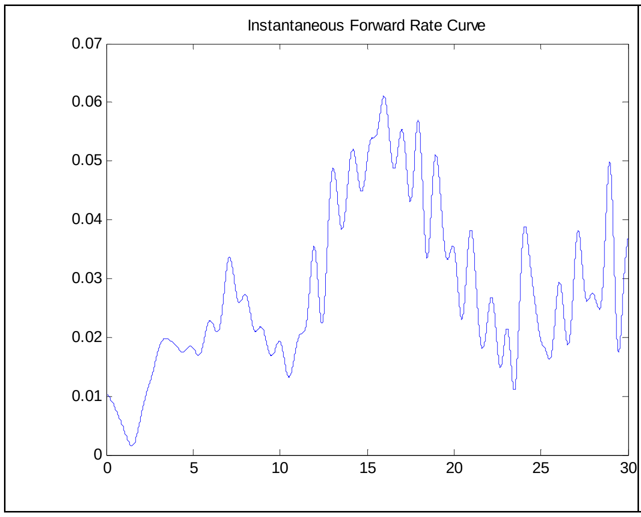
Part b)

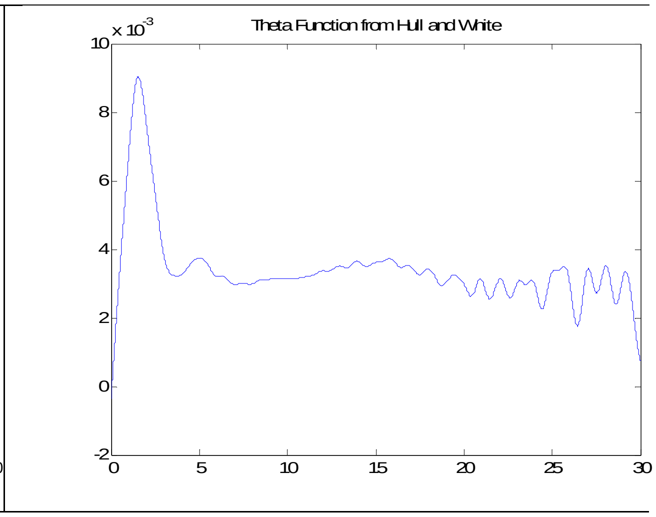
The next step is to calculate theta at each step of the simulation. This is done prior to running the simulation as the values of theta are the same across all simulation paths. The theta function allows us to match the current term structure. Using the updated theta generation function we obtain an estimated instantaneous forward curve.

A kernel smoothing technique (local regression) is used to estimate the derivatives required in the theta function calculation.

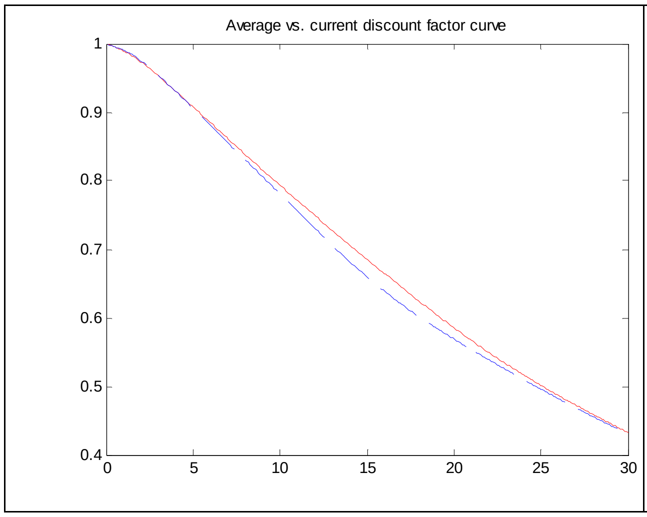
The theta function requires the instantaneous forward rate, F(0,t) = -d/dt ln PDB(t), and the derivative of the instantaneous forward rate, dF(0,t)/dt = -d2/dt2 ln PDB(t). Therefore we need to find the first and second derivatives of ln PDB with respect to time. Since we do not have a functional form for PDB in terms of time we must use a numerical method to derive these derivatives. We use a kernel smoothing technique - local regression. The derivatives - both first and second - of the ln PDB function at each point in time are given by a weighted regression of all the PDB observations. The regression coefficients are the derivatives of interest. We will use an exponential kernel as the weight. An important variable is the choice of the bandwidth, h. This value determines the smoothness of the function. A common choice is $n^{-1/5}$ where n is the number of observations. Since this is a local regression, a separate regression must be run for each time step in the simulation (dt).

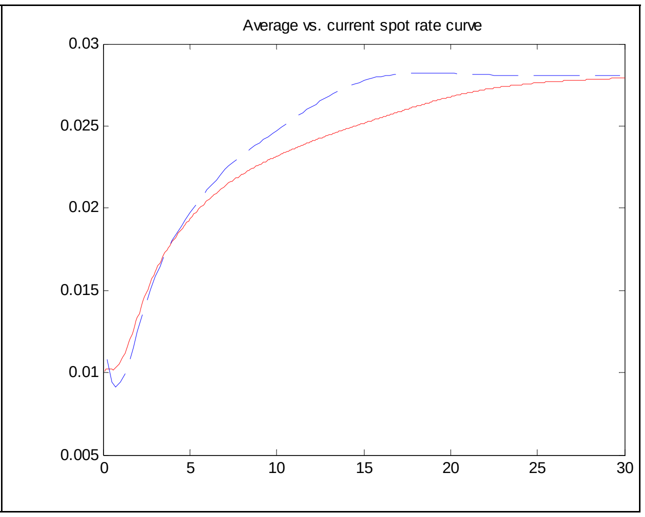
First, we calculate first and second differences (numerical first and second derivatives) as starting values. Next, we set up a local regression at each time step in the simulation (dt) on the observed values of PDB. The dependent variable is ln(PDB) and the explanatory variables are transformations of maturity times. The weights at each time step are calculated using an exponential kernel: K = exp{ -0.5*([maturityTime - simulationTime]/bandwidth) ^2 }. Then perform the regression at each simulation time and calculate the theta function and the instantaneous forward rates.





In the following graphs the solid line is the average over all simulation paths and the dotted line is the current curve. The average over all simulation paths is taken in terms of discount factors. The spot rate curve is derived from the average discount factor curve.





The variance reduction method most suited to this pricing example is the use of antithetic variables. To run the simulation with antithetic variables we first take one set of draws from a standard normal distribution and then calculate two sets of interest rate paths, one using the draws as given and one using the negative of the draws. In the pricing section of the simulation we will use these two sets of interest rate paths to calculate two prices for each simulation run and take the average (of those two prices) as the price corresponding to that simulation run.

Part c)

The zero-coupon bond pays $100,000 at 30 years with no other payments due. With a coupon of 2.12%, the monthly payment for a 30 year annuity is $375.64. Next, we calculate the amount of this payment corresponding to interest by multiplying 2.12% times the remaining balance. The principal component is the portion that is left over. We subtract the principal component from the balance, and repeat this process until the balance is fully paid out. If the calculation is done properly this will occur in exactly 30 years. Because of this construction, you should confirm that the annuity price is equal to the sum of the IO and PO prices.

Part d)

To calculate the price of each bond you use the simulated interest rate paths to calculate the present value of all cash flows under each simulated scenario. The price is the average present value over all simulation paths. By construction the average of the simulated monthly interest rate paths should be close to the observed term structure. This would imply that since the cash flows are fixed we could price these bonds using the current term structure. Using Monte Carlo simulation we price the zero bond at $43,494. The price of the annuity is $94,623. The price of the IO is $27,759, and the price of the PO is $66,864.

Part e)

The benefits of the Hull White model are that it is constructed to match the current term structure, and unlike the Ho and Lee model it allows for mean reversion. A problem may arise if the calibrated parameter turns out to be negative. This will lead to an inverted yield curve and instability. Another source of instability is negative interest rates. We observed that at least some of your paths had negative interest rates. While the Hull and White model allows for negative interest rates, this is not something that we are likely to observe in the market. As a result bonds would be overpriced. To account for this one could estimate a variation of the Cox, Ingersoll, and Ross model that still allows for fitting of the current term structure. This is discussed in Hull and White (1990). While it may seem reasonable to simply truncate the negative interest rates to be 0, note that this changes the assumptions of the model and therefore you would no longer be able match the current term structure.

Part f)

The par coupon (P) is the coupon on the annuity that will give a market price equal to the face value. While the par coupon can be found using a grid search, realize that since an annuity has a fixed payment in each month it is possible to use the average of the simulated discount factors to solve for the par annuity payment (A).

$$A = \$100,000 \cdot \frac{P}{12} \cdot \frac{\left(1+\frac{P}{12}\right)^{360}}{\left(1+\frac{P}{12}\right)^{360} - 1}$$

$$\$100,000 = \frac{A}{N} \sum_{n=1}^{N} \sum_{t=1}^{360} D_n(t)$$

Converting the par annuity payment to a coupon we obtain a par rate of 2.54%.

Appendix

CalibrateHW.m

```
% *** CalibrateHW.m ***

% This macro calibrates the Hull-White model to a given series of caplets
% The required data is Black volatilities from a series of caplets,
% the corresponding maturity in years, and a series of discount factors (with
% maturities).

% Prior to running this macro the following variables must be in matlab:
% cap_rate:  a scalar with the rate on the cap
% maturity:  a vector of maturity in years, corresponding to each caplet
% caplet_vol:  a vector with the quoted volatility for each caplet
% PDB: series of pure discount bond prices
% PDB_t: maturities corresponding to PDB

% Hull-White:  dr = [theta(t) - alpha*r]dt + sigma*dz
% T: maturity of the derivative, s: maturity of the underlying bond
% PDB: series of pure discount bond prices
% PDB_t: maturities corresponding to PDB

% From the discount factors calculate the spot rate and the forward rate
% Adjust T and s to match timing of forward rate
T = maturity(1:end-1);
s = maturity(2:end);
% Select the elements of the discount function corresponding to T and s
PT = zeros(size(T));    % P(0, T)
Ps = zeros(size(s));    % P(0, s)
for i=1:length(T)
    PT(i) = PDB(find(PDB_t==T(i)));
    Ps(i) = PDB(find(PDB_t==s(i)));
end
dT = T(2)-T(1);
R = (-1./T).*log(PT); % Spot Rate, adjust to match maturities on forward rate
F = (-1./dT).*log(Ps./PT); % Forward rate for F(0,T,T+1)

% Caplets act like options with the forward rate underlying the option
% and a strike price equal to the cap rate. Since the volatilities reported
% by Yieldbook are the implied black vols, we can use Black's formula to
% infer the cash price of the option. See Clewlow & Strickland p. 212
K = cap_rate;
b_vol = caplet_vol(1:end-1);
d1 = (log(F/K) + (b_vol.^2).*T*0.5)./(b_vol.*sqrt(T));
d2 = d1 - b_vol.*sqrt(T);
caplet = PT.*(F.*normcdf(d1,0,1) - K*normcdf(d2,0,1)).*dT;

% The general syntax for a Matlab optimization routine is as follows:
% The first argument is the objective function.  In this case the residual
% to be used in the least squares optimization.  The "at" sign is part of
% the syntax and indicates to the optimization routine that a function is
% being referenced.  This argument is normally a user defined function.
% The second argument is a vector with the starting values for the
% parameters to be solved.  It also corresponds to the first argument of
% the objective function.  The first argument of an objective function
% must be the parameters to be solved.  The following three arguments
% of the optimization routine are optimization options.  An empty vector []
% acts as a placeholder.  The rest of the arguments represent the arguments
% of the objective function.

options = optimset('Display','off');  % Set optimization options
p0 = [0.1 0.01]; % Set initial values for alpha and sigma, resp.
[param res1 res2 flag output] = lsqnonlin(@HW_cap_resid, p0, [], [], options,
caplet, T, s, PDB_t, PDB, cap_rate);
param
flag
output
```

HW_cap_resid.m

```
% *** HW_cap_resid.m ***

% This function calculates the residual between the price implied
% by Hull-White and the quoted cash prices for a series of caplets

% Hull-White:  dr = [theta(t) - alpha*r]dt + sigma*dz
% T: maturity of the derivative, s: maturity of the underlying bond
% PDB: series of pure discount bond prices
% PDB_t: maturities corresponding to PDB
% K: strike price on the option

function resid = HW_cap_resid(param, cash_price, T, s, PDB_t, PDB, cap_rate)

alpha = param(1);
sigma = param(2);

% Calculate the spot rates
PT = zeros(size(T));    % P(0, T)
Ps = zeros(size(s));    % P(0, s)
for i=1:length(T)
    PT(i) = PDB(find(PDB_t==T(i)));
    Ps(i) = PDB(length(s));
end
R = (-1./T).*log(PT);

% Calculate the caplet price implied by the Hull White model
% A caplet covering the period from t to t+1 is equivalent to a put
% option with maturity t on a pure discount bond with maturity t+1
% with the face value of the bond L(1+cap_rate*dT) and strike price of L,
% where L is the principal underlying each cap.

% The formula for a put option under the Hull White model given in Clewlow
% and Strickland p. 217 calculates the price per dollar of the underlying
% bond.  We adjust so that the corresponding strike price is
% L/L(1+cap_rate*dT) = 1/(1+cap_rate*dT). The final price is readjusted
% prior to calculating the residual.

dT = T(2)-T(1);
K = 1/(1+cap_rate*dT);

% Calculate the components of the Black formula
sig_p2 = ((sigma^2)/(2*alpha^3))*(1 - exp(-2*alpha*T)).*((1 - exp(-
1*alpha*(s-T))).^2);
sig_p = sqrt(sig_p2);
d1 = log(Ps./(K.*PT))./sig_p + sig_p/2;
d2 = d1 - sig_p;
hw_price = K.*PT.*normcdf(-1*d2, 0, 1) - Ps.*normcdf(-1*d1, 0, 1);  % Put
price
hw_price = hw_price/K;

% Calculate the residual
resid = hw_price - cash_price;
```

Gen_Theta_Kernel.m

```
%*** Gen_Theta_Kernel.m ***

% This function calculates the theta function to be used in a Hull White
simulation
% with time steps dt for a bond with maturity T.  Based on the parameters and
% discount information provided by the user.  Recall that in the Hull White
% model theta is a time varying function so the output of this function is
% a vector.  A kernel smoothing technique (local regression) is used to
% estimate the derivatives required in the theta function calculation.

function [theta, F0] = gen_theta_kernel(alpha, sigma, T, dt, PDB_t, PDB)

disp('Generating Theta...');
sim_t = [dt:dt:T];      % Vector with simulation time steps

% The theta function requires the instantaneous forward rate,
% F(0,t) = -d/dt ln PDB(t), and the derivative of the inst. forward
% rate, dF(0,t)/dt = -d2/dt2 ln PDB(t), therefore we need to find the
% first and second derivatives of ln PDB with respect to time. Since
% we do not have a functional form for PDB in terms of time we must
% use a numerical method to derive these derivatives.



% We use a kernel smoothing technique - local regression.  The
% derivatives - both first and second - of the ln PDB function at each
% point in time are given by a weighted regression of all our PDB
observations.
% Refer to the section notes for further details.

% The regression coefficients are the derivatives of interest. We will
% use an exponential kernel as the weight.  An important variable is
% the choice of the bandwidth, h.  This value determines the smoothness
% of the function.  A common choice is n^(-1/5) where n is the number of
% observations.  Since this is a local regression, a separate regression
% must be run for each time step in the simulation (sim_t).

h = length(PDB_t)^(-1/5);

% Calculate the central differences to use as starting values.
% Note that these calculations return vectors so that starting
% values for all runs are calculated simultaneously.
delta = 0.01;
ln_PDB = log(spline(PDB_t, PDB, sim_t      ));
up = log(spline(PDB_t, PDB, sim_t+delta));
dn = log(spline(PDB_t, PDB, sim_t-delta));
dPdt = (up - dn)/(2*delta);
d2Pdt2 = (up - 2*ln_PDB + dn)/(delta^2);

% Set up a local regression on the observed values of PDB.
% The dependent variable is ln PDB and the explanatory variables are
% transformations of time (PDB_t).  Loop through each time step.

for i = 1:length(sim_t)
    % The local regression is evaluated at the current element of sim_t.
    this_t = sim_t(i);
    % Set the starting values - numerical derivatives evaluated at current t
    d0 = [ln_PDB(i), dPdt(i), d2Pdt2(i)];
    % Calculate the Kernel for each observation - this is a vector
    K = exp(-0.5*((PDB_t - this_t)/h).^2);
    % Calculate ln PDB for each observation - this is a vector
    Ys = log(PDB);

    % Perform the regression
    options = optimset('Display','off');  % Set optimization options
    [d_vec flag output] = lsqnonlin(@Kernel_Resid, d0, [], [], options, K,
Ys, PDB_t);

    % Calculate the theta function at the current t
    theta(i) = -d_vec(3) - alpha*d_vec(2) + ((sigma^2)/(2*alpha))*(1-exp(-
2*alpha*this_t));
    % Calculate the forward rate for reference
    F0(i) = -1*d_vec(2);
end

function u = Kernel_Resid(d0, K, Ys, PDB_t)

a = d0(1);
b = d0(2);
c = d0(3);

u = sqrt(K).*(Ys - a - b*PDB_t - 0.5*c*PDB_t.^2);
```

"MGMT237J_HW1.m"

```matlab
% *** MGMT 237J HW1 Solution - Bond Pricing *** %

% This function runs all the calculations for Homework 1  %
% Inputs:
% vol_file: name, as a string, of the file containing the %
%            caplet vols with the corresponding maturities %
% pdb_file: name, as a string, of the file containing the %
%           discount factors with corresponding maturities %
% ***** Note the files downloaded in Yield book may have  %
%       to be adjusted in Excel to put them in the        %
%       correct format for use in Matlab            ***** %
% dt: the size of the time steps used in the simulation   %
% N: the number of simulation runs                        %
% plot_flag: 1 to generate plots                          %

% To run type the following at Matlab control prompt
% [param, annuity, zero, zero_se, ann_price, ann_se, IO_price, IO_se,
PO_price, PO_se, par_rate] = HW1MatlabKey('2008VOL.txt', '2008PDB.txt',
0.004, 1000, 1)

function [param, annuity, zero, zero_se, ann_price, ann_se, ...
        IO_price, IO_se, PO_price, PO_se, par_rate] = HW1MatlabKey(vol_file,
pdb_file, dt, N, plot_flag)

tic

% Load data
vols_in = load(vol_file);
pdb_in = load(pdb_file);

maturity = vols_in(:,1);
caplet_vol = vols_in(:,2);
PDB_t = pdb_in(:,1);
PDB = pdb_in(:,2);

cap_rate = 0.0275;
short_rate = 0.010672;

% Set bond terms
term = 30;
freq = 12;
coupon = 0.0212;
face = 100000;

% Calibrate Hull-White
CalibrateHW
rate_t = T;



if plot_flag == 1
    % Plot the yieldbook data
    plot(PDB_t, PDB);
    title('Discount Factors from Swap Rates');
    figure;
    plot(rate_t, R);
    title('Spot Curve from Swap Rates');
    figure;
    plot(rate_t, F);
    title('Forward Curve from Swap Rates');
    figure;
    plot(maturity, caplet_vol);
    title('Caplet Volatilities from YB');
    figure;
end

[theta, F] = gen_theta_kernel(param(1), param(2), term, dt, PDB_t, PDB);


if plot_flag == 1
    % Plot the instantaneous forward rate and theta for reference
    plot([dt:dt:term], F);
    title('Instantaneous Forward Rate Curve');
    figure;
    plot([dt:dt:term], theta);
    title('Theta Function from Hull and White');
end

% Generate cash flows
annuity = ann_payment(coupon, term, freq, face)
balance = face;
for i=1:term*freq
    IO(i,1) = (coupon/12)*balance;
    PO(i,1) = annuity - IO(i,1);
    balance = balance -PO(i,1);
end

% Generate interest rate paths and calculate discount factors using
% the antithetic variance reduction technique

cont_pos = [ones(term*freq+1,N)];         % Matrix to store continuous
discount factors
cont_neg = [ones(term*freq+1,N)];
path_pos = [short_rate*ones(1,N)]; % Matrix to store full interest rate paths
path_neg = [short_rate*ones(1,N)];

j = 1;              % Monthly counter
for i=1:term/dt

%    disp(strcat('time step: ', num2str(i*dt)));

    dW = sqrt(dt)*randn(1,N);
    new_pos = path_pos(i,:) + (theta(i) - param(1) * path_pos(i,:)) * dt +
param(2) * dW;
    new_neg = path_neg(i,:) + (theta(i) - param(1) * path_neg(i,:)) * dt -
param(2) * dW;
    path_pos(i+1,:) = new_pos;
    path_neg(i+1,:) = new_neg;

    % If the current period matches the cash flows save the rates for
discount factor calculation
```

```
    check = mod((i+1) * dt, 1/freq);
    if check > 0 & check - dt <= 10^(-5)
        j = j+1;
        cont_pos(j,:) = sum(path_pos(2:i+1,:)*dt);
        cont_neg(j,:) = sum(path_neg(2:i+1,:)*dt);
    end

end

% Make final adjustments to discount factors
cont_pos = exp(-1*cont_pos(2:end,:));
cont_neg = exp(-1*cont_neg(2:end,:));

% Calculate the average PDB curve
pdb_avg = mean(((cont_pos + cont_neg)/2),2);
% Extract the corresponding discount factors
bond_t = [1:360]'/12;
pdb_org = spline(PDB_t, PDB, bond_t);

% Calculate the spot curve derived from the average PDB curve
r_avg = (-1./bond_t).*log(pdb_avg);

if plot_flag == 1
    % Plot the average curves vs. the current curves
    figure;
    plot(bond_t, pdb_avg, 'r', bond_t, pdb_org, '--');
    title('Average vs. current discount factor curve');
    figure;
    plot(bond_t, r_avg, 'r', rate_t, R, '--');
    title('Average vs. current spot rate curve');
end

% Price the zero
zero = face*(cont_pos(end,:) + cont_neg(end,:))/2;
zero_se = std(zero)/sqrt(length(zero));
zero = mean(zero);
zero

% Price the other bonds
[ann_price, ann_se] = ann_pricing(annuity, cont_pos, cont_neg);
[IO_price, IO_se] = cf_pricing(IO, cont_pos, cont_neg);
[PO_price, PO_se] = cf_pricing(PO, cont_pos, cont_neg);
ann_price
IO_price
IO_se
PO_price
PO_se
% Now price the annuity using the current curve for further checking
ann_curr = annuity*sum(pdb_org)

% Solve for the par rate of the annuity
options = optimset('Display','off');  % Set optimization options
[par_rate, diff] = fsolve(@solve_par, coupon, options, term, freq, face,
cont_pos, cont_neg)
```

```
par_rate
toc

function annuity = ann_payment(coupon, term, freq, face)
    % Calculate the annuity payment given terms of the annuity
    per_coup = coupon/12;
    annuity =
face*(per_coup*(1+per_coup)^(term*freq))/((1+per_coup)^(term*freq)-1);

function [price, se] = ann_pricing(annuity, disc_pos, disc_neg)
    % Prices an annuity
    price_vec = annuity*(sum(disc_pos) + sum(disc_neg))/2;
    price = mean(price_vec);
    se = std(price_vec)/sqrt(length(price_vec));

function [price, se] = cf_pricing(cash_flow, disc_pos, disc_neg)
    % Prices a bond with a fixed cash flow
    [t,N] = size(disc_pos);
    cash_flow = repmat(cash_flow, 1, N);
    price_vec = (sum(cash_flow.*disc_pos) + sum(cash_flow.*disc_neg))/2;
    price = mean(price_vec);
    se = std(price_vec)/sqrt(length(price_vec));

function diff = solve_par(coupon, term, freq, face, disc_pos, disc_neg)
    % Objective function used to solve for the par rate
    annuity = ann_payment(coupon, term, freq, face);
    ann_price = ann_pricing(annuity, disc_pos, disc_neg);
    diff = face - ann_price;
```