Principles of Technical Computing with Matlab (Optimization)

Miracle Amadi, Veera Vilkkilä

Prerequisites: Matlab basics
Software used: `MATLAB`

General form of a model

Generally, a model may be written in the form

$$s = f(x, \theta, \text{const})$$
$$y = g(s)$$

where

s	state
x	input variables (experimental(observation) conditions(points))
θ	estimated parameters (unknown parameters for estimation)
const	known constants or fixed values
y	the observables
f	the model function
g	the observation function

Linear least squares estimation in Matlab

Given a model $f(x, \theta)$ and observed data y, the idea of parameter estimation and optimization is to find the best-fitting values of unknown parameters in the model by optimizing a specific objective function (e.g. the least square (LSQ) objective function)

For linear models, we can derive a direct formula for the LSQ estimator

By linearity here we mean linearity with respect to the unknown model parameters.

$$f(x;\theta) = x_1\theta_1 + x_2\theta_2 \qquad \text{linear model}$$

$$f(x;\theta) = \theta_1 e^{-x\theta_2} \qquad \text{nonlinear model}$$

Mathematically the least squares estimation corresponds to the task

$$\text{argmin} \sum_{i=1}^{n} (\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i, \theta))^2,$$

where n is the number of observations (measurements), and $\mathbf{f}$ is mathematical model evaluated at i'th measurement point. The square must be interpreted as an element-wise operation on the vector elements.

Linear least squares estimation in Matlab

For parameter estimation, we need to create an over-determined system of linear equations.

Consider a linear model $f(\mathbf{x}, \theta) = \theta_0 + \theta_1 \mathbf{x}_1 + \theta_2 \mathbf{x}_2 + \ldots + \theta_p \mathbf{x}_p$.

Assuming we have noisy measurements $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ obtained at points $\mathbf{x}_i = (x_{1i}, x_{2i}, \ldots, x_{ni})$ where $i = 1, \ldots, p$. We can write the model in

matrix notation:

$$\mathbf{y} = \mathbf{X}\theta + \varepsilon,$$

where $\mathbf{X}$ is the coefficient matrix that contains the measured values for the independent variables, augmented with a column of ones to account for the intercept term θ_0:

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1p} \\ 1 & x_{21} & x_{22} & \ldots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \ldots & x_{np} \end{pmatrix}$$

Linear least squares estimation in Matlab

For linear models, the LSQ estimate, that minimizes $SS(\theta) = \|\mathbf{y} - \mathbf{X}\theta\|_2^2$, is obtained as the solution to the normal equations $\mathbf{X}^T\mathbf{X}\theta = \mathbf{X}^T\mathbf{y}$:

$$\hat{\theta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

It is obvious that using matrices makes calculations much easier.

In MATLAB, one can use the 'backslash' shortcut to obtain the LSQ estimate $\hat{\theta} = X \backslash \mathbf{y}$.

Linear least squares estimation in Matlab

Let us consider fitting the linear model equation

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_{11} x_1^2 + \theta_{12} x_1 x_2 + \theta_{22} x_2^2$$

to the data below:

x_1	100.0	220.0	100.0	220.0	75.1	244.8	160.0	160.0	160.0	75.1	75.1
x_2	2.0	2.0	4.0	4.0	3.0	3.0	1.5	4.4	3.0	3.0	3.0
y	25.0	14.0	6.9	5.9	14.1	9.3	18.2	5.6	9.6	14.9	14.8

We specify the data matrices and construct the design matrix by computing the second powers and interaction terms, and augmenting the matrix with a row of ones to account for the intercept in the model:

Linear least squares estimation in Matlab

```
% Fitting a linear model to data
% The data:
X = [100.0 2.0
220.0 2.0
100.0 4.0
220.0 4.0
75.1 3.0
244.8 3.0
160.0 1.5
160.0 4.4
160.0 3.0
75.1 3.0
75.1 3.0];
Y = [25.0 14.0 6.9 5.9 14.1 9.3 18.2 5.6 9.6 14.9 14.8]';
n = length(Y); % number of data points
% constructing the design matrix
X2 = [ones(n,1) X X(:,1).^2 X(:,1).*X(:,2) X(:,2).^2];
b = X2\Y; % LSQ fit
yfit = X2*b; % model response
% visualizing the fit
plot(1:n,Y,'o',1:n,yfit); title('model fit');
```

Linear least squares estimation in Matlab

It is possible to transform a nonlinear model into a linear one by applying some mathematical operations.

However, it is important to note that not all nonlinear models can easily be made linear.

Below is a general step to make a nonlinear model linear:

- Identify the specific nonlinear terms within the model that make it nonlinear (e.g. exponential, divisions, etc.)
- Choose an appropriate transformation that eliminates or simplifies the nonlinear terms (logarithmic transformation, reciprocal transformation, etc.)
- After transformation, manipulate the equation to isolate the dependent variable y on one side and express it in terms of linear combinations of the independent variables x and model parameters (e.g. $y = ax + b$).
- Estimate the parameters of the linear model using the Mathlab 'backslash' operator.
- Back-transform the estimated parameters to their original scale

Linear least squares estimation in Matlab

The model $\theta_1 e^{-x\theta_2}$ is nonlinear with respect to the variable x and the parameter θ_2.

If we take logs of both sides, we get $\ln(y) = \ln(\theta_1) - x\theta_2$.

Renaming $\mathrm{y} = \ln(y), a = \ln(\theta_1)$, we get a linear model $\mathrm{y} = a + \theta_2(-x)$.

Assuming that we perform the estimation using Matlab 'backslash', we can retrieve the original parameter for θ_1 as $\theta_1 = \exp(a)$.

Linear least squares estimation in Matlab

Now, let us consider fitting the linearized model equation (from the slide above) $\ln(y) = \ln(\theta_1) - x\theta_2$ to the data as given in the code below:

```
clear; close all; clc;
X = [2:5:30]'; % X(input) data
Y = [0.8 0.5 0.3 0.2 0.1 0.07]'; % Y(response) data
n = length(X); % extract the length of data
XX = [ones(n,1) -X]; % Define the coefficient matrix
YY = log(Y); % log transformation
params = XX\YY; % estimate the parameters
theta1 = exp(params(1)); % recover the original parameter values
theta2 = params(2);
theta = [theta1, theta2];
Yfit = theta1*exp(-theta2*X); % Get the fitted values

figure;
plot(X,Y,'o',X,Yfit) % plot data and fitted values
title('model fit');
grid on
```

Parameter estimation for nonlinear models

When a model cannot be written so that the parameters appear linearly, or no direct formula is available for LSQ estimator, we must use iterative methods for the sum of squares (SS) minimization for obtaining the least squares solution (fminsearch optimizer).

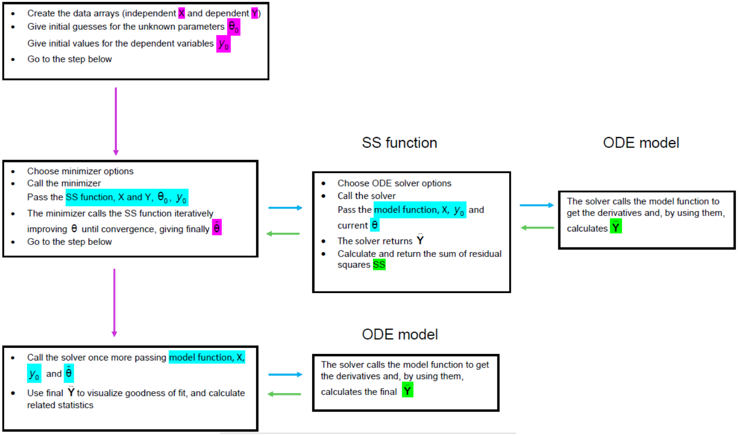
See the flow diagram for optimization below: Notice that:

- the minimizer calls iteratively the SS function until convergence
- the SS function calls the ode solver
- the ode solver calls the ode model function

Therefore, it is good to avoid unnecessary calculations in the model function, or in the SS function.

NOTE: Matlab's fminsearch is a minimiser. If you have a maximisation problem and want to maximize $f(x)$, minimize $-f(x)$, because the point at which the minimum of –f(x) occurs is the same as the point at which the maximum of f(x) occurs.

Parameter estimation for nonlinear models



Parameter estimation for nonlinear dynamical models

Example: Consider again the reaction $A \to B \to C$, modelled as the ODE system

$$\frac{dA}{dt} = -k_1 A$$
$$\frac{dB}{dt} = k_1 A - k_2 B$$
$$\frac{dC}{dt} = k_2 B$$

The data y consists of the values of (any of) the components A, B, C, measured at some sampling instants $t_i, i = 1, 2, ...n$. The unknowns to be estimated are rate constants, $\theta = (k_1, k_2)$.

MATLAB solution

The parameter estimation will be done by the **fminsearch** optimizer. Let us first suppose that only values of B have been measured, with an initial values $A(0) = 1.0, B(0) = C(0) = 0$.

To do the LSQ fitting, we have to write a script file for initializations, a call of the optimizer, and plots for the solution:

```
%SCRIPT file for commands to call FMINSEARCH optimizer
k1    = 0.3;      %initial guesses of the unknown
k2    = 0.2;      %parameters for optimizer
teta  = [k1 k2];  %just collect in 1 vector

ydata= [1.0003      0.0001      -0.0002
       0.4962      0.4512      0.0527
       0.2455      0.5928      0.1601
       0.1227      0.5969      0.2812
       0.0613      0.5448      0.3953
       0.0299      0.4723      0.4976
       0.0146      0.4004      0.5843
       0.0074      0.3354       0.658
       0.0031      0.2787      0.7184
       0.0011      0.2297       0.768
       0.0012      0.1879      0.8107];
```

MATLAB solution

```
t      = [0:1:10]';    %the sampling instants
y      = ydata(:,2);   %assume only B measured is measured
data   = [t y];        %data for the fitting:
                       %sampling instants t and measured B
s0     = [1 0 0];      %initial values for ODE

% Call the optimizer:
teta_opt = fminsearch(@my1lsq,teta,[],s0,data);
% INPUT: my1lsq, the filename of the objective function
%        teta,   the starting point for optimizer
%        []      options (not used)
%        s0,data    parameters needed in 'my1lsq'
% OUTPUT:  teta_opt,  the optimized value for teta

%ODE solver called once more, to get the optimized solution
k1 = teta_opt(1);
k2 = teta_opt(2);
[t,s] = ode23(@myfirstode,t,s0,[],k1,k2);
plot(t,y,'o',t,s)  %plot the data vs solution
```

MATLAB solution

The LSQ objective function is coded in the 'my1lsq' function:

```
function lsq = my1lsq(teta,s0,data);
%INPUT   teta,  the unknowns k1,k2
%        s0, data  the constants needed:
%        s0 initial values needed by the ODE
%        data(:,1) time points
%        data(:,2) responses: B values
%OUTPUT  lsq value

t     = data(:,1);
y_obs = data(:,2);   %data points
k1 = teta(1); k2 = teta(2);

%call the ODE solver to get the states s:
[t,s] = ode23(@myfirstode,t,s0,[],k1,k2);
%the ODE system in 'myfirstode' is just as before: at each
%row (time point), s has the values of the components [A,B,C]
y_cal = s(:,2); %separate the measured B

%compute the expression to be minimized:
lsq   = sum((y_obs-y_cal).^2);
```

Example: Beer cooling

At time $t = 0$, a glass of beer is at an initial temperature T_0. Beer will be cooled from outside by water, which has a fixed temperature $T_{water} = 5°C$. We measure the temperature of the beer at different times and get the following data:

time	0	60	120	180	240	300	360	420	480	540	840	1020	1320
temp ($°C$)	31	28	24	20	17.5	15.5	13.5	12	11	10	8	7	6.5

Note that the heat transfer takes place both through the glass and via the air/water surface and a model for the beer temperature can be written as

$$dT/dt = -k_1(T - T_{\text{water}}) - k_2(T - T_{\text{air}})$$

The temperature of the surrounding air is constant, $T_{air} = 23°C$. Estimate the unknown heat transfer coefficients $\theta = (k1; k2)$.

Example fit: Beer cooling