



Fitting Smooth Functions with Spline Basis Functions

Daymond Ling
Senior Director, Modelling & Analytics
Customer Marketing
CIBC

Toronto Data Mining Forum Presentation
May, 2006

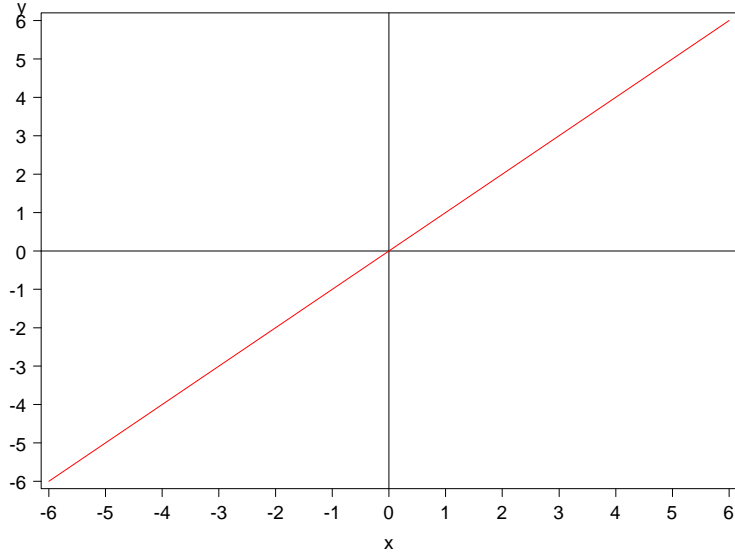
$$Y = \beta_0 + \beta_1 x_1 + \dots + \beta_n x_n$$

- ❑ X_1 to X_n are “basis functions” that are used to describe the arbitrary function Y
- ❑ Often when the relationship between Y and X is not a straight line, higher order terms or transformations are added to handle curvature

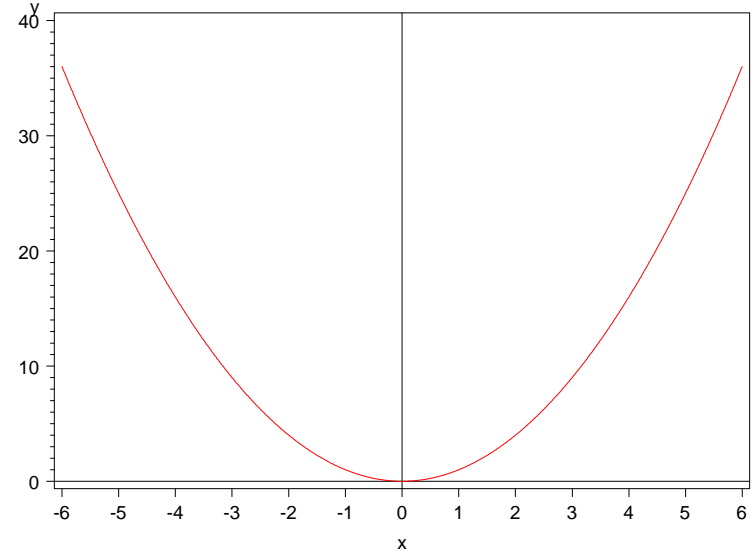


Some Typical Transformations

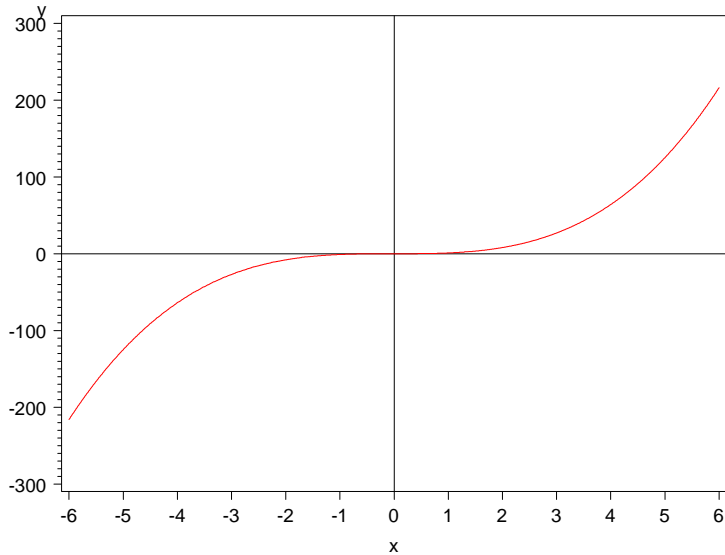
Linear Function



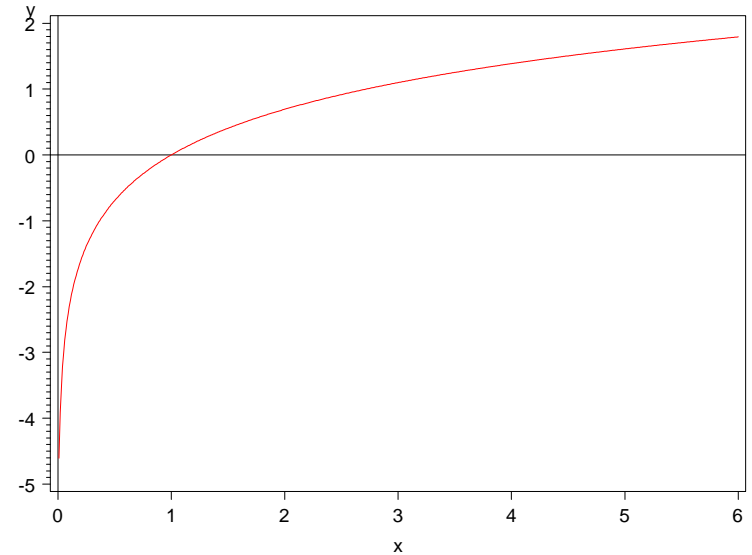
Quadratic Function



Cubic Function



Natural Log Function



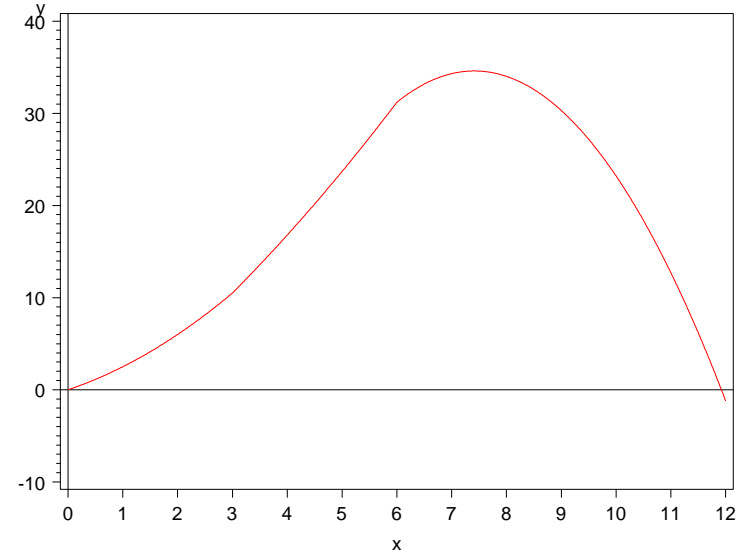


Arbitrary Functions are complex

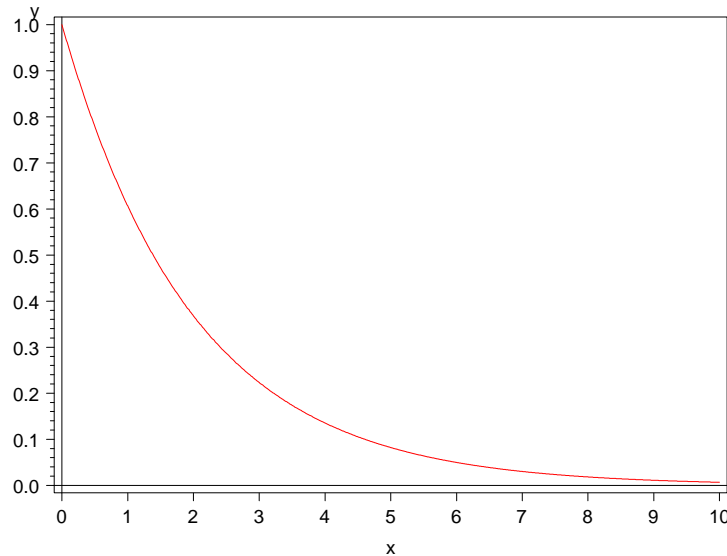
What about these functions?

X , X^2 , X^3 , $\ln(x)$ are, in reality, quite restrictive. They can't handle arbitrary functions well.

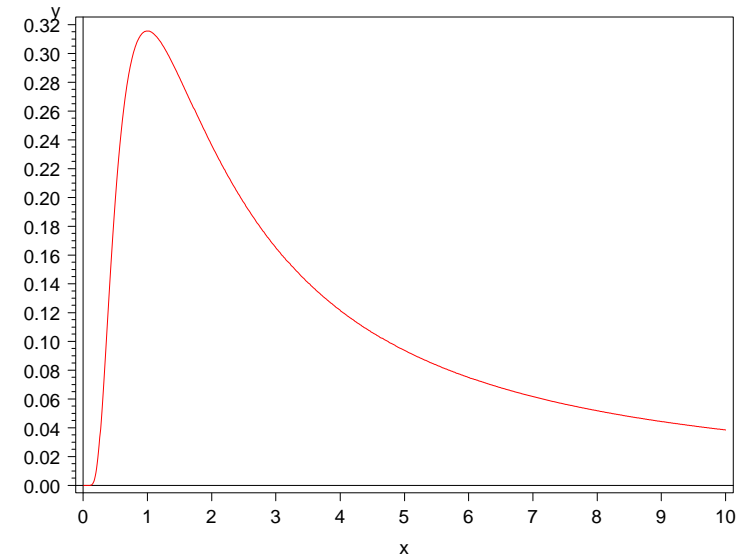
Arbitrary Distribution



Decay Function



Velocity Distribution

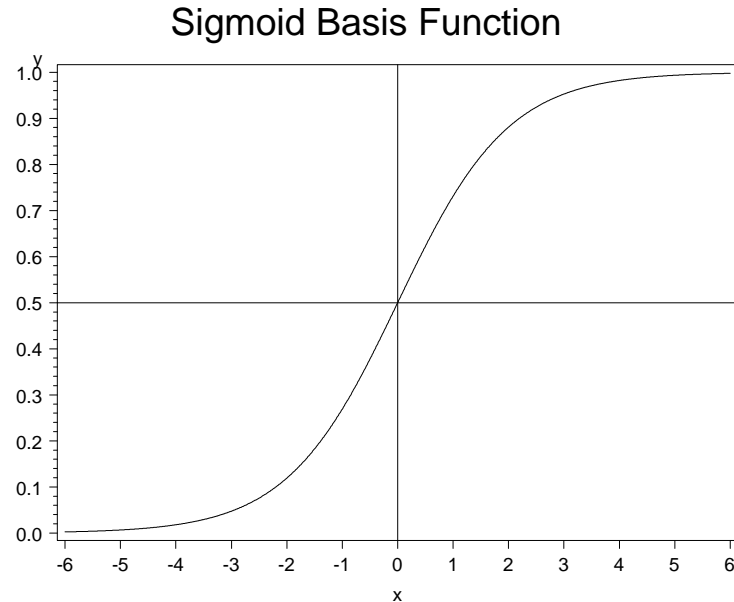


In general, we often encounter smooth curves that aren't well approximated over its entire range by a single smooth function, e.g., a polynomial.

One approach then is to use piece-wise weighted sum of multiple functions (basis) in the hope that it improves the approximation.

Sigmoid functions is a very nice smooth curve:

$$\frac{1}{1 + e^{-x}}$$



- ❑ Using sigmoid function as the basis is a special case of **Neural Network**
- ❑ To use Sigmoids as basis, we need to translate and scale both the Y and X axis
- ❑ We can think of it as

$$Y = \beta_0 + \beta_1 \left(\frac{1}{1 + e^{-(\beta_2 + \beta_3 x)}} \right)$$

where b_0, b_1 translate & scale the Y axis, and b_2, b_3 translate & scale the X axis

- ❑ In general, NN would be made up of the sum of a number of these basis. Each of these is a “node”. Each node contributes a single sigmoid curve. Hint: Number of low-high transitions in the data tells you how many sigmoids you need.
- ❑ The equation is nonlinear with respect to the parameters. There is no closed form analytic solution. It is solved through numerical optimization. It can be very computationally expensive, but we’ve got powerful computers.
- ❑ It’s important to note that NN software can automatically determine the translation & scaling required to achieve good fit. We don’t have to pre-specify them. We only need to specify the number of sigmoids to use.

- ❑ Cubic Spline is a set of piecewise third degree polynomial, joined at “knot” points. At the knot points the polynomials are continuous up to the second derivative. This makes the splines very smooth.

- ❑ **Cubic Splines:**
$$Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - k_1)_+^3 + \beta_5 (x - k_2)_+^3$$

where

$$(u)_+ = u \text{ if } u > 0, 0 \text{ otherwise}$$

Note: Beyond the range of data the function continues as cubic polynomial. If there are n knots, this requires n+4 parameters, including the intercept

- ❑ **Natural Splines** (a.k.a. Restricted Cubic Splines)

Same as Cubic splines but is linear beyond the boundary knots. Achieved by removing the quadratic and cubic terms

$$Y = \beta_0 + \beta_1 x + \beta_4 (x - k_1)_+^3 + \beta_5 (x - k_2)_+^3$$

where

$$(u)_+ = u \text{ if } u > 0, 0 \text{ otherwise}$$

Requires n+2 parameters, including the intercept

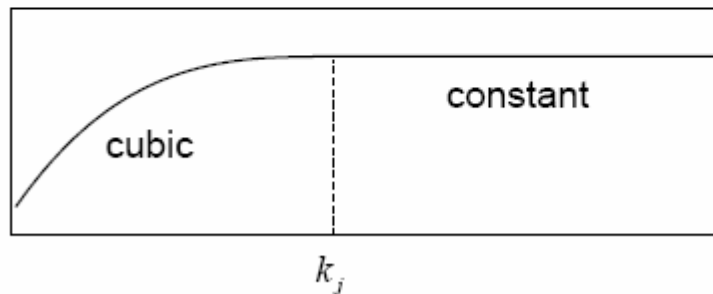
Spline Basis Functions

- Here's another spline that's useful for modeling time component of survival functions. It looks like the shape of the "radioactive decay" curve hence the name of "**Decay Spline**".

$$\psi(t, \mathbf{a}) = \alpha_{00} + \alpha_0 t + \sum_{j=1}^{\#\{\text{knots}\}} \alpha_j \text{csb}(t, k_j)$$

$$\text{csb}(t, k_j) = I\{t > k_j\} (t - k_j)^3 - t^3 + 3k_j t^2 - 3k_j^2 t$$

$$= \begin{cases} -t^3 + 3k_j t^2 - 3k_j^2 t & \text{if } t \leq k_j & \text{cubic} \\ -k_j^3 & \text{if } t > k_j & \text{constant} \end{cases}$$



continuous: $\text{csb}(t, k_j), \text{csb}'(t, k_j), \text{csb}''(t, k_j)$



Fitting Cubic Splines

- ❑ In general, we need to decide three things:
 - Shape of function
 - Number of basis functions
 - Knot points

- ❑ If we want to automatically find the translation & scaling that produces optimal fit, we need nonlinear optimization. In SAS, PROC MODEL can do this, but you need to know what you are doing. If you're not familiar with numerical optimization, you may not even know when mistakes happen nor how to rectify the situation.

- ❑ If you are willing to sacrifice one degree of freedom – pre-specify the knots – then the situation reduces down to normal regression which we are all intimately familiar with. So, our chore boils down to looking at the curve and deciding the type of curve, and choosing the knots.

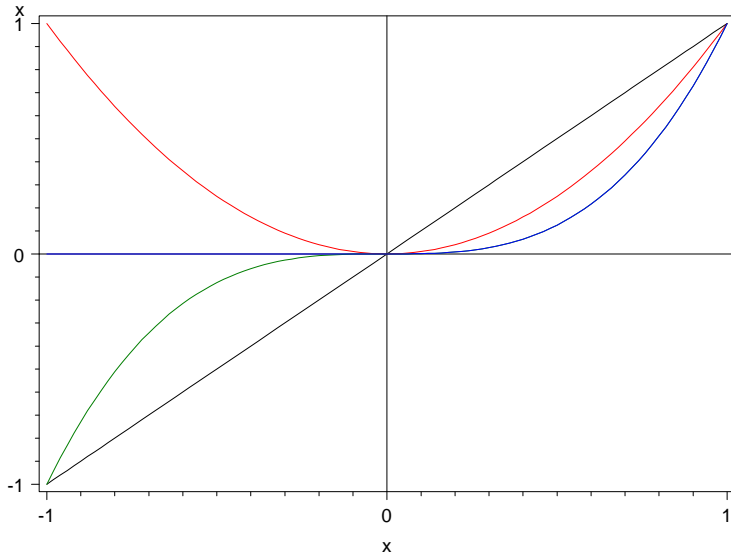
- ❑ Developing a feel for the shape of Splines help with choosing the knots

- ❑ Reminder: For NN, it's easy – number of "lo-hi" transitions tell you how many sigmoids or nodes (in the hidden layer) to use; software figures out where to put each sigmoid and how to scale it.

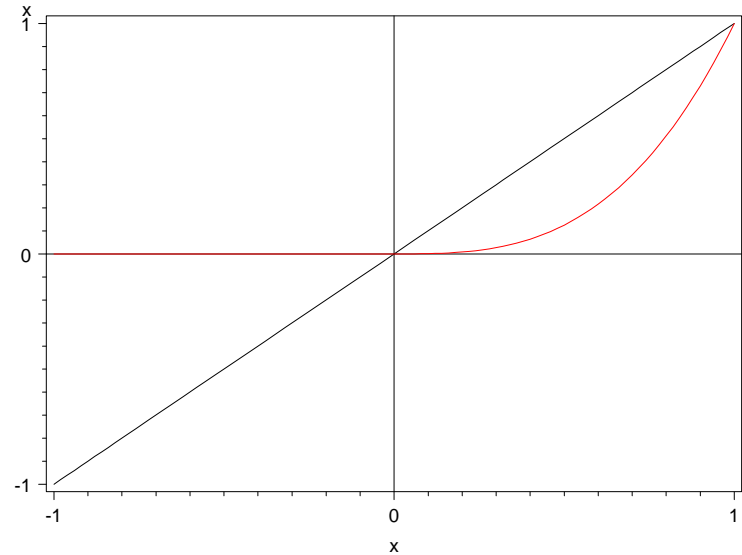


Cubic Spline Basis Functions

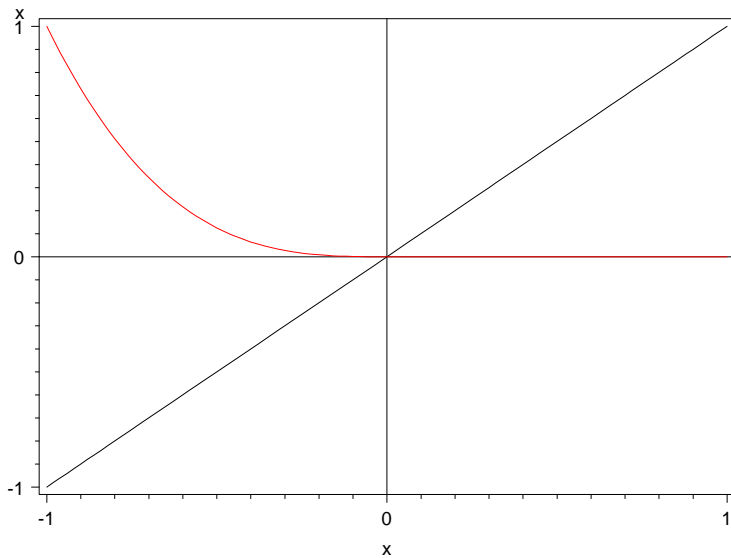
Cubic Spline



Natural Spline



Decay Spline

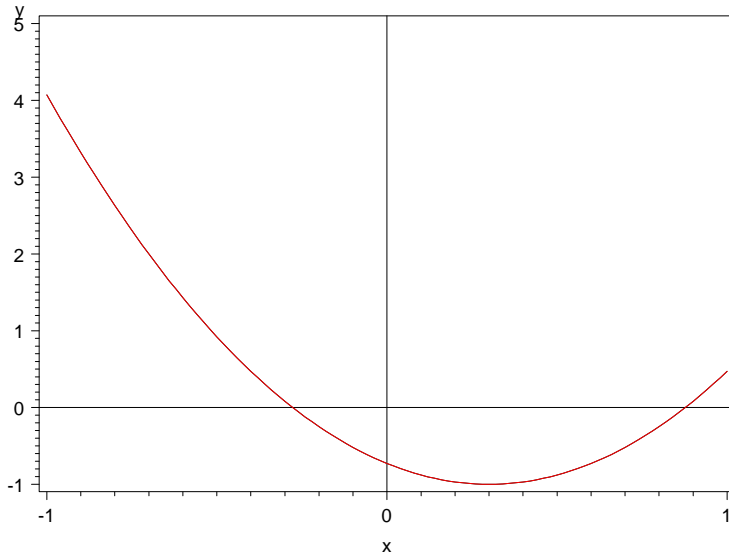


Basis Functions with knot at 0

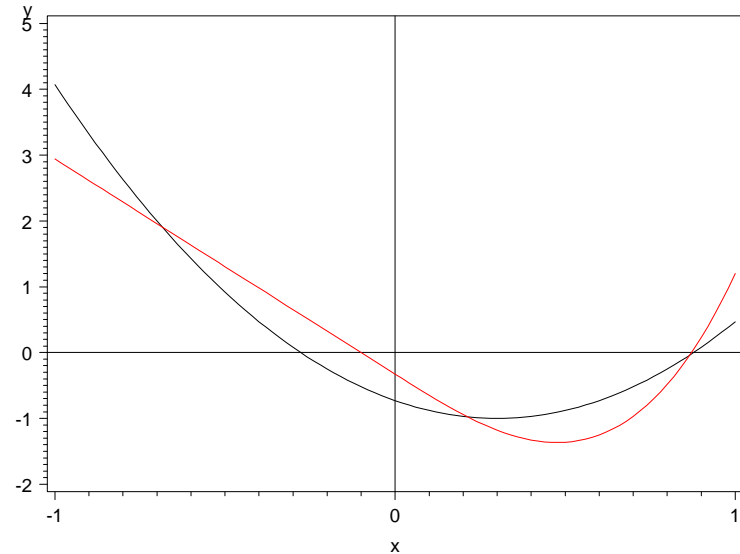


A pure parabola

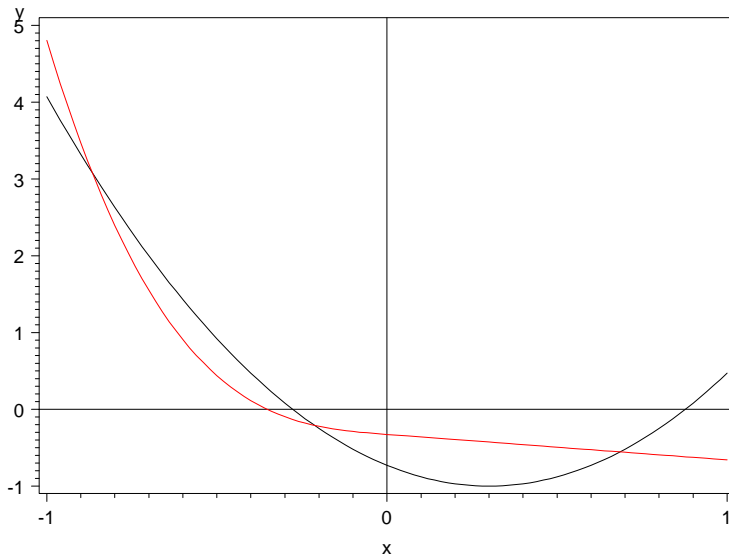
Cubic Spline knots=0



Natural Spline knots=0



Decay Spline knots=0



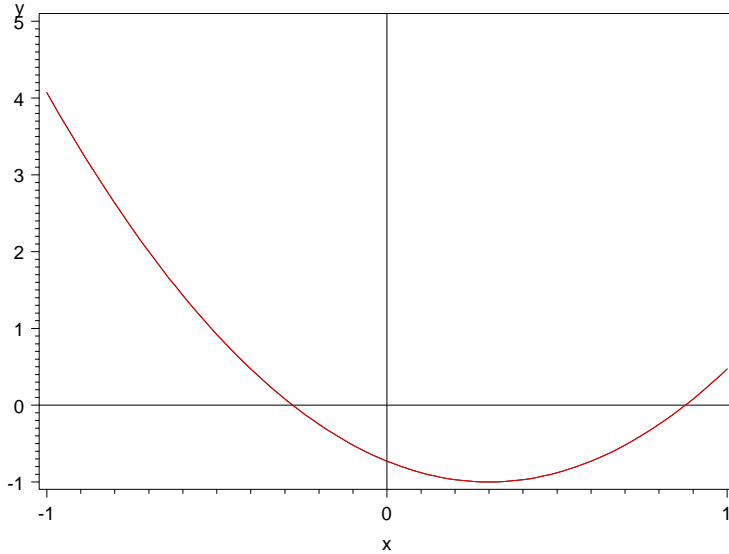
```
data source;  
  do x = -1 to 1 by 0.02;  
    y = 3 * (x - 0.3) **2 -1;  
    output;  
  end;  
run;
```

Cubic is perfect fit
Natural & Decay are off
Note Natural's linear extensions & its cusp is not @ 0

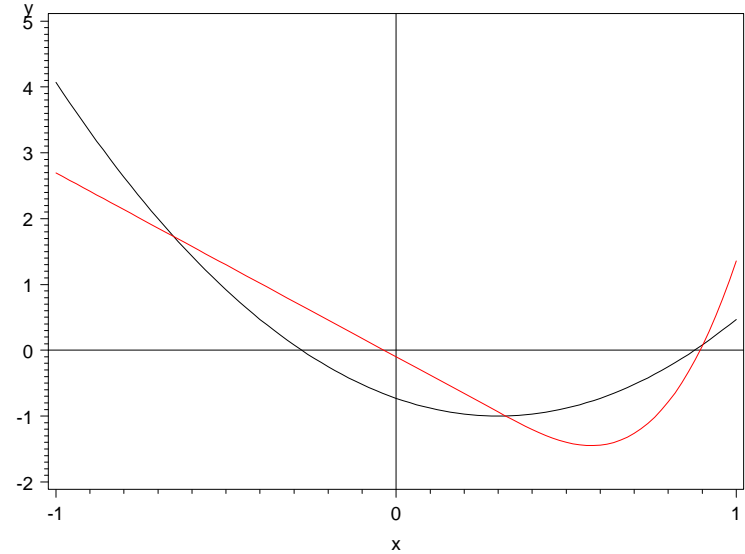


A pure parabola

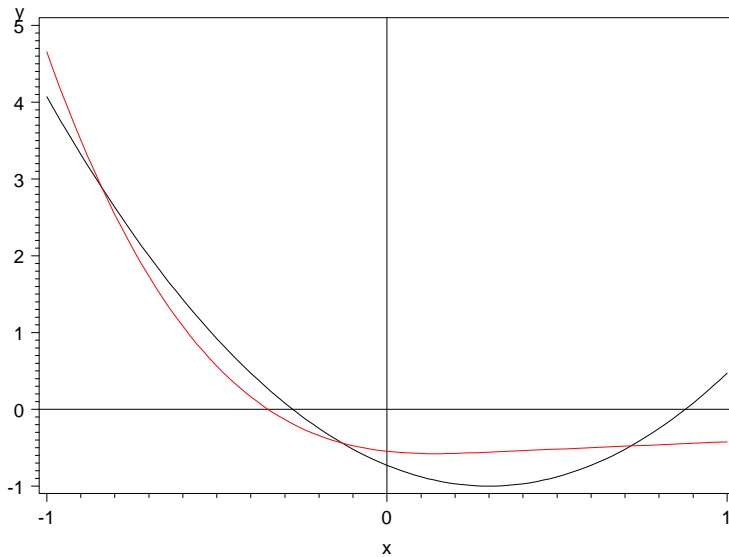
Cubic Spline knots=0.3



Natural Spline knots=0.3



Decay Spline knots=0.3

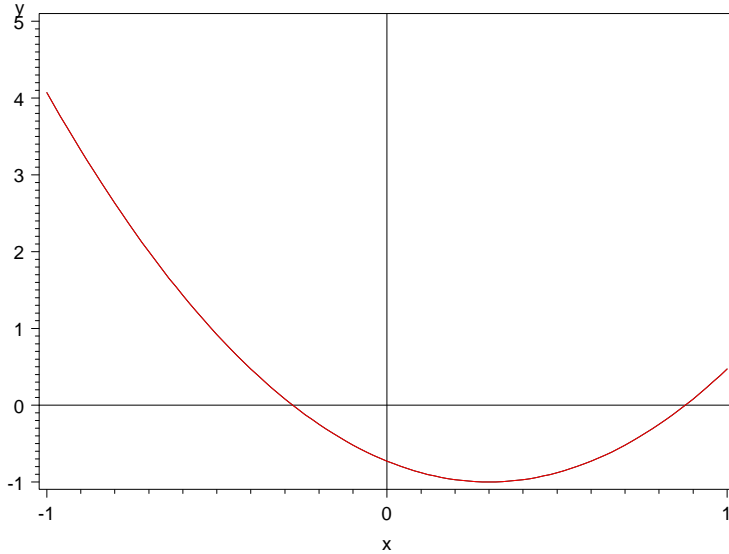


Cubic is perfect fit
Natural became worse
Decay became better

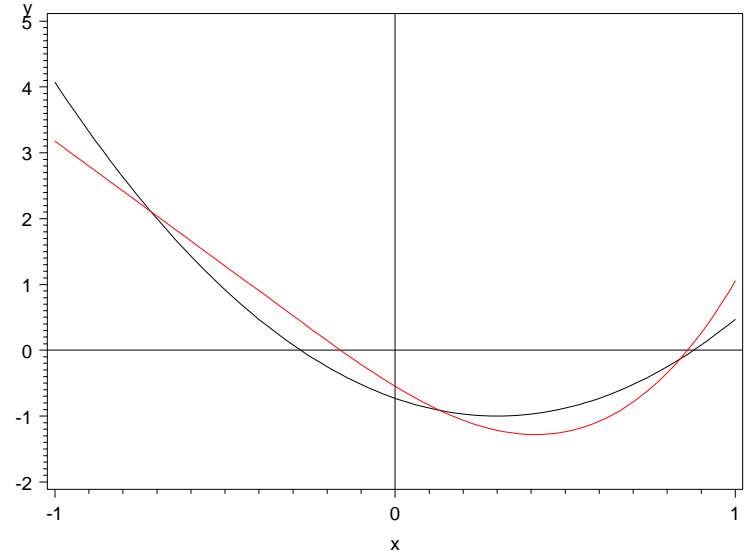


A pure parabola

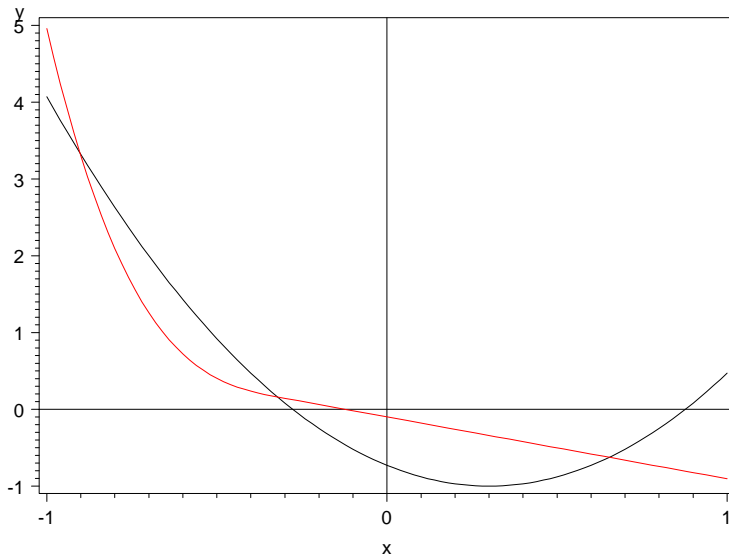
Cubic Spline knots=-0.3



Natural Spline knots=-0.3



Decay Spline knots=-0.3

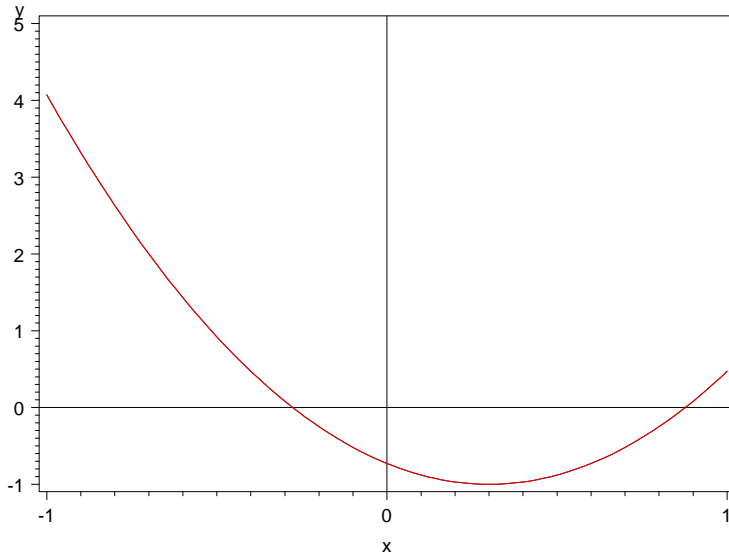


Cubic is perfect fit
Natural is closer
Decay became worse

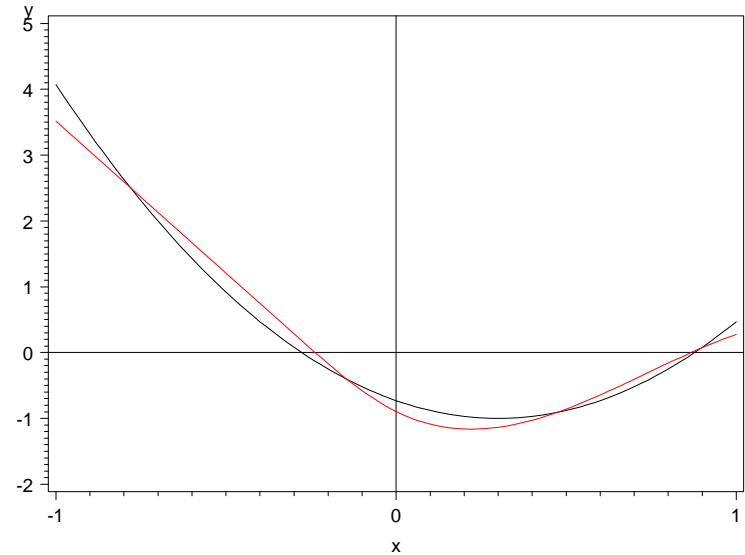


A pure parabola

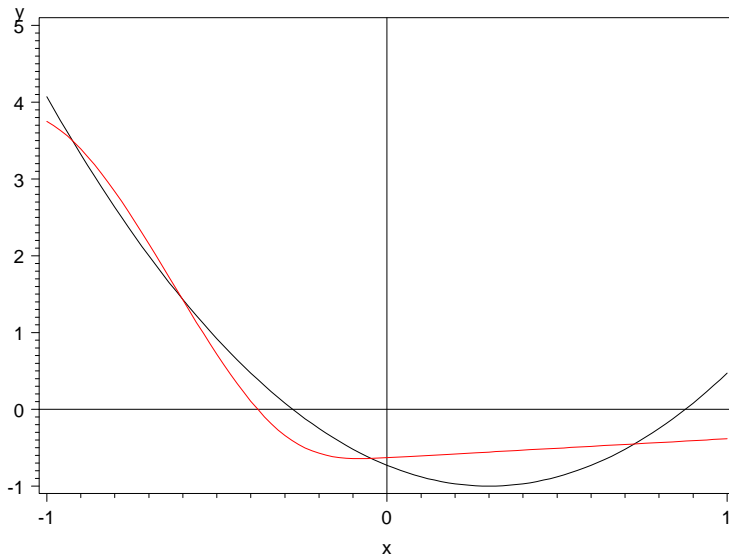
Cubic Spline knots=-0.3 0



Natural Spline knots=-0.3 0



Decay Spline knots=-0.3 0



Two knots now:

Cubic is perfect fit

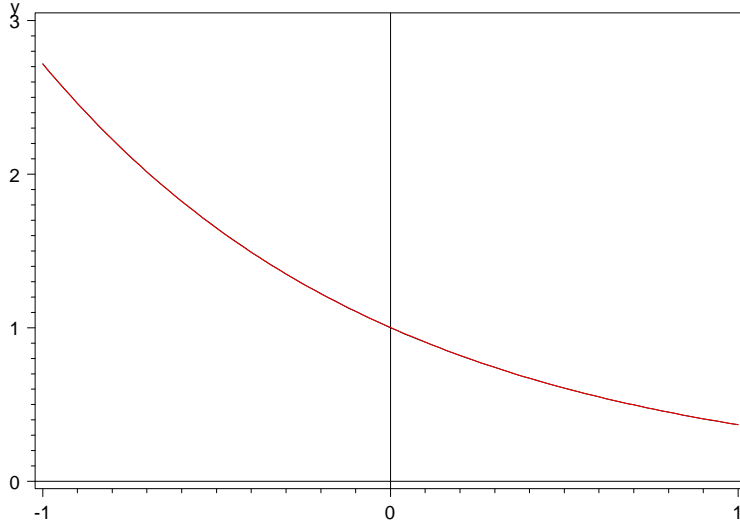
Natural is much closer now

Decay is still struggling

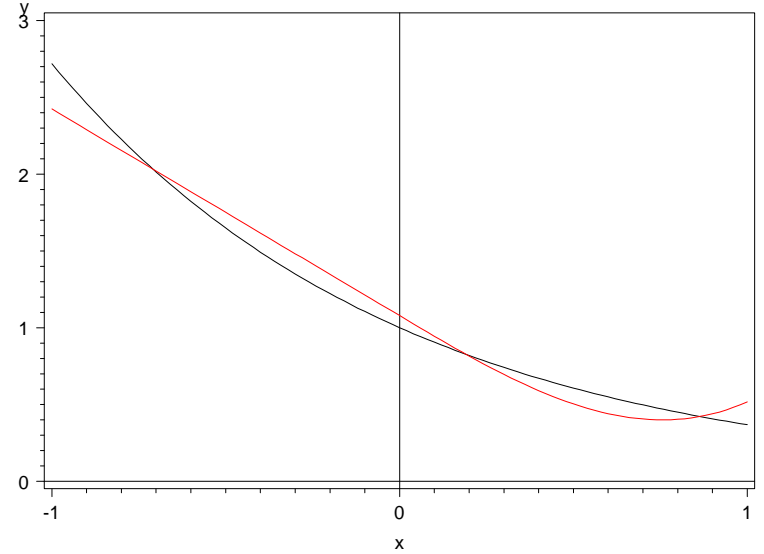


Exponential

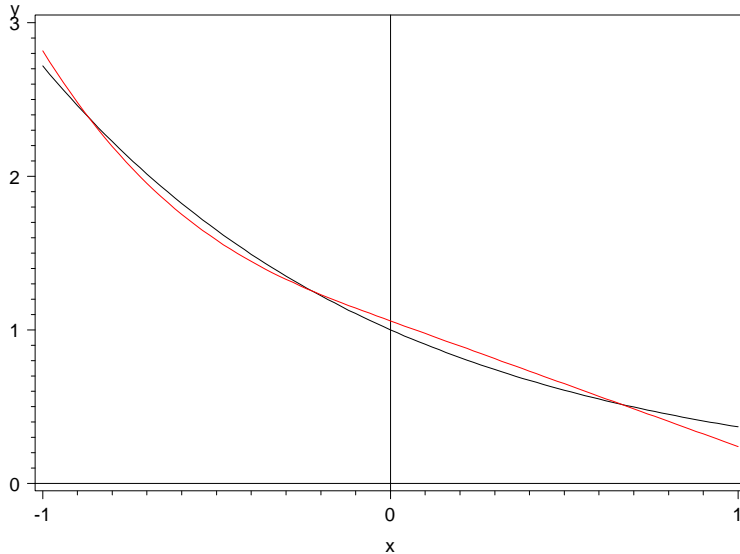
Cubic Spline knots=0



Natural Spline knots=0



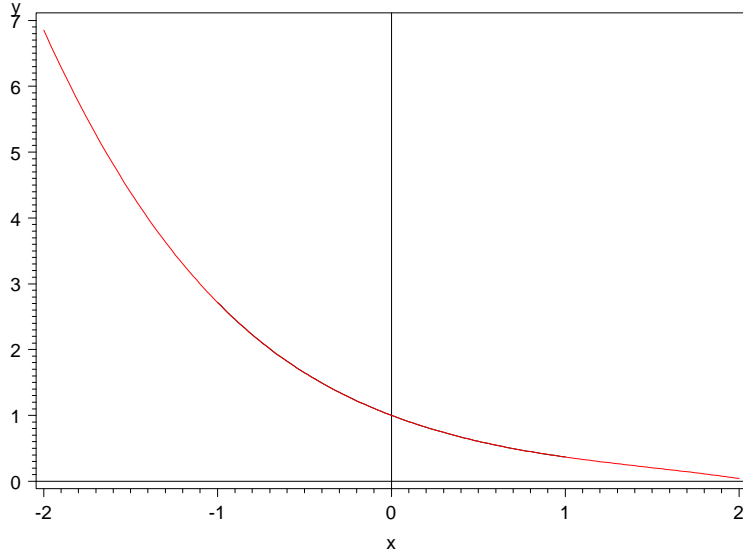
Decay Spline knots=0



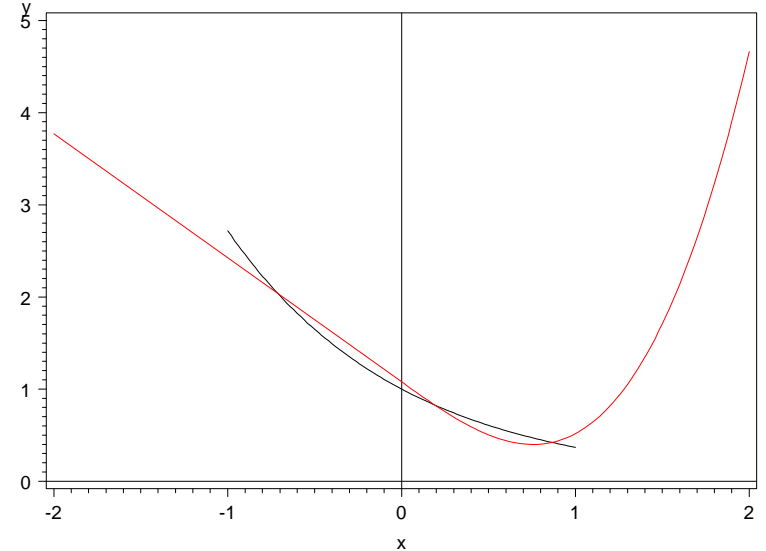


Let's extend the range...

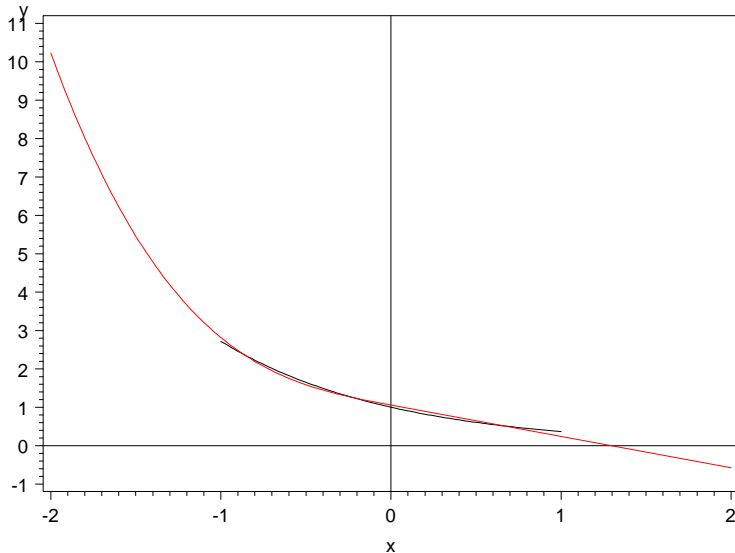
Cubic Spline knots=0



Natural Spline knots=0



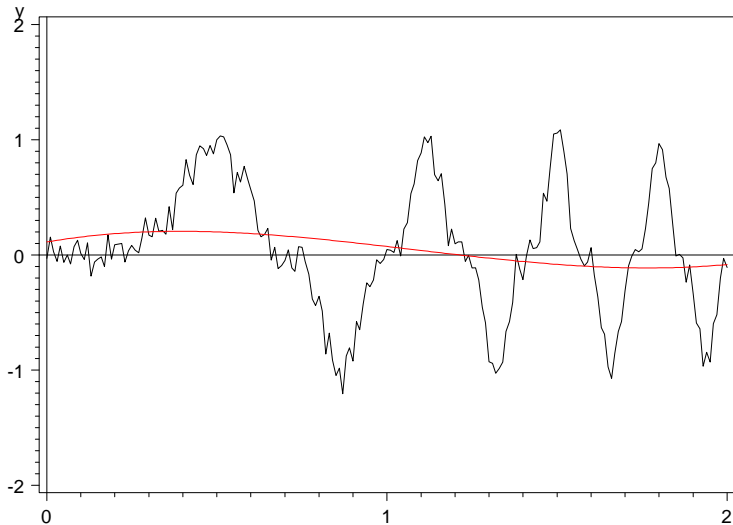
Decay Spline knots=0



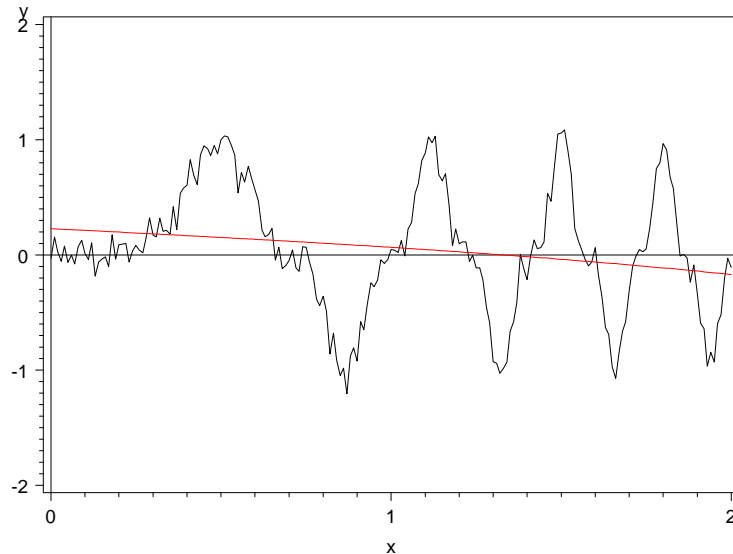
Cubic extends via cubic polynomial
Natural is linear beyond the knots eventually
Decay is linear beyond knot

A Nasty Function...

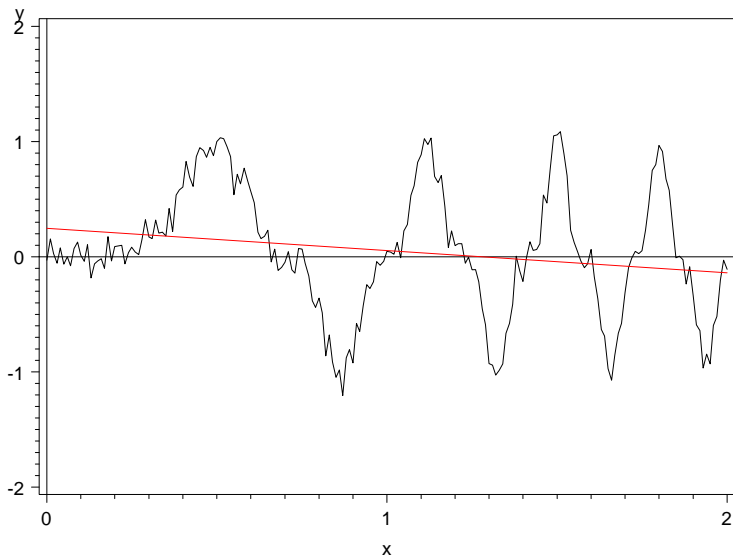
Cubic Spline knots=0



Natural Spline knots=0



Decay Spline knots=0



```

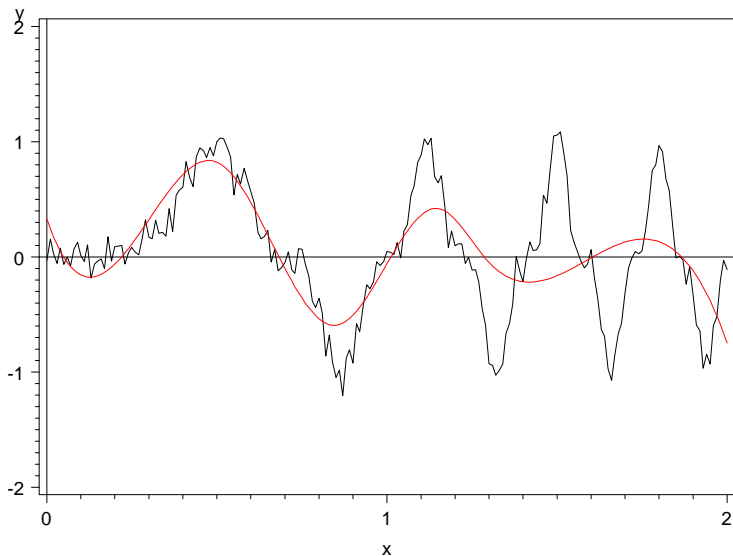
data source;
  do x = 0 to 2 by 0.01;
    ytrue = sin( 2 * 3.14159 * x**2 )**3;
    y = ytrue + normal( 123 ) * 0.1;
  output;
end;
run;

```

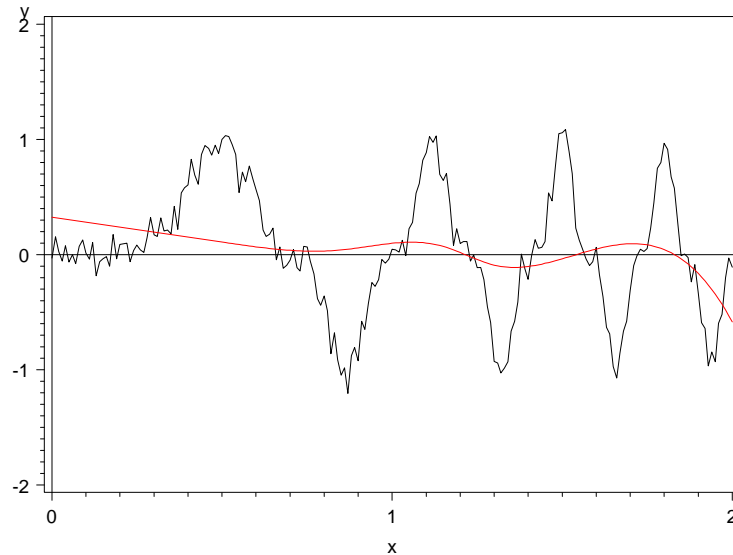


A Nasty Function...

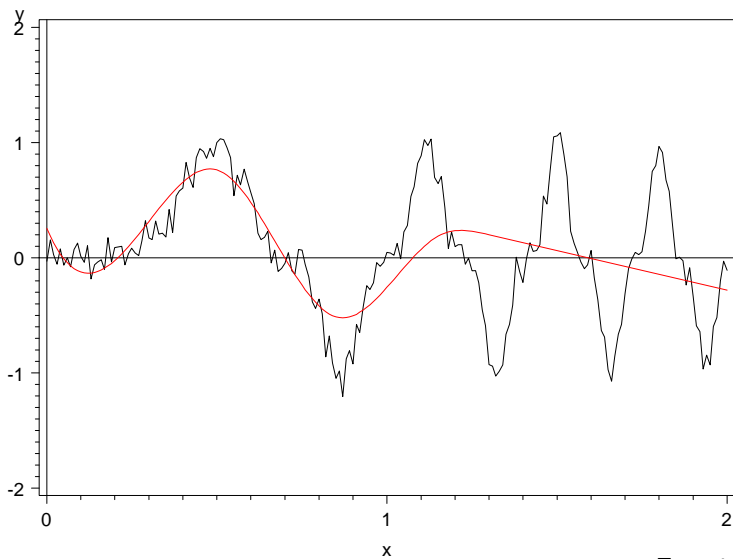
Cubic Spline knots=0.5 0.85 1.15 1.3



Natural Spline knots=0.5 0.85 1.15 1.3

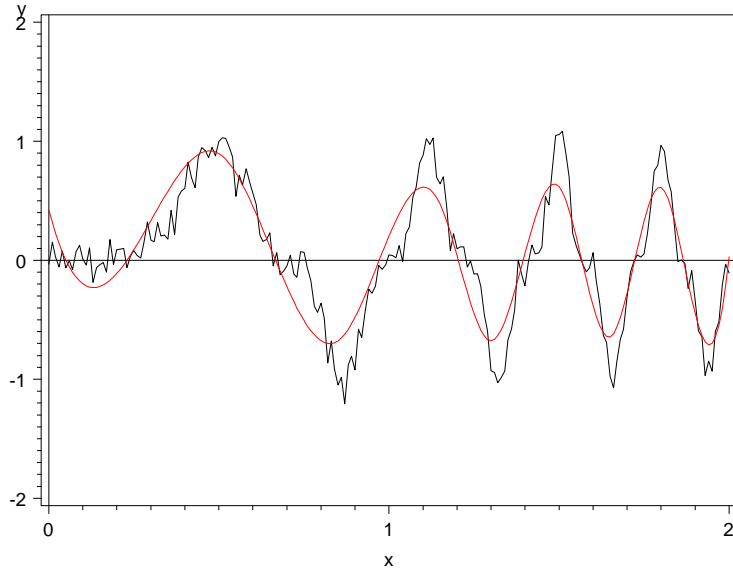


Decay Spline knots=0.5 0.85 1.15 1.3

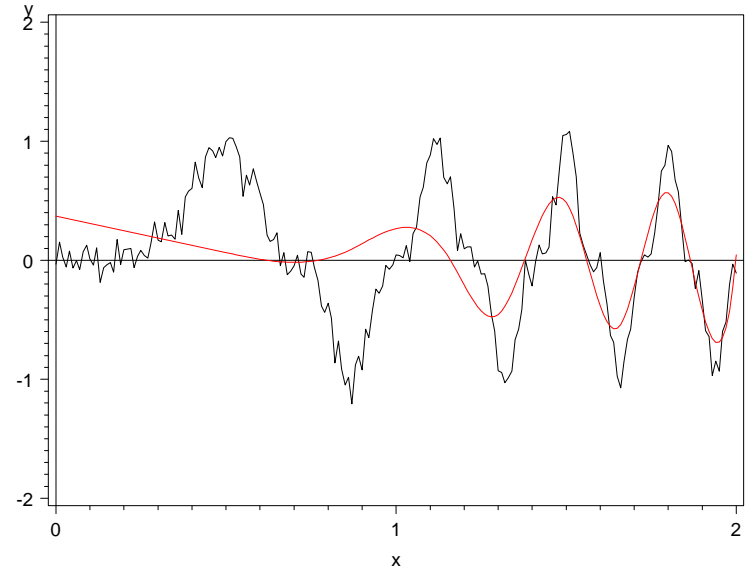


A Nasty Function...

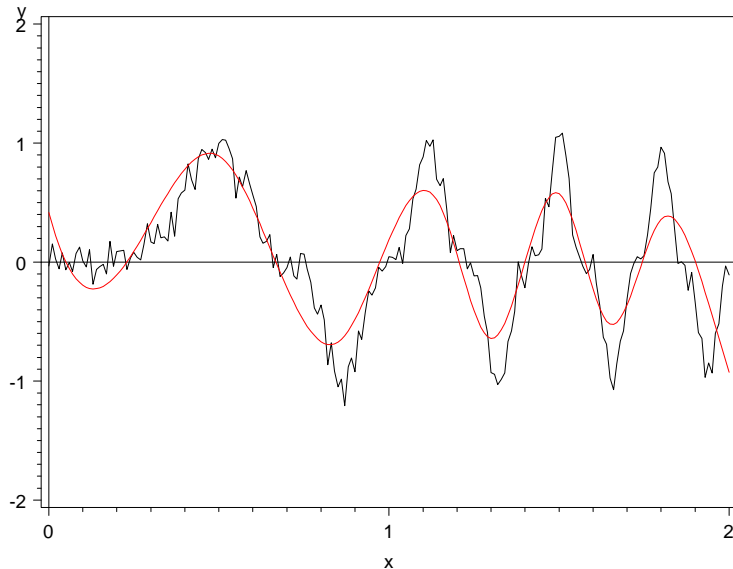
Cubic Spline knots=0.5 0.85 1.15 1.3 1.5 1.65 1.8 1.95



Natural Spline knots=0.5 0.85 1.15 1.3 1.5 1.65 1.8 1.95



Decay Spline knots=0.5 0.85 1.15 1.3 1.5 1.65 1.8 1.95



For the same knots, note

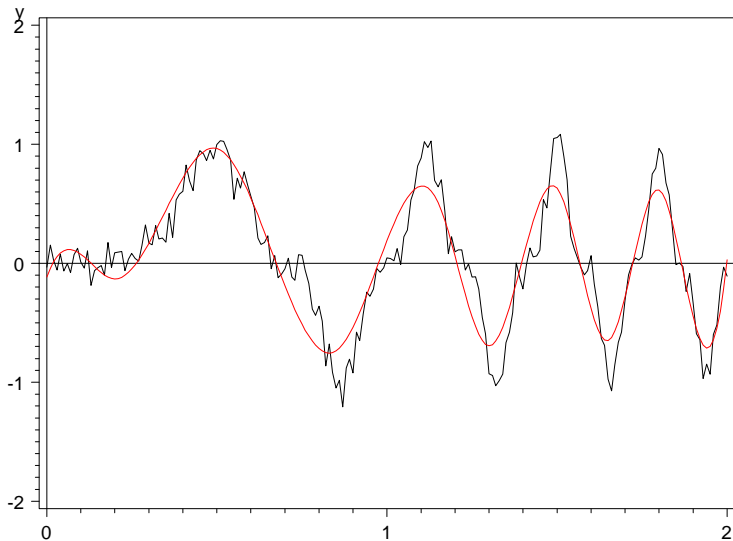
Cubic is "best" as it has the most degrees of freedom

Natural is disadvantaged relative to both as it's missing two terms relative to CS, and its basis functions are simpler relative to DS.

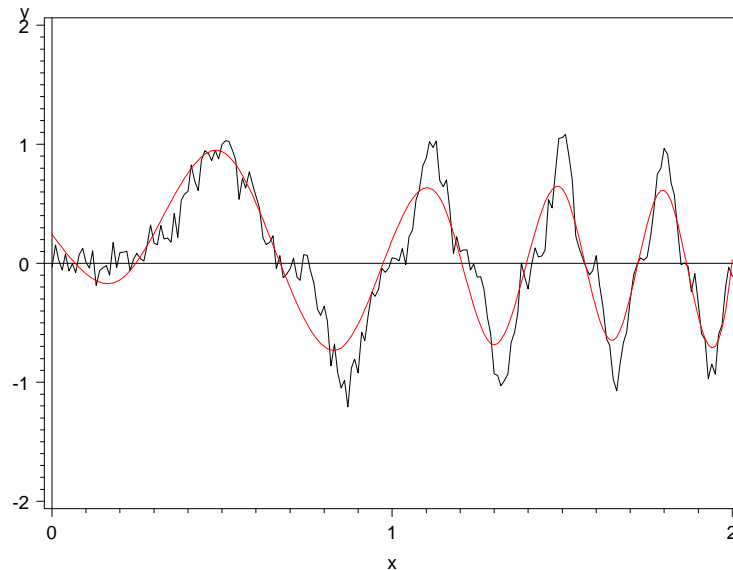


A Nasty Function...

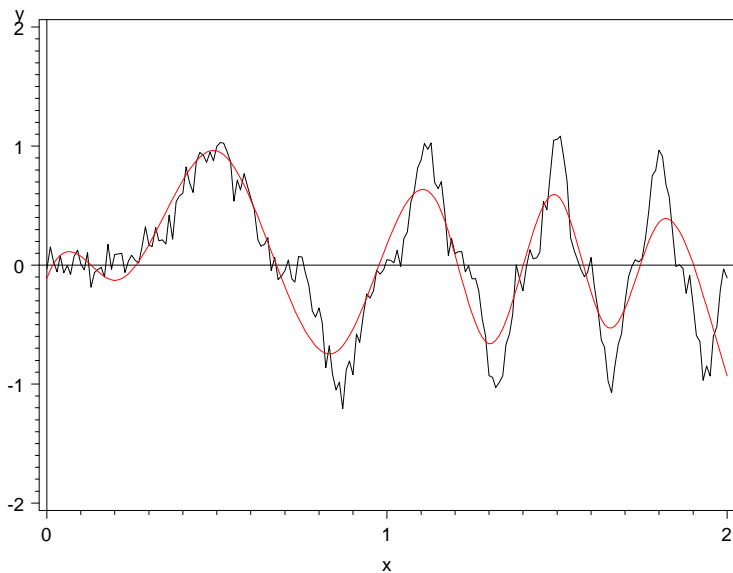
Cubic Spline knots=0 0.2 0.5 0.85 1.15 1.3 1.5 1.65 1.8 1.95



Natural Spline knots=0 0.2 0.5 0.85 1.15 1.3 1.5 1.65 1.8 1.95



Decay Spline knots=0 0.2 0.5 0.85 1.15 1.3 1.5 1.65 1.8 1.95

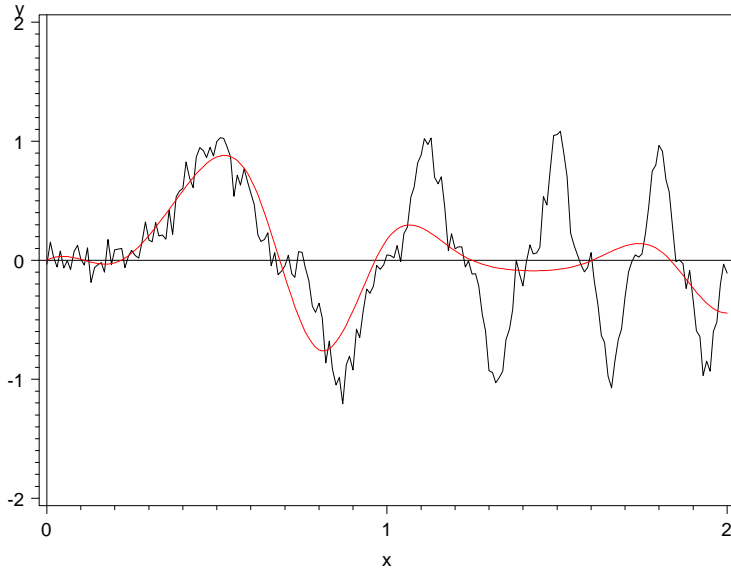


Once we give NS two more knots than the number of "transitions", it performs much better than before.

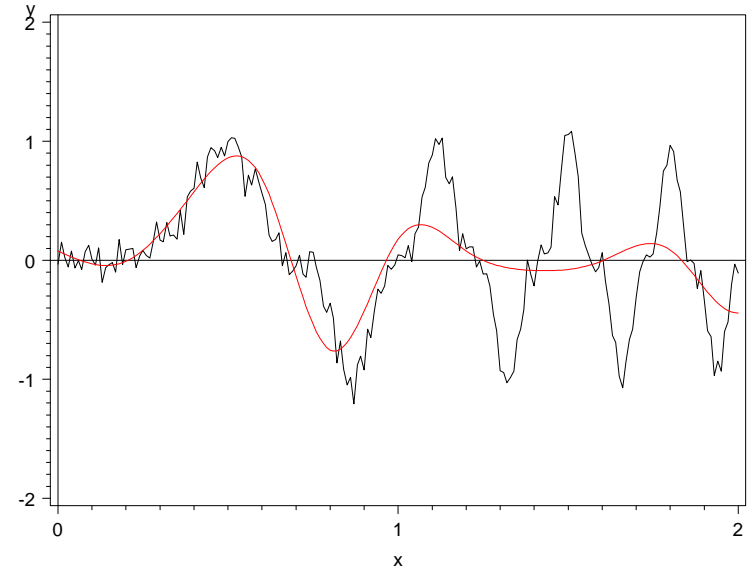
DS is having trouble with the last wiggle.

A Nasty Function...

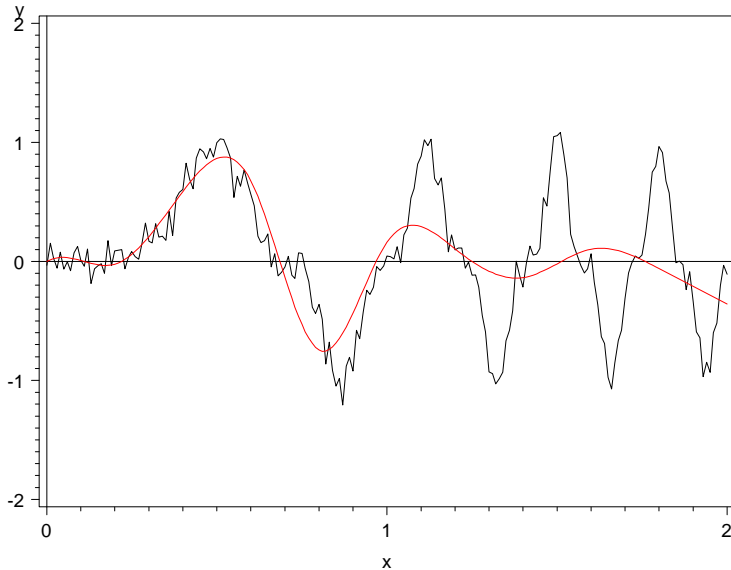
Cubic Spline knots=0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8



Natural Spline knots=0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8



Decay Spline knots=0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8



Knot placement can be important.

This example uses equally spaced knots and is much worse than before. Remember, this is a highly nonlinear function.



Thoughts...

- ❑ Before it has the right number of knots, it has lots of trouble.
 - CS wants at least the same # of knots as there are “transitions” to either up or down
 - NS wants two more knots than CS to achieve same degree of flexibility as CS
 - DS is good for trending & monotonic stuff. Highly nonlinear curve is not what it’s designed to do.

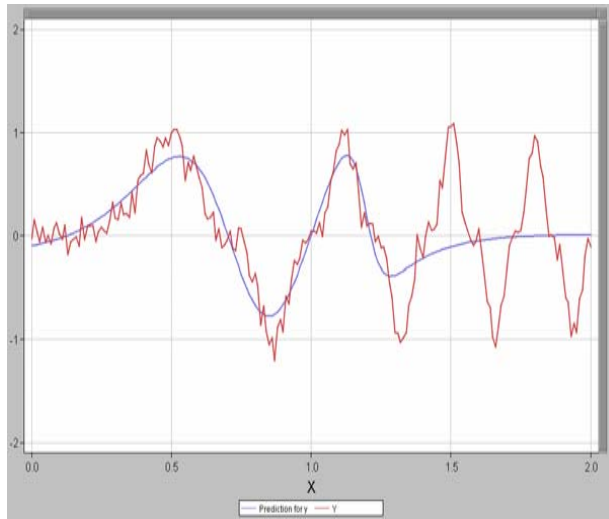
- ❑ Putting knots at the cusps seem to work well for CS

- ❑ NS wants a couple more knots at the left of the data range

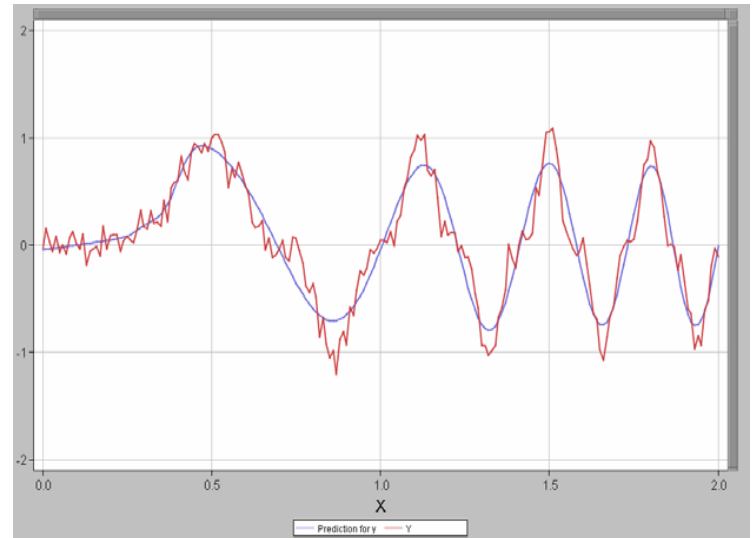


NN – using sigmoids

Hidden=3

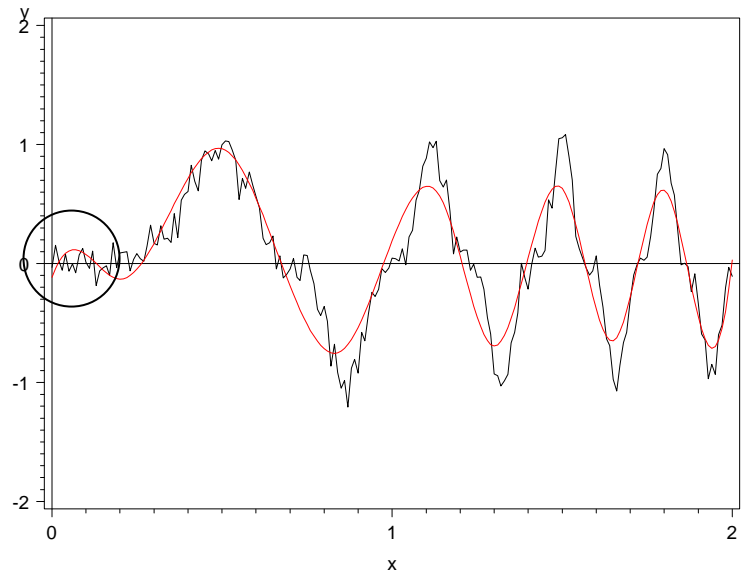
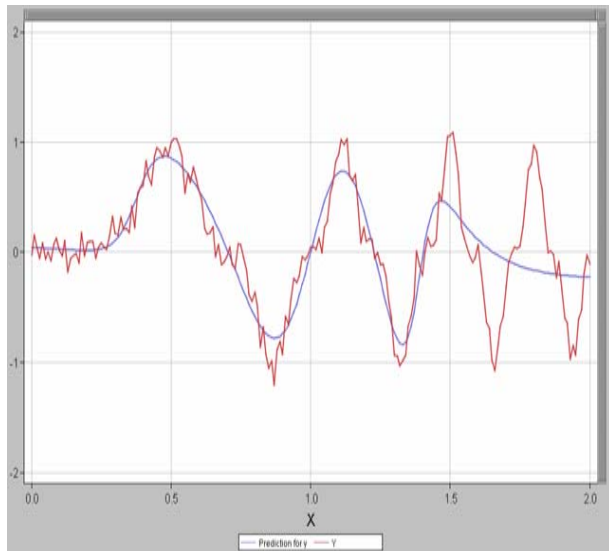


Hidden=8



Cubic Spline knots=0.2 0.2 0.5 0.85 1.15 1.3 1.5 1.65 1.8 1.95

Hidden=4





Spline vs. NN

	Cubic Spline	NN
Specify	<ul style="list-style-type: none"><input type="checkbox"/> Type of function<input type="checkbox"/> How many pieces<input type="checkbox"/> Each knot point	<ul style="list-style-type: none"><input type="checkbox"/> Type of NN<input type="checkbox"/> Number of hidden units
Estimation	Standard Linear Regression	Numerical Optimization
Diagnostics	<ul style="list-style-type: none"><input type="checkbox"/> SSE, parameters,...<input type="checkbox"/> Seldom have non-convergence issues	<ul style="list-style-type: none"><input type="checkbox"/> Tougher to read<input type="checkbox"/> Need to have mental image of NN in your head<input type="checkbox"/> Look for Convergence
Run Time	Short	Much longer than Regression
Accuracy	Better than using single functional form	Better than Cubic Splines
Interpretability	Worse than Linear Regression	Forget it
Extrapolation	<ul style="list-style-type: none"><input type="checkbox"/> CS is cubic at both ends<input type="checkbox"/> NS is linear at both ends<input type="checkbox"/> DS is cubic at left, linear at right	Flatten out