Matlab 13: Data Fitting and Regression Analysis

Cheng-Hsin Hsu

National Tsing Hua University

Department of Computer Science

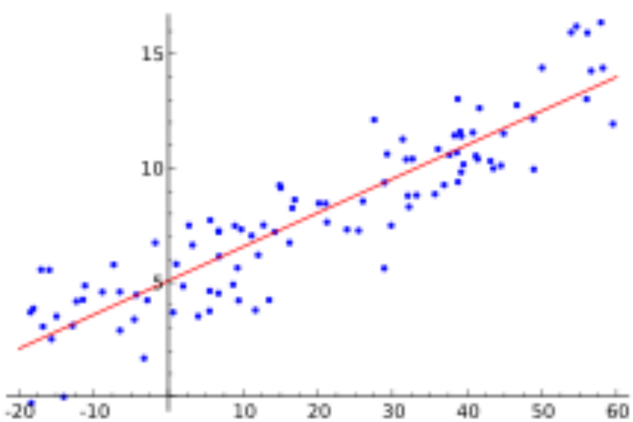
Slides and sample codes are based on the materials from Prof. Roger Jang

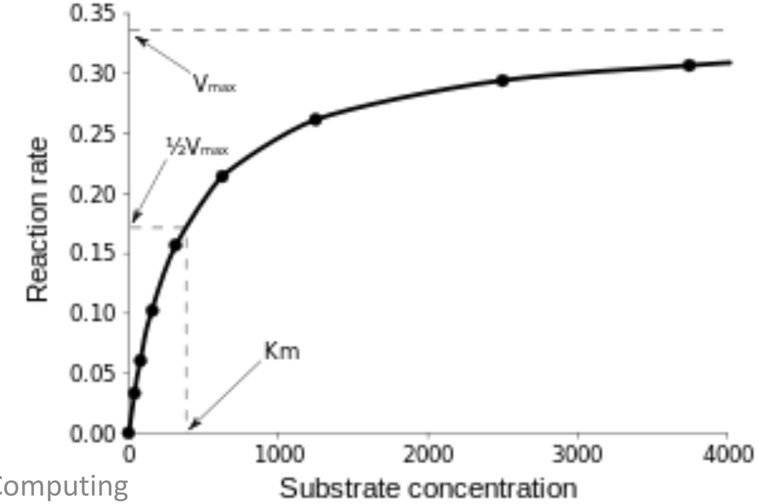
What is Data Fitting

- For a set of data, both inputs and outputs, construct a mathematical model to approximate the outputs given the inputs
  - Single input: e.g., height → weight
  - Multiple inputs: e.g., height, sex, and body-fat percentage → weight
- Ways to derive the mathematical models, e.g.,
  - Single input: curve fitting
  - Two inputs: surface fitting

Regression Analysis

- Statistical process to perform data fitting, including modeling and analyzing variables (inputs and outputs)
- Linear regression: linear models
- Non-linear regression: non-linear models



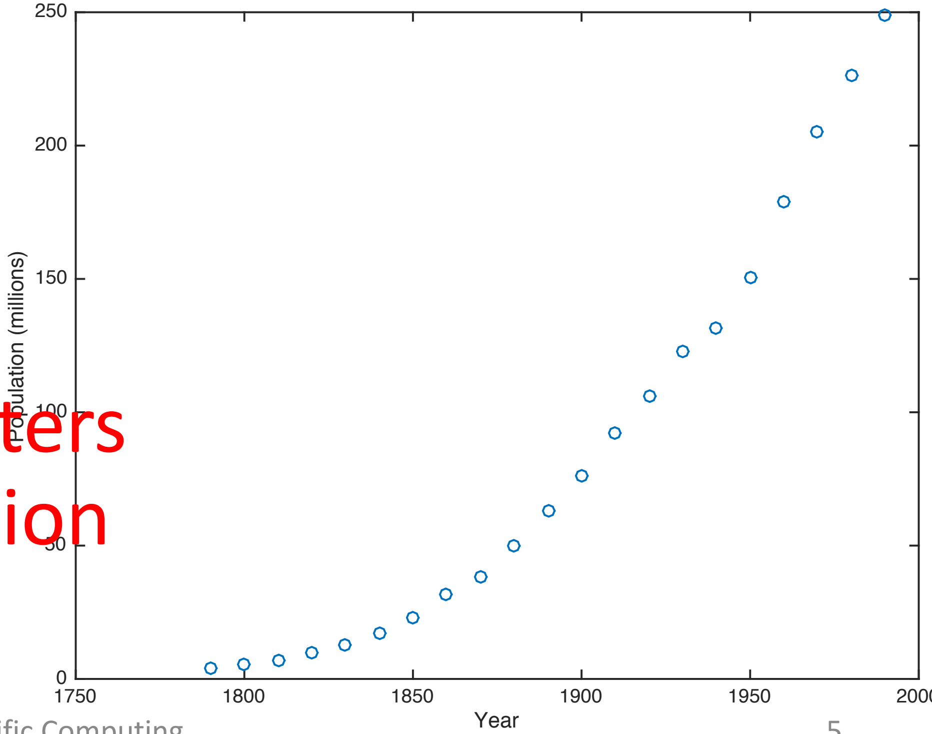


Data Fitting: Census Dataset

- The file census.mat contains U.S. population data for the years 1790 through 1990
- Load it into Matlab, using
  - load census
  - observe the variables: cdate (years) and pop (population in millions)
- Plot the data points
  - plot(cdate, pop, 'o');
  - xlabel('Year'); ylabel('Population (millions)')

Model Selection

- By visual inspection, we may choose to fit the dataset to a quadratic function
  - $y = f(x; a_0, a_1, a_2) = a_0 + a_1 x + a_2 x^2$
  - x is the input
  - y is the output
  - a_0—a_2 are model parameters
- Goal: the best parameters to minimize the deviation



Objective Function for Data Fitting

- Objective func.: Sum of mean-squared errors
- Dataset (x_i, y_i) for i =1, 2, …, 21; the output is y_i when the input is x_i
- Modeled output is: $f(x_i; a_0, a_1, a_2) = a_0 + a_1 x_i + a_2 x_i^2$
- Squared error: $[y_i - f(x_i)]^2$
- Now we can write the objective function as:

$$E(a_0, a_1, a_2) = \sum_{i=1}^{21} [y_i - f(x_i)]^2 = \sum_{i=1}^{21} \left[y_i - \left(a_0 + a_1 x_i + a_2 x_i^2 \right) \right]^2$$

Minimizing the Objective Function

- Note that E(...) is a function of a_0, a_1, a_2
- Find the partial derivative of E(...) wrt a_0, a_1, a_2, and then set them to zero for the extreme values
- $\frac{\partial E}{\partial a_0}, \frac{\partial E}{\partial a_1}, \frac{\partial E}{\partial a_2}$ are linear functions
- Setting them to be zeros, we get a system of three linear euqations with three unknowns
- Solving the system leads to optimal solution

Matrix Representation

- Consider the 21 data points, putting them into the quadratic function gives

$$\begin{cases} a_0 + a_1 x_1 + a_2 x_1^{\ 2} = y_1 \\ a_0 + a_1 x_2 + a_2 x_2^{\ 2} = y_2 \\ \vdots \\ a_0 + a_1 x_{21} + a_2 x_{21}^{\ 2} = y_{21} \end{cases}$$

- Written in matrices
  - Parameters are θ

$$\underbrace{\begin{bmatrix} 1 & x_1 & x_1^{\ 2} \\ 1 & x_2 & x_2^{\ 2} \\ & \vdots & \\ 1 & x_{21} & x_{21}^{\ 2} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}}_{\boldsymbol{\theta}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{21} \end{bmatrix}}_{\mathbf{b}}$$

Data Fitting in Matlab

- Observation: We have 3 parameters but 21 sample points ← Most likely there is no "perfect" model parameters for all 21 points
- Instead, we search for an optimal θ^* to minimize the difference at the two side of the equality
  - Minimizing the sum of squared error (SSE)

$$E(\boldsymbol{\theta}) = \|\mathbf{b} - A\boldsymbol{\theta}\|^2 = (\mathbf{b} - A\boldsymbol{\theta})^T(\mathbf{b} - A\boldsymbol{\theta})$$

Solving Systems of Linear Equations

- To solve $A\theta = b$, we use theta = A\b in Matlab
  - If A is scalar, then A\b is the same as A.\b
  - If A is n x n and b is n x 1, then A\b gives the unique solution, if exists
  - If A is m x n and b is n x 1, where m >= n, then A\b gives the least-squares solution

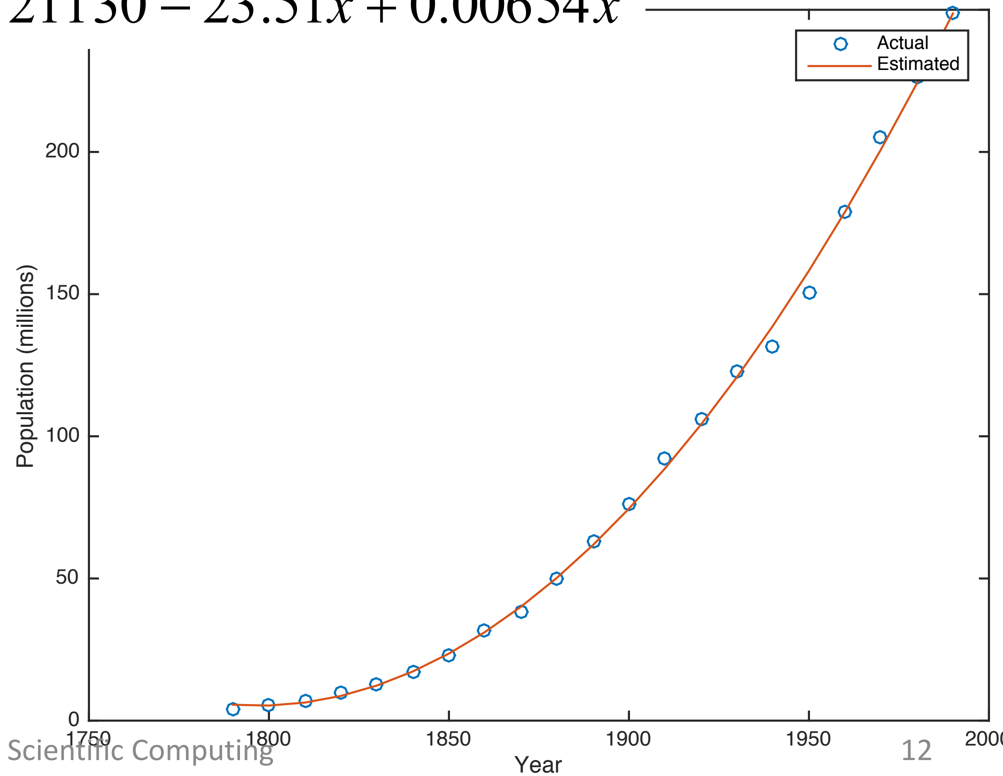
Data Fitting Example

```
load census.mat
A = [ones(size(cdate)), cdate, cdate.^2];
b = pop;
theta = A\b;
plot(cdate, pop, 'o', cdate, A*theta, '-');
legend('Actual', 'Estimated');
xlabel('Year');
ylabel('Population (millions)');
```

Data Fitting Results

- We know $\theta^T = [a_0, a_1, a_2]^T = [21130, -23.51, 0.00654]^T$
- Then, our model is

$$y = f(x) = a_0 + a_1 x + a_2 x^2 = 21130 - 23.51x + 0.00654x^2$$



Forward Slash is Similar

mrdivide, /

Solve systems of linear equations *xA* = *B* for *x*

Syntax

```
x = B/A
x = mrdivide(B,A)
```

Description

`x = B/A` solves the system of linear equations `x*A = B` for `x`. The matrices `A` and `B` must contain the same number of columns. MATLAB® displays a warning message if `A` is badly scaled or nearly singular, but performs the calculation regardless.

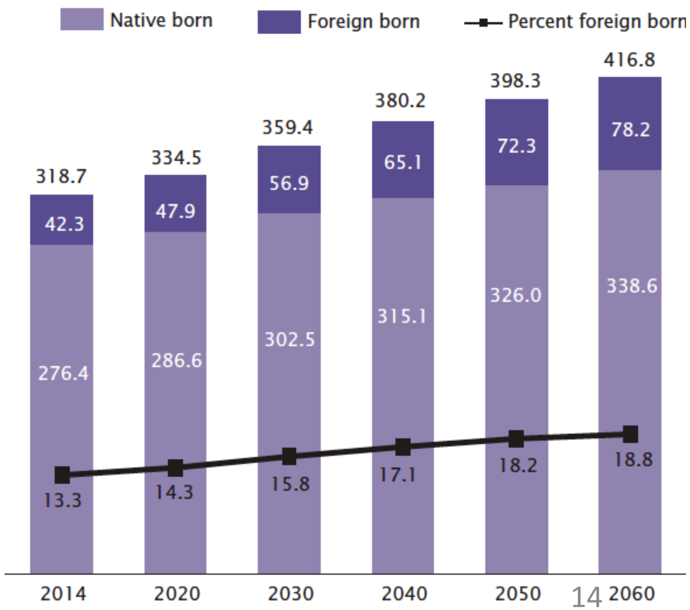
- If `A` is a scalar, then `B/A` is equivalent to `B./A`.
- If `A` is a square `n`-by-`n` matrix and `B` is a matrix with `n` columns, then `x = B/A` is a solution to the equation `x*A = B`, if it exists.
- If `A` is a rectangular `m`-by-`n` matrix with `m ~= n`, and `B` is a matrix with `n` columns, then `x = B/A` returns a least-squares solution of the system of equations `x*A = B`.

`x = mrdivide(B,A)` is an alternative way to execute `x = B/A`, but is rarely used. It enables operator overloading for classes.

Estimate the Populations

- Predict the populations using the derived model
  - t=2010; pop2010 = [1, t, t^2]*theta
  - t=2014; pop2014 = [1, t, t^2]*theta
  - pop2014 = 313.0710

Figure 1.
U.S. Population by Nativity: 2014 to 2060
(Population in millions)



Source: U.S. Census Bureau, 2014 National Projections.

Polynomial Fitting

- Generalization of quadratic functions
- $y = f(x) = a_0 + a_1 x + \cdots + a_n x^n$
- Matlab offers two commands for polynomial fitting
  - polyfit: finding the best model parameters
  - polyval: evaluate the value for a given model

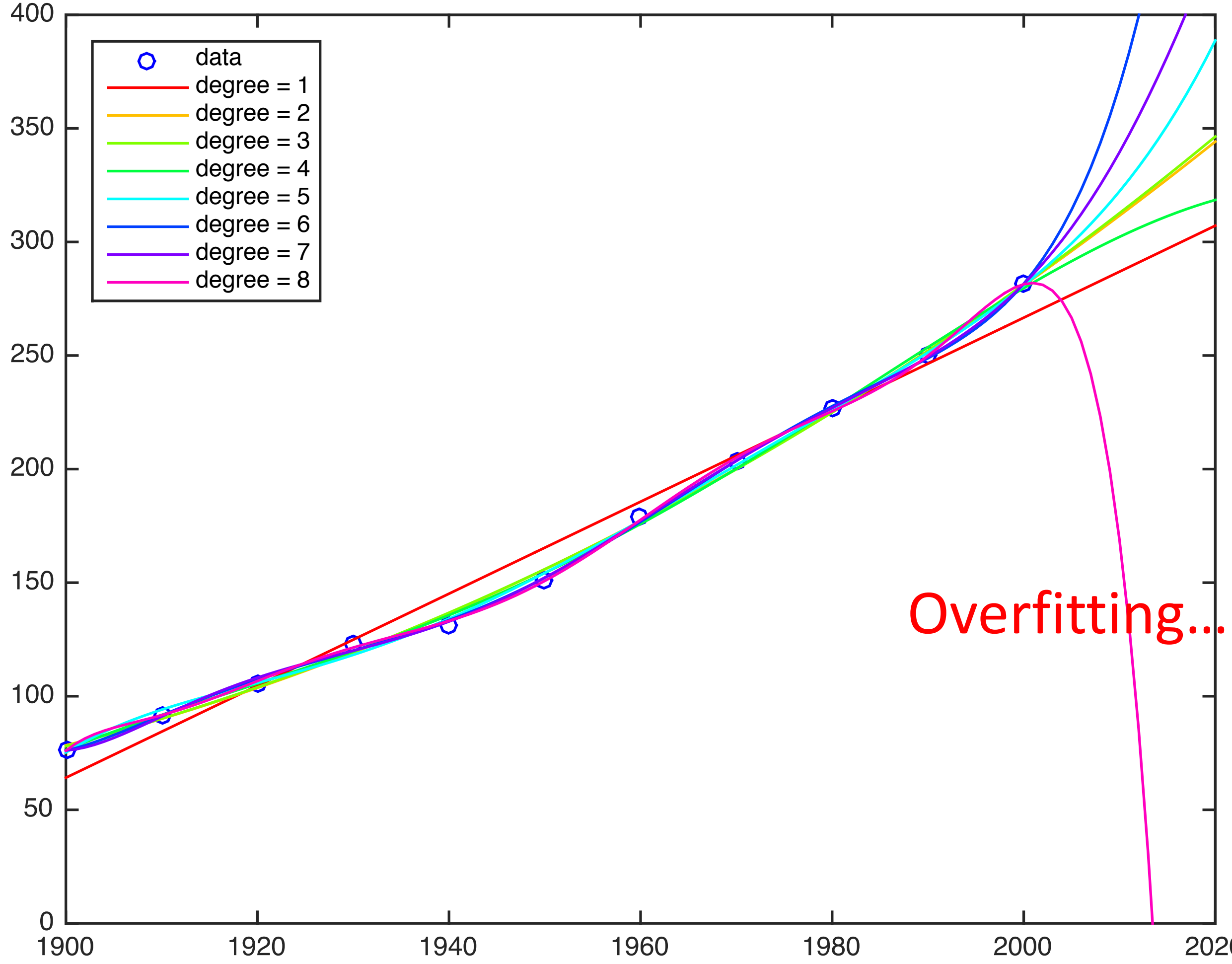
Using Polyfit and Polyval

- For the same tasks, polyfit/polyval lead to more readable code

```
load census.mat
theta = polyfit(cdate, pop, 2);
polyval(theta, 2000)
polyval(theta, 2014)
```

More Accurate Data Fittings?

- census

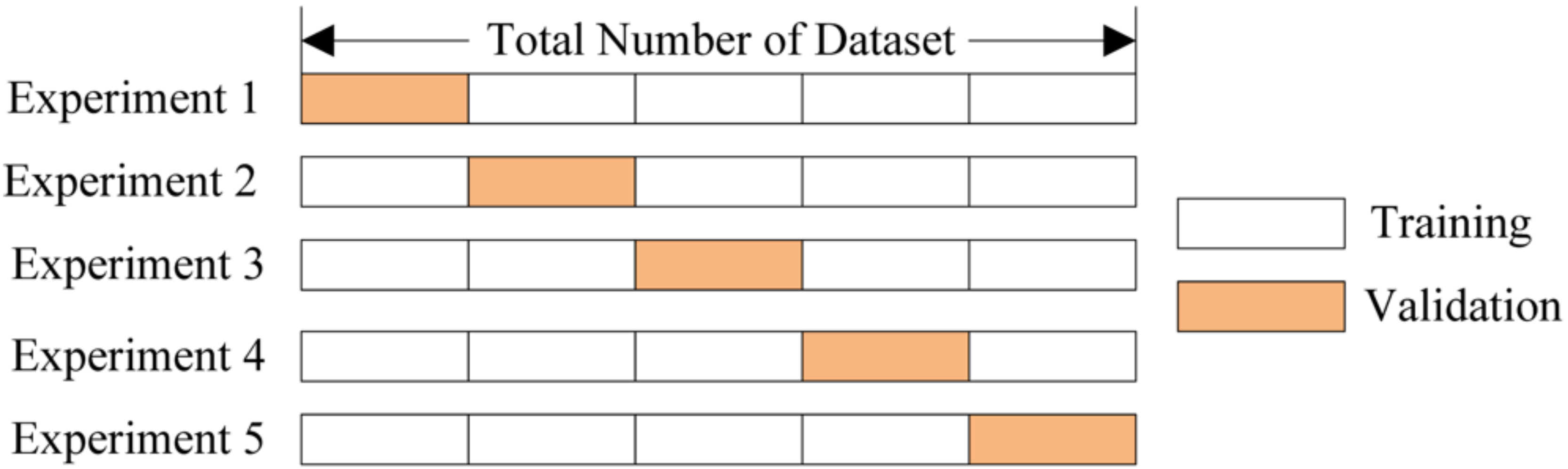


Model Complexity and Accuracy

- Selection of models is crucial
  - More complexity models (more parameters) lead to smaller sum of squared errors
  - Extreme case in polynomial, if the order is the same as the number data points, we may even have zero squared errors
  - But our model may faithfully reproduce randomness and noise! → less accurate
- Known as over-fitting

Overfitting

- When the model describes the randomness and noise rather than the actual relations
- Possible solution: K-fold cross validation



Multiple Inputs Single Output

- Mathematical model:

$$y = f(\mathbf{x}) = \theta_1 f_1(\mathbf{x}) + \theta_2 f_2(\mathbf{x}) + \cdots + \theta_n f_n(\mathbf{x})$$

- **x** is input, y is output, $\theta_1, \theta_2, \cdots \theta_n$ are model parameters
- $f_i(\mathbf{x}), i = 1 \cdots n$ are known functions, called basis functions
- $(\mathbf{x}_i, y_i), i = 1 \cdots m$ are the sample data or training data

Matrix Representation

- What we have

$$\begin{cases} y_1 = f(\mathbf{x}_1) = \theta_1 f_1(\mathbf{x}_1) + \theta_2 f_2(\mathbf{x}_1) + \cdots + \theta_n f_n(\mathbf{x}_1) \\ \vdots \\ y_m = f(\mathbf{x}_\mathrm{m}) = \theta_1 f_1(\mathbf{x}_\mathrm{m}) + \theta_2 f_2(\mathbf{x}_\mathrm{m}) + \cdots + \theta_n f_n(\mathbf{x}_\mathrm{m}) \end{cases}$$

- In matrix representation

$$\underbrace{\begin{bmatrix} f_1(\mathbf{x}_1) & \cdots & f_n(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ f_1(\mathbf{x}_m) & \cdots & f_n(\mathbf{x}_m) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}}_{\boldsymbol{\theta}} = \underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}}_{\mathbf{b}}$$

Sum of Squared Error

- Since $m > n$ (number of data points is more than number of parameters), we need to add an error vector e, so that : $A\boldsymbol{\theta} + \mathbf{e} = \mathbf{b}$
- Squared error: $E(\boldsymbol{\theta}) = \|\mathbf{e}\|^2 = \mathbf{e}^T\mathbf{e} = (\mathbf{b} - A\boldsymbol{\theta})^T(\mathbf{b} - A\boldsymbol{\theta})$
- Optimal solutions
  – Partial derivative of $E(\theta)$ wrt θ and set it be zero for a system of n linear equations with n unknowns

Least-squares Estimate

- Derivation of LSE

$$
\begin{aligned}
E(\boldsymbol{\theta}) &= (A\boldsymbol{\theta} - \mathbf{b})^T (A\boldsymbol{\theta} - \mathbf{b}) \\
&= (\boldsymbol{\theta}^T A^T - \mathbf{b}^T)(A\boldsymbol{\theta} - \mathbf{b}) \\
&= \boldsymbol{\theta}^T A^T A\boldsymbol{\theta} - \boldsymbol{\theta}^T A^T \mathbf{b} - \mathbf{b}^T A\boldsymbol{\theta} + \mathbf{b}^T \mathbf{b} \\
&= \boldsymbol{\theta}^T A^T A\boldsymbol{\theta} - 2\boldsymbol{\theta}^T A^T \mathbf{b} + \mathbf{b}^T \mathbf{b}
\end{aligned}
$$

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \left(\boldsymbol{\theta}^T A^T A\boldsymbol{\theta} - 2\boldsymbol{\theta}^T A^T \mathbf{b} + \mathbf{b}^T \mathbf{b}\right) \\
&= 2A^T A\boldsymbol{\theta} - 2A^T \mathbf{b}
\end{aligned}
$$

$$
\nabla_{\boldsymbol{\theta}} E(\hat{\boldsymbol{\theta}}) = 0 \Rightarrow \underbrace{A^T A\hat{\boldsymbol{\theta}} = A^T \mathbf{b}}_{\text{Normal equation}} \Rightarrow \hat{\boldsymbol{\theta}} = \underbrace{\left(A^T A\right)^{-1} A^T}_{\text{Pseudo inverse of A}} \mathbf{b}
$$

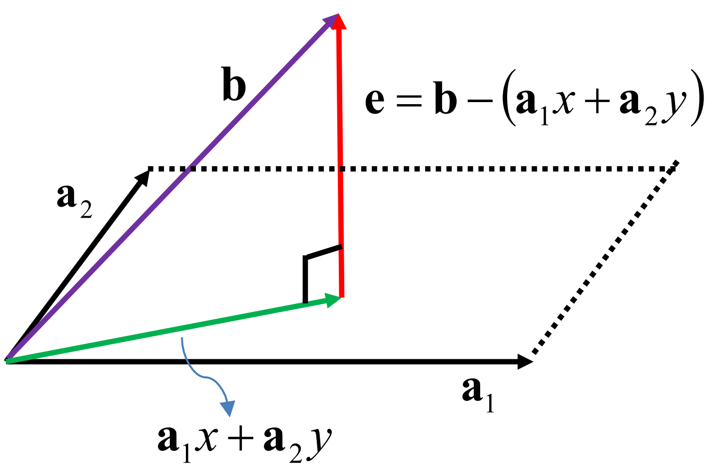
Least-squares Estimate (cont.)

- Optimal solutions
  - Using matrix operations, the optimal solution can be written as $\left(A^T A\right)^{-1} A^T \mathbf{b}$
  - Matlab's backslash can also be used $\hat{\boldsymbol{\theta}} = A \backslash \mathbf{b}$
- Backslash adopts some variations of the optimal solution ($\left(A^T A\right)^{-1} A^T \mathbf{b}$) based on the properties of A for more stable and accurate results

Geometric View of Least Squared Error

- Derivation of LSE via geometric view

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \Leftrightarrow A\mathbf{x} + \mathbf{e} = \mathbf{b} \Leftrightarrow \mathbf{a}_1 x + \mathbf{a}_2 y + \mathbf{e} = \mathbf{b}$$



$$\begin{cases} \mathbf{a}_1^T(\mathbf{a}_1\hat{x} + \mathbf{a}_2\hat{y} - \mathbf{b}) = 0 \\ \mathbf{a}_2^T(\mathbf{a}_1\hat{x} + \mathbf{a}_2\hat{y} - \mathbf{b}) = 0 \end{cases}$$

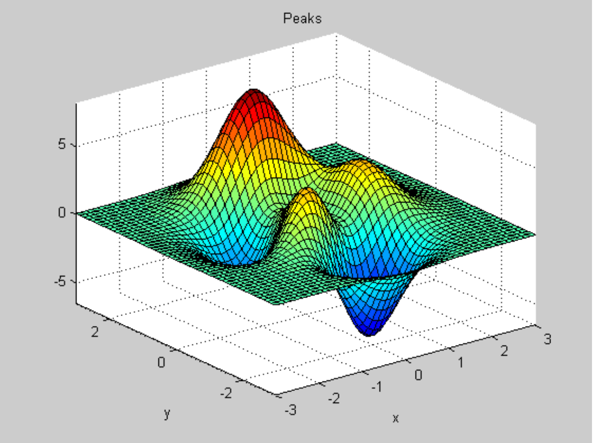
$$\Rightarrow A^T(A\hat{\mathbf{x}} - \mathbf{b}) = \mathbf{0}$$

$$\Rightarrow A^T A\hat{\mathbf{x}} = A^T\mathbf{b} \Rightarrow \hat{\mathbf{x}} = (A^T A)^{-1} A^T\mathbf{b}$$

$$\Rightarrow \text{Best projection} = A\hat{\mathbf{x}} = A\underbrace{(A^T A)^{-1} A^T}_{\text{Pseudo inverse of A}}\mathbf{b}$$

Example of Surface Fitting（1/6）

- Recall that peaks gives a surface with 3 local minimums and 3 local maximums



- Let' s cheat, and assume that we know peaks is generated using this function:

$$z = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$$

Example of Surface Fitting（2/6）

- That is, we assume the basis functions are known; in addition, we assume that training data contain zero-mean unit-variance Gaussian noise
- So our training data can be written as

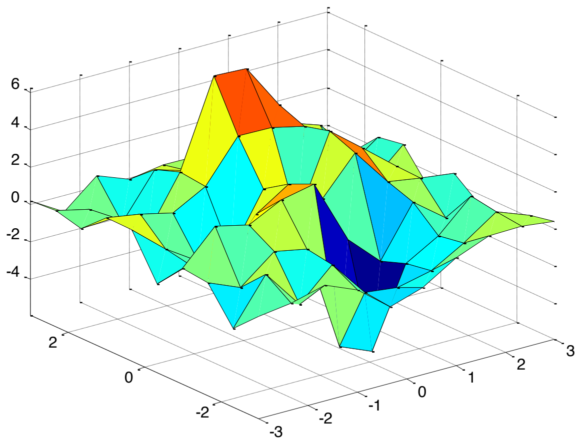
$$
\begin{aligned}
z &= 3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2} + n \\
&= 3f_1(x,y) - 10f_2(x,y) - \frac{1}{3}e^{-(x+1)^2-y^2} + n \\
&= \theta_1 f_1(x,y) + \theta_2 f_2(x,y) + \theta_3 f_3(x,y) + n
\end{aligned}
$$

  - where θ_1, θ_2 ,and θ_3 are unknowns and n is the Gaussian noise

Example of Surface Fitting（3/6）

- Let’ s generate some training data:

```
pointNum = 10;
[xx, yy, zz] = peaks(pointNum);
zz = zz + randn(size(zz));
surf(xx, yy, zz);
axis tight
```



- Notice that the resulting surface of training data is quite different from the original one generated by peaks ← but we will still figure out the unknowns

Example of Surface Fitting（4/6）

- Let's use the assumed basis functions to find the best θ_1, θ_2 , and θ_3

```
pointNum = 10;
[xx, yy, zz] = peaks(pointNum);
zz = zz + randn(size(zz))/10;
x = xx(:);
y = yy(:);
z = zz(:);
A = [(1-x).^2.*exp(-(x.^2)-(y+1).^2), (x/5-x.^3-y.^5).*exp(-x.^2-y.^2), exp(-(x+1).^2-y.^2)];
theta = A\z   % The backslash trick!
```

```
theta =
    3.0021
   -9.9764
   -0.4387
```

  - The resulting theta values are close to $\left(3, -10, -\frac{1}{3}\right)$
  - Run it multiple times, what you observe? Why?

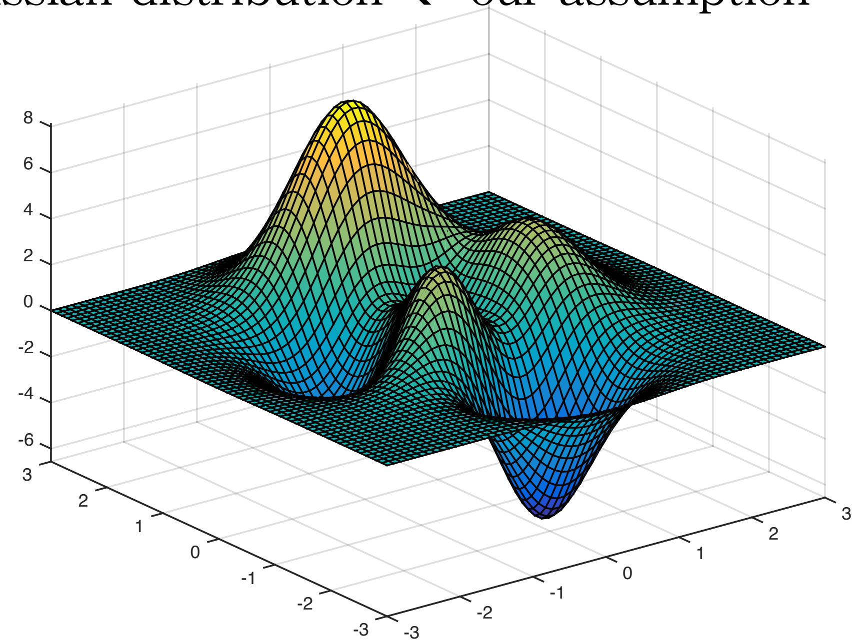
Example of Surface Fitting（5/6）

- Let’ s next plot the derived model surface

```
pointNum = 10;
[xx, yy, zz] = peaks(pointNum);
zz = zz + randn(size(zz))/10;
x = xx(:); y = yy(:); z = zz(:);
A = [(1-x).^2.*exp(-(x.^2)-(y+1).^2), (x/5-x.^3-y.^5).*exp(-x.^2-y.^2), exp(-
(x+1).^2-y.^2)];
theta = A\z;
pointNum = 64;
[xx, yy] = meshgrid(linspace(-3, 3, pointNum), linspace(-3, 3, pointNum));
x = xx(:); y = yy(:);
A = [(1-x).^2.*exp(-(x.^2)-(y+1).^2), (x/5-x.^3-y.^5).*exp(-x.^2-y.^2), exp(-
(x+1).^2-y.^2)];
zz = reshape(A*theta, pointNum, pointNum);
surf(xx, yy, zz);
axis tight
```

Example of Surface Fitting（6/6）

- The resulting surface follows the original peaks function closely
- The least-squared fitting works when if
  - The basis functions are correct ← our assumption #1
  - The noise term follows Gaussian distribution ← our assumption #2



Non-Linear Regression

- Nonlinear regression is harder because
  - Cannot find the optimal solution in one step (analytic or closed-form solution) ← iterative approaches?
  - May not even know where is the optimal solution
  - Have to leverage non-linear optimization algorithms
  - Usually don't have clear mathematic properties
- Mathematically, we write the model as $y = f(\vec{x}, \vec{\theta})$
  - Where $\vec{x}$ is the input vector, $\vec{\theta}$ is the vector of non-linear functions, and y is the output vector
  - The total squared error is: $E(\vec{\theta}) = \sum_{i=1}^{m} \left[y_i - f(\vec{x}_i, \vec{\theta}) \right]^2$

Minimizing the Error

- Apply mathematic optimization algorithms to minimize the error $E(\vec{\theta})$ (objective function)
  - Gradient Descent
  - Simplex Downhill Search ← adopted by fminsearch
- Example of math model: $y = a_1 e^{\lambda_1 x} + a_2 e^{\lambda_2 x}$
  - Where a_1, a_2 are linear parameters, but λ_1, λ_2 are nonlinear

$$E(a_1, a_2, \lambda_1, \lambda_2) = \sum_{i=1}^{m} \left(y_i - a_1 e^{\lambda_1 x_i} + a_2 e^{\lambda_2 x_2} \right)^2$$

  - Total squared error
  - Goal: write E(.) as a function of a_1, a_2, λ_1, λ_2; then minimize E(.)

Example of fminsearch (1/3)

- Create a function: errorMeasure1.m

```
function squaredError = errorMeasure1(theta, data)
x = data(:,1);
y = data(:,2);
y2 = theta(1)*exp(theta(3)*x)+theta(2)*exp(theta(4)*x);
squaredError = sum((y-y2).^2);
```

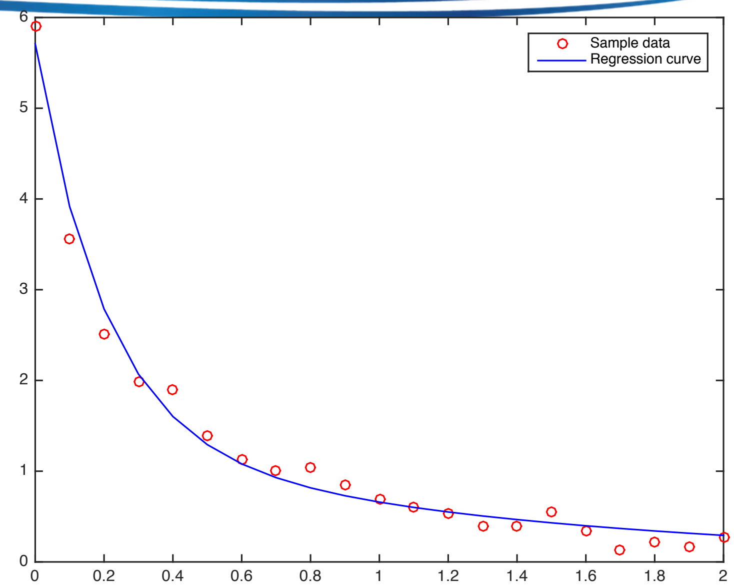
- theta is the vector of all parameters, containing a_1, a_2, λ_1, λ_2
- data are the training data points
- return value is the total squared error

Example of fminsearch (2/3)

```
load data.txt
theta0 = [0 0 0 0];
tic
theta = fminsearch(@errorMeasure1, theta0, [], data);
fprintf('running time = %g\n', toc);
x = data(:, 1);
y = data(:, 2);
y2 = theta(1)*exp(theta(3)*x)+theta(2)*exp(theta(4)*x);
plot(x, y, 'ro', x, y2, 'b-');
legend('Sample data', 'Regression curve');
fprintf('total squared error = %d\n', sum((y-y2).^2));
```

Example of fminsearch (3/3)

running time = 0.0435498
total squared error = 5.337871e-01



- The fitted curve is created by fminsearch
- fminsearch implements Simplex Downhill Search algorithm
- We use it to find the minimum value of E(.) for optimal theta values

Enhancing the Above Algorithm

- We treat all parameters in $y = a_1 e^{\lambda_1 x} + a_2 e^{\lambda_2 x}$ as nonlinear parameters!
- Hybrid method: uses different algorithm for linear and non-linear parameters
  - Linear parameters: use least squared error, or backslash
  - Non-linear parameters: use Simplex Downhill Search $\leftarrow$ fminsearch
- Why hybrid? Number of variables for fminsearch is largely reduced from 4 to 2

Example of Hybrid Approach（1/3）

- New error measure function: errorMeasure2.m

```
function squaredError = errorMeasure2(lambda, data)
x = data(:,1);
y = data(:,2);
A = [exp(lambda(1)*x) exp(lambda(2)*x)];
a = A\y;
y2 = a(1)*exp(lambda(1)*x)+a(2)*exp(lambda(2)*x);
squaredError = sum((y-y2).^2);
```

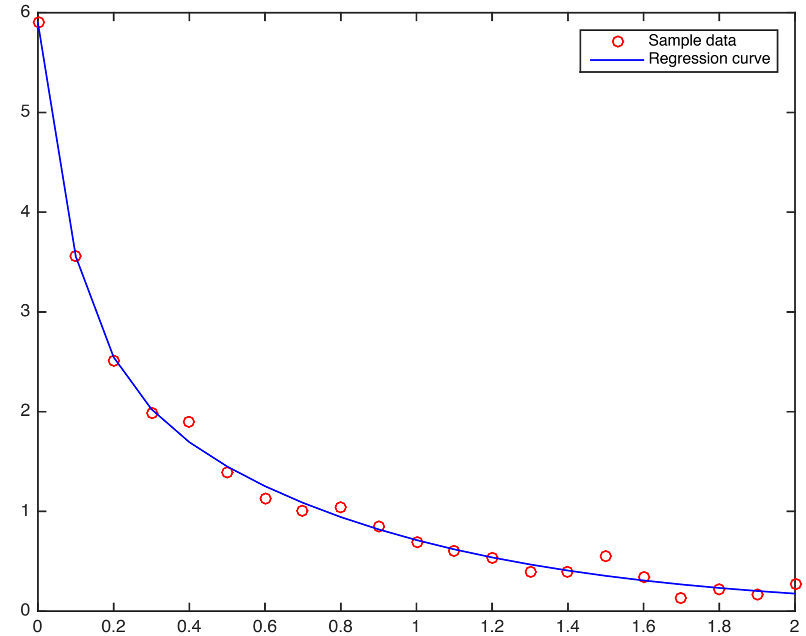
  - lambda are the vector nonlinear parameters
  - other aspects are not changed

Example of Hybrid Approach（2/3）

```
load data.txt
lambda0 = [0 0];
tic
lambda = fminsearch(@errorMeasure2, lambda0, [], data);
fprintf('running time = %g\n', toc);
x = data(:, 1);
y = data(:, 2);
A = [exp(lambda(1)*x) exp(lambda(2)*x)];
a = A\y;
y2 = A*a;
plot(x, y, 'ro', x, y2, 'b-');
legend('Sample data', 'Regression curve');
fprintf('total squared error = %d\n', sum((y-y2).^2));
```

Example of Hybrid Approach（3/3）

running time = 0.0363858
total squared error = 1.477226e-01



– Smaller total squared error and shorter running time

Transformation

- Approach: Let's transform a nonlinear math model into a linear one!
- Consider a sample function $y = ae^{bx}$
- Take a natural log, we have $\ln y = \ln a + bx$
- Let' s consider ln(a) and b as our parameters, since they are linear, we can apply least squared error algorithm

```
load data2.txt
x = data2(:, 1);
y = data2(:, 2);
A = [ones(size(x)) x];
```

Data2.txt

0.0000000e+000	4.8294773e+000
1.0000000e-001	4.1213066e+000
2.0000000e-001	3.3910515e+000
3.0000000e-001	2.7342027e+000
4.0000000e-001	2.2642898e+000
5.0000000e-001	1.6556118e+000
6.0000000e-001	1.3557419e+000
7.0000000e-001	1.3149045e+000
8.0000000e-001	9.8602581e-001
9.0000000e-001	6.6333465e-001
1.0000000e+000	6.4488254e-001
1.1000000e+000	4.7438692e-001
1.2000000e+000	5.2266978e-001
1.3000000e+000	3.6716688e-001
1.4000000e+000	3.3645444e-001
1.5000000e+000	2.9958098e-001
1.6000000e+000	1.0095206e-001
1.7000000e+000	1.7680898e-001
1.8000000e+000	1.2498356e-001
1.9000000e+000	1.8077775e-001
2.0000000e+000	2.7990727e-001

Example of Transformation（1/2）

```
theta = A\log(y);
subplot(2,1,1)
plot(x, log(y), 'o', x, A*theta); xlabel('x'); ylabel('ln(y)');
title('ln(y) vs. x');
legend('Actual value', 'Predicted value');
a = exp(theta(1))
b = theta(2)
y2 = a*exp(b*x);
subplot(2,1,2);
plot(x, y, 'o', x, y2); xlabel('x'); ylabel('y');
legend('Actual value', 'Predicted value');
title('y vs. x');
fprintf('total squared error = %d\n', sum((y-y2).^2));
```

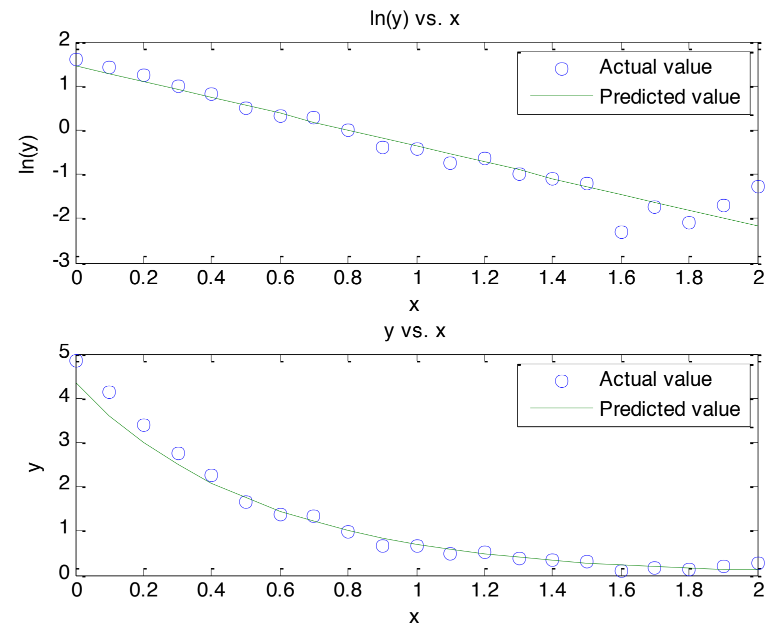
a = 4.3282

b =-1.8235

total squared error = 8.744185e-01

Example of Transformation（2/2）

- The top figure is ln(y) on x
- The bottom figure is y on x
- After transformation, the least squared approach gives the minimum of: $$E' = \sum_{i=1}^{m} \left(\ln y_i - \ln a - bx_i \right)^2$$
- Not the original one: $$E = \sum_{i=1}^{m} \left(y_i - ae^{bx_i} \right)^2$$
- Minimum E' doesn't mean minimum E; but they should be close! ← what if we are picky?



Revised Transformation（1/3）

- If we really want to get the minimum E value, we can use the results from the transformation approach as the starting point, and the invoke fminsearch
- The error function looks like this

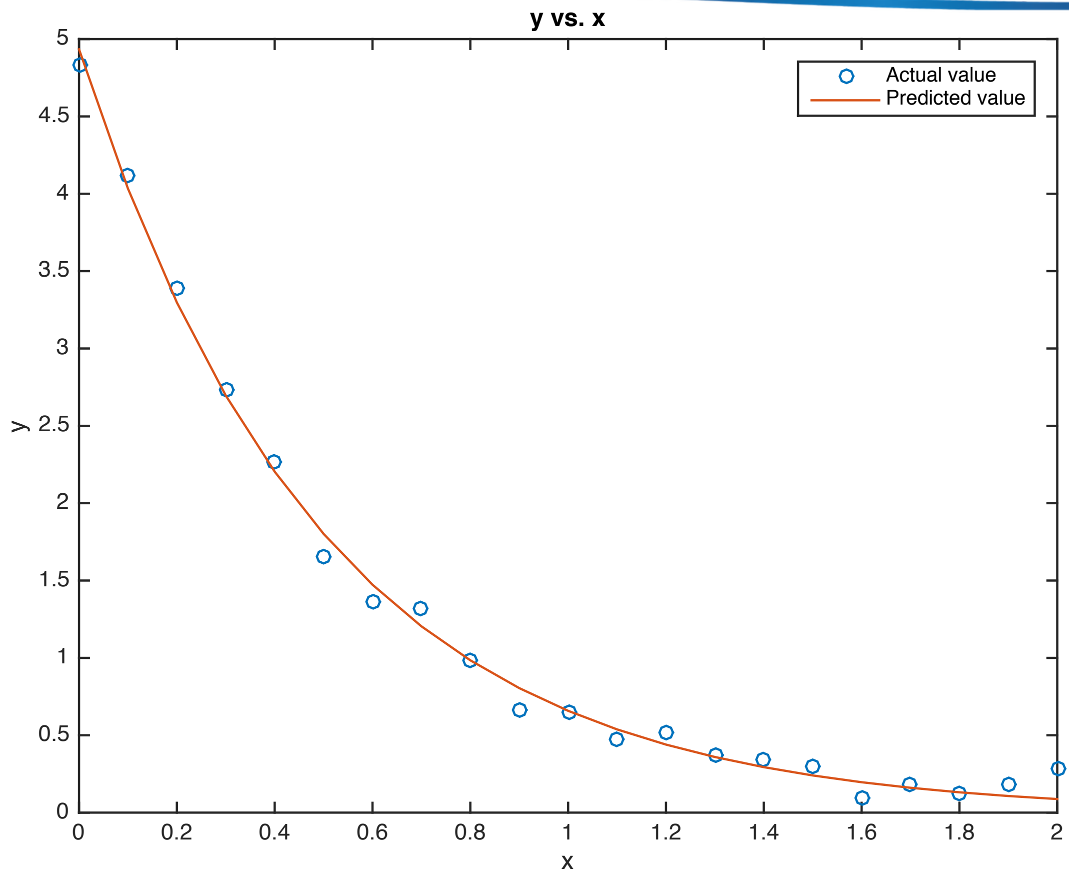
```
function squaredError = errorMeasure3(theta, data)
if nargin<1; return; end
x = data(:,1);
y = data(:,2);
y2 = theta(1)*exp(theta(2)*x);
squaredError = sum((y-y2).^2);
```

Revised Transformation（2/3）

```
load data2.txt
x = data2(:, 1);
y = data2(:, 2);
A = [ones(size(x)) x];
theta = A\log(y);
a = exp(theta(1))
b = theta(2)
theta0 = [a, b];
theta = fminsearch(@errorMeasure3, theta0, [], data2);
x = data2(:, 1);
y = data2(:, 2);
y2 = theta(1)*exp(theta(2)*x);
plot(x, y, 'o', x, y2); xlabel('x'); ylabel('y');
legend('Actual value', 'Predicted value');
title('y vs. x');
fprintf('total square error = %d\n', sum((y-y2).^2));
```

total square error = 1.680455e-01

Revised Transformation (3/3)



- The resulting error is smaller than the ordinary transformation approach

Mapping for Transformation（1/3）

No.	Nonlinear Model	Transformed Model	Parameters
1	$y=\dfrac{ax}{1+bx}$	$\underbrace{\dfrac{1}{y}}_{Y}=\underbrace{\dfrac{1}{a}}_{\alpha}\dfrac{1}{x}+\underbrace{\dfrac{b}{a}}_{\beta}$	$a=\dfrac{1}{\alpha},\ b=\dfrac{\beta}{\alpha}$
2	$y=\dfrac{a}{x+b}$	$\underbrace{\dfrac{1}{y}}_{Y}=\underbrace{\dfrac{1}{a}}_{\alpha}x+\underbrace{\dfrac{b}{a}}_{\beta}$	$a=\dfrac{1}{\alpha},\ b=\dfrac{\beta}{\alpha}$
3	$y=\dfrac{ax}{x^2+b^2}$	$\underbrace{\dfrac{1}{y}}_{Y}=\underbrace{\dfrac{1}{a}}_{\alpha}x+\underbrace{\dfrac{b^2}{a}}_{\beta}\dfrac{1}{x}$	$a=\dfrac{1}{\alpha},\ b^2=\dfrac{\beta}{\alpha}$

Mapping for Transformation（2/3）

4	$y = ax^b$	$\underbrace{\ln y}_{Y} = \underbrace{b}_{\alpha} \ln x + \underbrace{\ln a}_{\beta}$	$a = e^{\beta}, b = \alpha$
5	$y = \dfrac{1}{1 + ax^b}$	$\underbrace{\ln\left(\dfrac{1-y}{y}\right)}_{Y} = \underbrace{b}_{\alpha} \ln x + \underbrace{\ln a}_{\beta}$	$a = e^{\beta}, b = \alpha$
6	$y = \dfrac{1}{1 + \exp\left(\dfrac{ax}{b+x}\right)}$	$\underbrace{\left[\ln\left(\dfrac{1-y}{y}\right)\right]^{-1}}_{Y} = \underbrace{\dfrac{b}{a}}_{\alpha} \dfrac{1}{x} + \underbrace{\dfrac{1}{a}}_{\beta}$	$a = \dfrac{1}{\beta}, b = \dfrac{\alpha}{\beta}$
7	$y =$ $\ln a + x - \ln\left(e^x + b\right)$	$\underbrace{e^{-y}}_{Y} = \underbrace{\dfrac{b}{a}}_{\alpha} e^{-x} + \underbrace{\dfrac{1}{a}}_{\beta}$	$a = \dfrac{1}{\beta}, b = \dfrac{\alpha}{\beta}$

Mapping for Transformation（3/3）

8	$\dfrac{x^2}{a^2}+\dfrac{y^2}{b^2}=1$	$\underbrace{y^2}_{Y}=\underbrace{-\dfrac{b^2}{a^2}}_{\alpha}x^2+\underbrace{b^2}_{\beta}$	$a^2=-\dfrac{\beta}{\alpha},\ b^2=\beta$
9	$y=$ $a\exp\left[-\left(\dfrac{x-c}{b}\right)^2\right]$	$\underbrace{\ln y}_{Y}=\underbrace{-\dfrac{1}{b^2}}_{\alpha}x^2+\underbrace{\dfrac{2c}{b^2}}_{\beta}x+\underbrace{\ln a-\dfrac{c^2}{b^2}}_{\gamma}$	$a=\exp\left(\gamma-\dfrac{\beta^2}{4}\right),$ $b=\pm\sqrt{-\dfrac{1}{\alpha}},$ $c=-\dfrac{\beta}{2\alpha}$
10	$y=$ $\dfrac{a}{\sqrt{\left(1+bx^2\right)^2+c}}$	$\underbrace{\dfrac{1}{y^2}}_{Y}=\underbrace{\dfrac{b^2}{a^2}}_{\alpha}x^4+\underbrace{\dfrac{2b}{a^2}}_{\beta}x^2+\underbrace{\dfrac{c+1}{a^2}}_{\gamma}$	$a=\pm\dfrac{\sqrt{4\alpha}}{\beta},\ b=\dfrac{2\alpha}{\beta},$ $c=\dfrac{4\alpha\gamma}{\beta^2}-1$

Circle & Ellipse Fitting

- Circle fitting
- Ellipse fitting

$$(x-a)^2 + (y-b)^2 = c^2$$

$$\Rightarrow x^2 - 2ax + a^2 + y^2 - 2by + b^2 = c^2$$

$$\Rightarrow 2ax + 2by + c^2 - a^2 - b^2 = x^2 + y^2$$

$$\Rightarrow [2x \; 2y \; 1]\begin{bmatrix} a \\ b \\ c^2 - a^2 - b^2 \end{bmatrix} = x^2 + y^2$$

$$\Rightarrow \begin{bmatrix} 2x_1 & 2y_1 & 1 \\ \vdots & \vdots & \vdots \\ 2x_i & 2x_i & 1 \\ \vdots & \vdots & \vdots \\ 2x_n & 2y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c^2 - a^2 - b^2 \end{bmatrix} = \begin{bmatrix} x_1^2 + y_1^2 \\ \vdots \\ x_i^2 + y_i^2 \\ \vdots \\ x_n^2 + y_n^2 \end{bmatrix}$$

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1$$

$$\Rightarrow [x^2 \; y^2]\begin{bmatrix} 1/a^2 \\ 1/b^2 \end{bmatrix} = 1$$

$$\Rightarrow \begin{bmatrix} x_1^2 & y_1^2 \\ \vdots & \vdots \\ x_i^2 & y_i^2 \\ \vdots & \vdots \\ x_n^2 & y_n^2 \end{bmatrix} \begin{bmatrix} 1/a^2 \\ 1/b^2 \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

Curve Fitting Tool (1/4)

- Curve fitting steps
  - Observe the data points, remove the outliers
  - Based on the data points and select mathematical models (and maybe parameters)
  - Using linear and nonlinear regression to derive the optimal parameters based on a set of training data
  - Use a set of test data to validate the quality of the derived model; if passes then stop, otherwise, try a different model and go back to step 2

Curve Fitting Tool (2/4)

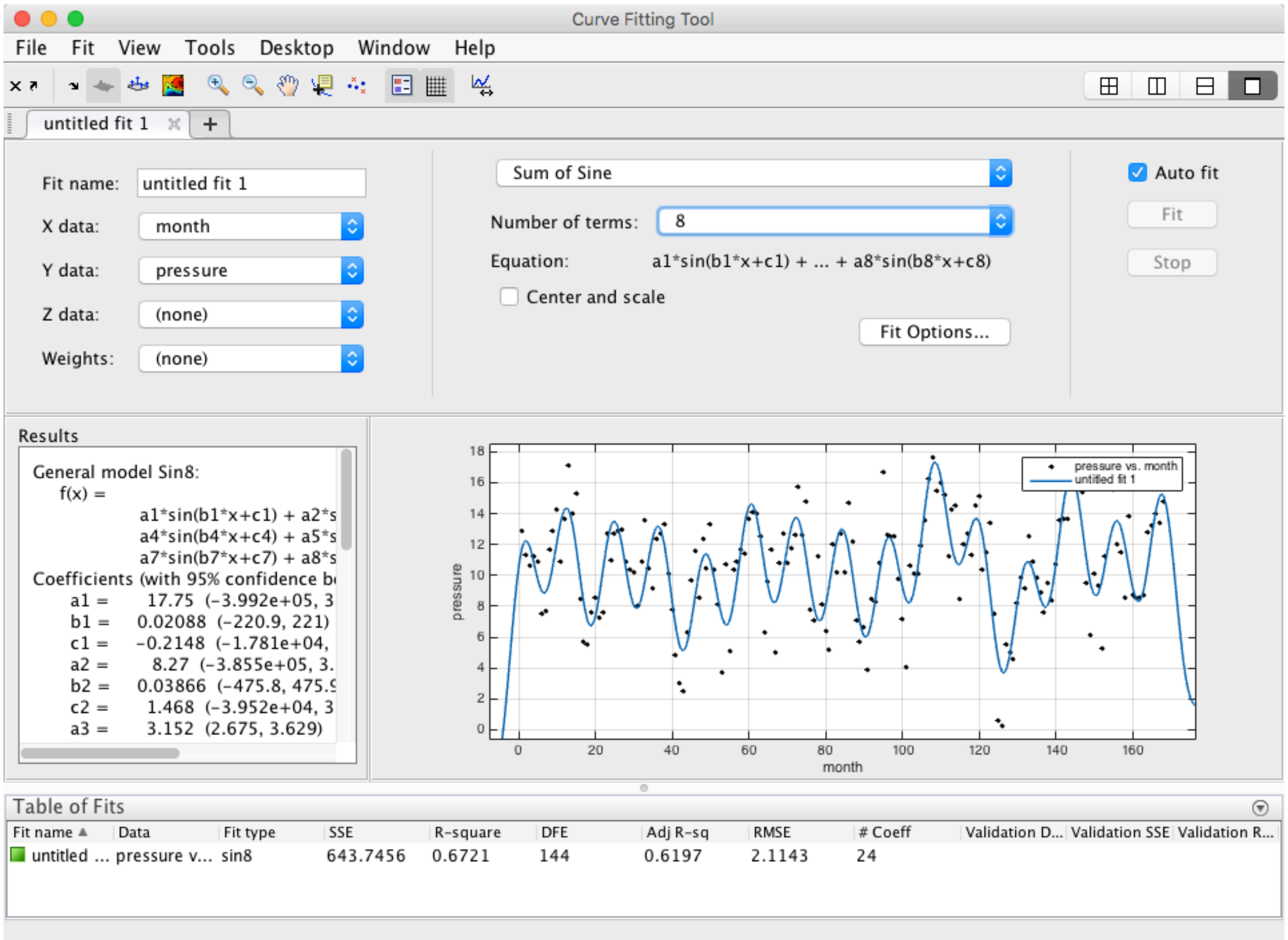
- The above steps takes time and extensive experience
- Curve fitting toolbox allows Matlab users to perform curve fitting in GUI and quickly check the fitting results and quality

Curve Fitting Tool (3/4)

- Example: first load enso.mat, which has two parameters
  - Month: the month when the measurements were taken
  - Pressure: the air pressure between Easter Island and Darwin ← some how this affects the Trade Winds in the South Hemisphere
- Load and launch the curve fitting toolbox

```
load enso.mat
cftool(month, pressure);
```

Curve Fitting Tool (4/4)



Matlab #12 Homework (M12)

- (3%) In this exercise, you need to write a MATLAB function that can select the order of a fitting polynomial based on the leave-one-out criterion described in the text. More specifically, you need to write a function polyOrderSelect.m with the following input/output format:

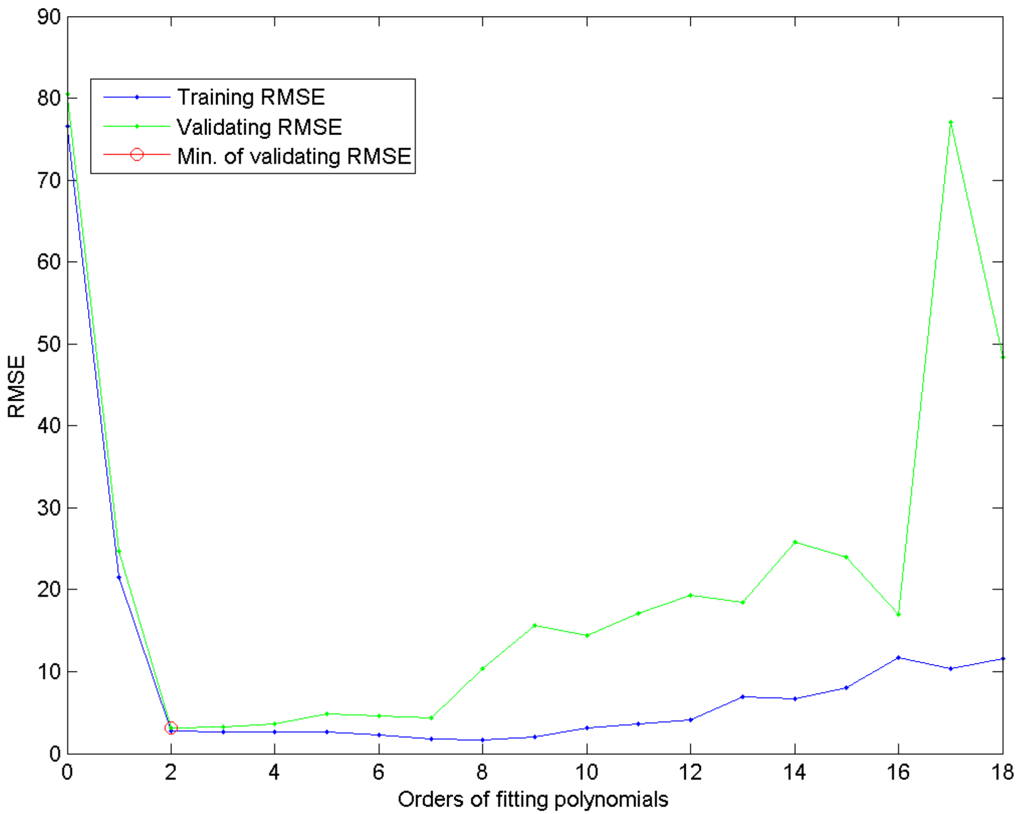
  bestOrder=polyOrderSelect(data, maxOrder, showPlot)

  where
  - "data" is the single-input-single-output dataset for the problem. The first row is the input data, while the second row is the corresponding output data.
  - "maxOrder" is the maximum order to be tried by the function. Note that maxOrder should not be greater than size(data,2)-1. (Why?)
  - "showPlot" is an option for plotting. If it is 1, the function plots the training and validating RMSE with respect to the order. Otherwise there is no plotting.
  - "bestOrder" is the order that generates the minimum validating RMSE.

  Please use your function on the population dataset.

Matlab #12 Homework (M12) cont.

Your first figure may look like this:



Matlab #12 Homework (M12) cont.

From the plot, you can observe that the training RMSE sometimes goes up when the order increases. This is not quite right, since as we have a higher order (which implies more model complexity and larger modelling power), the fitting error should be smaller.

Solution: Apply some form of normalization

1. Convert the input data to have zero sample mean and unit sample variance.
2. Scale the input data to be within [0, n], where n is a small integer.
3. Scale the input data to be within [-n, n], where n is a small integer.

Matlab #12 Homework (M12) cont.

Now apply one of the normalization approach to your training dataset, and replot the figure. Please write a short report to discuss what is the best order of polynomials your program has found.

Please submit the following files in ILMS without compressing them:

- Your .m file
- Your .eps file of the final figure
- Your .pdf report