Modelling methodology in Process Engineering

Laboratory of Applied Mathematics
Lappeenranta University of Technology
heikki.haario@lut.fi
tuomo.kauranne@lut.fi

Prerequisites: Basic mathematics courses (A & B).

Software used: Matlab

Background books:

Ullman's Chemical Engineering and Plant Design, Vol 1, Wiley 2005,

J. Ingham, I.J. Dunn, E. Heinzle, J.E. Prenosil: Chemical Engineering Dynamics, Wiley 1994, 2nd Edition.

G.Box, J.S.Hunter, W.G.Hunter: Statistics for Experimenters, 2nd Edition, 2005.

1 Introduction

Contents of the course

- Empirical models
  - Visualization of data
  - Calibration of models: Least squares
    - Curve fitting
    - Regression models
    - Design of experiments
- Mechanistic modelling
  - Material balances: kinetics, mass transfer, reactors
  - Parameter estimation
- Laboratory work with modelling

The concept of a 'Model' here: a computational approximation of a real–life process. Where do you need models ?

- Chemical kinetics
- Mass transfer
- Reactor design
- Planning of experiments

Models are useful for

- Process optimization
- Experimental work: maximize information, minimize costs
- Understanding the phenomena
- Training and education

Modelling is an increasing trend in engineering, due to increasing computer power, evolving software and scientific knowledge.

Models in Research

A model:

$$X \longrightarrow Y1 \longrightarrow Y2, \text{ etc}$$

Input response response

Input, the 'independent' variables, for example:

- the controlled variables in a laboratory experiment
- the controlled process conditions in a factory
- variables measured from a process

Response, the dependent variables: the result of an experiment, the quality of a product, etc.

A model gives a way to compute the response, when the input variable values are given.

The model predictions always are uncertain, due to

- Incomplete understanding of the process
- Uncertainties in computational implementation.
- Errors, 'noise', in experimental data

Typically models contain unknown parameters, that have to be calibrated ('fitted') by data: models depend on measured data.

The task of statistical analysis is to quantify the uncertainty in model predictions, due to uncertainties in data.

Motifs of modelling

The motifs may be various:

- *Understanding* the key characteristics of a process
  - Example. The deterioration of a lake water quality, due to N or P?
- *Optimization* of an industrial product cost–effectively
  - Example. Preparation of food: the best time and temperature for a beef in an oven ?
- *Simulation* of situations, where we have no experimental information yet. Examples
  - Planning of new industrial processes: scale-up
  - Design of new reactors, instruments.

Types of models

- 'soft', statistical models. 'Black box' calibration of existing data:
  - + Easy computations, problems in interpretation
  - + Plenty of software and literature
  - - Limitations: not for predictions of essentially new situations (scale-up, new processes, etc).
  - Methodology employed: *regression analysis*.
- 'hard', mathematical models based on physics and chemistry:
  - - Require knowledge of numerical methods, modelling often more demanding.
  - + Enable accurate prediction of new phenomena (new process conditions, scale-up, etc)
  - Methodology employed: *Simulation and optimization* algorithms.

Types of data

- Active: done by the researcher, e.g., a designed set of experiments in a laboratory
  - Methodology employed: *design of experiments* for creation of data, *regression analysis* for analysis of data.
- Passive: collected data., e.g., from the environment or an industrial process or factory.
  - Methodology employed: various *data–analysis* methods, e.g., methods of *Chemometrics*

Sources of experimental errors

Accuracy of an instrument: the 'noise' level of measurements, gives a *lower bound* for the total uncertainty.

term: Repeatability

Uncertainties in performing an experiment: heterogeneity of raw materials, impurities in equipment, differences between persons in charge, etc.*Often the largest errors are caused by these*

term: Reproducibility

Experiments not carried out in sufficient density or coverage: results only cover some subset of relevant phenomena.

term: Relevance

An ' R & R & R ' study aims at estimating the magnitudes of the various error sources.

Knowledge of the experimental errors is the basis for estimating the Reliability of modelling results !

Stages of model building

Input, the experimental variables		Response
$x_1, x_2, \ldots x_m$	$\rightarrow$	$y_1, \ldots, y_n$

- *Which* variables should be selected to explain the response ?

  - Screening of variables

- *How* do the (selected) variables explain the response ?

  - Empirical/Statistical modelling

  Example:

$$y = b_0 + b_1 x_1 + b_{12} x_1 x_2 + b_{22} x_2^2 + \epsilon$$

- *Why* (by which mechanism) do the variables explain the response

  - Mechanistic, Physicochemical modelling

Choice of modelling approach

The choice of the model is dictated by the aim of the project. A model should be 'as simple as possible - but not more simple than that' (A.Einstein).

An empirical model is chosen, if

- a 'local' description (calibration of measurements) of the process is enough
- the mechanism of the process is not known

A mechanistic model is chosen, if

- the mechanism is understood well enough - or the aim is to understand it
- the quality of the data is good enough for identification and validation of the model
- the model is needed outside the available experimental region.

2 Empirical modelling

Contents

- Empirical models
- Visualization of data
- Calibration of models: Least squares
  - Fitting of a straight line
  - Curve fitting
  - Regression models: several independent variables
  - Design of experiments

Empirical Models

We consider here situations where the model that describes the dependency between X and Y,

$$X \longrightarrow Y$$

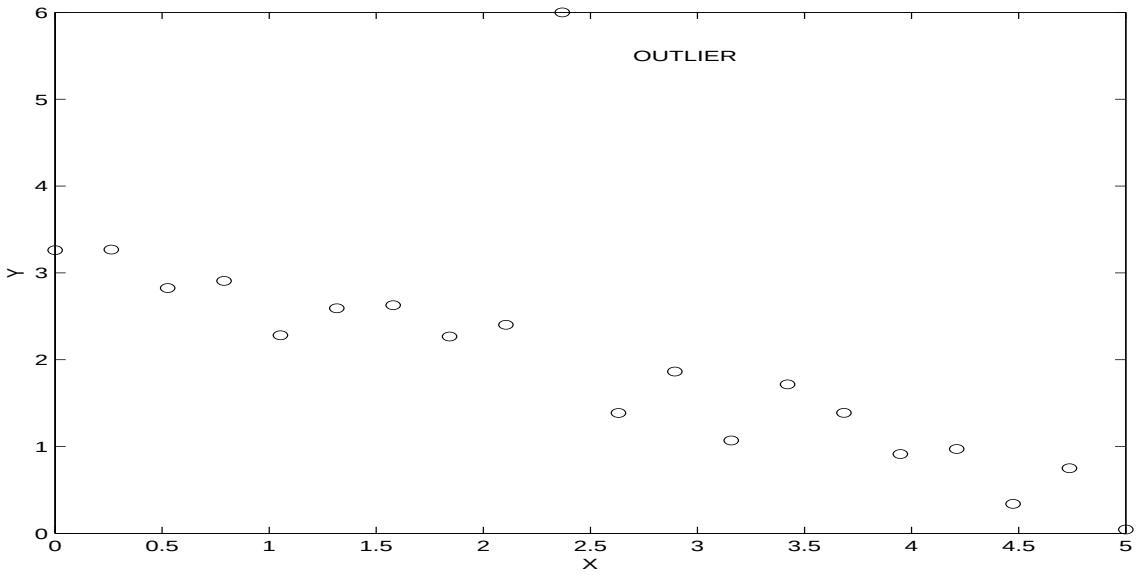
Input response

is constructed by *empirical data* only. If the input values consist of p variables $x_1, ..., x_p$ and we have n experimental measurements done, the data values may be expressed as a *design matrix*

$$X = \begin{array}{c} \begin{matrix} x_1 & x_2 & \dots & x_p \end{matrix} \\ \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{pmatrix} \end{array},$$

Visualization of data

All data contains measurement noise. But we should avoid using data that contains obvious 'big errors', *outliers* due to typing errors, malfunctioning instruments, etc. Before any modelling, it is advisable to check the data by visual plots.



2.1 Least Squares Fitting

Suppose we want to *fit* a line $y = b_0 + b_1 x$ to data that was measured at points $x_i, y_i, i = 1, 2, ..., n$. We must find values for the model parameters b_0, b_1 so that the values computed by the model and the measured values 'agree' as closely as possible. The most common way of doing this is to construct the *least squares*, LSQ, function

$$\ell(b) = \sum_{i=1}^{n} (y_i - (b_0 + b_1 x_i))^2$$

and to find values b_0, b_1 that minimize this expression.

More generally, a model often is written in the form

$$y = f(x, b) + \epsilon$$

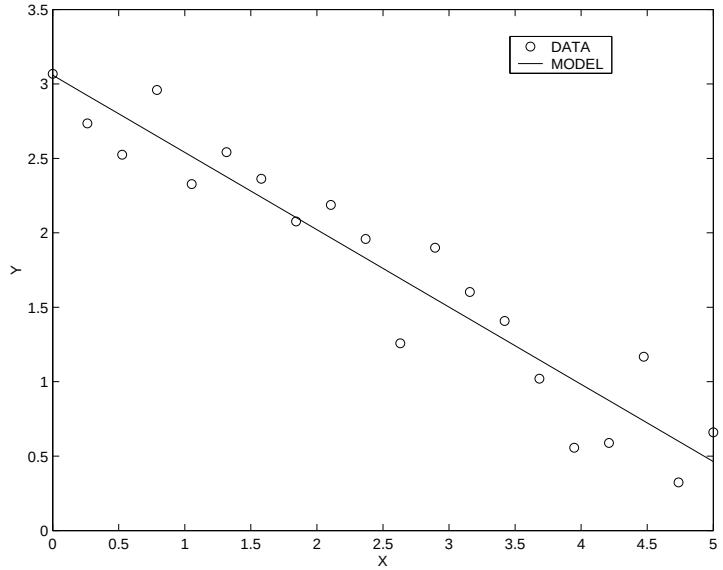
where x and y are input and response, b denotes the vector of unknown parameters, and ϵ represents the measurement noise. The LSQ function then assumes the form

$$\ell(b) = \sum_{i=1}^{n} (y_i - f(x_i, b))^2$$

and again we have to find the parameter values that minimize the sum. The minimization is generally performed by numerical optimization algorithms. In some cases, typically with empirical models, we can derive formulas that directly calculate the LSQ fit.

Fitting of a straight line

Let us return to the case, where the model is given by a straight line $f(x, b) = b_0 + b_1 x$.



The coefficients for minimizing the LSQ can now be explicitly calculated. Let us yet consider separately the special cases where one of the coefficients vanishes.

If $f(x, b) = b_0$, the LSQ function reads as

$$\ell(b) = \sum_{i=1}^{n} (y_i - b_0)^2$$

The necessary condition for a minimum is that the derivative with respect to b_0 vanishes. We get the condition

$$\frac{d}{db}\ell(b) = \sum_{i=1}^{n} 2(y_i - b_0)(-1) = -2\sum_{i=1}^{n} y_i + 2\sum_{i=1}^{n} b_0 = 0$$

from which b_0 assumes the formula $b_0 = \frac{\sum_{i=1}^{n} y_i}{n}$. We see that if we fit a constant value to the data, *the best LSQ fit is the mean value of the data*.

Goodness of fit, the R2 value

The result is not just a math exercise, but gives a valuable tool for assessing the *goodness of fit* of a model to data. The 'model' $f(x, b) = constant$ is the simplest possible, and lacks any explanatory power. But so we can use it as a comparison case to see how well our 'real' model works: generally, it should fit to the data clearly better than the trivial, constant model. The comparison is commonly written in the form of the R2 value:

$$R2 = 1 - \frac{\sum (y^i - f(x^i, \hat{b}))^2}{\sum (y^i - \bar{y})^2}.$$

Here $\bar{y}$ denotes the mean value of y, the fit by a constant. If our model fits the data better than the mean value, R2 should b close to the value 1. If our model is as bad as the mean value, R2 is close to the value 0. As a rule of thumb, a R2 value between 0.7 and 0.9 is often regarded good enough for an empirical model.

Next, consider a straight line that goes through the origin , $f(x, b) = b_1 x$. Now the LSQ function reads as

$$\ell(b) = \sum_{i=1}^{n} (y_i - b_1 x_i)^2$$

Again, we compute the derivative with respect to the parameter, now b_1, and get

$$\frac{d}{db}\ell(b) = \sum_{i=1}^{n} 2(y_i - b_1 x_i)(-x_i) = -2\sum_{i=1}^{n} y_i x_i + 2b_1 \sum_{i=1}^{n} x_i^2 = 0$$

from which we get for $b_1 = \sum_{i=1}^{n} x_i y_i / \sum_{i=1}^{n} x_i^2$.

Using the vector-matrix notations, the result may be written in a compact form. Recall the definitions of a *transpose* of a vector and the *inner product* of to column vectors $x = (x_1, x_2, ..., x_n)$, $y = (y_1, y_2, ..., y_n)$:

$$x'y = \sum_{i=1}^{n} x_i y_i$$

(a special case of a matrix-matrix product). If $y = x$, we have

$$x'x = \sum_{i=1}^{n} x_i x_i$$

So the previous result for b_1 can be written as

$$b_1 = \frac{x'y}{(x'x)} = (x'x)^{-1}x'y$$

.

Finally, let us consider the full equation of a line, $f(x, b) = b_0 + b_1 x$, and suppose we have measured the response values $y = (y_1, y_2, ..., y_n)$ at the points $x = (x_1, x_2, ..., x_n)$. Using again the vector-matrix notations, we may write the equations $y_i = b_0 + b_1 x_i$, $i = 1, 2, ...n$ as one matrix equation

$$y = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \\ 1 & x_n \end{pmatrix} \times \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = Xb$$

By analogy to the previous case (we skip a detailed derivation here) we may write the LSQ solution as $b = (X'X)^{-1}X'y$. We shall later see that this formula is valid for models with several input variables x, too.

Fitting of a polynomial

If the data clearly is nonlinear – it is curved, exhibits a maximum or minimum, etc – we must drop the straight line and seek for other functions to be fitted. A first extension might be a polynomial of 2. degree, a parabola,

$$y = b_0 + b_1 x + b_2 x^2$$

We note that this model is nonlinear only with respect to the x variable, but *linear* with respect to the unknown coefficients $b_i, i = 0, 1, 2$. Using again the matrix–vector notations, the above expression may be written as

$$y = (1, x, x^2) \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}$$

Suppose then that we have again measured some response values $y = (y_1, y_2, ..., y_n)$ at the points $x = (x_1, x_2, ..., x_n)$. Using again the vector-matrix notations, we may write the equations $y_i = b_0 + b_1 x_i + b_2 x_i^2$, $i = 1, 2, ...n$ as one matrix equation

$$y = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_n & x_n^2 \end{pmatrix} \times \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = Xb$$

In this form, the equation is just the same as before, and the same solution formula applies. So we may write the LSQ solution as $b = (X'X)^{-1}X'y$. Using Matlab, we may also use the shortcut notation $b = X \backslash y$

Example

```
x=0:5;    x = x';
b = [2 -.02 2 ]';
X = [ones(6,1) x x.^2];
y = X*b;
plot(x,y)

 y_noisy = y + randn(6,1)*1.5;
 bn      = X\y_noisy
 y_fit   = X*bn;

 plot(x,y_noisy,'o',x,y_fit,'r-')
```

Example of the effect of an outlier in linear regression

```
 x=0:10;    x = x';
 X = [ones(11,1) x];
 y = 2*x + 3; % the "true" solution
 plot(x,y)
% adding noise to observations
  y_noisy = y + randn(11,1)*2;
  bn      = X\y_noisy
  y_fit   = X*bn;
  plot(x,y_noisy,'o',x,y_fit,'r-')
% the effect of an outlier
  y_noisy(5) = 100;
  bn      = X\y_noisy
  y_fit   = X*bn;
  plot(x,y_noisy,'o',x,y_fit,'r-')
```

Higher Polynomials

A polynomial of 3. degree, $y = b_0 + b_1 x + b_2 x^2 + b_3 x^3$, may equally well be written as

$$y = (1, x, x^2, x^3) \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

So, to fit a polynomial of 3. degree, we just have to add yet another column, containing the x^3 values, in the matrix X. Again, we arrive at the system $y = Xb$, and solve it as previously.

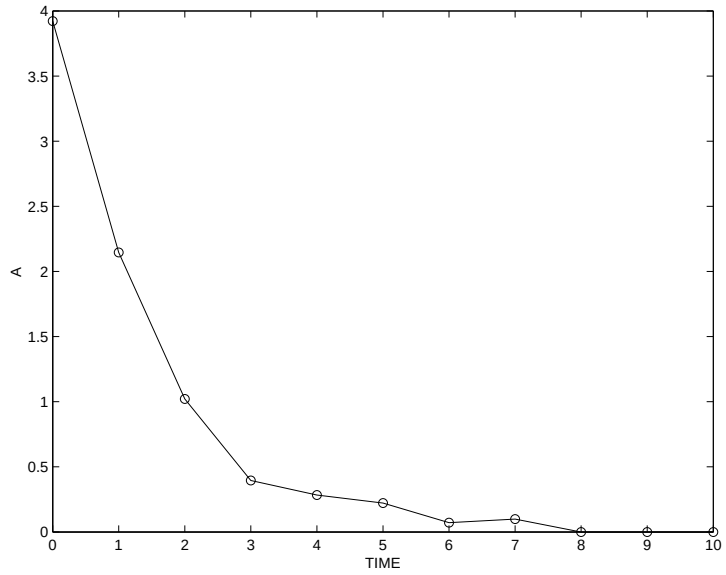
Warning: high order polynomials may behave curiously, see the example 'census' in Matlab!

2.2 Curve Fitting

Example. A chemical component A reacts following an exponential decay law,

$$A(t) = A_0 e^{-k*t}$$

for time $t \geq 0$.



A polynomial no more is a good choice for a model, if we know more in detail how the data should behave: it goes to zero and stays there - but a polynomial never does so. To find the best parameters $A0, k$ that fit the data, we use methods of *Curve Fitting*.

Additional tasks

- write the function to be optimized
- use some optimization algorithm,
- give an *initial guess* for the optimizer

For these purposes we may use, e.g., the function 'lsqcurvefit' in Matlab.
Below an example:

```
%create data first:
 t=0:1:10;                              % xdata
 y=4*exp(-0.75*t);                      % the 'true' profile
 y_noisy = y+randn(1,11)*0.25; % ydata, the 'experimental'
                                        % noisy data
%do the fit:
 initguess = [1 1];                     % the initial guess
                                        % for the optimizer
 x_lsq= lsqcurvefit(@myfun1,initguess,t,y_noisy);% do the fit
 y_fit = myfun1(x_lsq,t);               % compute the solution
                                        % with the optimized parameter
 plot(t,y_noisy,'o',  t,y_fit);% plot both data and fit
```

The function we fit is given in a separate 'function' file with the name, for instance, 'myfun1.m':

```
  function F = myfun1(teta,t)
% INPUT parameter list:
%  teta   the parameters to be estimated
%  t      'xdata', here the times of observation

%  y = A0*exp(-k*t)  this is just a comment row
     F = teta(1)*exp(-teta(2)*t);
```

2.3 Regression models

So far we have considered cases where the response only depends on one experimental factor, the time for instance. But it often depends on several factors – and, in advance, we do not even know which ones. *Regression analysis* provides methods for studying responses with several independent explanatory factors.

Example. We want to optimize the quality y of food prepared in an oven. y depends on two factors, x_1, the temperature of the oven , and x_2, the time how long we keep it in the oven. There clearly is an optimum: too small or large values of x_1, x_2 would spoil the food. How should we find the optimum, with minimal costs?

The question of minimizing the costs – the number of experiments performed – leads us to the topic of *design of experiments*. We return to this question later. Let us first suppose that we have, somehow, decided to make n experiments at the points collected in the design matrix

$$X = \begin{array}{c} \begin{array}{cc} x_1 & x_2 \end{array} \\ \begin{pmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ \vdots & \\ x_{1n} & x_{2n} \end{pmatrix} \end{array}$$

and have measured the response values $y = (y_1, y_2, ..., y_n)$ in the experiments. We may now try to fit various models that depend on the x variables to the data.

A *linear model* of two variables has the form

$$y = b_0 + b_1 x_1 + b_2 x_2.$$

Using the vector-matrix notations, the equations $y_i = b_0 + b_1 x_{1i} + b_2 x_{2i}$, $i = 1, 2, ..., n$ may be written as a single equation

$$y = \begin{pmatrix} 1 & x_{11} & x_{21} \\ 1 & x_{12} & x_{22} \\ \vdots & & \\ 1 & x_{1n} & x_{2n} \end{pmatrix} \times \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = \tilde{X} b$$

Formally, this is the same matrix system we have met before, and we may write the LSQ solution as $\hat{b} = (\tilde{X}'\tilde{X})^{-1}\tilde{X}'y$ (in Matlab, use the shortcut notation $b = \tilde{X}/y$)

The terms of a quadratic model

But, in our food example, we know that a linear model can not explain the data: the optimum inside the experimental region requires some quadratic terms in the model. We may separately consider the parts

The linear part, *the main effects*

$$b_0 + b_1 x_1 + b_2 x_2$$

The products, *the interaction terms*

$$b_{12} x_1 x_2$$

The 2. powers, *the quadratic terms*

$$b_{11} x_1^2 + b_{22} x_2^2$$

A full quadratic model contains all the above parts,

$$y = b_0 + b_1 x_1 + b_2 x_2 + b_{11} x_1^2 + b_{12} x_1 x_2 + b_{22} x_2^2.$$

If measurement data is available at the points $x_i, y_i,\ i = 1, 2, ..., n$, the equations $y_i = b_0 + b_1 x_{1i} + b_2 x_{2i} + b_{11} x_{1i}^2 + b_{12} x_{1i} x_{2i} + b_{22} x_{2i}^2$ may again be written as a single equation, just by adding the respective columns for $x_1^2, x_1 x_2$ and x_2^2 in the design matrix X. The LSQ solution for the model coefficients b is then obtain just as before.

So, technically, we know how to calculate LSQ solutions for regression linear or quadratic models with two (or more) independent variables. But the *statistical analysis* remains: how good is the model, which model should be selected?

Statistics for regression

The *Residual* of a fit is the difference between data and the model values, for regression models given as:

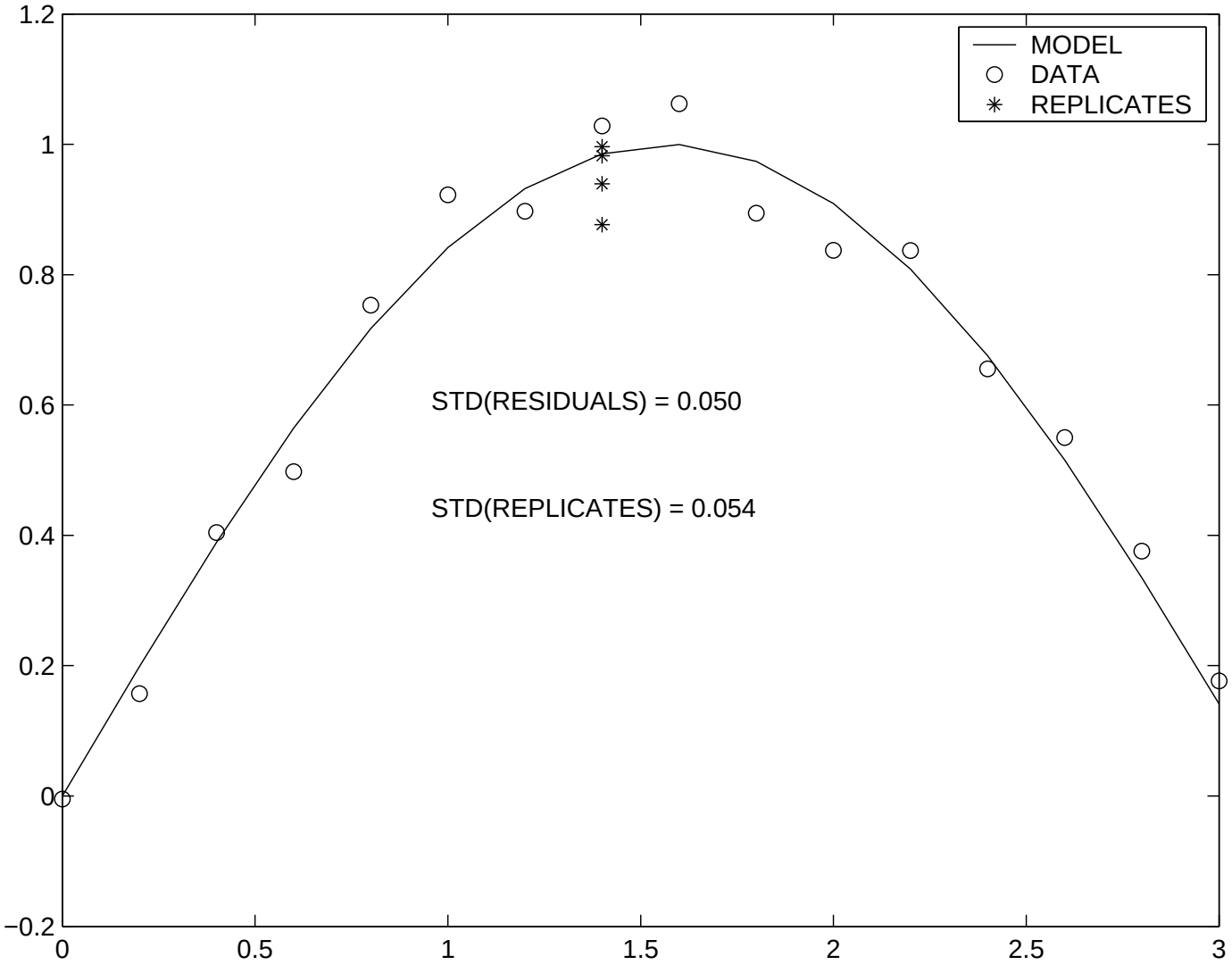
$$res = y - X\hat{b}$$

These values should be compared to the size of the *experimental error*, the noise in the measurements. In the simple (and most typical) case we may suppose that the noise level is the same in all experiments. The noise should be estimated by *replicated measurements*: measurements repeated several times in the same experimental condition. The size of the noise is expressed as the standard deviation (std) σ of the repeated measurement values.

The basic statistical requirement is that a model should fit the data *with same accuracy as the measurements are obtained*, the residual of the fit should roughly equal the estimated size of the measurements noise:

$$\text{std}\, res \simeq \text{std}\, noise$$

Example


MODEL
DATA
REPLICATES
STD(RESIDUALS) = 0.050
STD(REPLICATES) = 0.054
1.2
1
0.8
0.6
0.4
0.2
0
−0.2
0
0.5
1
1.5
2
2.5
3

t–values

Recall the LSQ estimator for the coefficients of the model, $\hat{b} = (X'X)^{-1}X'y$. Since the data y contains noise, the values of $\hat{b}$ are noisy, too.

The std of the coefficients may be calculated by the so called *covariance* matrix $\mathrm{cov}(\hat{b}) = (X'X)^{-1}\sigma^2$. The diagonal of $\mathrm{cov}(\hat{b})$ gives the variances of $\hat{b}$, so $\mathrm{std}(\hat{b}_i)$ for each coefficient is obtained as the square root of the diagonal values.

The 'signal to noise ratio', the calculated value of the coefficient divided by the std of it, is called the *t–value* of the coefficient:

$$t_i = \hat{b}_i / \mathrm{std}(\hat{b}_i)$$

.

The t–values may be used to select the terms of a model. The idea is that if a term is not known reliably enough, it should be dropped away from the model. This is the situation if the uncertainty, $\mathrm{std}(\hat{b}_i)$ is large if compared to the calculated value of the coefficient, $\hat{b}_i$. As a rule of thumb, terms with

$$\|t_i\| = \hat{b}_i / \mathrm{std}(\hat{b})_i < 3$$

should be abandoned.

```
X    = [ones(size(x,1),1), x];   % add the '1' column
bhat = X\y; yhat = X*bhat;       % the LSQ solution
res  = y-yhat;                   % the residuala
rss  = sum(res.^2);              % the residual sum of squares
s2   = rss/(n-p);                % the variance of noise in y
cb   = inv(X'*X)*s2;             % covariance of b
sdb  = sqrt(diag(cb));           % the standard deviations of
tb   = bhat./sdb;                % the t-values
tss  = sum((y-mean(y)).^2);      % the total sum of squares
R2   = 1-rss/tss;                % R^2 value
```

2.4 Creation of data: Design of Experiments

Choice of the experimental region

Concepts:

- *Operational region*: the conditions in which the experiments reasonably may be carried out
- *Experimental plan*: the set of experiments performed, typically covers a subdomain of the operational region

Risks in choosing the region (min /max values) for the experiments:

- Too small → all experiments (almost) replications of one situation, the effects of factors confounded by experimental noise.
- Too big → the effects of factors not covered by an (linear or quadratic) regression model

A rule of thumb: the effects due to different values of the factors should be 2 – 3 times larger than the size of experimental noise.

The selection of min/max values of the factors may be problematic. The experimenter needs to know a 'reasonable' region for experiments, here the statistical methods (alone) do not help!

Suppose we have p factors, $x = (x_1, ..., x_p)^T$, whose effect on a response variable is studied. The values for a set of experiments are given as a table, **Experimental Design**.

$$X = \begin{array}{c} \begin{matrix} x_1 & x_2 & \dots & x_p \end{matrix} \\ \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{pmatrix} \end{array},$$

Experimental plans for various purposes

Screening: From a large number of possibly important factors , we want to find out the most significant ones. Properties of plans to be used:

+ A minimal number of experiments.

- Preliminary experiments, a 'final' model not yet found

Experiments for a regression model.

+ a model reliably constructed, enables an analysis of the effects of the factors and, e.g., an optimization of the response values.

- a small number of factors (max 4, 5)

Coded Units

Design plans are typically expressed in *coded units*: the center point of the experimental plan is moved to the origin, the min/max points to the values ± 1. If $\bar{x}_i$ denotes the mean value and Δ_i difference between max and min values of the factor x_i, the transformation is given by

$$X_i = \frac{x_i - \bar{x}_i}{\Delta x_i / 2}$$

The coded units (X) give a generic way to present various plans. The values in the real 'laboratory' units (x) are obtained from coded units by solving the above equations.

The most important design plans

Different designs allow the determination of different models. So the design of experiments should be selected according to the expected behavior of the response:

- 2^N **design**. Enables the estimation of a *linear model* (plus interaction terms, see below)
- **CCD, Central Composite Design** Allows the estimation of a full quadratic model (or *response surface*)

2^N designs

- Each factor gets values at two levels (± 1)
- The design contains all combinations of the levels.
- 2^N plans allows the estimation of *the main effects and the interaction terms*.

Example 2^N experiment in coded units, when $N = 2$

$$X = \begin{array}{c} \begin{matrix} x_1 & x_2 \end{matrix} \\ \begin{pmatrix} +1 & -1 \\ +1 & +1 \\ -1 & -1 \\ -1 & +1 \end{pmatrix} \end{array}$$

Replicated measurements

By the above design, it is possible to a linear model together with the interaction term, $y = b_0 + b_1 x_1 + b_2 x_2 + b_{12} x_1 x_2$. But we should have more experiments than unknowns (the coefficients b). As a rule of thumb, the number of experiments should be roughly twice the number of unknowns.

It is always advisable to perform several (e.g. 4–5) measurements that are repeated at the same point. From replicates we get an estimate of the *noise level* of the measurement. Replicates may be done in many ways, e.g., by performing all experiments twice.

Most commonly, the replicated measurements are carried out in the center point. Then the noise level is easily computed by the std (standard deviation) of the replicates

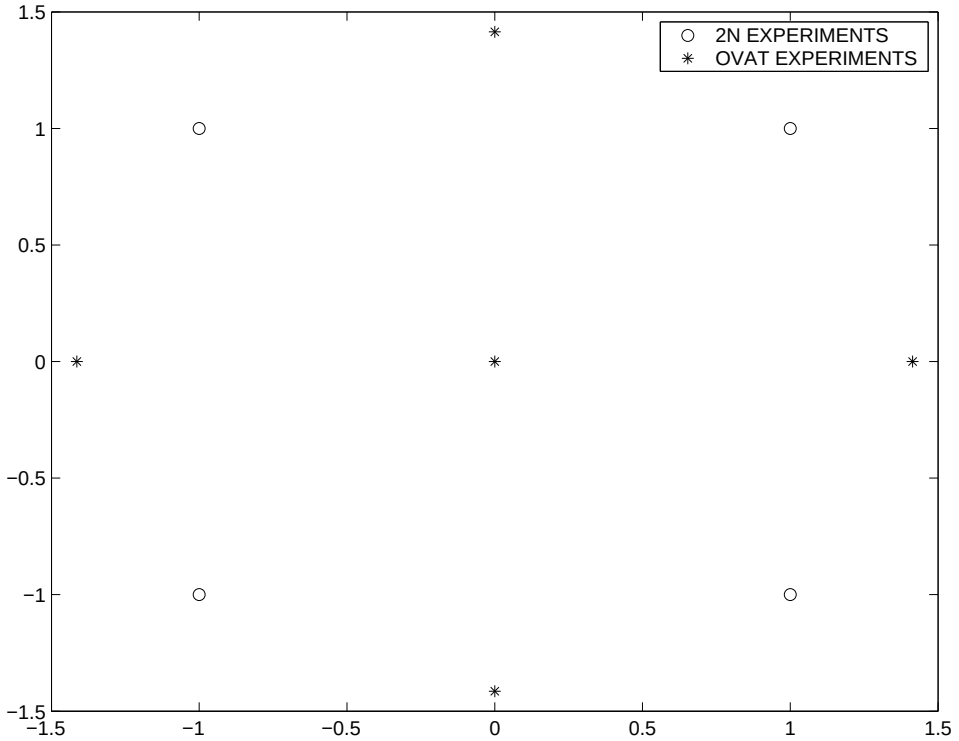
Example 2^N experiment with 4 repeated measurements in the center, in coded units, when $N = 2$

$$X = \begin{array}{c} \begin{array}{cc} x_1 & x_2 \end{array} \\ \begin{pmatrix} +1 & -1 \\ +1 & +1 \\ -1 & -1 \\ -1 & +1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \end{array}$$

Central Composite Design (CCD) plans

A 2^N plan may be extended by doing experiments 'One Variable at a Time (OVAT) ': change only the values of one variable, keeping the rest constant at the center point.

Example with N=2



Example CCD plan with 2 center point replicates, in coded units, $N = 2$

$$X = \begin{pmatrix} +1 & -1 \\ +1 & +1 \\ -1 & -1 \\ -1 & +1 \\ \sqrt{2} & 0 \\ -\sqrt{2} & 0 \\ 0 & \sqrt{2} \\ 0 & -\sqrt{2} \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

The plans enables the construction of a full quadratic model
$y = b_0 + b_1 x_1 + b_2 x_2 + b_{11} x_1^2 + b_{12} x_1 x_2 + b_{22} x_2^2$. with the *the main effects, the interaction terms* and *the quadratic terms*

Remarks

- It is common to do *only* the 'One Variable at a Time' experiments. This is *not* recommended: the interaction between factors will not be seen.
- Experiments may be done in two stages: First a 2^N design with center point replicates. If the center point values give an indication of quadratic behavior (the values are larger/smaller than at the corner points), the design may be extended by OVAT experiments to the full Central Composit Design

Experimental Optimization

Purpose: optimize the quality of a product, by doing experiments and analyzing the results by regression methods.

The procedure is sequential:

- **1** Select an experimental plan:
  - A 2^N plan when far from optimum
  - A CCD plan when near to the optimum
- Perform the experiments, collect the results in data matrices
- Create the regression model
- Do (a few) experiments towards the optimum, as guided by the regression model ('response surface')
- When results deteriorate, go back to step **1**

3 Mathematical Modelling

Example A chemical reaction (or radioactive decay): $A \rightarrow B \rightarrow C$. Assume that the reaction rates are proportional to the amounts of the components A, B. They may then be written as the expressions $k_1 A, k_2 B$ where k_1, k_2 are the *reaction rate constants*. Modelling by material balances leads us to a system of 'ordinary differential equation' (ODE):

$$\begin{aligned} \frac{dA}{dt} &= -k_1 A \\ \frac{dB}{dt} &= k_1 A - k_2 B \\ \frac{dC}{dt} &= k_2 B \end{aligned}$$

Note that the mass balance always is satisfied: $d/dt(A + B + C) = 0$, $A + B + C =$ constant.

In this example, the solution may be obtained integrating 'by hand'. However, typically a solution only is available by numerical methods. Using MATLAB, this requires the steps

- write a m–file (a *script file*) that gives all the necessary *initial information* and calls an *ODE solver*.
- write a m-file (a *function file*) that contains the model equations.

Note that the ODE solver may remain a 'black box' for the user - it is usually enough to know just which solver to use.

The solver call, as well as the model function file, must be written is a specific way, as given in the example (and MATLAB's *help files*). The files may be named,e.g., as 'myfirst.m' and 'myfirstode.m'. The solution is obtained by writing the command 'myfirst' in MATLAB.

```
%SCRIPT file to run the ODE simulation for A->B->C.
s0    = [1 0 0];  %initial values for A,B,C
tspan = [0,10];   %time interval
k1    = 0.7;      %model parameter
k2    = 0.2;      %model parameter
% Call of the MATLAB ODEsolver 'ode23':
[t,s] = ode23(@myfirstode,tspan,s0,[],k1,k2);
%inputs: myfirstode the name of the m-file, where the
%                   ODE is given
%        tspan      time interval where solution wanted
%        s0         initial values at time t=0
%        []         options, empty: not used here
%        k1,k2      model parameters used in 'myfirstode'
%outputs: t    the time points where solution presented,
%         s    the solution matrix
plot(t,s)  %   plot the solution
```

```
function ds = myfirstode(t,s,k1,k2);
%input  t     the time variable (not used in this case)
%       s     the state vector
%       k1,k2 model parameters
%output ds    the derivative ds/dt at time t

A = s(1);  %for clarity & readability, write the
B = s(2);  %model using the notation A,B,C for the
C = s(3);  %components

dA = -k1*A;           %the ODE system equations
dB =  k1*A - k2*B;
dC =       k2*B;
ds = [dA;dB;dC]; %collect the output in vector ds
```

3.1 Parameter estimation for nonlinear dynamical models

Examples of linear and nonlinear models:

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

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

Here x denotes the experimental points, θ the parameters to be estimated.

In both examples the model is written in an *algebraic* form, i.e., in terms of some 'simple' formulas. No numerical solvers are then required.

A *dynamical model* is written as an ODE system, and the solution is obtained by numerical solver.

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	experimental conditions
θ	estimated parameters
const	known constants
y	the observables
f	the model function
g	the observation function

Example 1 Consider again the reaction $A \rightarrow B \rightarrow C$. 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
clear all; % eliminate earlier definitions
% Generate the true solution
odesolver;
%the sampling instants, correspond to tspan in odesolver
t = 0:1:10;
t = t'
% generate noisy observations with noise proportional to solu
y = y + y .* randn(11,1)*0.1;
data = [t y]; %data for the fitting:
```

```
% begin parameter identification
k1 = 0.3; %initial guesses of the unknown
k2 = 0.2; %parameters for optimizer
teta = [k1 k2]; %just collect in 1 vector
%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
```

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);
```

The script 'odesolver' generates the true solution:

```
%SCRIPT file to run the ODE simulation for A->B->C.
s0 = [1 0 0]; %initial values for A,B,C
tspan = [0:1:10]; %time interval with observations at every i
k1 = 0.7; %model parameter
k2 = 0.2; %model parameter
% Call of the MATLAB ODEsolver ode23:
[t,s] = ode23(@myfirstode,tspan,s0,[],k1,k2);
%inputs: myfirstode the name of the m-file, where the
% ODE is given
```

```
% tspan time interval where solution wanted
% s0 initial values at time t=0
% [] options, empty: not used here
% k1,k2 model parameters used in myfirstode
%outputs: t the time points where solution presented,
% s the solution matrix
plot(t,s) % plot the solution
y = s(:,2); % extract component B as the observed quantity
```

The function 'myfirstode' is the same as before:

```
function ds = myfirstode(t,s,k1,k2);
%input t the time variable (not used in this case)
% s the state vector
% k1,k2 model parameters
%output ds the derivative ds/dt at time t
A = s(1); %for clarity & readability, write the
B = s(2); %model using the notation A,B,C for the
C = s(3); %components
dA = -k1*A; %the ODE system equations
dB = k1*A - k2*B; dC = k2*B;
ds = [dA;dB;dC]; %collect the output in vector ds
```

A more complicated example

Chemical kinetics with reactions

$$A + B \xrightarrow{k_1} C + F$$

$$A + C \xrightarrow{k_2} D + F$$

$$A + D \xrightarrow{k_3} E + F$$

Modelled as an ODE:

$$\frac{d[A]}{dt} = -k_1[A][B] - k_2[A][C] - k_3[A][D]$$
$$\frac{d[B]}{dt} = -k_1[A][B]$$
$$\frac{d[C]}{dt} = +k_1[A][B] - k_2[A][C]$$
$$\frac{d[D]}{dt} = +k_2[A][C] - k_3[A][D]$$
$$\frac{d[E]}{dt} = +k_3[A][D].$$

With know initial values $A(0), \ldots, E(0)$ the solution $f(t, \theta)$ is again obtained by one of the `ODExx`"-solvers of Matlab.

- The data: y_i the analyzed concentrations at time points t_i.
- The parameters to be estimated: $\theta = (k_1, k_2, k_3)$.

The system is more complicated, but solved using just the same procedure as above.

The Matlab solution

The ODE solution is obtained by a call to a ODE solver

```
[t,y] = ode45('odefun',time,y0,[],theta);
```

where the function 'odefun' contains the code for the ODE system.

The LSQ objective function to be minimized may be written as

```
function ss = lsqfun(theta,data,y0)
t    = data(:,1);
yobs = data(:,2);
[t,ymodel] = ode45('odefun',t,y0,[],theta);
ss = sum((yobs-ymodel).^2);
```

And the optimization – after necessary initializations – by a call

```
theta_opt = fminsearch('lsqfun',theta0,[],data,y0)
```

Example: heat transfer. A glass of beer is at $t = 0$ in temperature T_0 in a glass. It will be cooled from outside by water, which has a fixed temperature T_{water}. We measure temperatures and get the data (t_i, T_i), $i = 1, \ldots, n$. Based on this data we want to fit parameters in a model that describes the heat transfer between the the glass ('reactor') and water ('cooler'). Note that the heat transfer takes place both through the glass, and via the air/water surface:

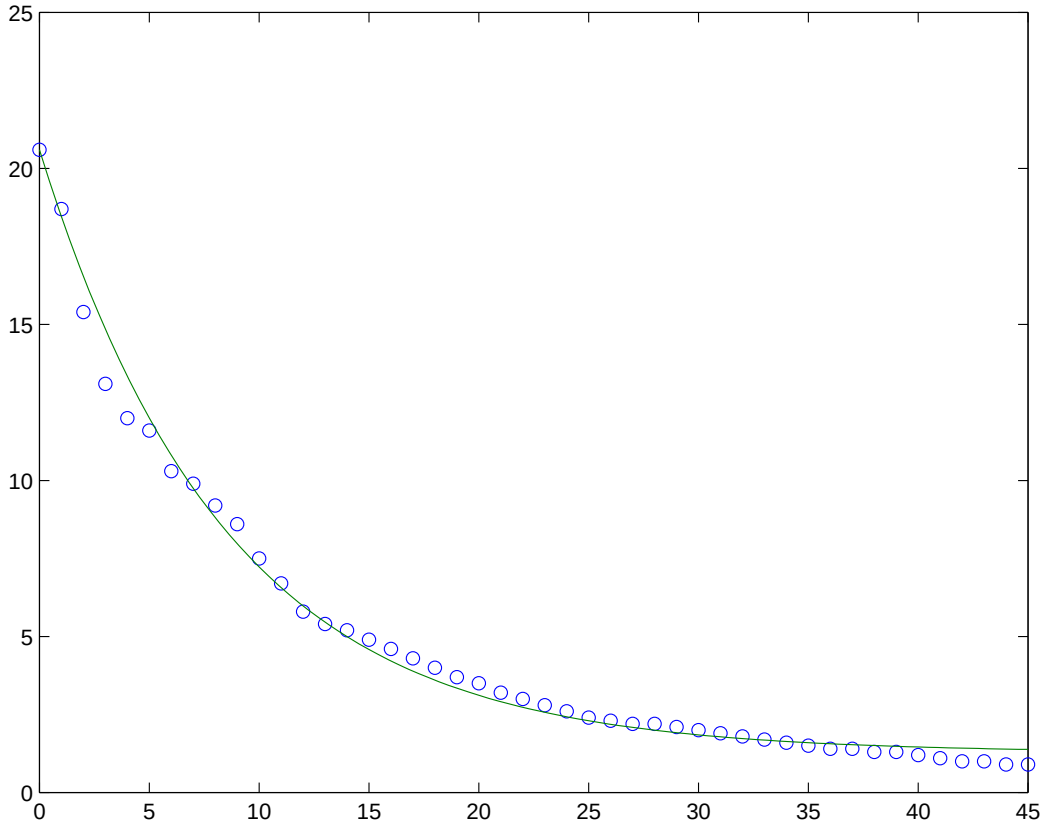
$$dT/dt = -k_1(T - T_{\text{water}}) - k_2(T - T_{\text{air}})$$

The solution may be obtained either by an ODE solver, or by integrating the equation by hand:

$$T(t) = (T_0 - T_{inf})e^{-(k_1+k_2)t} + T_{inf}$$

HereT_{air} is the temperature of the air, $T_{inf} = (k_1 T_{water} + k_2 T_{air})/(k_1 + k_2)$ is the 'steady state' temperature ($T' = 0$) and k_1, k_2 are the unknown parameters to be fitted.

An example fit:



Note the non–ideality of the data.

3.2 Data assimilation for nonlinear dynamical models

Examples of linear and nonlinear models:

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

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

Here x denotes the experimental points. In data assimilation, the "parameter" to be estimated is the initial state of the system, when some later observations are given.

A *dynamical model* is written as an ODE system, just like in parameter estimation, and the solution is obtained by numerical minimization, with the components of the initial state as the independent variables in minimization.

The least squares cost function of data assimilation on a nonlinear model may have very many local minima!

Example 1 Consider again the reaction $A \rightarrow B \rightarrow C$. 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 the initial concentrations $\theta = (A(0), B(0), C(0))$.

Matlab solution

Data assimilation will be carried out again by the FMINSEARCH optimizer. Let us first suppose that only values of B have been measured, and that the reaction rate coefficients are known to be $k_1 = 0.7, k2 = 0.2$.

```
%SCRIPT file for commands to call FMINSEARCH optimizer
% data assimilation for initial conditions
clear all;
% Generate the true solution
odesolver;
%the sampling instants, correspond to tspan in odesolver
t = 0:1:10;
t = t'
% generate noisy observations with noise proportional to solu
y = y + y .* randn(11,1)*0.1;
data = [t y]; %data for the fitting:
```

```
% begin data assimilation
s0 = [0.8 0.2 0]; %first guess for initial values
teta = s0; %just collect in 1 vector
% Call the optimizer:
teta_opt = fminsearch(@assimlsq,teta,[],data);
% INPUT: assimlsq, the filename of the objective function
% teta, the starting point for optimizer
% [] options (not used)
% data parameters needed in my1lsq
% OUTPUT: teta_opt, the optimized value for teta
%ODE solver called once more, to get the optimized solution
s0(1) = teta_opt(1); s0(2) = teta_opt(2); s0(3) = teta_opt(3)
[t,s] = ode23(@myfirstode,t,s0,[],k1,k2);
plot(t,y,'o',t,s) %plot the data vs solution
```

The LSQ objective function is coded in the 'assimlsq' function, different from parameter optimization since the unknown is different:

```
function lsq = assimlsq(teta,data);
%INPUT unknown teta,  k1,k2 known
% data the constants needed:
% data(:,1) time points
% data(:,2) responses: B values
%OUTPUT lsq value
t = data(:,1);
y_obs = data(:,2); %data points
k1 = 0.7; k2 = 0.2; s0(1)=teta(1); s0(2)=teta(2); s0(3)=teta
```

```
%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);
```

The script 'odesolver' generates the true solution, unchanged from parameter estimation or modelling:

```
%SCRIPT file to run the ODE simulation for A->B->C.
s0 = [1 0 0]; %initial values for A,B,C
tspan = [0:1:10]; %time interval with observations at every i
k1 = 0.7; %model parameter
k2 = 0.2; %model parameter
% Call of the MATLAB ODEsolver ode23:
[t,s] = ode23(@myfirstode,tspan,s0,[],k1,k2);
%inputs: myfirstode the name of the m-file, where the
% ODE is given
```

```
% tspan time interval where solution wanted
% s0 initial values at time t=0
% [] options, empty: not used here
% k1,k2 model parameters used in myfirstode
%outputs: t the time points where solution presented,
% s the solution matrix
plot(t,s) % plot the solution
y = s(:,2); % extract component B as the observed quantity
```

The function 'myfirstode' is the same as before:

```
function ds = myfirstode(t,s,k1,k2);
%input t the time variable (not used in this case)
% s the state vector
% k1,k2 model parameters
%output ds the derivative ds/dt at time t
A = s(1); %for clarity & readability, write the
B = s(2); %model using the notation A,B,C for the
C = s(3); %components
dA = -k1*A; %the ODE system equations
dB = k1*A - k2*B; dC = k2*B;
ds = [dA;dB;dC]; %collect the output in vector ds
```

A Recipe for parameter identification with Matlab

You need four separate routines, namely:

1. **ODE step calculator** that calculates the differential of the components, using the appropriate ODE for each component
2. **ODE solver** that sets the initial values and calls a Matlab ODE solver, with Routine 1. given as a function parameter
3. **Parameter estimator** that sets initial guesses for the parameters and calls a Matlab minimizer to find optimal estimates
4. **Cost function calculator** that calculates synthetic observations using the current guesses for the parameters, and computes their squared difference from the observations

The routines one to four are used in the following pattern:

1. Use Routine 3. to set up a minimization problem with a new guess for the parameters
2. Use Routine 2. to set up the creation of synthetic observations with the current guesses at the parameters
3. Use Routine 1. to compute the synthetic observations with the ODE system at hand
4. Use Routine 4. to compute the difference between synthetic and real observations
5. Iterate from 1.

Temperature dependency in chemical kinetics

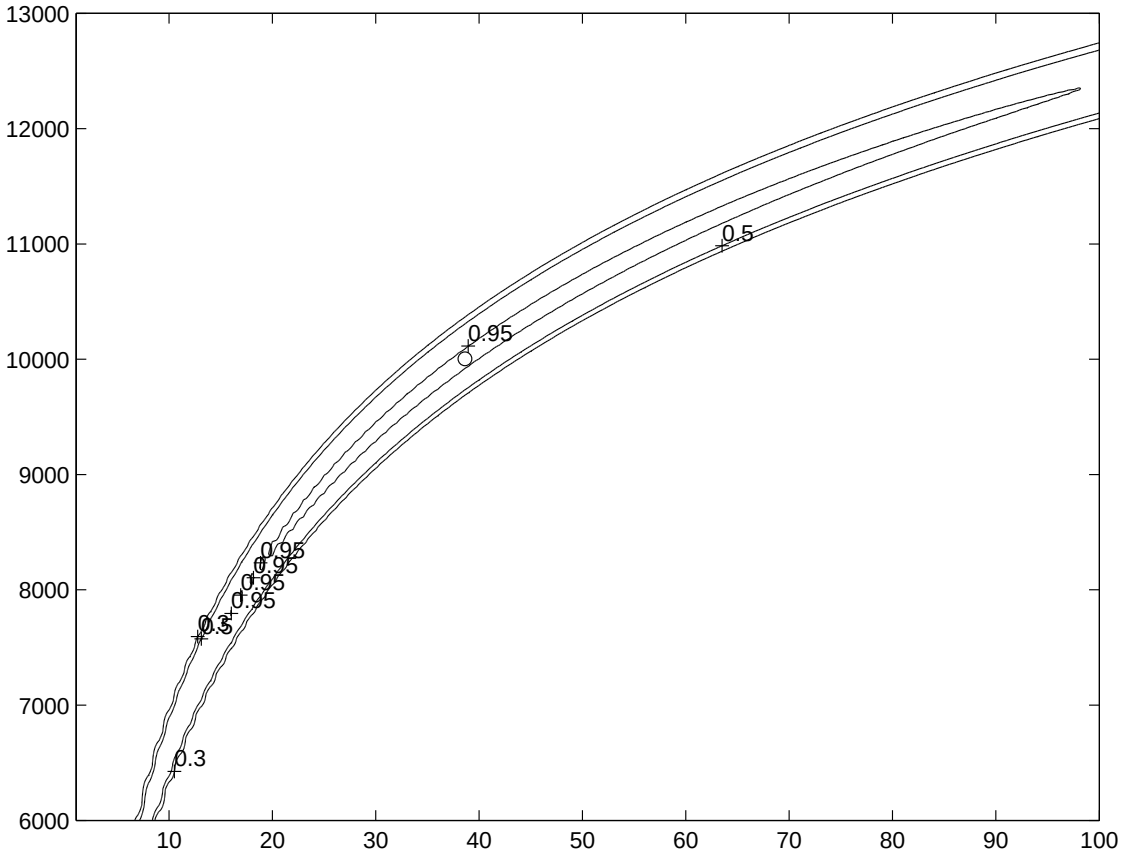
So far the reaction rate parameters have been assumed to be constants. But chemical reactions depend on temperature – they only take place if temperature is high enough. Typically, the dependency is expressed by the *Arrhenius law*:

$$k(T) = A\, e^{-E/RT}$$

where T is temperature (in Kelvin), A the amplitude, E the activation energy, and R the gas constant ($R = 8.314$ in SI units). To find out how the reaction rate depends on temperature, the parameters A and E should determined by measured data.

Example Suppose that the rate coefficient values $k = 0.55, 0.7, 0.8267$ have been measured at the temperatures $T = 283K, 300K, 313K$. The parameters A, E may be fitted in the usual way by least squares. A perfect fit is obtained,

we may compute the R2 value of the fit, $R2 \simeq 1$. But if we compute the R2 value at *other* parameter values, we get the contour picture



Contour of R2 values for the fit of Arrhenius parameters A,E

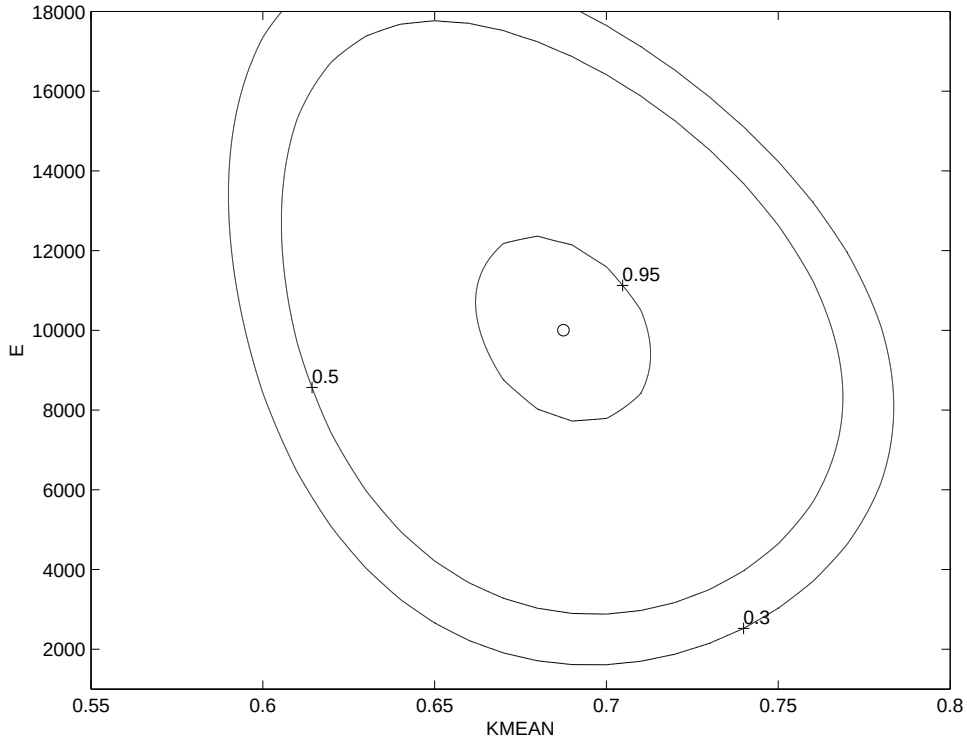
We see that the parameters are *badly identified*: several values on a 'banana shape' region will give an equally good fit (high R2 value).

To overcome the situation, it is customary to make a *change of parameters*:

$$k = Ae^{-E/RT} = k_{mean}e^{-zE}$$

where $k_{mean} = Ae^{-E/RT_{mean}}$, $z = 1/R(1/T - 1/T_{mean})$, and T_{mean} is some 'mean' temperature value, between the minimum and maximum used in the experiments. Instead of the the original A, E we now estimate k_{mean}, E (or E/R, in order to avoid dimensional errors).

Now the contour plot for the 'landscape' of the LSQ objective function becomes nicely roundish, the parameters are well identified - and the optimizer more easily will find the best fit.



Contour of R2 values for the fit of Arrhenius parameters k_{mean}, E

Example: Sterilizing of apple juice

The Arrhenius type temperature dependency is often used for other processes, too. Here we discuss a biological example. Below is a table of data from an experiment where a bottle of apple juice is sterilized by heating

```
%
time        temperature of   temperature of      number of
 t (min)    juice, T (C)     the heat bath,T_h   microbes,B
 0           20               94                 > 50
 1.5         38               94                   50
 3           50               94                   25
 4.5         60               94                   3
 6           69               94                   1
 7.5         78               94                   0
```

To simulate the process we need to assume some model for the decay rate of the microbes, and how it depends on temperature. Let us use the 'standard' assumptions: a first order kinetics for the decay rate, so it is proportional to the number B of bacteria microbes, $Re = k * B$, where the temperature dependency of the parameter k is given by the Arrhenius law.

We have to model the temperature T of the juice, too. The heat transfer is given by the equation

$$mC_p \frac{dT}{dt} = UA(T_h - T)$$

where $m = 0.330kg$ is the mass of the juice, $A = 0.034m^2$ is the area of the bottle, C_p is the specific heat of the juice (we may us the C_p of water), and U is the heat conductivity parameter.

So we arrive at the model, in a form of an ODE system:

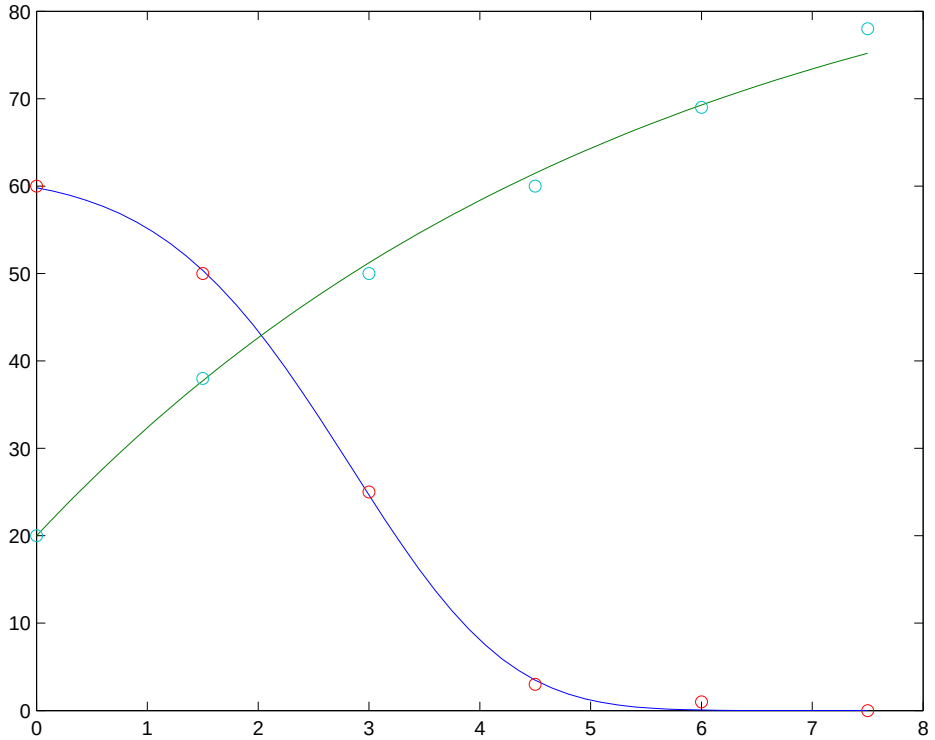
$$\frac{d[T]}{dt} = UA(T_h - T)$$
$$\frac{d[B]}{dt} = -k(T) * B,$$

where $k(T) = k_{\text{mean}} e^{-E/R*(1/T - 1/T_{\text{mean}})}$. Note that the initial value of bacteria density is not properly known, only a lower bound $B > 50$ is given. The unknown parameters to be fitted are the conductivity U and the Arrhenius parameters k_{mean}, E. Note also that the the reference temperature T_{mean} may be freely chosen.

Several questions should be taken into account:

- should $B(0)$ also be estimated?
- could the heat conductivity U be identified independently first?
- is modelling of the heat conductivity necessary, indeed we have measurements for t, T, B and might deal with the latter equation separately?

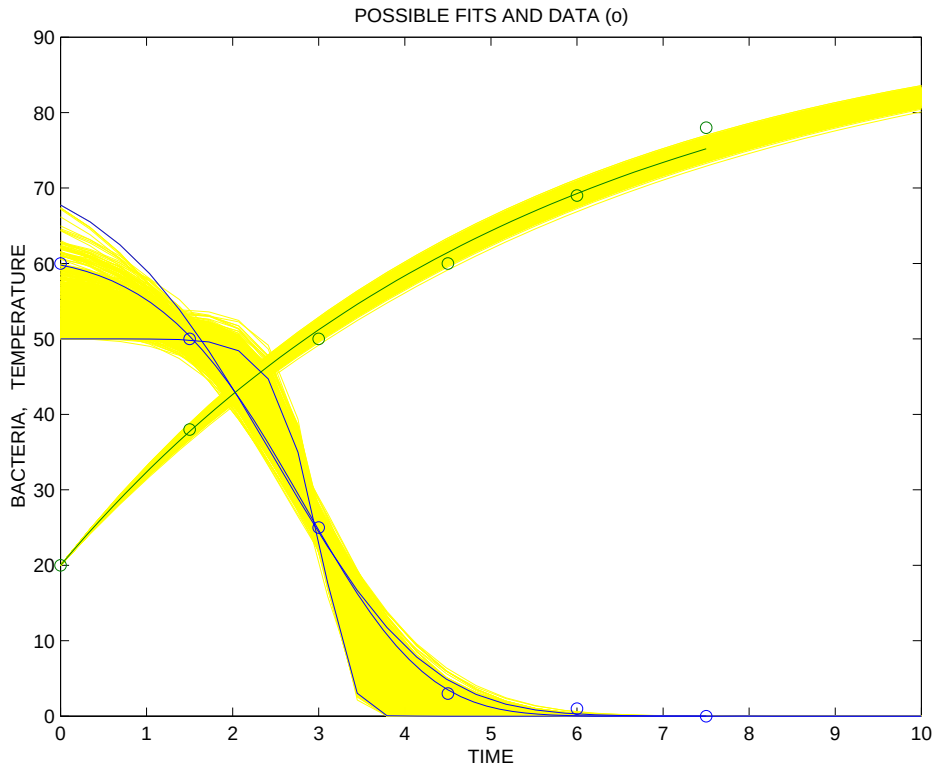
Below is the fit of the model to the data, in case where the initial amount of the bacteria also is one of the estimated parameters:



But how unique is the result? How reliable are the predictions made by the model?

These questions may be answered, if we create *all* the possible fits to the data, instead of just the best fitting one.

The production of 'all' the solutions – all that statistically reasonably well fit the noisy data – may be done by 'Monte Carlo' sampling methods. We skip the details, and only present the solution as a figure:



We see that there is considerable uncertainty in the results, due to the unknown initial amount of the bacteria:

- either the initial amount of bacteria is low (50), almost no bacteria dies at low temperatures, but then rapidly dies at temperatures around 50 C.
- or the initial amount of bacteria is higher (60), and the bacteria starts vanishing already at lower temperatures.

To resolve the ambiguity we should repeat the measurements at room temperatures, or use available 'a priori' knowledge about the behavior of bacteria - it hardly starts decaying at room temperatures.

The heat conductivity U does not depend on the reaction, and it may be indeed estimated separately. Note however, that the uncertainty in estimating U also effects (increases) the uncertainty concerning the decay of bacteria, so they better would be estimated together.

In principle, the decay of bacteria might be calculated separately, too. But the integration of the equation requires temperature values T for all time values t, and the temperatures only are measured for a few time points. So we should somehow interpolate the measured – and noisy – values between the measured time points. Again, the analysis of the modelling results is more reliable if the modelling of B and T is done together.