Principles of Technical Computing with Matlab (Statistics)

Miracle Amadi, Veera Vilkkilä

Prerequisites: Matlab basics
Software used: `MATLAB`

Prerequisities: sample statistics

We usually need to estimate the statistics of a population from measured data by using some sample statistics (estimators).

Let us consider a $n \times p$ matrix of data, where each column contains n measurements for variables $x_1, \ldots, x_p$:

$$\mathbf{X} = \begin{array}{c} \begin{array}{ccccc} \mathbf{x_1} & \mathbf{x_2} & \mathbf{x_3} & \ldots & \mathbf{x_p} \end{array} \\ \begin{pmatrix} x_{11} & x_{12} & x_{13} & \ldots & x_{1p} \\ x_{21} & x_{22} & x_{23} & \ldots & x_{2p} \\ x_{31} & a_{32} & x_{33} & \ldots & x_{3p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \ldots & x_{np} \end{pmatrix} \end{array}$$

Prerequisities: sample statistics

The sample mean for the kth variable: $\bar{x}_k = \frac{1}{n}\sum_{i=1}^{n} x_{ik}$

The sample variance of the kth variable:

$$\mathrm{Var}x_k = \frac{1}{n-1}\sum_{i=1}^{n}(x_{ik} - \bar{x}_k)^2 = \sigma_k^2$$

The sample STD: $\mathrm{Std}(x_k) = \sqrt{\mathrm{Var}(x_k)} = \sigma_k$

The sample covariance between two variables:

$$\mathrm{Cov}(x_k, x_l) = \frac{1}{n-1}\sum_{i=1}^{n}(x_{ik} - \bar{x}_k)(x_{il} - \bar{x}_l) = \sigma_{x_k x_l}$$

The sample correlation coefficient between two variables:

$$\mathrm{Corr}(x_k, x_l) = \frac{\sigma_{x_k x_l}}{\sigma_{x_k}\sigma_{x_l}} = \rho_{x_k x_l}$$

Prerequisities: sample statistics

The sample variances and covariances for p variables can be compactly represented as a sample covariance matrix, which is given as

$$\mathrm{Cov}(\mathbf{X}) = \begin{pmatrix} \sigma_{x_1}^2 & \sigma_{x_1 x_2} & \cdots & \sigma_{x_1 x_p} \\ \sigma_{x_1 x_2} & \sigma_{x_2}^2 & \cdots & \sigma_{x_2 x_p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{x_1 x_p} & \sigma_{x_2 x_p} & \cdots & \sigma_{x_2}^2 \end{pmatrix}$$

Similarly, one can form a correlation matrix using the relationship of covariance and correlation:

$$\mathrm{Corr}(\mathbf{X}) = \begin{pmatrix} 1 & \rho_{x_1 x_2} & \cdots & \rho_{x_1 x_p} \\ \rho_{x_1 x_2} & 1 & \cdots & \rho_{x_2 x_p} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{x_1 x_p} & \rho_{x_2 x_p} & \cdots & 1 \end{pmatrix}$$

Useful MATLAB commands: mean, var, std, cov, corr, corrcov.

Analyzing (Linear) Modeling Results: covariance of estimates

To obtain the statistics for the estimate $\hat{\theta} = (X^T X)^{-1} X^T \mathbf{y}$, we can compute the covariance matrix $\mathrm{Cov}(\hat{\theta})$

Assume i.i.d. measurement error: $\mathrm{Cov}(y) = \sigma^2 \mathbf{I}$.

Then, we can show that

$$\mathrm{Cov}(\hat{\theta}) = \sigma^2 (X^T X)^{-1} \tag{1}$$

Assuming Gaussianity of the measurement errors, it follows that the unknown parameter follows the normal distribution with mean and covariance matrix given by $\theta \sim \mathbf{N}(\hat{\theta}, \sigma^2 (X^T X)^{-1})$.

Note: The measurement error σ^2 can be estimated using repeated measurements.

Analyzing (Linear) Modeling Results: *t-values*

After computing the covariance matrix of the unknown parameters in the model, we can compute the 'signal-to-noise' ratios for each parameter, also known as *t-values*.

For parameter θ_i, the *t-value* is given as

$$t_i = \frac{\hat{\theta}_i}{\text{std}(\hat{\theta_i})}$$

where the standard deviations of the estimates can be read from the diagonal of the covariance matrix in Eq. 1

The higher the t-value is, the smaller the relative uncertainty is in the estimate, and the more sure we are that the term is relevant in the model.

We can retain terms for which the t-values are clearly separated from zero. As a rule of thumb, we can require that, for instance, $|t_i| > 3$

Analyzing (Linear) Modeling Results: $R^2 - value$

The *t-values* do not tell how well the model fits the observations in general.

A classical way to measure the goodness of the fit is the coefficient of determination, or R^2 value which is written as:

$$R^2 = 1 - \frac{\sum(y_i - f(x_i, \theta))^2}{\sum(y_i - \bar{y})^2}$$

Consider the simplest possible model $y = \theta_0$, that is, the data are modeled by just a constant. In this case, the LSQ estimate is just the empirical mean of the data, $\hat{\theta}_0 = \bar{y}$

That is, the R2 value measures how much better the model fits to the data than a constant model. The better the model fit is, the closer the R^2 value is to 1

Analyzing (Linear) Modeling Results: Cross-validation

The R^2 value does not tell about the predictive power of the model, that is, how well the model is able to generalize to new situations.

The predictive power of the model can be assessed, for instance, via cross-validation:

- Leave out a part of the data from X and y.
- Fit the model using the remaining data.
- Using the fitted model, predict the data that were left out.
- Repeat the first three steps so that each response value in y is predicted.

The goodness of the predictions at each iteration can be assessed using the R^2 formula. The obtained number is called the Q^2 value.

Analyzing (Linear) Modeling Results: Cross-validation

Cross-validation is commonly used for model selection. The idea is to test different models with different terms included and choose the model that gives the best Q^2 value.

If the model is either too complex ('over-parameterized') or too crude, the model will predict poorly and give a low Q^2 value.

In practice, the cross-validation procedure can be carried out by stepwise regression, which can be implemented in many ways, for instance by:

- *Forward stepping:* test all terms individually, choose the one with highest Q^2 value. Continue by testing all remaining terms, and choose the one with highest Q^2 value at each step.
- *Backward stepping:* similarly, but starting with the full model, and dropping, one by one, the worst term on each iteration step.

Analyzing (Linear) Modeling Results: Example

We can now compute the statistics of the linear model ($y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_{11} x_1^2 + \theta_{12} x_1 x_2 + \theta_{22} x_2^2$) fitting results in the Optimisation slide

```
% To make the example code below work, you must first
% run the model fitting code
% COPY THE EXAMPLE CODE FROM OPTIMISATION SLIDES HERE

% estimating the covariance
repmeas = [5 10 11]; % indices of repeated measurements
sig = std(Y(repmeas)); % sigma estimate
cov_b = sig^2*inv(X2'*X2);

% t-values
std_b = sqrt(diag(cov_b));
t_b = b./std_b

% R2 value
yfit = X2*b; % model response
R2 = 1-sum((Y-yfit).^2)/sum((Y-mean(Y)).^2)
```

Example cross-validation

```
load('Cval_data.mat')
n=length(x);
predictions = zeros(1,n); % holds the prediction for each data point
for p=1:n
% figure out indices
trainx = setdiff(1:n,p);
testx = p;
% train the model
X = [x(trainx)' ones(length(trainx),1)]; % coefficient matrix
Y=y(trainx)';
%b = inv(X'*X)*X'*y(trainx)'; % estimate parameters
b = X\Y; % estimate parameters
% Test the model; compute the prediction for the left-out data point
predictions(p) = [x(testx)' ones(length(testx),1)]*b;
end
% Check the goodness of the prediction
Q2 = 100 * (1 - sum((y-predictions).^2) / sum((y-mean(y)).^2))
```

Analyzing (Nonlinear) Modeling Results: *covariance of estimates*

As discussed earlier, for nonlinear models, no such direct methods are available, and one has to resort to numerical methods and different approximations.

To obtain the $\mathrm{Cov}(\hat{\theta})$, an approximative error analysis can be performed for the parameters of a nonlinear model, based on linearizing the model at the LSQ estimate $\hat{\theta}$ (details skipped).

The approximative error analysis for nonlinear models, assuming i.i.d. Gaussian errors with measurement error variance σ^2, is given by the covariance matrix

$$\mathrm{Cov}(\hat{\theta}) = \sigma^2 (J^T J)^{-1}$$

Observe that the Jacobian matrix J assumes the role of the design matrix X in the linear case.

Analyzing (Nonlinear) Modeling Results: *covariance of estimates*

The Jacobian matrix J has elements:

$$[\mathbf{J}]_{ip} = \left. \frac{\partial f\left(x_i;\theta\right)}{\partial \theta_p} \right|_{\theta=\widehat{\theta}},$$

where the derivatives are evaluated at the estimate $\hat{\theta}$

The Jacobian can be computed analytically for a simple model.

Analyzing (Nonlinear) Modeling Results: *estimating* σ^2

Often, replicated measurements are not available to estimate the measurement error σ^2.

In this case, the measurement noise can be estimated using the residuals of the fit, using the assumption that $residuals \approx measurement\ error$

An estimate for the measurement error can be obtained using the mean square error (MSE):

$$\sigma^2 \approx MSE = \frac{RSS}{n-p},$$

where RSS (residual sum of squares) is the minimum of the least squares function, n is the number of measurements and p is the number of parameters

Analyzing (Nonlinear) Modeling Results: example computing J analytically

Let us here consider a simple nonlinear model with two parameters, $y = \theta_1(1 - e^{-\theta_2 x})$. The goal is to estimate the parameters and their uncertainty using the data x =(1, 3, 5, 7, 9) and y = (0.076, 0.258, 0.369, 0.492, 0.559).

For this example, it is easy to make the model linear and use linear statistics to estimate the parameters and their uncertainties.

But let's employ nonlinear statistics and show how to compute the Jacobian matrix analytically.

```
%%%%% LSQ fitting with the BOD model
clear all; close all; clc;
b_0 = [1 0.1]; % initial quess for the optimizer
x = (1:2:9)'; % x-data
y = [0.076 0.258 0.369 0.492 0.559]'; % y-data
data = [x,y]; % data matrix
n = length(x); % number of data points
```

Analyzing (Nonlinear) Modeling Results: example computing J analytically

```
%%%% Get estimate for sigma**2 from the residual Sum of Squares
[bmin,ssmin] = fminsearch(@bod_ss,b_0,[],data);

%%%% Compute the Jacobian analytically
J = [1-exp(-bmin(2).*x), x.*bmin(1).*exp(-x.*bmin(2))];

%%%% Compute the covariance and print the parameter estimates
sigma2 = ssmin/(n-2); % std of measurument noise estimated by the resid
C = sigma2*inv(J'*J);
disp('(theta, std, t-values):');
[bmin(:) sqrt(diag(C)) bmin(:)./sqrt(diag(C))]
```

Analyzing (Nonlinear) Modeling Results: example computing J analytically

```
%%%% Visualize the fit
xx = linspace(0,10);
yy = bmin(1)*(1-exp(-bmin(2)*xx));
plot(xx,yy,x,y,'ro');
xlabel('x'); ylabel('y=\theta_1(1-exp (\theta_2 x))');

%%%% Create the sum of square function
function ss = bod_ss(theta,data)
x = data(:,1);
y = data(:,2);
ss = sum((y - theta(1)*(1-exp(-theta(2)*x))).^2);
end
```

Analyzing (Nonlinear) Modeling Results: *covariance of estimates*

Usually computing the Jacobian analytically is difficult, and we need to approximate it numerically.

The numerical approximations for the derivatives in the Jacobian matrix can be computed, using, for instance, the two-sided finite difference formula (see implementation in the function file jacob.m; this function is part of the MCMC toolbox)

$$[\mathbf{J}]_{ip} = \left.\frac{\partial f\left(x_i, \theta\right)}{\partial \theta_p}\right|_{\theta=\widehat{\theta}} \approx \frac{f\left(x_i; \theta + h\right) - f\left(x_i; \theta - h\right)}{2h},$$

where h is a small difference added to the pth component of the θ vector.

Analyzing (Nonlinear) Modeling Results: example computing J via numerical approximation

Let us again consider the usual $A \rightarrow B \rightarrow C$ reaction model, where the unknown parameters are the reaction rate coefficients, $\theta = (k1, k2)$. The data consists of values of the components A and B:

time	1.0	3.0	5.0	7.0	9.0
A	0.504	0.217	0.101	0.064	0.008
B	0.415	0.594	0.493	0.394	0.309

```
% A --> B --> C demo
clear all; close all; clc;
% the data structure
data.time = 1:2:9;
data.ydata = [.504 .415
.217 .594
.101 .493
.064 .394
.008 .309];
data.y0 = [1 0 0];
```

Analyzing (Nonlinear) Modeling Results: example computing J via numerical approximation

```
% calling fminsearch
theta0 = [1 1];
[thopt,ssopt] = fminsearch(@ABCss,theta0,[],data);

% visualization: solve model with thopt and compare to data
t = linspace(0,10);
[ymod,t] = ABCmodel(t,thopt,data.y0);

plot(t,ymod); hold on;
plot(data.time,data.ydata,'o'); hold off;
xlabel('time'); ylabel('concentration');
legend('A','B','C');

% numerical Jacobian
J = jacob(@ABCmodel,[0 data.time],thopt,[],data.y0);

% covariance estimate
sigma2 = ssopt/(10-2);
C = sigma2*inv(J'*J);
```

Analyzing (Nonlinear) Modeling Results: example computing J via numerical approximation

```
% sum of square function
function ss = ABCss(theta,data)
time = [0 data.time]; % adding zero to time vector! because the IC(s) a
ydata = data.ydata;
y0 = data.y0;
ymod = ABCmodel(time,theta,y0);
ymod = ymod(2:end,1:2); % taking A and B, removing initial value
ss = sum(sum((ydata-ymod).^2)); % the total SS
end

% create a function that solves the ABC ODE
function [y,t] = ABCmodel(time,theta,y0)
[t,y] = ode45(@ABCode,time,y0,[],theta);
end
```

Analyzing (Nonlinear) Modeling Results: example computing J via numerical approximation

```
% the function that defines the ODE equation
function dy = ABCode(t,y,theta)
% take parameters and components out from y and theta
k1 = theta(1); k2 = theta(2);
A = y(1); B = y(2);
% define the ODE
dy(1) = -k1*A;
dy(2) = k1*A-k2*B;
dy(3) = k2*B;
dy = dy(:); % make sure that we return a column vector
end
```

Analyzing (Nonlinear) Modeling Results: example computing J via numerical approximation

Let us again consider the beer cooling example:

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

```
%% Script file for fitting the beer model
clearvars; close all; clc;
t = [0:60:540 840 1020 1320]; % time
y = [31 28 24 20 17.5 15.5 13.5 12 11 10 8 7 6.5]; % temperature
data = [t' y']; % collect the data in a matrix
th0 = [0.01 0.01]; % initial guess for the parameters k1 and k2
T0 = 31; % initial value for ode

% compute LSQ estimate
opts = optimset('Display','iter');
[thopt,ssopt] = fminsearch(@beerss,th0,opts,data,T0);

% plot the initial fit
time = linspace(0,1400);
Tfit = beerfun(time,thopt,T0);
plot(time,Tfit,t,y,'o');
```

Analyzing (Nonlinear) Modeling Results: example computing J via numerical approximation

```
% compute the covariance matrix of the parameters
sig2 = ssopt/(length(y)-2);
J = jacob(@beerfun,t,thopt,[],T0);
C = sig2*inv(J'*J);
th_std = sqrt(diag(C));
thopt'./th_std

% the sum of square cost function
function ss = beerss(theta,data,T0)
t = data(:,1);
Tobs = data(:,2);
Tmod = beerfun(t,theta,T0);
ss = sum((Tobs-Tmod).^2);
end
```

Analyzing (Nonlinear) Modeling Results: example computing J via numerical approximation

```
% function that solves the beer cooling ODE
function T = beerfun(t,theta,T0)
[t,T] = ode45(@beerode,t,T0,[],theta);
end

% Beer cooling ODE function
function dT = beerode(t,T,k)
Tw = 5; Ta = 23;
dT = -k(1)*(T-Tw)-k(2)*(T-Ta);
end
```

Adding Noise to Data

An alternative way to obtain statistics for parameter estimates is to use various Monte Carlo (MC) random sampling methods.

Recall that uncertainty in the model parameters θ in a model

$$y = \mathbf{f}(\mathbf{x}, \theta) + \varepsilon$$

is caused by the noise ε.

The LSQ fit with given data leads to a single estimated value $\hat{\theta}$.

So, to obtain a distribution of θ, a natural idea is to generate new data by adding random noise to the existing data and repeatedly fit different values $\hat{\theta}$.

Simple to implement, and can work well, if the noise is correctly generated so that it agrees with the true measurement noise.

Often the structure of the noise is not properly known.

Bootstrapping

A very popular statistical analysis method, in spirit similar to the above 'adding noise to data' approach.

No new data is generated, but new random combinations of the existing data are used. The basic idea can be written as a pseudo-algorithm as follows:

1. From the existing data $\mathbf{x} = (x_1, \ldots, x_n), \mathbf{y} = (y_1, \ldots, y_n)$, sample new data $\tilde{\mathbf{x}}, \tilde{\mathbf{y}}$ with replacement. In practice, select n indices randomly from $1, \ldots, n$ and choose the data points corresponding to the chosen indices.
2. Compute the fit using the resampled data $\tilde{\mathbf{x}}, \tilde{\mathbf{y}}$.
3. Go to step 1, until a desired number of θ samples are obtained.

Both Bootstrapping and the "adding noise to data' approach suffer two common problems: depends on the success of the optimization step and is CPU intensive (repeated calls to an optimization routine).

Bootstrapping: example

The example below shows bootstrapping for a simple linear model $y = \theta_0 + \theta_1$

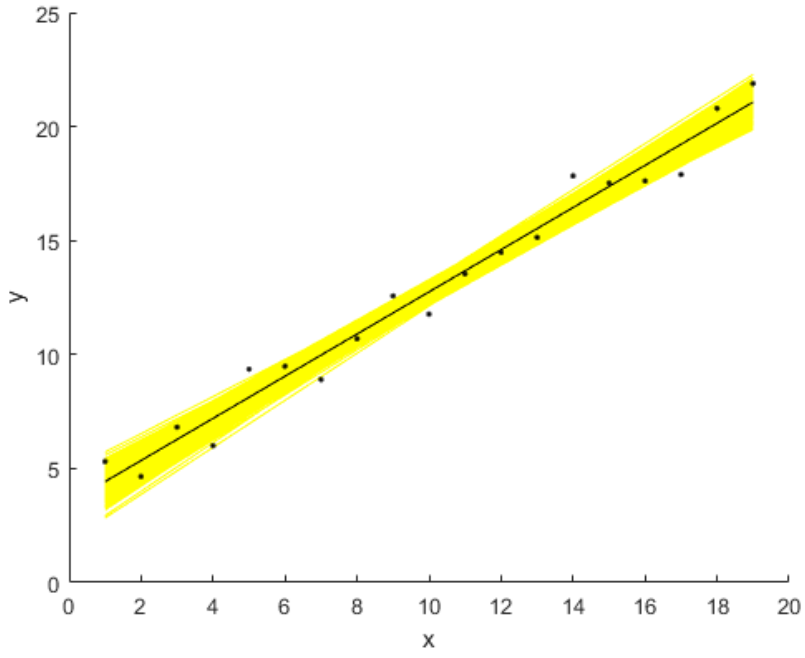
```
xdata = 1:1:19; % x-values to evaluate the model at
ydata = [5.3058 4.6509 6.812  6.006  9.3565   9.4842  8.903  10.692...
12.569 11.77 13.534 14.481 15.133 17.836 17.522 17.608 17.894...
20.795 21.882]; % y-values

n = length(xdata);
X = [ones(n,1) xdata']; % coefficient matrix
Y = ydata';
% estimate parameters
theta = X\Y;
modelfit = X*theta; % model fit with original data

% perform the bootstraps
nsample = 1000; % number of bootstraps to perform
modelfit_ = zeros(nsample,n);
params = zeros(nsample,2);
inds = ceil(n*rand(nsample,n));
%inds = randi(n,nsample,n);
xdata_new = arrayfun(@(ind)xdata(ind),inds);
ydata_new = arrayfun(@(ind)ydata(ind),inds);
```

Bootstrapping: example

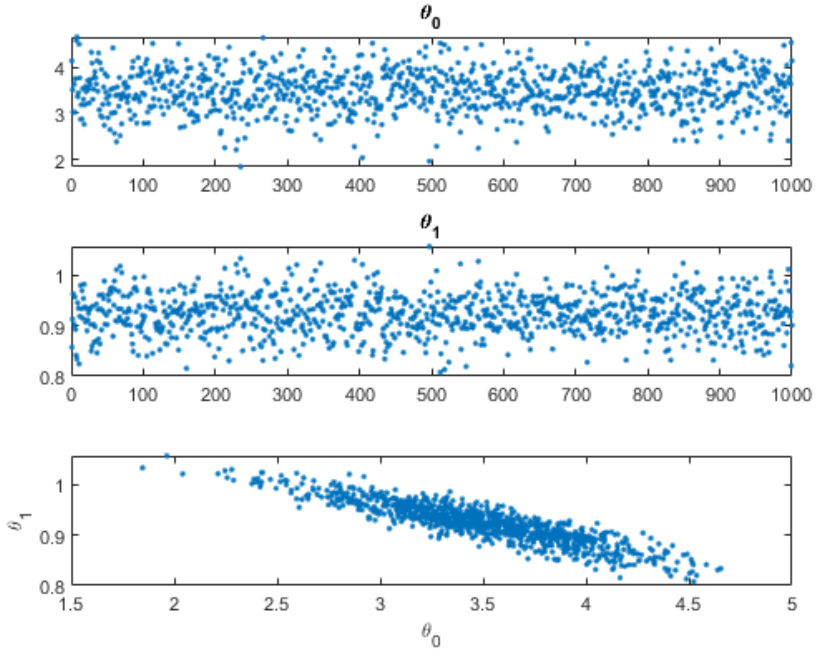
```
for i = 1:nsample
% construct coefficient matrix
X = [ones(n,1) xdata_new(i,:)'];
Y = ydata_new(i,:)';
% estimate parameters
theta_ = X\Y;
% evaluate the model
modelfit_(i,:) = theta_(1) + xdata*theta_(2);
% record the parameters
params(i,:) = theta_;
end

% visualize data and fits
figure;
hold on;
plot(xdata,modelfit_','y')
plot(xdata,modelfit','k')
plot(xdata,ydata,'k.');
xlabel('x');
ylabel('y');
```

Bootstrapping: example

```
% visualize the parameter chains
figure;
subplot(3,1,1)
plot(params(:,1),'.')
title('\theta_0')
subplot(3,1,2)
plot(params(:,2),'.')
title('\theta_1')
subplot(3,1,3)
plot(params(:,1),params(:,2),'.')
xlabel('\theta_0')
ylabel('\theta_1')
```



(a) model fits



(b) parameter chains