Template for parameter estimation with Matlab Optimization Toolbox; including dynamic systems

1. Curve fitting

A *weighted least squares fit* for a model which is less complicated than the system that generated the data (a case of so-called 'undermodeling').
System:

$$y = \begin{cases} e^{-3x} & 0 < x < 1 \\ (x-1)^2 & 1 < x < 2 \end{cases} \tag{1.1}$$

Data (noise model):

$$y_{data} = y + N(0, 0.05) \tag{1.2}$$

with additive zero mean white noise ZMWN with a standard deviation of 0.05.
Model to fit the data:

$$y_{fit} = e^{a(x+b)} \tag{1.3}$$

with parameters to be fitted $\theta = [a,b]$.
The model can be 'forced' to fit the second part of the data ($1 < x < 2$) by assigning different weights to the data. Error function:

$$e = w\left(y_{data} - y_{fit}(x,\theta)\right) = \begin{cases} w_1\left(y_{data} - y_{fit}(x,\theta)\right) & 0 < x < 1 \\ w_2\left(y_{data} - y_{fit}(x,\theta)\right) & 1 < x < 2 \end{cases} \tag{1.4}$$

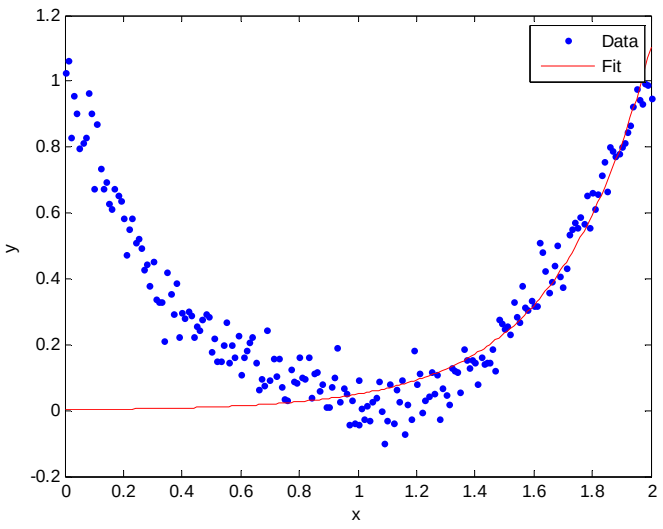
with $w_1 = 1$ and $w_2 = 10$.
To implement and solve the weighted least squares fitting problem in Matlab the function LSQNONLIN of the Optimization Toolbox is used. Optimization algorithms (in fact a minimization is performed) require the user to specify an initial guess θ^0 for the parameters. Here we use θ^0 = [0.1, -1].
The Matlab code in the box below can be copied and paste in the Matlab editor and then saved (or push the Run button which will save and automatically run the code).

```
function myFit
%myFit Weighted least squares fit

%% create the first half of the data
xdata1 = 0:.01:1;
ydata1 = exp(-3*xdata1) + randn(size(xdata1))*.05;
weight1 = ones(size(xdata1))*1;

%% create the second half of the data
% use a different function and with higher weights
xdata2 = 1:.01:2;
ydata2 = (xdata2-1).^2 + randn(size(xdata2))*.05;
weight2 = ones(size(xdata2))*10;
```

```
%% combine the two data sets
xdata = [ xdata1 xdata2 ];
ydata = [ ydata1 ydata2 ];
weight = [ weight1 weight2 ];

%% call |LSQNONLIN|
parameter_hat = lsqnonlin(@mycurve,[.1 -1],[],[],[],xdata,ydata)

%% plot the original data and fitted function
plot(xdata,ydata,'b.')
hold on
fitted = exp(parameter_hat(1).*(parameter_hat(2) +xdata));
plot(xdata,fitted,'r')
xlabel('x'); ylabel('y')
legend('Data', 'Fit')

%% function that reports the error
function err = mycurve(parameter,real_x, real_y)
fit = exp(parameter(1).*(real_x + parameter(2)));
err = fit - real_y;

% weight the error according to the |WEIGHT| vector
err_weighted = err.*weight;
err = err_weighted;
end

end
```

Executing this Matlab program results in:

```
parameter_hat =

    3.1001   -1.9679
```



So $y_{fit} = e^{3.1001(x-1.9679)}$

However, here we do not obtain any information about the accuracy of the estimated $\hat{\theta}$. For example, we do not know how critical the fit depends on all digits in the parameter values returned by Matlab.

2. Parameter estimation for a dynamic model

In the second example we consider a dynamical system. If blood plasma and a tissue or organ of interest can be considered as connected compartments then the following model can be used to describe tissue perfusion:

$$\frac{dC_e}{dt} = \frac{K^{trans}}{\upsilon_e}\left(C_a - C_e\right)$$
$$C_t = \upsilon_e C_e$$

in which the arterial plasma concentration $C_a(t)$ is the input and the tissue concentration $C_t(t)$ is the output.

This linear differential equation model can be transformed into 2 descriptions often used in system and control engineering.

a) State-space format:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases}$$

$$\Rightarrow u = C_a, \quad x = C_e, \quad y = C_t \;\; A = -\frac{K^{trans}}{\upsilon_e}, \quad B = \frac{K^{trans}}{\upsilon_e}, \; C = \upsilon_e, \quad D = 0$$

b) Transfer function:

$$H(s) = \frac{C_t(s)}{C_a(s)} = \frac{K^{trans}}{s + K^{trans}/\upsilon_e}$$

The Control Toolbox from Matlab can be used to implement and simulate this model.

The parameters $\theta = [K^{trans}, \upsilon_e]$ need to be estimated from clinical data. Tissue perfusion can be measured with Dynamic Contrast Enhanced MRI (DCE-MRI). A contrast agent is injected into the bloodstream, circulates through the body and diffuses into tissues and organs. The kinetics of contrast enhancement is measured in a voxel in the tissue of interest, which can be translated into a concentration.

The following noise (error) model is assumed/proposed:

$$y = C_t + \xi \tag{1.5}$$

with ξ ZMWN and therefore the sum of squared errors is used as estimator.

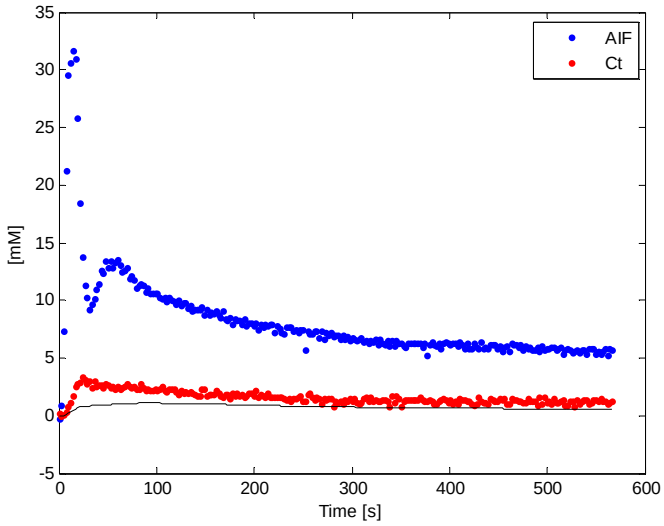
In this case the purpose of the model is not just to be able to describe the data, but differences in the estimated parameter values have a biological meaning and might be used for clinical diagnosis and decisions. Hence it is critically important to assess the accuracy of the estimates. For this problem the estimator is the Maximum Likelihood Estimator (MLE) and it is possible to calculate the covariance

matrix of the parameter estimates, which quantifies the accuracy of the estimate. (The inverse of the covariance matrix is known as the Fisher Information Matrix.)
In the subsequent Matlab code it is shown how the covariance matrix can be calculated from the outputs provided by the LSQNONLIN function. As initial guesses θ^0 = [2e-3, 0.1] will be used.

```
function compartment
%compartment Estimating pharmacokinetic parameters using a dynamic 2
%compartment model
%Loads datafile pmpat010.txt.

%% Load data
datafile = 'pmpat010.txt';
data = load(datafile);
t=data(1,:);    %[s]
Ca=data(2,:);   %[mM] arterial input function
Ct=data(3,:);   %[mM] tissue response

figure; plot(t,Ca,'.b',t,Ct,'.r'); hold on
xlabel('Time [s]'); ylabel('[mM]')
legend('AIF','Ct')

N=length(Ct);

%% Simulate model with initial parameter values
%Simulate
Ktrans = 2e-3; ve = 0.1;
p = [Ktrans, ve];
[y,t] = compart_lsim(t,Ca,p);
plot(t,y,'k');

%% Estimate parameters and variances
optim_options = optimset('Display', 'iter',...
...%'TolFun', 1e-6,...          %default: 1e-4
...%'TolX', 1e-6,...            %default: 1e-4
'LevenbergMarquardt', 'on');    %default: 'off'
%optim_options = [];
p0 = [Ktrans, ve];
[p,resnorm,residual,exitflag,OUTPUT,LAMBDA,Jacobian] =
lsqnonlin(@compart_error, p0, [],[],optim_options, t,Ca,Ct);
disp(' ')
p

%Estimated parameter variance
Jacobian = full(Jacobian);  %lsqnonlin returns the Jacobian as a sparse
matrix
varp = resnorm*inv(Jacobian'*Jacobian)/N
stdp = sqrt(diag(varp));    %standard deviation is square root of variance
stdp = 100*stdp'./p;        %[%]

disp([' Ktrans: ', num2str(p(1)), ' +/- ', num2str(stdp(1)), '%'])
disp([' ve:     ', num2str(p(2)), ' +/- ', num2str(stdp(2)), '%']);

%% Simulate estimated model
[y,t] = compart_lsim(t,Ca,p);
figure; subplot(211); plot(t,Ct,'.r',t,y,'b');
xlabel('Time [s]'); ylabel('[mM]')
legend('data','model')
```

```
xi = Ct(:)-y(:);    %same as residual from lsqnonlin
subplot(212); plot(t,xi)
xlabel('Time [s]'); legend('residuals \xi')
assen=axis;

%% function to simulate compartment model
function [y,t] = compart_lsim(t,Ca,p)

Ktrans = p(1); ve = p(2);
num = [Ktrans*ve];
den = [ve, Ktrans];
sys = tf(num,den);
[y,t] = lsim(sys,Ca,t);
end

%% function to calculate MLE error
function xi = compart_error(p, t,Ca,Ct)

[y,t] = compart_lsim(t,Ca,p);
xi = Ct(:)-y(:);    %make sure both vectors are columns

%figure(1); plot(t,Ct,'.',t,y,'g'); hold off; drawnow %uncomment this line
to show
%datafir for each optimization iteration (slows down the execution)
end

end %main function
```

When the program is executed, first a figure is returned with the input data (concentration in blood plasma, AIF, in blue), the output data (tissue compartment, Ct, in red) and the model simulation given the initial parameter values θ^0 (solid black line).

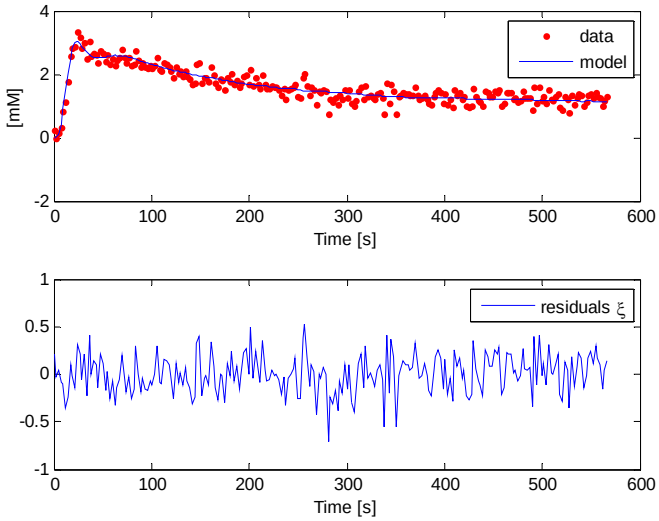


Secondly the results of the parameter estimation (vector of estimated parameters, estimated parameter covariance matrix, and the parameter estimates with their relative standard deviation, in (%)):

```
p =
```

```
    0.0097    0.2018

varp =

  1.0e-005 *

    0.0133   -0.0025
   -0.0025    0.2574

 Ktrans: 0.0096992 +/- 3.764%
 ve:     0.20176 +/- 0.79515%
```

Finally, a figure with a plot of the identified model and the data (top panel), and a plot of the difference between model and data (residuals, bottom).



3. Template

Finally a template is provided to estimate a *subset* of the parameters in a model (some parameters are assumed to be known and therefore are fixed) and the model is composed of a set of coupled first order *nonlinear differential equations* (simulated with one of the Matlab ode-solvers).

```
Template for Nonlinear Least Squares estimation and Fisher Information
Matrix

%% Load data
%for example from Excel
data = xlsread(...
%obtain / define experimental time vector
texp =

%% Simulation time and input
tsim = texp;    %not necessarily the same
%Input matrix (1st column time (can be different from experimental time),
2nd column, etc. corresponding input values
tu = [texp,u1, u2,...

%% Parameters and initial conditions of state variables
k1 = ...
%assign parameters to vector with known (fixed) parameters and a vector of
parameters to be estimated
pest = [...
pfix  = [...
%total parameter vector: p = [pfix pest]

%the same for the initial conditions
x0est = [...
x0fix  = [...
%total initial conditions: x0 = [x0fix x0est]

%total vector of quantities to be estimated:
px0est=[pest x0est]

%% Optimization algorithm
%Settings lsqnonlin
lb = zeros(size(px0est));
ub = [];
options = optimset('TolFun', 1e-4,...  %default: 1e-4
    'TolX',1e-5,...                    %default: 1e-4
    'LevenbergMarquardt','on',...      %default: on
    'LargeScale','on');                %default: on

%LSQNONLIN: objective function should return the model error vector
[px0est,resnorm,RESIDUAL,EXITFLAG,OUTPUT,LAMBDA,Jacobian] =
lsqnonlin(@objfnc,px0est,lb,ub,options,...
            texp,data,tu,pfix, x0fix);

%% (estimated) Precision:
Jacobian = full(Jacobian);  %lsqnonlin returns the jacobian as a sparse
matrix
%Parameter covariance matrix (inverse of Fisher Information Matrix)
varp = resnorm*inv(Jacobian'*Jacobian)/length(texp);
%Standard deviation is the square root of the variance
stdp = sqrt(diag(varp));
```

```
%% Reconstruct parameter and initial condition vectors
pest = px0est(.:.)
p = [pfix pest];
x0est = px0est(.:.)
x0 = [x0fix x0est];

%% Simulation with estimated parameters
ode_options = [];
[t,x] = ode15s(@odefnc,tsim,x0,ode_options, tu,p);

--
%Objective / cost function for least squares
function e = objfnc(px0est, texp, data, tu,pfix,x0fix)
% e: vector of model residuals

%Reconstruct parameter and initial condition vectors
pest = px0est(.:.)
p = [pfix pest];
x0est = px0est(.:.)
x0 = [x0fix x0est];

%% Simulation
ode_options = [];
[t,x] = ode15s(@odefnc,texp,x0,ode_options, tu,p);

%Model output(s)
y = x(:,1);

%% Model residuals
e = data-y;

--
%Model in state-space format (system of coupled 1st order ODE's)
function dx = odefnc(t,x,tu, p)
% t : current time
% x : state values at time t
% tu: matrix with time points (1st column) and corresponding input signal
value (2nd column)
% p : parameters
% dx: new state derivatives (column vector)

%calculate input u(k) by interpolation of tu
u = interp1(tu(:,1),tu(:,2), t);

%ode's
dx1 = ...

%collect derivatives in column vector
dx = [dx1; dx2 ...];
```