Parameter estimation in non-equidistantly sampled nonlinear state space models;
a Matlab implementation

by Natal van Riel, Eindhoven University of Technology

1 Model structure and simulation

The model structure consists of a set of nonlinear differential equations to describe the dynamics of a general nonlinear input-output system parameterized by a vector $\boldsymbol{\theta}$, which represents the unknown constants in the system. The state equation is:

$$\dot{\mathbf{x}}(t,\boldsymbol{\theta}) = f(\mathbf{x}(t,\boldsymbol{\theta}),\tilde{\mathbf{u}}(t),t,\boldsymbol{\theta}) \qquad \mathbf{x}(t_0,\boldsymbol{\theta}) = \mathbf{x}_0 \tag{1}$$

(dot indicates time-derivative) and the output equation is:

$$\mathbf{y}(t,\boldsymbol{\theta}) = g(\mathbf{x}(t,\boldsymbol{\theta}),\boldsymbol{\theta}) \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state vector, $\tilde{\mathbf{u}} \in \mathbb{R}^r$ the input vector, and $\mathbf{y} \in \mathbb{R}^m$ the output vector. The components of the vector field f and g are (nonlinear and linear) functions which describe, respectively, the structure of the system and the output configuration, parameterized by vector $\boldsymbol{\theta} \in \mathbb{R}^p_{\geq 0}$. $\tilde{\mathbf{u}}(t)$ is an interpolated version of a defined (or measured) input, sampled at N discrete times t_k ($\mathbf{u}(t_k)$). The sampling times can be non-equidistant.

Box 1 Simulating general / nonlinear input-output systems in Matlab.
For example, use Runga-Kutta-Fehlenberg 4/5 order variable step integration method to simulate a step response u = 1 at t = t1:

```
tu=[t1,  1        %time and corresponding input
    tend,1];
[t,x]=ode45(@f,tu(:,1),x0,options, p,tu);
y=g(x);

function dxdt=f(t,x, p,tu)
u=interp1(tu(:,1),tu(:,2),t); %linear interpolation
                              %see help interp1
dxdt(1)=u+…       %enter the ODE's here
```


DOI: 10.13140/2.1.3335.5846

```
    dxdt(2)=…
    dxdt=dxdt(:);     %ode45 requires output to be a column
```

- 'options' defines settings of the simulation algorithm and can be changed using `odeset`; usually default (`options=[]`) is OK.
- All input arguments of ode45 after 'options' are user defined; the function with the ODE's has to accept these as the 3rd (and so forth) inputs. `p` is a vector containing the model parameters.
- In case of a step input, the interpolation is trivial, however also arbitrary user defined or measured signals can be used.
- The output function 'g' has to be defined by the user; often g is just a selection of x, e.g. `y=x(:,2)`
- Note: multiple user-defined Matlab functions can be combined in a single m-file if that m-file itself is also defined as a function.
- If a fixed step integration method is used, the simulation time vector t is known beforehand and an interpolated input signal (vector) can be pre-calculated before the simulation starts. When a variable step solver is used, the interpolated input value has to be calculated in each simulation step, given the new time sample selected by the solver.

Simulation procedure of systems with time-varying input u:

1. Perform an equilibration simulation (i.e. until system is in steady-state).
2. Simulate the perturbation / stimulation of the system. The final values of the state variables of the first step are used as initial values in the second step.

Owing to the presence of unmodeled dynamics, modeling errors and measurement noise, the measured data are assumed to be obtained from a stochastic process. To represent the discrete time measurements, a description of the general properties of the data is introduced:

$$\mathbf{z}_l(t_k) = \mathbf{y}_l(t_k) + \boldsymbol{\varepsilon}_l(t_k) \qquad k = 1,\ldots,N \qquad l = 1,\ldots,m \tag{3}$$

where $\mathbf{z}_l$ are the measurements of the lth output (sampled at N non-equidistant discrete times t_k) and $\boldsymbol{\varepsilon}_l$ is the measurement error, assumed to be additive Zero Mean White Noise with known variance $\sigma_l^2(t_k)$.

Box 2 Variable step integration methods in Matlab

a) As 2nd input argument of the Matlab ODE-solvers a simulation time vector 'tspan' can be defined. However, Matlab does NOT take the user-defined time steps into account during the simulation.

To deliver simulation values at the time points tspan = [t1,t2,…,tend], Matlab applies post-simulation interpolation.

```
[t,x]=ode45(@f,[t1,t2,...,tend],x0,options, p,tu);
```

is equivalent to

```
[t,x]=ode45(@f,[t1,tend],x0,options, p,tu);
x=interp1([t1,t2,...,tend],x,t);
```

b) When a variable-step method is applied, it is especially important to perform an equilibration simulation and the perturbation simulation as two separate simulation runs (see Box 1). When a single simulation would be performed, the variable step method will increase its step size as the system approaches steady-state during the equilibration phase. Therefore, the moment when the input starts to change (the perturbation the user wants to simulate) is likely to be missed.

2 Parameter estimation

The model contains p unknown parameters $\boldsymbol{\theta} = [\theta_1,..., \theta_p]$, which need to be estimated based on experimental time-series data. For identification a time-discrete model output $\mathbf{y}(t_k)$ is generated for the m outputs corresponding to the N time-discrete data ($\mathbf{z}$).

$\mathbf{y}(t_k) = [y_1(t_k),\ldots, y_i(t_k),\ldots, y_m(t_k)]^T$ and
$\mathbf{y} = [y_1(t_1),\ldots,y_1(t_N), \ldots, y_m(t_1),\ldots,y_m(t_N)]^T$ (length $m{\cdot}N$)

A commonly used estimation technique is the Least Squares method. The difference between the measurements aligned in column vector $\mathbf{z}$ and the simulated time-discrete model output aligned in $\mathbf{y}$ (column vector), that is the model error $\mathbf{e}_k$ (also referred to as the residuals), is weighted in a quadratic criterion J_N:

$$\mathbf{e}_k = \mathbf{z}(t_k) - \hat{\mathbf{y}}(t_k, \tilde{\mathbf{u}}, \hat{\boldsymbol{\theta}}) \qquad k = 1,\ldots,N \tag{4}$$

$$J_N(\hat{\boldsymbol{\theta}}) = \sum_{k=1}^{N} \mathbf{e}_k^T \mathbf{W} \mathbf{e}_k \tag{5}$$

where $\hat{\boldsymbol{\theta}}$ is the vector of estimated parameters, $\hat{\mathbf{y}}$ is the model output for the parameter realization $\hat{\boldsymbol{\theta}}$ and $\mathbf{W}$ is a $[m{\cdot}N \times m{\cdot}N]$ positive definite symmetric[1] weighting matrix (the weighted Least Squares algorithm). The parameter estimate is denoted by the hat ^:

[1] Thus $\mathbf{W} = \mathbf{W}^T$ (all eigenvalues real), $\mathbf{x}^T \mathbf{W} \mathbf{x} > 0$ (nonnegative eigenvalues).

$$\hat{\boldsymbol{\theta}} = \arg\min_{\hat{\boldsymbol{\theta}} \geq \mathbf{0}} J_{\mathrm{N}}(\hat{\boldsymbol{\theta}}) \tag{6}$$

The parameters are bounded to $\geq \mathbf{0}$ ($\boldsymbol{\theta} \in \mathbb{R}^{p}_{\geq 0}$). For the optimal estimates the functional *J* reaches a minimum. Due to the inherent model errors (model bias, variance error) a zero value of the identification function cannot be expected.
The covariance matrix of unbiased parameter estimates $\mathrm{cov}(\hat{\boldsymbol{\theta}})$ has the inverse of the Fisher Information Matrix (FIM) $\mathbf{F}_{\theta}$ as lower bound ($\mathrm{cov}(\hat{\boldsymbol{\theta}}) \geq \mathbf{F}_{\theta}^{-1}$, the so-called Cramér-Rao bound [3], [9]). The FIM is based on the weighted sum of squared residuals $\mathbf{e}^{\mathrm{T}}\mathbf{W}\mathbf{e}$ and the Jacobian $\mathbf{J}$ of the cost function with respect to the parameters for $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$ and the number of data points *N*:

$$\mathbf{F}_{\theta} = N\left(\mathbf{e}^{\mathrm{T}}\mathbf{W}\mathbf{e}\right)^{-1}\mathbf{J}\mathbf{J}^{T}\Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \tag{7}$$

This accommodates Gaussian model residuals under maximum likelihood estimation (MLE) and is asymptotically correct for arbitrary distribution of the residuals under weighted least-squares estimation [2], [4]. If the weighting matrix $\mathbf{W}$ used in the least-squares fit is selected as the inverse of the data covariance matrix cov(**z**), then $\mathrm{cov}(\hat{\boldsymbol{\theta}}) = \mathbf{F}_{\theta}^{-1}$ [3], [5]. Now $\hat{\boldsymbol{\theta}}$ is the minimum variance, unbiased estimate and the diagonal elements of the matrix $\mathbf{F}_{\theta}^{-1}$ are approximations of the variance of the estimated parameters ($\hat{\sigma}_{\theta}^{2}$).
Alternatively, the distribution of the parameter estimates could also have been obtained using a Monte Carlo approach, which does not require the model residuals to be Gaussian distributed, but is computationally more costly.

Implementation For parameter estimation the Levenberg-Marquardt algorithm *lsqnonlin* can be used from MATLAB's Optimization Toolbox version 2.2. Parameters can be estimated with lower and/or upper bounds. The termination tolerance for the objective function and the termination tolerance for the parameter estimates can be defined in the 'options'. Convergence to the global minimum of the objective function cannot be guaranteed. An option is to start the algorithm with different initial values for the unknown parameters to verify potential local minimums.
The *lsqnonlin* function also returns the value of the objective function *J* (Eq 5) at the solution $\hat{\boldsymbol{\theta}}$ and the corresponding Jacobian, which can be used to calculate the FIM to obtain estimates of the parameter accuracy.

Box 3 Parameter estimation in differential equation models using Optimization Toolbox in Matlab.
For example, use Levenberg Marquardt method:

```
p0 = […]; %initial values of parameters
lb = zeros(size(p0));     %lower bounds
optim_options = optimset('Display', 'iter',...
    'LevenbergMarquardt','on',...   %default: on
    'TolFun', 1e-4);    %default: 1e-4
[p,resnorm,RESIDUAL,EXITFLAG,OUTPUT,LAMBDA,Jacobian] =
lsqnonlin(@cost, p0, lb,[],optim_options, time,u,data);

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

%Compare fitted model with data
[y,t] = simu(p,time,u);
figure; plot(t,data, '*'); hold on
plot(t,y, 'r','LineWidth',2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function e = cost(p, time,u,data)
%p: parameters
[y,t] = simu(p,time,u);
e = y-data;      %model error

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [y,t] = simu(p,time,u)
…
```

See Box 1 how to simulate a differential equation input-output model.

3 Example: Minimal model for glucose kinetics

Minimal models of glucose and insulin plasma levels are commonly used to analyse the results of glucose tolerance tests in humans and laboratory animals (e.g. Dr. Richard N.

Bergman and co-workers since the 1970's, see references [1], [6], [7]). In a typical frequently-sampled intravenous glucose tolerance test (FSIGTT), blood samples are taken from a fasting subject at regular (but non-equidistant) intervals of time, following a single intravenous injection of glucose. The blood samples are then analyzed for glucose and insulin content. Fig. 1 shows a typical response from a normal subject.

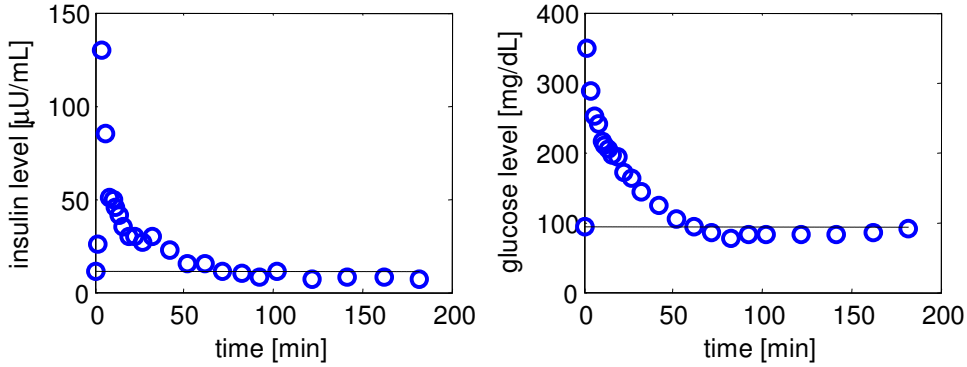

Fig. 1 FSIGT test data from a normal subject ([6]). (See Appendix A for data table.)

Model The glucose and insulin minimal models provide a quantitative and parsimonious description of glucose and insulin concentrations in the blood samples following the glucose injection. The glucose minimal model involves two physiologic compartments: a plasma compartment and an interstitial tissue compartment; the insulin minimal model involves only a single plasma compartment. The glucose and insulin minimal models allow us to characterize the FSIGT test data in terms of two metabolic indices:
S_I = insulin sensitivity: a measure of the dependence of fractional glucose disappearance on plasma insulin,
S_G = glucose effectiveness: a measure of the fractional ability of glucose to lower its own concentration in plasma independent of increased insulin,

The diagram in Fig. 2 summarizes the minimal model for glucose kinetics.

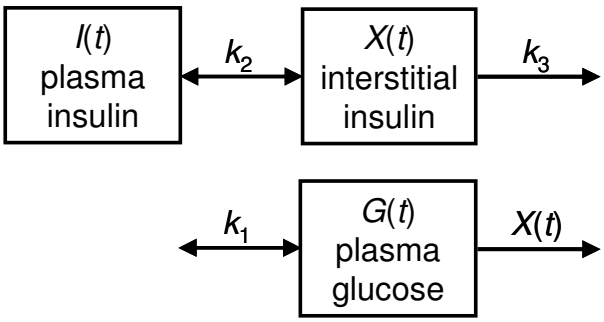

Fig. 2 Minimal model for glucose kinetics.

Glucose leaves or enters the plasma compartment at a rate proportional to the difference between the plasma glucose level, $G(t)$, and the basal plasma level, G_b; if the plasma glucose level falls below the basal level, glucose enters the plasma compartment, and if the glucose level rises above the basal level, glucose leaves the plasma compartment. Glucose also disappears from the plasma compartment via a second pathway at a rate proportional to the 'activity' of insulin in the interstitial tissue $X(t)$.
Insulin leaves or enters the interstitial tissue compartment at a rate proportional to the difference between the plasma insulin level, $I(t)$, and the basal plasma level, I_b; if the plasma insulin level falls below the basal level, insulin leaves the interstitial tissue compartment, and if the plasma insulin level rises above the basal level, insulin enters the interstitial tissue compartment. Insulin also disappears from the interstitial tissue compartment via a second pathway at a rate proportional to the amount of insulin in the interstitial tissue compartment. $I(t)$ is the model input and the course of plasma insulin in time is given by linear interpolation of the time-insulin values listed in Appendix A.
The differential equations corresponding to the glucose minimal model are:

$$\frac{dG(t)}{dt} = k_1\left(G_b - G(t)\right) - X(t)G(t) \qquad G(t_0) = G_0 \qquad (8)$$

$$\frac{dX(t)}{dt} = k_2\left(I(t) - I_b\right) - k_3 X(t) \qquad X(t_0) = 0 \qquad (9)$$

In these equations, t is the independent model variable time [min], t_0 is the time of glucose injection, $G(t)$ is the plasma glucose concentration [mg/dL], $I(t)$ is the plasma insulin level [μU/mL] and $X(t)$ is the interstitial insulin activity. Looking at the structure of Eqn. (8), it is clear that $X(t)$ does *not* represent a physiological, measurable quantity, but a variable with the unit [min^{-1}], mimicking an effective insulin activity. G_b is the basal plasma glucose concentration [mg/dL] and I_b is the basal plasma insulin concentration [μU/mL]. Basal plasma concentrations of glucose and insulin are typically measured

before administration of glucose. There are four unknown parameters in this model: k_1, k_2, k_3, and G_0.
Note that in this model, glucose is utilized with a constant rate constant k_1, when we neglect *feedback* effects due to interstitial insulin as represented by the term $-X(t)G(t)$. An additional amount of plasma insulin will cause the amount of interstitial insulin to change, which in turn, will cause the rate of glucose utilization to change. The *insulin sensitivity* is defined as $S_I = k_2/k_3$ and the *glucose effectiveness* is defined as $S_G = k_1$. Equation 9 can be reformulated as:

$$\frac{dX(t)}{dt} = k_3 \left(S_I \left(I(t) - I_b \right) - X(t) \right) \tag{10}$$

Implementation The built-in Matlab variable step ode-solvers (`ode45`) is used to simulate the glucose profile given the measured time course of plasma insulin as model input signal (data listed in Appendix A). Linear interpolation of the input signal was used to obtain values for each simulation time sample (Matlab function `interp1`). (Note: the application of a measured profile as input is different from the general, independent input usually considered in system theory [9]).
The Matlab commands in Appendix B ("gluc_mm_mle2006.m") estimate values for the parameters given the time course of plasma insulin and glucose. The values of the parameters found minimize (in the weighted least squares sense) the difference between the measured time course of plasma glucose and the parameter-dependent solution to the glucose minimal model differential equations. For numerical minimization the Levenberg-Marquardt algorithm *lsqnonlin* is used from Matlab's Optimization Toolbox version 2.2. The termination tolerance for the objective function was set to 10^{-4}. Parameters were estimated with lower bounds equal to zero and the termination tolerance for the parameter estimates was 10^{-5}. The *lsqnonlin* function also returns the norm (objective function) J at the solution $\hat{\boldsymbol{\theta}}$ and the Jacobian which can be used to calculate the Fisher information matrix to obtain estimates of the parameter accuracy.

Results The simulation results of the identified glucose minimal model are shown in Fig. 3.

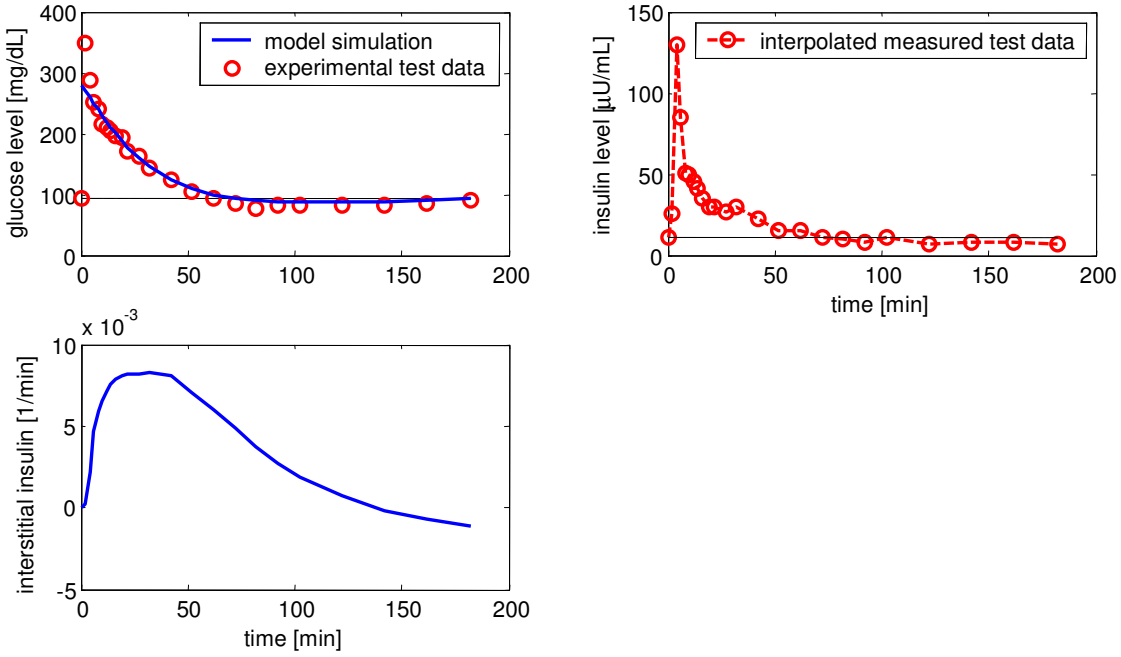

Fig. 3 Simulation results of the glucose minimal model for a normal subject. Solid lines: simulation results, circles: data. G_0 = 279 [mg·dL^{-1}], S_G = 2.6e-2 [min^{-1}], k_3 = 0.025 [min^{-1}] and S_I = 5.0e-4 [mL·µU^{-1}·min^{-1}].

The insulin sensitivity, S_I, for this data set is estimated as 5.039 ×10^{-4} min^{-1}·(µU/ml)$^{-1}$ which is within the normal range reported in [7]: 2.1 to 18.2 ×10^{-4} min^{-1}·(µU/ml)$^{-1}$. The glucose utilization, S_G, for this data set is estimated as 0.0265 min^{-1}, which is also within the normal range reported in [7]: 0.0026 to 0.039 min^{-1}.

4 Summary

This tutorial has shown how Matlab can be used to estimate parameters in non-equidistantly sampled input-output models formulated as a set of nonlinear differential equations. As an illustration, diagnostically important metabolic indices which arise in the glucose minimal models were estimated from intravenous glucose tolerance test data. Another application of parameter estimation in non-equidistantly sampled biological models can be found in [8].

Analysis of model structure and information content of the data can be used to analyze potential identifiability bottlenecks and/or to further improve the accuracy of the estimates. The results of such analysis can be used to design experiments (or modifications of the models) that will provide less ambiguous results.

References

[1] Bergman, R.N., Ider, Y.Z., Bowden, C.R. and Cobelli, C. (1979) Quantitative estimation of insulin sensitivity. *Am. J. Physiol. Endocrinol. Metab.* **236**: E667-677.

[2] Carson, E.R., Cobelli, C., and Finkelstein L. (1983) The mathematical modeling of metabolic and endocrine systems. Wiley, New York.

[3] Damen, A.A.H. (2003) lecture notes, Eindhoven University of Technology, The Netherlands.

[4] Fedorov, V.V. (1972) Theory of optimal experiments. Academic, New York.

[5] Ljung, L. (1999) Chapter 7 'Parameter estimation methods', in: 'System identification - theory for the user', 2nd ed, PTR Prentice Hall, Upper Saddle River, N.J.

[6] Pacini, G. and Bergman, R.N. (1986) MINMOD: a computer program to calculate insulin sensitivity and pancreatic responsivity from the frequently sampled intravenous glucose tolerance test. *Computer Methods and Programs in Biomedicine* **23**: 113-122.

[7] Steil, G.M., Volund, A., Kahn, S.E. and Bergman, R.N. (1993) Reduced sample number for calculation of insulin sensitivity and glucose effectiveness from the minimal model. *Diabetes* **42**: 250-256.

[8] Van Riel, N.A.W. and E.D. Sontag Parameter estimation in models combining signal transduction and metabolic pathways: the dependent input approach. *IEE Syst. Biol*. in press.

[9] Walter, E. and L. Pronzato (1990) Qualitative and quantitative experiment design for phenomenological models – a survey. *Automatica*, **26**(2): 195-213.

Appendix A: Experimental data

Reference [6] provides the FSIGT test data (also shown in Fig. 1) from a normal individual:

time (minutes)	glucose level (mg/dl)	insulin level (μU/ml)
0	92	11
2	350	26
4	287	130
6	251	85
8	240	51
10	216	49
12	211	45
14	205	41
16	196	35
19	192	30
22	172	30
27	163	27
32	142	30
42	124	22
52	105	15
62	92	15
72	84	11
82	77	10
92	82	8
102	81	11
122	82	7
142	82	8
162	85	8
182	90	7

Appendix B: Matlab implementation for parameter estimation

```
function gluc_mm_mle2006
%GLUC_MM_MLE2006 Maximum Likelihood Estimation of minimal model of
glucose kinetics.

%History
%29-Jan-04, Natal van Riel, TU/e
%17-Jun-03, Natal van Riel, TU/e

%REMARK 'if 1/0' CAN BE USED TO (DE)ACTIVATE THE CODE IN BETWEEN THE if-
end STATEMENT

close all; clear all

%DATA
    %Data from Pacini & Bergman (1986) Computer Methods and Programs in
    %Biomed. 23: 113-122.
    %time (minutes)  glucose level (mg/dl)  insulin level (uU/ml)
    tgi = [ 0    92   11
            2    350  26
            4    287  130
            6    251  85
            8    240  51
            10   216  49
            12   211  45
            14   205  41
            16   196  35
            19   192  30
            22   172  30
            27   163  27
            32   142  30
            42   124  22
            52   105  15
            62   92   15
            72   84   11
            82   77   10
            92   82   8
            102  81   11
            122  82   7
            142  82   8
            162  85   8
            182  90   7];
    gluc_exp = tgi(:,2);
    insul_exp = tgi(:,3);

%Fixed initial conditions
x0(2) = 0;  %state variable denoting insulin action; assume fastened
            %state
%Fixed model parameters
Gb = gluc_exp(1);       %[mg/dL] baseline glucose conc. in plasma
Ib = insul_exp(1);      %[uU/mL] baseline insulin conc. in plasma
    x0(1) = 279;%100;    %[mg/dL] glucose conc. in plasma
    Sg = 2.6e-2;    %[1/min] glucose effectiveness
    k3 = 0.025;     %[1/min]
```

```
    Si = 5.0e-4;    %[mL/uU*min] insulin sensitivity
p = [Sg, Gb, k3, Si, Ib];

%Input
%insulin concentration in plasma [uU/mL]; assumed to be known at each
%simulation time sample from linear interpolation of its measured
%samples
    tu = tgi(:,[1,3]);
    t_insu = tu(:,1);
    u = tu(:,2);
    t_gluc = t_insu;

figure; plot(t_insu, u, 'o'); hold on
plot( [t_insu(1) t_insu(end)], [Ib Ib], '--k','Linewidth',1.5)
%baseline level
xlabel('t [min]'); ylabel('[\muU/mL]')
title('measured input signal (insulin conc. time course)')
tspan = [0:1:200];  %to verify interpolation of input signal
h = plot(tspan, interp1(tu(:,1),tu(:,2), tspan), '.r');
legend(h,'interpolated (resampled) signal')

%Simulation time vector:
tspan = t_insu;

disp(' Enter to continue with MLE'); disp(' '); pause

    %4 unknown model parameters:
    p_init = [Sg    %glucose effectiveness
              k3    %
              Si    %insulin sensitivity
              x0(1)];  %G0 initial glucose conc. in plasma
    %3 known parameters:
    p_fix = [Gb Ib x0(2)];

lb = 0*ones(size(p_init));%[0 0 0 0];
ub = [];
options = optimset('Display','iter','TolFun', 1e-4,...%default: 1e-4
    'TolX',1e-5,...                 %default: 1e-4
    'LevenbergMarquardt','on',...   %default: on
    'LargeScale','on');             %default: on
plt = 0;

    %LSQNONLIN: objective function should return the model error
    [p_est,resnorm,RESIDUAL,exitflag,OUTPUT,LAMBDA,Jacobian] =
lsqnonlin(@obj_fn,p_init,lb,ub,options,...
        p_fix,gluc_exp,tspan,tu, plt); disp(' ')

%Accuracy:
%lsqnonlin returns the Jacobian as a sparse matrix
varp = resnorm*inv(Jacobian'*Jacobian)/length(tspan);
stdp = sqrt(diag(varp));    %The standard deviation is the square root
                            %of the variance

%p = [Sg,       Gb,       k3,       Si,       Ib];
p = [p_est(1), p_fix(1), p_est(2), p_est(3), p_fix(2)];
```

```
%x0 = [G0,      X0]
x0 = [p_est(4), p_fix(3)];
disp(' Parameters:')
disp([' Sg = ', num2str(p_est(1)), ' +/- ', num2str(stdp(1))])
disp([' Gb = ', num2str(p_fix(1))])
disp([' k3 = ', num2str(p_est(2)), ' +/- ', num2str(stdp(2))])
disp([' Si = ', num2str(p_est(3)), ' +/- ', num2str(stdp(3))])
disp([' Ib = ', num2str(p_fix(2))])
disp(' Initial conditions:')
disp([' G0 = ', num2str(p_est(4)), ' +/- ', num2str(stdp(4))])
disp([' X0 = ', num2str(p_fix(3))]); disp(' ')

plt = 1;
gluc = gluc_sim(tspan,x0,tu, p,plt);

%compare model output with measured data
figure(2); subplot(221)
h1 = plot(t_gluc,gluc_exp,'or', 'Linewidth',2);
%legend(h1, 'experimental test data')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function e = obj_fn(p_var, p_fix,data,tspan,tu, plt)

%4 unknown model parameters:
% p_var = [Sg k3 Si G0];
%3 known parameters:
% p_fix = [Gb Ib x0(2)];
%p = [Sg,      Gb,       k3,       Si,       Ib];
p = [p_var(1), p_fix(1), p_var(2), p_var(3), p_fix(2)];
%x0 = [G0,      X0]
x0 = [p_var(4), p_fix(3)];
gluc = gluc_sim(tspan,x0,tu, p, 0);

%LSQNONLIN: objective function should return the model error
e = gluc-data;
if plt==1   %fast update during estimation process
    N=length(tspan);
    figure(2)
    plot(1:N,gluc,'b',1:N,data,'r'); drawnow
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function gluc = gluc_sim(tspan,x0,tu, p,plt)
%SIM_INPUT_SIM Simulation of glucose minimal model.
%x0: initial conditions for glucose conc. and state variable denoting
%    insulin action
%gluc: model output, glucose conc. in plasma [mg/dL]

if tspan(1) < tu(1,1)
    error(' no input defined at start of simulation (compare tspan(1))
vs. tu(1,1)')
end
```

```
    ode_options = [];
    [t,x] = ode45(@gluc_ode,tspan,x0,ode_options, tu, p);
%Output
    gluc = x(:,1);

if plt==1
    gluc_plt(tspan,x,tu,p)
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function dxout = gluc_ode(t,xin,tu,p)
%GLUC_ODE ODE's of glucose minimal model
% xin: state values at previous time sample x(k-1) = [G(k-1); X(k-1)]
% dxout: new state derivatives dx(k) = [dG(k); dX(k)], column vector
% tu: u(k) (input signal at sample k) OR matrix tu
% p = [Sg,      Gb,       k3,       Si,       Ib];

%History
%17-Jun-03, Natal van Riel, TU/e

idG = 1; idX = 2;   %glucose conc. [mg/mL] and state variable denoting
insulin action [uU/mL]
Sg = p(1);  %[1/min] glucose effectiveness
Gb = p(2);  %[mg/mL] baseline glucose conc.
k3 = p(3);  %[1/min]
Si = p(4);  %[mL/uU*min] insulin sensitivity
Ib = p(5);  %[uU/mL] baseline insulin conc. in plasma

if length(tu)==1    %u(k) is provided as input to this function
    u = tu;
else    %calculate u(k) by interpolation of tu
    u = interp1(tu(:,1),tu(:,2), t);
end

%ode's
dG = Sg*(Gb - xin(idG)) - xin(idX)*xin(idG);
dX = k3*( Si*(u-Ib) - xin(idX) );

dxout = [dG; dX];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function gluc_plt(tspan,x,tu,p)

Gb = p(2);
Ib = p(5);

figure
subplot(221); h = plot(tspan,x(:,1), '-','Linewidth',2); hold on
plot( [tspan(1) tspan(end)], [Gb Gb], '--k','Linewidth',1.5)
%baseline level
ylabel('glucose level [mg/dL]')
```

```
legend(h, 'model simulation')
u = interp1(tu(:,1),tu(:,2), tspan);    %reconstruct used input signal
subplot(222); plot(tspan,u, '--or','Linewidth',2); hold on
plot( [tspan(1) tspan(end)], [Ib Ib], '--k','Linewidth',1.5)
%baseline level
ylabel('insulin level [\muU/mL]'); xlabel('time [min]')
legend('interpolated measured test data')

%figure
subplot(223); plot(tspan, x(:,2), 'Linewidth',2)
xlabel('time [min]'); ylabel('interstitial insulin [1/min]')
```