General linear least-squares regression:

Linear Regression:

Fit a curve of the form $y = ax+b$ to data points

Use *polyfit* (x, y, 1)

Polynomial Regression:

Fit a curve of the form $y = a_n x^n + a_{n-1} x^{n-1} + \ldots\ a_1 x + a_0$ to data points

Use *polyfit* (x, y, n), where n is the order of the polynomial

Linear regression is a special case ($n = 1$) of polynomial regression

General linear least squares regression:

Fit a curve of the form $y = a_0 z_0(x) + a_1 z_1(x) + \ldots\ a_m z_m(x)$ to data points

$z_0, z_1, \ldots\ z_m$ (the *basis functions)* are arbitrary functions of x

Polynomial regression is a special case ($z_0 = 1$, $z_1 = x$, $z_2 = x^2$, …) of the general case

Example:

We can fit a curve of the form $y = a_0(1) + a_1\sin(\omega x) + a_2\cos(\omega x)$ to data points

In this case the basis functions are (1), $\sin(\omega x)$, and $\cos(\omega x)$

$a_{0,}$ a_1, and a_2 are chosen to as to minimize the sum of the squares of the errors

The "linear" in general linear least squares regression come from that fact that y is a linear combination of functions of x.

The basis functions themselves can be highly nonlinear (e.g. *sin* and *cos*)

The basis functions must involve only constants and x

$y = a_0(1\text{-}\exp(-a_1x))$ is unacceptable as it cannot be converted to the required form

For the first data point (x_1, y_1):

$$y_1 = a_0 z_0(x_1) + a_1 z_1(x_1) + a_2 z_2(x_1) + \dots + a_m z_m(x_1) + e_1$$

where e_1 is the error for this point

Similarly for the second data point (x_2, y_2):

$$y_2 = a_0 z_0(x_2) + a_1 z_1(x_2) + a_2 z_2(x_2) + \dots + a_m z_m(x_2) + e_2$$

And so on.

In matrix form:

$$y = Za + e$$

$$\text{where } Z = \begin{bmatrix} z_0(x_1) & z_1(x_1) & \dots & z_m(x_1) \\ z_0(x_2) & z_1(x_2) & \dots & z_m(x_2) \\ \dots & \dots & \dots & \dots \\ z_0(x_n) & z_1(x_n) & \dots & z_m(x_n) \end{bmatrix}$$

Z has one row for every data point
Z has one column for every basis function
Z_{ij} is the j^{th} basis function evaluated at x_i

We want to find the a that minimizes the sum of the squares of the elements of e

The desired a can be found by solving $Z^T Z a = Z^t y$ (see next slide for proof)

Z is n x $(m + 1)$ Z^T is $(m + 1)$ x n $Z^T Z$ is $(m + 1)$ x $(m + 1)$

a is $(m + 1)$ x 1 y is n x 1 $Z^t y$ is $(m + 1)$ x 1

These m+1 equations in m+1 unknowns are called the *normal equations*

Example:

```
% basis functions = (1), sin(ωx), cos(ωx)

Z = zeros (length(x), 3);
for k = 1 : length(x)
   Z(k, 1) = 1 ; Z(k,2) = sin (w*x(k)); Z(k,3) = cos (w*x(k));
end

Zt = Z';

a = (Zt* Z) \ (Zt * y);   % solve Z^T Z a = Z^t y
```

$$y = Za + e \quad \Rightarrow \quad e = y - Za$$

For interested students only, not in text:

$$S_r = \sum_{i=1}^{n} e_{2i} = e^T e$$

$$= (y - Za)^T (y - Za) = (y^T - a^T Z^T)(y - Za)$$

$$= y^T y - y^T Za - a^T Z^T y + a^T Z^T Za$$

$$\frac{\partial S_r}{\partial a} = -2Z^T y + 2Z^T Za$$

at minimum S_r : $\quad \dfrac{\partial S_r}{\partial a} = -2Z^T y + 2Z^T Za = 0$

$$\therefore \quad Z^T Za = Z^T y$$

QR Factorization:

$Z^T Z\, a = Z^t y$ can be ill conditioned (sensitive to round-off errors)

QR factorization is more robust

Z is factored into Q and R ($Z = QR$)

Q is an (m+1) x (m+1) orthogonal matrix ($Q^{-1} = Q^T$)
R is an (m+1)xn upper triangular matrix

We actually need only Q_0 (first m+1 columns of Q) and R_0 (first n rows of R)

$Q_0 R_0 = Z$ (this is still the case because the rest of R is all zero)

The least squares solution is found by solving R_0 a $=Q_0{}^T$y

Matlab:

[Q0, R0] =qr (Z, 0); % the optional 0 gives only the required parts of Q and R
A = R0\(Q0' *y);

General least squares and left division:

Assuming that $n > m + 1$ (more points than functions) $Z\,a = y$ is *overdetermined*

Number of equations is greater than number of unknowns
No unique solution

In such cases the left division operator produces a “best fit” (least squares) solution

Once Z has been created a can be found directly using left division:

```
a = Z\y;     % fit curve to data points
```

Matlab uses QR factorization to find the least squares solution

Example:

```
x =   [   -8.0   -6.0   -4.0   -2.0     0    2.0    4.0    6.0    8.0]';
y =   [ -817.5 -279.9 -139.4  -41.6  -23.8   -8.7   36.0  158.1  339.4 ]';
```

We want to fit a curve of the form $y = a_0x^3+a_1x^2+a_2x+a_3$

This is polynomial regression:

```
p = polyfit (x, y, 3);

% p is  1.1105   -3.2687   0.2947   0.7874

f = @(x) p(1) * x .^ 3 + p(2) * x .^ 2 + p(3) * x + p(4);
r = correlate (x, y, f);

% r is  0.9936
```

We can also use general least squares regression with

$$z_0(x) = x^3, \quad z_1(x) = x^2, \quad z_2(x) = x, \quad z_3(x) = 1$$

Creating *Z*:

```
Z = zeros (length(x), 3);  % pre allocate for efficiency
for k = 1 : length(x)
   Z(k, 1) = x(k)^3;
   Z(k, 2) = x(k)^2;
   Z(k, 3) = x(k);
   Z(k, 4) = 1;
end
```

Using normal equations:

```
Zt = Z';

a = (Zt* Z) \ (Zt * y)   % solve Zt Z a = Zt y

% a’ is  1.1105   -3.2687    0.2947    0.7874
```

Normal equations perhaps not a very good idea here (although the answer is OK):

```
c = cond(Zt* Z);

% c is 1.5999e+005,  the matrix is very ill conditioned
```

Using QR decomposition:

```
[Q0, R0] = qr(Z, 0);

a = R0 \ (Q0' * y);

% a’ is  1.1105   -3.2687    0.2947    0.7874
```

Using left division:

```
a = Z \ y;

% a’ is  1.1105   -3.2687    0.2947    0.7874
```