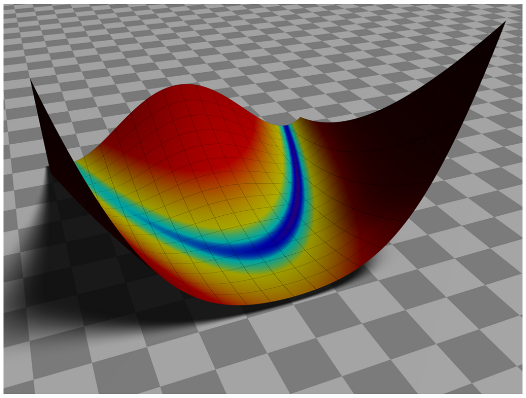
# Constructing a Surrogate

Lecture 32



ME EN 575
Andrew Ning
aning@byu.edu

## Outline

Polynomial Models

Kriging

# Polynomial Models

## Try It!

Let's simulate experimental (or noisy computational) data:

The actual underlying function is:

$$f = 3x^2 + 2x + 1$$

but we will add some Gaussian noise with mean 0 and standard deviation 1 (use `randn`).

Create 20 data points from $[-2 \ldots 2]$.

Now, assume you are just given these data points:

$$(x^{(i)}, f^{(i)}) \text{ for } i = 1 \ldots 20$$

You don't know the undelrying function but you assume it is quadratic:

$$f = ax^2 + bx + c$$

Estimate $a, b,$ and $c$.

## An n-Dimensional Polynomial Model

$$\hat{f} = \psi^T w$$

where $w$ is a vector of weights and $\psi$ is a vector of preselected polynomial basis functions like:

$$\psi \in \{1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2^2, x_1^2x_3 \ldots\}$$

Two main tasks:

1. Decide what terms to put in $\psi$
2. Estimate $w$

We will start with task 2, assuming that we already completed task 1.

## Estimate weights ($w$)

From sampling we selected a bunch of points and evaluated the function at those points: $(x^{(i)}, f^{(i)})$ for $i = 1 \ldots n$

We call this data the training data, because we will use it to train our model.

The error for a given point is:

$$\hat{f}(x^{(i)}) - f^{(i)}$$

We don't want to just sum the errors (because negative errors would cancel with positive errors), so we sum the square of the errors

$$\text{minimize} \sum_i \left( \hat{f}(x^{(i)}) - f^{(i)} \right)^2$$

This is called a least squares solution.

We can rewrite this in a more familiar matrix notation.

$$\hat{\mathbf{f}} = \Psi w$$

where $\Psi$ is matrix:

$$\Psi = \begin{bmatrix} - & \psi(x^{(1)})^T & - \\ - & \psi(x^{(2)})^T & - \\ & \vdots & \\ - & \psi(x^{(n)})^T & - \end{bmatrix}$$

The previous minimization problem can be expressed as:

$$\text{minimize } ||\Psi w - f||^2$$

where

$$f = \begin{bmatrix} f^{(1)} \\ f^{(2)} \\ \vdots \\ f^{(n)} \end{bmatrix}$$

You've seen this before: $y = Ax$ where $A \in \mathcal{R}^{m \times n}$ and $m > n$

This means there are more equations than unknowns (i.e., we sampled at more points than the number of coefficients $w$ we need to estimate).

There is generally not a solution, the problem is overdetermined. Instead, we seek a least squares solution, i.e., one that minimizes

$$||Ax - y||^2$$

Matlab:

```
x = A\b
```

Python:

```
x = numpy.linalg.lstsq(A, b)
```

## Try It Again!

Given these data points:

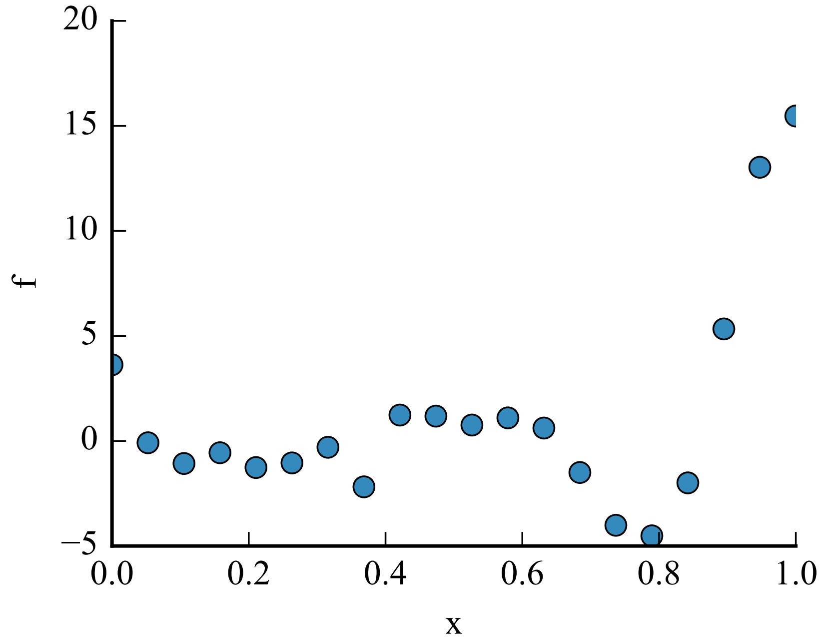
$$(x^{(i)}, f^{(i)}) \text{ for } i = 1 \ldots 20$$

Assume that the underlying function is quadratic:

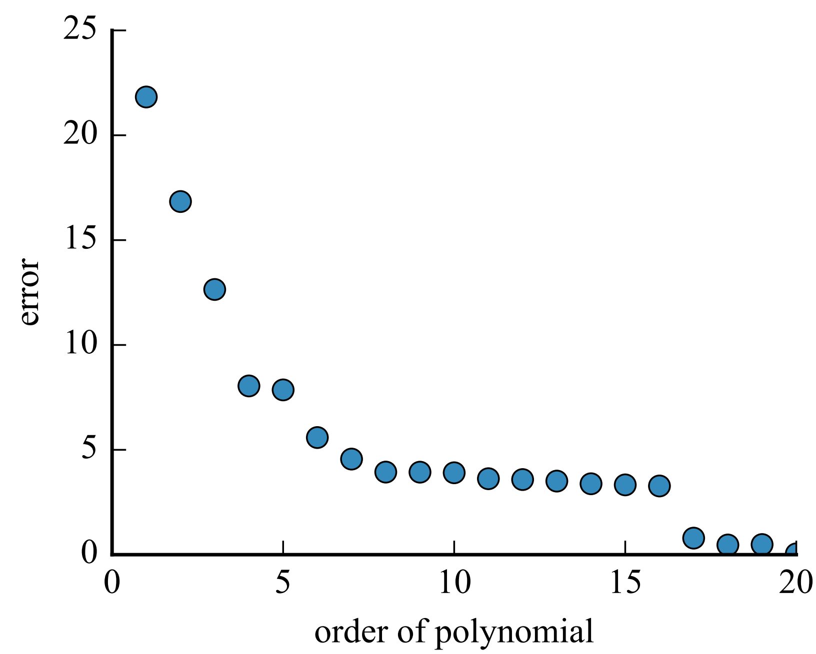
$$f = ax^2 + bx + c$$

Estimate $a, b$, and $c$.

# Choosing the basis functions $\psi$

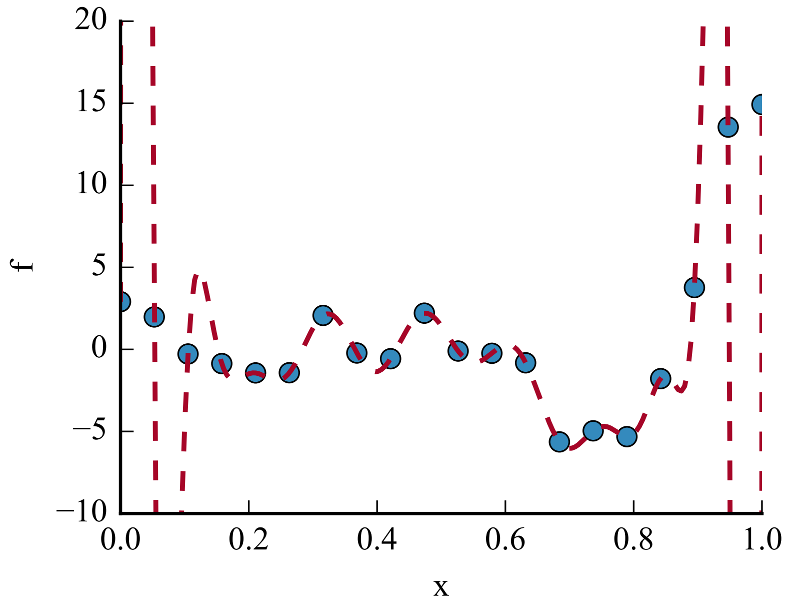



Proposal: Let's try different orders of polynomials and see what the error looks like.



So, the higher order the polynomial, the better the model!

Here is what the model predicts for a 20th order polynomial:




This problem is called overfitting (underfitting can also be a problem). What is the solution?

Cross-validation. The fundamental problem is that our training data and testing data are the same data. We need our training data and testing data to be separate.

# Simple Cross Validation

1. Randomly split data into a training set and a validation set (i.e., a 70/30 split).
2. Train each candidate models (the different options for $\psi$) using only the training set, but evaluate the error with the validation set.
3. Choose the model with the lowest error on the validation set, and optionally retrain that model using all of the data.

# k-fold Cross Validation

1. Randomly split your data into n sets (e.g., n = 10).
2. Train each candidate models using the data from all sets except one (e.g., 9 of the 10 sets), and use the remaining set for validation. Repeat for all n possible validation sets and average your performance.
3. Choose the model with the lowest average error on all the n validation sets.

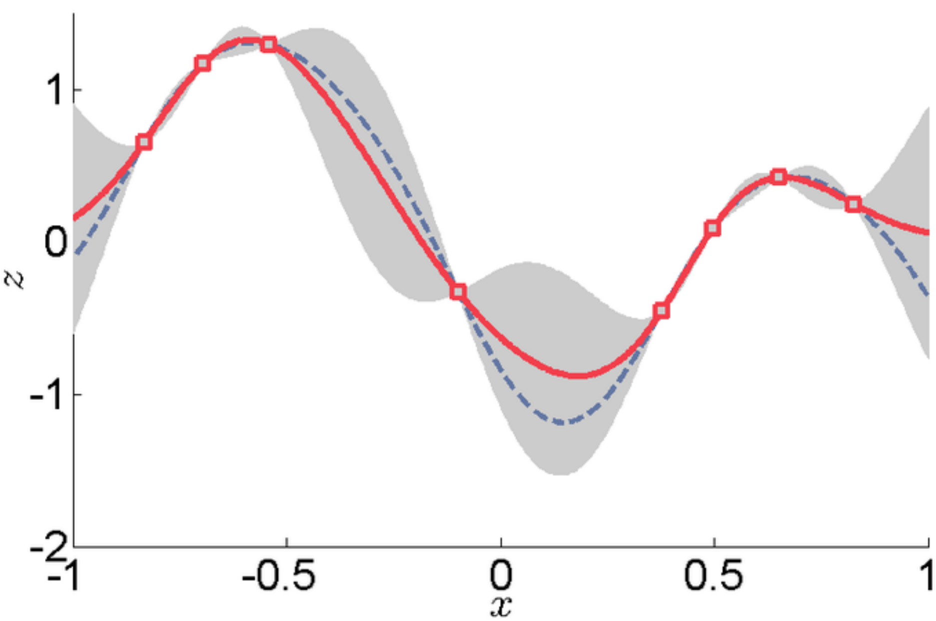
See notebook PolySurrogateModel

# Kriging

Basis functions:

$$\psi^{(i)} = \exp\left(-\sum_j \theta_j |x_j^{(i)} - x_j|^{p_j}\right)$$

Note that this is a Gaussian basis if $p = 2$ and $\theta = 1/\sigma^2$.



Kriging advantages:

- Makes very few assumptions about the shape of the underlying function space.
- It is like a Gaussian basis, in that we can predict not only function values, but also uncertainties.

Disadvantages:

- Because we make few assumptions about underlying shape, generally many more function calls are required to get a reasonable model.
- Often introduces many local minima.

See Kriging Branin example from:

https://optimizationcodes.wordpress.com/chapter-2-constructing-a-surrogate/

# Toolboxes

Matlab: ooDACE, DACE, ...

Python: pyKriging, pyKrige, ...