Algorithmica
Research AB

A biased comparison – using Quantlab and Matlab for financial calculations

Reflections by Robert Thorén, Partner, Algorithmica Research
Email: robert.thoren@algorithmica.se

Let me start this comparison on a personal note by admitting that I am Matlab's greatest fan. All through my university years and for some years thereafter, working as a financial consultant, I have pushed the limits of what my Matlab could do for me. Such great relationships can only be superseded by having your own baby. Quantlab is such a brainchild, conceived by several brilliant financial engineers and programmers at Algorithmica Research together with some key customers (not including myself among the brilliant). The new baby talks my language, has a financial engineer's brain, and limbs that fit seamlessly into historical and realtime information, and I get all this without compromising too much on what I love in Matlab. Now, let me get down to business.

The overriding design goal for the Quantlab development environment is to empower the financial engineer with out-of-the-box building blocks on which to test new ideas. Point being – the user should not have to invent the wheel over and over again just to handle user interface interaction and basic financial math, and the result must be a usable application, not only to the developer, but to his non-engineering colleagues at the trading and sales desks. Graphs and tables should be formatted in a way we have become accustomed to in Bill Gates world, i.e. through visual click-and-change.

This document will deal with a comparison of presenting a simple interest rate curve based on a Nelson-Siegel model as a graphic plot. Why take this example? In many applied research projects, the task of building sufficiently smooth interest rate curves from which to extract spot and forward yields is not the main objective. It is merely a necessary building block for more applied analysis. I found a piece of representative and excellent written Matlab code on the web coming from an academic researcher having just such a problem. The only alteration to his Matlab code that I have made has been to reduce the spacing for readability and the financial input data in the example has been updated to present time.

In the next part I will try to show the difference in using a general-purpose high-level development environment and one specifically targeted to get the job done.

Comparing the steps and the code

Let me lay down some ground rules for the task at hand. In the code example used in Matlab the researcher has only used data and instruments from one particular market on a given day. The scope can of course be limited to this one-day and market but it seems unnecessary to limit the analysis, so this "feature" will be generalized in the Quantlab application.

It should also be noted that the Matlab code has been generalized for use in the clone "Octave", but this overhead does not change the job description much.

The steps to compare;

1. get hold of all relevant raw data
2. manipulate raw data to desired format for the analysis at hand
3. calibrate the Nelson-Siegel model to the data
4. extract the result of the model fit in the desired formats
5. plot and format the graphical output
6. look at a screenshot of the resulting window

First presented is the Matlab code snippet and then the subsequent Quantlab one (if there is one). The Matlab code is structured to have a main program calling a number of helper functions that are all presented in appendix B. To really compare the amount of work in building this from scratch, you will have to look at the all the code. The Quantlab code (using the high level language "Qlang") is WYSIWYN - what you see is what you need – and the rest has been taken care of for you.

1. Get hold of all relevant raw data

```
%---------------------------------------------------------------------------
%  BondNSxEstPsTsT.m
%
%  Note:     In MatLab. this function uses fminsearch for optimization.
%            In Octave, it uses mdsmax from the Matrix Computation Toolbox, available
%            (free) from http://www.maths.man.ac.uk/~higham/mctoolbox/.
%  Paul.Soderlind@unisg.ch, April 2002
%---------------------------------------------------------------------------

%Swedish bond data for 3 Sept 2003 (UPDATED BY ROBERT THOREN)

%coupons in %/yr
c  = [ 0        0          0          0          11.5       10.75 ...
       11       13         10.25      6          9 ];

%time to maturity in year
tm = [ 0.00274   0.21096    0.46027    0.88219    1.67397    3.06849 ...
       5.06301   7.46027    9.34795    11.11507   15.30685 ];

%interest rates.  Bills: simple rates in %/yr; bonds: yield to maturity in %/yr
y  = [ 7.75      6.835     6.655      6.41       6.215      6.195 ...
       6.41      6.755     7.01       7.21       7.325 ];
%--------------------------------------------------------------------------
```

A classical case of feeding the analysis with hand punched data. Quantlab could actually take more or less this exact code if one wanted to do it this way, … but we don’t. Historical- and realtime quotes and instrument data are implicitly available to all functions that need them.

2. Manipulate raw data to desired format for the analysis at hand

```
%-----------------------------------------------------------------------------
%transform the data

c  = c'/100; % -> column vector, coupons and yields as 0.05 rather than 5
tm = tm';
y  = y'/100;

vvc = find(c ==0);                %if bill, change from simple to effective rate
y(vvc) = (1+tm(vvc).*y(vvc)).^(1./tm(vvc)) - 1;

pdat = repmat(NaN,length(y),1);         %calculate bond prices
for i = 1:length(y);
  ti = ( mod(tm(i),1):tm(i) )';
  pdat(i) = BondPricePs(c(i),ti,y(i));
end;
%-----------------------------------------------------------------------------
```

Again the Quantlab developer takes a pass. All instruments "know" which conventions they are traded on, which implies that the "rate type switch" and the "yield to price" calculations are not necessary for the Swedish bonds in this example.

3. Calibrate the Nelson-Siegel model to the data

MATLAB:

```
%-----------------------------------------------------------------------------
%estimating parameters in extended Nelson&Siegel model

parX0 = [0.1045,-0.03,-0.0562,1.2,0,0.5];       %starting guess
NSXbR = BondNSxEstPs(parX0,pdat,tm,c,[1e-4,NaN],log(1+y(1)));
%-----------------------------------------------------------------------------
```

QUANTLAB:

```
%-----------------------------------------------------------------------------
% Fit a curve of instruments for a given date to a Nelson-Siegel-Svensson model
% using an equal weighing scheme and Levenberg-Marquardt fitting algorithm
% return value is of type "fit_result" from which curve data can be extracted/plotted
%-----------------------------------------------------------------------------
fit_result fr(curve_name c, date trade) = fit(curve(c, trade), ns_svensson(),
      weights('equal'), lm());
%-----------------------------------------------------------------------------
```

Ok, here I have to start doing some work of my own. As you can see I have to declare variable types in Quantlab. In fact “Qlang”, is a smooth fusion between Matlab like script and more classic programming languages such as Java, VB or C/C++.

Now what is a “fit_result” you might ask? Well it is an object describing the Nelson-Siegel-Svensson interest rate curve for any input curve (pre-defined group of instruments on a certain date). Remember I mentioned in the beginning that I would generalize the analysis to take into account *any* market on any historical date or in realtime. Anytime you use input parameters in a Quantlab function, the appropriate control boxes will be created automatically when the function is dropped to a table or graph.

4. Extract the result of the model fit in the desired formats

MATLAB:

```
%-----------------------------------------------------------------------------
%calculate implied rates (spot, forward, yield to maturity) to plot

tmFig = linspace(eps,max(tm),101)';   %maturities to plot
[shx,fhx,dhx] = BondNSxPs(tmFig,NSXbR(1),NSXbR(2),NSXbR(3),NSXbR(4),NSXbR(5),NSXbR(6));
shx = exp(shx) - 1;     %effective interest rate
fhx = exp(fhx) - 1;


n    = length(c);
ytmx = repmat(NaN,n,1);
for i = 1:n;                %loop over bonds
  ti = ( mod(tm(i),1):tm(i) )';
  [s,f,d] = BondNSxPs(ti,NSXbR(1),NSXbR(2),NSXbR(3),NSXbR(4),NSXbR(5),NSXbR(6));
  Qx = sum(d.*c(i)) + d(length(d));
  ytmx(i) = BondYieldToMat2Ps(Qx,c(i),ti,1,0.05,1e-7);
end;
%-----------------------------------------------------------------------------
```

QUANTLAB:

```
%-----------------------------------------------------------------------------
%Extract the effective spot rate from the model fit
%-----------------------------------------------------------------------------
out series(number) zc(curve_name c, date trade)
{
  % Loop from 0 to 11 years maturity with step 0.1
  return series(t : range(0, 11, 0.1), fr(c,trade).zero_rate(0,t,'effective'));
}
%-----------------------------------------------------------------------------
% Extract the instantaneous forward rates from the model fit
%-----------------------------------------------------------------------------
out series(number) fwd(curve_name c, date trade)
{
     return series(t : range(0, 11, 0.1), fr(c,trade).inst_fwd(t)); %Loop over maturity
}
%-----------------------------------------------------------------------------
% Extract the implied yields for all instruments on
% the curve using the model fit
%-----------------------------------------------------------------------------
out vector(point_number) impl_yield(curve_name c, date trade)
{
     vector(instrument) i = curve(c,trade).instruments; % Place instruments in vector i
     vector(number) y = fr(c,trade).yield(i);          % Calculate implied yields for i
     vector(number) x = (i.maturity-trade)/365;        % Get the maturity in years
     return point(x,y);                                % Return the yield vs. maturity
}

%-----------------------------------------------------------------------------
% Extract the market yields for all instruments on
% the curve on specific date
%-----------------------------------------------------------------------------
out vector(point_number) mkt_yield(curve_name c, date trade)
{
     vector(instrument) i = curve(c,trade).instruments;% Place instruments in vector i
     vector(number) y = i.yield();                     % Get the market yields
     vector(number) x = (i.maturity-trade)/365;        % Get the maturity in years
     return point(x,y);                                % Return the yield vs. maturity
}

%-----------------------------------------------------------------------------
% Extract the coupons of each instrument on the curve
```

```
%------------------------------------------------------------------------------
out vector(point_number) cpn(curve_name c, date trade)
{
     vector(instrument) i = curve(c,trade).instruments;% Place instruments in vector i
     vector(number) y = i.coupon();                    % Get the coupons
     vector(number) x = (i.maturity-trade)/365;        % Get the maturity in years
     return point(x,y);                                % Return the yield vs. maturity
}
%------------------------------------------------------------------------------
% Get labels to attach to the instruments in the curve
%------------------------------------------------------------------------------
out vector(string) myLabels(curve_name c, date trade)
{
     vector(instrument) i = curve(c,trade).instruments;
     return i.name();
}
%------------------------------------------------------------------------------
```

Now the difference is really starting to show. First you can see that I have created five different “out” functions, one for each of the results I want to display. I could as easily created one function returning all of them in a 5xN matrix, much like the Matlab code. I chose to have different functions because of the nice way I later can attach multiple instances of any of them in graphs or tables using simple drag-and-drop.

Further you can see that in the Quantlab code there are no real financial or mathematical calculations going on. The most advanced being the calculation of maturity as a year fractions using the instruments maturity date as the starting point. It is just about fetching and displaying the data.

I have also gone the extra mile here to produce a readable output. In Quantlab you will see dynamic labels attached to the points representing the instruments (and hence the last bit of code that is not included in the Matlab equivalent).

5. Plot and format the graphical output

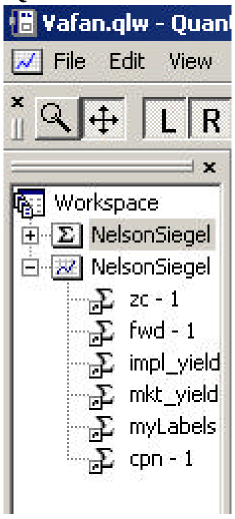
MATLAB:

```
%-----------------------------------------------------------------------------
%plotting

if exist('OCTAVE_VERSION');
  title('Swedish Interest Rates 29 Dec 1993');
  xlabel('Settlement date (years from trade date)');
  ylabel('per yr, continuously compunded');
  plot(tm,c,'@2;Coupon;',tm,y,'@7;YTM;',tmFig,shx,'0;Est spot rate;',...
       tmFig,fhx,'-1;Est forward rate;',tm,ytmx,'@16;Est YTM;');
else;
  set(0,'DefaultAxesColorOrder',[0 0 0]);
  set(0,'DefaultTextFontName','Times-Roman');
  set(0,'DefaultAxesFontName','Times-Roman');
  ha = plot(tm,c,tm,y,tmFig,shx,tmFig,fhx,tm,ytmx);
  set(ha,{'LineStyle'},{ 'none';'none';'--';'-';'none'}, ...
         {'Marker'},{'+';'s';'none';'none';'.'});
  pos = get(gca,'Position');
  set(gca,'Position',pos./[1 1 1 1.37]);
  axis([0,16,0,0.12]);
  title('Swedish Interest Rates 29 Dec 1993')
  xlabel('Years to Maturity');

  legend('Coupon rate','Yield to maturity','Estimated spot rate', ...
         'Estimated forward rate','Estimated yield to maturity',4);
end;
%-----------------------------------------------------------------------------
```

QUANTLAB:



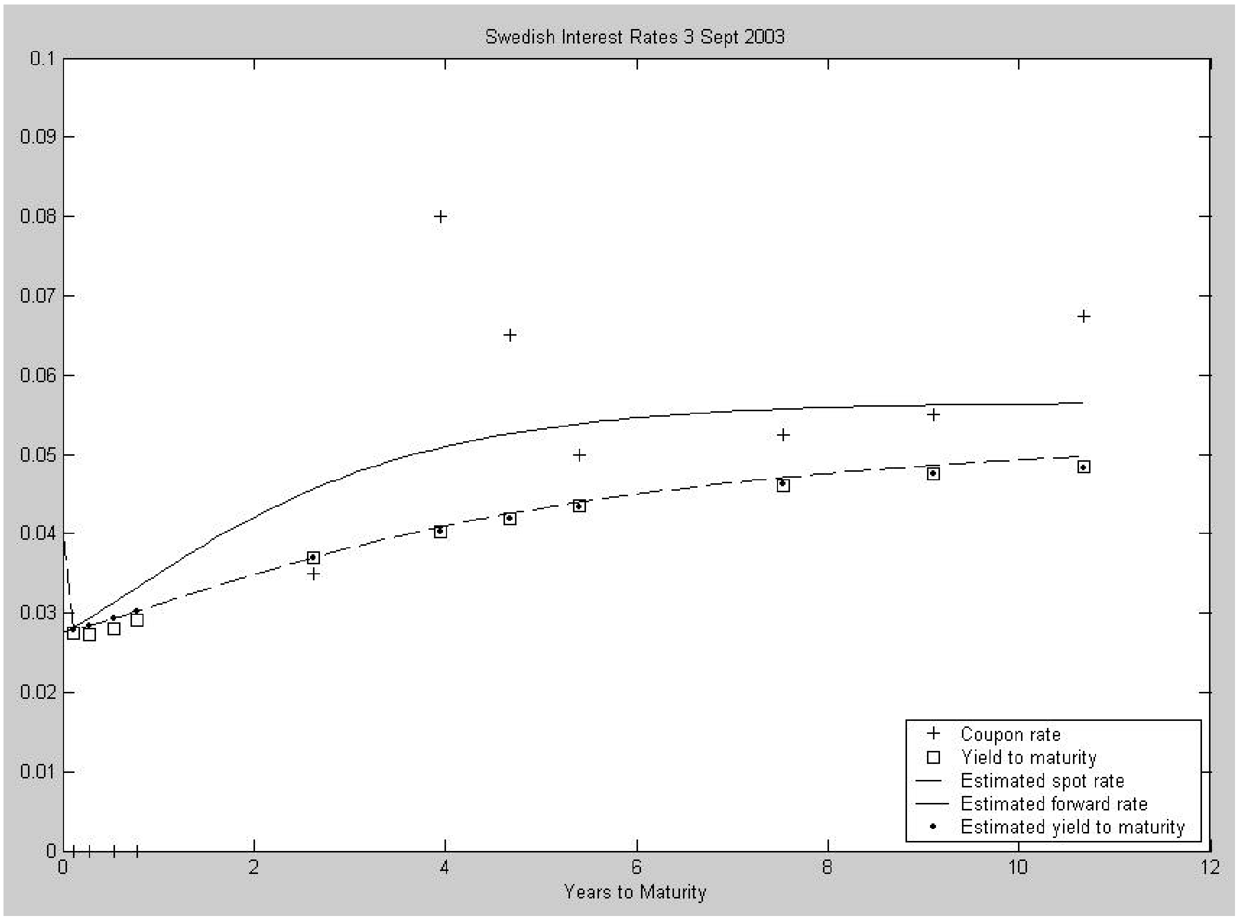
```
Workspace
+ NelsonSiegel
- NelsonSiegel
    zc - 1
    fwd - 1
    impl_yield
    mkt_yield
    myLabels
    cpn - 1
```

Hmm, … I have to get visual to compare now. Matlab, as you certainly know, uses code to produce graphics. Quantlab will display all my “out” declared functions in a workspace browser. The picture above show that I have already created a graph window called “NelsonSiegel” and drag-and-dropped all the relevant functions into it.

Algorithmica Research AB

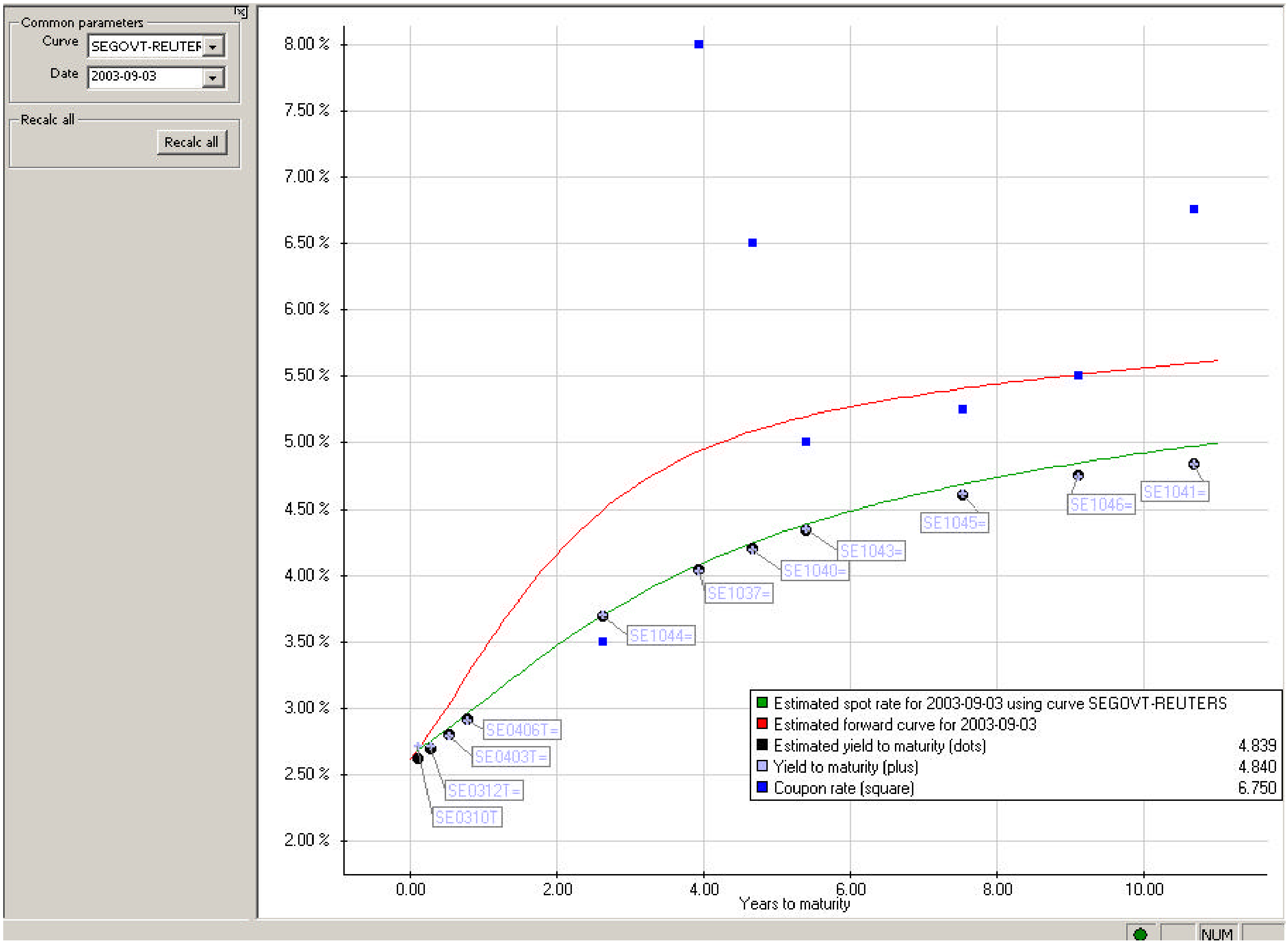
6. Screenshots of the resulting windows

MATLAB:



Algorithmica Research AB

QUANTLAB:



🟩	Estimated spot rate for 2003-09-03 using curve SEGOVT-REUTERS	
🟥	Estimated forward curve for 2003-09-03	
■	Estimated yield to maturity (dots)	4.839
■	Yield to maturity (plus)	4.840
🟦	Coupon rate (square)	6.750

The comparison is complete. The result is quite similar isn't it?

Concluding thoughts

Perhaps the comparison is not entirely fair, but then again, I never said it was unbiased. The main points to be made are,

1. It is nice to have all supporting data for financial calculations implicitly available when programming. Even better, as in my case, data automatically updated by a server, each and every day, enabling an up to date analysis every time I open this particular workspace.

2. It is nice to have a supported, market proven, and prepackaged library of important financial and mathematical functions to use. Personally, I wouldn't even try to implement the Nelson-Siegel model from scratch unless someone held a gun to my head.

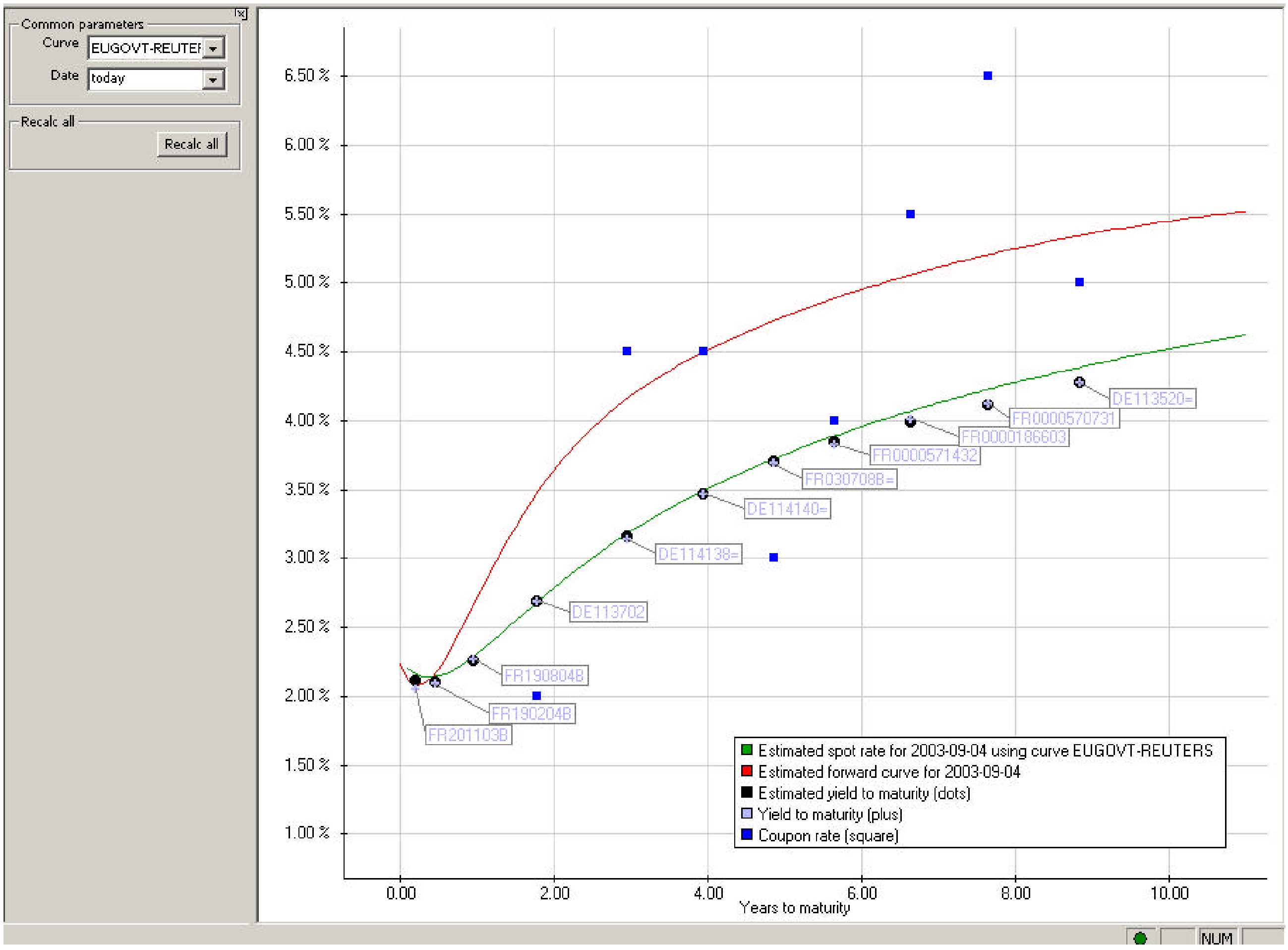
3. I have learned to depend on graphical formatting the way you do it in Excel. I don't have to go back to school to learn how to get a descent looking plot. Just point and click …

There is also a small issue of performance. However techno and boring, the Matlab analysis took 3.6 seconds to perform on my desktop and less than 0.05 seconds in Quantlab (which certainly helps if you want the Euro curve in realtime, as the example shows in appendix a). If you want the 4-year forward rate picked out and plotted for the last 200 days using the same method you get the picture. (Couldn't help showing this example in the appendix as well.) Now performance will matter, also to the user on the trading or sales desk, and imagine getting this done in Excel…

Algorithmica Research AB

APPENDIX A

To show the same analysis applied to another market (curve) but now displayed in realtime I have attached the same window but using the Reuter definition of the EUGOVT curve. The date "today" will automatically trigger Quantlab to go the realtime source for all market quotes. Dynamic legends and labels will change automatically as well.



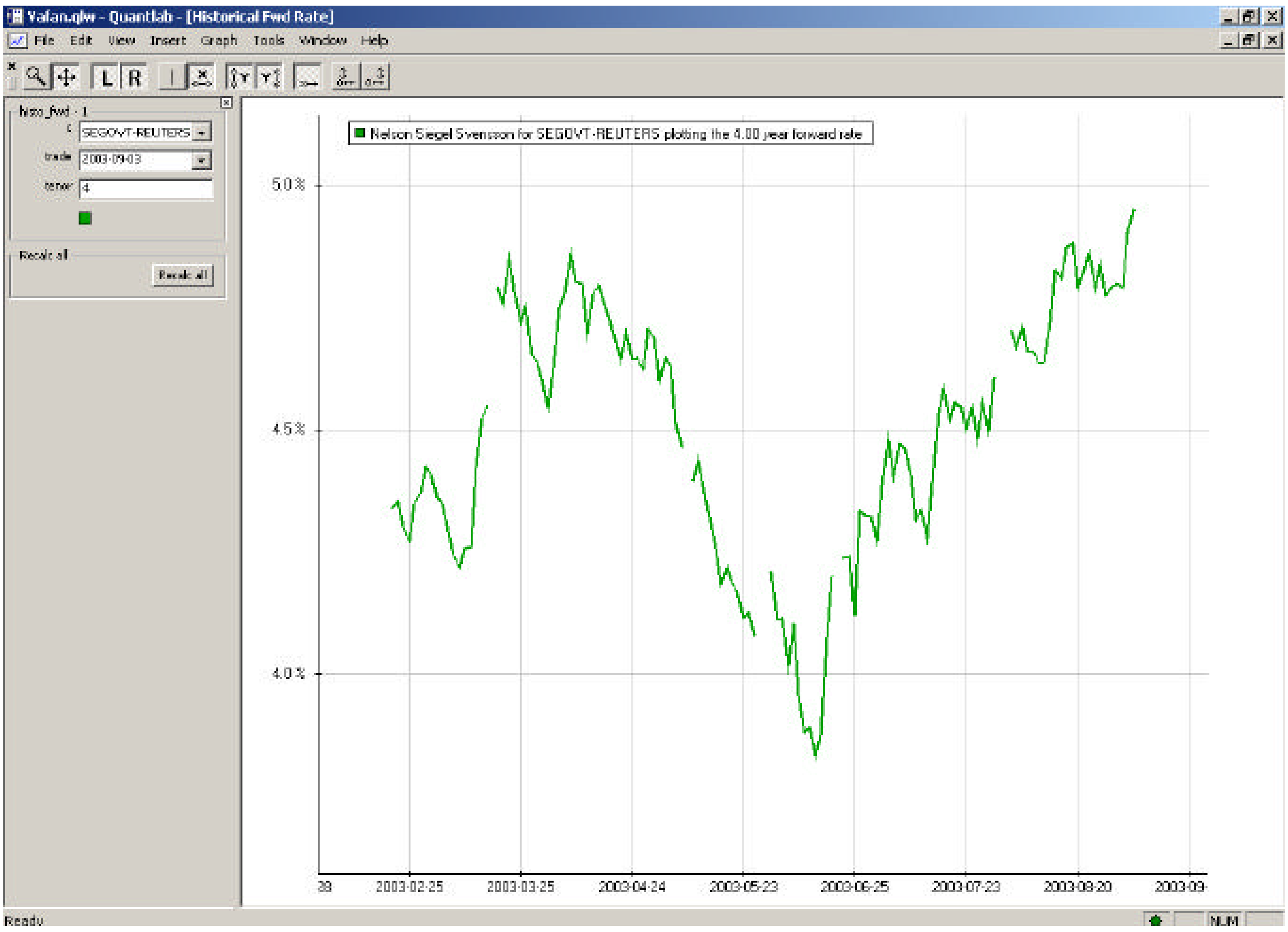
Common parameters

Curve EUGOVT-REUTEF

Date today

I added the following lines of code to create a historical (200 days) N-year instantaneous forward rate plot. I'm still using the Nelson Siegel Svensson model to estimate the complete curve for every date and then extracting the N-year tenor from the "fit_result". The knowledge of which instruments that were on the curve for a given date is all implicit in the data model. Pretty neat, no!?

```
%---------------------------------------------------------------------------
% The instataneous forward rate from the model fit for historical dates.
%---------------------------------------------------------------------------
out series(number) histo_fwd(curve_name c, date trade, number tenor)
{
     return series(t : range(trade - 200,trade), fr(c,t).inst_fwd(tenor)); % Loop over
dates
}
%---------------------------------------------------------------------------
```


Vafan.qlw - Quantlab - [Historical Fwd Rate]

histo_fwd - 1
c: SEGOVT-REUTERS
trade: 2003-09-03
tenor: 4

🟩 Nelson Siegel Svensson for SEGOVT-REUTERS plotting the 4.00 year forward rate

Algorithmica
Research AB

APPENDIX B

```
function dQ_dy = BonddPricePs(c,t,y);
%dBondPricePs    Calculates derivative of bond price wrt. yield: the gradient of
BondPricePs(.).
%
%
%  Usage:     dQ_dy = BonddPricePs(c,t,y);
%
%  Input:     see BondPricePs(.)
%
%  Output:    dQ_dy   -   Nx1 vector, derivative of bond price wrt. yield
%
%
%
%
%
%  Paul Soderlind (Paul.Soderlind@unisg.ch), 4 June 1997, March 2001
%-----------------------------------------------------------------------

c = c(:);               % -> column vector
t = t(:);
y = y(:);

nc = length(c);
m  = length(t);
n  = length(y);

y = repmat(y',m,1);          %nx1 -> mxn
t = repmat(t ,1,n);          %mx1 -> mxn matrix

if nc==1;                      %scalar c
  c = repmat(c,m,n);
elseif nc==n;                  %one c given for every bond (as row vector)
  c = repmat(c',m,1);
end;


cfac  = c./((1+y).^t);               %c/(1+y)^t1 + c/(1+y)^t2 + ...+ c/(1+y)^m
cfac  = cfac .* (-t./(1+y));
dQ_dy =  sum(cfac,1) + (-t(m,:)./(1+y(m,:)))./((1+y(m,:)).^t(m,:));  %theoretical price
dQ_dy = dQ_dy';
%------------------------------------------------------------------------

function NSb = BondNSxEstPs(par0,Q,tm,c,opt,s0);
%BondNSxEstPs    Estimates parameters in (extended) Nelson-Siegel yield curve model by
%              non-linear least squares (minimizing squared differences between
%              actual and fitted bond prices).
%
%
%
%  Usage:    NSb = BondNSxEstPs(par0,Q,tm,c,opt,s0);
%
%
%  Input:    par0       4x1 or 6x1 vector, initial guess of paremeters
%                         if 4x1: standard NS with par0 = [b0,b1,b2,tau]
%                         if 6x1: extended NS with par0 = [b0,b1,b2,tau,b3,tau2]
%            Q          nx1 vector, data on bond prices, eg. 1.01
%            tm         nx1 vector, data on time to maturity (in years), eg. 2.54
%            c          nx1 vector, data on bond coupons, eg. 0.06
%            opt        2x1  vector, [convergence criterion, max iterations],
%                       eg. [1e-4,500]. A NaN in either places implies that
%                       a default value is used.
%            s0         scalar, restricted value of short rate:
%                       if [] no restriction, else b0+b1=s0 is imposed
%
%  Output:   NSb        1x4 or 1x6 vector, estimated parameters
```

```
%
%
%
%  Note:     For more details, see
%            (a) Svensson (1995), "Estimating Forward Interest Rates with
%                the Extended Neolson & Siegel Method," Quarterly Review,
%                Sveriges Riksbank, 1995:3, 13-26.
%            (b) Soderlind and Svensson (1997), "New Techniques to Extract
%                Market Expectations from Financial Instruments,"
%                Journal of Monetary Economics 40, 383-429.
%
%  Note:     In MatLab. this function uses fminsearch for optimization.
%            In Octave, it uses mdsmax from the Matrix Computation Toolbox, available
%            (free) from http://www.maths.man.ac.uk/~higham/mctoolbox/.
%
%  Calls on: mdsmax (if Octave)
%
%
%  Paul.Soderlind@unisg.ch, April 2002
%-----------------------------------------------------------------------------


Qtc = [Q,tm,c];


if length(par0) == 4;         %standard Nelson-Siegel
  if isempty(s0)==0;          %b1 = s0 - b0 is then imposed by BondNSxLossPs
    par0 = par0([1,3,4]);
  end;
elseif length(par0) == 6;     %extended Nelson-Siegel
  if isempty(s0)==0;          %b1 = s0 - b0 is imposed by BondNSxLossPs
    par0 = par0([1,3,4,5,6]);
  end;
end;


if isnan(opt(1));                 %optimization options, use defaults in opt(i)=NaN
  tol = 1e-4;
else;
  tol = opt(1);
end;
if isnan(opt(2));
  MaxIter = 800;
else;
  MaxIter = opt(2);
end;

if exist('OCTAVE_VERSION');
  [NSb,fval,nf] = mdsmax('BondNSxLossPs',par0,[tol,MaxIter,inf,0,0],[],Qtc,s0);
  if nf == MaxIter;
    disp(' ');warning('Maximum number of iterations reached without convergence');disp('
');
  end;
else;
  options = optimset('Display','notify','TolX',tol,'MaxIter',MaxIter);
  [NSb,fval,exitflag,output] = fminsearch('BondNSxLossPs',par0,options,Qtc,s0);
  if output.iterations == MaxIter;
    disp(' ');warning('Maximum number of iterations reached without convergence');disp('
');
  end;
end;

if length(NSb) == 3;         %standard NS with restriction
  NSb = [NSb(1),g_s0-NSb(1),NSb(2),NSb(3)];
elseif length(NSb) == 5;     %extended NS with restriction
  NSb = [NSb(1),s0-NSb(1),NSb(2),NSb(3),NSb(4),NSb(5)];
end;
%----------------------------------------------------------------------------
```

```
function Loss = BondNSxLossPs(b,Qtc,s0);
%NS2xLoss    Defines loss function for bond prices in extended Nelson-Siegel model.
%            Used for estimation of the parameters in the model by minimizing
%            Loss (squared price deviations).
%
%
%  Usage:    Loss = BondNSxLossPs(b);
%
%  Input:    b       3x1, 4x1, 5x1, or 6x1 vector with parameters in Nelson-Siegel model
%                      if 3x1: [b0,b2,tau], NS with restriction that b1 = g_s0 - b0
%                      if 4x1: [b0,b1,b2,tau], NS without restrictions
%                      if 5x1: [b0,b2,tau,b3,tau2], extended NS with restriction that b1
= g_s0 - b0
%                      if 6x1: [b0,b1,b2,tau,b3,tau2], extended NS without restrictions
%
%  Output:   Loss    scalar, sum of squared differences between implied and actual bond
%                    prices
%
%
%
%  Note:     For more details, see
%            (a) Svensson (1995), "Estimating Forward Interest Rates with
%                the Extended Neolson & Siegel Method," Quarterly Review,
%                Sveriges Riksbank, 1995:3, 13-26.
%            (b) Soderlind and Svensson (1997), "New Techniques to Extract
%                Market Expectations from Financial Instruments,"
%                Journal of Monetary Economics 40, 383-429.
%
%
%  Calls on: BondNSxPs
%
%  Paul.Soderlind@unisg.ch, 8 April 2002
%-------------------------------------------------------------------------------


Q  = Qtc(:,1);           %data on bond prices
tm = Qtc(:,2);           %data on time to maturity, fraction of years
c  = Qtc(:,3);           %data on coupons

n = length(c);           %number of bonds


if length(b) == 3;            %standard Nelson-Siegel with restriction b1 = s0-b0
  b0   = abs(b(1));
  b1   = s0 - b0;
  b2   =     b(2);
  tau  = abs(b(3));
  b3   = 0;
  tau2 = 1;
elseif length(b) == 4;        %standard Nelson-Siegel
  b0   = abs(b(1));
  b1   =     b(2);
  b2   =     b(3);
  tau  = abs(b(4));
  b3   = 0;
  tau2 = 1;
elseif length(b) == 5;        %extended Nelson-Siegel with restriction b1 = s0-b0
  b0   = abs(b(1));
  b1   = s0 - b0;
  b2   =     b(2);
  tau  = abs(b(3));
  b3   =     b(4);
  tau2 = abs(b(5));
elseif length(b) == 6;        %extended Nelson-Siegel
  b0   = abs(b(1));
  b1   =     b(2);
  b2   =     b(3);
  tau  = abs(b(4));
  b3   =     b(5);
  tau2 = abs(b(6));
end;


QNS = repmat(NaN,n,1);
for i = 1:n;                        %loop over bonds
  ti = ( mod(tm(i),1):tm(i) )';    %vector for coupon stream
  [s,f,d] = BondNSxPs(ti,b0,b1,b2,tau,b3,tau2);  %NSx: spot, forward, discount fn
  QNS(i) = sum(d.*c(i)) + d(length(d));      %theoretical bond price
end;

Loss = sum( (QNS - Q).^2 );             %squared price deviations

if exist('OCTAVE_VERSION');
  Loss = -Loss;     %Using maximization routine in Octave
end;
%-------------------------------------------------------------------------------

function [s,f,d] = BondNSxPs(m,b0,b1,b2,tau,b3,tau2);
%NS2x    Extended Nelson and Siegel (1987) spot rate, forward rate, and discount
function
%
%
%  Usage:    [s,f,d] = BondNSxPs(m,b0,b1,b2,tau,b3,tau2); or
%                      BondNSxPs(m,b0,b1,b2,tau);
%
%  Input:    m         NxK matrix  times to maturity in years
%            b0        scalar, beta0 parameter in NS
%            b1        scalar, beta1 parameter in NS
%            b2        scalar, beta2 parameter in NS
%            tau       scalar, tau   parameter in NS
%            b3        scalar, beta2 parameter in extended NS
%            tau2      scalar, tau2  parameter in extended NS
%
%
%
%  Output:   s         NxK matrix,  spot rate (continously compounded)
%            f         NxK matrix,  forward rate (continously compounded)
%            d         NxK matrix,  discount rate
%
%
%
%  Note:     For more details, see
%            (a) Svensson (1995), "Estimating Forward Interest Rates with
%                the Extended Nelson & Siegel Method," Quarterly Review,
%                Sveriges Riksbank, 1995:3, 13-26.
%            (b) Soderlind and Svensson (1997), "New Techniques to Extract
%                Market Expectations from Financial Instruments,"
%                Journal of Monetary Economics 40, 383-429.
%
%
%
%  Paul.Soderlind@unisg.ch, April 2002
%-----------------------------------------------------------------------------*/

if nargin == 5;                     %standard Nelson-Siegel
  b3   = 0;
  tau2 = 1;
end;

                                               %forward rate
f =  b0 + b1*exp(-m/tau) + b2*(m/tau).*exp(-m/tau) + b3*(m/tau2).*exp(-m/tau2);


s =  b0 + b1*(1 - exp(-m/tau))./(m/tau) ...
        + b2*((1 - exp(-m/tau)) ./(m/tau)  - exp(-m/tau)) ...
        + b3*((1 - exp(-m/tau2))./(m/tau2) - exp(-m/tau2));        %spot rate
```

```
d = exp( -s.*m );             %discount function
%--------------------------------------------------------------------------

function Q = BondPricePs(c,t,y);
%BondPricePs   Calculates price from the yield to maturity and the payment stream
(arbitrary periods).
%
%  Usage:     Q = BondPricePs(c,t,y);
%
%  Input:     c      scalar or nx1, coupon rate, e.g. 0.06 or 0.09/2
%             t      mx1 vector, dates of coupon payments. Principal
%                    is paid at the same time as the last coupon. Dates
%                    should be expressed as fractions of the period.
%             y      nx1 vector, effective yield to maturity, e.g. 0.07.
%
%  Output:    Q      nx1 vector, bond price (eg. 1.01)
%
%
%  Note: (1) the following is calculated:
%
%             Q =  c/(1+y)^t(1) + c/(1+y)^t(2) + ... + (1+c)/(1+y)^t(m)
%
%        (2) With semi-annual coupons after 2,8, and 14 months, use
%            t = [0.333;0.333+1;0.333+2] to calculate semiannual yield.
%
%
%  Example:        t = (1:2)';
%                  c = [0.09;0.10];
%                  y = [0.0626;0.07];  gives Q = [1.05;1.05]
%
%
%
%   Paul Soderlind (Paul.Soderlind@unisg.ch), 3 June 1997, Mar 2001
% -----------------------------------------------------------------------

c = c(:);                    % -> column vector
t = t(:);
y = y(:);

nc = length(c);
m  = length(t);
n  = length(y);


y = repmat(y',m,1);          %nx1 -> mxn
t = repmat(t ,1,n);          %mx1 -> mxn matrix

if nc==1;                      %scalar c
  c = repmat(c,m,n);
elseif nc==n;                  %one c given for every bond (as row vector)
  c = repmat(c',m,1);
else;
  error('dimensions of c, t, and y are not correct');
end;


cfac = c./((1+y).^t);               %c/(1+y)^t1 + c/(1+y)^t2 + ...+ c/(1+y)^m
Q    = sum(cfac,1) + 1./((1+y(m,:)).^t(m,:));  %theoretical price
Q    = Q';
%-------------------------------------------------------------------------

function y = BondYieldToMat2Ps(Q,c,t,Method,yLH,tol);
%BondYieldToMat3Ps    Calculates yield to maturity from bond price (several methods
available).
%
%
%  Usage:     y = BondYieldToMat2Ps(Q,c,t,Method,yLH,tol);
%
%
```

```
%  Input:     Q        nx1 vector, bond prices (for instance, 1.01)
%             c        scalar or nx1 vector, c rate, e.g. 0.06, or 0.09/2
%             t        mx1 vector, dates of coupon payments. Principal
%                      is paid at the same time as the last c. Dates
%                      should be expressed as fractions of the period.
%                      Example: coupons after 2,8, and 14 months ->
%                      t = [0.333;0.333+1;0.333+2]
%             Method   1: Newton-Raphson; 2: bisection
%             yLH      if Method==1: scalar, yLH(1) is initial guess of roots
%                      if Method==2: 2x1 vector, yLH(1) is lower boundary and yLH(2)
upper boundary
%             tol      scalar, convergence criterion for y, eg. 1e-7
%
%  Output:    y        nx1 vector, effective yield to maturity (per period)
%
%
%
%  Note:      (1)  Method 1 (Newton-Raphson) is pretty fast. Works with arbitrary coupon
%                  periods. The Newton-Raphson iterations are based on
%                  Q = F(y0) + J0*Dy, where Dx = y1-y0 and J0 is the Jacobian at y0.
%                  We choose Dy to make this exact, that is as  (Q - F(y0))./J0 = Dy
%
%             (2)  Method 2 (bisection) is a bit slow, but very robust. Works with
arbitrary
%                  coupon periods.
%
%
%
%  Calls on:  BonddPricePs
%
%
%  Paul Soderlind (Paul.Soderlind@unisg.ch), March 2002
%-----------------------------------------------------------------------------

Q = Q(:);                      % -> column vectors
c = c(:);
t = t(:);

n  = length(Q);
nc = length(c);
m  = length(t);
%---------------------------------------

if Method == 1;                   %Newton-Raphson

  y0 = repmat(yLH(1),n,1);

  Dy = 1E+198;
  while max( abs(Dy) ) > tol;
    J0 = BonddPricePs( c,t,y0 );  %derivative wrt y, nx1
    F0 = BondPricePs(  c,t,y0 );  %F(y0), nx1
    Dy = (Q - F0)./J0;               %nx1
    y = y0 + Dy;
    y0 = y;
  end;
%---------------------------------------

elseif Method == 2;                  %bisection

  yL = repmat(yLH(1),n,1);        %lower boundary for yield
  yH = repmat(yLH(2),n,1);        %upper boundary
  if any(yL>yH)
    error('Lower bound greater than upper bound');
  end

  QL = BondPricePs(c,t,yL);       %call on function to calculate theoretical price
  QH = BondPricePs(c,t,yH);
  if any( (QL < Q) | (QH > Q));          %note: decreasing function so QL > QH
    error('roots not bracketed');
  end;                                   %done checking for errors

  while any( (yH-yL)>tol );            %iteration loop
    y = (yL + yH)/2;                   %mid point for yield
    %disp([yL,y,yH]);
    Qs = BondPricePs(c,t,y);             %price at guessed yield
    vvH     = find(Qs > Q);              %indices where Qs > Q  (decreasing function)
    yL(vvH) = y(vvH);                    % => root must be higher than y
    vvL     = find(Qs < Q);              %indices where Qs < Q
    yH(vvL) = y(vvL);                    %=> root must be lower than y
  end;                                   %note x([])= z does nothing


end;             %end different methods
%-------------------------------------------------------------------------------
```