Matlab 6: Data Fitting and Regression Analysis

Cheng-Hsin Hsu

National Tsing Hua University

Department of Computer Science

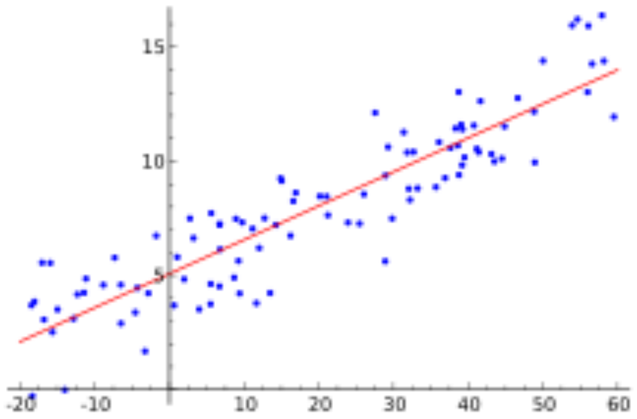
Slides and sample codes are based on the materials from Prof. Roger Jang

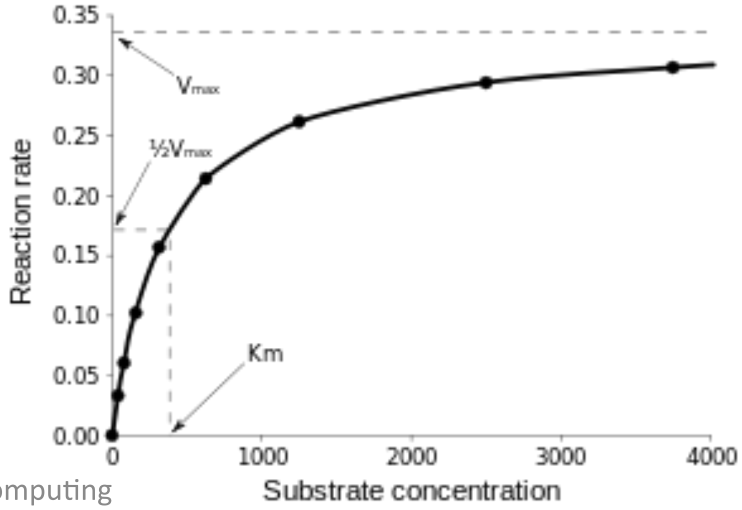
What is Data Fitting

- 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



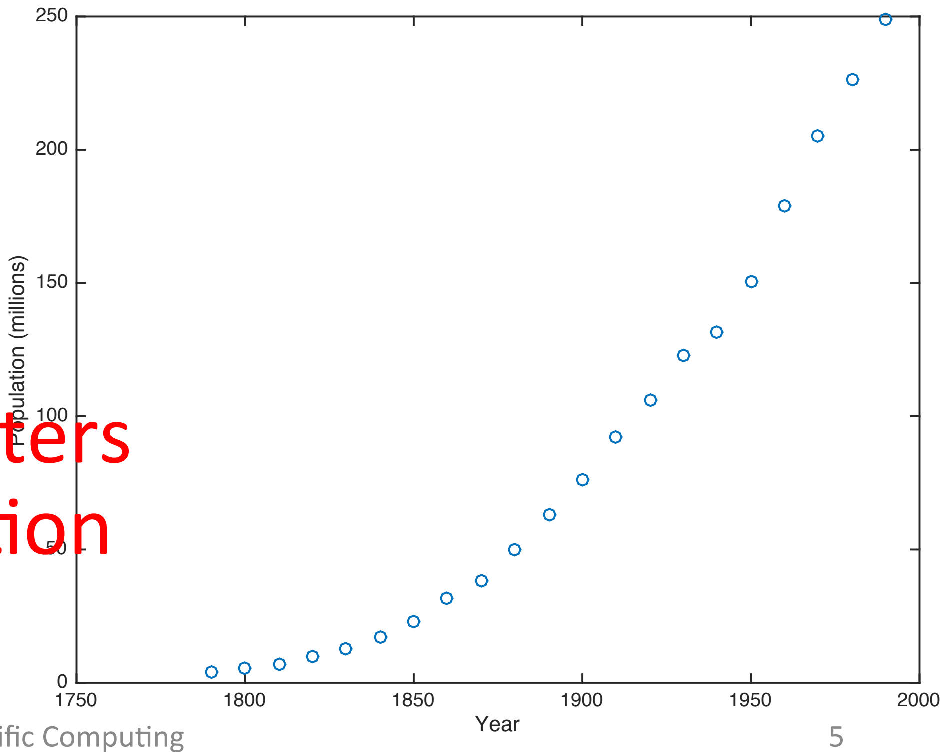


Data Fitting: Census Dataset

- 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

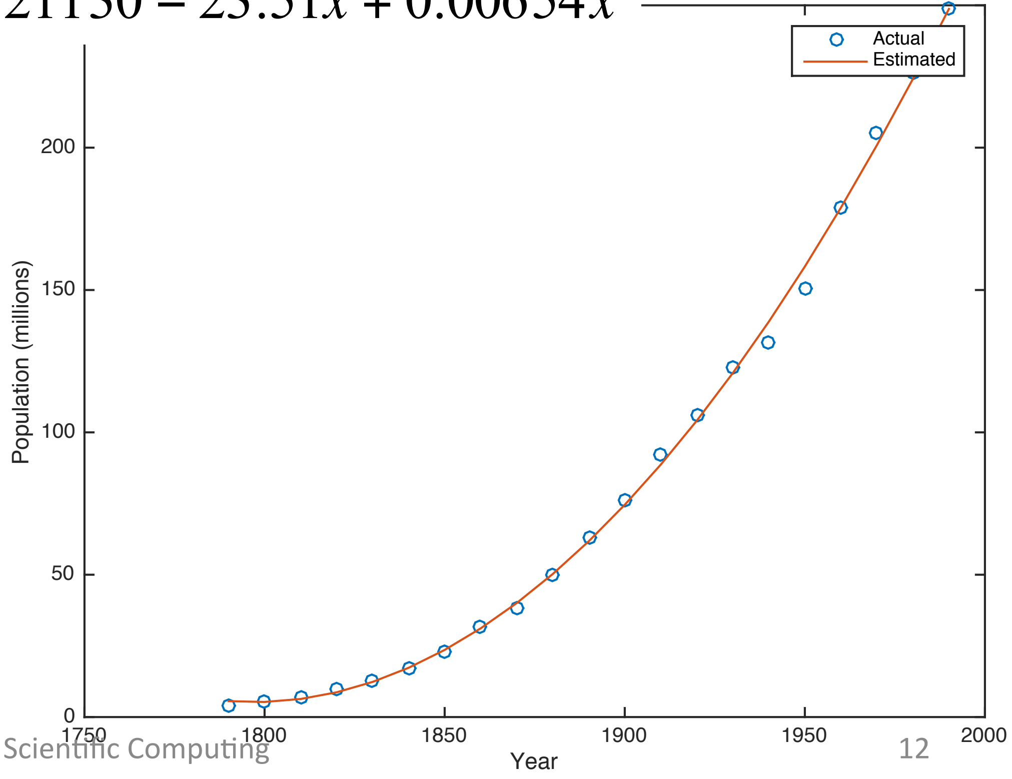
Data Fitting Example

```
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)');
```

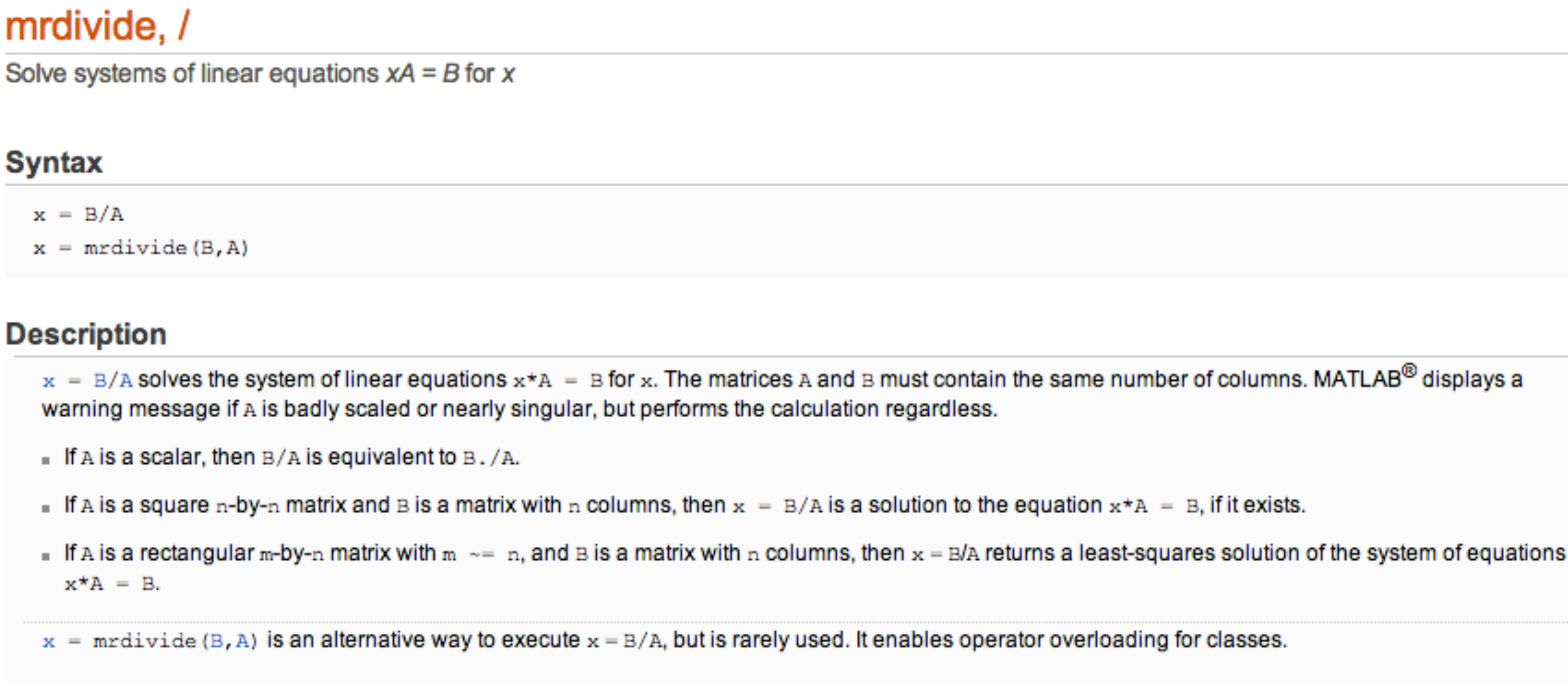
Data Fitting Results

- 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.

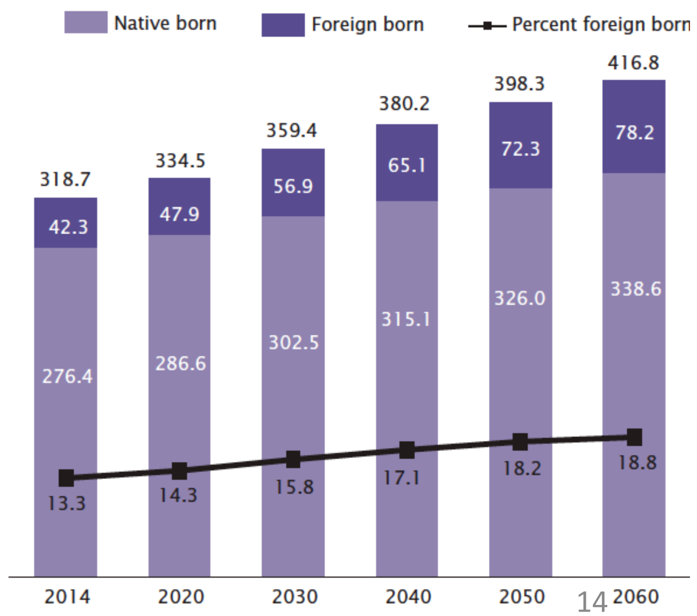
- If `A` is a scalar, then `B/A` is equivalent to `B./A`.
- If `A` is a square `n`-by-`n` matrix and `B` is a matrix with `n` columns, then `x = B/A` is a solution to the equation `x*A = B`, if it exists.
- If `A` is a rectangular `m`-by-`n` matrix with `m ~= n`, and `B` is a matrix with `n` columns,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

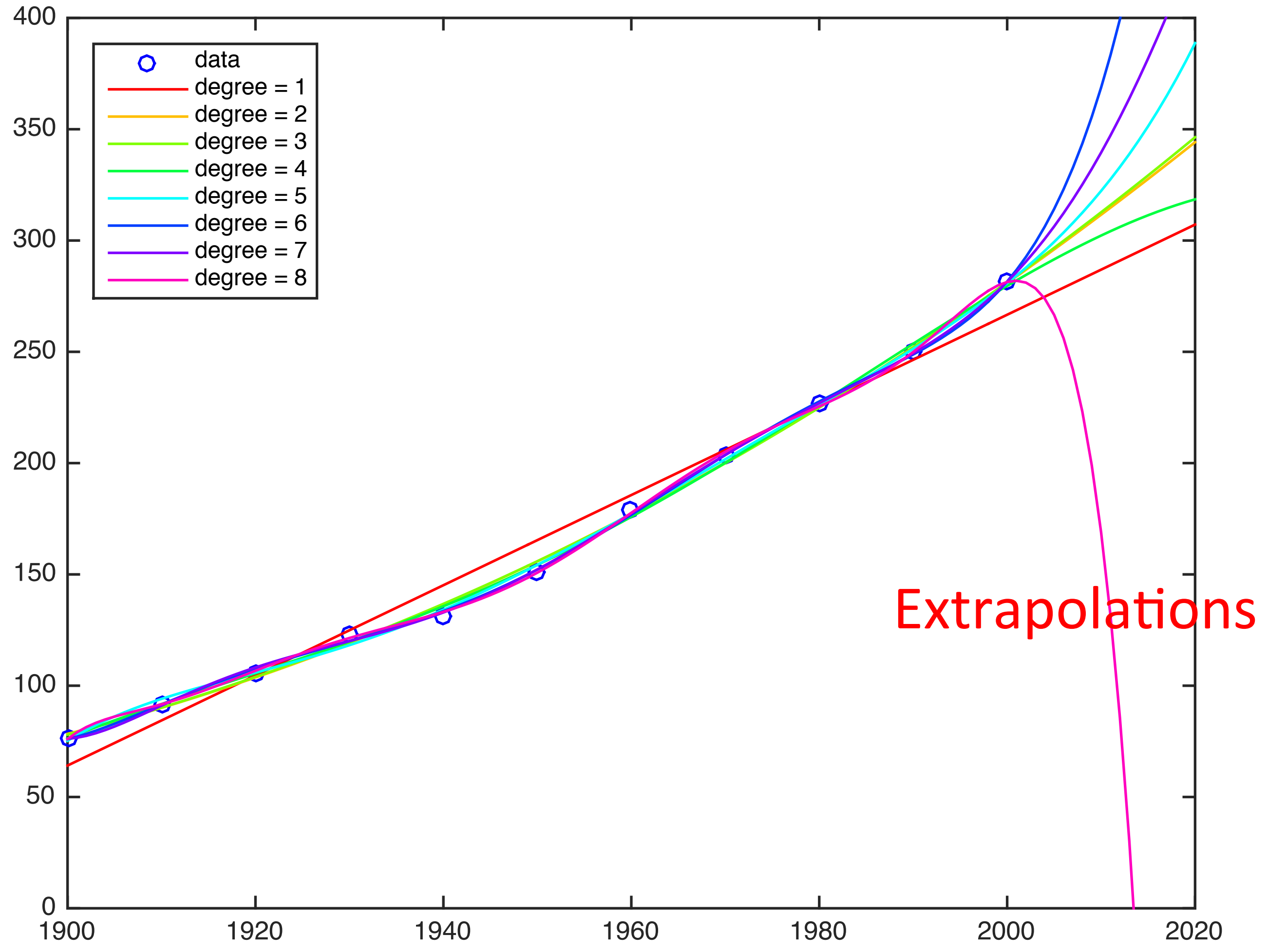
Using Polyfit and Polyval

- 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

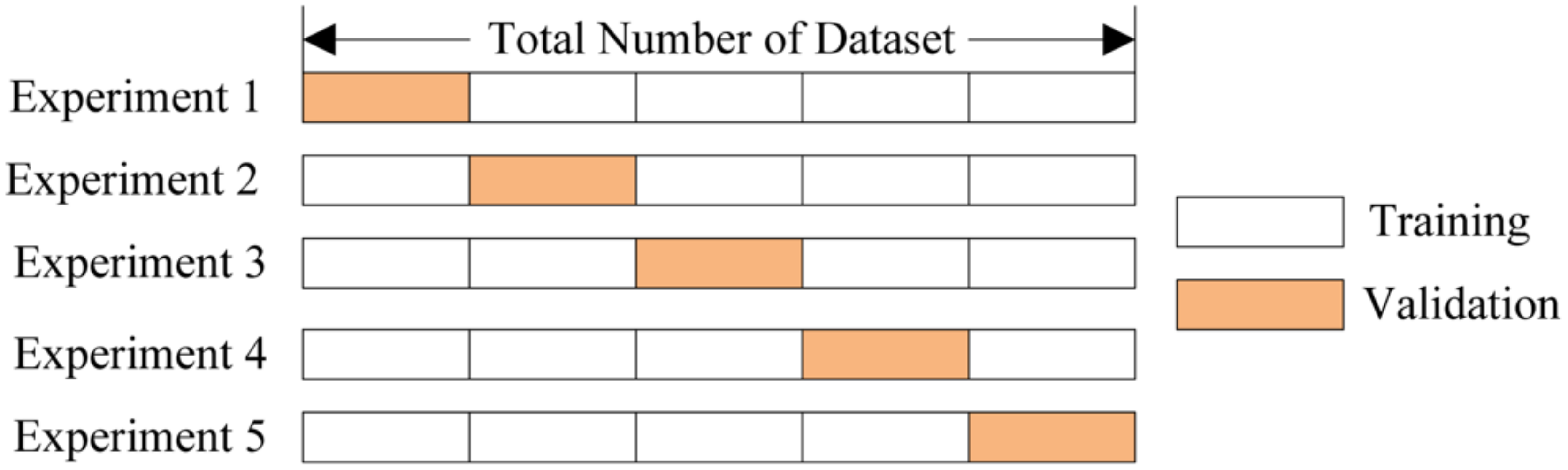


Model Complexity and Accuracy

- 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

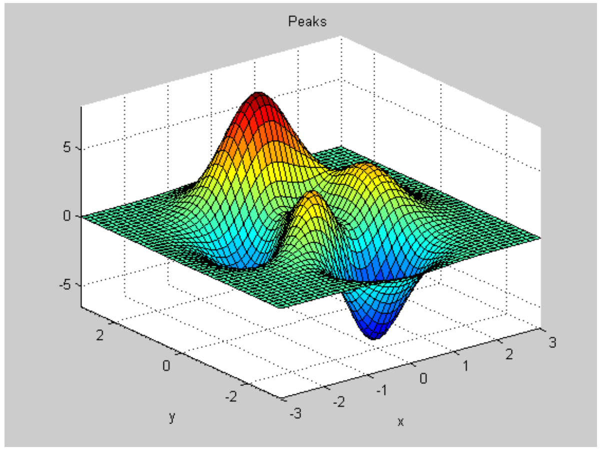
- Since $m > n$ (number of data points is more than number of parameters), we need to add an error vector e, so that : $A\boldsymbol{\theta} + \mathbf{e} = \mathbf{b}$
- Squared error: $E(\boldsymbol{\theta}) = \|\mathbf{e}\|^2 = \mathbf{e}^T\mathbf{e} = (\mathbf{b} - A\boldsymbol{\theta})^T(\mathbf{b}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

Sum of Squared Error (cont.)

- Optimal solutions
  - Using matrix operations, the optimal solution can be written as $\left(A^T A\right)^{-1} A^T \mathbf{b}$ ← derive later
  - 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

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

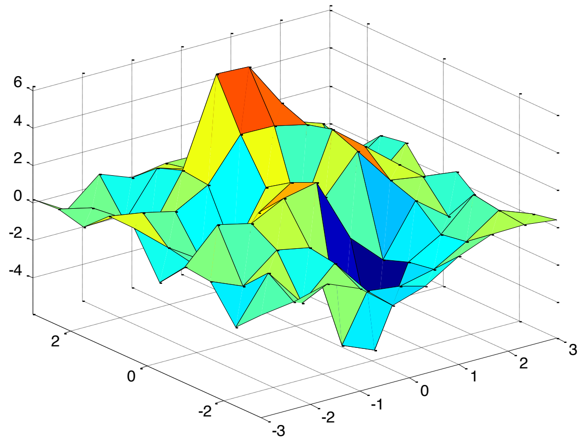
$$
\begin{aligned}
z &= 3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2} + n \\
&= 3f_1(x,y) - 10f_2(x,y) - \frac{1}{3}e^{-(x+1)^2-y^2} + n \\
&= \theta_1 f_1(x,y) + \theta_2 f_2(x,y) + \theta_3 f_3(x,y) + n
\end{aligned}
$$

  – 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?

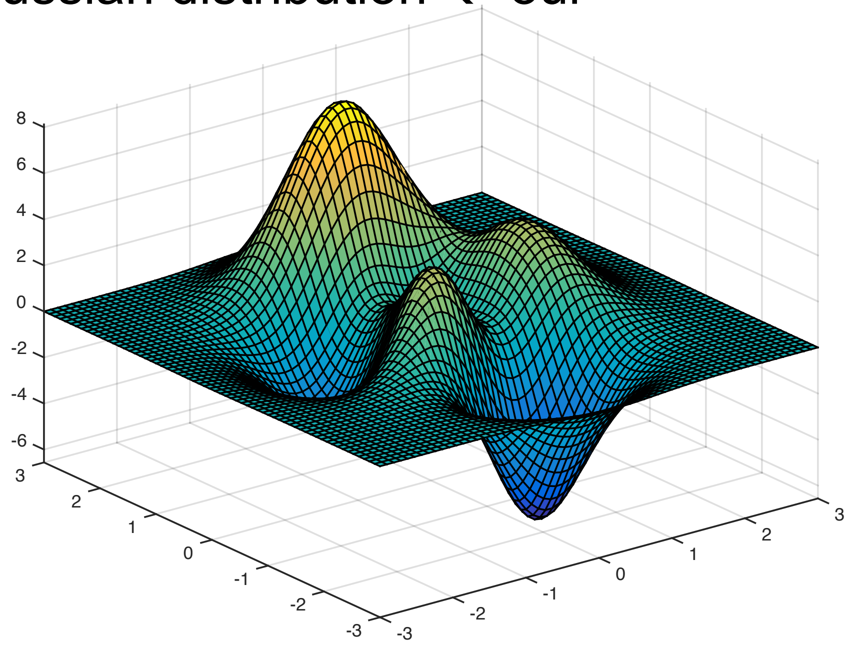
Example of Surface Fitting（5/6）

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

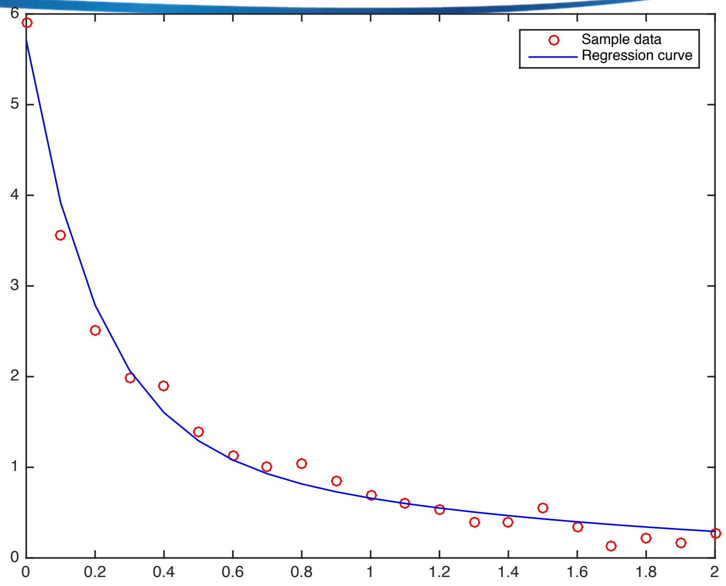
- theta is the vector of all parameters, containing a_1, a_2 , λ_1, λ_2
- data are the training data points
- return value is the total squared error

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);
```

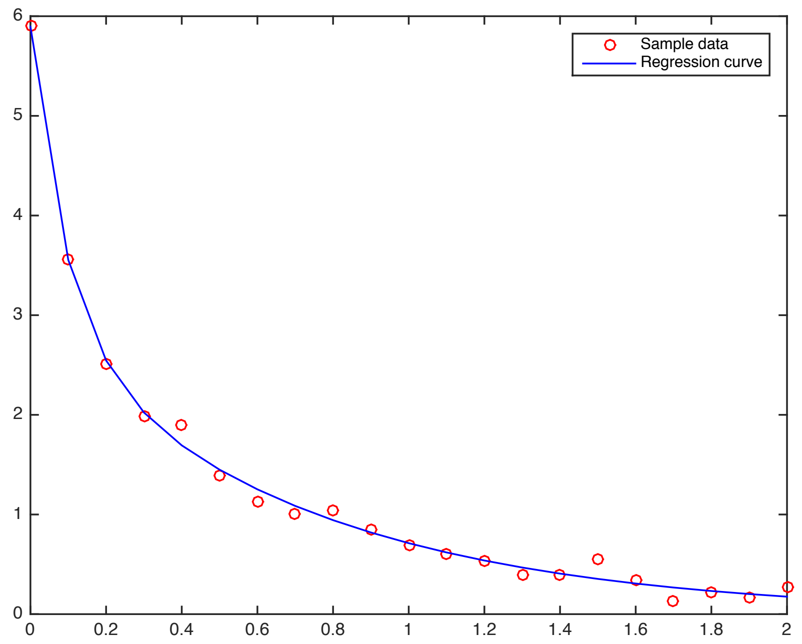
  – lambda are the vector nonlinear parameters
  – other aspects are not changed

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];
```

Note: The dataset file (data2.txt) can be downloaded from
http://mirlab.org/jang/books/matlabProgramming4guru/example/10-曲線擬合與迴歸分析/data2.txt

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)

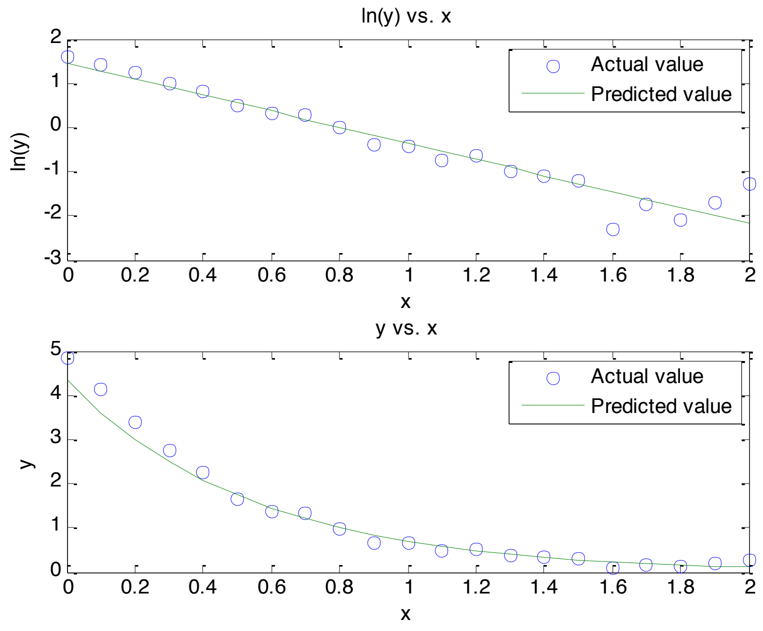
- The top figure is ln(y) on x
- The bottom figure is y on x
- After transformation, the least squared approach gives the minimum of:

$$E' = \sum_{i=1}^{m} \left(\ln y_i - \ln a - bx_i\right)^2$$

- 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

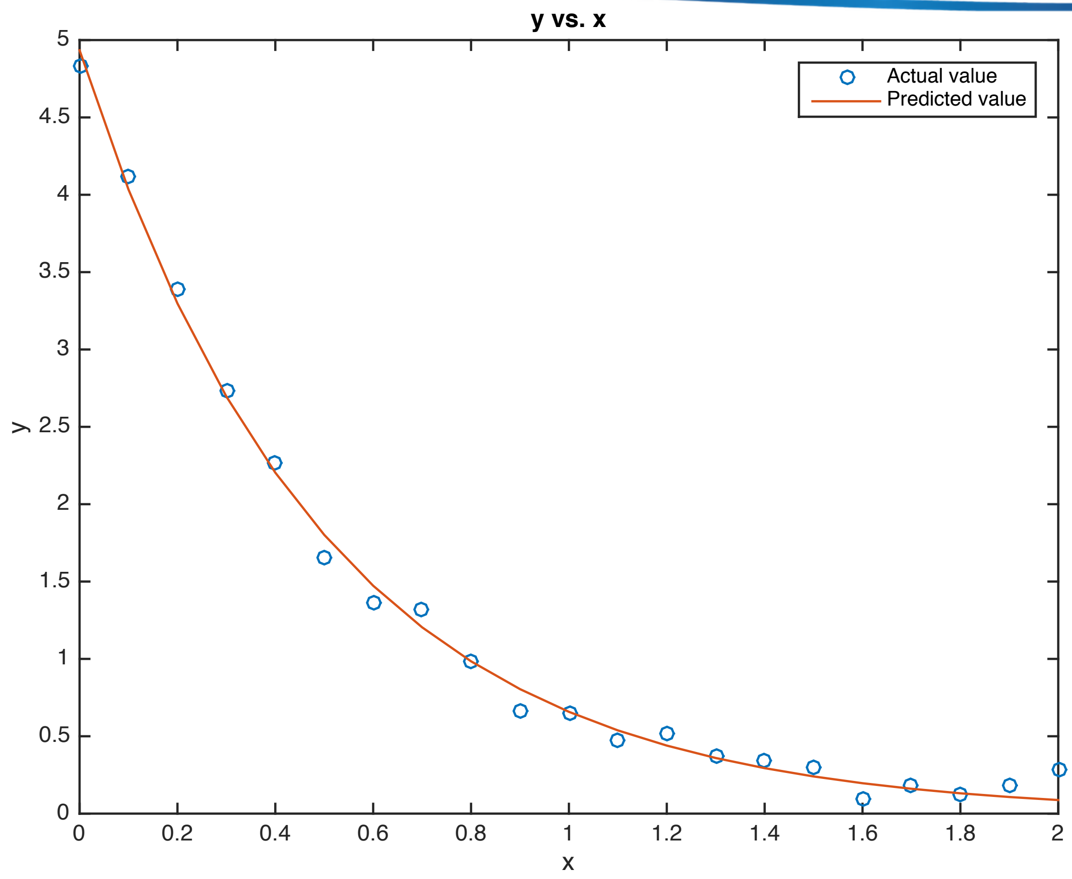
```
function squaredError = errorMeasure3(theta, data)
if nargin<1; return; end
x = data(:,1);
y = data(:,2);
y2 = theta(1)*exp(theta(2)*x);
squaredError = sum((y-y2).^2);
```

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	$y = \dfrac{ax}{1+bx}$	$\underbrace{\frac{1}{y}}_{Y} = \underbrace{\frac{1}{a}}_{\alpha}\frac{1}{x} + \underbrace{\frac{b}{a}}_{\beta}$	$a = \dfrac{1}{\alpha}, b = \dfrac{\beta}{\alpha}$
2	$y = \dfrac{a}{x+b}$	$\underbrace{\frac{1}{y}}_{Y} = \underbrace{\frac{1}{a}}_{\alpha}x + \underbrace{\frac{b}{a}}_{\beta}$	$a = \dfrac{1}{\alpha}, b = \dfrac{\beta}{\alpha}$
3	$y = \dfrac{ax}{x^2+b^2}$	$\underbrace{\frac{1}{y}}_{Y} = \underbrace{\frac{1}{a}}_{\alpha}x + \underbrace{\frac{b^2}{a}}_{\beta}\frac{1}{x}$	$a = \dfrac{1}{\alpha}, b^2 = \dfrac{\beta}{\alpha}$

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

Mapping for Transformation（3/3）

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

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)

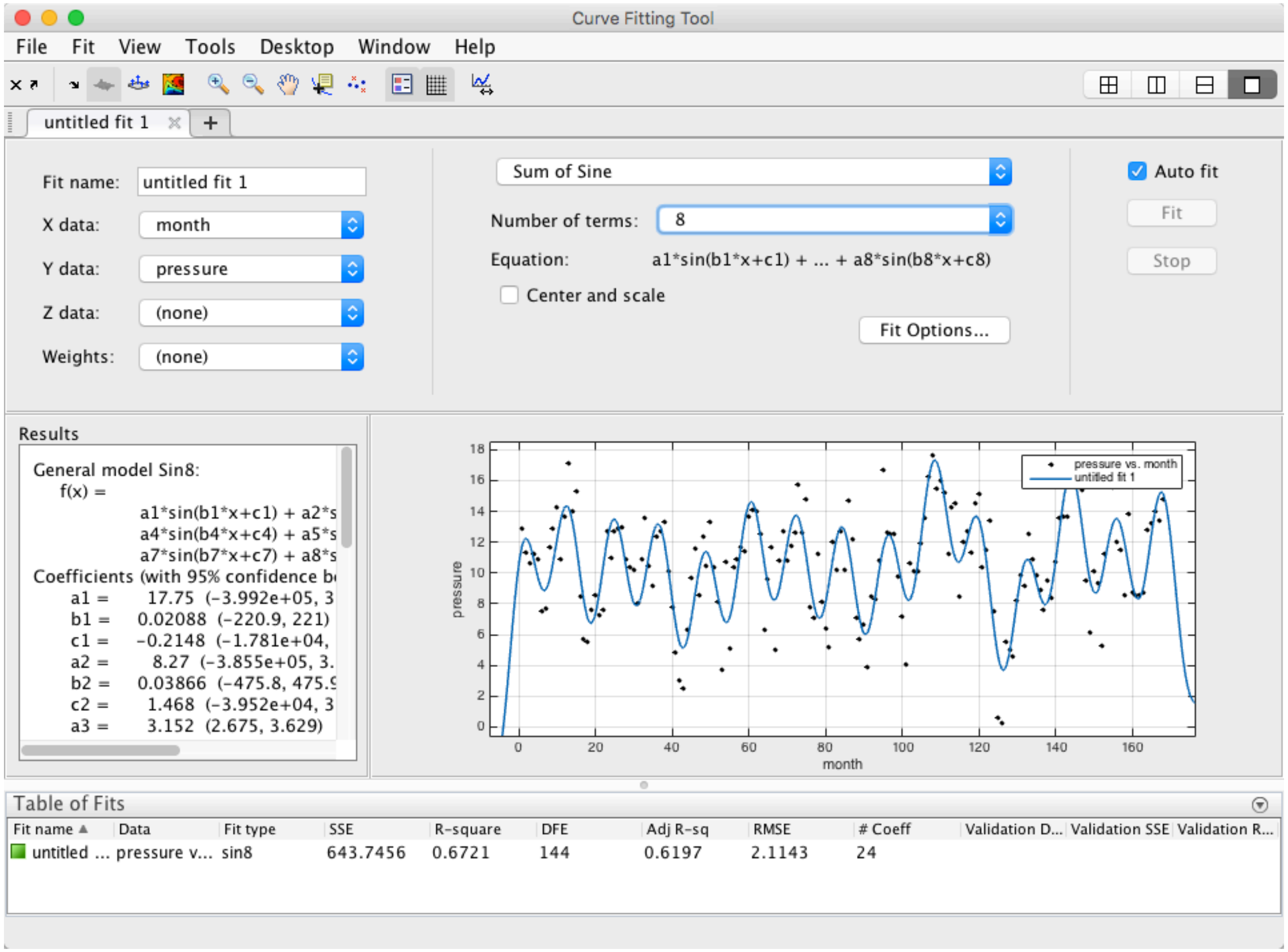
- The above steps takes time and extensive experience
- Curve fitting toolbox allows Matlab users to perform curve fitting in GUI and quickly check the fitting results and quality

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範例

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

Curve Fitting Tool (4/4)



Fit name: untitled fit 1

X data: month

Y data: pressure

Z data: (none)

Weights: (none)

Sum of Sine

Number of terms: 8

Equation: a1*sin(b1*x+c1) + ... + a8*sin(b8*x+c8)

☐ Center and scale

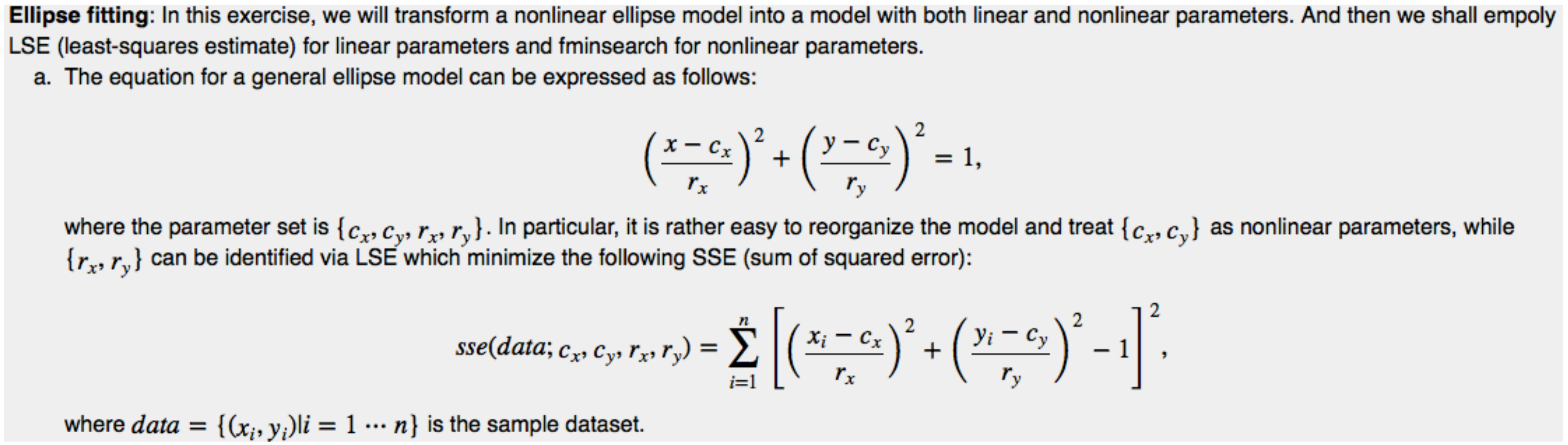
✓ Auto fit

```
General model Sin8:
   f(x) =
              a1*sin(b1*x+c1) + a2*s
              a4*sin(b4*x+c4) + a5*s
              a7*sin(b7*x+c7) + a8*s
Coefficients (with 95% confidence b
     a1 =      17.75  (-3.992e+05, 3
     b1 =    0.02088  (-220.9, 221)
     c1 =    -0.2148  (-1.781e+04,
     a2 =       8.27  (-3.855e+05, 3.
     b2 =    0.03866  (-475.8, 475.9
     c2 =      1.468  (-3.952e+04, 3
     a3 =      3.152  (2.675, 3.629)
```

Fit name ▲	Data	Fit type	SSE	R-square	DFE	Adj R-sq	RMSE	# Coeff	Validation D...	Validation SSE	Validation R...
🟩 untitled ...	pressure v...	sin8	643.7456	0.6721	144	0.6197	2.1143	24			

Matlab #6 Homework (M6)

- (3% + 1% Bonus) ← total



Ellipse fitting: In this exercise, we will transform a nonlinear ellipse model into a model with both linear and nonlinear parameters. And then we shall empoly LSE (least-squares estimate) for linear parameters and fminsearch for nonlinear parameters.

a. The equation for a general ellipse model can be expressed as follows:

$$\left(\frac{x-c_x}{r_x}\right)^2+\left(\frac{y-c_y}{r_y}\right)^2=1,$$

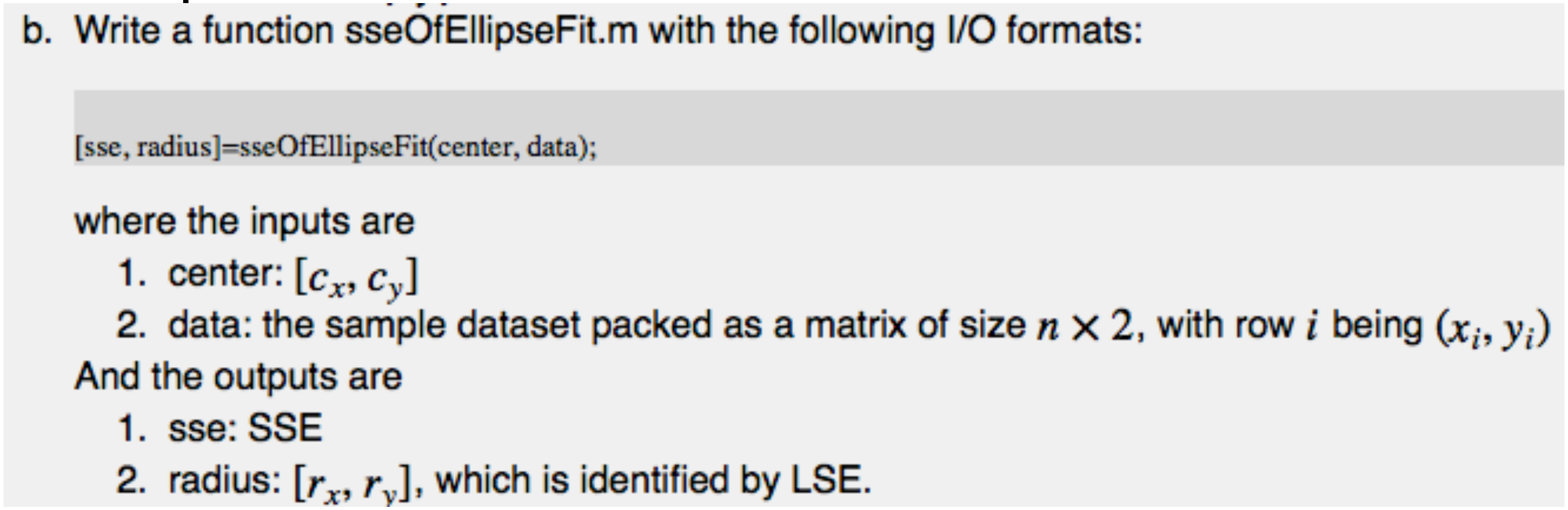
where the parameter set is $\{c_x, c_y, r_x, r_y\}$. In particular, it is rather easy to reorganize the model and treat $\{c_x, c_y\}$ as nonlinear parameters, while $\{r_x, r_y\}$ can be identified via LSE which minimize the following SSE (sum of squared error):

$$sse(data; c_x, c_y, r_x, r_y)=\sum_{i=1}^{n}\left[\left(\frac{x_i-c_x}{r_x}\right)^2+\left(\frac{y_i-c_y}{r_y}\right)^2-1\right]^2,$$

where $data=\{(x_i, y_i)|i=1\cdots n\}$ is the sample dataset.

Matlab #6 Homework (M6) cont.

– (1%) Present your function to the TA to get this point



b. Write a function sseOfEllipseFit.m with the following I/O formats:

```
[sse, radius]=sseOfEllipseFit(center, data);
```

where the inputs are

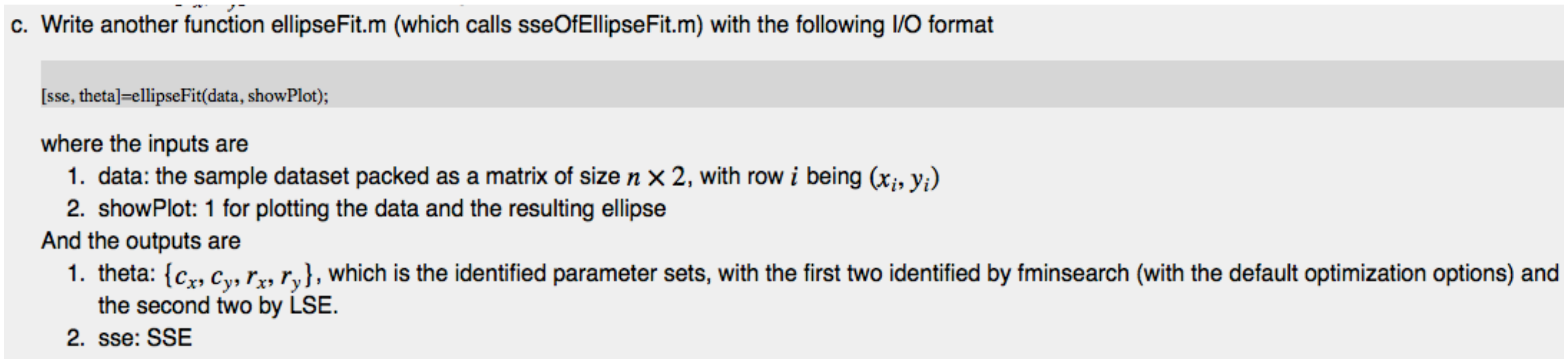
1. center: $[c_x, c_y]$
2. data: the sample dataset packed as a matrix of size $n \times 2$, with row i being (x_i, y_i)

And the outputs are

1. sse: SSE
2. radius: $[r_x, r_y]$, which is identified by LSE.

Matlab #6 Homework (M6) cont.

– (1%) Present your function to the TA to get this point



c. Write another function ellipseFit.m (which calls sseOfEllipseFit.m) with the following I/O format

```
[sse, theta]=ellipseFit(data, showPlot);
```

where the inputs are

1. data: the sample dataset packed as a matrix of size $n \times 2$, with row i being (x_i, y_i)
2. showPlot: 1 for plotting the data and the resulting ellipse

And the outputs are

1. theta: $\{c_x, c_y, r_x, r_y\}$, which is the identified parameter sets, with the first two identified by fminsearch (with the default optimization options) and the second two by LSE.
2. sse: SSE

Matlab #6 Homework (M6) cont.

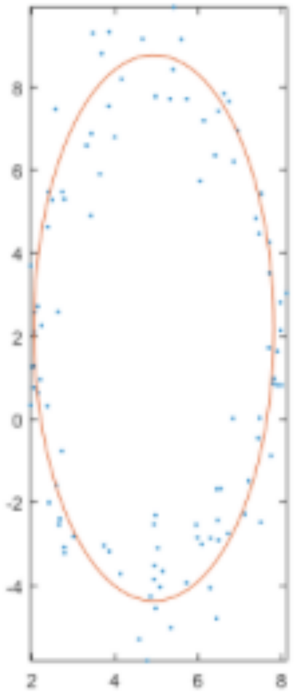
- (1%) Create your own test file, in which each line presents a sample point x and y real values separated by a single space. Invoke your function like this:

  data=load('ellipse.txt');

  [theta, sse]=ellipseFit(data, 1)

  We want to see the outputs of theta and sse.

  Also please a figure like this:



Matlab #6 Homework (M6) cont.

– (1% Bonus) TA will run your code with another sample dataset. You get one point if you code doesn't crash and generate the expected results, including theta, sse, and figure. No partial credit will be given in this bonus point.