Applied Programming

Curve Fitting:

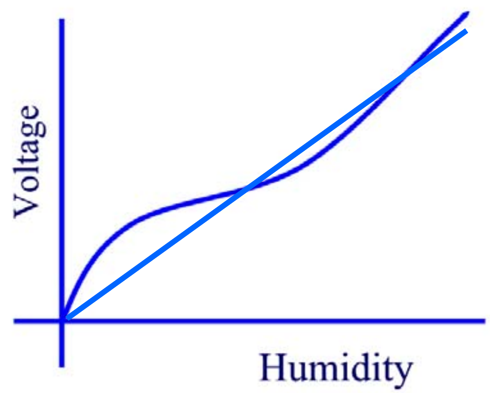
Least Squares Approximation

(Regression)

More details in: U. Ascher and C. Grief, "A First Course in Numerical Methods", chapter 6

Motivation

- Suppose that you have to calibrate a humidity sensor
  - The scale of the sensor is from 0 to 100%
- The sensor is supposed to output a linear voltage proportional to the ambient humidity
  - The actual sensor response is non-linear
- We want to find the nonlinear relation (curved line) to "calibrate" the sensor so it will produce the straight line



The Curve Fitting Problem

Given a set of data points (x_i, y_i) for $i = 1, 2, \ldots, n$ where x_i is the independent variable. Find the "best function" $\hat{f}(x)$ such that

$$\hat{f}(x_i) \approx y_i, \quad i = 1, 2, \ldots, n$$

- The "true function" $f(x)$ that makes $y_i = f(x_i)$ is unknown. We can only obtain an "approximation" $\hat{f}(x)$
- Note that $\hat{f}(x)$ ***does not "interpolate the data"***, *i.e.*, $\hat{f}(x_i) \neq f(x_i)$
- Curve fitting is used when we have ***noisy (inaccurate) data***: $y_i = f(x_i) + n_i$

We are seeking to ***find*** $\hat{f}(x)$ such that *some function of the* approximation ***error*** at every point, e.g., $e_i = \hat{f}(x_i) - f(x_i)$ is made ***as small as possible***

Curve Fitting

- How do we decide ***what class of functions*** to use for the approximation ?
  - ***Use prior information***, if available; e.g., "plot the data" choose suitable functions (e.g., lines, sinusoids, exponentials, polynomials).

- How do we choose the "objective function" that defines the ***approximation error*** to be minimized ?
  - The most common objective function is the ***sum-of-squares*** of the errors at each data point.
  - Mathematically, minimizing the sum-of-squares errors is a least squares problem. This minimization problem that can be easily solved using Calculus.

The Objective Function

- The ***choice of objective function*** (the *cost function*) ***is critical*** to obtain a numerically tractable problem.

- The objective function that we will minimize is the ***sum-of-squares (SOS) of the error*** incurred at each of the given data points.

  The error at point *k* is

$$e_k = \hat{f}(x_k) - y_k, \quad k = 1, 2, \ldots, n$$

- The objective function for least squares is chosen as

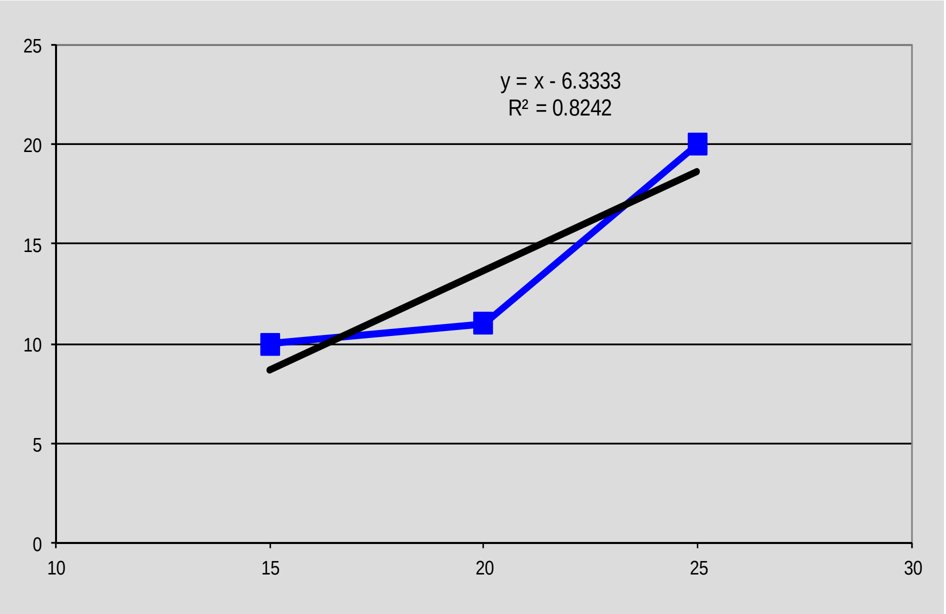
$$J = \sum_{k=1}^{n} e_k^2$$

SOS !

Linear Least Squares

- In linear least-square the "model" that explains the data is a *linear relation*

$$\hat{f}(x) = m\,x + b$$



- ***Problem***: Find ***m*** *(slope)* and ***b*** *(intercept)* such that the straight line ***y=m x+b*** passes ***as close as possible*** to a prescribed set of (data) points ***in the least squares sense***.

Linear Least Squares

- Find the *parameters m and b* such that the *sum-of-squares* of the *error* at the data points is *minimized*
- The ***error at each data point*** is

$$\begin{aligned} e_k &= \hat{f}(x_k) - y_k \\ &= (mx_k + b) - y_k, \quad k = 1, \ldots, n \end{aligned}$$

- The "cost'' or ***objective function*** to be minimized is

$$J(m,b) = \sum_{k=1}^{n} e_k^2 = \sum_{k=1}^{n} (mx_k + b - y_k)^2$$

- Note that $J(m,b)$ describes ***a paraboloid in the (m,b) coordinates***

Linear Least Squares

- Goal: to *minimize* the objective function

$$J(m,b)=\sum_{k=1}^{n}(mx_k+b-y_k)^2$$

with respect to the parameters m and b

- How ? Use Calculus:
  - ***A necessary condition for a minimum or maximum (extrema) of a continuous function is that the derivative of the function be zero***

- In our case $$\frac{\partial J(m,b)}{\partial m}=0, \qquad \frac{\partial J(m,b)}{\partial b}=0$$

Linear Least Squares

- Fact: Since $J(m, b)$ is a paraboloid (convex) in parameters m and b then the above condition is ***not only necessary but also sufficient*** for a extremum (e.g. we don't need to check the 2nd derivatives)

- The minimum (maximum) is found solving the following ***system of equations*** in ***m*** and ***b***

$$\frac{\partial J}{\partial m} = 0$$

$$\frac{\partial J}{\partial b} = 0$$

Linear Least Squares

- Finding the partial derivatives w.r.t. *m*

$$
\begin{aligned}
\frac{\partial J}{\partial m} &= \frac{\partial}{\partial m}\left(\sum_{k=1}^{n}(m x_k + b - y_k)^2\right) \\
&= 2\sum_{k=1}^{n}(m x_k + b - y_k)x_k \\
&= 2\left(m\sum_{k=1}^{n}x_k^2 + b\sum_{k=1}^{n}x_k - \sum_{k=1}^{n}x_k y_k\right) \\
&= 2\left(m\,S_{x^2} + b\,S_x - S_{xy}\right)
\end{aligned}
$$

Note: n – number of points

Linear Least Squares

- Finding the partial derivatives w.r.t. *b*

$$
\begin{aligned}
\frac{\partial J}{\partial b} &= \frac{\partial}{\partial b}\left(\sum_{k=1}^{n}(mx_k + b - y_k)^2\right) \\
&= 2\sum_{k=1}^{n}(mx_k + b - y_k) \\
&= 2\left(m\sum_{k=1}^{n}x_k + b\sum_{k=1}^{n}1 - \sum_{k=1}^{n}y_k\right) \\
&= 2\,(m\,S_x + b\,n - S_y)
\end{aligned}
$$

Note: n – number of points

Linear Least Squares

- Equating them to zero

$$
\begin{aligned}
\frac{\partial J}{\partial m} &= 2\left(m\,S_{x^2} + b\,S_x - S_{xy}\right) = 0 \\
\frac{\partial J}{\partial b} &= 2\left(m\,S_x + b\,n - S_y\right) = 0
\end{aligned}
$$

Rearranging in matrix form

$$
\begin{bmatrix} S_{x^2} & S_x \\ S_x & n \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} S_{xy} \\ S_y \end{bmatrix}
$$

- Now we need to solve for m and b

Linear Least Squares Approximation

- We can solve for *m* and *b* using ***Cramer's rule***

$$m = \frac{\begin{vmatrix} S_{xy} & S_x \\ S_y & n \end{vmatrix}}{\begin{vmatrix} S_{x^2} & S_x \\ S_x & n \end{vmatrix}} = \frac{nS_{xy} - S_x S_y}{nS_{x^2} - S_x S_x}$$

$$b = \frac{\begin{vmatrix} S_{x^2} & S_{xy} \\ S_x & S_y \end{vmatrix}}{\begin{vmatrix} S_{x^2} & S_x \\ S_x & n \end{vmatrix}} = \frac{S_{x^2} S_y - S_x S_{xy}}{nS_{x^2} - S_x S_x}$$

These were the formulas used to find the least square line in hw#1

Given:

$$\begin{bmatrix} S_{x^2} & S_x \\ S_x & n \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} S_{xy} \\ S_y \end{bmatrix}$$

Note: n – number of points

Summary: Linear Least Squares

- The parameters *m*,*b* of the "best line"

$$y = mx + b$$

that approximates the dataset

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

in the *least squares sense* are

$$m = \frac{nS_{xy} - S_x S_y}{nS_x{}^2 - S_x S_y} \qquad b = \frac{S_x{}^2 S_y - S_x S_{xy}}{nS_x{}^2 - S_x S_y}$$

where

$$S_x = \sum_{k=1}^{n} x_k, \quad S_y = \sum_{k=1}^{n} y_k$$

$$S_{xy} = \sum_{k=1}^{n} x_k y_k, \quad S_{x^2} = \sum_{k=1}^{n} x_k^2$$

Approximation Error

- A measure of the accuracy of the approximation is the "sum-of-squares error"

$$E^2 = J = \sum_{k=1}^{n} (mx_k + b - y_k)^2$$

where *m* and *b* are fixed.

- Therefore, the ***approximation error*** is

$$E = \sqrt{\sum_{k=1}^{n} (mx_k + b - y_k)^2}$$

Example: Linear Least Squares

- Data points:

  {(15, 10), (20, 11), (25, 20)}

- Coefficients of linear least squares matrix

$$\mathbf{S_{x^2}} = 15^2+20^2+25^2 = \mathbf{1250}$$

$$S_{x^2} = \sum_{k=1}^{n} x_k^2$$

$$\mathbf{S_{xy}} = 15\text{x}10+20\text{x}11+25\text{x}20 = \mathbf{870}$$

$$S_{xy} = \sum_{k=1}^{n} x_k y_k,$$

$$\mathbf{S_{x}} = 15+20+25 = \mathbf{60}$$

$$S_{x} = \sum_{k=1}^{n} x_k,$$

$$\mathbf{S_{y}} = 10+11+20 = \mathbf{41}$$

$$S_{y} = \sum_{k=1}^{n} y_k$$

Example: Linear Least Squares

- Solving for *a* and *b (using the formulas)*:

$$m = \frac{nS_{xy} - S_x S_y}{nS_x{}^2 - S_x S_y} \qquad b = \frac{S_x{}^2 S_y - S_x S_{xy}}{nS_x{}^2 - S_x S_y}$$

m = *(3x870-60x41)/(3x1250-60x60)*

= *150/150* = ***1.0***

b = *(1250x41-870x60)/(3x1250-60x60)*

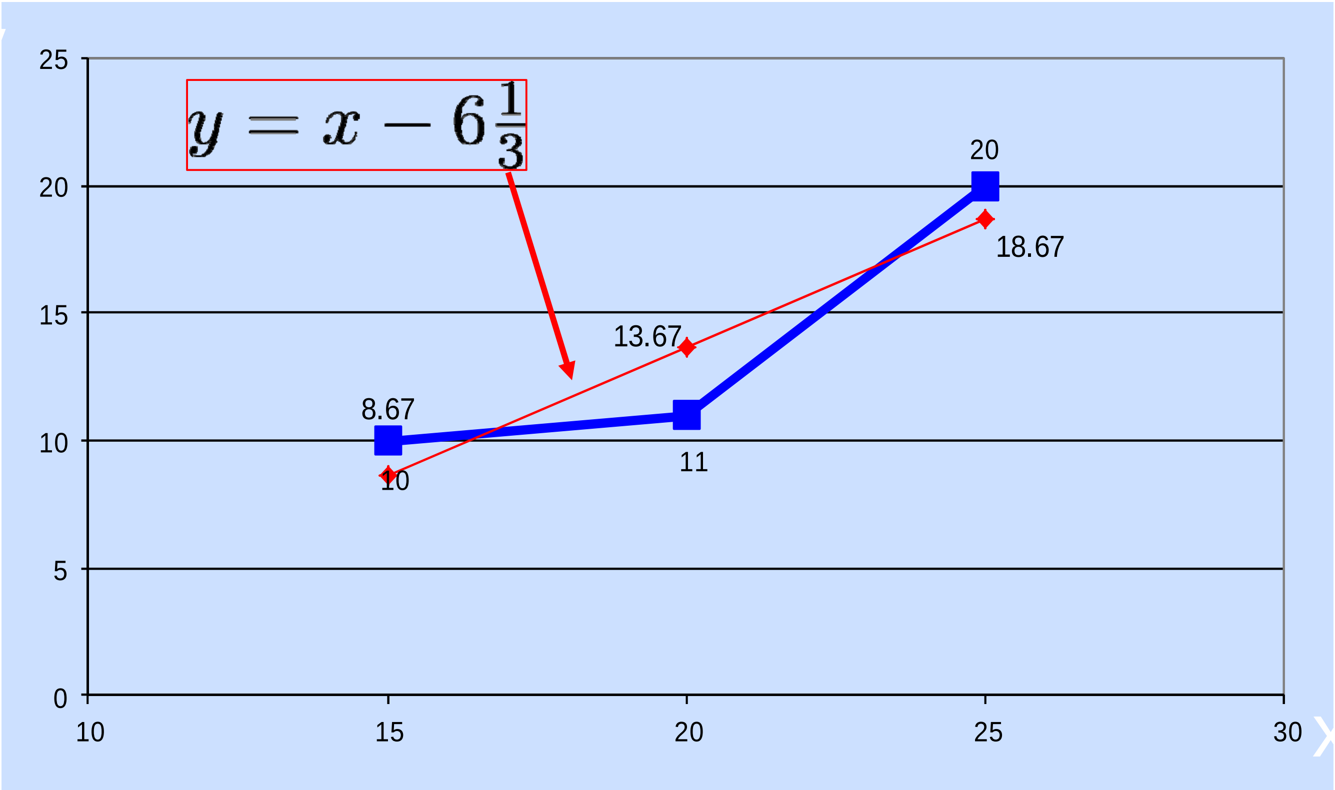
= *-950/150* ***= -6.333***…

- Best line fit (in the least square sense) is

$$y = x - 6\frac{1}{3}$$

Note: n – number of points

Example: Linear Least Squares



Note that the line does not pass (e.g., does not interpolate) through any of the data points

Quadratic Least Squares

- To ***fit a parabola*** follow the same approach except that the "model" is a quadratic function (e.g., a 2nd order polynomial)

$$\hat{f}(x) = a_0 + a_1 x + a_2 x^2$$

- The error function J now depends on the ***parameters*** a_2, a_1, a_0

- To find the minimum value of the approximation error we must **take partial derivatives with respect to each variable** (a_2, a_1, a_0), set them to zero and solve the resulting simultaneous equations using matrix algebra

Quadratic Least Squares

- The 3 simultaneous equations result from

$$\frac{\partial J}{\partial a_2} = 0, \quad \frac{\partial J}{\partial a_1} = 0, \quad \frac{\partial J}{\partial a_0} = 0$$

Note:

- In general, to find the formulas for the constants of a ***polynomial of degree n*** that fits the data (via least-squares) it is necessary to ***solve, symbolically, a system of n+1 simultaneous equations***

Quadratic Least Squares

- For the quadratic case (2nd order polynomial) the system of 3 linear equations is

$$\begin{bmatrix} \sum_{i=1}^{n} x_i^4 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^2 \\ \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} 1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} x_i^2 y_i \\ \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} y_i \end{bmatrix}$$

- In practice a **"better"** way to solve a general least squares fitting problem is by reducing it to a Linear Algebra problem.

Summary

- To solve a fitting problem we must
  a) ***Choose a model*** (***function***) for the data.
  b) ***Choose an error*** (***fitness***) criteria.
- The choice of model and error will determine how "easy" it is to solve the problem (***tractability***)

Key Observation: If the model is ***linear in the parameters*** and the ***error is the SOS*** (sum of squares) then, the fitting problem reduces to a ***system of linear equations***

Applied Programming

Curve Fitting:

Least Squares and Normal Equations

(Linear Algebra Approach)

General Least Squares Formulation

- A general formulation of least squares using linear algebra.
- We need to ***choose a fitting function linear in its parameters***, that is of the form

$$\hat{f}(x) = a_0\phi_0(x) + a_1\phi_1(x) + \cdots + a_N\phi_N(x)$$

where

- ❑ $\phi_i(x)$ are arbitrary ***basis functions***
- ❑ $a_0, \ldots, a_N$ are the ***parameters*** to be determined (by the least square algorithm)

General Least Squares Procedure

1. Collect dataset of input-output ordered pairs

   $$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

2. Choose a suitable approximating (fitting) function (*linear in the parameters*), with ***N << n***

   $$\hat{f}(x) = a_0\phi_0(x) + a_1\phi_1(x) + \cdots + a_N\phi_N(x)$$

3. Form the coefficient matrix ***A*** and vector ***b***

   $$A(k,:) = [\phi_0(x_k) \quad \phi_1(x_k) \cdots \phi_N(x_k)],$$
   $$z(k) = a_k, \quad k = 1,,2,\ldots,n$$
   $$b(k) = f(x_k), \quad k = 1,,2,\ldots,n$$

4. Solve the **over-determined system** for ***z***

   $$Az = b$$

Example 1: Linear Least Squares

Given {(1, 2.8),(3, 4.9),(5, 6.1),(7, 8.9)}

what are the coefficients ***A***, ***b*** and the vector ***x*** that give the ***best linear fit least squares problem*** ?

Solution:

The linear least squares fitting function is

$$\hat{f}(x) = a_0 + a_1 x = a_0\phi_0(x) + a_1\phi_1(x)$$

so the ***basis functions*** must be chosen as follows: $\phi_0(x) = 1, \quad \phi_1(x) = x$

Example 1: Linear Least Squares

Solution… (linear least squares)

Data set: {(1,2.8),(3,4.9),(5,6.1),(7,8.9)}, *n=4*

Basis functions: $\phi_0(x) = 1, \quad \phi_1(x) = x$

Matrix and vector coefficients: (***k=1,…,4***)

$$\begin{aligned} A(k,:) &= [\phi_0(x_k), \phi_1(x_k)] \\ &= [1, \quad x_k] \end{aligned} \qquad b(k) = f(x_k)$$

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 5 \\ 1 & 7 \end{bmatrix} \qquad b = \begin{bmatrix} 2.8 \\ 4.9 \\ 6.1 \\ 8.9 \end{bmatrix}$$

Example 1: Linear Least Squares

Solution… (linear least squares) using GE

Data set: {(1,2.8),(3,4.9),(5,6.1),(7,8.9)}, *n=4*

Basis Functions: $\phi_0(x) = 1, \quad \phi_1(x) = x$

Matrices:

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 5 \\ 1 & 7 \end{bmatrix}, \quad b = \begin{bmatrix} 2.8 \\ 4.9 \\ 6.1 \\ 8.9 \end{bmatrix}, \quad z = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

Equations to solve: $Az = b$

Key Observation: This formulation of the least squares problems leads to a ***system of linear equations*** provided that the fitting function is ***linear in the parameters***.

Linear GE Squares Example 1

Find the best fit for the following data using linear least Squares: (1, 2.8), (3, 4.9), (5, 6.1), (7, 8.9)

```
A = [1 1; 1 3; 1 5; 1 7]
b = [2.8; 4.9; 6.1; 8.9]

x=A\b
```

```
A = 1  1     b = 2.8
     1  3         4.9
     1  5         6.1
     1  7         8.9
x =
  1.77500
  0.97500
```

Giving: $f(x) = 1.775 + 0.975x$

Example 2: Quadratic Least Squares

Given the same data set

{(1, 2.8),(3, 4.9),(5, 6.1),(7, 8.9)}

what are the coefficients ***A***,***b*** and the vector ***x*** that arise if we want find the best ***quadratic fit*** in the least squares sense?

Solution: For ***quadratic*** least squares

$$\begin{aligned} f(x) &= a_0 + a_1 x + a_2 x^2 \\ &= a_0\phi_0(x) + a_1\phi_1(x) + a_2\phi_2(x) \end{aligned}$$

So the ***basis functions*** are

$$\phi_0(x) = 1, \phi_1(x) = x, \phi_2(x) = x^2$$

Example 2:Quadratic Least Squares

Solution… (quadratic least squares) using GE

Data set: {(1,2.8),(3,4.9),(5,6.1),(7,8.9)}, *n=4*

Basis functions: $\phi_0(x) = 1, \phi_1(x) = x, \phi_2(x) = x^2$

Coefficient Matrix ***A***:

$$A(k,:) = [\phi_0(x_k), \phi_1(x_k), , \phi_2(x_k)], k = 1, \ldots, 4$$

$$= [1, \quad x_k, \quad x_k^2], k = 1, \ldots, 4$$

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 9 \\ 1 & 5 & 25 \\ 1 & 7 & 49 \end{bmatrix}$$

Example 2: Quadratic Least Squares

Solution… (quadratic least squares) using GE

Data set: {(1,2.8),(3,4.9),(5,6.1),(7,8.9)}, *n=4*

Basis functions: $\phi_0(x) = 1, \phi_1(x) = x, \phi_2(x) = x^2$

Vectors ***b*** and ***z***:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 9 \\ 1 & 5 & 25 \\ 1 & 7 & 49 \end{bmatrix}, \quad b = \begin{bmatrix} 2.8 \\ 4.9 \\ 6.1 \\ 8.9 \end{bmatrix}, \quad z = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

Equations to solve: $Az = b$

How do we ***solve an over-determined system*** of equations since in general *they do not have a solution* ?

That's where **least squares** comes into play !!

Quadradic Least Square Example 2

Find the best fit for the following data using GE linear least Squares: (1, 2.8), (3, 4.9), (5, 6.1), (7, 8.9)

A = [1 1 1; 1 3 9; 1 5 25; 1 7 49]

b = [2.8; 4.9; 6.1; 8.9]

z=A\b

```
A = 1  1   1    b = 2.8
    1  3   9        4.9
    1  5  25        6.1
    1  7  49        8.9
z = 2.256250
    0.625000
    0.043750
```

Giving: $f(x) = 2.256250 + 0.625000x + 0.043750x^2$

Solving Over determined Systems

- The system of equations ***Az=b, n >> N does not have a solution***, it has more equations than unknowns and is ***usually inconsistent***

- To "solve" it the best we can do it find ***z that minimizes the "size" of the approximation error vector*** (or ***residual***)

$$e = Az - b$$

- The ***least squares*** solution uses the ***2-norm of the error vector*** as a measure of "size" .

- The Least Square problem becomes an optimization problem: $\min_{z} \|Az - b\|_2^2$

The Size of Vectors

- The most intuitive measure of size of vectors in $\mathbb{R}^n$ is their ***"Euclidean norm"***

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

- It turns out that there are many other possibilities, for instance the ***"Manhattan norm"***

$$\|x\|_1 = |x_1| + |x_2| + \cdots + |x_n|$$

- These are just particular cases of the ***p-norm*** of a vector in $\mathbb{R}^n$, defined as

$$\|x\|_p = \begin{cases} \left(\sum_i |x_1|^p\right)^{\frac{1}{p}} & , p \in [1, \infty) \\ \max_i |x_i| & , p = \infty \end{cases}$$

The Euclidean norms is technically called the 2-norm of a vector

Relation to the Sum-Of-Squares

- Going back to the Least Square problem note the ***error vector*** is (recall we have *n* equations)

$$\boldsymbol{e} = \boldsymbol{A}\boldsymbol{z} - \boldsymbol{b}$$

  where $\boldsymbol{e} = (e_1, \ldots, e_n)^T$

- The ***2-norm of the error vector squared*** is

$$\|\boldsymbol{e}\|_2^2 = e_1^2 + e_2^2 + \cdots + e_n^2 = \boldsymbol{e}^T\boldsymbol{e}$$

  which is just the sum of the squares of the errors

- The result of minimizing $\|\boldsymbol{e}\|_2$ or $\|\boldsymbol{e}\|_2^2$ is the same
- However, it turns out that it is easier to work with the sum of the squares

$$\|\boldsymbol{e}\|_2^2 = \boldsymbol{e}^T\boldsymbol{e} = (\boldsymbol{A}\boldsymbol{z} - \boldsymbol{b})^T(\boldsymbol{A}\boldsymbol{z} - \boldsymbol{b})$$

Summary: Least Squares Solution

- An ***overdetermined system*** of equations $\boldsymbol{e} = A\boldsymbol{z} - \boldsymbol{b}$

  does not have any regular solution
- We will instead solve a ***optimization problem*** to find a vector ***z*** that ***minimizes some norm of the error vector*** (or residual)

$$\min_{\boldsymbol{z}} \|A\boldsymbol{z} - \boldsymbol{b}\|$$

- For tractability we will choose the ***2-norm*** leading to a ***least squares problem***
  - Instead of minimizing the 2-norm we will ***minimize the 2-norm squared*** (both have the same solution)

$$\min_{\boldsymbol{z}} \|\boldsymbol{A}\boldsymbol{z} - \boldsymbol{b}\|_2^2 = \min_{\boldsymbol{z}} (\boldsymbol{A}\boldsymbol{z} - \boldsymbol{b})^T (\boldsymbol{A}\boldsymbol{z} - \boldsymbol{b})$$

Derivation: Least Squares Solution

- Cost Function (to be minimized)

$$J(\boldsymbol{z}) = (\boldsymbol{A}\boldsymbol{z} - \boldsymbol{b})^T (\boldsymbol{A}\boldsymbol{z} - \boldsymbol{b})$$

- (the vector ***z*** has the parameters of the least square fitting function we want)

- Expand the cost function

$$\frac{\partial J(z)}{\partial z} = \frac{\partial z^T A^T A z}{\partial z} - \frac{\partial b^T A z}{\partial z} - \frac{\partial z^T A^T b}{\partial z}$$

- Take partial derivatives w.r.t. the vector ***z***

$$\begin{aligned} J(z) &= ||Az - b||_2^2 = (Az - b)^T (Az - b) \\ &= z^T A^T A z - z^T A^T b - b^T A z + b^T b \end{aligned}$$

How ? (Using **“Matrix Calculus”**, see reference slide)

Reference: Matrix Calculus

- Differentiation Formulas for Matrices and Vectors)

Let $A \in \mathbb{R}^{n \times n}$, $\boldsymbol{c} \in \mathbb{R}^n$ and $\boldsymbol{x} \in \mathbb{R}^n$

1. $\dfrac{\partial}{\partial \boldsymbol{x}} \left(\boldsymbol{x}^T \boldsymbol{c} \right) = \boldsymbol{c}$
2. $\dfrac{\partial}{\partial \boldsymbol{x}} \left(\boldsymbol{c}^T \boldsymbol{x} \right) = \boldsymbol{c}$
3. $\dfrac{\partial}{\partial \boldsymbol{x}} \left(\boldsymbol{x}^T \boldsymbol{x} \right) = 2\boldsymbol{x}$
4. $\dfrac{\partial}{\partial \boldsymbol{x}} (A\boldsymbol{x}) = A^T$
5. $\dfrac{\partial}{\partial \boldsymbol{x}} \left(\boldsymbol{x}^T A \boldsymbol{x} \right) = A\boldsymbol{x} + A^T \boldsymbol{x} = (A + A^T)\boldsymbol{x}$
6. $\dfrac{\partial}{\partial \boldsymbol{x}} \left(\boldsymbol{x}^T A \boldsymbol{x} \right) = 2A\boldsymbol{x}$ if A is symmetric

Derivation: Least Squares Solution

- Set partials to zero (necessary cond.)

$$\frac{\partial J(z)}{\partial z} = \frac{\partial z^T A^T A z}{\partial z} - \frac{\partial b^T A z}{\partial z} - \frac{\partial z^T A^T b}{\partial z}$$
$$= 2A^T A z - 2A^T b = 0$$

Main Result: The least squares problem reduces to finding the regular solution to the ***NxN* "normal equations"**

$$\left(A^T A\right) \boldsymbol{z}_{ls} = A^T \boldsymbol{b}$$

- Warning: $A\boldsymbol{z}_{ls} \neq \boldsymbol{b}$

Summary: General LS Algorithm

Input: dataset of input-output ordered pairs

$$\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

1. Choose ***approximating function*** *(N<<n)*

$$\hat{f}(x) = a_0\phi_0(x) + a_1\phi_1(x) + \cdots + a_N\phi_N(x)$$

2. Form the ***coefficient matrices A*** *and* ***b***

$$A(k,:) = [\phi_0(x_k) \cdots \phi_N(x_k)], k = 1, \dots, n$$

$$z(k) = a_k, \quad k = 1, \dots, n$$

$$b(k) = f(x_k), \quad k = 1, \dots, n$$

3. Solve the ***normal equations*** for ***z*** (vector of parameters)

$$(A^T A)\boldsymbol{z} = A^T\boldsymbol{b}$$

Example 1: Linear Least Squares

Solution… (continued)

Data set: {(1,2.8),(3,4.9),(5,6.1),(7,8.9)}, $n=4$

Basis Functions: $\phi_0(x) = 1, \quad \phi_1(x) = x$

Matrices:

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 5 \\ 1 & 7 \end{bmatrix}, \quad b = \begin{bmatrix} 2.8 \\ 4.9 \\ 6.1 \\ 8.9 \end{bmatrix}, \quad z = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

Normal Equations: $(A^T A)\boldsymbol{z} = A^T \boldsymbol{b}$

$$A^T A = \begin{bmatrix} 4 & 16 \\ 16 & 84 \end{bmatrix}, \quad A^T b = \begin{bmatrix} 22.7 \\ 110.3 \end{bmatrix}$$

Example 1: Linear Least Squares

- The over-determined system of equations is transformed to *Normal Equations* to find the least squares solution $(A^T A)\boldsymbol{z} = A^T \boldsymbol{b}$

$$\begin{bmatrix} 4 & 16 \\ 16 & 84 \end{bmatrix} \boldsymbol{z} = \begin{bmatrix} 22.7 \\ 110.3 \end{bmatrix}$$

- The least squares solution is:

$$z = \begin{bmatrix} a_0 & a_1 \end{bmatrix}^T = \begin{bmatrix} 1.775 & 0.975 \end{bmatrix}^T$$

Finally, the equation of the best *least squares linear fit* (for this example) is

$$\hat{f}(x) = a_0 x^0 + a_1 x^1 = 1.775 + 0.975x$$

Warning: For large (and ill-conditioned) problems we must **avoid forming the product $A^T A$** (it exacerbates numerical errors)

Normal Linear Example 1

Find the best fit for the following data using linear least Squares: (1, 2.8), (3, 4.9), (5, 6.1), (7, 8.9)

```
A = [1 1; 1 3; 1 5; 1 7]
b = [2.8; 4.9; 6.1; 8.9]

AtA=A'*A

Atb=A'*b
z=AtA\Atb
```

```
A = 1  1     b = 2.8
    1  3         4.9
    1  5         6.1
    1  7         8.9
AtA =   4   16
       16   84
Atb =  22.700
      110.300

z =  1.77500
     0.97500
```

Giving: $f(x) = 1.775 + 0.975x$

Normal Quadradic Example 2

Find the best fit for the following data using linear least Squares: (1, 2.8), (3, 4.9), (5, 6.1), (7, 8.9)

A = [1 1 1; 1 3 9; 1 5 25; 1 7 49]

b = [2.8; 4.9; 6.1; 8.9]

AtA=A'*A

Atb=A'*b

z=AtA\Atb

```
A = 1  1   1    b = 2.8
    1  3   9        4.9
    1  5  25        6.1
    1  7  49        8.9
z = 2.256250
    0.625000
    0.043750
```

Giving: $f(x) = \mathbf{2.256250} + \mathbf{0.625000}\,x + \mathbf{0.043750}x^2$

Least-Squares

- The simplest functions that we can use for least squares approximation are polynomials of degree N
- The complexity of the problem increases with the degree of the polynomial: we need to solve a system of N+1 simultaneous equations (the ***normal equations***) to determine the coefficients of polynomial of degree N

- What happens if the data points are complex numbers (*e.g.*, $a+jb$) ?

  The formulation is the same, we just need to perform complex arithmetic.

Applied Programming

Interactive Data Fitting with Matlab

Fit Quality

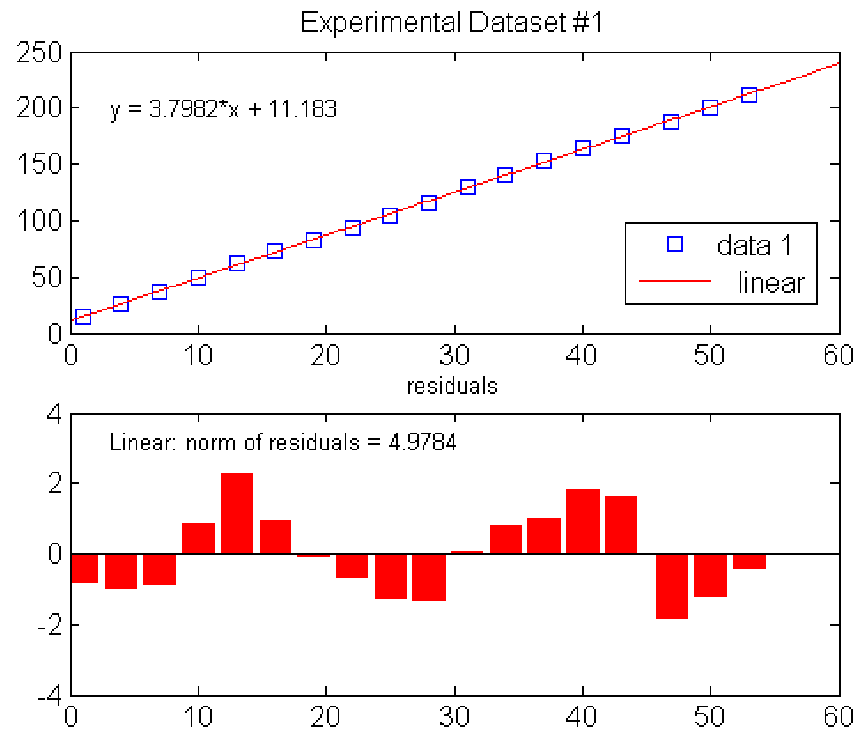
- How do you determine if the fit is “good”
  – Plot the results
  – Plot the error
  – Calculate the Norm of Residuals
    - Smaller better
  – Calculate the R**2 error
    - Closer to 1 better

- Better numbers – not always better
  – Often we are just fitting the noise

Plot the Error

- Calculate the point by point difference between the data and the value of the function at the point

- Look at the shape and values of the result



Norm of Residuals

- Square root of the sum of the errors squared
  - Smaller better
  - Grows with the number of data points

- Norm = $\sqrt{\sum_{i=1}^{n}\left(y_i - f(x_i)\right)^2}$

Base one

Pearson's Correlation

- The ***Pearson's coefficient*** **(r, R)** is a ***measure of the strength and direction of a linear relationship between two variables***
  - closer to 1 is better
  - -1 is fully uncorrelated

Excel correl() function

- Computed as follows: R =

$$\frac{n\left(\sum_{i=1}^{n}\left(y_i * f(x_i)\right)\right) - \left(\sum_{i=1}^{n} y_i\right) * \left(\sum_{i=1}^{n} f(x_i)\right)}{\sqrt{\left[n\sum_{i=1}^{n}(y_i)^2 - \left(\sum_{i=1}^{n} y_i\right)^2\right] * \left[n\sum_{i=1}^{n} f(x_i)^2 - \left(\sum_{i=1}^{n} f(x_i)\right)^2\right]}}$$

R^2 coefficient of Determination

- The ***R^2 coefficient*** indicates ***how close the function is fitting the data***
  - R^2 close to 1 is better
  - Not the same as Pearson's correlation

- This ***statistical coefficient*** is computed as

follows: $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - f(x_i))^2}{\sum_{i=1}^{n}(y_i - u)^2}$

Where u= $\frac{1}{n}\sum_{i=1}^{n}(y_i)$

Ex#1: Matlab Interactive Fitting

X	Y
1	14.2
4	25.4
7	36.9
10	50
13	62.8
16	72.9
19	83.3
22	94.1
25	104.9
28	116.2
31	129
34	141.1
37	152.7
40	164.9
43	176.1
47	187.9
50	199.9
53	212.1

Data points are store in a plain text file called **`data.txt`**

Load and plot the data points in Matlab (Octave):

```
>>% Load Data Points
>>load -ascii data.txt
>>x=data(:,1);
>>y=data(:,2);
>>% Plot point
>>plot(x,y);
>>title('Experimental
Dataset #1')
```

In the figure window menu select **`Tools`** and then **`Basic Fitting`** (Only available in Matlab)

Matlab Settings

- Enable Matlab in your current session:

  module load matlab

- Run Matlab: matlab
  - Note: Matlab will start in GUI mode with a splash screen
  - Requires Xwindows

Module info: wiki.rit.edu/display/kgcoeuserdocs/Modules

Optional Matlab Settings

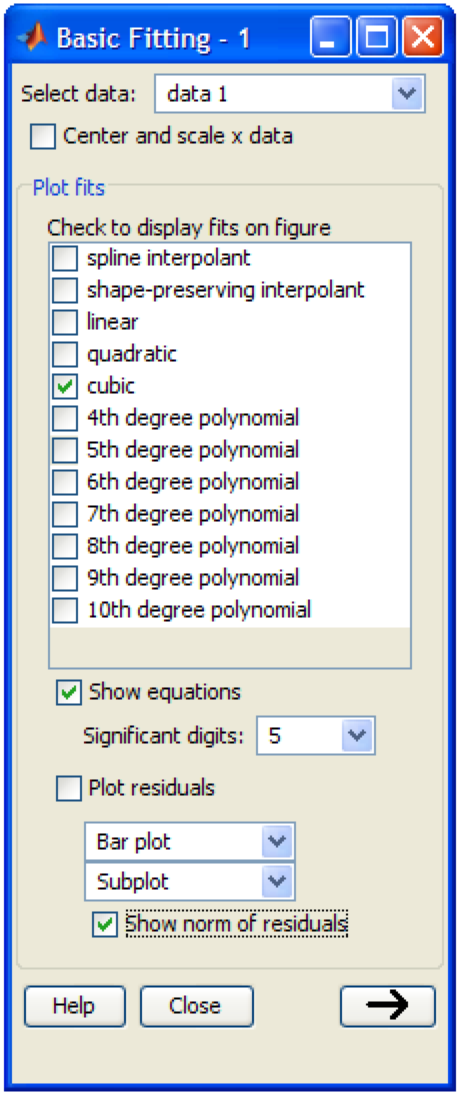
- Matlab can be run in character terminal mode.
  - No graphical displays or plots (no X needed)
  - matlab -nosplash -nodisplay –noawt
  - Faster graphical startup
  - matlab -nosplash

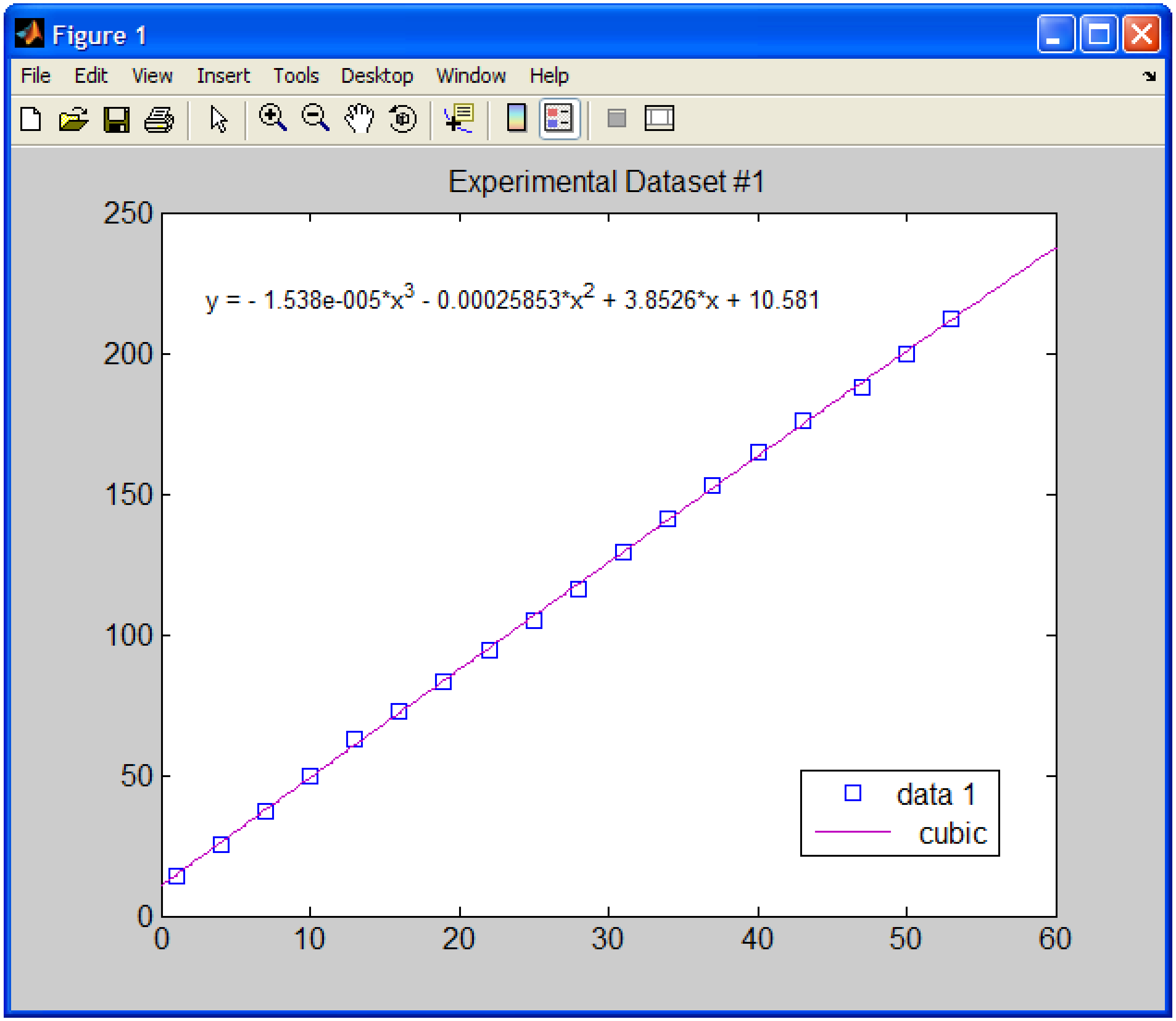
- Add an alias to your .bashrc:

```
alias tmatlab=‘matlab -nosplash -nodisplay –noawt’
alias xmatlab=‘matlab –nosplash’
```

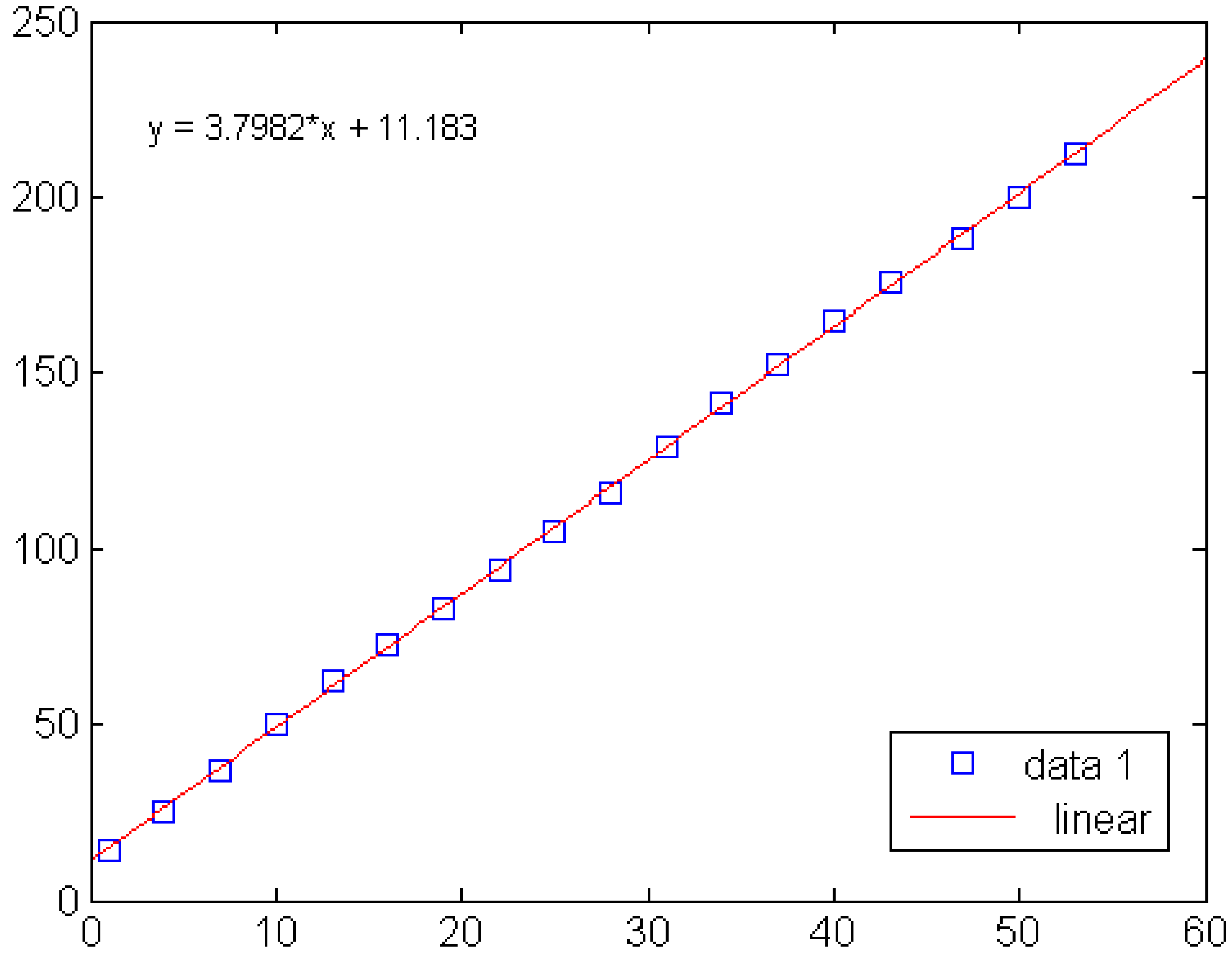
  - Rerun your .bashrc file (source ./.bashrc)

Ex#1: Matlab Interactive Fitting

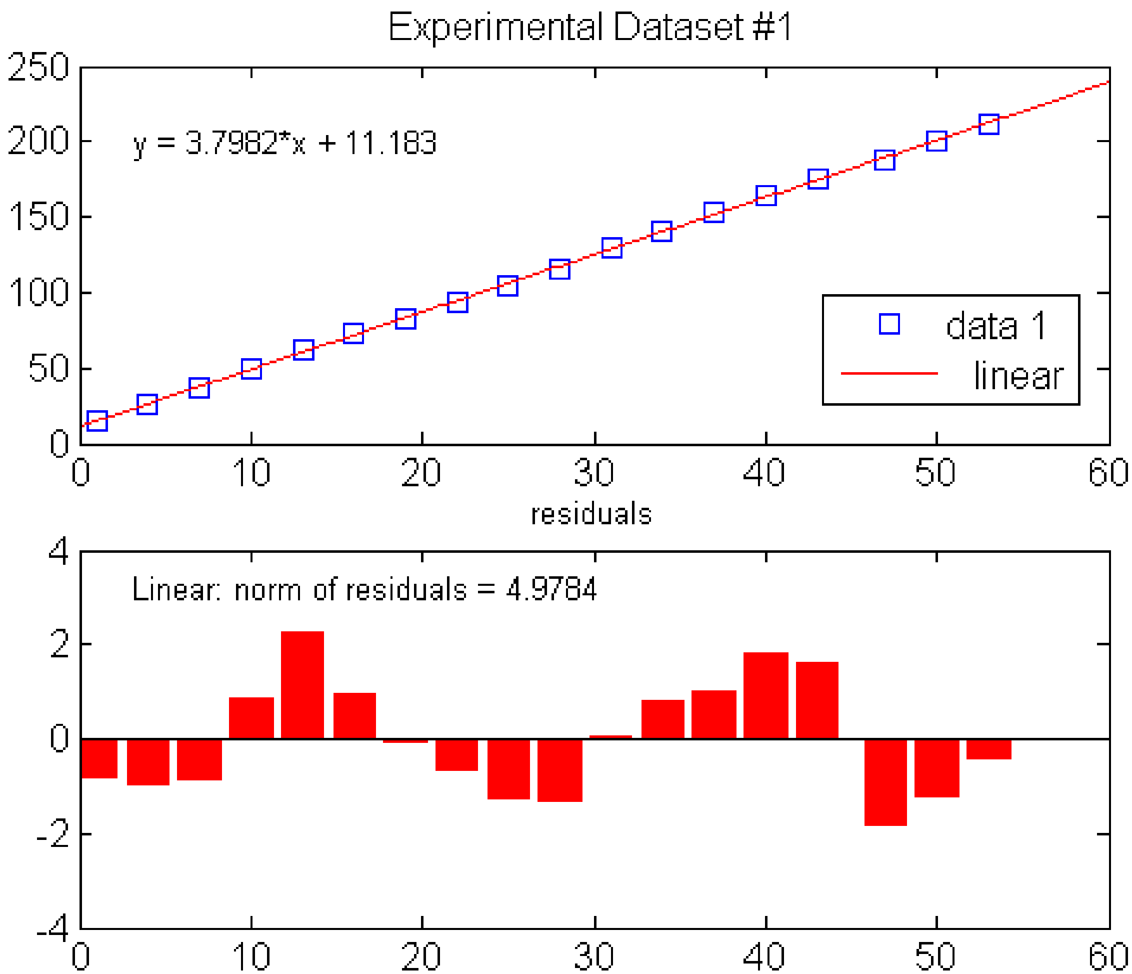




Ex#1: Matlab Interactive Fitting



Ex#1: Matlab Interactive Fitting



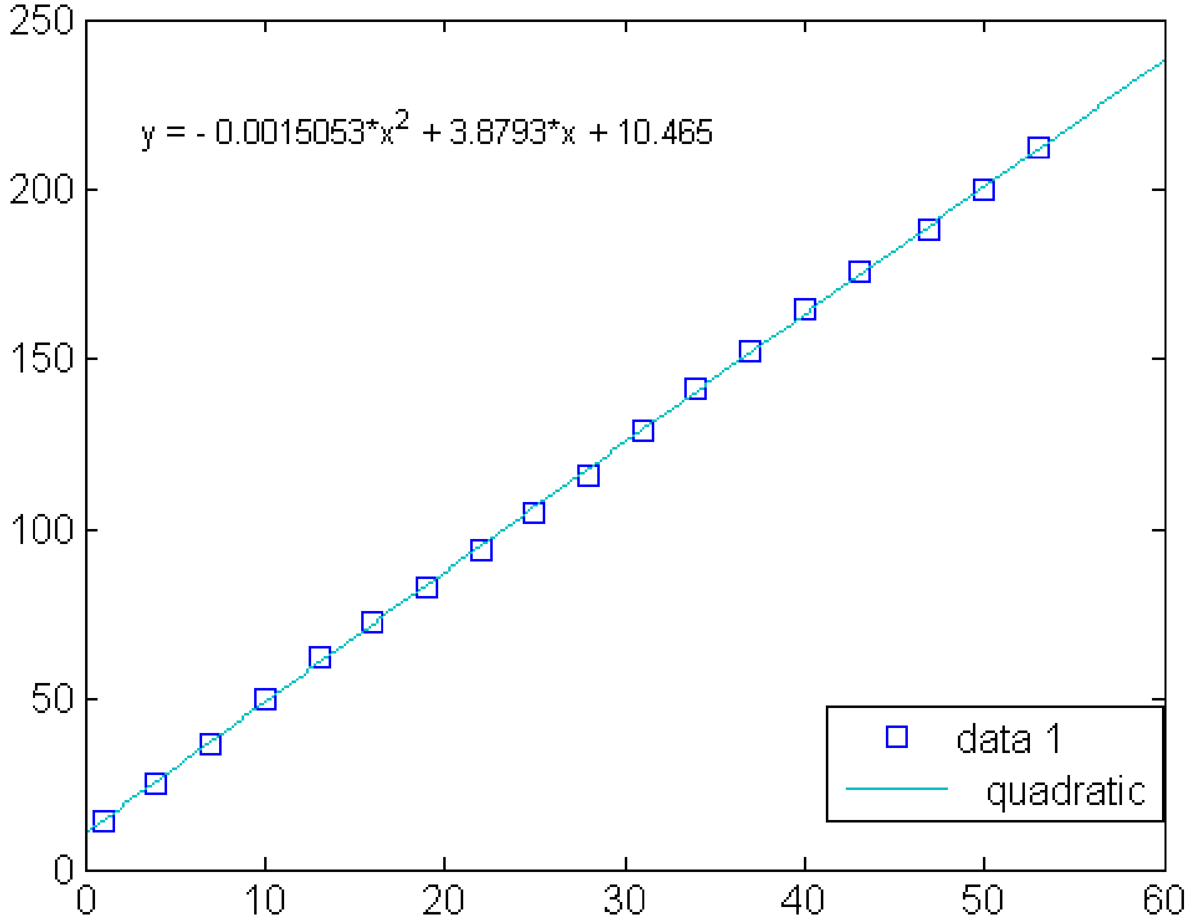
Example #1

- Q: Is the fitting good enough? Is the chosen function appropriate ?

- A: It “looks” good enough,

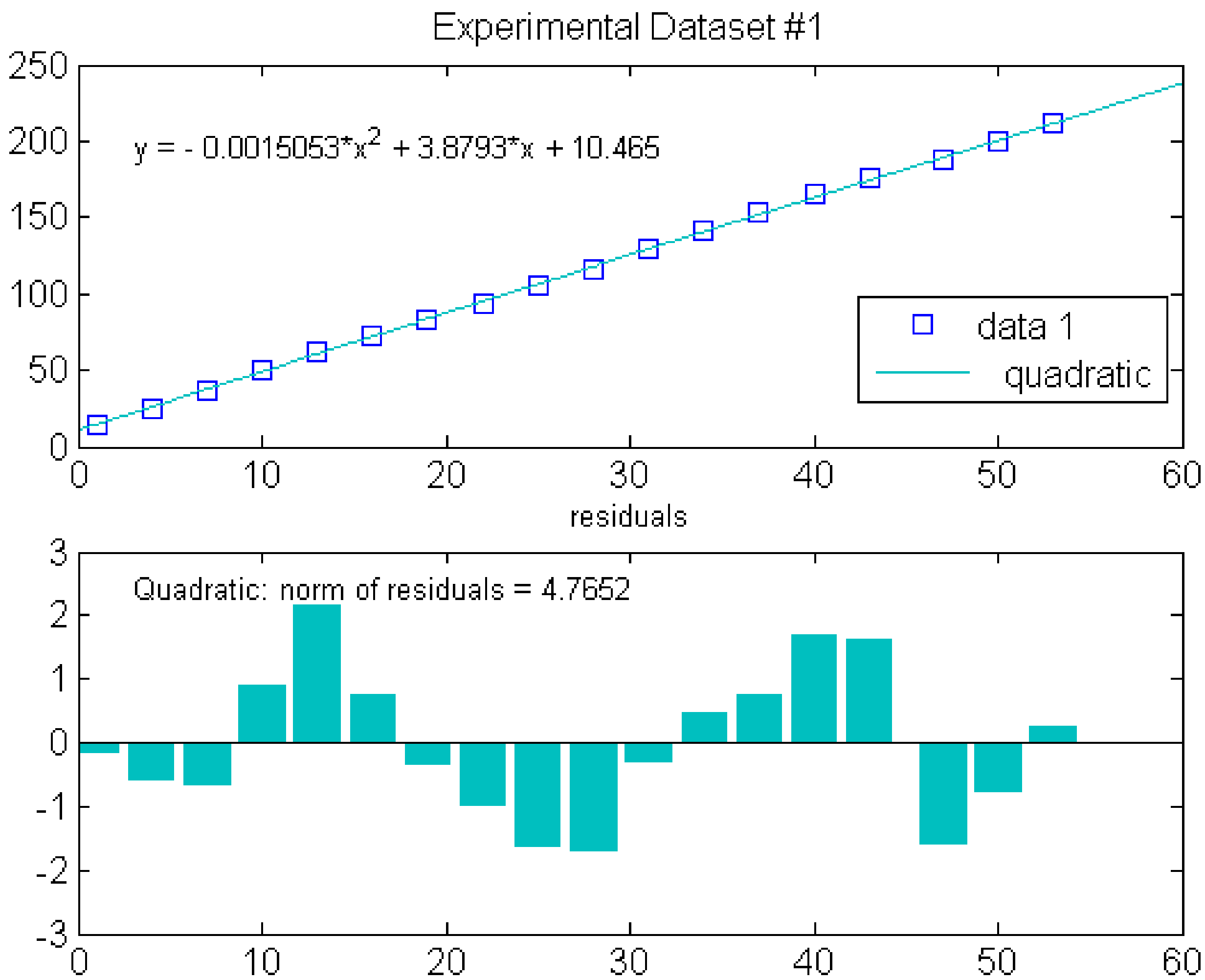
- If working interactively we could increase the complexity of the model (quadratic, cubic, etc.) and see if the ***total error*** is reduced

Example #1: Quadratic Fit



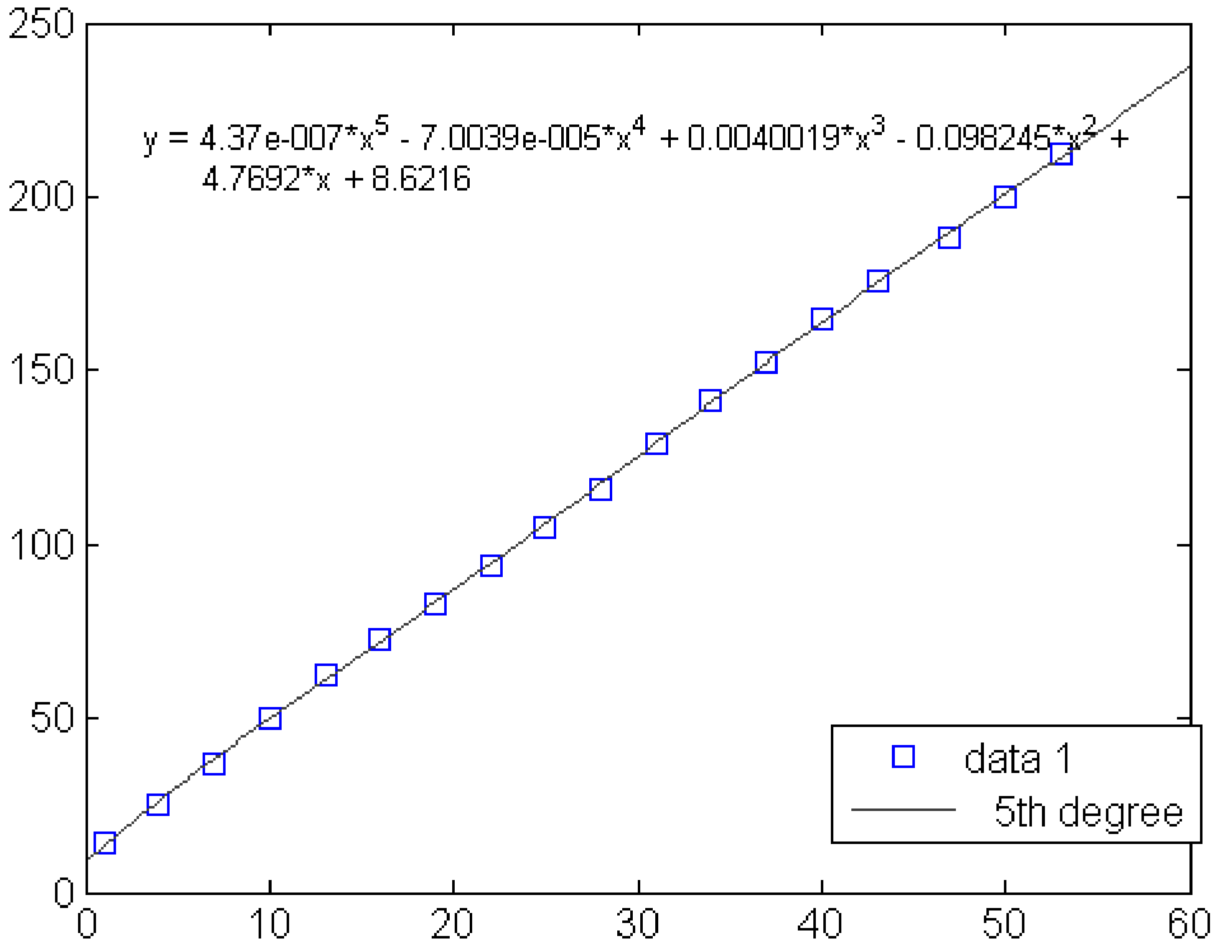
Q: Is it better than a linear fit ?

Example #1: Quadratic Fit

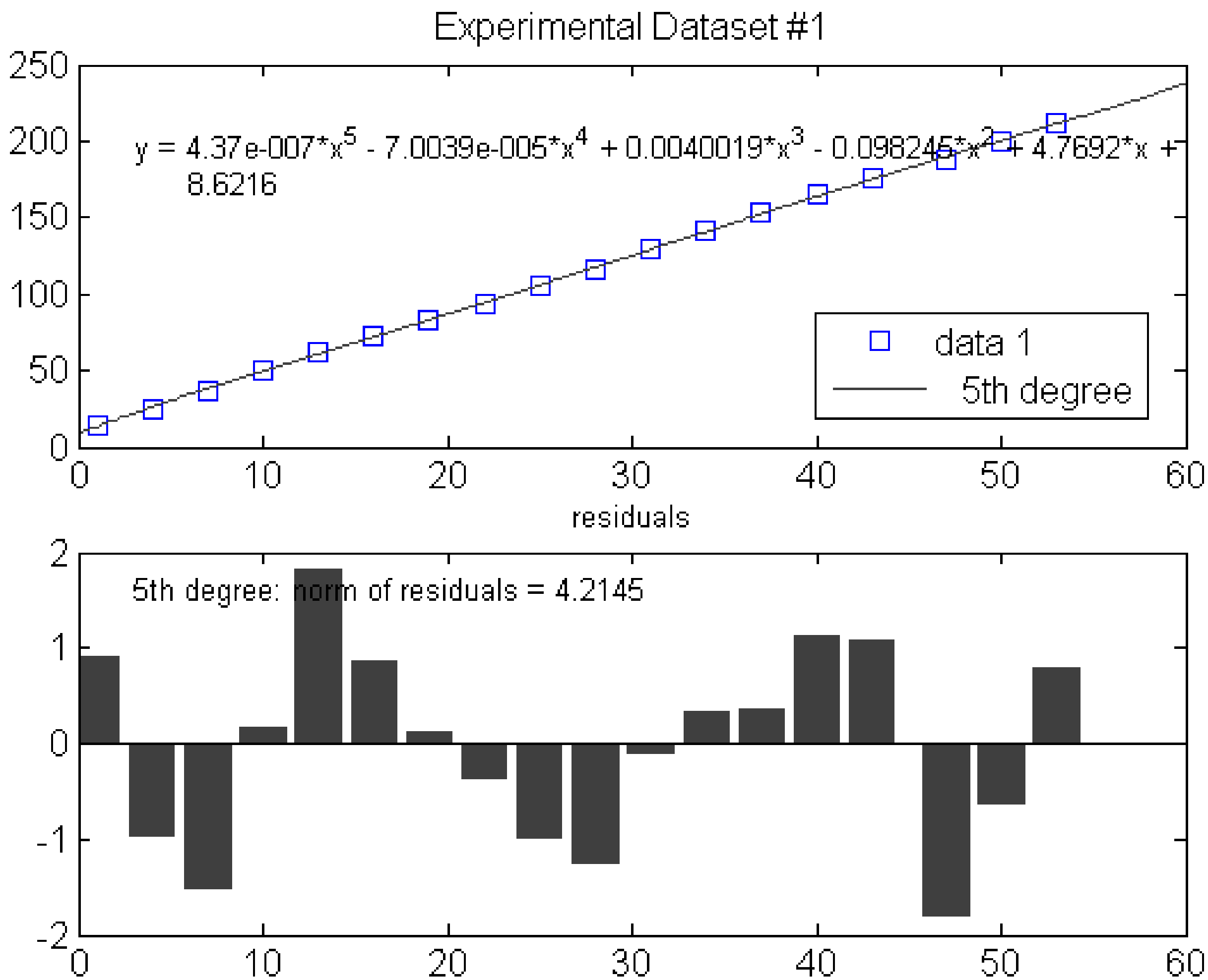


Compare residuals: 4.98 (linear) , 4.77 (quadratic)

Example #1: 5th order fit



Example #1: 5th order fit



Residuals: 4.98 (linear) , 4.77 (quadratic), 4.21 (5th order)

Summary: LS Fitting

- Data fitting is the process of finding a function that models a noisy data set.
- Matlab offers an interactive GUI for least squares data fitting using polynomials as models (MS Excel does too, Octave does not)
- In polynomial fitting, the quality of the fit can be measured by the norm of the residual vector (square root of sum-of-squares of the errors)

Warning: Choosing a high degree polynomial will general reduce the error but you will be actually ***fitting not only the data but also the noise.***

Applied Programming

Solving

Least Squares Problems

Efficiently

Solving LS Problems Efficiently

- Least Squares fitting problems can be reduced to a system of ***overdetermined*** equations

  $$Az = b$$

  which can be reduced to a system of ***Normal Equations:*** $(A^T A)z = A^T b$
  - "squaring A" may introduce unacceptable errors

- We need an algorithms than can solve the overdetermined systems without forming the Normal Equations.
  - One algorithm is: ***QR factorization***

QR factorization

- The QR factorization algorithm applies a ***sequence of orthogonal transformations*** (as opposed to elementary transformations as in Gaussian Elimination) to the matrix ***A***
- These orthogonal transformations induce the ***QR factorization*** of $A \in \mathbb{R}^{m \times n}$

$$A = QR$$

$$Q \in \mathbb{R}^{m \times n}, \text{ an isometry, } \boxed{Q^T Q = I}$$

$$\boxed{R \in \mathbb{R}^{n \times n}, \text{ upper triangular}}$$

- Using the ***QR factorization***

$$\boldsymbol{A z} = \boldsymbol{b} \Rightarrow Q R \boldsymbol{z} = \boldsymbol{b} \Rightarrow Q^T Q R \boldsymbol{z} = Q^T \boldsymbol{b} \Rightarrow R \boldsymbol{z} = Q^T \boldsymbol{b}$$

We will use standard ***QR factorization*** libraries

Least Squares via QR Factorization

- The ***most reliable*** way to solve the normal equations of a least squares problem is:

1. Find A and $\boldsymbol{b}$ for the least squares problem.
2. Compute the QR factorization of A, $A = QR$
3. Find $c = Q^T b$
4. Solve $R\boldsymbol{z} = \boldsymbol{c}$ for $\boldsymbol{z}$ by back substitution

Notes:

- The matrix **Q is (in general) not square**
- The matrix ***R* is upper triangular**
- The QR algorithm is used in Matlab (and Octave) when you use the backslash operator: $A \backslash \boldsymbol{b}$

QR in Octave/Matlab

Given any matrix $A \in \mathbb{R}^{m \times n}$

- Compute the ***compact QR*** factorization of ***A*** using

  `[Q,R]=qr(A,0);`

  Where both ***Q and R are full rank*** and **`A = Q*R`**

Warning:

- There is also a "full QR" factorization

  `[Q,R]=qr(A);`

  – *Do not use* the *full QR for least squares problems.*

GSL

- The ***GNU Scientific Library (GSL)*** provides many algorithms to a broad range of problems (commonly available in Matlab or Python/Sciply) solve linear algebra problems, including QR factorizations.

- GSL includes algorithms for:
  - **Linear Algebra** (LU, **QR**, SVD, etc.)
  - Interpolation
  - Numerical Differentiation and Integration
  - ODE’s
  - FFT
  - Etc.

General GSL

- Compilation:

gcc -ansi -g -lgsl -lgslcblas file.c –o runTime

- Additional includes:

```
#include <gsl/gsl_math.h>
#include <gsl/gsl_blas.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_linalg.h>
```

- Documentation:

www.gnu.org/software/gsl/manual/html_node/Introduction.html#Introduction

GSL Error Handling

- By default, the GSL functions check all GSL memory allocations and WILL EXIT if there is an error
  - Normally this works fine

- There are methods to override this behavior
  - **gsl_set_error_handler** *(gsl_error_handler_t * new_handle)*
  - Not used in this class

General GSL

- Data types:

```
gsl_vector *v;
gsl_matrix *A;
```

- Allocating and freeing space

```
v  = gsl_vector_alloc(nc);
A = gsl_matrix_alloc(nr, nc);
gsl_vector_free(v);
gsl_matrix_free(A);
```

gsl_vector

- Normally hidden from the programmer
  - GSL has its own form of memory management

```
typedef struct {
size_t size;              // Number of elements
size_t stride;            // Step-size from one
                          // element to the next
                          // in physical memory
double * data;            // Points to the data
gsl_block * block;        // Higher level memory
                          // abstraction
int owner;                // block ownership status
}  gsl_vector;
```

gsl_matrix

- Normally hidden from the programmer
  - Matrices are stored in “C style” row-major order.
- typedef struct {

```
size_t size1;          // Row    *Might need these
size_t size2;          // Column
size_t tda;            // Size of a row in memory
double * data;         // points to the data
gsl_block * block;     // Internal use
int owner;
} gsl_matrix;
```

Allocating/freeing vectors

- GSL does not use malloc/free to manage memory but has specific functions
  - Don't use malloc() or free() for GSL objects

- Allocating

  (gsl_vector *) gsl_vector_alloc(size_t nc)
  - nc – the number of "columns" (entries)
  - E.g. c = gsl_vector_alloc(100);

- Freeing

  void gsl_vector_free(gsl_vector *c)
  - c – pointer to a previously allocated vector
  - E.g. gsl_vector_free(c);

Allocating/freeing matrices

- Allocating

  (gsl_matrix *) gsl_matrix_alloc(size_t nr, size_t nc)

  – nr, nc – the number of “rows and columns”

  – E.g. A = gsl_matrix_alloc(10, 20);

- Freeing

  void gsl_matrix_free(gsl_matrix *A)

  – A – pointer to a previously allocated matrix

  – E.g. gsl_matrix_free(A);

- GSL also offers gsl_vector_calloc() and gsl_matrix_calloc()

General GSL

- Read and writing vectors and matrices

```
gsl_matrix_set(A, i, j, num);
gsl_vector_set(b, i, num);

val = gsl_matrix_get(A, i, j);
val = gsl_vector_get(b, i);
```

Write a vector

void gsl_vector_set(gsl_vector * v, const size_t nc, double x)

– v – pointer to a GSL vector

– nc – the index item to set

– x – the value to put into the vector

- e.g.: Put the value 3.5 into b[5]

```
gsl_vector_set(b, 5, 3.5);
```

Read a vector

double gsl_vector_get (const gsl_vector * v, const size_t nc)

– v – pointer to a GSL vector

– nc – the index item to get

- e.g.: Get the value from b[5]

```
num = gsl_vector_get(b, 5);
```

Write a matrix

void gsl_matrix_set(gsl_matrix * A, const size_t nr, size_t nc, double x)

- v – pointer to a GSL vector
- nr, nc – the row and column item to set
- x – the value to put into the matrix

- e.g.: Put the value 3.5 into A[5][7]

  gsl_vector_set(A, 5, 7, 3.5);

Read a matrix

double gsl_martix_get (const gsl_matrix * A, const size_t nr, size_t nc)

- A – pointer to a GSL matrix
- cr, nc – the row and column item to get

- e.g.: Get the value from A[5][7]

  ```
  num = gsl_matrix_get(A, 5, 7);
  ```

GSL Matrix Vector dot product

int gsl_blas_dgemv(CBLAS_TRANSPOSE_t TransA, double alpha,
const gsl_matrix * A, const gsl_vector * x, double beta, gsl_vector * y)

- CBLAS_TRANSPOSE_t TransA – always CblasNoTrans in this class
- double alpha – always 1.0 in this class
- const gsl_matrix *A - input matrix
- const gsl_vector *x - input vector
- double beta – always 0.0 in this class
- gsl_vector * y - The resulting vector

- E.g.

  gsl_blas_dgemv (CblasNoTrans, 1.0, AT, b, 0.0, ATB);

  - AT, b – Input matrix and vector
  - ATB - Resulting vector

GSL Matrix Matrix dot product

int gsl_blas_dgemm (CBLAS_TRANSPOSE_t TransA, CBLAS_TRANSPOSE_t TransB, double alpha, const gsl_matrix *A, const gsl_matrix * B, double beta, gsl_matrix * C)

- CBLAS_TRANSPOSE_t TransA – always CblasNoTrans in this class
- CBLAS_TRANSPOSE_t TransB – always CblasNoTrans in this class
- double alpha – always 1.0 in this class
- const gsl_matrix *A - input matrix
- const gsl_matrix *B - input matrix
- double beta – always 0.0 in this class
- gsl_matrix *c - The resulting matrix

- E.g.

  gsl_blas_dgemm (CblasNoTrans, CblasNoTrans, 1.0, AT, A, 0.0, ATA);

  - AT, A – Input matrices
  - ATA - Resulting matrix

Matrix transpose

int gsl_matrix_transpose_memcpy (gsl_matrix * dest, const gsl_matrix * src)

- gsl_matrix *dest - the destination transposed matrix
- gsl_matrix *str - source matrix to transpose

- E.g.

  gsl_matrix_transpose_memcpy (AT, A);

  - A – Original matrix
  - AT – resulting transposed matrix

QR decomposition

int **gsl_linalg_QR_decomp** *(gsl_matrix * A, gsl_vector * tau)*

– Gsl_matrix *A –The input matrix A AND the output matrix R.

On output the diagonal and upper triangular part contain. (original A is destroyed). The of the lower triangular part of the matrix contain the Householder coefficients

– Gsl_vector *tau - output householder vectors.

E.g.

gsl_linalg_QR_decomp(A, tau);

– On return, the vector tau and the columns of the lower triangular part of the matrix A have the Householder coefficients and vectors
– Note: overwrites A!

Least Squares Solver

int gsl_linalg_QR_lssolve(const gsl_matrix *QR, const gsl_vector *tau,
const gsl_vector *b, gsl_vector *x, gsl_vector *residual)

- const gsl_matrix *QR - from gsl_linalg_QR_decomp
- const gsl_vector *tau - same
- const gsl_vector *b - “b” vector
- gsl_vector *x - solution returned
- gsl_vector *residual -

- E.g.

  gsl_linalg_QR_lssolve(A, tau, b, x_sol, res);

  - x_sol – contains the solution vector
  - Res - the 2 norm error residual vector

QR Code fragment

- Solve Ax=b directly via QR factorization

```
gsl_linalg_QR_decomp(A, tau);
gsl_linalg_QR_lssolve(A, tau, b, x_ls, res);
printf("Least Squares Solution via QR
factorization:\n");
for(i = 0; i < nc; i++)   {
    printf(" x_ls[%1d] = % 20.16e \n",i,
                                    gsl_vector_get(x_ls, i));   }
printf("\n");
printf(" LS error = %f:\n",gsl_blas_dnrm2(res));
```

GSL_LS_with_QR.c

- 10 rows, 3 columns, filled with simple dummy data

```
A (10 x 3)                                        b (10 x 1)
 0:         1          0          0 |      1
 1:       0.5    0.33333       0.25 |      2
 2:   0.33333       0.25        0.2 |      3
 3:      0.25        0.2    0.16667 |      4
 4:       0.2    0.16667    0.14286 |      5
 5:   0.16667    0.14286      0.125 |      6
 6:   0.14286      0.125    0.11111 |      7
 7:     0.125    0.11111        0.1 |      8
 8:   0.11111        0.1   0.090909 |      9
 9:       0.1   0.090909   0.083333 |     10

Least Squares Solution via QR
factorization:
 x_ls[0] =  1.064293446794555le+00
 x_ls[1] = -3.069218329799226le+02
 x_ls[2] =  4.063461418940621 6e+02
 LS error = 6.713737:
```

Summary

- A general *least squares approximation* problem can be *reduced to* the solution of the ***normal equations***

$$A^T A\boldsymbol{z} = A^T\boldsymbol{b}$$

when the fitting function is *linear in the parameters*

$$f(x) = a_0\phi_0(x) + a_1\phi_1(x) + \cdots + a_N\phi_N(x)$$

- The matrices ***A*** and ***b*** are formed using the ***data points and basis functions.***

$$A(k,:) = [\phi_0(x_k)\cdots\phi_N(x_k)], k = 1,\ldots,n$$

$$b(k) = y_k, \quad k = 1,\ldots,n$$

- The ***best way to solve least squares*** problems is using ***QR factorization***

Solving Least Squares with GSL

- To solve a least squares problem: $Az = b$
- Use:

gsl_linalg_QR_decomp(A, tau);

  - After calling, The diagonal and upper triangular part of A contain *R*. The vector *tau* and the columns of the lower triangular part of A contain the Householder coefficients and vectors which encode the *Q*.

gsl_linalg_QR_lssolve(A, tau, b, z, res);

- Documentation about these function can be found in http://www.gnu.org/software/gsl/

- Example: **`GSL_LS_with_QR.c`**

Reusing DynamicArrays code

- HW8/9 requires reusing your HW2 DynamicArray code by changing the definition of “Data”.
  - DynamicArrays.h was built with #define options
- Use –DHW8 to enable the HW8 section
  - “Data” can be defined as something new!
  - Like: double X;
  - All your old DynamicArrays.c code will work with the “new data”

```
#ifdef HW8
typedef struct {
  /* add here */
} Data;
#elif HW9
typedef struct {
   /* add here */
} Data;
#else
typedef struct {
   int Num;
   char String[MAX_STR_LEN];
} Data;
#endif
```

Fun with LS

- The following table contains data on the amount of garbage (in millions of tons per day) a city generated from 1975 to 2010

year	1975	1980	1985	1990	1995	2000	2005	2010
trash	86.0	99.8	115.8	125.0	132.6	143,1	156.3	169.5

  – What landfill capacity will be required in 2014?

Octave Code

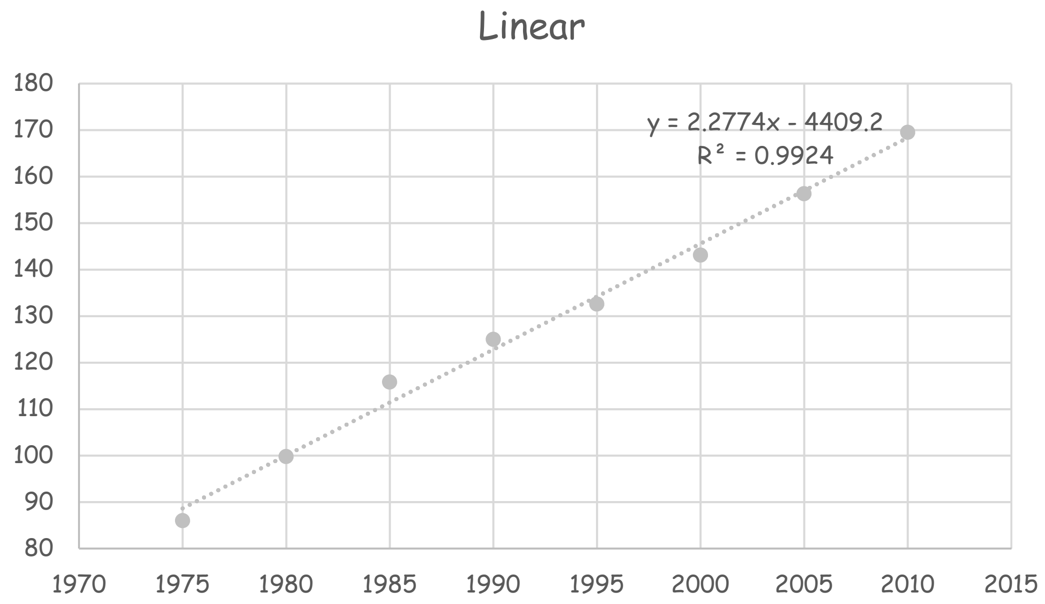
```
%% Load Data For Problem 1 - waste projection
    load -ascii waste.txt
    x=waste(:,1);
    y=waste(:,2);
    n = length(x);         % find #of data points
    plot(x,y,'s');
    title('Experimental Dataset #1')

   A=[ones(n,1) x];     % Linear Fit.  Form A and b
   b=y;
   x_ls=A\b;               % solve
    f=flipud(x_ls)‘       % Make Polynomial pretty

%% Plot linear fit over current figure
  hold on
  plot(x,polyval(f,x),'-r')
  hold off
```

Solution

- Use LS to calculate a data model
  - Results: f(x) = 2.2774x -4409.1690
- At year 2015
  - 178 ton capacity required



Problem 1

Find the linear least squares equation for:

{(-2;-3), (-2; 1), (-1; 2), (1;-3)}

Ans: (A'A)z = A'b

$$A = \begin{bmatrix} 1 & -2 \\ 1 & -2 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} \quad z = \begin{bmatrix} z1 \\ z2 \end{bmatrix} \quad b = \begin{bmatrix} -3 \\ 1 \\ 2 \\ -3 \end{bmatrix}$$

$$A' = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -2 & -2 & -1 & 1 \end{bmatrix}$$

Problem 1

$$A'A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -2 & -2 & -1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -2 \\ 1 & -2 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}$$

$$A'A = \begin{bmatrix} 1+1+1+1 & -2-2-1+1 \\ -2-2-1+1 & 4+4+1+1 \end{bmatrix} = \begin{bmatrix} 4 & -4 \\ -4 & 10 \end{bmatrix}$$

$$A'b = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -2 & -2 & -1 & 1 \end{bmatrix} * \begin{bmatrix} -3 \\ 1 \\ 2 \\ -3 \end{bmatrix}$$

$$A'b = \begin{bmatrix} -3+1+2-3 \\ 6-2-2-3 \end{bmatrix} = \begin{bmatrix} -3 \\ -1 \end{bmatrix}$$

Problem 1

(A'A) z = A'b

$$\begin{bmatrix} 4 & -4 \\ -4 & 10 \end{bmatrix} z = \begin{bmatrix} -3 \\ -1 \end{bmatrix}$$

skipping the G matrix step

row2 = row2 + 4/4 row1 = [-4 10 -1] + [4 -4 -3] = [0 6 -4]

$$\begin{bmatrix} 4 & -4 \\ 0 & 6 \end{bmatrix} z = \begin{bmatrix} -3 \\ -4 \end{bmatrix}$$

back substitution

$$\begin{bmatrix} z1 \\ z2 \end{bmatrix} = \begin{bmatrix} -1.4167 \\ -.666 \end{bmatrix}$$

f(x) = -1.4167 - .6667x

Problem 2

Find the quadratic least squares solution to the same data: {(-2;-3), (-2; 1), (-1; 2), (1;-3)}

Ans. (A'A)z = A'b

$$A = \begin{bmatrix} 1 & -2 & 4 \\ 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad z = \begin{bmatrix} z1 \\ z2 \\ z3 \end{bmatrix} \quad b = \begin{bmatrix} -3 \\ 1 \\ 2 \\ -3 \end{bmatrix}$$

(A'A)z = A'b

$$\begin{bmatrix} 4 & -4 & 10 \\ -4 & 10 & -16 \\ 10 & -16 & 34 \end{bmatrix} z = \begin{bmatrix} -3 \\ -1 \\ -9 \end{bmatrix}$$

Problem 2

The G matrix $\begin{bmatrix} 4 & -4 & 10 & -3 \\ -4 & 10 & -16 & -1 \\ 10 & -16 & 34 & -9 \end{bmatrix}$

row2 = row2 + row1 = [-4 10 -16 -1] + [4 -4 10 -3] = [0 6 -6 -4]

row3 = row3 - 10/4row1 = [10 -16 34 -9] - 2.5*[4 -4 10 -3] = [0 -6 9 -1.5]

$$\begin{bmatrix} 4 & -4 & 10 & -3 \\ 0 & 6 & -6 & -4 \\ 0 & -6 & 9 & -1.5 \end{bmatrix}$$

row3 = row3 + row2 = [0 -6 9 -1.5] +[0 6 -6 -4] = [0 0 3 -5.5]

$$\begin{bmatrix} 4 & -4 & 10 & -3 \\ 0 & 6 & -6 & -4 \\ 0 & 0 & 3 & -5.5 \end{bmatrix}$$

Problem 2

back to normal form

$$\begin{bmatrix} 4 & -4 & 10 \\ 0 & 6 & -6 \\ 0 & 0 & 3 \end{bmatrix} \mathrm{z} = \begin{bmatrix} -3 \\ -4 \\ -5.5 \end{bmatrix}$$

back substitution

$$\begin{bmatrix} \mathrm{z}1 \\ \mathrm{z}2 \\ z3 \end{bmatrix} = \begin{bmatrix} 1.333 \\ -2.5 \\ -1.833 \end{bmatrix}$$

$y = -1.8333x^2 - 2.5x + 1.3333$

Problem 3

Defined the third order least squares A matrix for the following data:

{(-2;-3), (-2; 1), (-1; 2), (1;-3)}

$$A = \begin{bmatrix} 1 & -2 & 4 & -8 \\ 1 & -2 & 4 & -8 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$