Forward Rate Curve Models

CS522 – Spring 2005

Tibor Janosi

Waggoner's Model

$$\min\left(\sum_{i=1}^{n}(P_i - \hat{P}_i(f))^2 + \int_0^T \lambda(s)(f''(s))^2 ds\right)$$

$$\lambda(t) = \begin{cases} 0.1, & 0 \le t < 1 \\ 100, & 1 \le t \le 10 \\ 100,000, & 10 < t \end{cases}$$

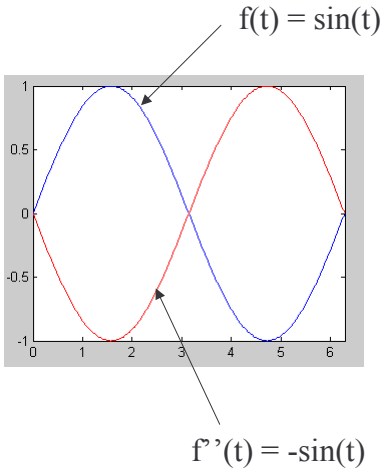
price error

smoothness term

Note: For simplicity, we use a one-argument notation for f, f(t, T) => f(T – t). Thus, for example, r(t) = f(t, t) => f(0).

Smoothness and 2nd Derivative

- Frequently used as a measure of smoothness; other measures are also possible.
- We want to capture the contribution of the entire curve, hence we integrate.
- Why square the second derivative?



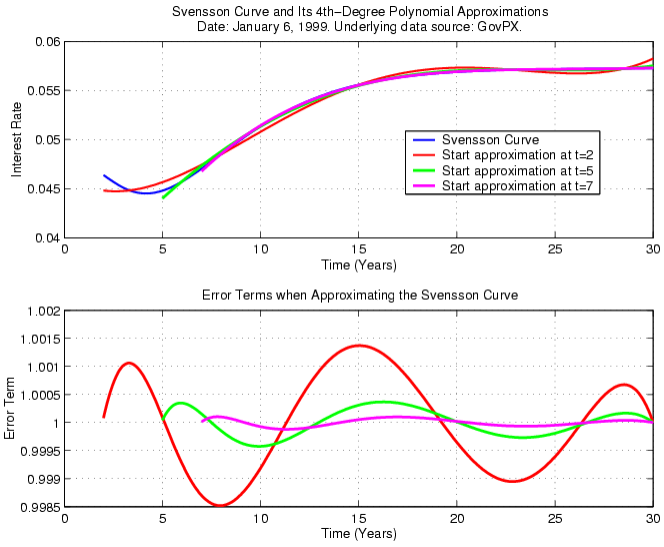
Waggoner's Model (2)

- We have not specified function f(t); in principle any suitable function can be used.
- Waggoner uses cubic splines.
- Constants are arbitrary
- The time intervals are less arbitrary, especially at the short end. Why?

Waggoner's Model (3)

- The tail (long end) of the Waggoner curve will be stiff; it resembles Svensson.
- The short end is flexible, for two reasons:
  (a) Low smoothness penalty constant;
  (b) Short interval.
- In other words: Let the data shape the curve at the short end, do what you can at the long end.

Polynomials vs. Svensson



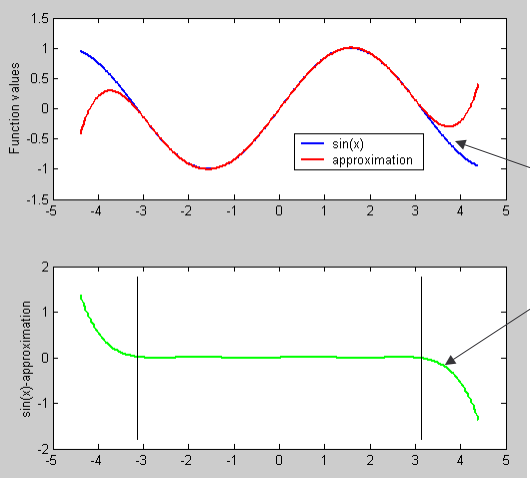
Polynomials vs. Svensson (2)

- Good: Approximation of Svensson curves by polynomials works well even for very long intervals.
- Bad: Polynomials oscillate around the curve they approximate. This is typical for splines.
- Good: Oscillations are small if we approximate from t =7, even from t = 5.

Polynomials vs. Svensson (3)

- Good: While polynomials can approximate Svensson, they could also describe other shapes. We can thus get rid of asymptotically horizontal curves.
- Bad: Polynomials are notoriously bad when extrapolated. In the limit, when t -> infinity, all non-constant polynomials will have infinite values.

Polynomial Extrapolations



deviation increases rapidly!

$\sin(x) \sim 0.00564312x^5-0.155271x^3+0.987862x$ on $[-\pi, \pi]$

Differential Information

- Why is the long end (almost) straight?
- The forward rate curve embeds some information about the (expected) future.
- The marked can not distinguish between t_1 and t_2, when these are "close" and "far out" toward the long end.
- Hence, only the most general tendencies are expressed in the curve.

Smoothest Forward Rate Curves

- We have large and arbitrary constants in the penalty term; why not look for the "smoothest" curve?
- But is the short end "smoothest"?
- In some sense, yes, but we need the curve to follow the data.
- We'll choose short intervals at the left end.

The Problem

$$\min \int_0^T (f''(t))^2 dt$$ ← minimize smoothness penalty

$$\int_0^{t_i} f(t)dt = -\ln p_i,\ i = 1, n$$

$$0 < p_i < 1$$

We know the prices of n instruments (0-coupon bonds); p_i is the price of a 0-coupon bond paying one default-free dollar at time t_i.

Note: We assume that f is "nice." Is this reasonable?

The Solution

- Such problems can be solved, for example, by using the calculus of variations.
- The optimal solution is a piecewise 4^{th}-degree polynomial, such that f, f', f'', f''' are continuous at all interior knot points t_i
- Given n prices p_i, can we determine the curve? We have n intervals, and need 5n coefficients.

The Equations

- 4(n-1) equations can be obtained from the conditions at the interior points on f, f', f'', f'''.
- Let us not forget the n price equations given initially.
- We need 4 more equations. Where do we find them?
- We impose conditions at the ends of the curve!

Conditions at the Ends

End	**Conditions at** $t = t_e$
"fixed"	$f(t_e)$, $f'(t_e)$ given
"free"	$f''(t_e) = f'''(t_e) = 0$
"supported"	$f(t_e)$-given, $f''(t_e) = 0$
"other"	$f'(t_e)$-given, $f'''(t_e) = 0$

Conditions can be imposed independently at the ends of the curve.
One can thus have 16 “smoothest” curves.
Does this make sense? What conditions should one choose?

The System of Equations

$left_1$			...	
$left_2$			...	
$price$			...	
f	$-f$		...	
f'	$-f'$		...	
f''	$-f''$		...	
f'''	$-f''$		...	
	$price$		...	
	f	$-f$	...	
	f'	$-f'$	...	
	f''	$-f''$	...	
	f'''	$-f'''$	...	
...	...	...	...	...
				$-f$
				$right_1$
				$right_2$

Each row is an equation; each column corresponds to a polynomial coefficient. The first 5 columns represent the coefficients of the first polynomial, defined on [0, t_1], etc.
All right-hand values are 0, except the ones that relate to prices and – possibly – the ones that relate to end-conditions.
For large n, the resulting system of equations is sparse (most values on the left-hand side will be 0).

An Example

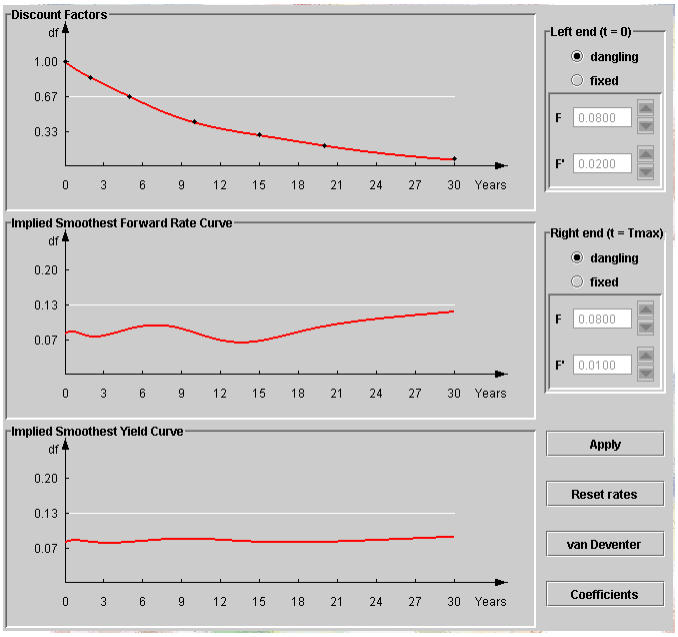
$f(t) = a_{i4}t^4 + a_{i3}t^3 + a_{i2}t^2 + a_{i1}t + a_{i0}, t_{i-1} \le t \le t_i$
$t_0 = 0 < t_1 = 1 < t_2 = 2$

	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_{24}	a_{23}	a_{22}	a_{21}	a_{20}		RHS
f'(0)			2								=	
f''(0)		6									=	
p(1)	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{2}$	1						=	$\log \frac{1}{p_1}$
f(1)	1	1	1	1	1	-1	-1	-1	-1	-1	=	
f'(1)	4	3	2	1		-4	-3	-2	-1		=	
f''(1)	12	6	2			-12	-6	-2			=	
f'''(1)	24	6				-12	-6				=	
p(2)						$\frac{31}{5}$	$\frac{15}{4}$	$\frac{7}{3}$	$\frac{3}{2}$	1	=	$\log \frac{p_1}{p_2}$
f'(2)						48	12	2			=	
f''(2)						48	6				=	

Note: 1. $\left.\begin{array}{l} \int_0^{t_i} f(t)dt = -\log p_i \\ \int_0^{t_{i-1}} f(t)dt = -\log p_{i-1} \end{array}\right\} \Rightarrow \int_{t_{i-1}}^{t_i} f(t)dt = \log \frac{p_{i-1}}{p_i}$

2. p(0) = 1 (why?)

A Live Demo…



Overview of the Method

- We choose knot points t_i; if we knew parameters p_i and the conditions at the end of the curve, then we could compute the smoothest curve.
- But we do not know p_i! We know only prices of Treasuries, which a mixture of bills, notes and bonds.
- We use an optimization algorithm to find the "best" p_i, so that we can reproduce observed prices.

Overview of the Method (2)

- We start with "reasonable" parameter values, e.g. we set p_i to the price of a zero-coupon bond that has the same yield as the bond whose leftover maturity is closest to t_i.
- We then change the values of the parameters to minimize a suitably chosen measure of the error:

$$\min \sum_{i=1}^{n} (P_i - \hat{P}_i(f))^2$$

There is More to This…

- Minimizing the sum of squared errors is not enough, the size and distribution of errors must be "right."
- Smoothest forward rate curves can have odd behavior (see applet).
- Other conditions can be imposed: for example, the short rate should not be higher than the federal funds rate.

We will blissfully ignore most of these issues…

Matlab

- A great tool for numerical computations.
- We do not have resources to teach Matlab from scratch, but we are available to help.
- If you did not have exposure to it, you should do your best to learn it **fast**.
- Just knowing the syntax will not be enough.
- Learn more than syntax, get into the spirit of the language. Do you know what "vectorization" means? Do you use it?

Matlab (2)

- We will show you many useful functions and pieces of code. However, you should browse the documentation on your own.
- Read the help on **lang**, **elmat**, **elfun**, **specfun**, **matfun**.
- We will make ourselves available for consultations on Matlab and the homework. Do not hesitate to talk to us if you have difficulties!

Time vs. Simplicity in Matlab

```
>> for N = [100, 500]
  tic; for i = 1:N; for j = 1:N; a(i, j) = 17; end; end; toc
  tic; b(N, N) = 17; for i = 1:N; for j = 1:N; b(i, j) = 17; end; end; toc
  tic; c(1:N, 1:N) = 17; toc
  clear all;
end
```

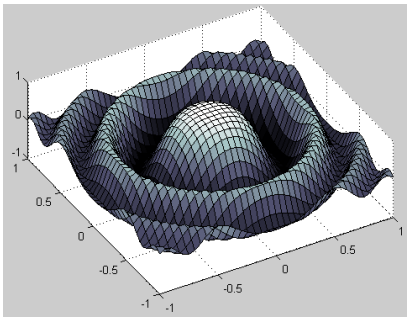
	N = 100	N = 500
	0.080	5.578
	0.040	1.192
	0	0.010

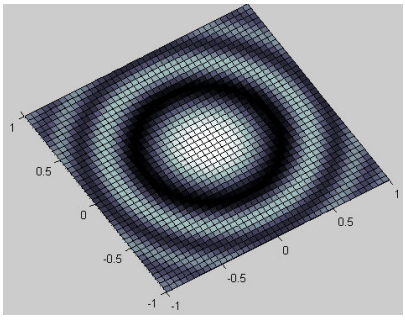
Two simple solutions to a trivial program can have a difference in speed of 1 to 550! What if you perform the loop a million times? **Let Matlab's implicit loops do the work whenever possible, i.e. use vectorization.** (Note: These time values depend on the characteristics of, and the current state of your computer.)

Non-Linear Optimization

- Must have access to optimization toolbox. The CSUGLAB should have it, your home version might not.
- Two functions of interest: **lsqcurvefit** and **lsqnonlin**.
- Non-linear optimization is as much an art as a science. One needs to adjust parameters (e.g. convergence criteria) to balance the time spent on a computation and the quality of the results.
- Global optimum (minimum) not guaranteed!

Local Minima Are a Big Problem





Optimizers at Work

```
% Generate a regular grid of values.
[x y] = meshgrid(-2:.1:2);

% Reshape the matrices
x = reshape(x, [prod(size(x)), 1]);
y = reshape(y, [prod(size(y)), 1]);

% Distance from origin in the XY plane
r = sqrt(x .* x + y .* y);

% Compute values.
vals = exp(-r) .* cos(10 * r);

% Add some random, normally distributed error.
vals = vals + .1 * randn(size(vals));

options = optimset('Display', 'iter');

% One could define lower and upper bounds for parameters.
lb = []; ub = [];

% lsqcurvefit(fun, x0, xdata, ydata)
[params, sse, res] = lsqcurvefit(@hat, [0, 1], [x y], vals, lb, ub, options);

params
```

```
function v = hat(params, xdata)
  x = xdata(:, 1);
  y = xdata(:, 2);
  r = sqrt(x .* x + y .* y);
  v = exp(-params(1)*r).*cos(params(2)*r);
```

Optimizers at Work(2)

```
                                        Norm of      First-order
 Iteration  Func-count     f(x)          step          optimality   CG-iterations
     1          4         539.204             1           634           0
     2          7         225.218       0.582754          108           1
     3         10         142.936       0.802229         21.7           1
     4         13         112.564        1.33095         5.47           1
     5         16         94.6193        3.77522         1.19           1
     6         19         87.3107             10       0.0685           1
     7         22         87.1573             10         0.13           1
     8         25         86.4093            2.5        0.143           1
     9         28         83.5992              5        0.555           1
    10         31         83.5992             10        0.555           1
    11         34         79.0935            2.5          1.4           0
    12         37         17.8228              5          7.8           1
    13         40         16.6546       0.135853        0.985           1
    14         43         16.6398      0.0137327       0.0182           1
    15         46         16.6398   0.000296995     0.000276           1
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun
params =
    0.99796815257395 -10.01438336970244
```

The true parameter values are 1 and 10!

Other Functions You Might Like

- **mkpp**, **ppval**: define and evaluate piecewise polynomials
- **datestr**, **datenum**: date conversions
- **cfdates**: cash flow dates for Treasuries (and other fixed-income instruments)
- **optimset**, **optimget**: set and retrieve the optimization options