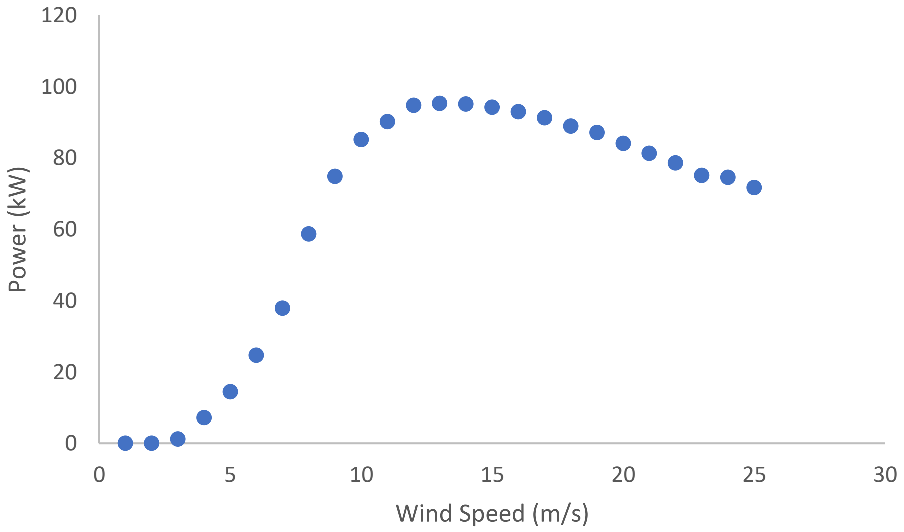
Lecture 4

Polynomial Regression & Splines

Motivating Example

I want to predict power output from a wind turbine based on wind speed.

There is an obvious relationship, but it's not linear, so a linear model won't work (obviously)



Polynomial Regression

$f(x) = x^n$

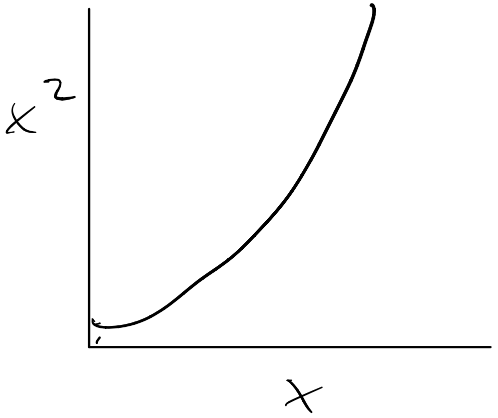
- Maybe I can fit it to some polynomial function of wind speed?

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

X = np.concatenate([[wind_speed], [wind_speed**2], axis=0).T

lr = LinearRegression()
lr.fit(X, power)
y_pred = lr.predict(X)

# import statsmodels.api as sm
# model = sm.OLS(power, X).fit()
# y_pred = model.predict(X)

print(mean_squared_error(power, y_pred))
>>>81.52
```

$$\begin{bmatrix} x_1, & x_1^2 \\ x_2, & x_2^2 \\ x_3, & x_3^2 \end{bmatrix}$$

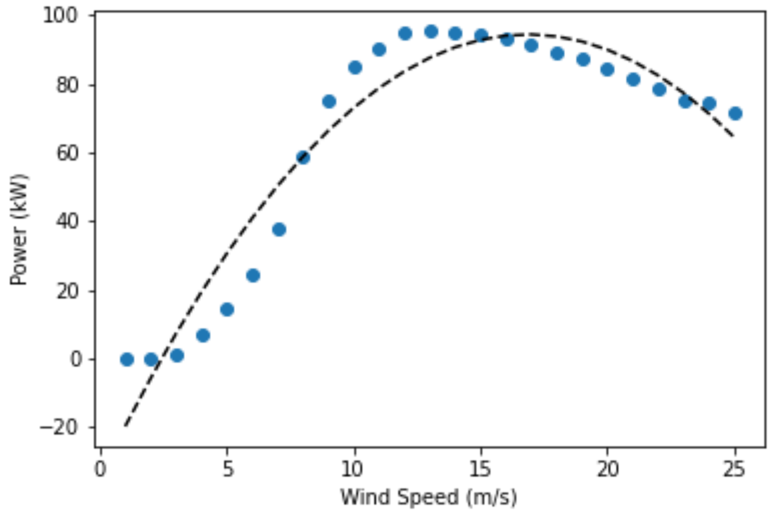
sklearn

Statsmodel

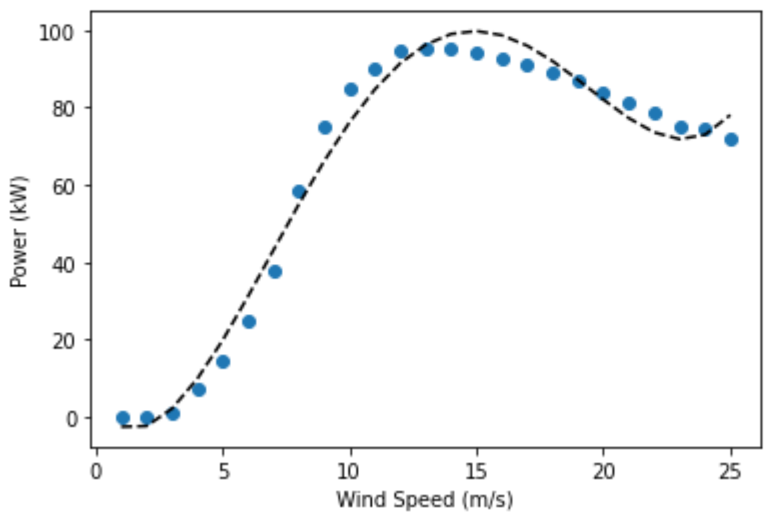


Polynomial Regression

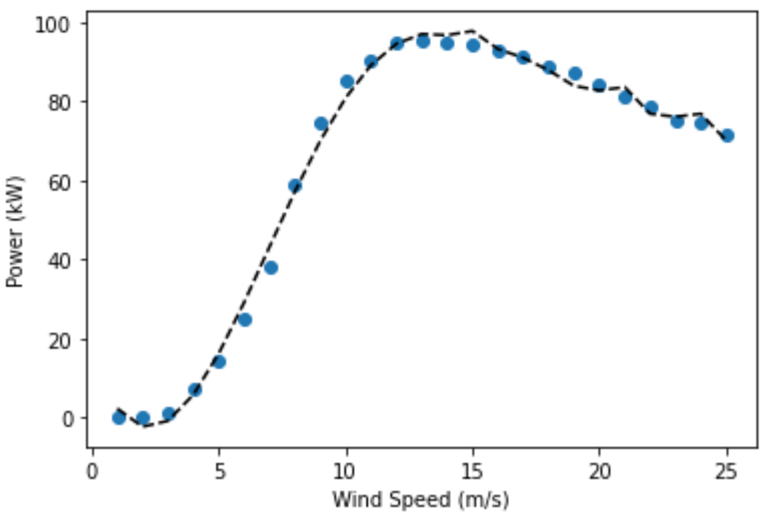
- Maybe I can fit it to some polynomial function of wind speed?



2 degrees



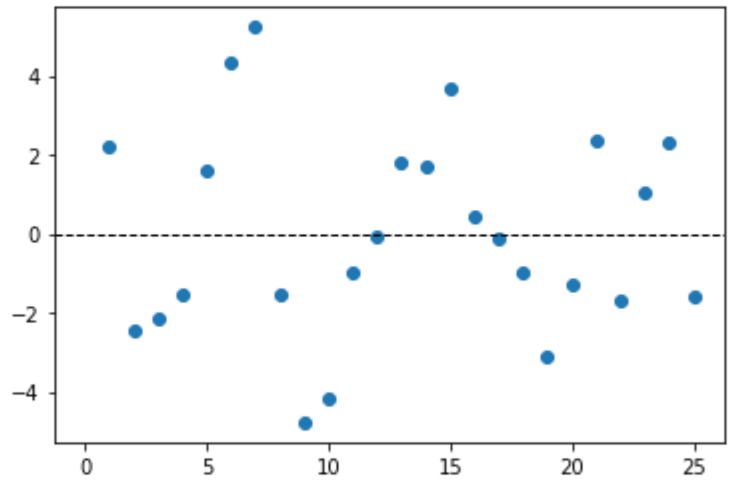
4 degrees



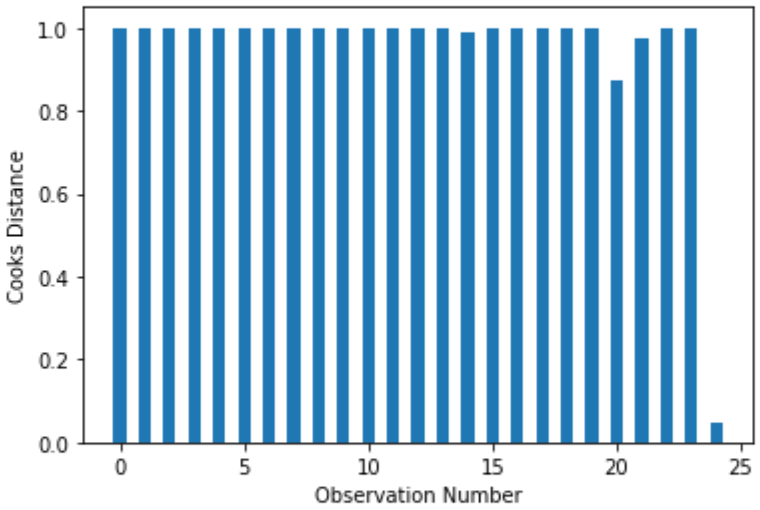
8 degrees

Checking Fit for Polynomial Regression

Plot the residuals



Calculate Cook's distance

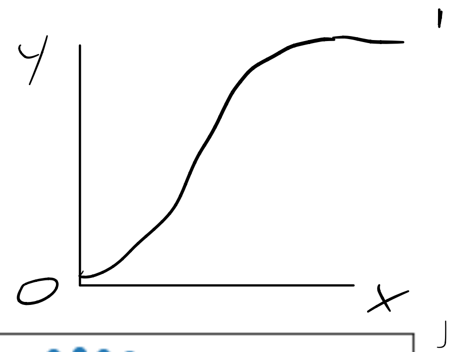


Check the significance of each order

	coef	std err	t	P>|t|	[0.025	0.975]
x1	-2.1513	2.709	-0.794	0.438	-7.867	3.564
x2	-0.3549	1.303	-0.272	0.789	-3.103	2.393
x3	0.5504	0.227	2.422	0.027	0.071	1.030
x4	-0.0646	0.018	-3.572	0.002	-0.103	-0.026
x5	0.0027	0.001	4.102	0.001	0.001	0.004
x6	-3.976e-05	9.2e-06	-4.321	0.000	-5.92e-05	-2.03e-05
x7	2.41e-09	1.23e-09	1.959	0.067	-1.86e-10	5.01e-09
x8	-1.127e-09	1.02e-09	-1.103	0.285	-3.28e-09	1.03e-09

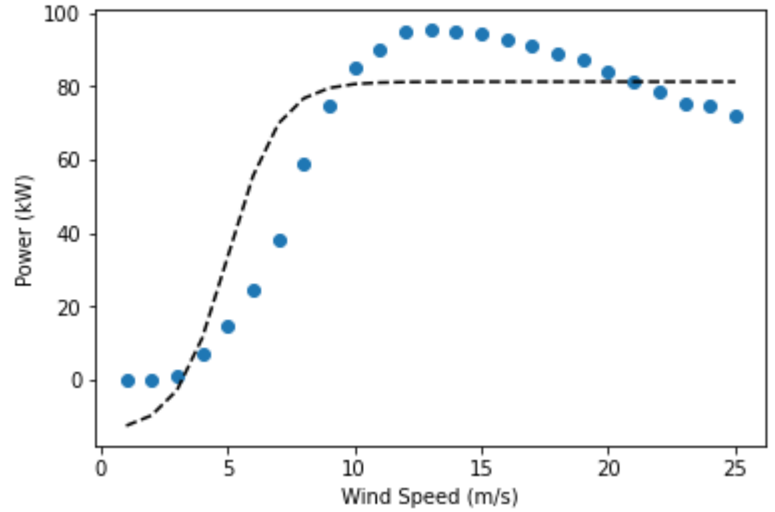
Other Functional Regression

$$\frac{1}{1+e^{\frac{(x-L)}{s}}}$$



Sigmoid

```
def sigmoid(x, loc=0, scale=1):
    return 1 / (1+np.exp(-(x-loc)/scale))

X = sigmoid(wind_speed, 5, 1.0).reshape(-1,1)
lr = LinearRegression()
lr.fit(X, power)
y_pred = lr.predict(X)
```



Other Functional Regression

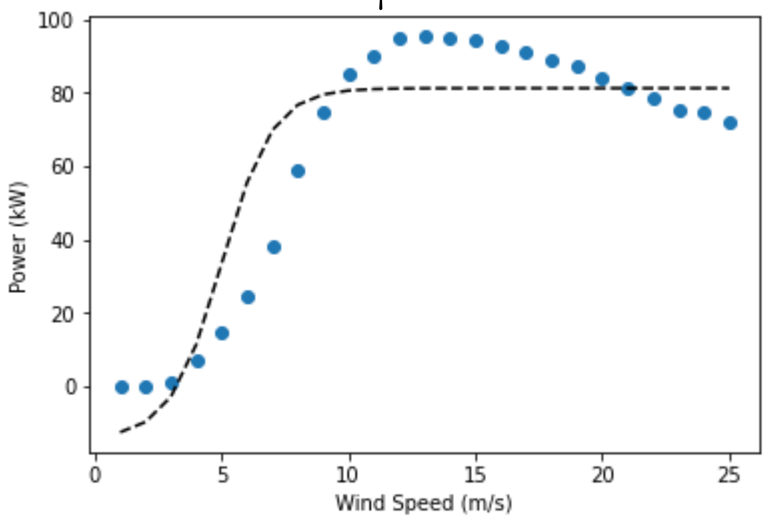
$$\tanh(y) = \frac{e^{2y-1}}{e^{2y+1}}$$

Low scale

High scale

Tanh

```
def tanh(x, loc=0, scale=1):
    y = (x-loc)/scale
    return (np.exp(2*y)-1) / (np.exp(2*y)+1)

X = tanh(wind_speed, 5, 2).reshape(-1,1)lr =
LinearRegression()
lr.fit(X, power)
y_pred = lr.predict(X)
```

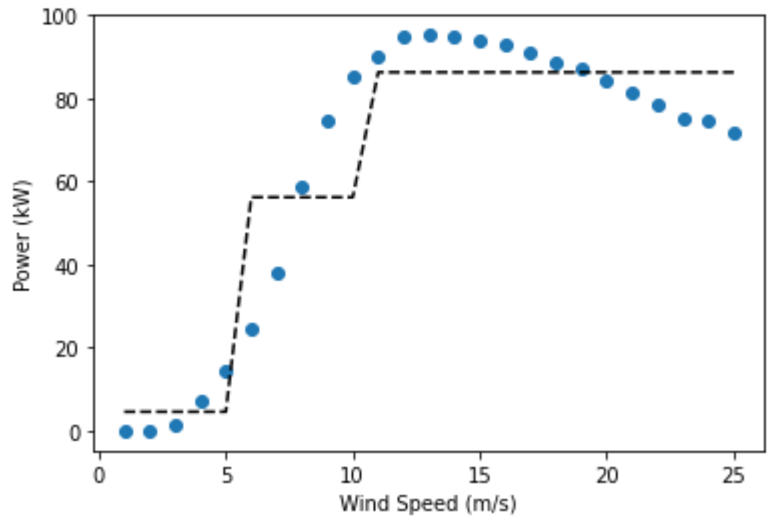


Other Functional Regression

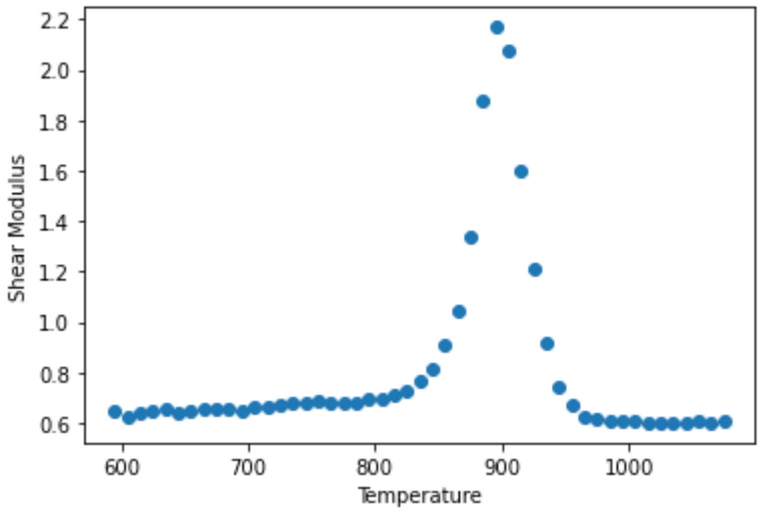
$$H(x) = \begin{cases} 1 & x \geq L \\ 0 & x < L \end{cases}$$

Heaviside

```
def heaviside(x, loc=0):
    return x>loc

X = np.concatenate([
    [heaviside(wind_speed, 5)],
    [heaviside(wind_speed, 10)]],
                    axis=0).T
lr = LinearRegression()
lr.fit(X, power)
y_pred = lr.predict(X)
```

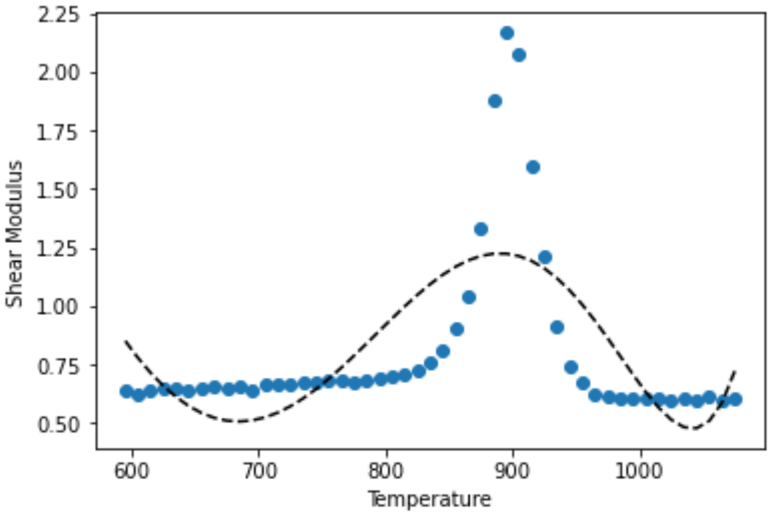


More Complex Example

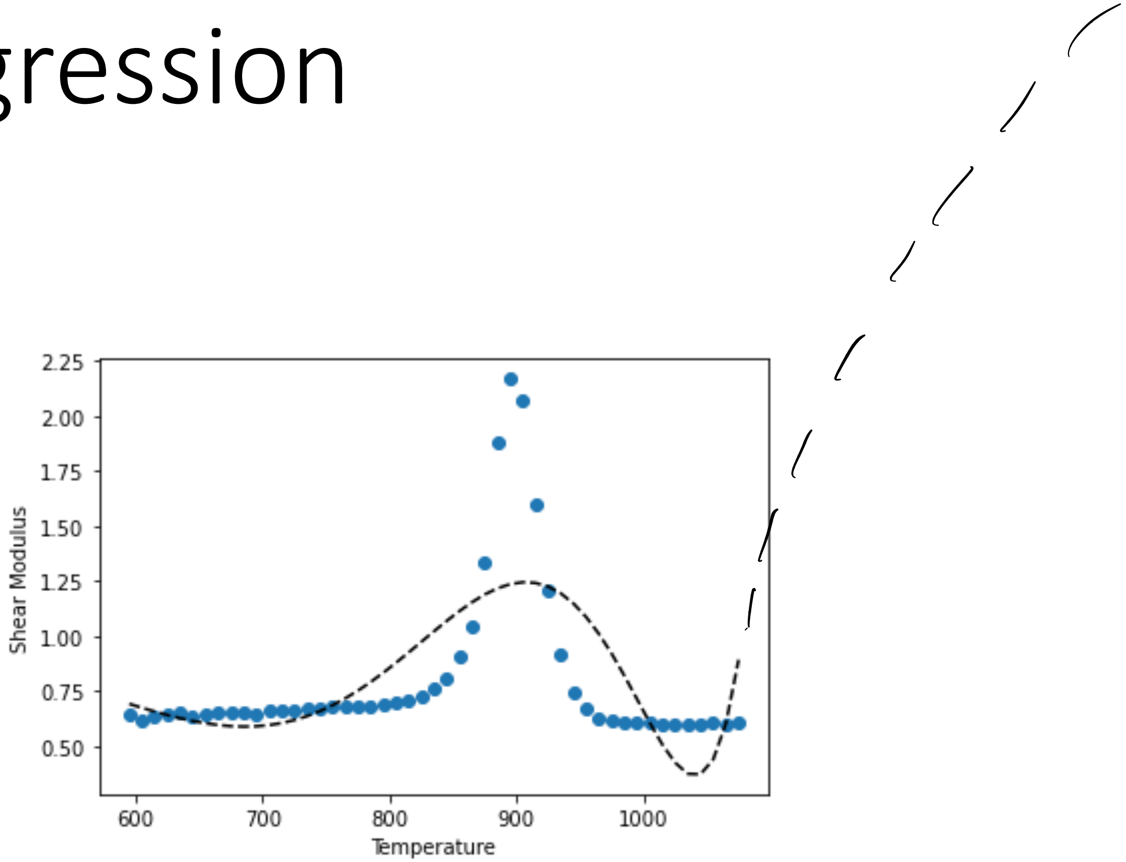


I want to predict the shear modulus of a material based on its temperature.

Trying Polynomial Regression



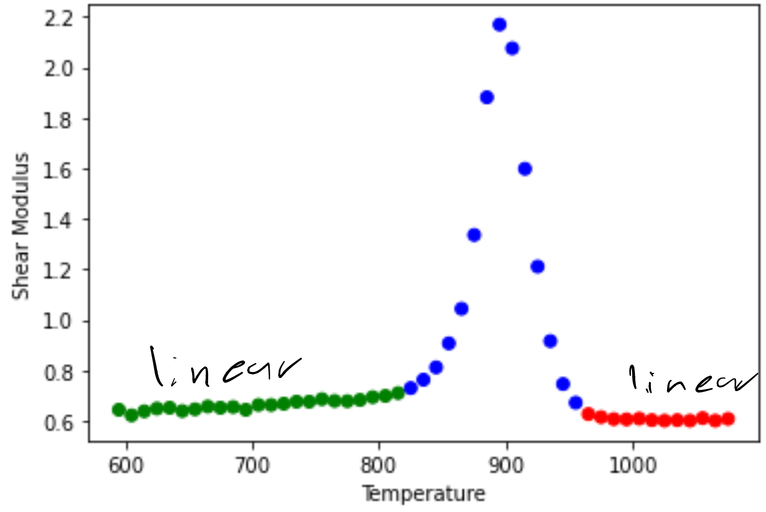
8 degrees



12 degrees

Piecewise Polynomials

interesting bit



What if I break this up into multiple polynomials?

Splines

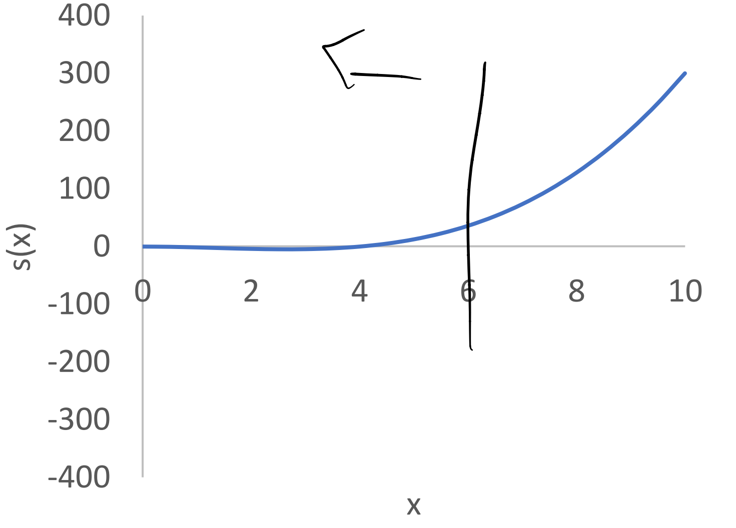
- Piecewise polynomials, or **splines**, are a flexible way to model all smooth functions

- Splines merge two polynomials at a point called a **knot**

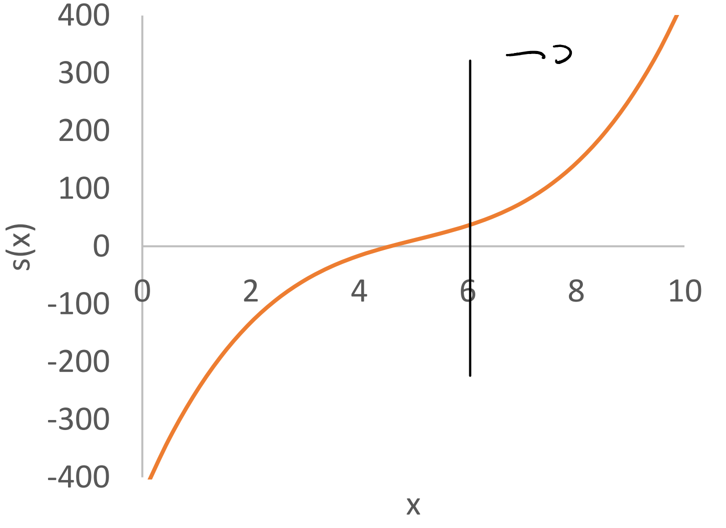
$$s(x) = \begin{cases} -2x^2 + 0.5x^3\,, & x < 6 \\ -2x^2 + 0.5x^3, -2(x-6)^3\,, & x \geq 6 \end{cases}$$

Splines

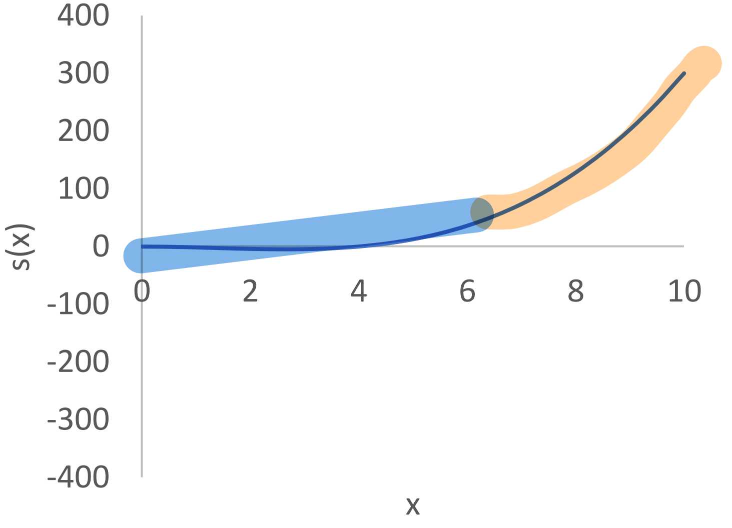
$$s(x) = \begin{cases} -2x^2 + 0.5x^3\,, & x < 6 \\ -2x^2 + 0.5x^3, -2(x-6)^3\,, & x \geq 6 \end{cases}$$



$s(x) = -2x^2 + 0.5x^3$



$s(x) = -2x^2 + 0.5x^3, -2(x-6)^3$

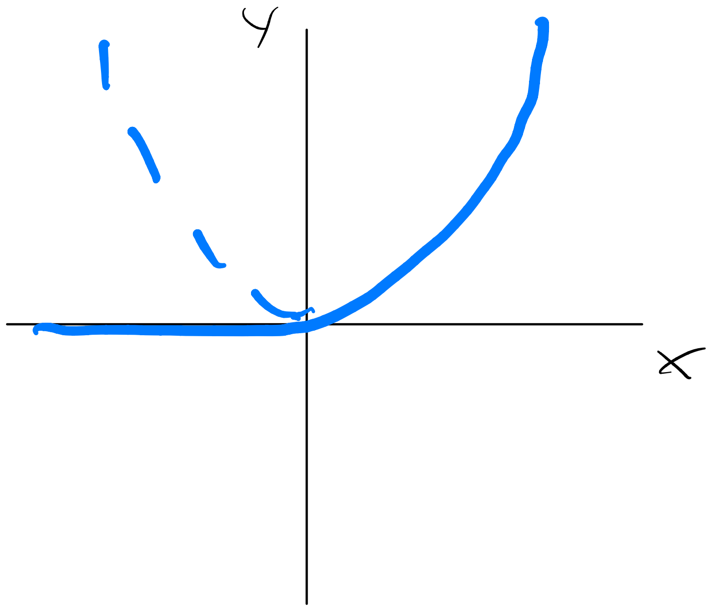


Truncated Power Functions

- A **truncated power function** is a polynomial function of the form

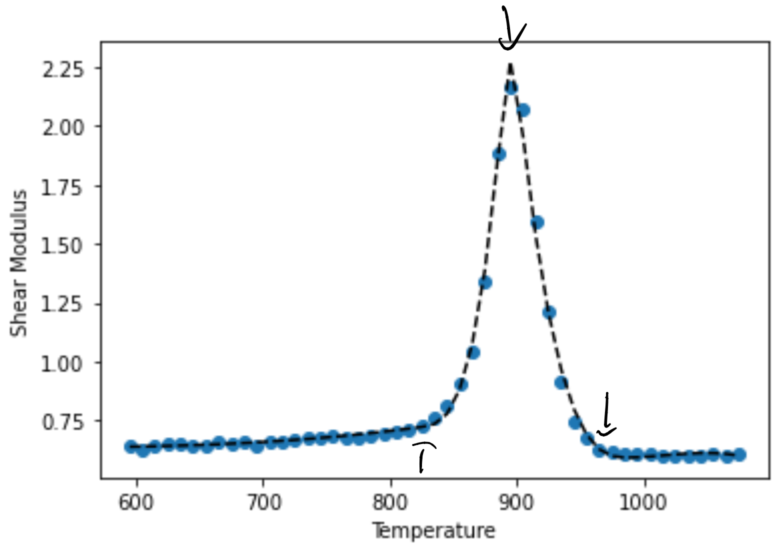
$$x_+^n = \begin{cases} x^n : x > 0 \\ 0 : x \leq 0 \end{cases}$$

```
def truncated_power_function(x, knot, degree=2):
    return x > knot * (x-knot)**degree
```

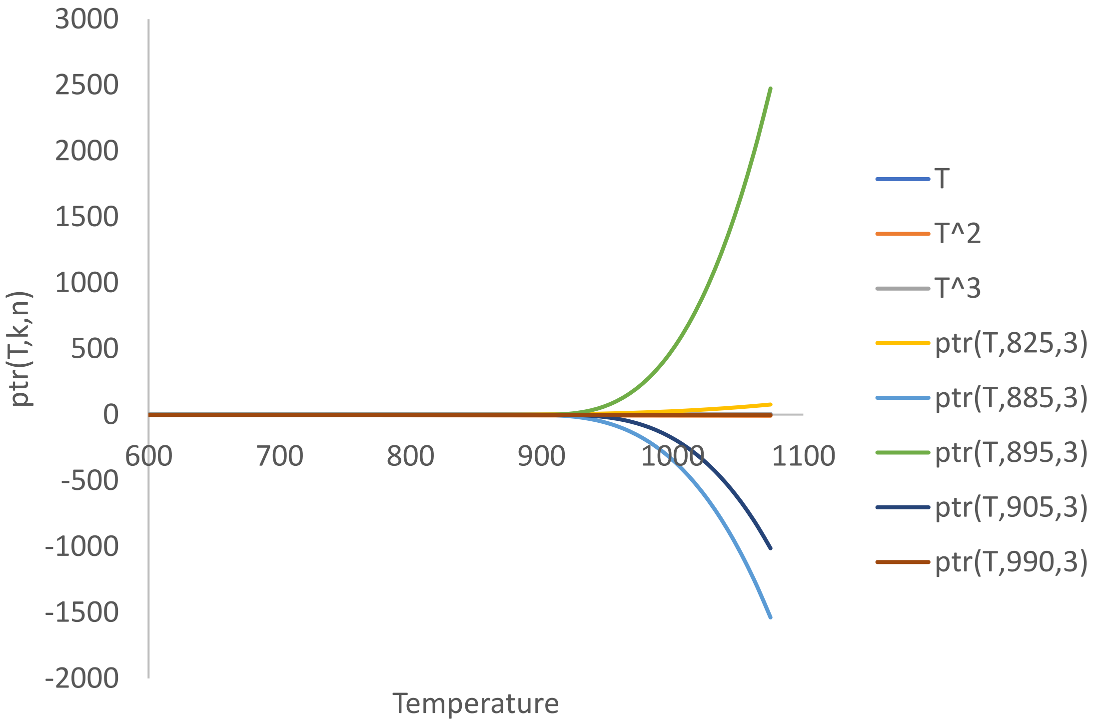


Fitting a TPF

```
X = np.concatenate([[temperature],
                    [temperature**2],
                    [temperature**3],
                    [tpf(temperature, 825, 3)],
                    [tpf(temperature, 885, 3)],
                    [tpf(temperature, 895, 3)],
                    [tpf(temperature, 905, 3)],
                    [tpf(temperature, 990, 3)]],
                    axis=0).T
lr.fit(X, heat)
```

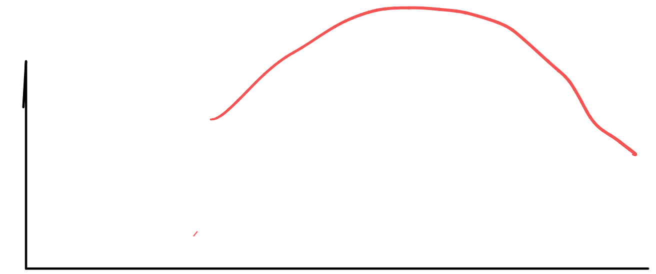


Components of a Spline

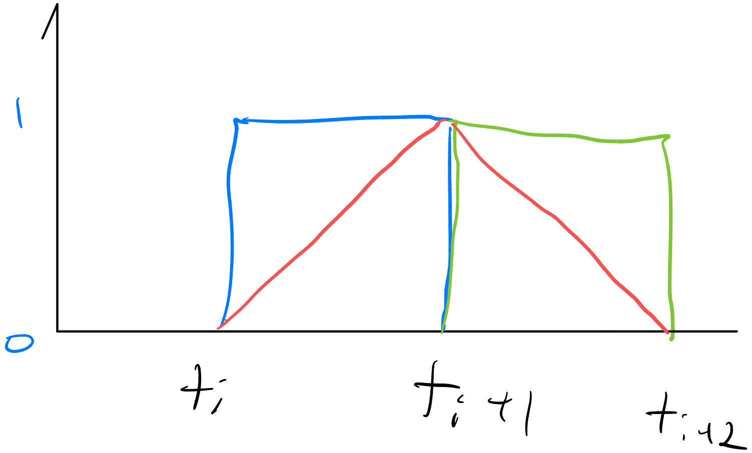


B-Splines

- Alternative to P-Splines
- Uses a function defined only between two knots:



$$B_{i,0}(x) = \begin{cases} 1\,;\, t_i \leq x \leq t_{i+1} \\ 0\,;\, otherwise \end{cases}$$



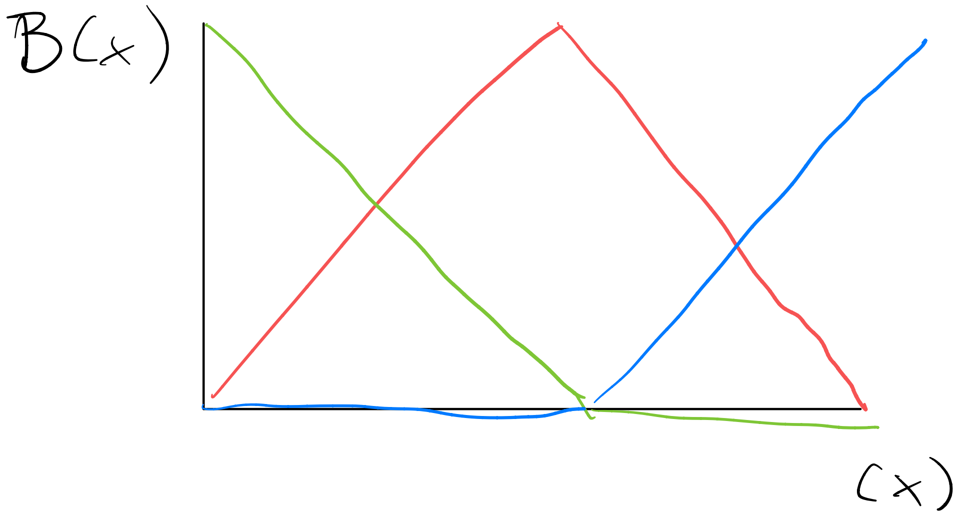
- Higher orders are defined recursively

degree

$$B_{i,p}(x) = \frac{x - t_i}{t_{i+p} - t_i} B_{i,k-1}(x) + \frac{t_{i+p+1} - x}{t_{i+p+1} - t_{i+1}} B_{i+1,p-1}(x)$$

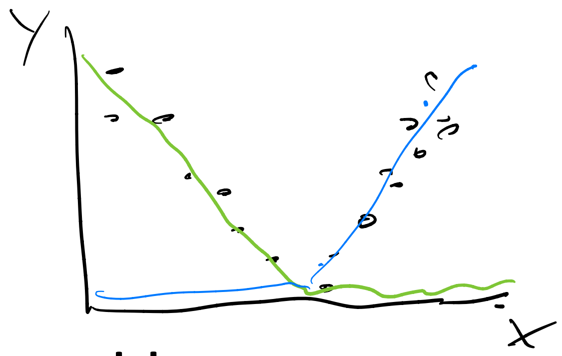
https://observablehq.com/@herbps10/b-splines <- great tool for visualizing this!

B-Splines



- Value estimate is a sum of all predictions

$$\hat{y} = s(x) = \sum_{i=1}^{n} c_i * B_{i,\cancel{k,t}}(x)$$

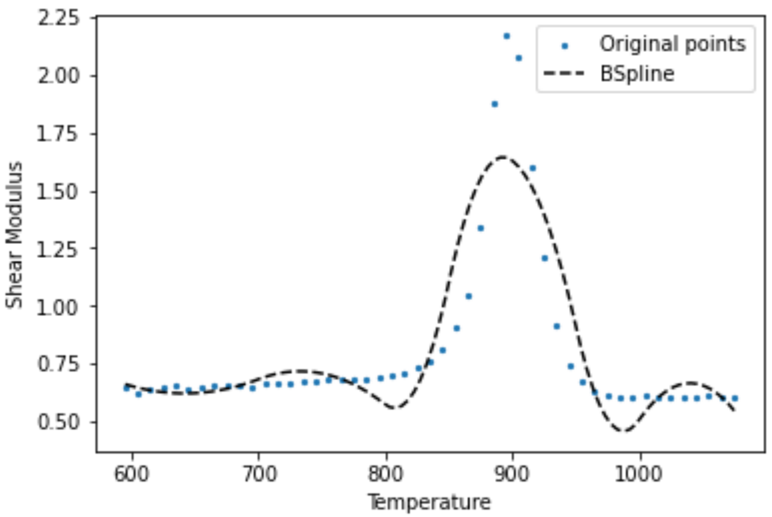


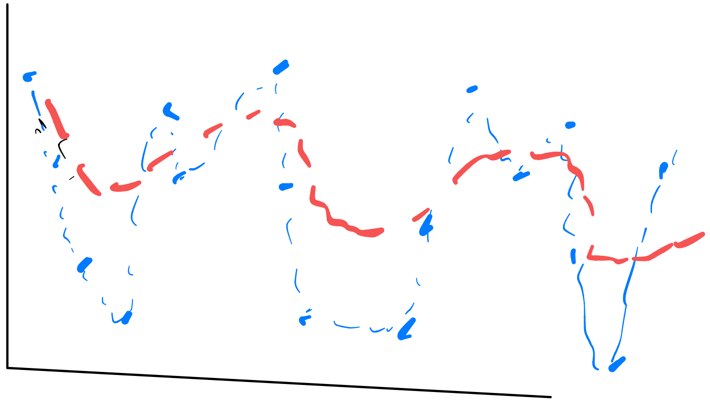
where $B_{i,k,t}(x)$ are B-spline basis functions of degree *k* and knots *t*.

$$s(x) = C_1 B_1 (x) + C_2 B_2 (x) + C_3 B_3 (x)$$

B-Spline Results

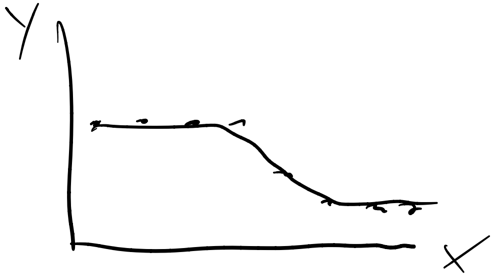
```
import scipy.interpolate as interpolate                X          y                      degree

knots = [700., 800., 850., 900., 950., 1000.]
knots, coefs, degree = interpolate.splrep(temperature, heat, task=-1, t=knots, k=2)

xr = np.linspace(x.min(), x.max(), 100) # x = temperature
spline = interpolate.BSpline(knots, coefs, degree, extrapolate=False)
                                 xr
y_pred = interpolate.splev(x, (knots, coefs, degree))
```





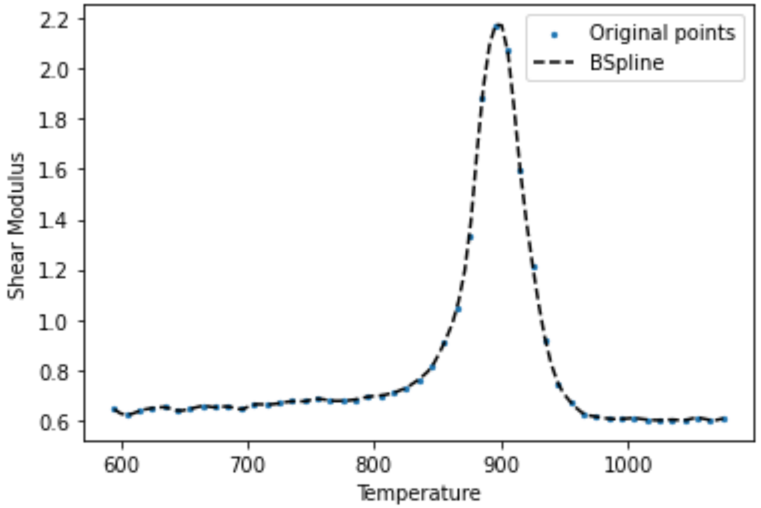
unsmoothed
smoothed

B-Spline Results



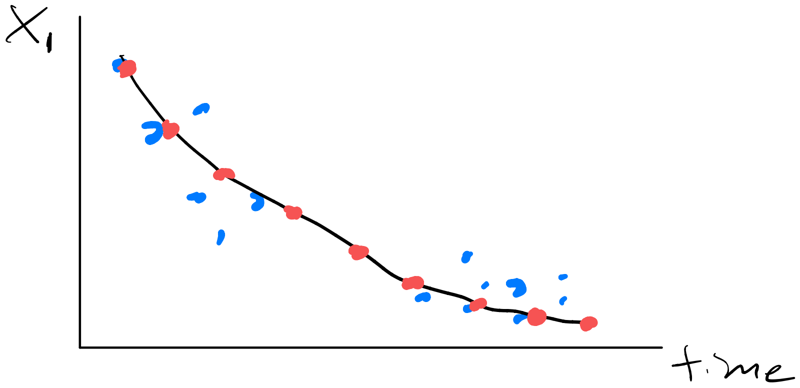

```
import scipy.interpolate as interpolate

knots, coefs, degree = interpolate.splrep(temperature, heat, k=2)

xr = np.linspace(x.min(), x.max(), 100)
spline = interpolate.BSpline(knots, coefs, degree, extrapolate=False)

y_pred = interpolate.splev(xx, (knots, coefs, degree))
```

s= 100



52 knots!

- interpolate.splrep expects x to be sorted



norm (μ, σ). rvs (100)

np. random. normal $(\mu, \sigma, N=100$

Takeaways

- You can transform your independent variables to make better predictors
- Polynomials can increase fit, but often don’t work well out-of-sample
- Polynomial splines and B-Splines fit incrementally over smaller ranges of the data
  - B-Splines do not work well on noisy data