Surrogate Modeling

AE 6310: Optimization for the Design of Engineered Systems
Spring 2017
Dr. Glenn Lightsey
Lecture Notes Developed By Dr. Brian German

What is a surrogate model?

Let's say we have a typical design analysis function of the form,

$$y = f(\boldsymbol{x})$$

A surrogate model $\hat{f}(\boldsymbol{x})$ is another function that approximates the function $f(\boldsymbol{x})$. That is,

$$f(\boldsymbol{x}) = \hat{f}(\boldsymbol{x}) + \varepsilon(\boldsymbol{x})$$

where $\varepsilon(\boldsymbol{x})$ is the error between the true function and the surrogate model.

Why would we need a surrogate model?

Engineering analysis tools such as CFD, FEA, are computationally expensive.

If we want to run many analyses in a design or optimization study, the computational cost can become prohibitive.

By creating a surrogate model, we may be able to develop a "reasonable enough" approximation of the function to do the study while requiring a fewer overall number of function calls of the computational tool.

Common Uses of Surrogate Models

- Optimization
- Trade space exploration
- Probabilistic studies and robust design
- Transmitting information between different organizations

A well-developed surrogate model is an “investment” that we can use over and over again for the same design space.

What do we need in order to create a surrogate?

- A sample of points $\boldsymbol{x}$ in the design space
- The values of the function $f(\boldsymbol{x})$ at each of the sampled points
- A choice of the functional form $\hat{f}(\boldsymbol{x})$ for the surrogate that depends on certain parameters
- A systematic way to set the parameters in the function $\hat{f}(\boldsymbol{x})$ to "best match" the true function $f(\boldsymbol{x})$

Let's look at each of these ingredients…

Sampling the Design Space

We can sample the design space either ***a priori*** to developing the surrogate, or **adaptively** while developing the surrogate.

Most classical methods rely on an *a priori* sampling.

Approaches for *a priori* sampling have been developed in the field of **designs of experiments (DoE).**

Designs of Experiments

Designs of experiments are "plans" or "stencils" that define where or how to place design points, $\boldsymbol{x}$, within the design space to achieve particular goals.

DoEs were developed initially for true physical experiments. In particular, they emerged in the agricultural field to study ways to improve crop growth.

Because there is variability in physical experiments, DoEs were developed to account for the possibility that the *same* $\boldsymbol{x}$ would produce *different* $f(\boldsymbol{x})$ in repeated trials.

Designs of Computer Experiments

In most of our cases in engineering design, we work with deterministic analysis tools, i.e. each sampled $\boldsymbol{x}$ produces the *same* value $f(\boldsymbol{x})$ in repeated trials.

This difference with traditional DoE has led to the development and use of DoE types for deterministic problems.

Some have termed this field of deterministic experiments for computer models as **Design and Analysis of Computer Experiments (DACE).**

(But we have to be careful… Some people refer to more specific types of DoEs and surrogates when they use the term DACE.)

Types of DoEs

There are many types of DoEs that are used to create surrogate models:

- Full factorial designs
- Fractional factorial designs
- Orthogonal arrays

Structured DoEs

- Latin hypercube designs
- Quasi-Monte Carlo designs

Unstructured or "Space Filling" DoEs

We will discuss DoEs in detail later. For now, just presume that we can select a "good" DoE in order to sample points.

Choice of the Surrogate Functional Form

There are many types of surrogate models that produce different functional forms of $\hat{f}(\boldsymbol{x})$:

- Polynomial response surface equations (RSEs)
- Radial basis functions (RBFs)
- Kriging
- Gaussian processes
- Artificial neural networks (ANNs)
- Support vector machines (SVMs)

There are many surprising similarities between these methods! We will discuss polynomials and RBFs in detail and then briefly discuss some of the other methods.

General Steps in Surrogate Modeling

1. **Sampling** with a DoE, i.e. specifying the points $\boldsymbol{x}$ and determining the corresponding $f(\boldsymbol{x})$ by running the original analysis tools.
2. **Fitting** or **training** of the surrogate models, i.e. determining the unknown parameters in $\hat{f}(\boldsymbol{x})$ to best match $f(\boldsymbol{x})$ at the sampled ("training") points.
3. **Validation** of the surrogate models, i.e. comparing the $\hat{f}(\boldsymbol{x})$ to $f(\boldsymbol{x})$ at *different* $\boldsymbol{x}$ *samples* than those used for training. Based on the results of validation, we may repeats steps 1 and 2 or even select a different functional form for $\hat{f}(\boldsymbol{x})$.
4. **Prediction** with the surrogate models, i.e. using the models for design studies.

Fitting a Surrogate Model

Many forms of surrogate models can be represented by the form,

$$\hat{f}(\boldsymbol{x_j}) = \sum_{i}^{n} w_i \phi_i(\boldsymbol{x_j})$$

Here, ϕ_i are the selected **basis functions,** and w_i are the **weights** that we have to determine by "training" or "fitting" the model. This form of model is linear in the model parameters (weights).

We get to choose the type of basis functions and the number n of them to include in the model (subject to certain constraints we will discuss later).

Fitting a Surrogate Model

With this functional form, we can write,

$$f(\boldsymbol{x_j}) = \hat{f}(\boldsymbol{x_j}) + \varepsilon(\boldsymbol{x_j})$$

$$f(\boldsymbol{x_j}) = \sum_{i}^{n} w_i \phi_i(\boldsymbol{x_j}) + \varepsilon(\boldsymbol{x_j})$$

The training error is then,

$$\varepsilon(\boldsymbol{x_j}) = f(\boldsymbol{x_j}) - \sum_{i}^{n} w_i \phi_i(\boldsymbol{x_j})$$

Fitting a Surrogate Model

$$\varepsilon(\boldsymbol{x_j}) = f(\boldsymbol{x_j}) - \sum_{i}^{n} w_i \phi_i(\boldsymbol{x_j})$$

We can write this in vector form as

$$\boldsymbol{\varepsilon} = \boldsymbol{f} - \Phi \boldsymbol{w}$$

where $\boldsymbol{\varepsilon}$ and $\boldsymbol{f}$ are $m \times 1$ vectors, $\boldsymbol{w}$ is an $n \times 1$ vector, and Φ is an $m \times n$ matrix whose columns are $\phi_i(\boldsymbol{x_j})$.

Fitting a Surrogate Model

To fit a surrogate model, we typically try to **minimize the sum-squared error (SSE)** with respect to the training data by changing the weights, $\boldsymbol{w}$:

$$\min_{\boldsymbol{w}} \ \boldsymbol{\varepsilon}^T \boldsymbol{\varepsilon}$$

Since this is an unconstrained problem, the optimality condition is,

$$\frac{d}{d\boldsymbol{w}} (\boldsymbol{\varepsilon}^T \boldsymbol{\varepsilon}) = \mathbf{0}$$

This approach gives the **least squares** solution.

Fitting a Surrogate Model

$$\frac{d}{d\boldsymbol{w}}(\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon}) = \mathbf{0}$$

or,

$$\frac{d}{d\boldsymbol{w}}(\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon}) = 2\left(\frac{d\boldsymbol{\varepsilon}}{d\boldsymbol{w}}\right)^{\mathrm{T}}\boldsymbol{\varepsilon} = \mathbf{0}$$

Note that $\frac{d\boldsymbol{\varepsilon}}{d\boldsymbol{w}}$ is $m \times n$ and $\boldsymbol{\varepsilon}$ is $m \times 1$.

Fitting a Surrogate Model

$$2\left(\frac{d\boldsymbol{\varepsilon}}{d\boldsymbol{w}}\right)^{\mathrm{T}}\boldsymbol{\varepsilon} = \mathbf{0}$$

But, $\boldsymbol{\varepsilon} = \boldsymbol{f} - \Phi\boldsymbol{w}$, so,

$$\frac{d\boldsymbol{\varepsilon}}{d\boldsymbol{w}} = \frac{d}{d\boldsymbol{w}}(\boldsymbol{f} - \Phi\boldsymbol{w}) = -\Phi$$

and we have,

$$-2\Phi^{\mathrm{T}}\boldsymbol{\varepsilon} = -2\Phi^{\mathrm{T}}(\boldsymbol{f} - \Phi\boldsymbol{w}) = 0$$

Fitting a Surrogate Model

$$-2\Phi^{\mathrm{T}}\boldsymbol{\varepsilon} = -2\Phi^{\mathrm{T}}(\boldsymbol{f} - \Phi\boldsymbol{w}) = 0$$

This expression simplifies to,

$$\Phi^{\mathrm{T}}\Phi\boldsymbol{w} = \Phi^{\mathrm{T}}\boldsymbol{f}$$

In principle, we can now just invert $\Phi^{\mathrm{T}}\Phi$ (which is $n \times n$) to solve for $\boldsymbol{w}$:

$$\boldsymbol{w} = \left(\Phi^{\mathrm{T}}\Phi\right)^{-1}\Phi^{\mathrm{T}}\boldsymbol{f}$$

The resulting $\boldsymbol{w}$ provides the model fit. This approach works for many types of surrogate models.

Implications of m and n

If the rows of Φ are linearly independent (e.g., no repeated fit points) and the columns, i.e. basis functions, are linearly independent, then:

- If $m > n$, the problem is overdetermined, and the resulting $\hat{f}$ is the least squares fit through the training points.
- If $m = n$, the problem is perfectly determined, and the model goes through all training points, i.e. the model is an **interpolant.**
- If $m < n$, the problem is underdetermined, and $\Phi^{\mathrm{T}}\Phi$ is rank deficient (singular), so $\left(\Phi^{\mathrm{T}}\Phi\right)^{-1}$ does not exist and we cannot fit a model with the ordinary least squares regression.

Polynomial Surrogates

For a polynomial surrogate for a 1-D problem, we select the polynomial basis,

$$\{x^0, x^1, x^2, x^3, \ldots x^{n-1}\}$$

and form the Φ matrix as,

$$\Phi = \begin{bmatrix} x_1^0 & \cdots & x_1^{n-1} \\ \vdots & \ddots & \vdots \\ x_m^0 & \cdots & x_m^{n-1} \end{bmatrix}$$

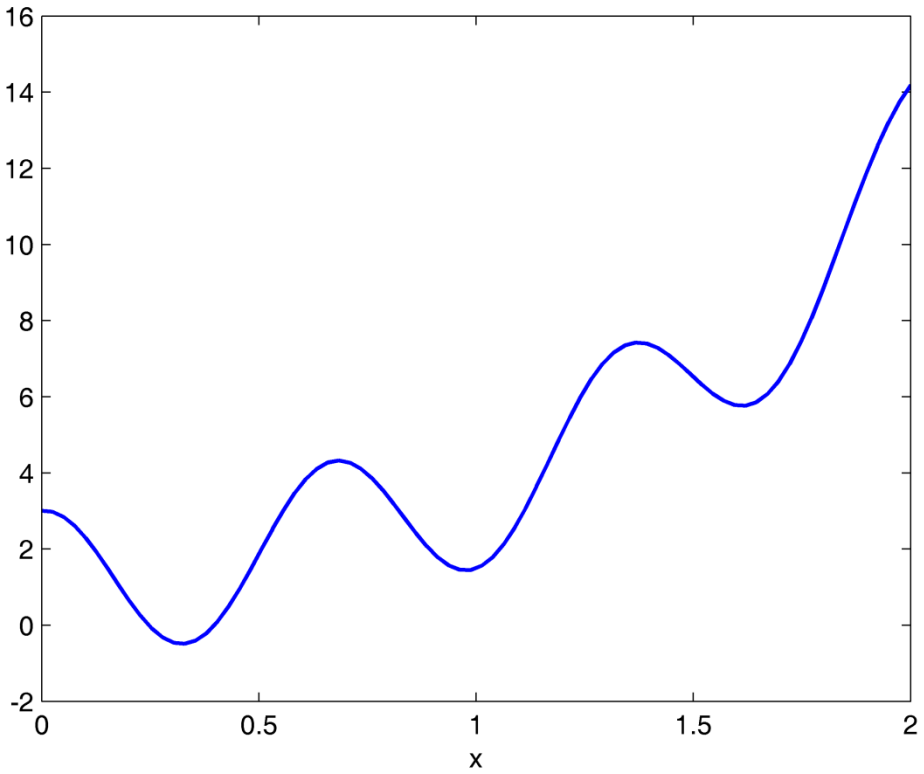
where m is the number of training points.

Polynomial Surrogates: Example

Consider the function,

$$y = \exp\left(\frac{5x}{4}\right) + 2\cos(3\pi x)$$

over the range $0 \leq x \leq 2$.



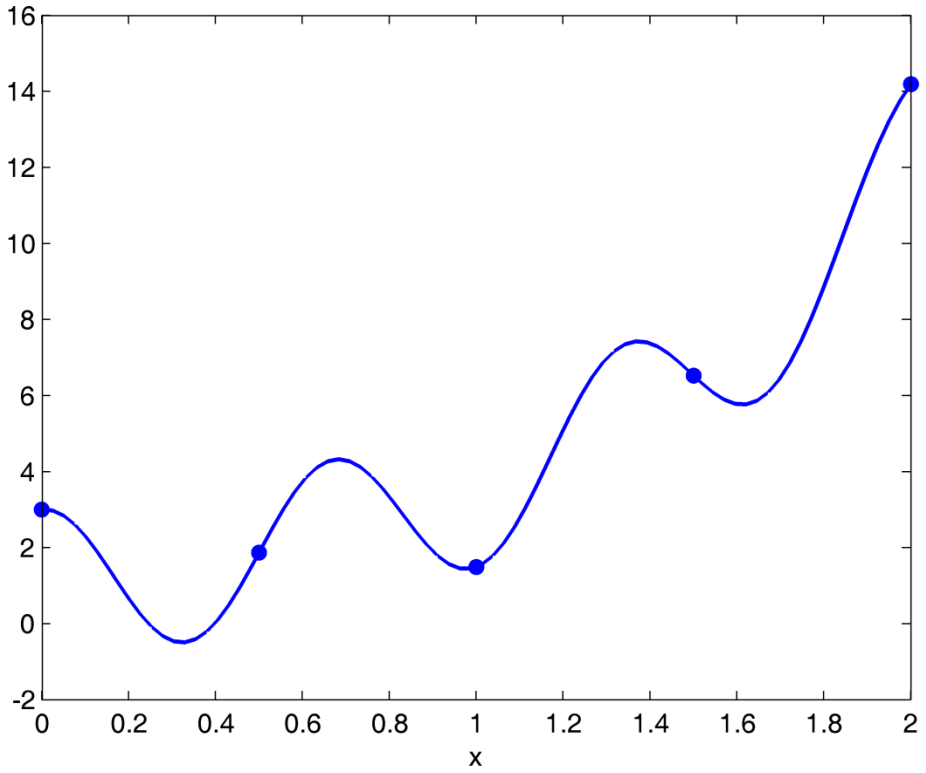
Polynomial Surrogates: Example

Let's sample 5 evenly spaced points:
$x = \{0.0, 0.5, 1.0, 1.5, 2.0\}$

Let's fit a constant model $(n = 1)$ to these 5 points.

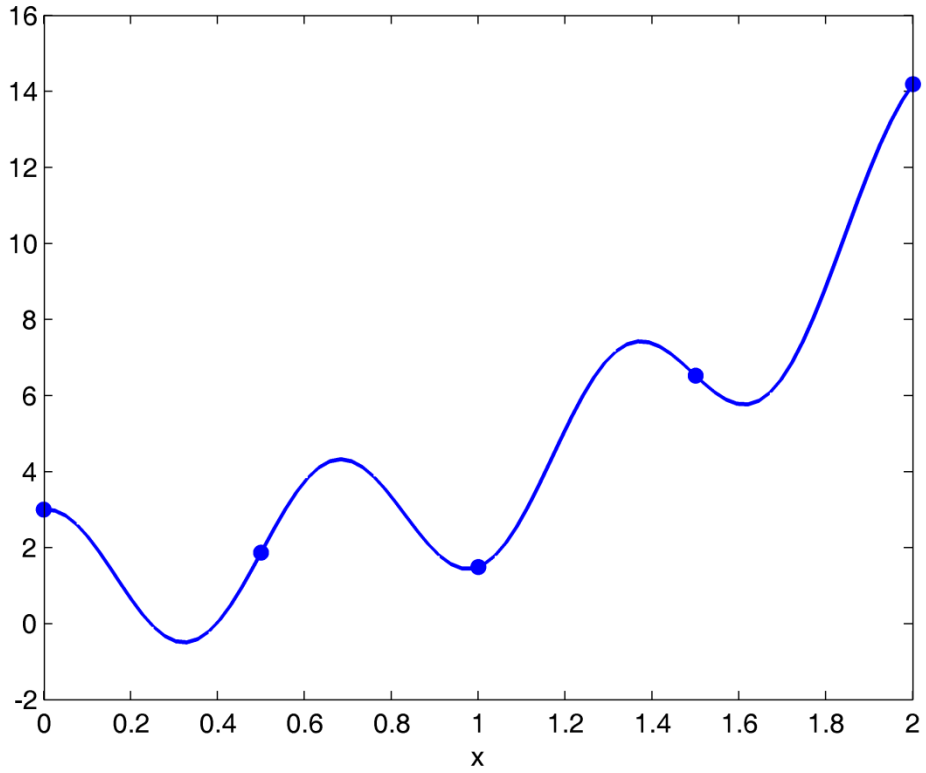
The Φ matrix is then,

$$\Phi = \begin{bmatrix} x_1^0 \\ \vdots \\ x_5^0 \end{bmatrix} = \begin{bmatrix} 0^0 \\ \vdots \\ 2^0 \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$



Polynomial Surrogates: Example

Next, we evaluate the y values at each of the 5 training points to form the vector $\boldsymbol{f}$.



We now fit the model,

$$\boldsymbol{w} = \left(\Phi^{\mathrm{T}}\Phi\right)^{-1}\Phi^{\mathrm{T}}\boldsymbol{f}$$

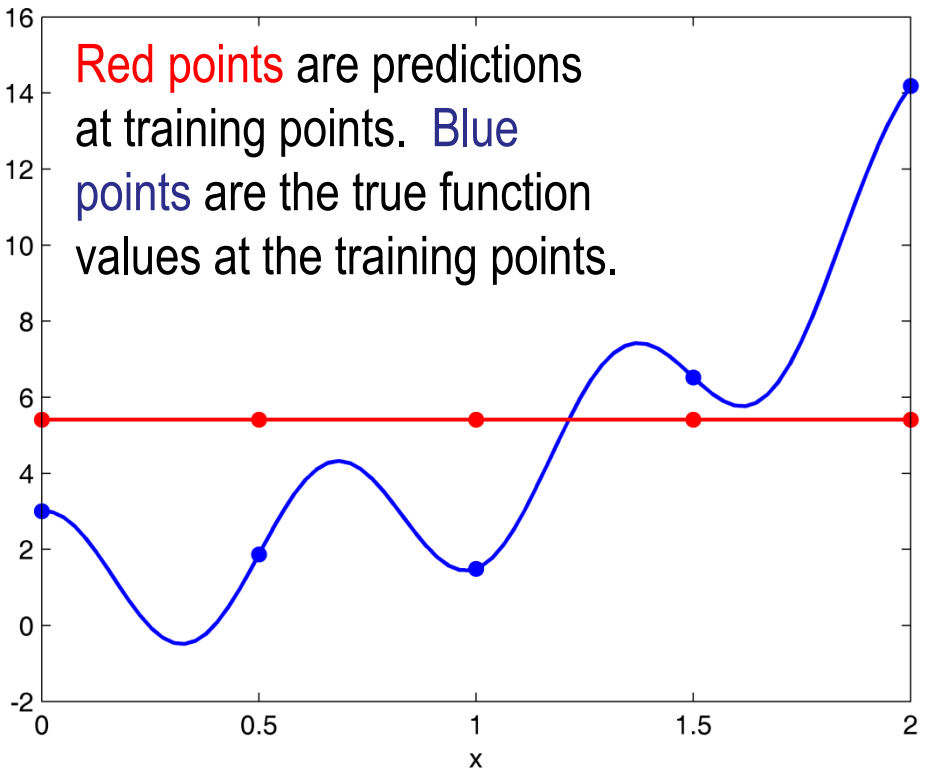
and in this case, $\boldsymbol{w}$ has one element which is found to be $w = 5.41$.

Polynomial Surrogates: Example

To apply the model for prediction, we form a new vector of x samples and a new Φ matrix.

We then find $\hat{y}$ as,

$$\hat{y} = \Phi \boldsymbol{w}$$



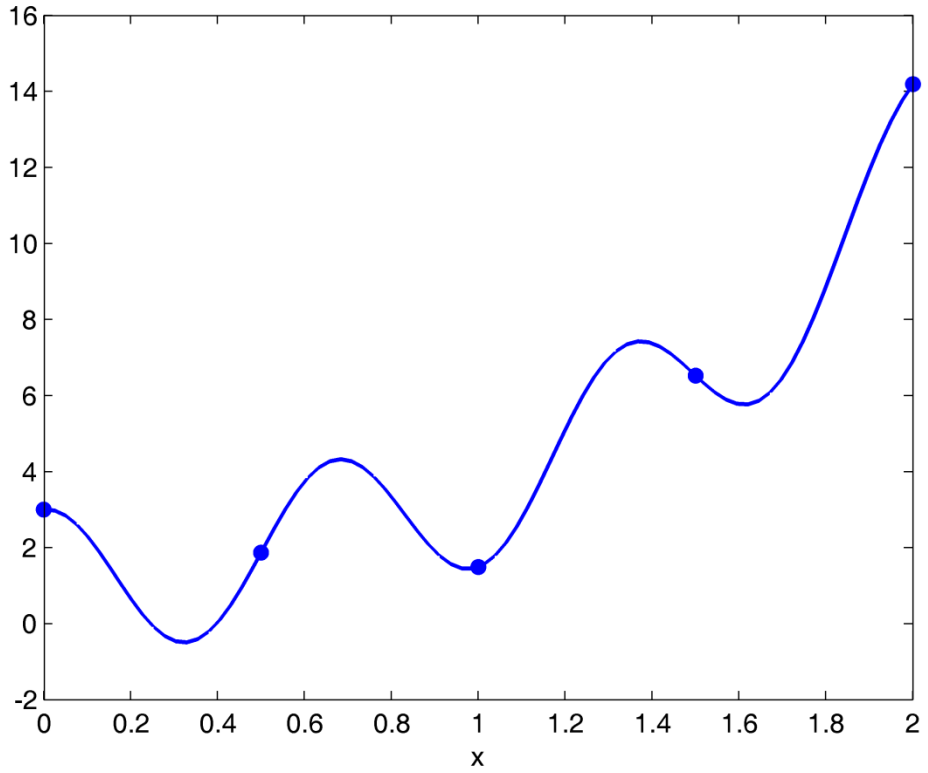
In this case, the model predicts a constant value (the mean value of the training points) at each x.

Polynomial Surrogates: Example

OK, let's now fit a linear model ($n = 2$) to the same training points.

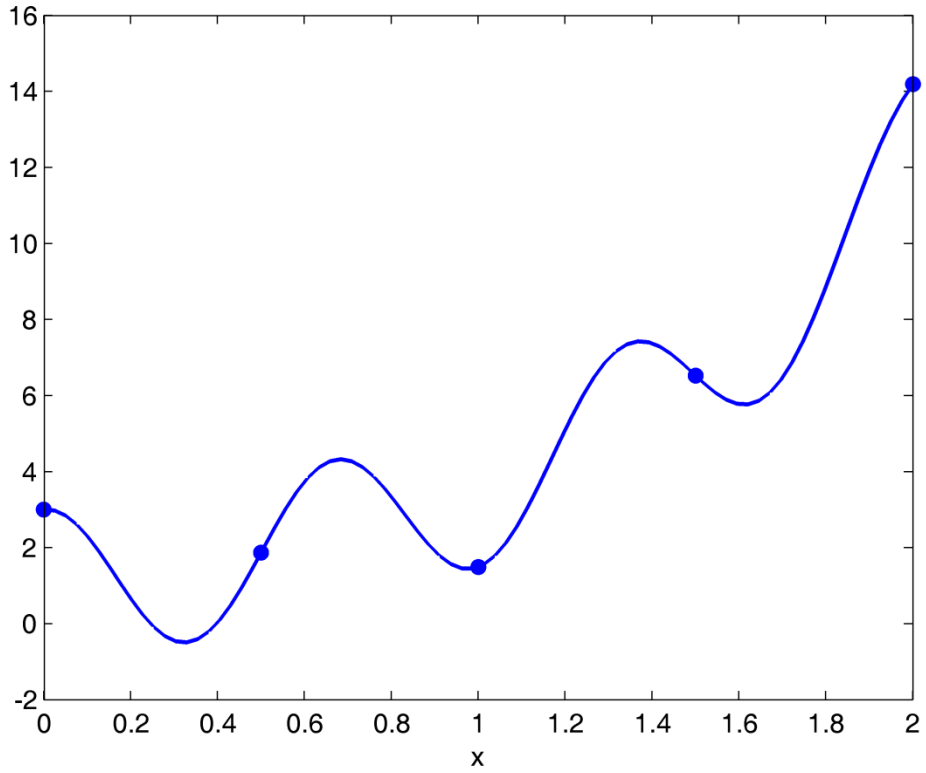
The Φ matrix is,

$$\Phi = \begin{bmatrix} x_1^0 & x_1^1 \\ \vdots & \vdots \\ x_5^0 & x_5^1 \end{bmatrix} = \begin{bmatrix} 0^0 & 0^1 \\ \vdots & \vdots \\ 2^0 & 2^1 \end{bmatrix}$$



Polynomial Surrogates: Example

Next, we evaluate the y values at each of the 5 training points to form the vector $\boldsymbol{f}$.



We now fit the model,

$$\boldsymbol{w} = \left(\Phi^{\mathrm{T}}\Phi\right)^{-1}\Phi^{\mathrm{T}}\boldsymbol{f}$$

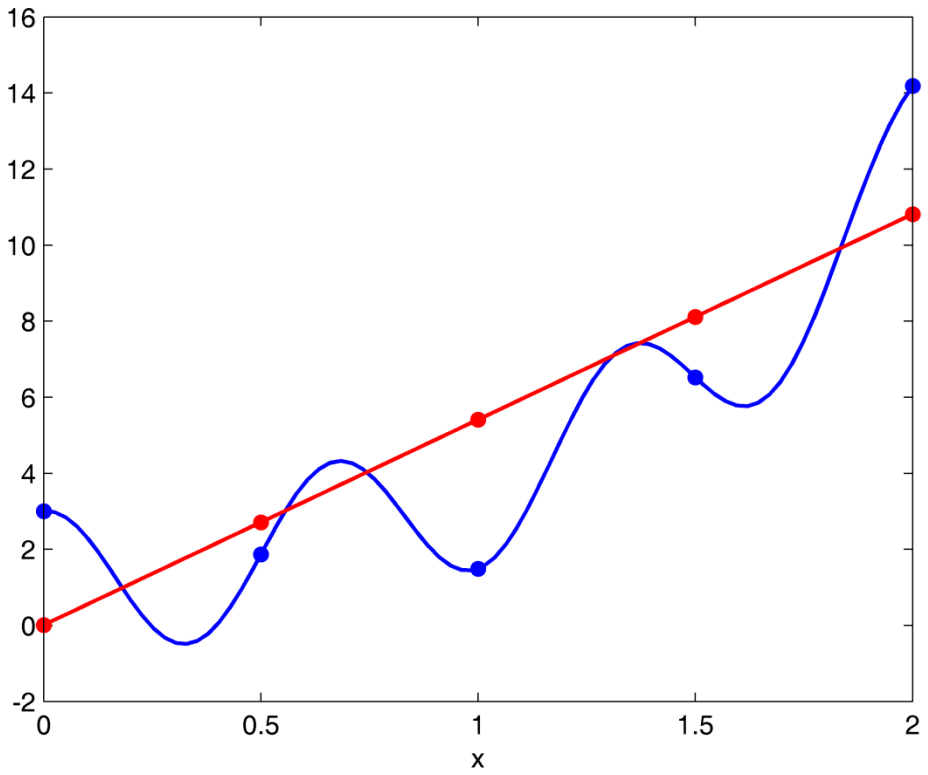
and in this case, $\boldsymbol{w}$ has two elements which are found to be $\boldsymbol{w} = [0.0089.\, 5.4035]^{\mathrm{T}}$

Polynomial Surrogates: Example

To apply the model for prediction, we form a new vector of x samples and a new Φ matrix.

We then find $\hat{y}$ as,

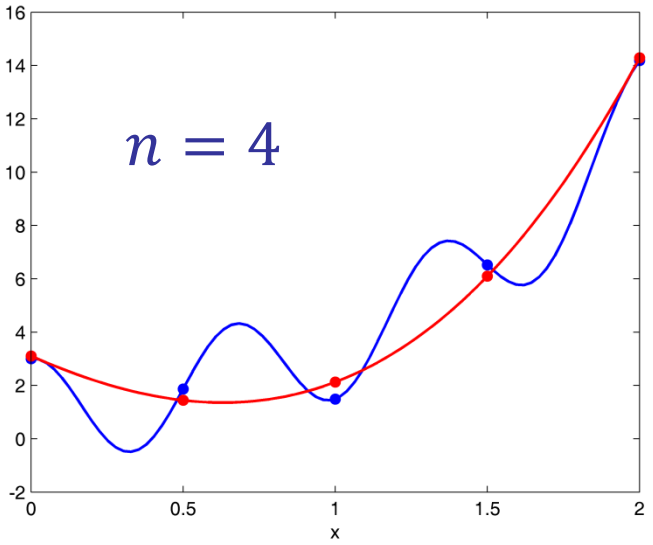
$$\hat{y} = \Phi \boldsymbol{w}$$

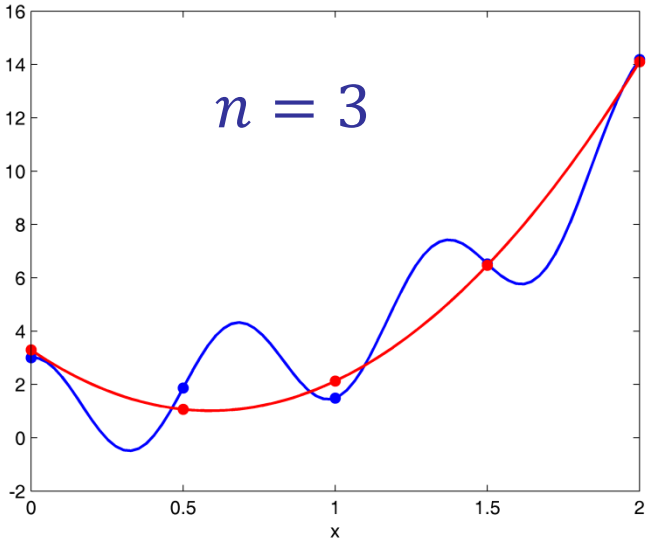


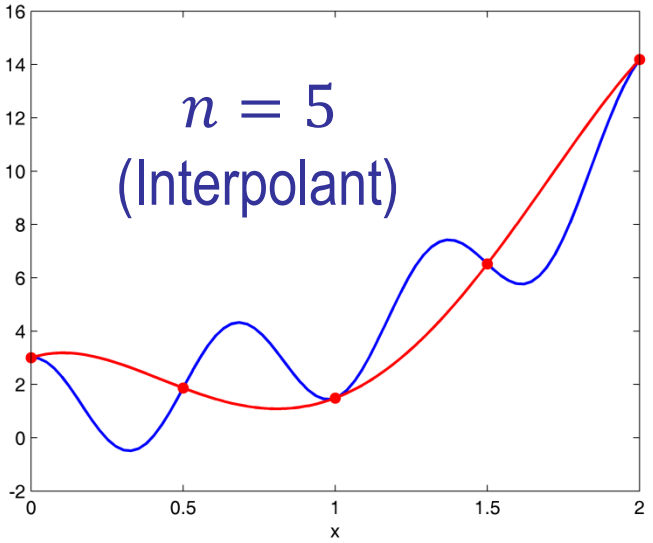
In this case, the model predicts a linear trend as we should expect.

Polynomial Surrogates: Example

We can keep increasing the order of the polynomial basis to improve the fit of the model to the 5 training points.







Overfitting

The models that we examined on the previous slide were all rather "reasonable" at capturing the *overall* variability of the function.

The models were at least smooth and changed slowly between training points, even if they did not predict the true values of the function. (Note that even the interpolant did not predict the true value of the function away from the training points.)

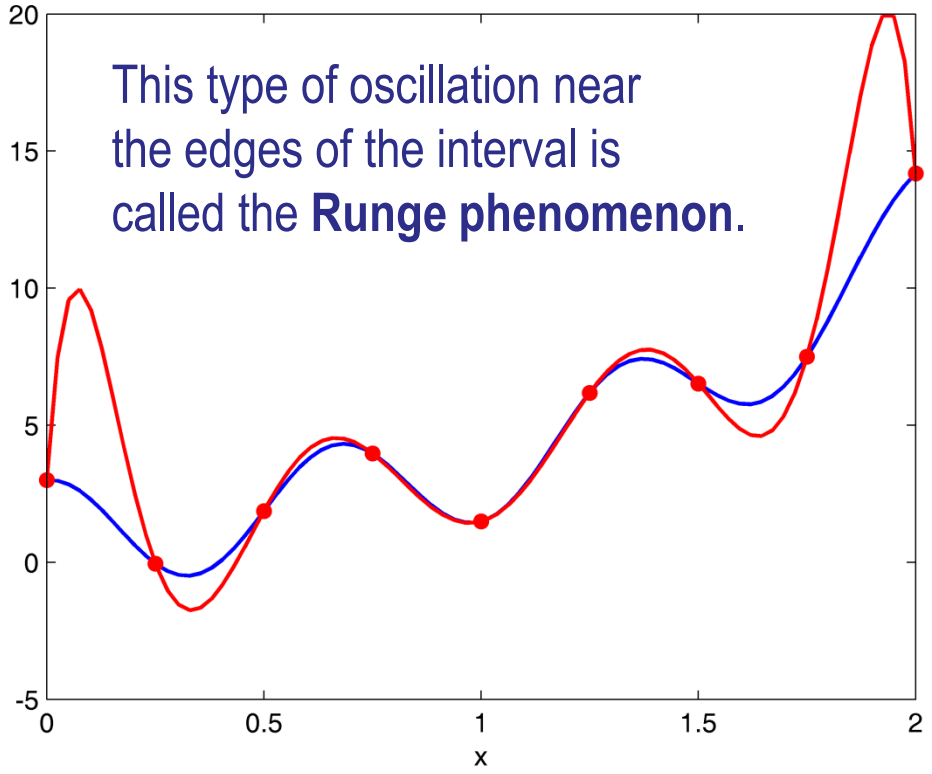
But this does not always happen. As we increase the number of basis functions, our model becomes susceptible to **overfitting.**

Overfitting

Let's try our same example with $n = m = 9$, a higher-order interpolating model. Here's what happens:

Notice how the model "overshoots" at low and high values of x.

This is a behavior indicative over overfitting.

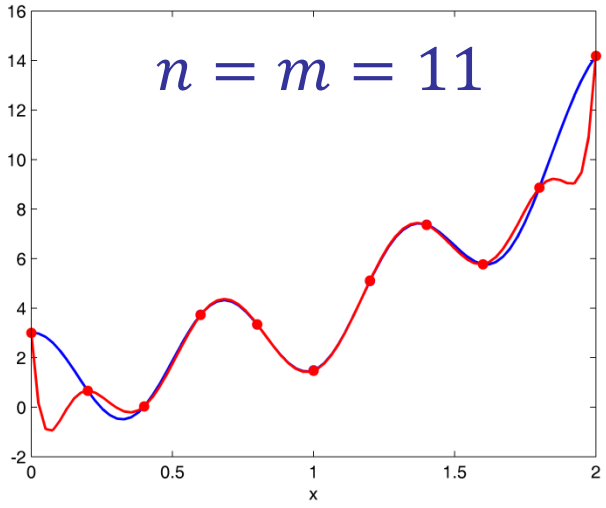


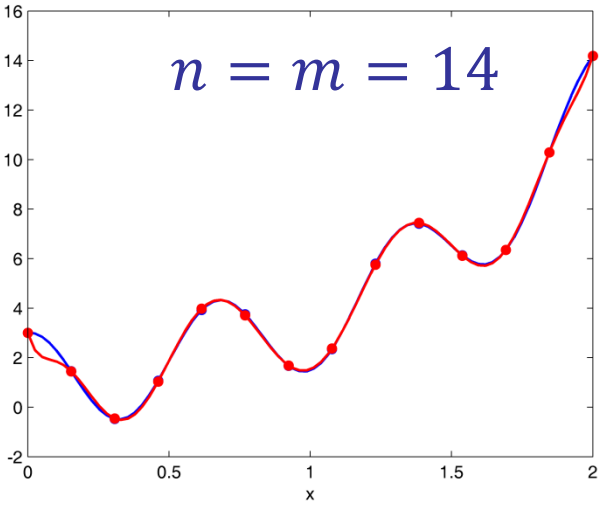
An overfit model cannot be trusted as "reasonable" at points other than the training points

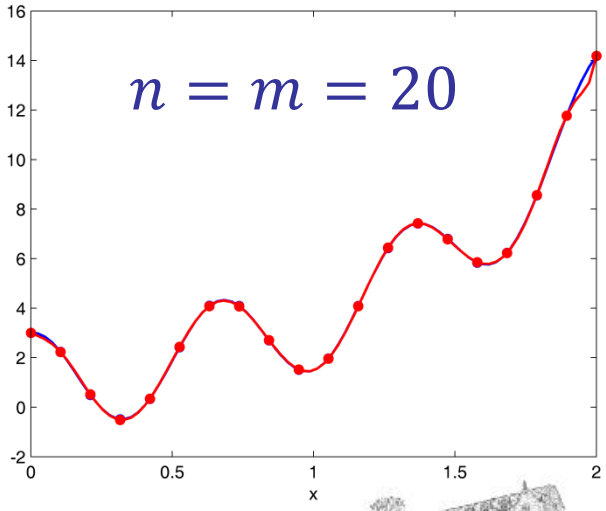
Overfitting

You may be thinking, "But if we kept increasing m and n, shouldn't the model just get better?"

In general, yes, but we never know when overfitting will start/stop being a problem.



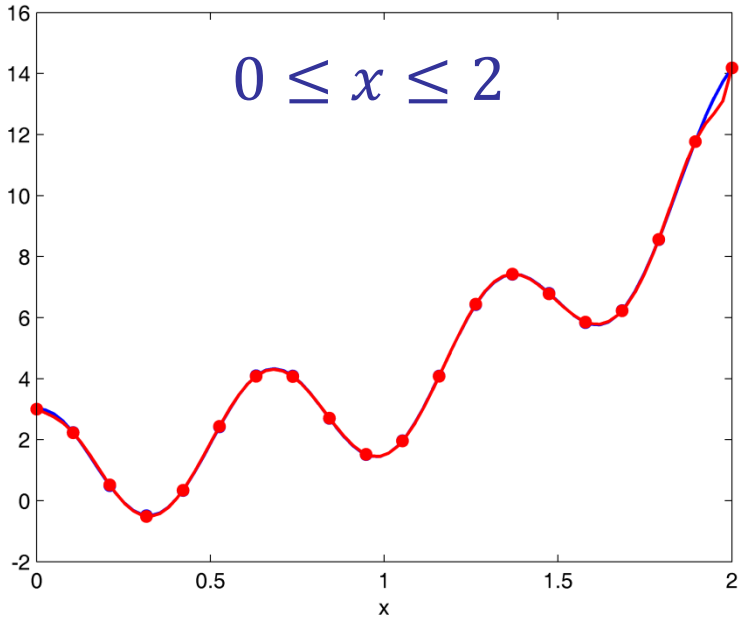


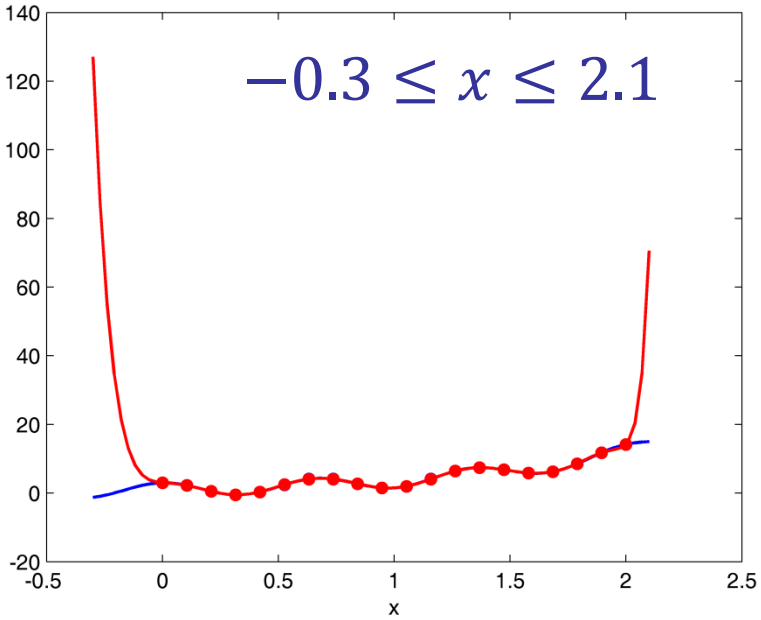


Extrapolation

So, $n = m = 20$ looks like a pretty great model, right?

Maybe we could try to use it over a wider range of x…





Rarely is extrapolating a surrogate model a good idea.

Measuring Quality of the Fit

It would be nice to have a measure of the quality of our model fit.

One measure is the sum squared error itself:

$$SSE = \boldsymbol{\varepsilon}^T \boldsymbol{\varepsilon} = (\boldsymbol{y} - \hat{\boldsymbol{y}})^{\mathrm{T}} (\boldsymbol{y} - \hat{\boldsymbol{y}})$$

Recall that the $\boldsymbol{\varepsilon}$ and $\boldsymbol{y}$ vectors are $m \times 1$ (one element per sample $\boldsymbol{x}$).

SSE therefore has units/scale of the square of the units/scale of the function y.

Measuring Quality of the Fit

It is often preferable to have a "unitless" metric of fit quality that always has the same scale.

The **coefficient of determination, R^2,** is such a metric.

R^2 is defined as,

$$R^2 = 1 - \frac{SSE}{SST}$$

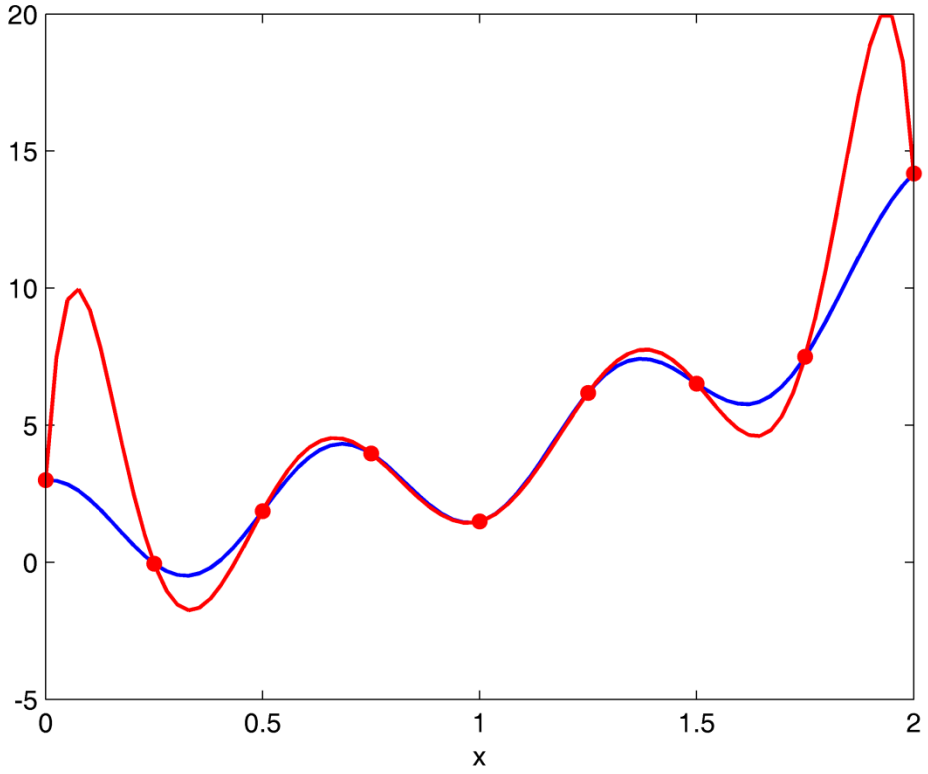
where SST is the total sum squares: $SST = (\boldsymbol{y} - \overline{\boldsymbol{y}})^{\mathrm{T}}(\boldsymbol{y} - \overline{\boldsymbol{y}})$ and $\overline{\boldsymbol{y}}$ is the mean value of $\boldsymbol{y}$.

Validating the Model

As we saw with overfitting, sometimes the model does well at the fitting points (maybe even interpolating them), but does not do well at other points.

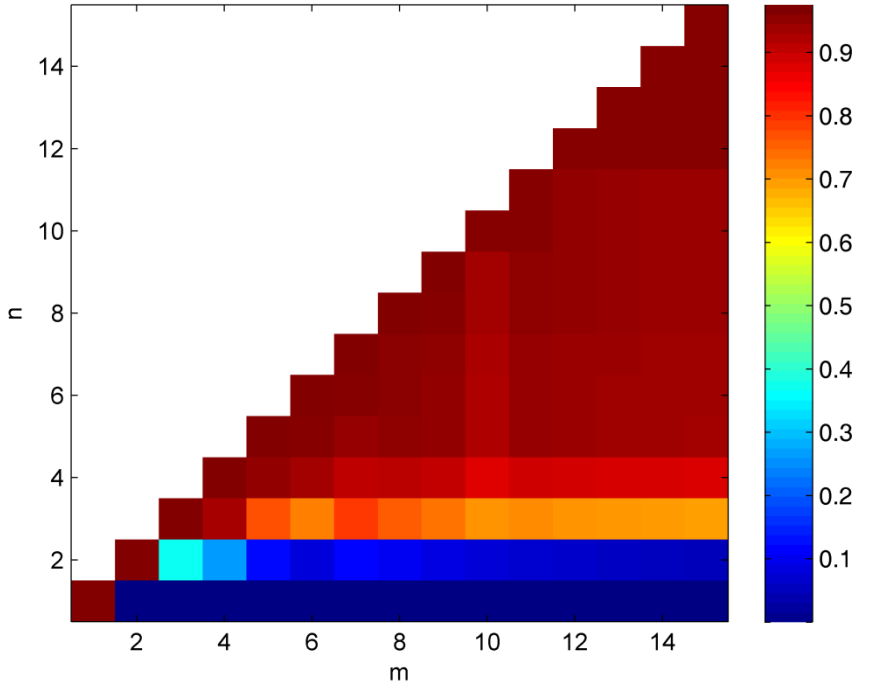
We therefore need to "validate" the model at *other* points than those we used to train it.

This is typically done by sampling the true function at additional points and then computing SSE and R^2 of the model prediction.

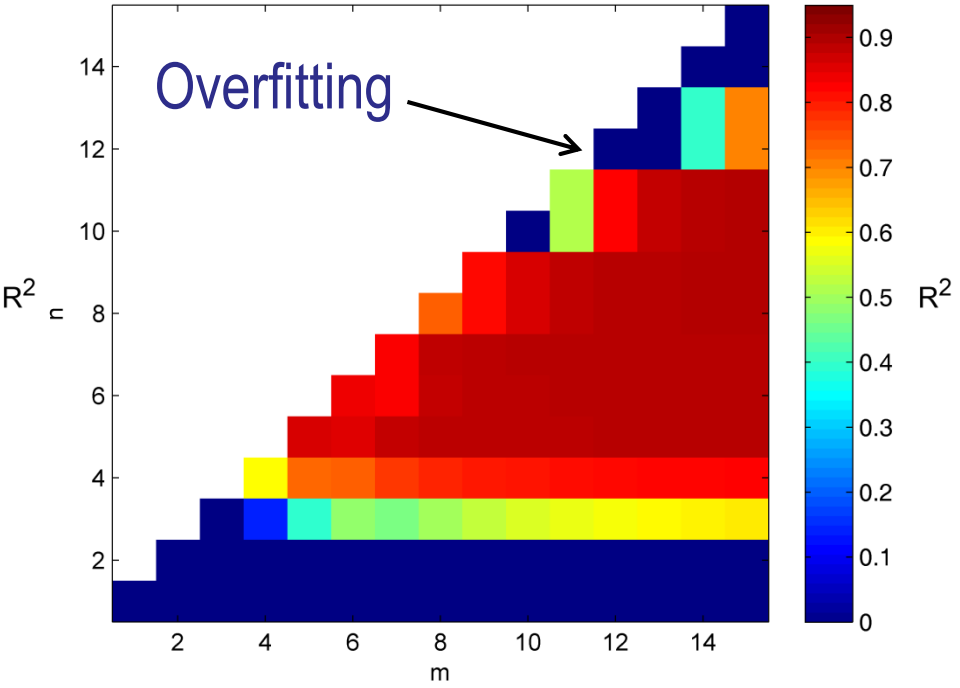


Validating the Model

R^2 to Fit Data

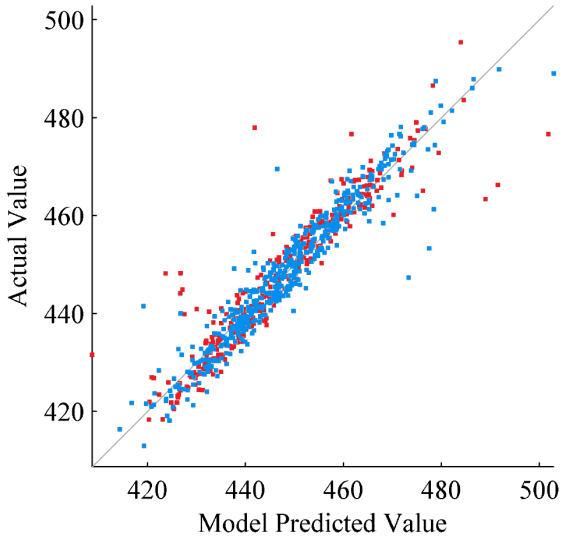


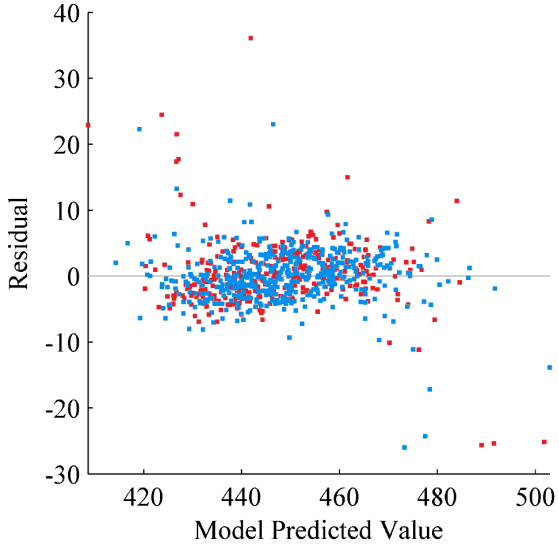
R^2 to Validation Data

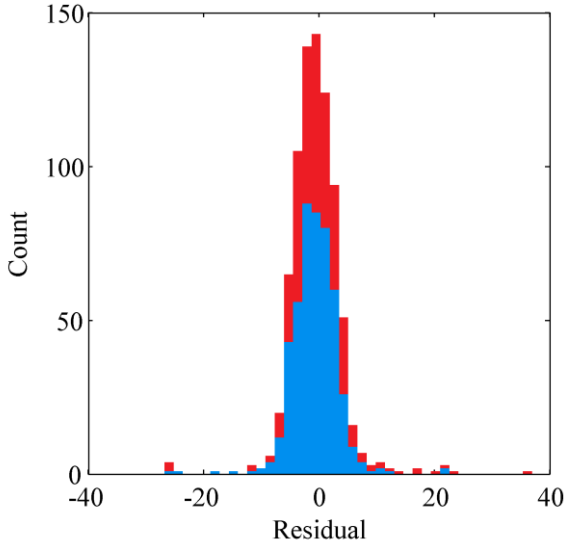


Polynomial basis fit to $y = 2\cos\left(\frac{3\pi x}{2}\right) - 3x + 2\exp\left(\frac{3x}{2}\right) - 3.5$

Validating the Model

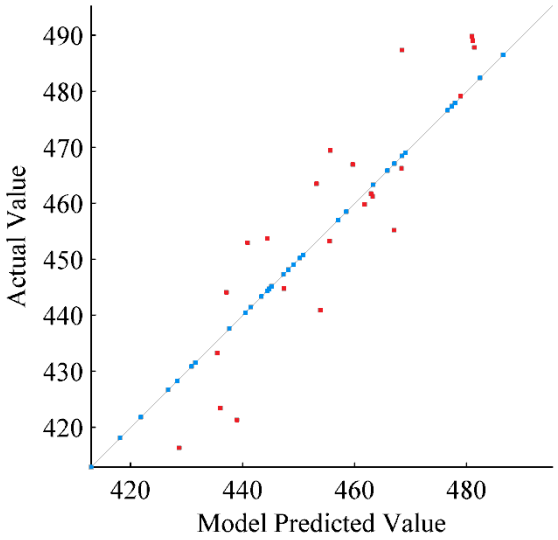


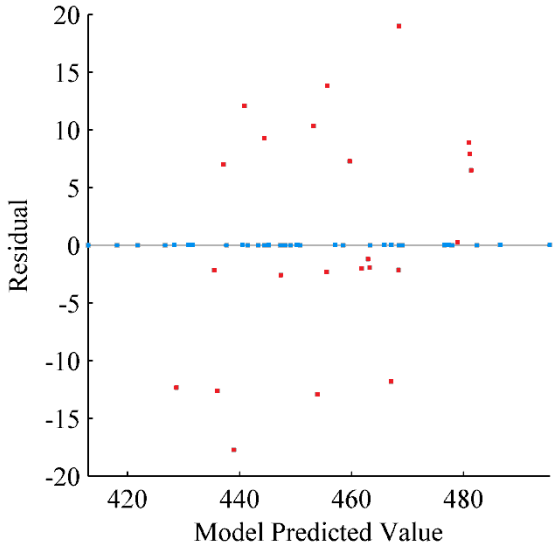


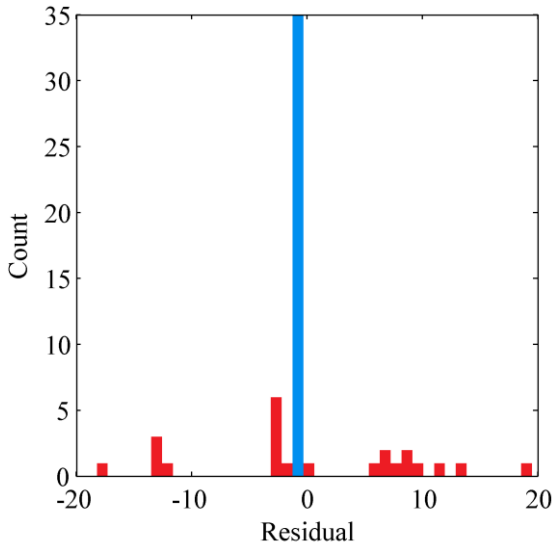


A rather bad fit

Validating the Model







A rather good fit

Regularization

A challenge in surrogate modeling is to determine which n basis functions to include for a given budget of samples, m.

For example, consider the following function:

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

It is unnecessary to include the basis function x^2 in the Φ matrix. If we did, it would "waste" at least one additional sampled design.

However, in general, we cannot know the functional form *a priori*, so we have to keep increasing m and n enough to capture the highest-order effect expected in the true function.

Regularization

Regularization is an approach that can allow us to solve problems with $m < n$.

This approach works, in general, when the true function depends on only m or fewer of the $n > m$ basis functions.

In this case, we expect that the vector $\boldsymbol{w}$ of weights either has a low norm or is ***sparse***, i.e. it has many elements that are zero.

Regularization

To implement regularization, we need to modify the optimization problem that we solve to determine the weights.

Recall the original optimization problem,

$$\min_{\boldsymbol{w}} \ \boldsymbol{\varepsilon}^T \boldsymbol{\varepsilon}$$

With regularization, we add a term to the objective function to *minimize a norm of the weights vector,* $\boldsymbol{w}$.

Regularization

For example, in **ridge regression**, the objective becomes,

$$\min_{\boldsymbol{w}} \quad \boldsymbol{\varepsilon}^T \boldsymbol{\varepsilon} + \lambda \boldsymbol{w}^T \boldsymbol{w}$$

where $\lambda > 0$ is called the "ridge parameter."

The idea is to make the elements of $\boldsymbol{w}$ as small as possible (in the sense of the 2-norm) while still minimizing the elements of the error vector $\boldsymbol{\varepsilon}$. We control which term in the objective is dominant by choosing λ.

Regularization

We can find the solution as,

$$\frac{d}{d\boldsymbol{w}}\left(\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon} + \lambda \boldsymbol{w}^T\boldsymbol{w}\right) = \mathbf{0}$$

$$2\left(\frac{d\boldsymbol{\varepsilon}}{d\boldsymbol{w}}\right)^T \boldsymbol{\varepsilon} + 2\lambda\boldsymbol{w} = \mathbf{0}$$

$$-2\Phi^T\boldsymbol{\varepsilon} + 2\lambda\boldsymbol{w} = \mathbf{0}$$

Regularization

$$-2\Phi^T \boldsymbol{\varepsilon} + 2\lambda \boldsymbol{w} = \mathbf{0}$$

$$\Phi^T (\boldsymbol{f} - \Phi \boldsymbol{w}) - \lambda \boldsymbol{w} = \mathbf{0}$$

$$(\Phi^T \Phi + \lambda I)\boldsymbol{w} = \Phi^T \boldsymbol{f}$$

$$\boldsymbol{w} = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T \boldsymbol{f}$$

Regularization

$$\boldsymbol{w} = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T\boldsymbol{f}$$

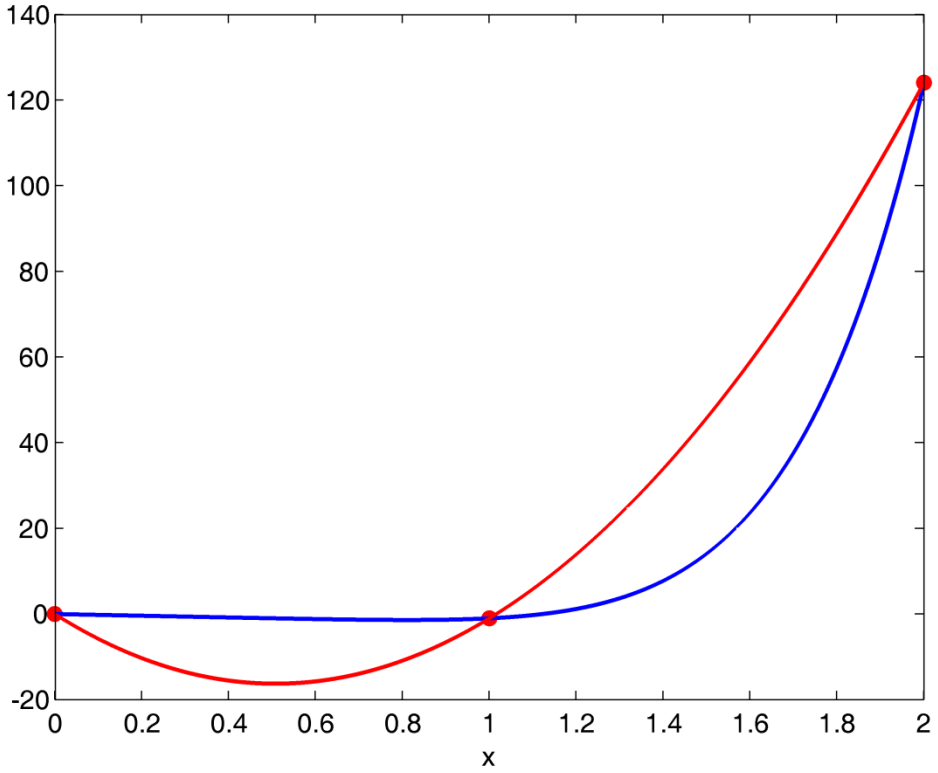
The new formulation results in a "ridge" along the diagonal of the basis matrix that must be inverted.

It can be shown that the matrix $\Phi^T\Phi + \lambda I$ is **always** invertible, regardless of m and n.

Regularization: Example

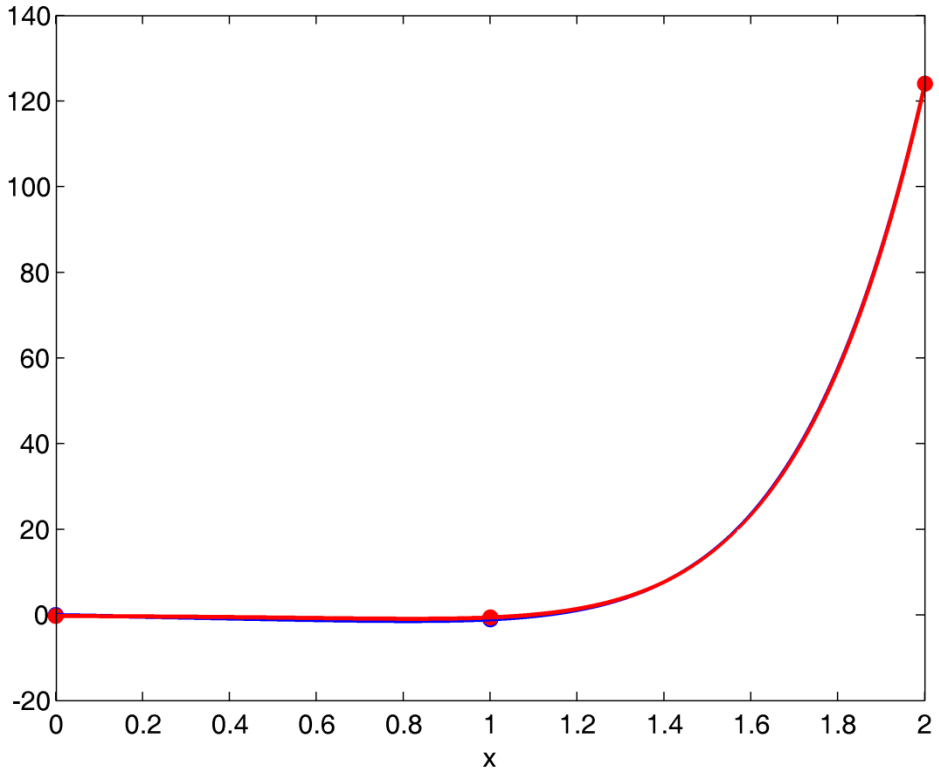
Consider the function $y = -2x + x^7$.

Here's the fit with ordinary least squares with $m = 3, n = 3$:



Regularization: Example

And here's the fit with ridge regression with $m = 3, n = 8, \lambda = 1$:



$\boldsymbol{w}$

-0.2154
-0.4200
-0.3984
-0.3551
-0.2685
-0.0954
0.2509
0.9435

Effects of x term spread out to minimize 2-norm of $\boldsymbol{w}$

x^7 term detected but some spread to x^6

Multidimensional Polynomial Fits

So far, we have looked at fitting only polynomial curves, i.e. the response y vs. a single dimension x with **simple linear regression** and ridge regression.

More generally, we can fit a model $y = f(\boldsymbol{x})$ in which $\boldsymbol{x}$ is a vector by using **multiple linear regression**.

The main difference is just what basis functions we use in the Φ matrix.

Multivariate Polynomial Fits

For multidimensional polynomial surrogates, we just modify the basis that forms the Φ matrix to include all of the multivariate monomials that we want.

So, for example, a valid polynomial basis is,

$$\{1, x_1, x_2, x_1^2, x_1 x_2, x_2^2\}$$

This is the maximal or "full" basis for a set of polynomial of maximum order 2 in 2 variables.

Multivariate Polynomial Fits

It is often impractical to use "full" high-degree polynomial bases in multiple variables.

Presume that we consider a polynomial basis with p variables (p elements of $\boldsymbol{x}$) of degree q.

There are $\binom{q+p-1}{q}$ monomials of degree q.

There are

$$\sum_{q=0}^{q_{max}} \binom{q+p-1}{q}$$

monomials up to and including degree q_{max}.

Multivariate Polynomial Fits

Total number of terms for a "full" polynomial basis in p variables with maximum degree q_{max}

$p = 1,$ $q_{max} = 1$

$p = 8,$ $q_{max} = 1$

2	3	4	5	6	7	8	9
3	6	10	15	21	28	36	45
4	10	20	35	56	84	120	165
5	15	35	70	126	210	330	495
6	21	56	126	252	462	792	1287
7	28	84	210	462	924	1716	3003
8	36	120	330	792	1716	3432	6435
9	45	165	495	1287	3003	6435	12870

$p = 1,$ $q_{max} = 8$

$p = 8,$ $q_{max} = 8$

Multivariate Polynomial Fits

At small p and q_{max}, this situation appears not so bad.

However, with expensive analyses, too many samples, m, may be required, even for rather small p and q_{max}.

This issue has resulted in considerable work in determining good "partial" polynomial basis models for use in model fitting.

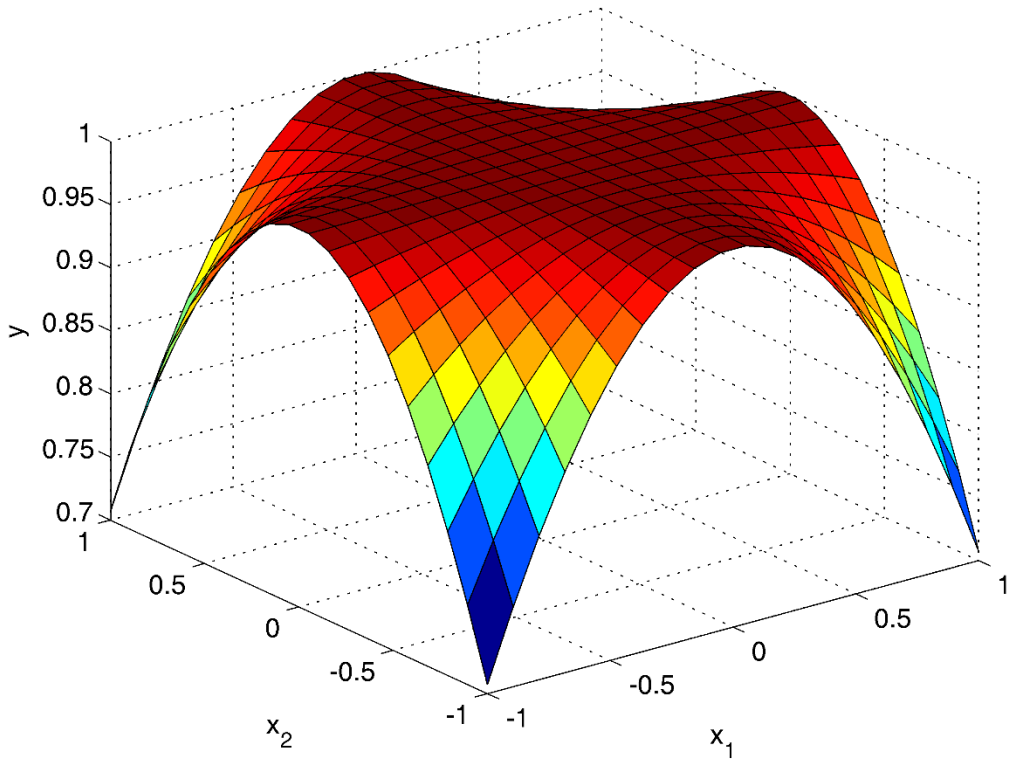
For example, $\{1, x_1, x_2, x_1 x_2\}$ is a common basis for a 2nd order "interaction model."

We will discuss these issues more in the context of DoEs.

Multivariate Polynomial Fits: Example

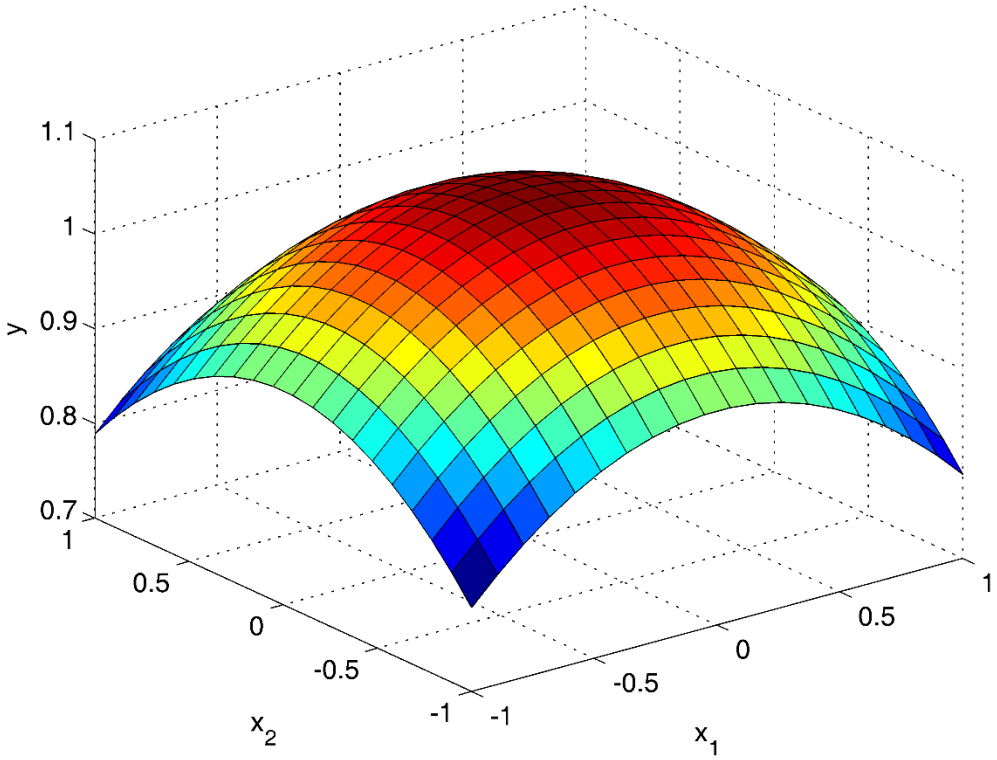
Let's look at an example. Consider the function

$$y = \sqrt{\cos(x_1 x_2)},\ -1 \le x_1 \le 1, -1 \le x_2 \le 1$$



Multivariate Polynomial Fits: Example

Fit to a 6×6 grid of evenly spaced points to the basis $\{1, x_1, x_2, x_1x_2, x_1^2, x_2^2\}$



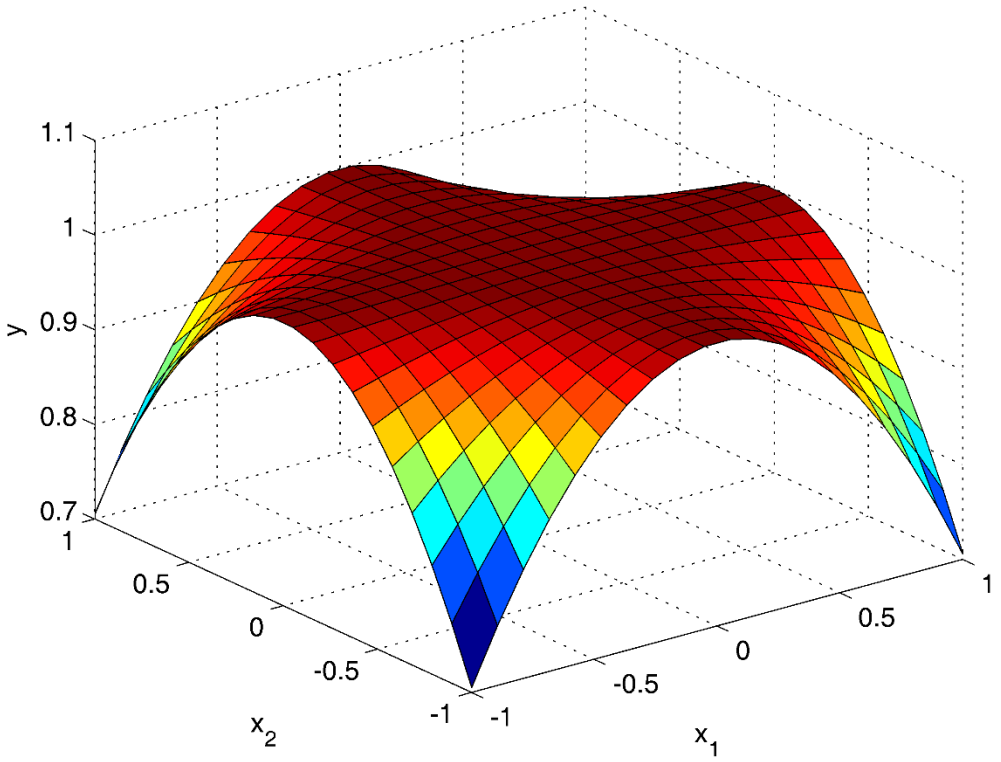
$R^2_{fit} = 0.7383$

$R^2_{val} = 0.6737$

Multivariate Polynomial Fits: Example

Fit to a 10×10 grid of evenly spaced points to the basis

$$\{1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3, x_1^4, x_1^3x_2, x_1^2x_2^2, x_1x_2^3, x_2^4\}$$



$R^2_{fit} = 0.9999$

$R^2_{val} = 0.9999$

Radial Basis Functions

So far, we have primarily discussed polynomial surrogate models.

Another class of surrogate model is called **radial basis functions (RBFs).**

As their name implies RBFs are based on basis functions that depend only on the radius between any two training points.

Typically the 2-norm (Euclidean distance) is used to define the radius:

$$r = \|\boldsymbol{x_j} - \boldsymbol{x_i}\| = \sqrt{\left(x_{j,1} - x_{i,1}\right)^2 + \cdots + \left(x_{j,k} - x_{i,k}\right)^2}$$

Radial Basis Functions

The form of an RBF surrogate model is,

$$\hat{f}(\boldsymbol{x_j}) = \sum_{i}^{n} w_i \phi(\|\boldsymbol{x_j} - \boldsymbol{x_i}\|)$$

RBFs are constructed as interpolating models with the number of training points equaling the number of basis functions $(m = n)$.

The basis functions ϕ can be formed into an $n \times n$ matrix Φ.

Notice that we described the basis function as ϕ instead of ϕ_i. This means that unlike polynomial regression, we use the same *form* of basis function for each column in Φ.

Radial Basis Functions

The Φ matrix has the form,

$$\Phi = \begin{bmatrix} \phi(\|\boldsymbol{x_1} - \boldsymbol{x_1}\|) & \cdots & \phi(\|\boldsymbol{x_1} - \boldsymbol{x_n}\|) \\ \vdots & \ddots & \vdots \\ \phi(\|\boldsymbol{x_n} - \boldsymbol{x_1}\|) & \cdots & \phi(\|\boldsymbol{x_n} - \boldsymbol{x_n}\|) \end{bmatrix}$$

Note that $\|\boldsymbol{x_1} - \boldsymbol{x_1}\| = \cdots = \|\boldsymbol{x_n} - \boldsymbol{x_n}\| = 0$.

Radial Basis Functions

An RBF can be trained/fit in a very similar way as we discussed for polynomials, i.e. multiple linear regression:

$$\boldsymbol{w} = \left(\Phi^{\mathrm{T}}\Phi\right)^{-1}\Phi^{\mathrm{T}}\boldsymbol{f}$$

However, RBFs are typically implemented as interpolating models so Φ is square and invertible, and we can note that,

$$\left(\Phi^{\mathrm{T}}\Phi\right)^{-1}\Phi^{\mathrm{T}} = \left[\Phi^{-1}(\Phi^{T})^{-1}\right]\Phi^{\mathrm{T}} = \Phi^{-1}$$

So,

$$\boldsymbol{w} = \Phi^{-1}\boldsymbol{f}$$

Radial Basis Functions

Since

$$\boldsymbol{\varepsilon} = \boldsymbol{f} - \Phi \boldsymbol{w}$$

and

$$\boldsymbol{w} = \Phi^{-1} \boldsymbol{f}$$

the fitting error is,

$$\boldsymbol{\varepsilon} = \boldsymbol{f} - \Phi \Phi^{-1} \boldsymbol{f} = \boldsymbol{f} - \boldsymbol{f} = \mathbf{0}$$

as we would expect.

Some Typical Basis Functions for RBFs

Just like we can select the monomials to include in a polynomial basis for the Φ matrix, we can select the functional form of the basis function for RBFs. Here are some common forms:

- Multiquadric: $\phi(r) = (r^2 + r_0^2)^{1/2}$
- Inverse multiquadric: $\phi(r) = (r^2 + r_0^2)^{-1/2}$
- Thin plate spline: $\phi(r) = r^2 \ln(r/r_0)$

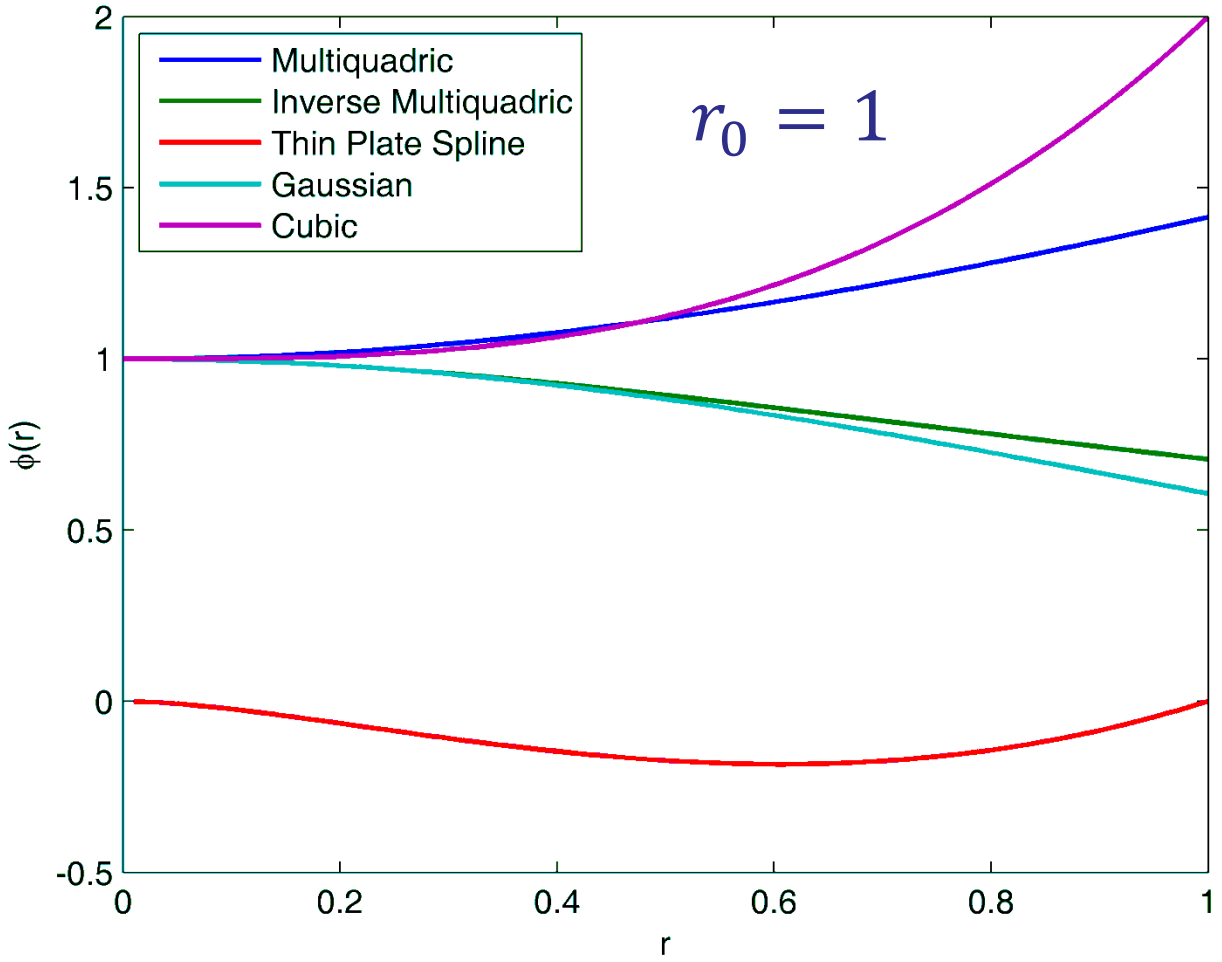
Some Typical Basis Functions for RBFs

- Gaussian: $\phi(r) = \exp\left(-\frac{1}{2}r^2/r_0^2\right)$

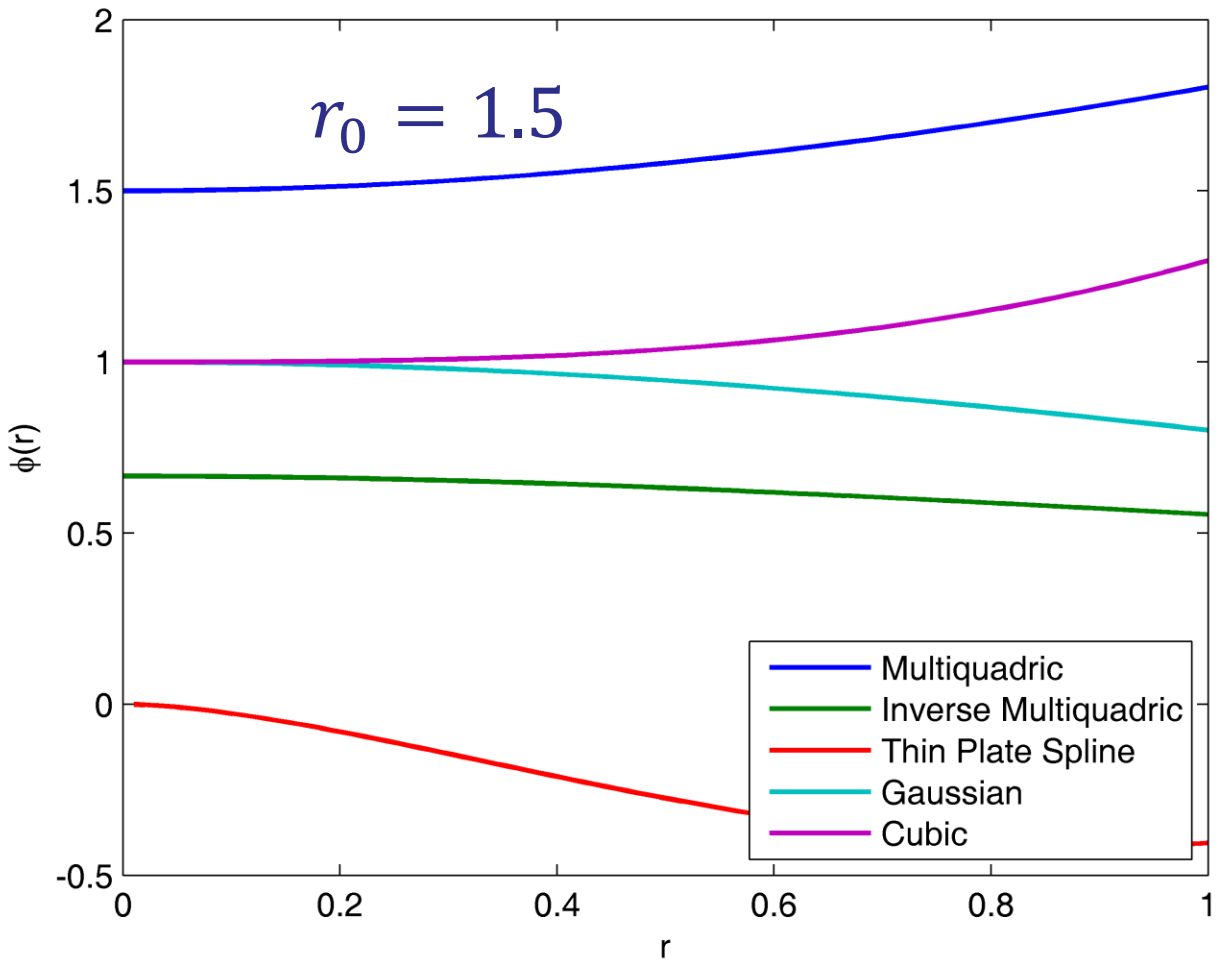
- Cubic: $\phi(r) = 1 + r^3/r_0^3$

In these basis functions, r_0 is a user-defined reference radius that serves as a scale factor.

Some Typical Basis Functions for RBFs

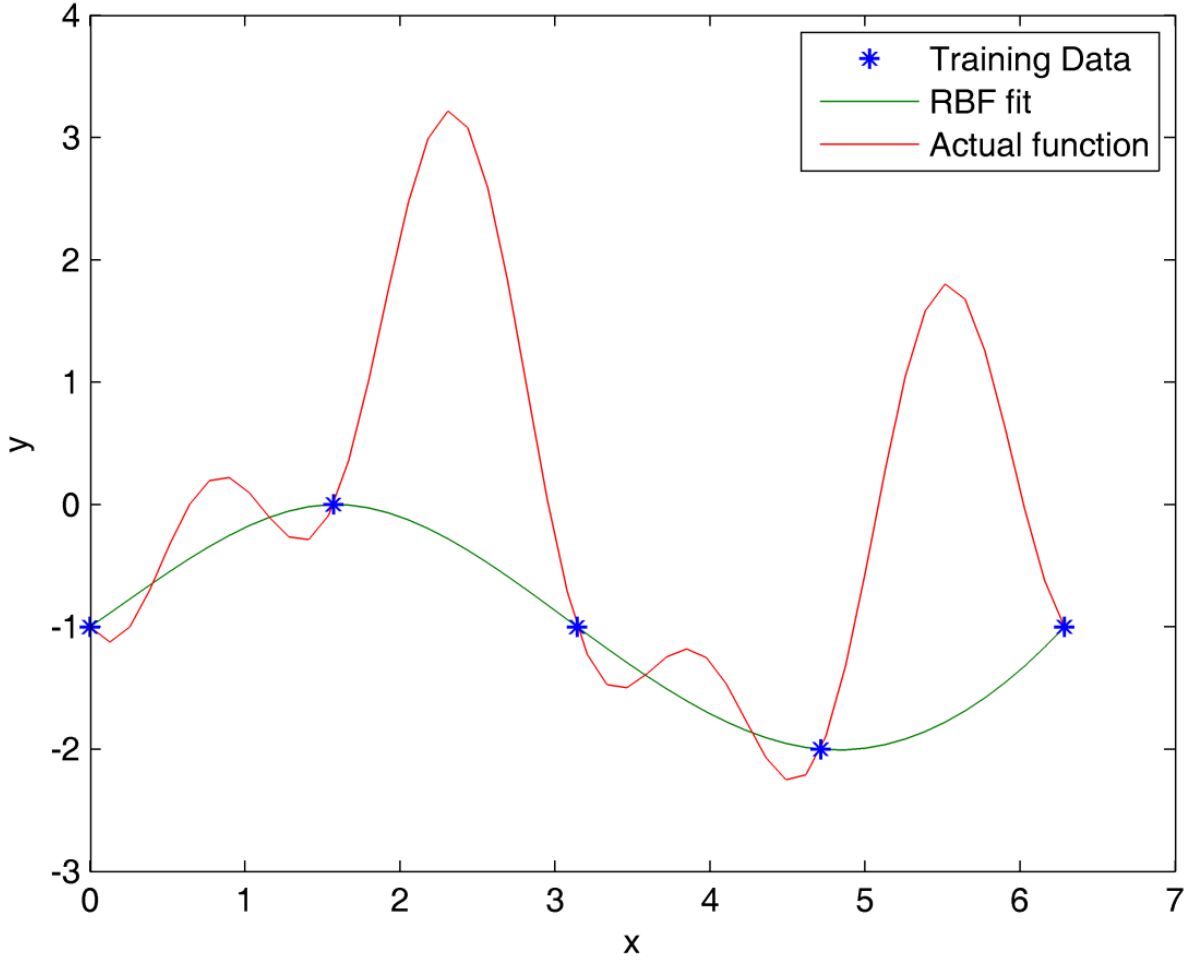


Some Typical Basis Functions for RBFs



RBFs: A 1-D Example

$$y = \sin(x) - 3\sin(x)\cos(x) - \cos(4x)$$



$$\phi(r) = 1 + r^3/r_0^3$$

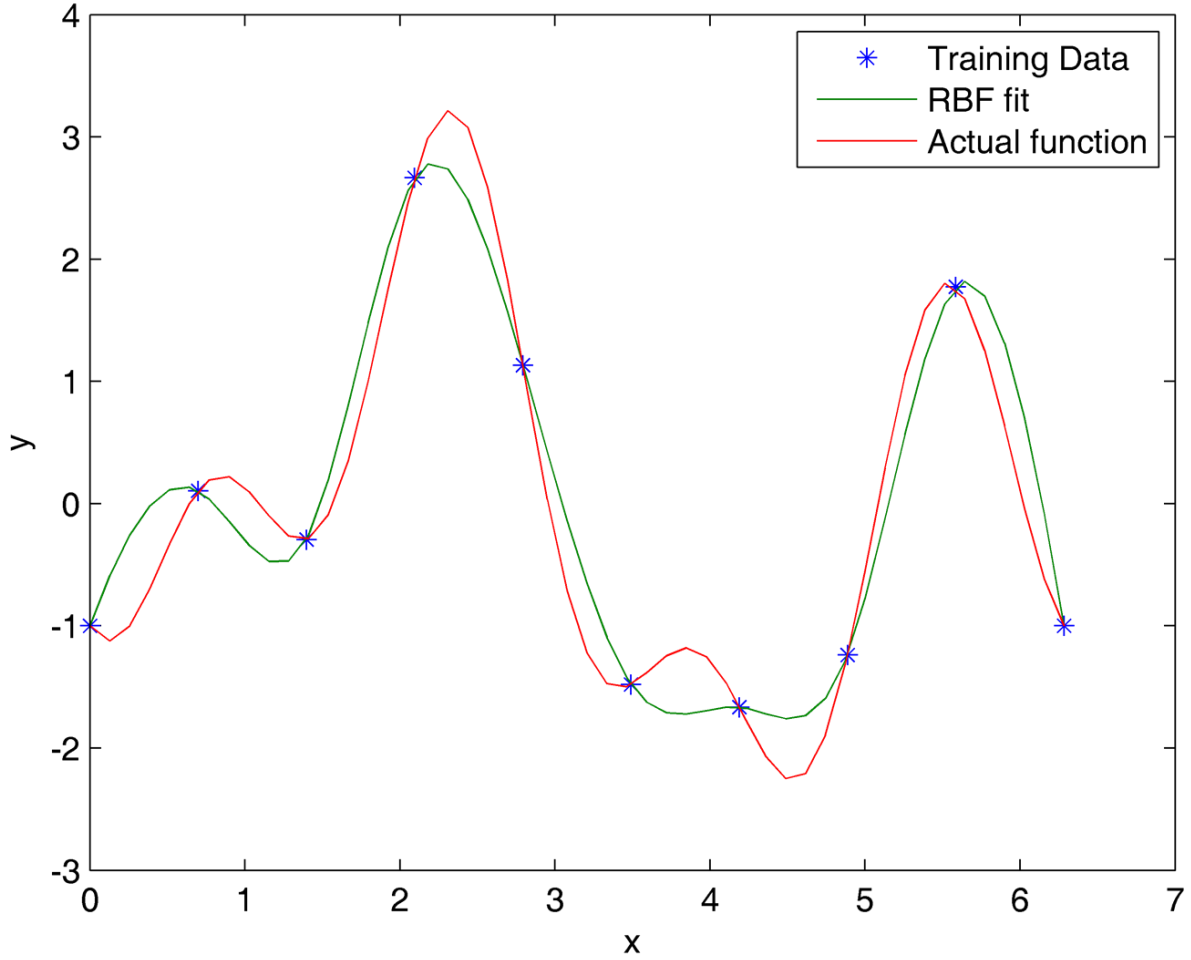
$$r_0 = 1$$

$$n = 5$$

Evenly spaced training points

RBFs: A 1-D Example

$$y = \sin(x) - 3\sin(x)\cos(x) - \cos(4x)$$



$$\phi(r) = 1 + r^3/r_0^3$$

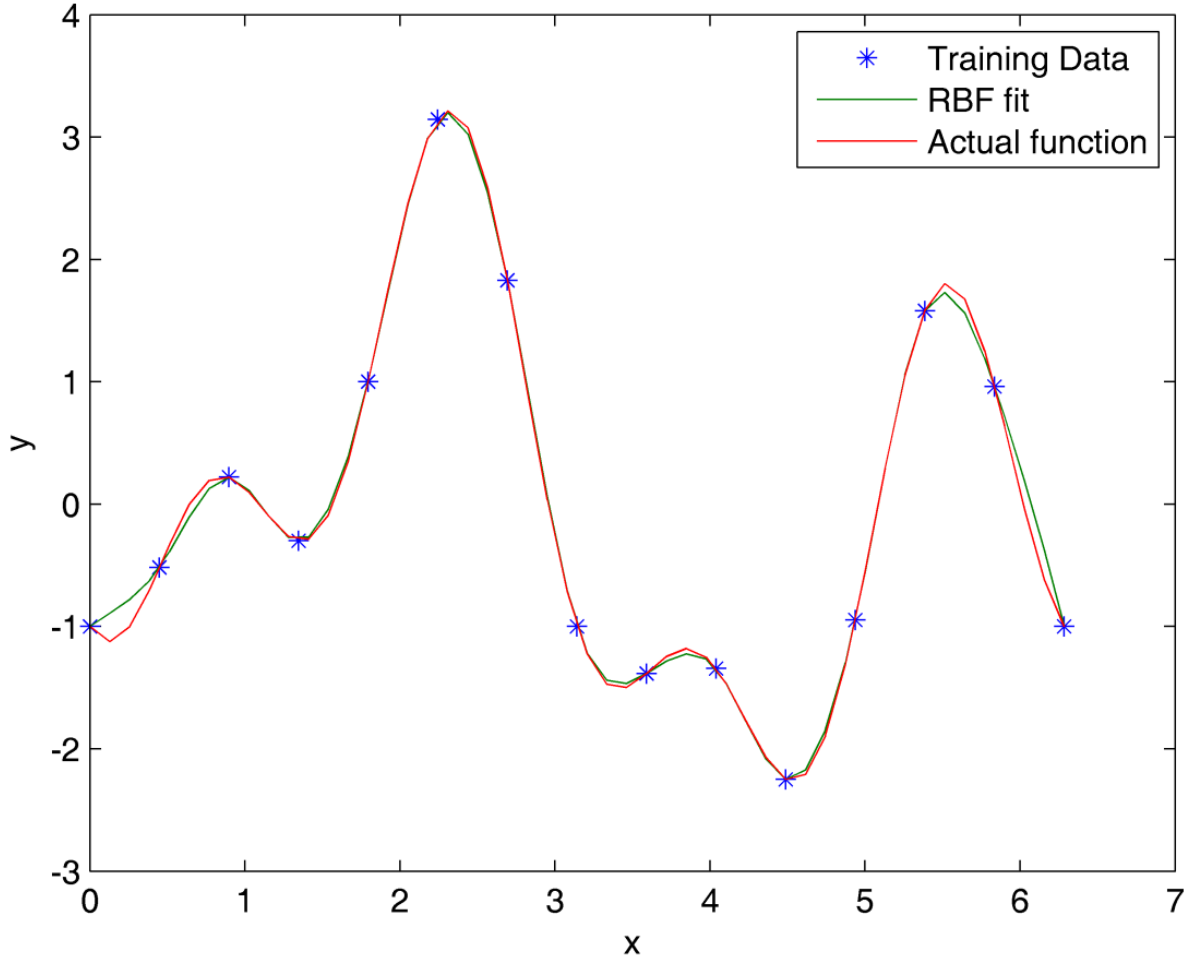
$$r_0 = 1$$

$$n = 10$$

Evenly spaced training points

RBFs: A 1-D Example

$$y = \sin(x) - 3\sin(x)\cos(x) - \cos(4x)$$



$$\phi(r) = 1 + r^3/r_0^3$$

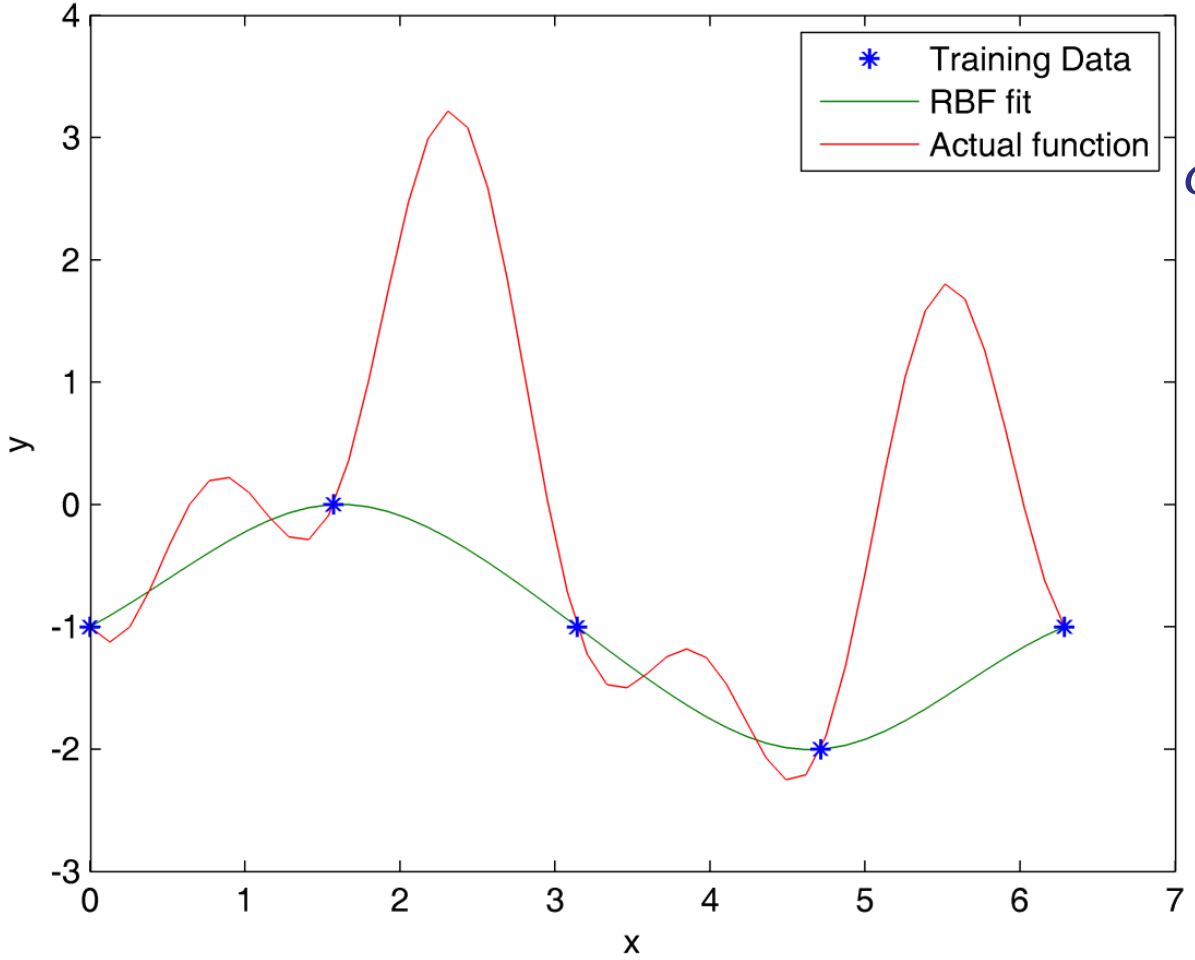
$$r_0 = 1$$

$$n = 15$$

Evenly spaced training points

RBFs: A 1-D Example

$$y = \sin(x) - 3\sin(x)\cos(x) - \cos(4x)$$



$$\phi(r) = (r^2 + r_0^2)^{1/2}$$

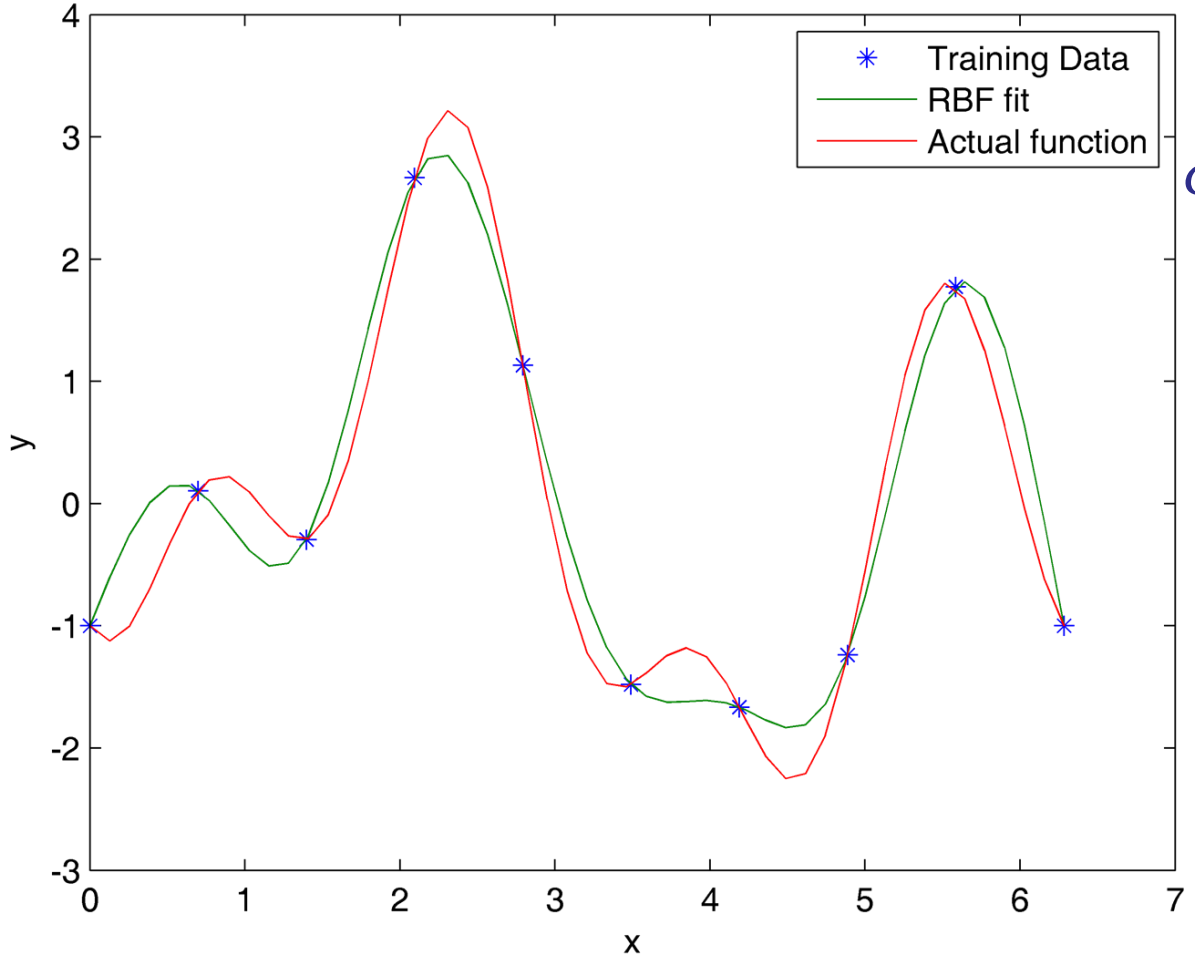
$$r_0 = 1$$

$$n = 5$$

Evenly spaced training points

RBFs: A 1-D Example

$$y = \sin(x) - 3\sin(x)\cos(x) - \cos(4x)$$



$$\phi(r) = (r^2 + r_0^2)^{1/2}$$

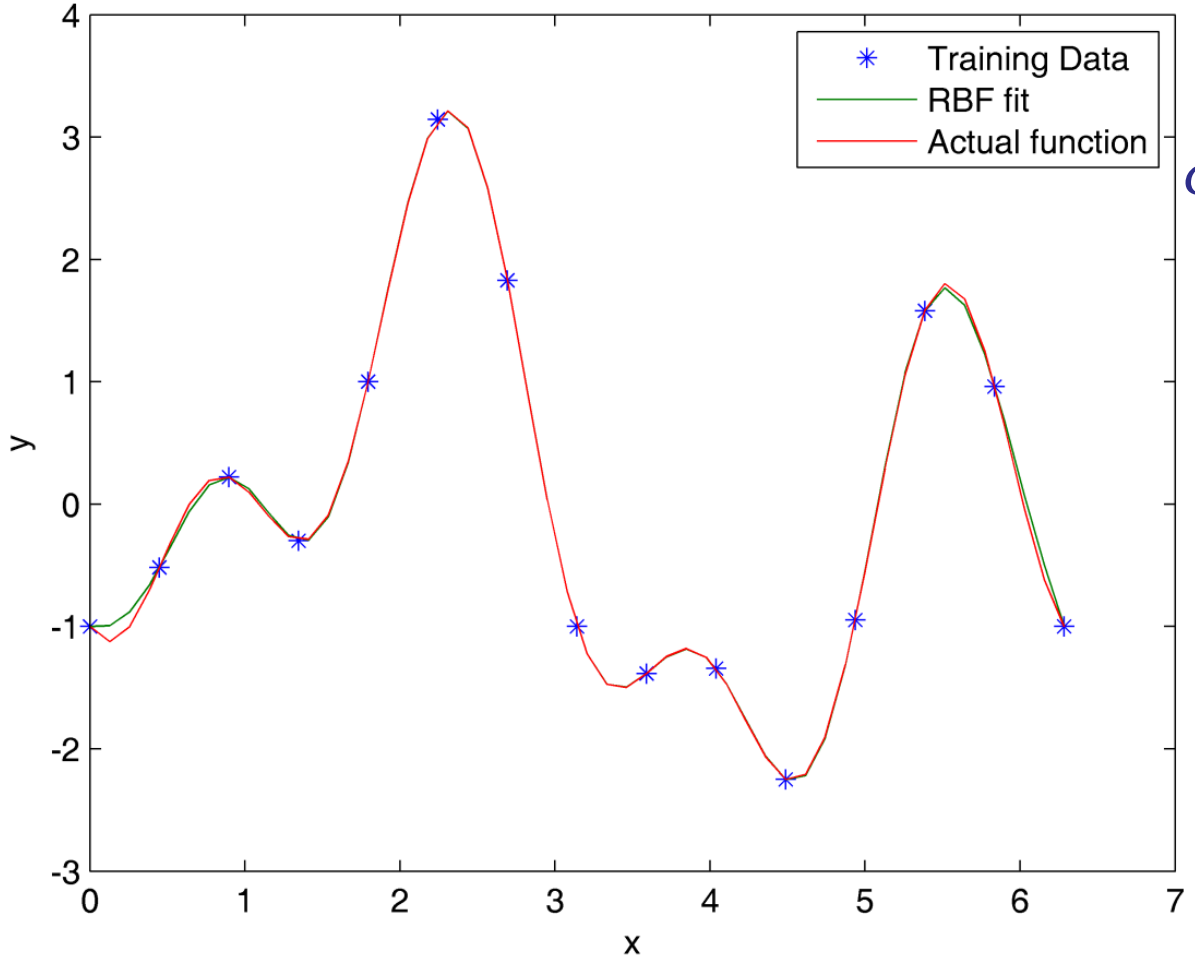
$$r_0 = 1$$

$$n = 10$$

Evenly spaced training points

RBFs: A 1-D Example

$$y = \sin(x) - 3\sin(x)\cos(x) - \cos(4x)$$



$$\phi(r) = (r^2 + r_0^2)^{1/2}$$

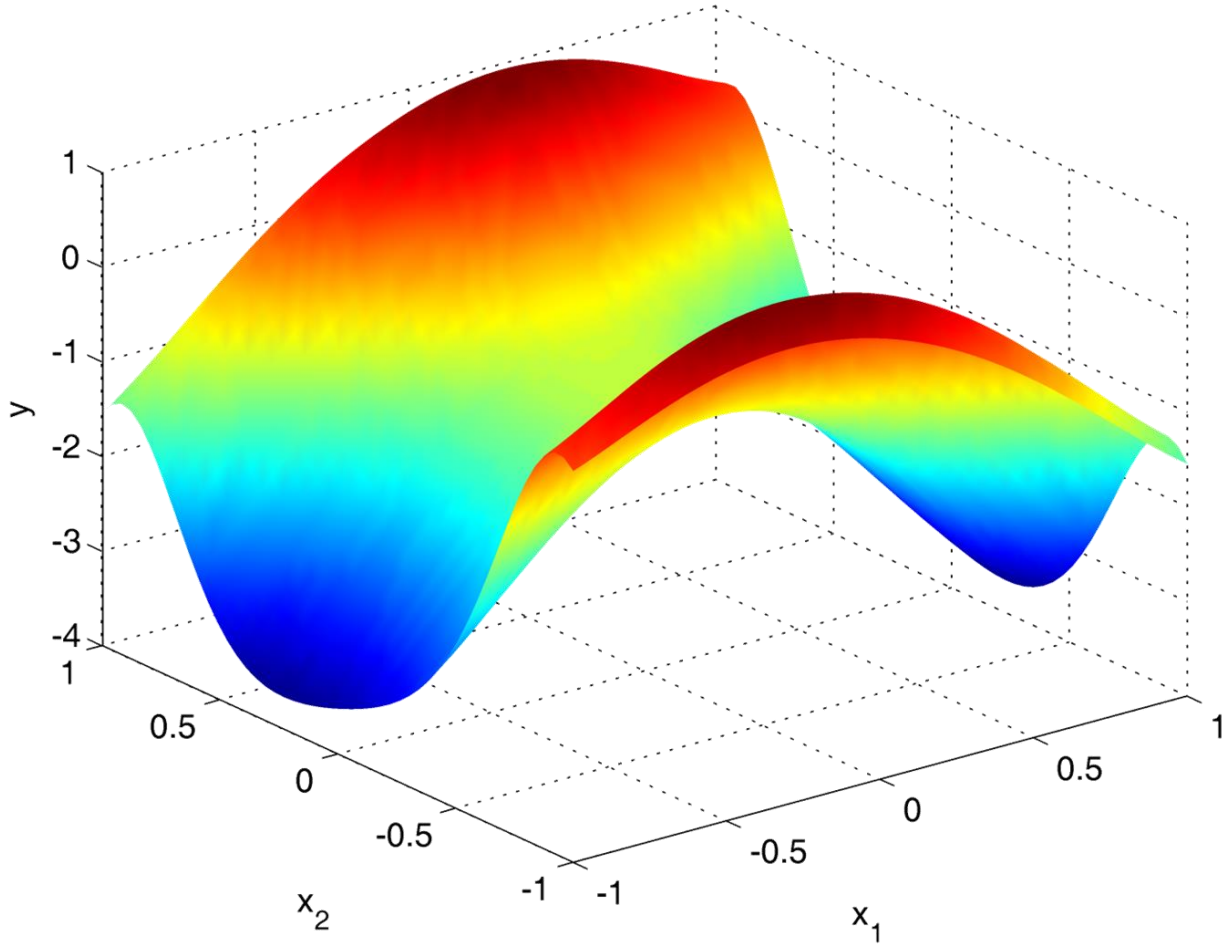
$$r_0 = 1$$

$$n = 15$$

Evenly spaced training points

RBFs: A 2-D Example

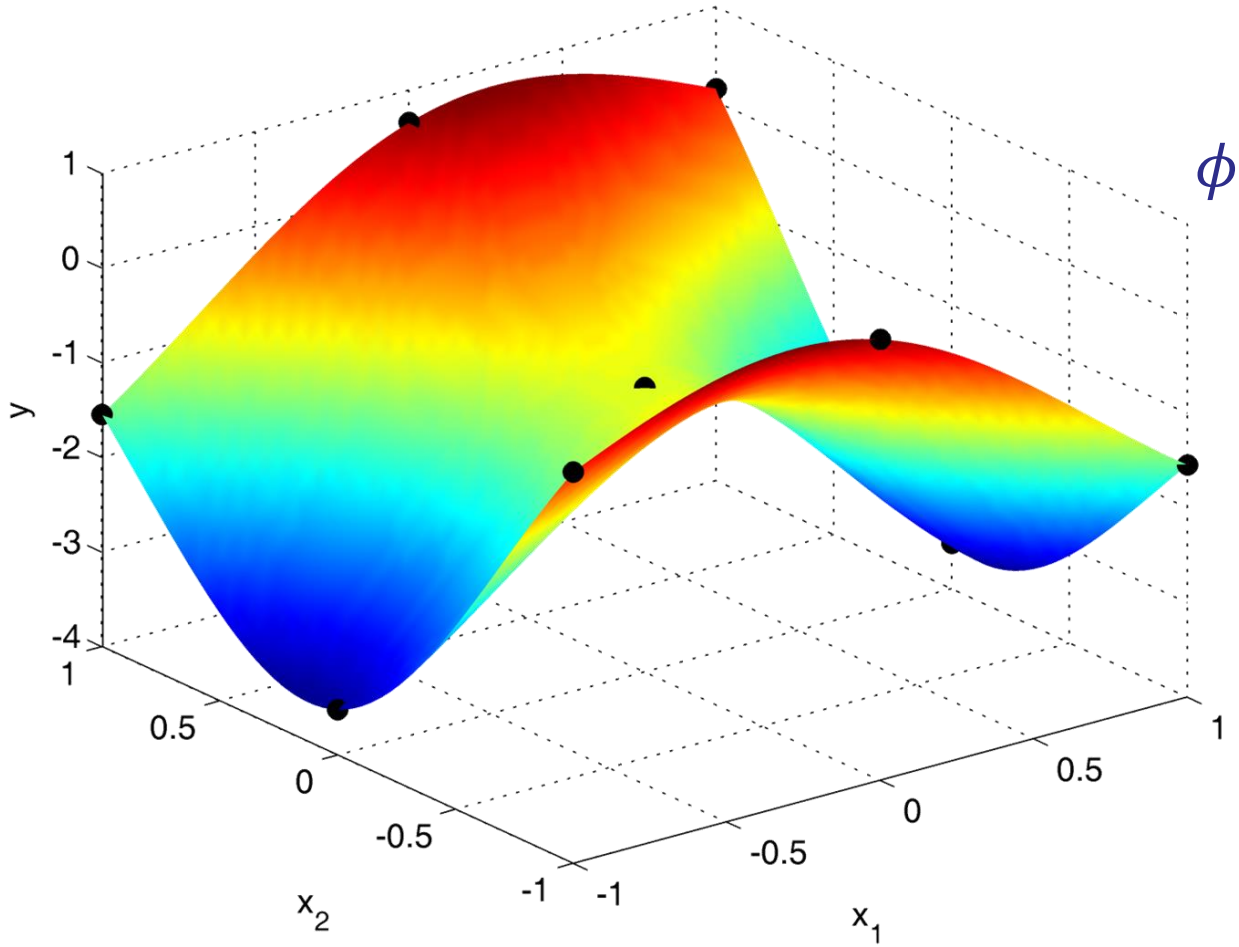
$$y = \sin(x_1 x_2) - 3\sin(x_1^2)\cos(x_2) - \cos(4x_2^2)$$



Original function

RBFs: A 2-D Example

$$y = \sin(x_1 x_2) - 3\sin(x_1^2)\cos(x_2) - \cos(4x_2^2)$$



$$\phi(r) = (r^2 + r_0^2)^{1/2}$$

$$r_0 = 1$$

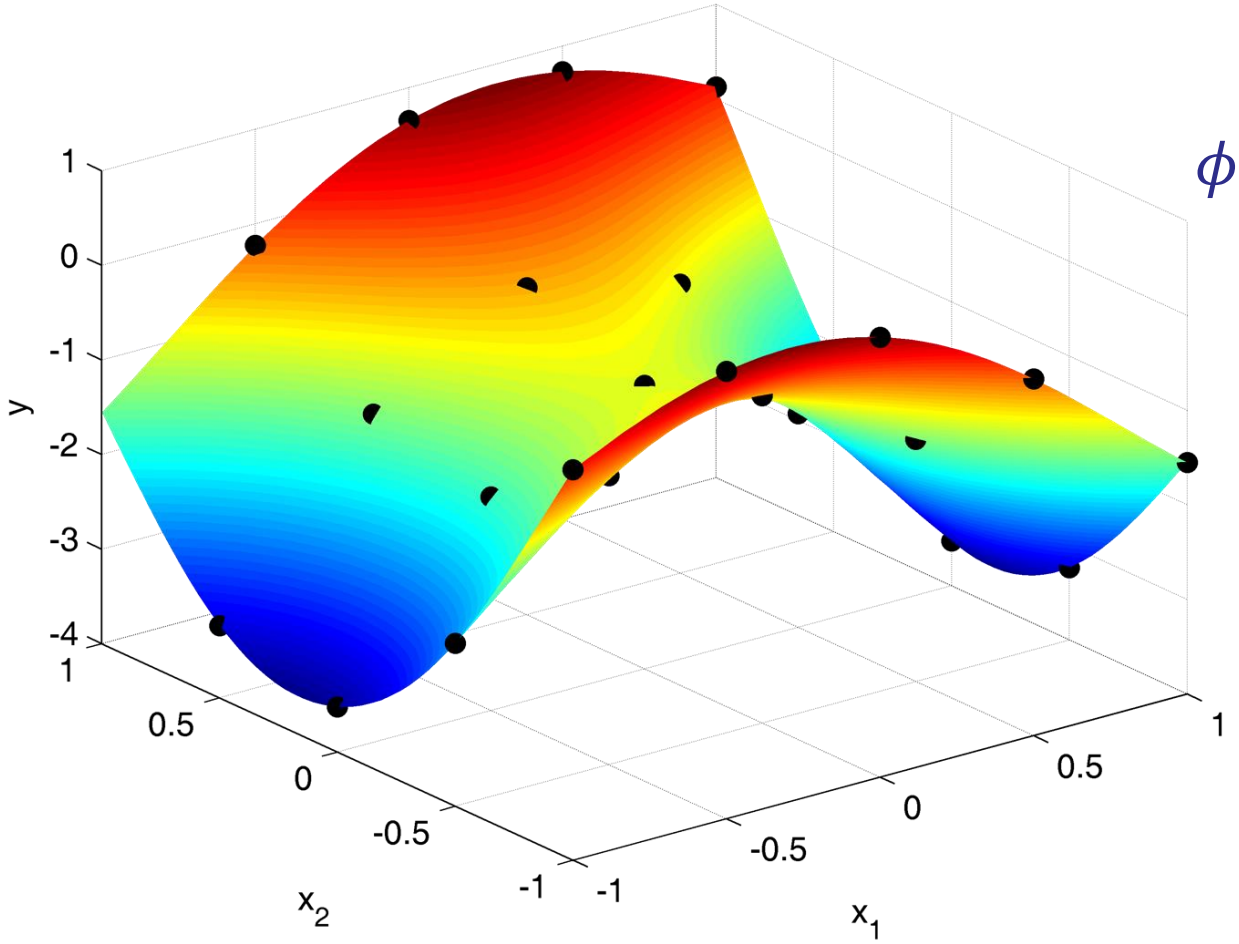
$$n = 3 \times 3 = 9$$

Evenly spaced training points

$$R_{val}^2 = 0.9257$$

RBFs: A 2-D Example

$$y = \sin(x_1 x_2) - 3\sin(x_1^2)\cos(x_2) - \cos(4x_2^2)$$



$$\phi(r) = (r^2 + r_0^2)^{1/2}$$

$$r_0 = 1$$

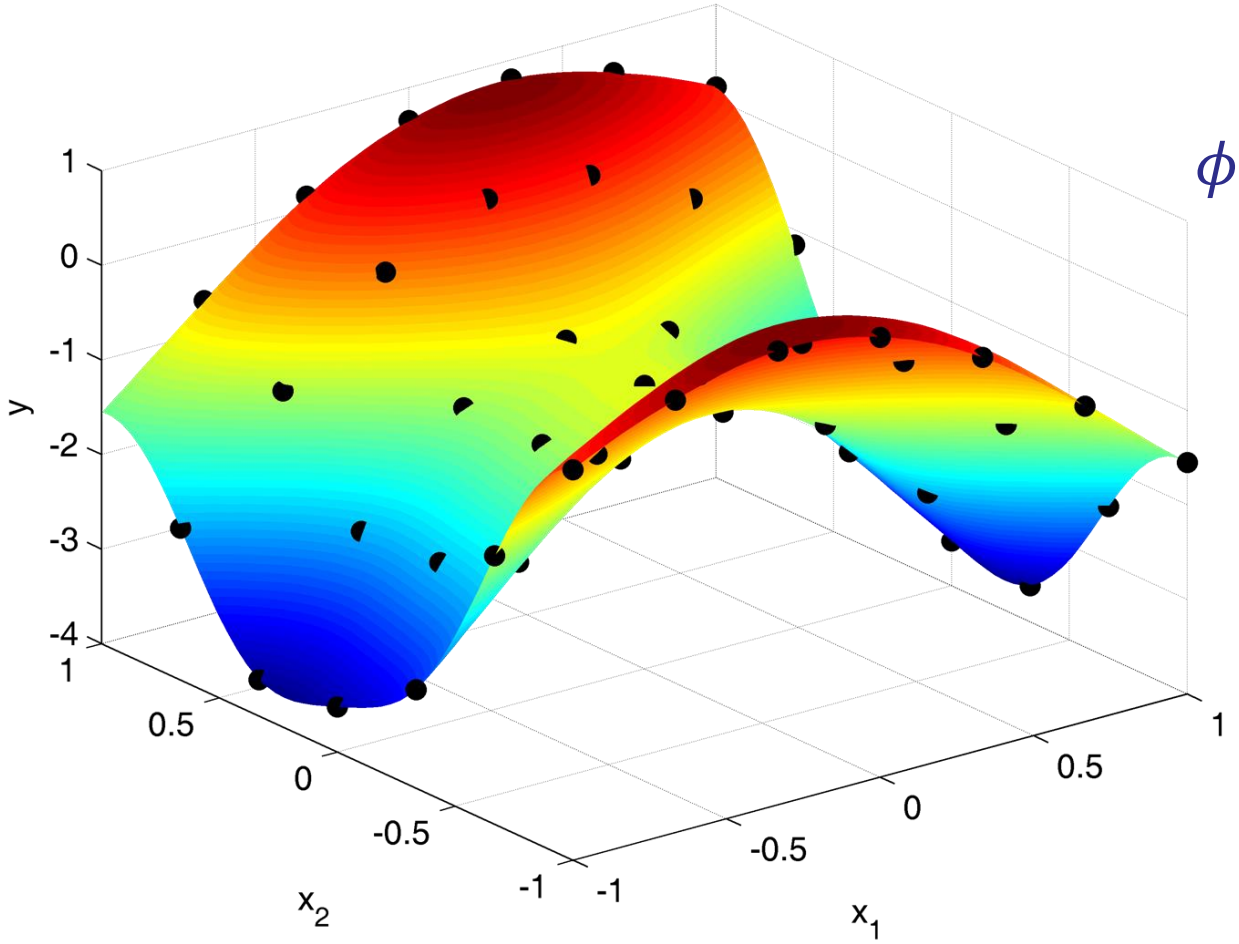
$$n = 5 \times 5 = 25$$

Evenly spaced training points

$$R_{val}^2 = 0.9189$$

RBFs: A 2-D Example

$$y = \sin(x_1 x_2) - 3\sin(x_1^2)\cos(x_2) - \cos(4x_2^2)$$



$$\phi(r) = (r^2 + r_0^2)^{1/2}$$

$$r_0 = 1$$

$$n = 7 \times 7 = 49$$

Evenly spaced training points

$$R_{val}^2 = 0.9918$$

Artificial Neural Networks

So far, we have looked at models that can be fit with **linear regression**, i.e. we can solve for the model parameters with a optimization problem that results in a linear system of equations.

The **artificial neural network** is one example of a model that requires **nonlinear regression** for fitting, i.e. we need to solve a nonlinear optimization problem.

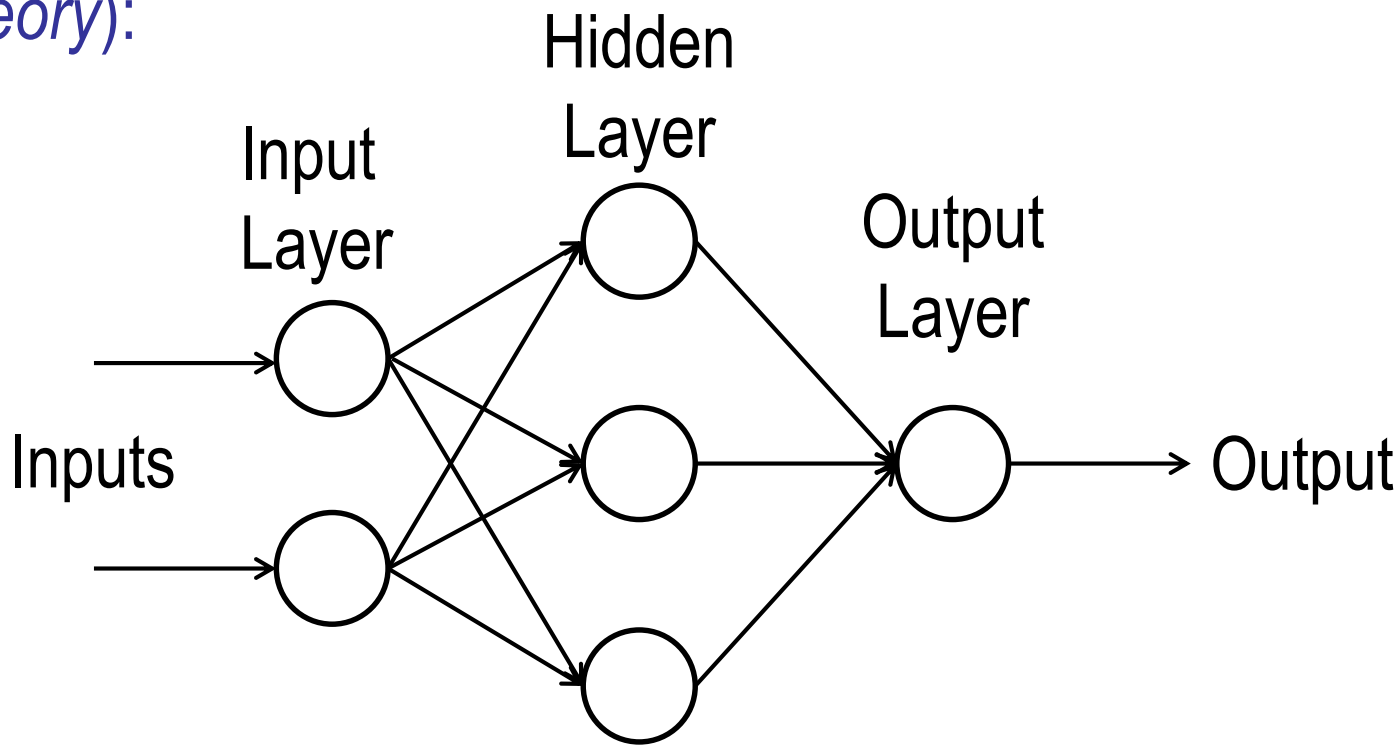
Artificial Neural Networks

Artificial neural networks are inspired by analogy with the structure of the human brain:

- **Neurons** are brain cells
- Neurons will **activate** ("fire") in response to electrical signals from other neurons if the signals are above some threshold
- Neurons are interconnected with **synapses**

Artificial Neural Networks

Artificial neural networks can be interpreted as modeling the neuronal anatomy with a weighted directed graph (in the context of *graph theory*):



Nodes are analogous to neurons; edges are analogous to synapses

Artificial Neural Networks

The **topology** of the ANN is defined by the following characteristics:

- Number of hidden layers
- Number of hidden nodes per layer
- Interconnection pattern of nodes and edges, e.g. feed-forwards vs. feedbacks

The most common topology of ANN for data regression has 1 hidden layer and all feed-forward connections. This was the class of topologies shown on the figure in the previous slide.

Artificial Neural Network

The single hidden layer feed-forward network for regression can be implemented as follows:

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i \phi\left(a_i\right)$$

where

$$a_i = \sum_{j=1}^{p} w_{ij} x_j + \beta_j$$

and p is the number of design variables, m is the number of hidden nodes, and α_i, w_{ij}, and β_j are the parameters of the model that have to be found in the training process.

Artificial Neural Network

The function ϕ is the activation function. It is typically chosen as a smooth approximation to a step function.

Most commonly, the logistic function is used:

$$\phi(a_i) = \frac{1}{1 + \exp(-a_i)}$$