A.1 main.m

%

% Author : Otto Sjoholm and Dan Franzen

% Created : 2013 -10 -14

% Edited : 2013 -12 -03

%

% Purpose : Main script for calculation of CVA under :

% 1. Internal Model Method (IMM)

% 2. Current Exposure Method ( CEM )

%

% Parameters : None

%

% Return : None

%

% dbstop if error

close all

clear all

clc

% Start time measurement

tic

%% Set Constants and read data files

fprintf ('\ nDefining Constants and reading data files... \n\n')

global year numYears horizon scenarioIndex irData swapFile cemFile

dateStr = '2013 -11 -05 ';

dateFormat = 'yyyy -mm -dd ';

dataPath = '.\ data \';

irData = strcat ( dataPath ,'euribor.xls ');

swapFile = strcat ( dataPath ,' swap\_portfolio.xls ');

cemFile = strcat ( dataPath ,'data\_cem.xls ');

[ rating , ratingMap ] = func\_read\_rating\_data ();

numSwaps = numel ( rating.CounterpartyID );

numCounterparties = max( rating.CounterpartyID );

Settle = datenum ( dateStr , dateFormat );

RateSpec = func\_create\_rate\_spec ( Settle , false );

swapData = func\_read\_swap\_data ( RateSpec );

stdCVAimm = struct ( ...

'Horizon ' ,[], ...

'Weights ' ,[], ...

'Maturities ' ,[], ...

'EAD ' ,[], ...

' SinglehPrincipal ' ,[], ...

' SinglehMaturities ' ,[], ...

'IndexhWeights ' ,[], ...

' IndexhPrincipal ' ,[], ...

' IndexhMaturites ' ,[]);

% set time horizon

year = 365;

numYears = 5;

horizon = 1;

scenarioIndex = 200;

% set counterparty data and hedges

w = zeros ( numCounterparties ,1);

for i=1: numCounterparties

w(i) = ratingMap ( rating.Rating {i ,1});

end

stdCVAimm.Weights = w;

stdCVAimm.SinglehPrincipal = zeros ( numCounterparties ,1);

stdCVAimm.SinglehMaturities = zeros ( numCounterparties ,1);

% set data for index hedges ( here : no hedges are used )

numInd = 1;

stdCVAimm.Horizon = horizon ;

stdCVAimm.IndexhWeights = zeros (numInd ,1);

stdCVAimm.IndexhPrincipal = zeros (numInd ,1);

stdCVAimm.IndexhMaturities = zeros ( numInd ,1);

stdCVAcem = stdCVAimm ;

% display time passed

toc

%% Compute CVA under IMM

fprintf ('\ nComputing CVA under IMM... \n\n')

numScenarios = 500;

alpha = 1.4;

[EPEcp , EffMaturities ] = func\_imm\_epe ( numScenarios , Settle , false );

stdCVAimm.Maturities = EffMaturities ;

stdCVAimm.EAD = alpha \* EPEcp ;

[ IMM\_CVAcp , IMM\_CVAport ] = func\_compute\_cva ( stdCVAimm );

% display time passed

toc

%% Compute CVA under CEM

fprintf ('\ nComputing CVA under CEM... \n\n')

stdCVAcem.Maturities = func\_cem\_eff\_maturity ( swapData , Settle );

stdCVAcem.EAD = func\_cem\_ead ( stdCVAcem.Maturities );

[ CEM\_CVAcp , CEM\_CVAport ] = func\_compute\_cva ( stdCVAcem );

%% Display Results

fprintf ('CVA for each counterparty under IMM ( SEK ):\n')

disp ( IMM\_CVAcp )

fprintf ('CVA for entire portfolio under IMM (SEK ):\ n')

disp ( IMM\_CVAport )

fprintf ('Effective maturities for each counterperty under IMM ( years ):\ n')

disp ( stdCVAimm.Maturities )

fprintf ('CVA for each counterparty under CEM ( SEK ):\n')

disp ( CEM\_CVAcp )

fprintf ('CVA for entire portfolio under CEM (SEK ):\ n')

disp ( CEM\_CVAport )

fprintf ('Effective maturities for each counterperty under CEM ( years ):\ n')

disp ( stdCVAcem.Maturities )

% display time passed

toc

fprintf ('\ nSimulation Successful !\n')

end

A.2 func read rating data.m

%

% Author : Otto Sjoholm and Dan Franzen

% Created : 2013 -10 -21

% Edited : 2013 -12 -03

%

% Function : func\_read\_rating \_data.m

%

% Purpose : reads the rating data and rating to weight mapping

% from 'swapFile '

%

% Parameters : None

%

% Return : rating - the rating of each counterparty on the form

% 'AAA ', 'AA ', e.t.c...

% ratingMap - the rating map ; mapping every possible rating

% to specific numeric weight

%

% Examples :

%

%

%

function [ rating , ratingMap ] = func\_read\_rating\_data ( )

global swapFile

mapData = dataset ( ...

'XLSFile ', swapFile , ...

'Sheet ', 'RatingMap ');

ratingData = dataset ( ...

'XLSFile ', swapFile , ...

'Sheet ', 'Rating ');

ratingMap = containers.Map ( ...

mapData.Rating , ...

mapData.Weight );

rating = struct ( ...

' CounterpartyID ' ,[], ...

'Rating ' ,[]);

rating.CounterpartyID = ratingData.CounterpartyID ;

rating.Rating = ratingData.Rating ;

end

A.3 func create rate spec.m

%

% Author : Otto Sjoholm and Dan Franzen

% Created : 2013 -11 -04

% Edited : 2013 -12 -03

%

% Function : func\_create\_rate\_spec.m

%

% Purpose : Creates structures containing information of an interest

% rate term structure.

%

% Parameters : Settle - settlement date of swaps

% display\_on - boolean which determines if the historical

% euribor rates should be displayed in a plot

% or not

%

% Return : RateSpec - structure containing properties of an

% interest term structure

% RateCurveObj - interest rate curve represented with data

% Tenor - vector containing the number of months

% where swap rates are specified

%

function [ RateSpec , RateCurveObj , Tenor ] = func\_create\_rate\_spec ( Settle , display\_on )

global swapFile

zeroCurveData = dataset ( ...

'XLSFile ',swapFile , ...

'Sheet ','Swap Curve ');

Tenor = 12\* zeroCurveData.Years + zeroCurveData.Months ;

ZeroRates = zeroCurveData.Rate ;

ZeroDates = datemnth ( Settle , Tenor );

Compounding = -1;

Basis = 0;

RateSpec = intenvset (...

'StartDates ', Settle , ...

'EndDates ', ZeroDates , ...

'Rates ', ZeroRates , ...

'Compounding ',Compounding , ...

'Basis ',Basis );

% Create an IRCurve object. We will use this for computing instantaneous

% forward rates during the calculation of the Hull - White short rate path.

RateCurveObj = IRDataCurve ( ...

'Zero ',Settle , ZeroDates , ZeroRates , ...

'Compounding ', Compounding , ...

'Basis ', Basis );

if display\_on

figure ;

plot ( ZeroDates , ZeroRates , 'o-');

xlabel ('Date ');

datetick ('keeplimits ');

ylabel ('Zero rate '); grid on;

title ('Yield Curve at Settle Date ');

end

end

A.4 func read swap data.m

%

% Author : Otto Sjoholm and Dan Franzen

% Created : 2013 -10 -21

% Edited : 2013 -12 -03

%

% Function : func\_read\_swap\_data.m

%

% Purpose : reads the swap data from the global variable 'swapFile '

% and uses the RateSpec to set the fixed rate so that the

% swap price is zero at time 0

%

% Parameters : RateSpec - structure containing properties of an

% interest term structure

%

% Return : swaps - structure containing all necessary data to

% price the swap instruments

%

function [ swaps ] = func\_read\_swap\_data ( RateSpec )

global swapFile

% Read swaps from spreadsheet

swapData = dataset ( ...

'XLSFile ',swapFile , ...

'Sheet ','Swap Portfolio Short ');

swaps = struct ( ...

'Counterparty ' ,[], ...

'NettingID ' ,[], ...

'Principal ' ,[], ...

'Maturity ' ,[], ...

'LegRate ' ,[], ...

'LegType ' ,[], ...

' LatestFloatingRate ' ,[], ...

' FloatingResetDates ' ,[]);

swaps.Counterparty = swapData.CounterpartyID ;

swaps.NettingID = swapData.NettingID ;

swaps.Principal = swapData.Principal ;

swaps.Maturity = datenum ( swapData.Maturity , 'yyyy -mm -dd ');

swaps.LegType = [ swapData.LegType ~ swapData.LegType ];

swaps.LegRate = [ swapData.LegRateReceiving swapData.LegRatePaying ];

swaps.LatestFloatingRate = swapData.LatestFloatingRate ;

swaps.Period = swapData.Period ;

swaps.LegReset = ones ( size ( swaps.LegType ));

%% Set the fixed rate so that the swap price initially is zero

numCounterparties = max( swaps.Counterparty );

for i=1: numCounterparties

LegRate = [ NaN swapData.LegRatePaying (i )];

[~, swapData.LegRateReceiving (i)] = ...

swapbyzero (...

RateSpec , ...

LegRate , ...

RateSpec.StartDates , ...

swaps.Maturity (i),...

swaps.LegReset (i ,:) , ...

RateSpec.Basis , ...

swaps.Principal (i), ...

swaps.LegType (i ,:));

end

swaps.LegRate = [ swapData.LegRateReceiving swapData.LegRatePaying ];

End

A.5 func imm epe.m

%

% Author : Otto Sjoholm and Dan Franzen

% Date : 2013 -11 -05

%

% Function : func\_imm\_epe.m

%

% Purpose : Computes the expected positive exposure (EPE ) and effective

% maturity under IMM.

%

% Parameters : numScenarios - the number of interest rate scenario

% simulations performed

% Settle - the settlement date of the contracts given

% in MATLAB date number format

% print\_on - boolean value indicating if function should

% print status messages when running

%

% Return : EPEcp - effective positive exposure per

% counterparty

% EffMaturity - effective maturity of each netting set

%

function [ EPEcp , EffMaturity ] = func\_imm\_epe ( numScenarios , Settle , print\_on )

global irData numYears scenarioIndex

%% Create RateSpec from the Interest Rate Curve

if print\_on

fprintf ('Creating Rate Curve Object... \n\n')

end

[ RateSpec , RateCurveObj , Tenor ] = func\_create\_rate\_spec ( Settle , false );

% ZeroDates = RateSpec.EndDates ;

%% Read Swap Portfolio

if print\_on

fprintf ('Reading Swap Portfolio... \n\n')

end

swaps = func\_read\_swap\_data ( RateSpec );

numSwaps = numel ( swaps.Counterparty );

numCounterparties = max( swaps.Counterparty );

%% Set Simulation Parameters

if print\_on

fprintf ('Setting Simulation Parameter... \n\n')

end

% Compute monthly simulation dates , then quarterly dates later.

simDates = datemnth ( Settle ,0:1:(12\* numYears )) ';

numDates = numel ( simDates );

%% Compute Floating Reset Dates

if print\_on

fprintf ('Computing Floating Rates... \n\n')

end

floatDates = cfdates ( Settle -360 , swaps.Maturity , swaps.Period );

swaps.FloatingResetDates = zeros ( numSwaps , numDates );

for i = numDates : -1:1

thisDate = simDates (i);

floatDates ( floatDates > thisDate ) = 0;

swaps.FloatingResetDates (:,i) = max ( floatDates ,[] ,2);

end

%% Read Euribor data

if print\_on

fprintf ('Reading EURIBOR Data... \n\n')

end

[num ,~ ,~] = xlsread ( irData );

% date\_string = txt (2: end ,1);

% date = datenum ( date\_string ,'yyyy -mm -dd ');

euribor\_3m = num (: ,3);

% plot (date , euribor\_3m );

% datetick ('x '), xlabel ('Date '), ylabel (' Annual Yield (%)

%% Fit model to data

if print\_on

fprintf ('Fitting Interest Rate Model Parameters To Data... \n\n')

end

yields = euribor\_3m ;

regressors = [ ones ( length ( yields ) - 1, 1) yields (1: end -1)];

[ coefficients , ~, residuals ] = regress ( diff ( yields ), regressors );

% The time increment is set depending on the simulation date frequency

dt = mean ( diff ( simDates ));

% Alpha = Speed : mean - reversion speed

Alpha = - coefficients (2)/ dt;

% Level : mean - reversion level

% Level = - coefficients (1)/ coefficients (2);

% Sigma : instantaneous volatility rate

Sigma = nanstd ( residuals )/ sqrt (dt );

%% Setup Hull - White Single Factor Model

if print\_on

fprintf ('Setting up Hull - White Single Factor Model... \n\n')

end

r0 = RateCurveObj.getZeroRates ( Settle +1, 'Compounding ' , -1);

%r0 = swaps.LatestFloatingRate (1);

t0 = Settle ;

% Construct SDE object

FwdRates = RateCurveObj.getForwardRates (...

t0 +1: max ( swaps.Maturity ), ...

'Compounding ', -1);

hullwhite1 = hwv (...

Alpha , ...

@(t,x) func\_hw\_level (t0 ,t, FwdRates ,Alpha , Sigma ), ...

Sigma , ...

'StartState ',r0 );

% Store all model calibration information

calibration.RateCurveObj = RateCurveObj ;

calibration.Tenor = Tenor ;

calibration.ShortRateModel = hullwhite1 ;

calibration.Alpha = Alpha ;

calibration.Sigma = Sigma ;

%% Simulate Scenarios

if print\_on

fprintf ('Simulating Interest Rate Scenarios... \n\n')

end

% Use reproducible random number generator ( vary the seed to produce

% different random scenarios ).

prevRNG = rng (0);

% Compute interest rate scenarios are their respective discount factors

[ scenarios , dfactors ] = ...

hgenerateScenario ( ...

calibration ,...

simDates , ...

numScenarios );

% Restore random number generator state

rng ( prevRNG );

%% Inspect a Scenario

if print\_on

fprintf ('Plotting Yield Curve Evolution for Single Scenarios... \n\n')

end

figure ;

surf (Tenor , simDates , scenarios (: ,: , scenarioIndex ))

axis tight

datetick ('y','mmm yy ');

xlabel ('Tenor ( Months )');

ylabel ('Observation Date ');

zlabel ('Rates ');

set (gca ,'View ' ,[ -49 32]);

title ( sprintf ('Scenario %d Yield Curve Evolution \n', scenarioIndex ));

%% Compute Mark to Market Swap Prices

if print\_on

fprintf ('Computing Mark to Market Swap Prices... \n\n')

end

% Compute all mark -to - market values for this scenario. We use an

% approximation function here to improve performance.

values = hcomputeMTMValues (swaps , simDates , scenarios , Tenor );

% The swap value at time t=0 is not simulated , but set to zero.

% The fixed rate set above assures that the price at t=0 equals zero.

values (1 ,: ,:) = 0;

%% Inspect Scenario Prices

if print\_on

fprintf ('Displaying Single Scenario Prices... \n\n')

end

figure ;

plot ( simDates , values (:,:, scenarioIndex ));

datetick ;

ylabel ('Mark -To - Market Price ');

title ( sprintf ('Swap prices along scenario %d', scenarioIndex ));

%% Visualize Simulated Portfolio Values

if print\_on

fprintf ('Displaying Portfolio Value in all Scenarios... \n\n')

end

% View portfolio value over time

figure ;

totalPortValues = squeeze ( sum( values , 2));

plot ( simDates , totalPortValues );

title ('Total MTM Portfolio Value for All Scenarios ');

datetick ('x','mmm yy ')

ylabel ('Portfolio Value (SEK)')

xlabel ('Simulation Dates ')

%% Compute Exposure by Counterparty

if print\_on

fprintf ('Computing Exposures for each Counterparty... \n\n')

end

instrument\_exposures = zeros ( size ( values ));

unnettedIdx = swaps.NettingID == 0;

instrument\_exposures (:, unnettedIdx ,:) = max ( values (:, unnettedIdx ,:) ,0);

% We compute exposures per netting agreement , but in this case each

% counterparty has only a single netting agreement.

for i = 1: numCounterparties

nettedIdx = swaps.NettingID == i;

numInst = sum ( nettedIdx );

% Exposures for instruments under netting agreements

nettingSetValues = values (:, nettedIdx ,:);

nettedExposure = max ( sum ( nettingSetValues ,2) ,0);

positiveIdx = repmat ( nettedExposure > 0 ,[1 numInst ]);

% Individual instrument contributions to netting set exposure

instrument\_exposures (:, nettedIdx ,:) = nettingSetValues .\* positiveIdx ;

end

% Sum the instrument exposures for each counterparty

exposures = zeros ( numDates , numCounterparties , numScenarios );

for i = 1: numCounterparties

cpSwapIdx = swaps.Counterparty == i;

exposures (:,i ,:) = sum ( instrument\_exposures (:, cpSwapIdx ,:) ,2);

end

% View portfolio exposure over time

figure ;

totalPortExposure = squeeze ( sum ( exposures ,2));

plot ( simDates , totalPortExposure );

title ('Portfolio Exposure for All Scenarios ');

datetick ('x','mmm yy ')

ylabel ('Exposure ( SEK )')

xlabel ('Simulation Dates ')

%% Exposure Profiles

if print\_on

fprintf ('Computing All Exposure Types... \n\n')

end

% Expected Exposure

EEcp = mean ( exposures ,3);

% Expected Positive Exposure : Weighted average over time of EE

% Here computed using a " trapezoidal " approach

simTimeInterval = yearfrac (Settle , simDates , 1);

simTotalTime = simTimeInterval ( end )- simTimeInterval (1);

EPEcp = 0.5 \*( EEcp (1: end -1 ,:)+ EEcp (2: end ,:)) ' ...

\* diff ( simTimeInterval )/ simTotalTime ;

% EPEport = 0.5 \*( EEport (1: end -1)+ EEport (2: end ))' ...

% \* diff ( simTimeInterval )/ simTotalTime ;

%% Compute effective maturity under IMM

EffMaturity = func\_imm\_eff\_maturity ( simDates , dfactors , EEcp );

End

A.6 func imm eff maturity.m

%

% Author : Otto Sjoholm and Dan Franzen

% Created : 2013 -10 -28

% Edited : 2013 -12 -03

%

% Function : func\_imm\_epe.m

%

% Purpose : Computes the effective maturity under IMM

%

% Parameters : simDates - vector of simulation dates on MATLAB serial

% date form

% dFactors - vector of discount factors corresponding to

% the simulation dates

% EE - vectors of expected exposures corresponding to

% the simulation dates

%

% Return : effMaturity - the effective maturity under IMM

%

% Examples :

%

%

%

function [ effMaturity ] = func\_imm\_eff\_maturity ( simDates , dFactors , EE)

global year horizon

% Effective Expected Exposure : Max EE up to time simTimeInterval

EEE = zeros ( size (EE ));

for i = 1: size (EE ,2)

% Compute cumulative maximum

m = EE (1,i);

for j = 1: numel ( simDates )

if EE(j,i) > m

m = EE(j,i);

end

EEE (j,i) = m;

end

end

DF = mean ( dFactors , 2);

DF = DF (2: end ,:);

EE = EE (2: end ,:);

EEE = EEE (2: end ,:);

dt = diff ( simDates ) / 365;

limit = simDates (1) + year ;

hIndex = find ( simDates > limit , 1, 'first ');

Numerator = EE( hIndex :end ,:) '\*( DF( hIndex :end ,:) .\*dt( hIndex :end ,:));

Denominator = EEE (1: hIndex -1 ,:) '\*( DF (1: hIndex -1 ,:).\*dt (1: hIndex -1 ,:));

effMaturity = horizon \* (1 + Numerator. / Denominator );

end

A.7 func cem ead.m

%

% Author : Otto Sjoholm and Dan Franzen

% Created : 2013 -10 -21

% Edited : 2013 -12 -03

%

% Function : func\_cem\_ead.m

%

% Purpose : Computes the expected positive exposure (EPE ) and effective

% maturity under IMM.

%

% Parameters : Maturities - a vector of the maturities of all swap

% instruments

%

% Return : EAD - the exposure at default for each netting

% set ( each counterparty )

%

function [ EAD ] = func\_cem\_ead ( Maturities )

global swapFile cemFile

% Read data from spreadsheets

swapData = dataset ( ...

'XLSFile ', swapFile ,...

'Sheet ', 'Swap Portfolio ');

cemData = dataset ( ...

'XLSFile ', cemFile ,...

'Sheet ', 'CCF ');

% init structs

swaps = struct ( ...

' CounterpartyID ' ,[], ...

'Principal ' ,[], ...

'Maturity ' ,[]);

CCF = struct (...

'Maturity ' ,[], ...

'IR ' ,[], ...

'FX ' ,[], ...

'Equities ' ,[], ...

'Metals ' ,[], ...

'Other ' ,[]);

% fill structs with data

swaps.CounterpartyID = swapData.CounterpartyID ;

swaps.Principal = swapData.Principal ;

swaps.Maturity = datenum ( swapData.Maturity , 'yyyy -mm -dd ');

CCF.Maturity = cemData.Maturity ;

CCF.IR = cemData.InterestRate ;

CCF.FX = cemData.ForeignExchange ;

CCF.Equities = cemData.Equities ;

CCF.Metals = cemData.Metals ;

CCF.Other = cemData.Other ;

numSwaps = numel ( swaps.CounterpartyID );

numCounterparties = max( swaps.CounterpartyID );

rho = 0.85; % Basel value

CE = 0; % current exposure

Collateral = 0; % collateral

EAD = zeros ( numCounterparties ,1);

for i=1: numCounterparties

PFE = [];

MtM = [];

for j=1: numSwaps

if swaps.CounterpartyID (j) == i

%

MtM = [ MtM ; 0]; % MtM equals 0 at time 0

if Maturities (j) < 1

index = 1;

elseif Maturities (j) < 5

index = 2;

else

index = 3;

end

% potential future exposure (add -on)

PFE = [ PFE ; swaps.Principal (j)\* CCF.IR ( index )/100];

end

end

% calculate NGR depending on MtM

if sum ( MtM. ^2) == 0

NGR = 1;

elseif sum (max(MtM , 0)) == 0

NGR = 0;

else

NGR = max ( sum ( MtM ), 0) / sum ( max (MtM , 0));

end

% compute EAD

Addon = (1- rho + rho \* NGR ) \* sum ( PFE );

EAD (i) = CE + Addon - Collateral ;

end

end

A.8 func cem eff maturity.m

%

% Author : Otto Sjoholm and Dan Franzen

% Created : 2013 -10 -21

% Edited : 2013 -12 -03

%

% Function : func\_cem\_eff\_maturity.m

%

% Purpose : Computes the effective maturity under CEM for the given

% swap maturities and principals

%

% Parameters : swapData - structure containing the information

% to price all swaps in the portfolio

% Settle - the settlement date for the swaps

%

% Return : eff\_maturity - the effective maturity for each netting set

% under CEM

%

function [ eff\_maturity ] = func\_cem\_eff\_maturity ( swapData , Settle )

global year

contracts = struct ( ...

' CounterpartyID ' ,[], ...

'Principal ' ,[], ...

'Maturity ' ,[]);

contracts.CounterpartyID = swapData.NettingID ;

contracts.Principal = swapData.Principal ;

contracts.Maturity = ( swapData.Maturity - Settle ) / year ;

numCounterparties = max( contracts.CounterpartyID );

numContracts = length ( contracts.CounterpartyID );

eff\_maturity = zeros ( numCounterparties ,1);

for i=1: numCounterparties

maturities = [];

principals = [];

for j=1: numContracts

if contracts.CounterpartyID (j) == i

maturities = [ maturities ; contracts.Maturity (j)];

principals = [ principals ; contracts.Principal (j )];

end

end

eff\_maturity (i) = max (1, ( maturities '\* principals ) / (sum( principals )));

end

end

A.9 func hw level.m

%

% Author : MathWorks

% Created : -

% Edited : -

%

% Function : func\_hw\_level.m

%

% Purpose : a built in matlab function to compute the mean - reversion

% level of the Hull - White / Vasicek mean - reverting Gaussian

% diffusion model

%

% Parameters : t0 - start time

% dt - time step

% FdwRates - vector of forward rates

% Alpha - mean - reversion speed

% Sigma - instantaneous volatility rate

%

% Return : level - mean reversion level of hwv model

%

function level = func\_hw\_level (t0 ,dt , FwdRates ,Alpha , Sigma )

% Compute the level function for the single factor Hull - White short rate

% model.

% Theta function

t = max (t0 + round (dt \* 365) , t0 +1);

theta = Ft(t) + Alpha \* F(t) + ( Sigma ^2/(2\* Alpha )) \* (1 - exp ( -2\* Alpha \*dt ));

% HW1 level is theta / alpha

level = theta / Alpha / 100;

function r = F(t)

% Instantaneous forward rate

r = FwdRates (t+1- t0 );

end

function dr = Ft(t)

% Derivative of the instantaneous forward rate w/ respect to time

Rates = FwdRates (t-t0:t+2- t0 );

dr = ( Rates (3) - Rates (2)) / (1/365);

end

end