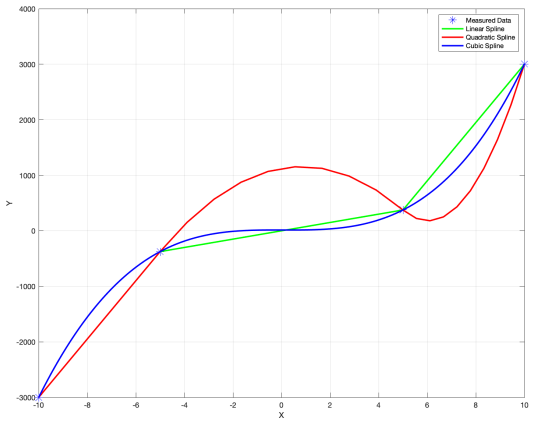

Splines and Polynomial Estimators

Jordan Winemiller

7/26/2021

Overview

There are many techniques that can be used in *Regression Analysis* for estimating relationships between dependent and independent variables. These relationships are normally used to find underlying trends that are used in a wide range of applications; such as making predictions, forecasting, and are widely used in machine learning. In this report the main focus will be on the topics of *Splines* and *Polynomial Estimators*; mainly the numerical methods will be discuss to view the advantages of splines over polynomial estimators.

Polynomial Estimators

Polynomial regression models allow the use of second (*squared*) and higher order terms of predictor values to be used, which makes the response function curvilinear. These models are special cases of the general linear regression model. In an model with one predictor variable the model has the following form:

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 X_i^2 + \varepsilon_i$$

When using higher order n^{th} polynomial estimators the typical problem of overfitting occurs, but there are use cases for polynomial estimators when the data does not have a linear trend. In following figure, the example data will be fit with four different polynomials with degrees of 1, 2, 3, and a n^{th} degree polynomial that travels through all the data points. This figure will display cases of polynomial estimators being better fit than a linear fit, and will also display the drawbacks of higher order polynomial estimators.

In the figure, the cubic model fits the data the best, and the polynomial fit has introduced a large drawback known as *polynomial wiggle*. This is caused due to the curve oscillating due to the amount of turns that occurred to fit the model; there are $n - 1$ turns that the polynomial function has to make. This is addressed by using splines.

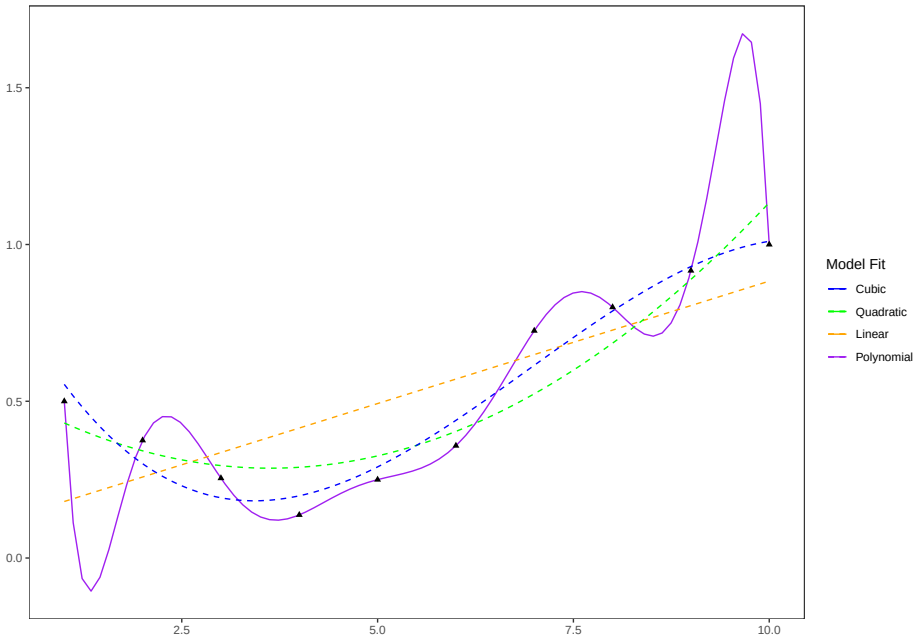

Figure 1: Polynomial Fits

Splines

In order to address overfitting and polynomial wiggle; cubic splines will be examined to see if there is way to overcome these drawbacks. Quadratic and higher order splines have smooth transitions between intervals, but splines unlike polynomial estimators break the global function into smaller sets of points.

Numerical Methods

A polynomial fit of n^{th} degree has the form:

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$$

Which has $n+1$ unknowns in the system, and for the data points $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$ there are $n+1$ constraints. Thus, we have a system of equations the is $(n+1) \times (n+1)$. By creating the system of equations or by Matrix form we have:

$$\begin{array}{l} (x_0\,,\,y_0) : y_0 = a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n \\ (x_1\,,\,y_1) : y_1 = a_1 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n \\ \qquad\qquad \vdots \\ (x_n\,,\,y_n) : y_n = a_n + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n \end{array} \quad or \quad \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^m \\ 1 & x_2 & x_2^2 & \cdots & x_2^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^m \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

For the system of equations this becomes $Ax = b$, which in MATLAB the command to solve this system is $x = a\backslash b$. The Matrix solved by the form $\overline{y} = X\overline{\beta} + \overline{\varepsilon}$. The downside to this form is as more data is collected the system of equations has to be updated, which can be cumbersome.

There is a more efficient way of writing the system of equation using Lagrange polynomial coefficients.

Lagrange Polynomial Coefficients

The simple example of a line fit using the data points $(x_0, y_0), (x_1, y_1)$, then for the equation $p_1(x) = y_0 \frac{x-x_1}{x_0-x_1} + y_1 \frac{x-x_0}{x_1-x_0}$, when $x = x_0 \Rightarrow y_0 = y_0 \times 1 + y_1 \times 0 = y_0$, and when $x = x_1 \Rightarrow y_1 = y_0 \times 0 + y_1 \times 1 = y_1$. The general form of the Lagrange coefficients for the data points $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$ is:

$$\sum_{k=0}^{n} y_k L_{1,k}(x)$$

From the previous example of the first two data points, this would be $\sum_{k=0}^{1} y_k L_{1,k}(x) = y_0 L_{1,0}(x) + y_1 L_{1,1}(x)$; where $L_{1,0} = \frac{x-x_1}{x_0-x_1}$, and $L_{1,0} = \frac{x-x_0}{x_1-x_0}$. Now, the Lagrange polynomial coefficients are indicators functions that are only have the value of 1 at their correlated data point. This is the method used in most current software solutions.

Cubic Splines

Cubic splines have an advantage over a polynomial fit, because the fit is local between each interval of the data. The spline still accounts for the overall fit of the data. The downside is cubic splines have more unknowns and constraints than the polynomial fit model. The form of a cubic spline is:

$$S_k(x) = S_{k,0} + S_{k,1}(x - x_k) + S_{k,2}(x - x_k)^2 + S_{k,3}(x - x_k)^3;\ x \in [x_k, x_{k+1}]$$

The cubic spline has four unknowns for the interval $x \in [x_k, x_{k+1}]$. Four constraints will be need to make our system of equations.

Cubic Spline Interval

Since there are four unknowns per cubic spline, four constraints will have to be needed:

1. The spline has to go through the data such that $S_k(x_k) = y_k$.
2. Has to match the next spline at the boundaries such that $S_k(x_{k+1}) = S_{k+1}(x_{k_1})$.
3. The first derivatives $S'_k(x_{k+1}) = S'_{k+1}(x_{k_1})$ have to match in he interval.
4. The same follows for the second Derivatives $S''_k(x_{k+1}) = S''_{k+1}(x_{k_1})$.

Now the cubic spline has all four of the constraints.

Cubic Spline System

For the data $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$, cubic splines have n intervals and $n + 1$ constraints. This initially leads to a problem since each intervals has four unknowns, and the first and last data points do not have a point on the outside them the last three cubic criteria can not be fulfilled. So, the system will have $4n$ unknowns, because the cubic spline has to travel through each point.

The first criteria will have $n + 1$ constraints, but the last three criteria will have $n - 1$ constraints. The system will have $4n$ unknowns, and $4n - 2$ constraints; which is an underdetermined system.
Then the user has to enforce two more constraints, and typically the first and last points have their second derivatives set to zero. Which makes the system consistent, and the system of equations is now $Ax = b$.

Method Comparison

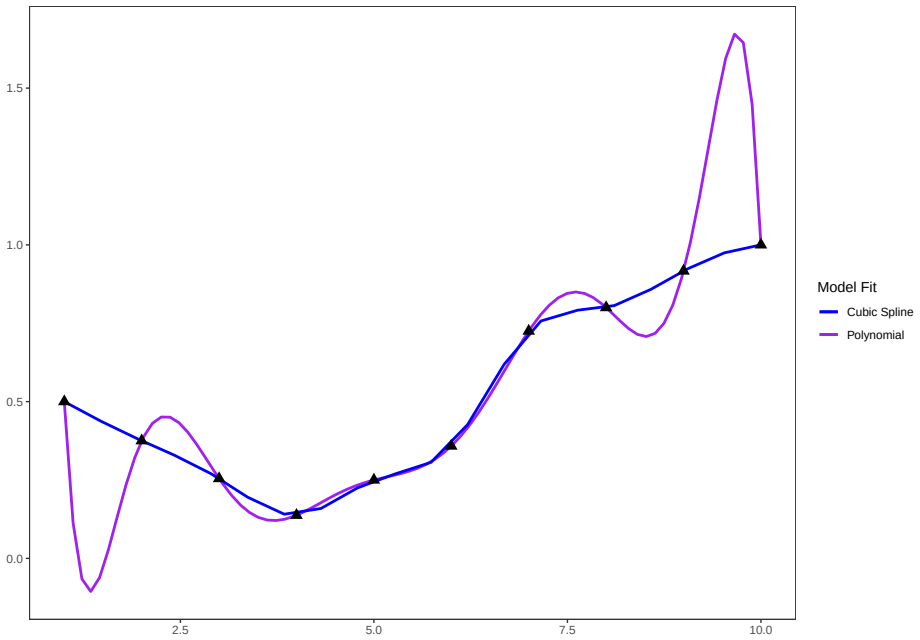

Figure 2: Cubic Splines vs Polynomial Fits

The advantage of using splines is for data interpolation, the user can now make predictions using local fit estimators opposed to an overall fit. The downside to this process of cubic splines is extrapolation cannot be used since the splines are only have information on the interior of the dataset.

Comparison Graph Code

```
ggplot2::ggplot(df, aes(x = x, y = y)) +
  geom_smooth(
    method = "lm",
    formula = y ~ poly(x, (length(df$x) - 1)),
    se = FALSE,
    aes(color = 'purple'),
    size = 1) +
  geom_line(
    data = data.frame(spline(df, n = length(df) *10, method = 'fmm')),
    aes(color = 'blue'),
    size = 1) +
  geom_point(shape = 17, size = 3) +
  theme_bw() +
  theme(
    panel.background = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank()) +
  xlab("") +
  ylab("") +
  scale_color_identity(
    name = "Model Fit",
    breaks = c("blue",  "purple"),
    labels = c("Cubic Spline", "Polynomial"),
    guide = "legend"
  )
```

MATLAB Implementation of Linear and Quadratic Splines

In the case of a quadratic splines; the spline must go through three data points with the constraints:

1. Traveling through the point.
2. The splines in the interval match at the boundary
3. The first derivatives also match at the point

And for linear splines; the spline just connects the points.

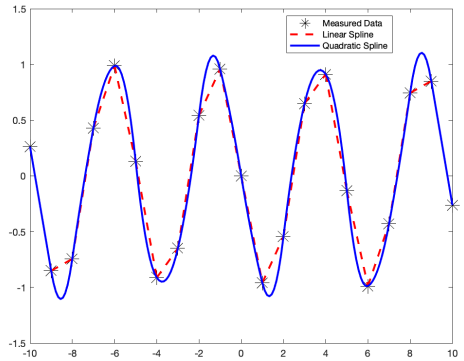

Figure 3: Linear and Quadratic Splines

MATLAB Code

Linear Splines

```
x = transpose(-10:10);
y = sin(5 * x);
p_0 = plot(x, y, 'k*', 'MarkerSize', 14);
hold on
% Linear Splines
for idx = 1:length(x) - 1
    m = (y(idx + 1) - y(idx)) / (x(idx + 1) - x(idx));
    x_spline = linspace(x(idx), x(idx + 1), 10);
    y_spline = m * (x_spline - x(idx)) + y(idx);
    p_1 = plot(x_spline, y_spline, '--r', 'LineWidth', 2);
end
```

Quadratic Splines

```
% Quadratic Splines
n = length(x) - 1;
% Constraint; function of interior points must match
% Create h_0 = [2 * (n - 1) x 3 * n]
h_0 = zeros(2 * n-2, 3 * n);
k_0 = zeros(2 * n-2, 1);
for idx = 1:(n - 1)
    col = idx;
    row = 2 * (idx - 1) + 1;
    h_0(row, col) = x(idx + 1)^2;
    h_0(row + 1, col + 1) = x(idx + 1)^2;
    h_0(row, n + col) = x(idx + 1);
    h_0(row + 1, n + col + 1) = x(idx + 1);
    h_0(row, 2 * n + col) = 1;
    h_0(row + 1, 2 * n + col + 1) = 1;
    k_0(row) = y(idx + 1);
    k_0(row + 1) = y(idx + 1);
end
```

```
% Constraint; first derivative of interior points must match
% The endpoints have an unknown first derivative, so they are set equal to
% each other addressed later
% Create h_1 = [n - 1 x 3 * n];
h_1 = zeros(n - 1, 3 * n);
for idx = 1:n - 1
    h_1(idx, idx) = 2 * x(idx + 1);
    h_1(idx, idx + 1) = -2 * x(idx + 1);
    h_1(idx, n + idx) = 1;
    h_1(idx, n + idx + 1) = -1;
end
% Constraint; the spline must pass through the first and last points
% Create h_e
h_e = zeros(2, 3 * n);
h_e(1, 1) = x(1)^2;
h_e(1, n + 1) = x(1);
h_e(1, 2 * n + 1) = 1;
h_e(2, n) = x(end)^2;
h_e(2, 2 * n) = x(end);
h_e(2, end) = 1;
% Gather the constraints, and address the matrix is not square by one
% constraint
% Compute h
h = [h_0; h_1; h_e];
[r, c] = size(h);
```

```
% This is the square matrix by matching first point to the last point
% Compute k
k = zeros(r, 1);
k(1:(2 * (n - 1))) = k_0;
k(end - 1) = y(1);
k(end) = y(end);
% This makes the first spline linear
% Assume now that a_1 = 0
h = h(:, 2:end);
coeffs = inv(h) * k;
% Get all your Constraints
A = [0;coeffs(1:n - 1)];
B = coeffs(n:2 * n - 1);
C = coeffs(2 * n:end);
% Plot the results
for idx = 1:n
    % Same as linear splines
    x_spline = linspace(x(idx), x(idx + 1), 10);
    % Quadratic Spline Equation
    y_spline = A(idx) * x_spline.^2 + B(idx) * x_spline + C(idx);
    p_2 = plot(x_spline, y_spline, 'b-', 'LineWidth', 2);
end
legend([p_0 p_1 p_2], 'Measured Data', 'Linear Spline', 'Quadratic Spline')
```

Discussion

The methodology of splines has the advantage of a piecewise construction where the spline is continuous between splines. So, the cubic splines match at each point in the interval, and first and second derivative match at each point in the interval. Where as the polynomial fit goes through all the points in the data in a continuous function, which leads to overfitting and polynomial wiggle.

In the Code files the quadratic_spline_animation.m was suppose to create an animation of the linear splines and then over lay the quadratic splines piecewise, but the videoWriter was not working in MATLAB.

Reference

Andre S. Douzette. 2017. "B-Splines in Machine Learning."

Kutner, Michael H., Christopher J. Nachtsheim, and John Neter. 2004. "Applied Linear Regression Models." McGraw-Hill Irwin.

Monte Carlos. 2018. "MATLAB Help - Cubic Splines." https://www.youtube.com/watch?v=gfSKOfHRooQ&t=107s.

Nathan Kutz. 2016. "Data Fitting: Matlab Implementation." https://www.youtube.com/watch?v=ItjSCEf-DdY.