The method of least squares is widely used to select parameters for statistical models. The most familiar example being the linear regression which models the response variable y as a function of the covariates.

A simple example would be the model

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i$$

where ε_i are iid normals with mean 0 and variance σ^2. To illustrate, we can make a simulation with

```
> x.i = rep(seq(0,2,length=10),4)
> x.i
 [1] 0.0000 0.2222 0.4444 0.6667 0.8889 1.1111 1.3333 1.5556
 [9] 1.7778 2.0000 0.0000 0.2222 0.4444 0.6667 0.8889 1.1111
[17] 1.3333 1.5556 1.7778 2.0000 0.0000 0.2222 0.4444 0.6667
[25] 0.8889 1.1111 1.3333 1.5556 1.7778 2.0000 0.0000 0.2222
[33] 0.4444 0.6667 0.8889 1.1111 1.3333 1.5556 1.7778 2.0000
> y.i = 1 + 2*x.i + 3*x.iˆ2 + rnorm(40,mean=0,sd=1)
> plot(x.i,y.i)                 # shows slight curve
```

1 Minimizing functions in R

To try and fit the model

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i$$

to the simulated data, the method of least squares would select $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$ which *minimize*

$$\sum_i \left[y_i - \widehat{\beta}_0 + \widehat{\beta}_1 x_i + \widehat{\beta}_2 x_i^2 \right]^2 .$$

A "brute force" approach would be to use one of R's functions for minimizing a function. These are (from their help pages) for one-dimensional optimization

- `uniroot()` The function 'uniroot' searches the interval from 'lower' to 'upper' for a root (i.e. zero) of the function 'f' with respect to its first argument. (Use with the derivative.)
- `optimize()`The function 'optimize' searches the interval from 'lower' to 'upper' for a minimum or maximum of the function 'f' with respect to its first argument.

And for multivariable optimization

- `optim()` General-purpose optimization based on Nelder-Mead, quasi-Newton and conjugate-gradient algorithms. It includes an option for box-constrained optimization and simulated annealing.
- `nlm()` This function carries out a minimization of the function 'f' using a Newton-type algorithm. See the references for details.

Lets see how to do this with the `optim()` function.
First we define a function of the parameters we wish to minmimze. This is

```
f = function(beta)  sum((y.i - (beta[1] + beta[2]*x.i + beta[3]*x.i^2))^2)
```

The vector `beta` contains our coefficients. Notice, their indexing is off from above where we used β_0, β_1 and β_2. Also notice that the value of `x.i` and `y.i` are found outside of the function and are not passed in here as arguments. How this is done is the topic of *scoping*.

To find the value that minimize this function, you only need to provide an initial guess for the parameters. For example, if we think they are near $1, 1, 1$ we could minimize f with

```
> beta.0 = c(1,1,1)
> optim(beta.0,f)
$par
[1] 1.072 1.847 3.166

$value
[1] 41.12

$counts
function gradient
     100       NA

$convergence
[1] 0

$message
NULL
```

The coefficients are found after `$par`. This means they can be found with `optim(c(1,1,1),f)$par`

```
> optim(c(1,1,1),f)$par
[1] 1.072 1.847 3.166
```

Recall, the data is simulated from the values $1, 2, 3$.

2 `lm()` for linear models

R is used for *statistical modeling* and so should have built-in routines to handle this type of standard problem and does. The `lm()` function handles linear models of which our example is. Recall, a linear model is linear in the paramters, not necessarily the covariates. To find *least square estimates* for the coefficients in our example we could also do

```
> lm(y.i ~ x.i + I(x.i^2))

Call:
lm(formula = y.i ~ x.i + I(x.i^2))

Coefficients:
(Intercept)          x.i     I(x.i^2)
       1.07         1.85         3.17
```

2.1 Model formulas

To use `lm()` you need to learn a little about the model formula. The model

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i$$

is expressed in R with the syntax

```
y.i ~ x.i + I(x.i^2)
```

The basic parts are

- The `~` which is read "is modeled by"
- The response variable, `y.i`, is on the left of the `~`.
- The covariates on the right side are represented in a manner different from ordinary mathematical notation.
  - First, there is an implicit intercept (a β_0) so it isn't specified. Rather you can unspecify it by adding `-1` to your formula.
  - Next, the `+` is not addition of numbers, but an additional covariate. In the example it "adds" `x.i^2` to the model.
  - The `*` is for interaction terms and not multiplication.
  - The `^` is not a power in the notation but rather gives interactions upto a certain level. So
    ```
    (a + b)^2
    ```
    in the model formula would correspond to three terms in the model

    $$\alpha a_i + \beta a_i * b_i + \gamma b_i$$

    As we want the power to be a power in this example, we need to *isolate* the power symbol using `I()` as illustrated.

More details on the notation are available in the built in manual "An Introduction to R" in the section "Statistical models in R". To find this, type `help.start()` and browse for it.

2.2 using `lm()`

Using the `lm()` function is a simple as specifying the model formula and optionally the data sets where the variable names come from. In our example this isn't applicable.

The result appears at first glance to just contain the coefficients, but actually contains much more. Usually, one saves the result and then can apply one of several functions to it. For example, the `summary` command gives a summary of the statistical inference for the model based on an assumption of iid errors with a normal distribution ($\mathsf{Normal}(0, \sigma)$).

```
> lm.res = lm(y.i ~ x.i + I(x.i^2))
> summary(lm.res)

Call:
lm(formula = y.i ~ x.i + I(x.i^2))

Residuals:
    Min      1Q  Median      3Q     Max
-2.2199 -0.5220 -0.0529  0.7872  1.6889

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    1.072      0.414    2.59    0.014 *
x.i            1.847      0.965    1.91    0.063 .
I(x.i^2)       3.167      0.464    6.82    5e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.05 on 37 degrees of freedom
Multiple R-Squared: 0.965,      Adjusted R-squared: 0.963
F-statistic:  514 on 2 and 37 DF,  p-value: <2e-16
```

The `plot()` function will provide some diagnostic plots

```
> plot(lm.rs)
...skipped...
```

The functions, `resid()` and `coef()` return the residuals and the estimated coefficients:

```
> plot(resid(lm.res))           # not shown
> coef(lm.res)
(Intercept)         x.i    I(x.i^2)
      1.072       1.847       3.167
```

2.2.1 Using `update()` to change the model

We may wish to consider other models. For example, the following models are a nested family of models

$$
\begin{aligned}
y_i &= \beta_0 + \varepsilon_i \\
y_i &= \beta_0 + \beta_1 x_i + \varepsilon_i \\
y_i &= \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i \\
y_i &= \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \varepsilon_i .
\end{aligned}
$$

One can use the following model formulas for these

```
> lm(y.i ~ 1)
> lm(y.i ~ 1 + x.i)
> lm(y.i ~ 1 + x.i + I(x.i^2))
> lm(y.i ~ 1 + x.i + I(x.i^2) + I(x.i^3))
```

Or, one can use the `update()` function to add terms to your model in a more convenient manner. Thus, we could get the above with

```
> lm.0 = lm(y.i ~ 1)
> lm.0

Call:
lm(formula = y.i ~ 1)

Coefficients:
(Intercept)
       7.38

> lm.1 = update(lm.0, . ~ . + x.i)
> lm.2 = update(lm.1, . ~ . + I(x.i^2))
> lm.3 = update(lm.2, . ~ . + I(x.i^3))
> lm.3

Call:
lm(formula = y.i ~ x.i + I(x.i^2) + I(x.i^3))

Coefficients:
(Intercept)          x.i     I(x.i^2)     I(x.i^3)
      0.980        2.605        2.167        0.333
```

The `update()` function takes a the result of an `lm()` call and allows you to add or drop terms. The `.` is the left or right hand side of the old model in the new formula.

2.3 Model selection using sum of squares

A general rule for model selection is to penalize those models with more parameters. A formula to do so is

$$\frac{\text{Sum of Squares}}{n - 2 * m},$$

where m is the number of parameters, n the size of the sample and the sum of squares, the value of the sum of squares after minimizing.

In R the sum of squares is found by adding the squared residuals. Thus the values could be found with

```
> sum(resid(lm.3)^2)/(n -2*4)
[1] 1.280
> sum(resid(lm.2)^2)/(n -2*3)
[1] 1.209
> sum(resid(lm.1)^2)/(n -2*2)
[1] 2.577
> sum(resid(lm.0)^2)/(n -2*1)
[1] 31.13
```

Recall, `resid()` will find the residuals. By this criterion the model

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i$$

would be preferred.

2.4 using `stepAIC()` (MASS)

The above criterion and analysis takes no account of the error terms having an assumed distribution.

In the case that the error terms are assumed to be normally distributed, the AIC criterion may be used for model selection. As before, one "chooses" the model with the lowest AIC. The AIC value is defined in terms of likelihood.

This process can be somewhat tedious to find in practice. The `stepAIC()` function from the MASS library will do the work for us

```
> stepAIC(lm.3)
Start:  AIC= 8.94
 y.i ~ x.i + I(x.i^2) + I(x.i^3)

           Df Sum of Sq  RSS  AIC
- I(x.i^3)  1       0.2 41.1  7.1
- I(x.i^2)  1       0.8 41.7  7.7
- x.i       1       1.6 42.5  8.4
<none>                  41.0  8.9

Step:  AIC= 7.1
 y.i ~ x.i + I(x.i^2)

           Df Sum of Sq  RSS  AIC
<none>                  41.1  7.1
- x.i       1       4.1 45.2  8.9
- I(x.i^2)  1      51.6 92.8 37.6

Call:
lm(formula = y.i ~ x.i + I(x.i^2))

Coefficients:
(Intercept)          x.i     I(x.i^2)
       1.07         1.85         3.17
```

Notice it chose the "correct" model in this case – the quadratic one.

3 `nls()` for non-linear least squares

The `lm()` model works on linear models. Other types of models exists. General linear models are one extension. Other non-linear models may be involved too. For example, seeing the growth in our example, one may be tempted to fit it with an exponential of the type

$$y_i = ae^{bx_i} + \varepsilon_i$$

This is a non-linear model as the coefficients (a, b) do not enter in a linear manner.

One can still use least squares methods here though to estimate a and b. The estimators, $\widehat{a}$, $\widehat{b}$ would be found by minimizing

$$\sum (y_i - \widehat{a}e^{\widehat{b}x_i})^2$$

This could be done using `optim()` as before

```
> f = function(ab) sum( (y.i - ab[1]*exp(ab[2]*x.i))^2 )
> optim(c(1,1),f)$par
[1] 1.854 1.142
```

However, the `nls()` function performs nonlinear least squares and returns a model object with the helper functions such as `coef()` etc. defined.

To use `nls()` is similar to `lm()`, but the models are defined differently. In fact, they are perhaps more natural as they use usual notations.

For example, we can define the right hand side with a function

```
> f = function(x.i,a,b) a * exp(b*x.i)
```

Then call `nls()` with the initial starting points for our parameters a, b as follows

```
> nls(y.i ~ f(x.i,a,b),start = list(a=1,b=1))
Nonlinear regression model
  model:  y.i ~ f(x.i, a, b)
   data:  parent.frame
    a     b
1.854 1.142
 residual sum-of-squares:  51.57
```

Again, this answer contains more than meets the eye. To see more, we store it, and then illustrate some additional functions

```
> nls.res = nls(y.i ~ f(x.i,a,b),start = list(a=1,b=1))
> summary(nls.res)
Formula: y.i ~ f(x.i, a, b)

Parameters:
  Estimate Std. Error t value Pr(>|t|)
a   1.8538     0.1637    11.3  9.6e-14 ***
b   1.1420     0.0512    22.3  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.16 on 38 degrees of freedom

Correlation of Parameter Estimates:
       a
b -0.973
```

The p values are calculated under the assumption of iid normal errors with mean zero and standard deviation estimate by the residual standard error.

The usual helper functions `coef()`, `formula()`, ‘`resid()`, `print()`, `summary()`, `AIC()`, `fitted()` and `vcov()`. For example,

```
> AIC(nls.res)                  # AIC of model
[1] 127.7
> plot(resid(nls.res))          # plot of residuals
```