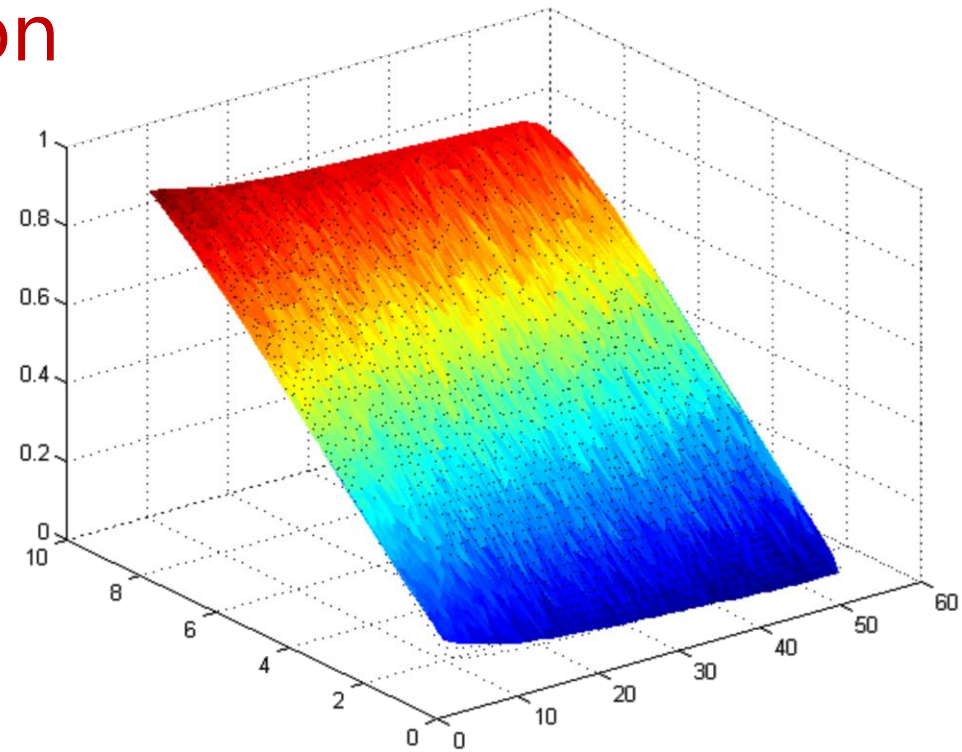
Chemical Engineering 4H03

Review of Regression



Jake Nease
McMaster University

Perspective

- Why are we going over Regression?
  - STATS sometimes does not cover the topic (fully)
  - It may have been 3 years since you reviewed regression
  - We want to visit this from an **optimization** perspective
  - Literally all predictive modeling methods do some sort of regression technique as a tool for model building

- This material is ripped directly from my 2E04 (formerly 3E04) class notes
  - Forgive me if you've done it before, but we can't make that assumption

Outline of this Section

- Derivation of Linear Least-Squares Regression
  - Optimization methodology
  - Coding in MATLAB

- Extension of ↑ to polynomial regression
  - In-class workshop

- Extension of ↑ to basis function regression
  - Critical for ANNs

- Training, testing, and fit metrics
  - R^2, RMSEP, PRESS…
  - More on these in their respective sections

Linear Least-Squares Regression

Maybe you'll learn something new.

Maybe you'll remember something old.

Maybe not.

Objective of Regression

- **Regression** is the act of creating a **model** intended to predict an **output** from a specific **input** given a **causal relationship** and a known model **structure** ... Whew

- Suppose you have a set of data points:

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$ KNOWN data

- You wish to approximate these data points by a regressing a line to the data in the form:

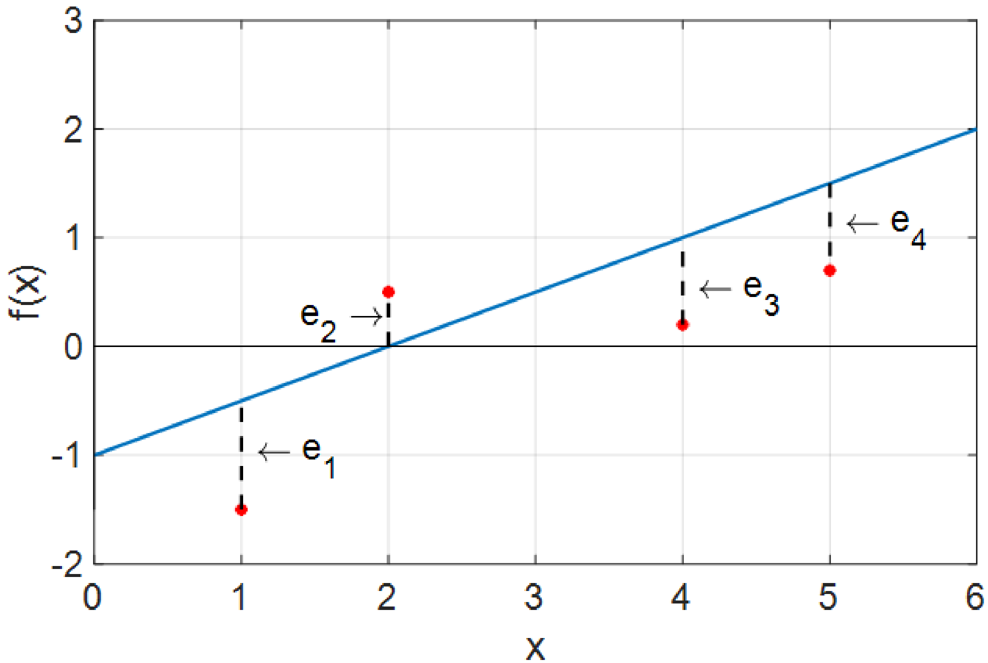
Some input

Some predicted output $\hat{y} = a_1 x + a_0$

Desired: find these coefficients (unknowns)

Objective of Regression

- Our first step is to quantify the **error** (ϵ_i) between the model prediction of the output ($\hat{y}_i$) and real point (y_i)
  - Our **OBJECTIVE** is to minimize the **total error**
  - Workshop: What types of error could we minimize?



x_i	y_i	$\hat{y}_i$	$\lvert e_i \rvert$
1	-1.5	0.5	2
2	0.5	0	0.5
4	0.2	1	0.8
5	0.7	1.5	0.8

Least-Squares Approach

- Since it is **CONVEX***, we like to represent total error of the model as the **sum of squared errors** (SSE)

$$SSE = \sum_{i=1}^{N} e_i^2$$

$$SSE = \sum_{i=1}^{N} (\hat{y}_i(x_i) - y_i)^2$$

$$SSE = \sum_{i=1}^{N} (a_0 + a_1 x_i - y_i)^2$$

This is called an **unconstrained** optimization formulation

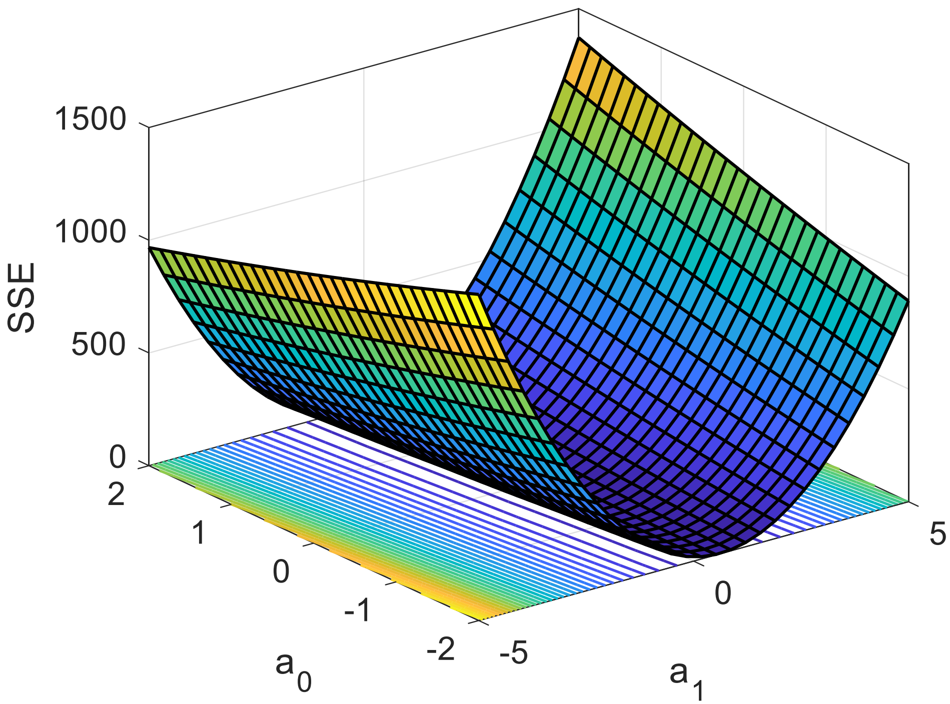
$$\min_{a_0 \; a_1} SSE \triangleq \sum_{i=1}^{N} (a_0 + a_1 x_i - y_i)^2$$

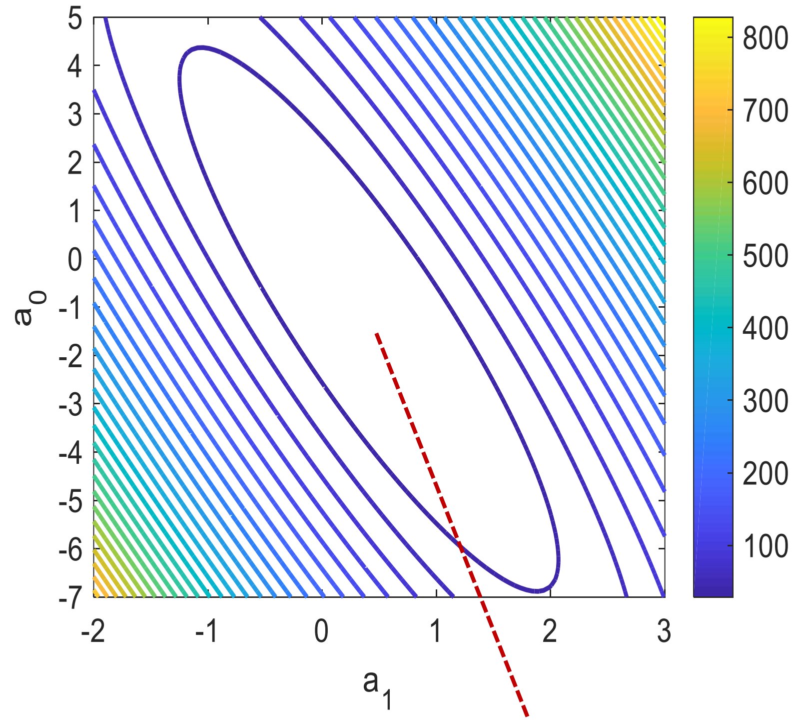
(Unconstrained)

*Anyone also in 4G will know way too much about this before the end…

Least-Squares Approach

- We can visualize the total SSE as a function of the decision variables a_0 and a_1 on contours/surfaces…





Looking for the bullseye

Analytical Solutions

- The function of SSE can be **solved to a global min** analytically using the principles of calculus:

1. Take the **partial derivative** of SSE with respect to one unknown, say a_0.
2. Take the **partial derivative** of SSE with respect to the second unknown, a_1.
3. Set each derivative to zero and solve for the two unknown parameters!
   – This is equivalent to solving $\nabla_{\boldsymbol{a}} SSE(\boldsymbol{a}) = 0$

Workshop – Analytical SSE Solution

- Complete the table below to identify the equations that must be solved to determine a_0 and a_1

You know these

$$\min_{a_0\ a_1} SSE \triangleq \sum_{i=1}^{N} (a_0 + a_1 x_i - y_i)^2$$

Partial WRT $\boldsymbol{a_0}$		Partial WRT $\boldsymbol{a_1}$	
$\frac{\partial SSE}{\partial a_0} =$		$\frac{\partial SSE}{\partial a_1} =$	
$0 =$		$0 =$	
$0 =$		$0 =$	

Workshop – Analytical SSE Solution

- Complete the table below to identify the equations that must be solved to determine a_0 and a_1

You know these

$$\min_{a_0 \ a_1} SSE \triangleq \sum_{i=1}^{N} (a_0 + a_1 x_i - y_i)^2$$

$$\underbrace{\begin{bmatrix} \sum_{i=1}^{N} 1 & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_0 \\ a_1 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} y_i x_i \end{bmatrix}}_{b}$$

Polynomial Regression

Bigger Models…

Better Results…

At What Cost!?

https://media1.tenor.com/images/1f5e60501265a7c9220c7ce29e49d781/tenor.gif?itemid=4715788

Linear-in-Parameters Regression

- Linear regression is wonderfully simple to implement
  - Example in MATLAB

- "Linear" regression in this context actually means **linear in the parameters**
  - That is, a_0 and a_1 as the variables are a linear combination weighted by **coefficients** $\sum_i x_i$ etc…

- We may therefore extend linear regression to any polynomial in one dimension. How?

$$\hat{y} = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + a_n x^n$$

Polynomial Regression

$$\hat{y} = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} + a_n x^n$$

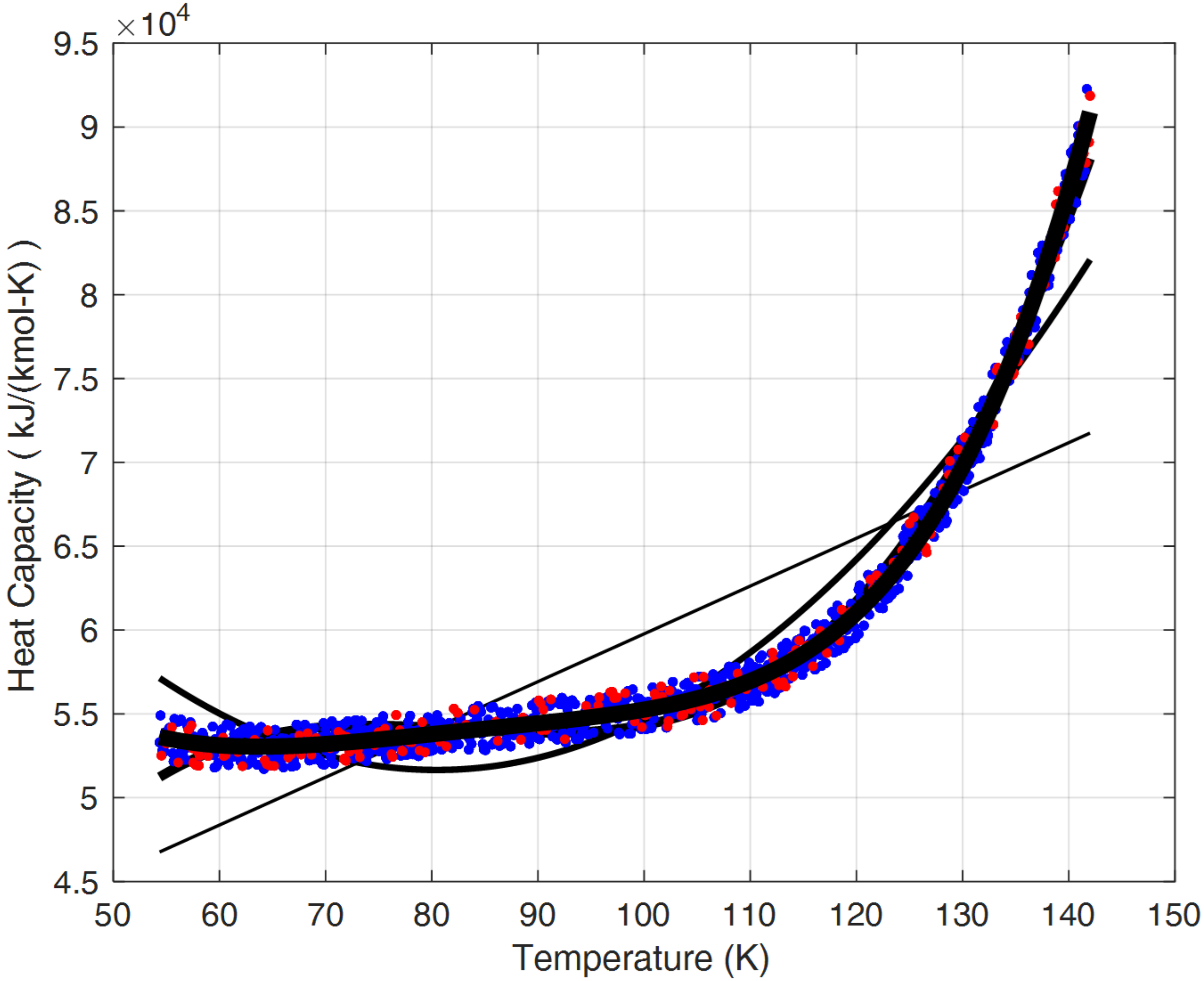
- Using ***math*** we can show that the linear-in-parameters regression for any polynomial of order n takes the form:

What happens when we take $n = 1$? What does that even mean?

$$\begin{bmatrix} \sum_{i=1}^{N} 1 & \cdots & \sum_{i=1}^{N} x_i^{n-1} & \sum_{i=1}^{N} x_i^{n} \\ \vdots & \vdots & \cdots & \vdots \\ \sum_{i=1}^{N} x_i^{n-1} & \ddots & \sum_{i=1}^{N} x_i^{2n-2} & \sum_{i=1}^{N} x_i^{2n-1} \\ \sum_{i=1}^{N} x_i^{n} & \cdots & \sum_{i=1}^{N} x_i^{2n-1} & \sum_{i=1}^{N} x_i^{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \vdots \\ \sum y_i \, x_i^{n-1} \\ \sum y_i \, x_i^{n} \end{bmatrix}$$

- This is derived the exact same way as linear regression
- ALSO very easy to code up!

Higher Degree = More Accurate?



Is the Model Any Good?

- We can assess "goodness of fit" using some rudimentary metrics such as R^2
  - We will examine other metrics when we get to more advanced modeling methods

This bad boy is just SSE

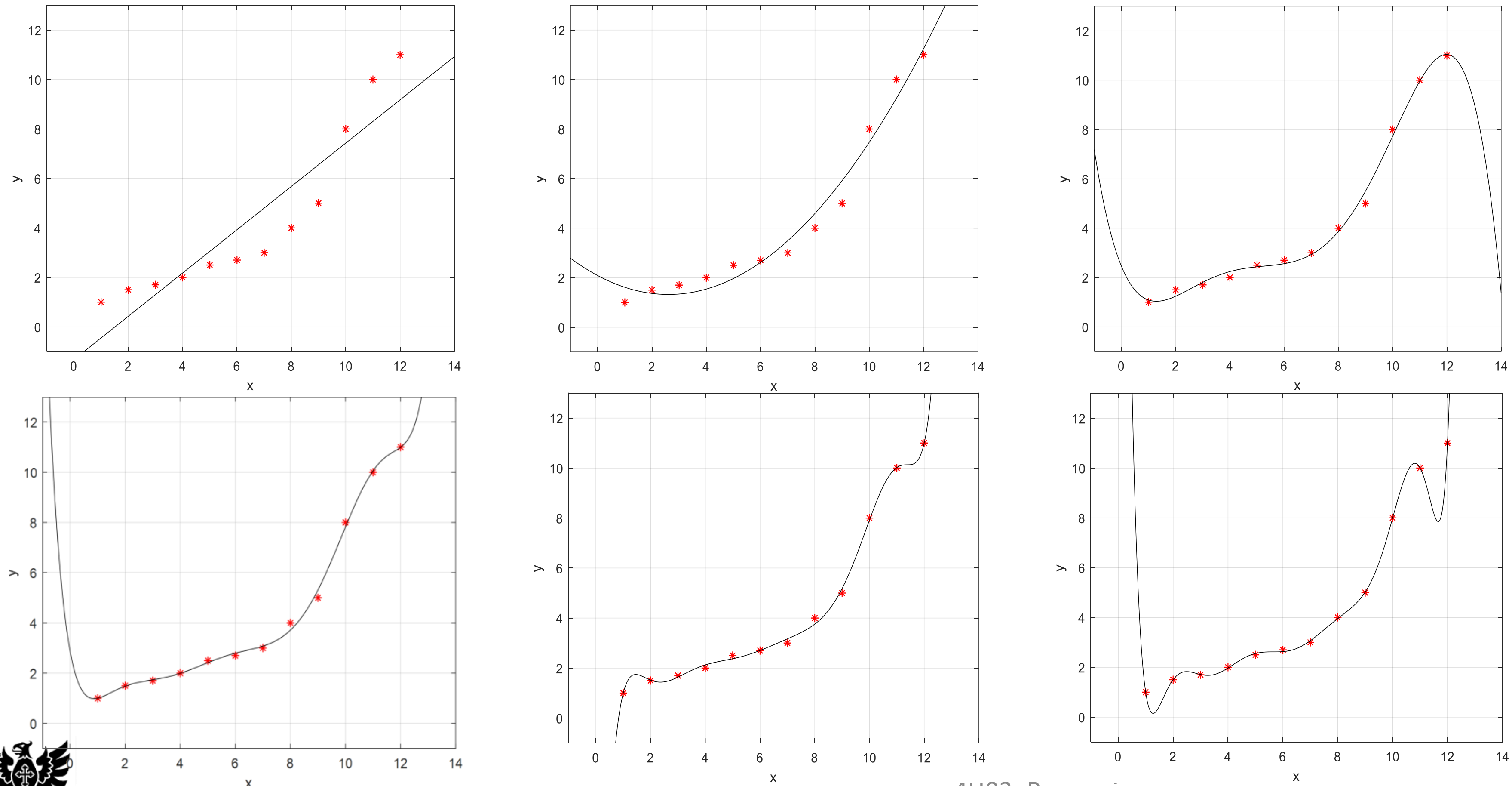
$$R^2 = 1 - \frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

Mean of all dependent outcomes

- R^2 is known as the "coefficient of determination"
  - Represents proportion of variance between the independent and dependent variables **predicted by the model**

Is the Model Any Good?

- A high R^2 **does not** always represent a well-fit model!
  - By mathematical definition, adding more terms **is guaranteed** to increase R^2 to unity ($R^2 = 1$)
  - If the data contain any sort of **noise**, overfitting the model can cause spurious and random results!



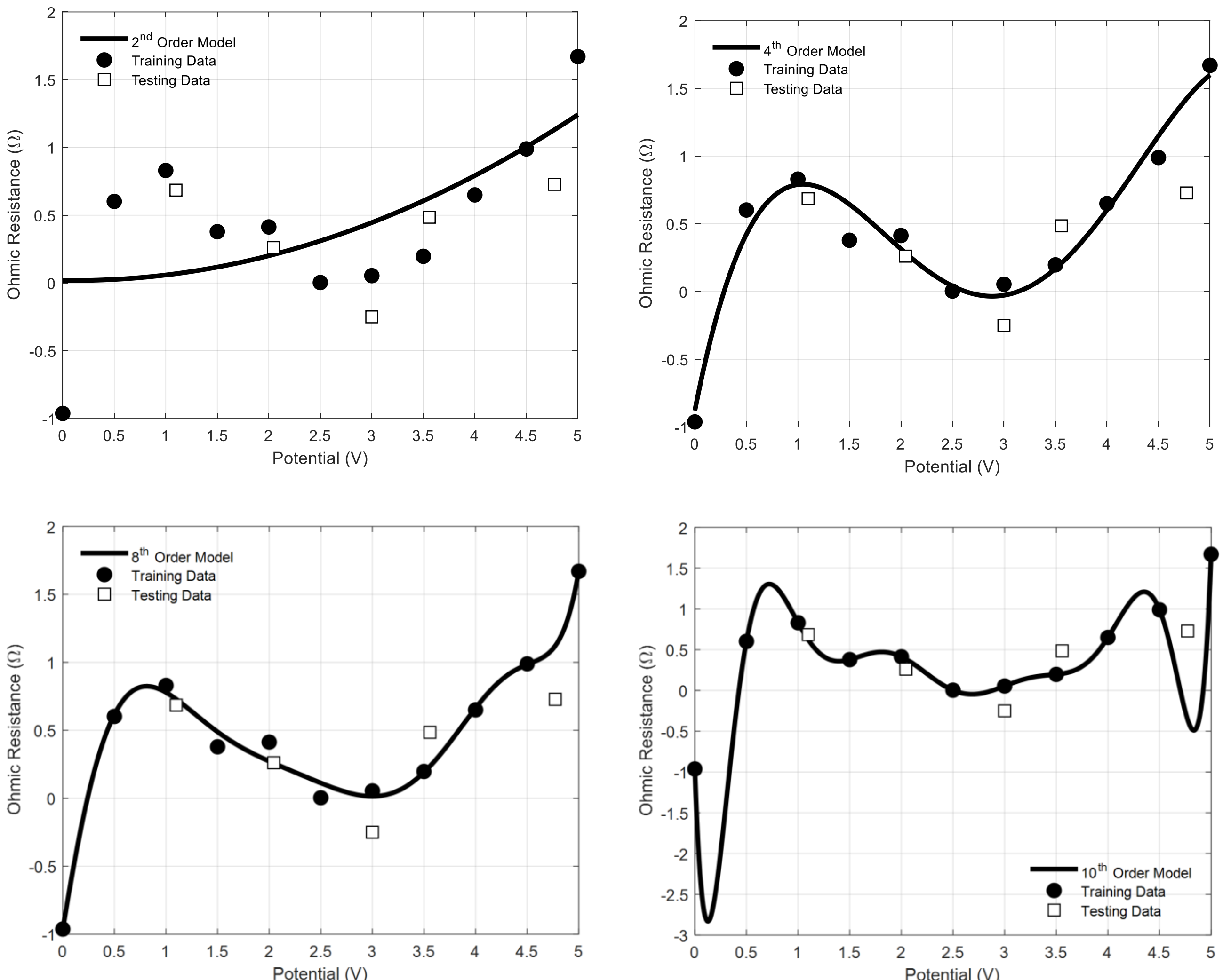
Is the Model Any Good?

- A high R^2 **does not** always represent a well-fit model!
  - In this course (and industry), models may be large, so we rely on metrics such as R^2 to guide us
  - Visualization is not always enough
  - We need a systematic method to assess model performance

- We have strategies to deal with this
  - **TRAINING**: predict model coefficients using 75-80% of data
    - Determine R^2 for the training set only (R^2_{TR})
  - **TESTING**: attempt to predict remaining 20-25% of data
    - Determine R^2 for testing set only (R^2_{TE})
  - If $\left|R^2_{TR} - R^2_{TE}\right| < \epsilon$ then the model is **not biased or overfit**

Workshop - Overfitting

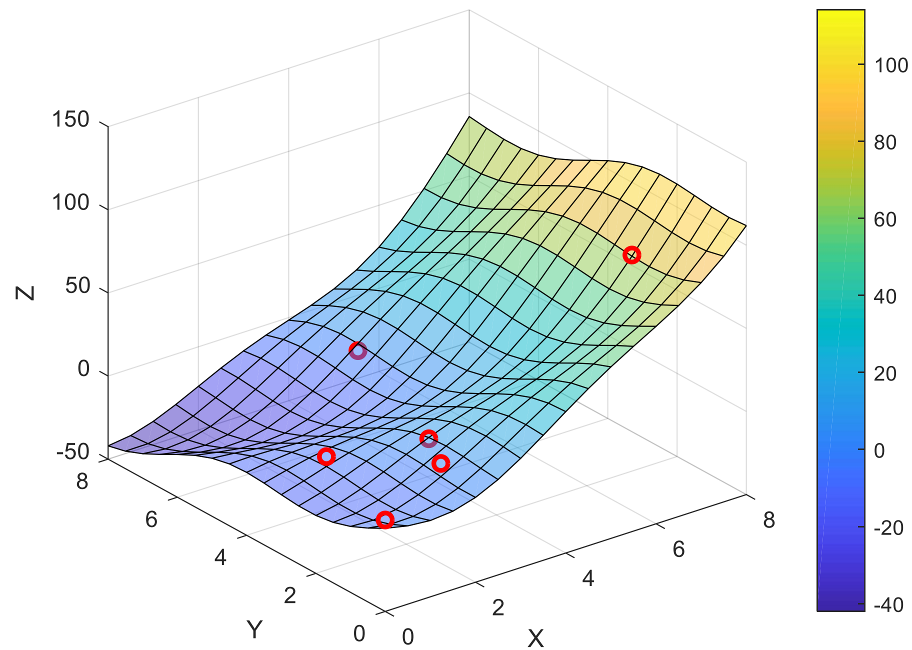
- Which model is most appropriate?
- Thought exercise: How would R^2_{TR} and R^2_{TE} evolve with higher order models?



McMaster University ENGINEERING

Basis Function Regression

This is where I lose the ChE 2E04 class.



Basis Function Regression

- We can handle polynomials, but we can actually also handle **any function** as long as the model is still linear with respect to the coefficients $\boldsymbol{a}$
- Consider some $\hat{y} = f(x)$ as an arbitrary function:

$$\hat{y} = a_1 f_1(x) + a_2 f_2(x) + \cdots + a_m f_m(x) + \cdots + a_M f_M(x)$$

Each of these f_i are functions of the independent data x

$$\boxed{\hat{y} = \sum_{m=1}^{M} a_m f_m(x)}$$

So, if $f_1(x) = 1$ and $f_2(x) = x$… We just get $\hat{y} = a_1 + a_2 x$. Wild.

Workshop: Basis Regression

$$\hat{y} = \sum_{m=1}^{M} a_m f_m(x)$$

- Starting with our definition of a basis function model, derive the system of equations that represents the minimum SSE for a selection of all a_m
  - Start with definition of SSE (or SE if you like)

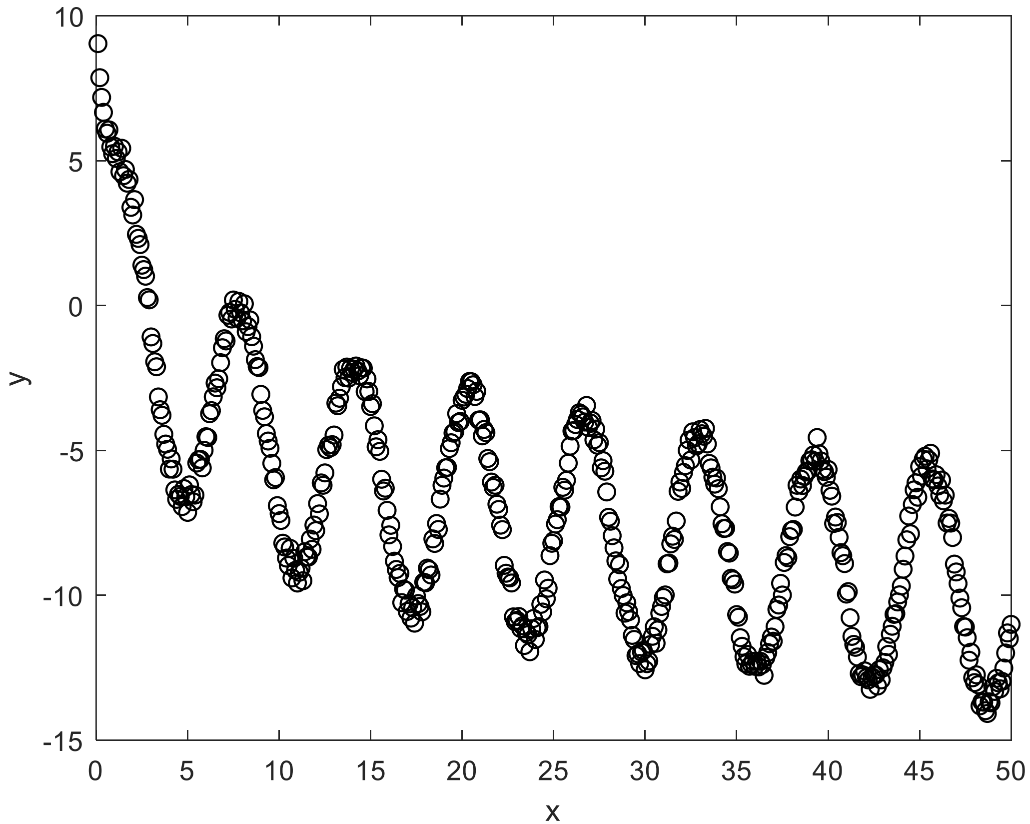
Workshop: Basis Regression

$$\boxed{\hat{y} = \sum_{m=1}^{M} a_m f_m(x)}$$

$$\underbrace{\begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_1 \\ \vdots \\ a_m \\ \vdots \\ a_M \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}}_{b}$$

Basis Function Example

- Consider the data set in the figure below
  - Suggest a good model (*i.e.* set of basis functions)
  - MATLAB demo



Ask yourselves… What are the features? Can you represent them mathematically?

I’m not sure what kind of system would behave like this, buuuut let’s just roll with it for now

Basis Regression in > 1D

- As a final note, it is possible to achieve **any** regression (linear, polynomial, BF) in multiple dimensions
  - Math does not change significantly
  - Computation costs takes a hit, but that's usually OK
  - Now have multiple dimensions to $\boldsymbol{x}$ and y
  - For below, $\boldsymbol{x}$ is two-dimensional (surface regression)
  - Remember: f_m could just be polynomials of $\boldsymbol{x}$

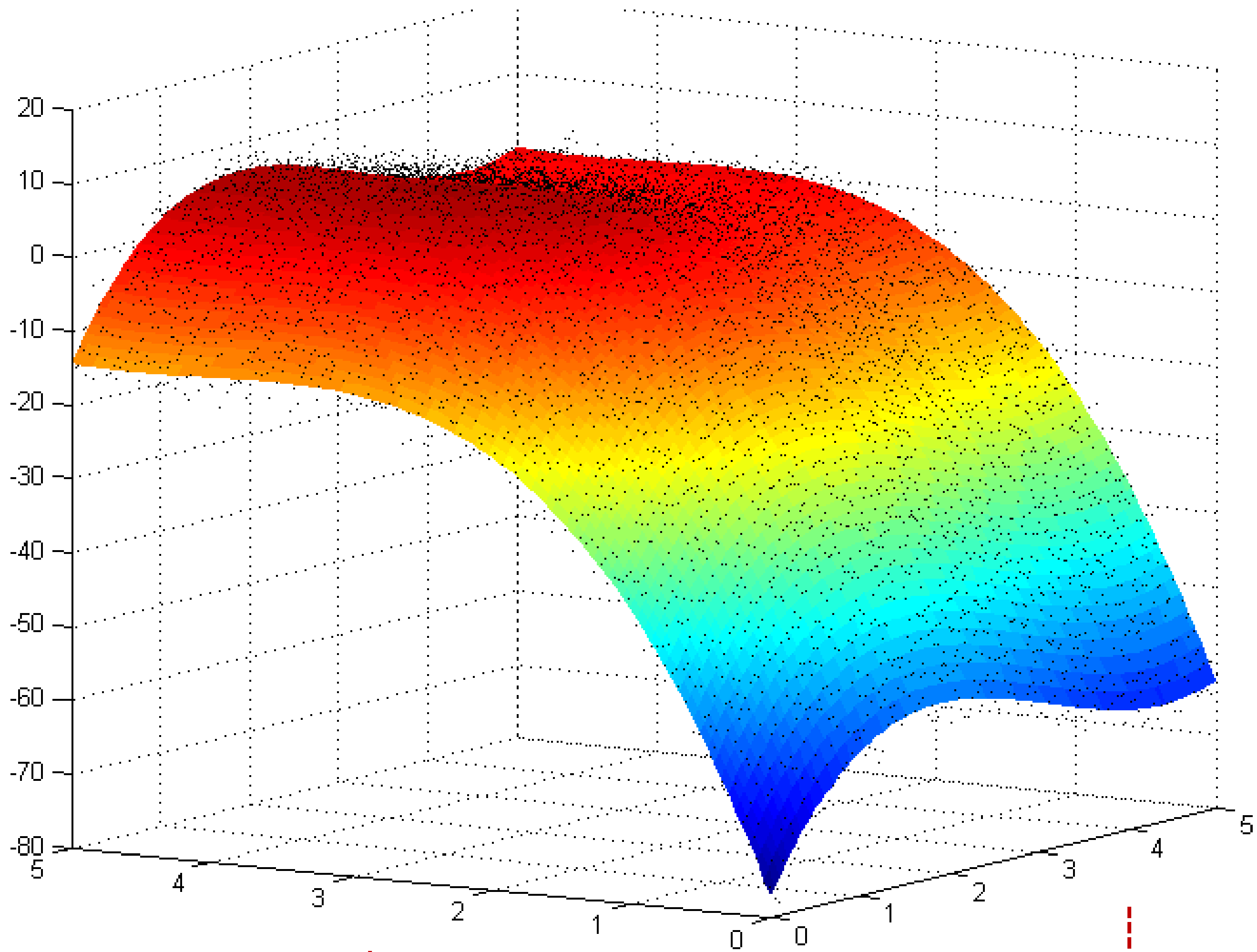
- Consider a data set with $\boldsymbol{x} \triangleq [x_1, x_2]^T$
  - We want our regression functions f_m to incorporate relationships between the output $y_{i,j}$ for some $\boldsymbol{x_{i,j}} = \left[x_{1,i}\ x_{2,j}\right]^T$
  - NOTE that any function f_m could contain or omit any elements in x. For example, $f_1(\boldsymbol{x}) = x_1^2$ is a function of x_1 AND x_2

Basis Regression in > 1D

$$\underbrace{\begin{bmatrix} \sum_{i=1}^{N_x}\sum_{j=1}^{N_y} f_1(\boldsymbol{x_{i,j}})f_1(\boldsymbol{x_{i,j}}) & \cdots & \sum_{i=1}^{N_x}\sum_{j=1}^{N_y} f_M(\boldsymbol{x_{i,j}})f_1(\boldsymbol{x_{i,j}}) \\ \vdots & \sum_{i=1}^{N_x}\sum_{j=1}^{N_y} f_m(\boldsymbol{x_{i,j}})f_m(\boldsymbol{x_{i,j}}) & \vdots \\ \sum_{i=1}^{N_x}\sum_{j=1}^{N_y} f_1(\boldsymbol{x_{i,j}})f_M(\boldsymbol{x_{i,j}}) & \cdots & \sum_{i=1}^{N_x}\sum_{j=1}^{N_y} f_M(\boldsymbol{x_{i,j}})f_M(\boldsymbol{x_{i,j}}) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_1 \\ \vdots \\ a_m \\ \vdots \\ a_M \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} \sum_{i=1}^{N_x}\sum_{j=1}^{N_y} f_1(\boldsymbol{x_{i,j}})y_{i,j} \\ \vdots \\ \sum_{i=1}^{N_x}\sum_{j=1}^{N_y} f_m(\boldsymbol{x_{i,j}})y_{i,j} \\ \vdots \\ \sum_{i=1}^{N_x}\sum_{j=1}^{N_y} f_M(\boldsymbol{x_{i,j}})y_{i,j} \end{bmatrix}}_{b}$$

- RECALL $\boldsymbol{x_{i,j}} = \left[x_{1,i}\ x_{2,j}\right]^T$
  - It is important we explore every combination of $\boldsymbol{x}_{i,j}$ during the regression step, even if $f_m(\boldsymbol{x}_{i,j})$ does not use it (WHY??)
- Note it is definitely possible to have $\boldsymbol{x} = [x_1 \ldots x_K]$
  - Recall K was our *number of columns* in the data set (previous class)

Basis Regression in > 1D Example



4th order polynomial in x_1

3rd order polynomial in x_2

Figure courtesy of Dr. Adams

Putting it all in Context

The truth behind LVMs and ANNs

https://vignette.wikia.nocookie.net/nintendo/images/d/d0/Lens_of_truth.png/revision/latest/scale-to-width-down/340?cb=20110810181524&path-prefix=en

Latent Variable Methods

- This is where I would like to go next
- Method is all about fitting **linear orthogonal hyperplanes** to a set of data to maximize explained variance between data and model prediction
  - This is linear regression in multiple dimensions
  - There is a small catch here, but that's about it!
- Algorithms will say "regress Y onto X" (for example)
  - We are looking for the hyperplane coefficients
  - PCA/PLS are therefore deeply rooted in **linear** regression

Artificial Neural Networks

- Method is all about finding **weights** (aka coefficients) that combine linear, nonlinear (*e.g.* tanh), and biases together in one or more layers (that feed each other)
  - ANNs are deeply rooted in **basis function** regression
  - Can't use SSE techniques because one "layer" feeds the next
    - This makes subsequent weights nonlinearly dependent on previous weights
  - Requires an iterative procedure

- Knowing how weights can be chosen to minimize SSE is a fundamental requirement for understanding ANNs

Final Words

- OK, enough with the review already

- Next up: **Latent Variable Methods (?)**
  - Terminology/uses
  - Derivation from eigenvector decomposition
  - Interpretation of scores/loadings
  - NIPALS algorithm