Inverse Methods in Astronomy: MCMC

Heikki Haario
Lappeenranta University of Technology
heikki.haario@lut.fi

Contents

- Introduction
  - Classical statistics: regression, covariance, error bounds
  - Basic distributions
- Nonlinear Models
  - Parameter Estimation
  - MCMC methods
    - Monte Carlo methods
    - Metropolis algorithm,Adaptive methods
    - Other variants, further topics
    - Using the MCMC Matlab Toolbox

1 Introduction

Types of models

- 'soft', statistical models. 'Black box' calibration of existing data:
  - + Easy computations, problems in interpretation
  - - Limitations: not for predictions of essentially new situations (scale-up, new processes, etc).
  - Methodology employed: *regression analysis*.
- 'hard', mathematical models based on physics
  - - 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.

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.

Mechanistic /first principles modelling Example; radioactive decay

$$A \rightarrow B \rightarrow C$$

Modelling by mass (energy, etc) balances leads to an ODE system

$$\frac{dA}{dt} = -\theta_1 A$$
$$\frac{dB}{dt} = \theta_1 A - \theta_2 B$$
$$\frac{dC}{dt} = \theta_2 B$$

Note that mass preserves: $d/dt(A + B + C) = 0$, $A + B + C =$ constant.

Additional tasks

- Solution of the model numerically by an (e.g., ODE) *solver*.

  Various numerical approaches for more complex models (e.g., CFD: Computational Fluid Dynamics)

  - FD, Finite Difference
  - FEM, Finite Elements

- The model *nonlinear* with respect to unknown parameters θ. LSQ by *optimization algorithms*

Model predictions are always uncertain, due to

- Incomplete understanding of the process
- Numerical etc approximations in implementation
- Measurement noise in data

The model is calibrated to data by 'fitting' the unknown parameters. Uncertainty in data implies uncertainty in fitted values. The task of statistical analysis is to quantify this.

Problem: Classical statistics for linear /empirical models !

Nonlinear models, Monte Carlo (MCMC) methods

Goal: proper statistical analysis for nonlinear models. How reliable are the model predictions?

No formulas exists for statistics of nonlinear models. But the probability distributions may be *created* by random number simulation methods.

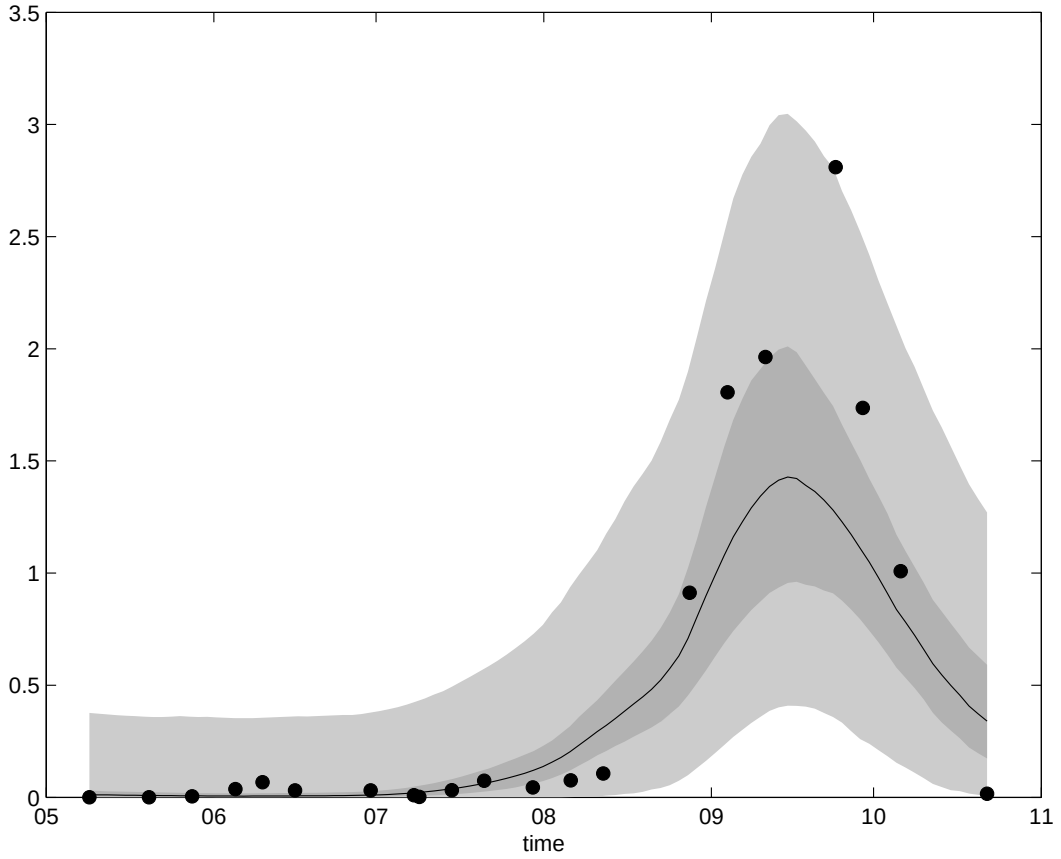
The (only) way to go is *sampling*: we computationally approximations of the distributions as histograms, of the unknown parameters as well as model predictions.

Instead of one 'best fitting' solution, we produce 'all' the solutions that reasonably – statistically well enough, taking noise in data into account – fit the data. This is the essence of the *Bayesian* approach: to find the distribution of solutions, instead of one solution.

A problem, solved by MCMC: we do not know the distribution from which to sample – but still can sample from it!

Example: Modelling an Algae bloom in a lake

The Algae concentration: data, fit, model prediction distribution.



2 Linear Models

Consider a linear model, for example

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

With given data at $x^i, i = 1, 2, ..., n$ the system is written in the form $y = Xb + \epsilon$, where X is the design (observation) matrix (extended with the 1's in the first column), y denotes the measured data, and ϵ represents the measurement noise. The task is to estimate the coefficients b together with their statistical uncertainty.

Often a linear model can not explain the data: the optimum inside the experimental region requires some higher terms to be added in the model.

Basic concepts

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.

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 measurement noise:

$$res \simeq noise.$$

LSQ solution, measurement noise, uncertainty of estimates

The LSQ objective function has the form

$$\sum_{i=1}^{n}(y^i - (x^i)^T b)^2 = ||y - Xb||_2^2$$

It can be shown (Exercise), that the LSQ estimate is obtained as the solution of the *normal equation* $X^T X b = X^T y$,

$$\hat{b} = (X^T X)^{-1} X^T y$$

The most typical assumption for the measurement noise is that the *covariance* of it is given by $cov(y) = \sigma^2 I$: the noise is independent between measurements, with equal size everywhere, given by std σ. Before discussing the uncertainty of the estimates, let us recall some concepts from basic statistics.

Variance and covariance

Consider an *observation matrix* X. Each row gives values of the variables $x_1, ..., x_p$:

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

We need the following basic concepts :

The *mean value*

$$\bar{x}_k = \frac{1}{n}\sum_{i=1}^{n} x_{ik}$$

The *variance*

$$var(x_k) = \frac{1}{n-1}\sum_{i=1}^{n}(x_{ik} - \bar{x}_k)^2 := \sigma_{x_k}^2$$

The *standard deviation*

$$std(x_k) = \sqrt{var(x_k)} := \sigma_{x_k}$$

The *covariance*

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

The *correlation*

$$cor(x_k, x_l) = \frac{\sigma_{x_k x_l}}{\sigma_{x_k}\sigma_{x_l}} := \rho_{x_k x_l}$$

The *covariance matrix*

$$cov(X) = \begin{bmatrix} \sigma^2_{x_1} & \sigma_{x_1 x_2} & \cdots & \sigma_{x_1 x_p} \\ \sigma_{x_1 x_2} & \sigma^2_{x_2} & & \\ \vdots & & \ddots & \\ \sigma_{x_1 x_p} & & & \sigma^2_{x_p} \end{bmatrix}$$

Note that if X_0 denotes the *centered* version of X, obtained by substracting the mean value from each column of X, then

$$cov(X) = \frac{1}{n-1} X_0' X_0$$

The diagonal of $cov(X)$ gives the variances of of the columns of X. Recalling how the *angle between vectors* in a Euclidean space is defined by the normalized inner product of the vectors, we see the geometric interpretation for correlation: the correlation coefficients, the off–diagonals of the correlation matrix, are given as the cosine of angles between the centered column vectors x_{0k}, x_{0l},

$$\text{corrcoef}(x_{0k}, x_{0l}) = \cos(x_{0k}, x_{0l}) = \frac{x_{0k}' x_{0l}}{||x_{0k}||||x_{0l}||}$$

If y denotes a n dimensional *random vector* – each component is a scalar random variable – the covariance matrix is defined as

$$\mathrm{cov}(y) = E[(y - Ey)(y - Ey)'],$$

where E denotes the expected value (**note that y here is a column vector, while the realizations/observations in the observation matrix notation are given as rows**). If the vector y is multiplied by a matrix S, it is easy to see that

$$\mathrm{cov}(Sy) = Scov(y)S'.$$

Especially, $\mathrm{cov}(y) = I$, we have $\mathrm{cov}(Sy) = SS'$. We will return to this fact a bit later, as we learn how to generate samples from multinormal distributions.

Basic distributions

The *probability density function*, PDF, of the Normal (or Gaussian) distribution with center point x_0 and variance σ^2 is given by the formula

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{x-x_0}{\sigma})^2} \tag{1}$$

For a random variable x we write $x \sim N(x_0, \sigma^2)$ if it obeys the distribution given by the above density. An often used fact is that 95% of the probability mass (area under the above function) lies in the interval between the points $x_0 \pm 1.96\sigma$.

If $x = (x_1, ..., x_n)$ is a n–dimensional random vector, each component of which is independently $N(0, 1)$ distributed, we have $\mathrm{cov} x = I$, and write

$x \sim N(0, I)$. By independence, the density function of the vector is the product of the density functions of the components, so we have

$$p(x) = \frac{1}{(\sqrt{2\pi}\sigma)^n} e^{-\frac{1}{2}\sum_{i=1}^{n} x_i^2}$$

This leads us to another standard distribution, the *chi-square distribution* with 'n degrees of freedom'. Indeed, the (central) χ_n^2 distribution is defined as the *sum of squares of n independent* $N(0, 1)$ *distributed* random variables,

$$t = \sum_{i=1}^{n} x_i^2 = x^T x \sim \chi_n^2,$$

if each $x_i \sim N(0, 1)$.

Sampling from multinormal distribution

Let us return to the formula $\mathrm{cov}(Sy) = SS'$, valid if $\mathrm{cov}(y) = I$. It gives us an algorithm to *generate Gaussian random vectors r with a given covariance C*: determine S for which $SS' = C$. Numerically, the 'square root' $S = C^{1/2}$ is computed be the Cholesky decomposition of C. We get the algorithm

- generate Gaussian $N(0, I)$ random vectors y
- decompose $C = SS'$, compute $r = Sy$.

In Matlab, we use the function RANDN to get $N(0, I)$ vectors. Note that for the one dimensional case – generate Normally distributed samples with given variance s^2 – the procedure boils down to the usual computation (with y by RANDN in Matlab),

$$r = s \quad y$$

Criteria for multinormality

Let us denote by $C^{1/2}$ and $C^{-1/2}$ the 'square roots' of C and the inverse of C, both obtained by the Cholesky decomposition of the respective matrix. Then if $\mathrm{cov}x = C$, we get for $y = C^{-1/2}x$ the covariance

$$\mathrm{cov}y = C^{-1/2}C(C^{-1/2})' = C^{-1/2}C^{1/2}(C^{-1/2}C^{1/2})' = I.$$

So y is $N(0, I)$ distributed (we use here the known fact that *if x is Normal, so is Ax* if A is a fixed matrix).

Substituting the expression $y'y = x'C^{-1/2'}C^{-1/2}x = x'C^{-1}x$ in the PDF of $N(0, I)$ above, we get the density function of a general multinormal distribution. It assumes the form

$$p(x) = const\ e^{-\frac{1}{2}x'C^{-1}x}$$

and we write $x \sim N(0, C)$. The normalizing constant may be obtained by a

direct integration or by the above change of variables in the $N(0, I)$ density function (Exercise).

Non-zero centered Gaussian samples $x \sim N(x_0, C)$ are obtained by adding the constant x_0 to $N(0, C)$–distributed samples .

Since the expression $y'y = x'C^{-1}x$ is a sum of n independent Normal $N(0, 1)$ variables it is χ^2_n distributed, i.e.,

$$x'C^{-1}x \sim \chi^2_n.$$

This result gives us a way to quantify *how well a given (sampled) set of vectors x follows a Gaussian distribution with known covariance*.

Linear regression

Let us return to the linear system $y = Xb$. We can show that (Exercise)

$$cov(\hat{b}) = \sigma^2 (X^T X)^{-1}.$$

Especially, the diagonal of the above matrix gives the *variances of the estimated* $\hat{b}$.

In case of Gaussian measurement error we can say more. Since the LSQ estimator $\hat{b}$ is obtained by a matrix multiplication of a Gaussian variable y, $\hat{b}$ *also is Gaussian, with the center point and covariance given by the above formulae*. Moreover, for $C = cov(\hat{b})$,

$$(\hat{b} - b)^T C^{-1} (\hat{b} - b)) \sim \chi^2_n.$$

3 Monte Carlo methods

The above formulae only are valid for linear models. Several 'classical' Monte Carlo methods exist for analyzing linear or nonlinear models, such as:

- Adding noise to data
- Bootstrap
- Jack-knife
- Crossvalidation

Monte Carlo by adding noise to data

Uncertainty in the model parameters θ in a model

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

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 for values of θ a natural idea might be to generate new random noise and repeatedly fit different values $\hat{\theta}$ by the new data sets.

This works if the noise is correctly generated, so that it agrees with the 'real' measurement noise.

But

- Often the noise structure is not properly known, so the generation of new data is questionable
- For a nonlinear model f, each new $\hat{\theta}$ requires a new, iterative, optimization. This might be time consuming.
- The fitted values $\hat{\theta}$ also depend on the optimizer stopping rules (like maximum number of iteration, tolerances, etc), not only on the statistics of the data

Bootstrap

Wikipedia: Bootstrapping alludes to a German legend about Baron Münchhausen, who was able to lift himself out of a swamp by pulling himself up by his own boot straps.

Try to generate the distribution of the model parameters by creating new samples of the data and repeatedly estimating the parameters (by LSQ fitting, typically).

No new data is actually produced, but new combinations from the existing data $x_I, y_I,\ I = [1, ..., n]$ by sampling *with replacement*:

- Randomly choose n indices into the vector ind from 1,2,...,n (some indices may occur more than once, some may be omitted)
- Fit the model to the data x_{ind}, y_{ind}.

Jackknife

As crossvalidation, but without the prediction step. Used for estimating the variability of parameter estimates.

1. Leave out part of the data from the matrixes X, Y
2. Fit the model parameters using the remaining data

   Repeat the steps 1,2 so that each observation has been omitted. Compute estimates for the variability of the parameters (Here, we skip more details)

The R2 value

The R2 value, *coefficient of determination*, measures the goodness of fit. The most simple 'model' is just a constant,

$$y = b_0.$$

The LSQ solution then is the mean value, $\hat{b}_0 = \bar{y}$ (verify this!). A non-dimensional measure for the fit is obtained by comparing with the the trivial one given by a constant:

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

If the model fits the data clearly better than a constant, the R2 is close to the value 1. As a rule of thumb, values 0.7 ... 0.9 are often be considered well enough for empirical models (but this depends on both data and noise!).

Crossvalidation

The R2 value does not tell the whole truth: the R2 of a *fit* may be fine but the model might *predict* poorly. But with given, fixed data we have nothing to predict. The ability to predict may be tested by *crossvalidation*:

1. Leave out part of the data from the matrixes X, Y
2. Fit the model using the remaining data
3. Using the model obtained, predict the values that were left out

   Repeat steps 1,2,3 so that each response value in the matrix Y will be 'predicted'

The goodness of the predictions may be quantified using exactly the same formula as in computing the R2 value. The fitted values are just replaced with the predicted ones.

The number so obtained is called the Q2 value.

Variants

Crossvalidation may be performed in several ways:

- In *Leave-one-out* crossvalidation the observations are omitted deterministically, one by one
- Several observations may be omitted deterministically at each step (1)
- Several observations may be omitted randomly at each step (1)

Even so, the the crossvalidation might give an overly optimistic picture of prediction. One can yet divide the (large enough) data set in two parts: a *learning* data set and a *prediction* set. The model is constructed using only the learning set, the prediction is only tested using the prediction set.

4 Mathematical/'Physicochemical' 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):

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

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.

Parameter estimation for nonlinear models

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

The model may be directly given as a ('algebraic') formula, or it may require a numerical solver.

Example 1 Consider again the reaction $A \to B \to C$, modelled as the ODE system

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

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

Example: Matlab solution

The standard way to estimate the parameters of a model is to minimize the LSQ function ℓ with respect to θ,

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

The parameter estimation will be done by the FMINSEARCH optimizer. Let us first suppose that only values of B have been measured, with an initial values $A(0) = 1.0, B(0) = C(0) = 0$. To do the LSQ fitting, we have to write a script file for initializations, a call of the optimizer, and plots for the solution:

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

```
t     = 0:1:10;     %the sampling instants
data  = [t y];      %data for the fitting:
                    %sampling instants t and measured B
s0    = [1 0 0];    %initial values for ODE
```

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

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

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'
%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);
```

```
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
```

4.1 Approximative error analysis

Finding a 'fit' between data and a model is not enough, one should be able to evaluate the reliability of the results produced by a model. As presented above, direct methods exist for linear models. For nonlinear models the situation is different.

For the LSQ estimator $\widehat{\theta}$ of a linear model $y = X\theta + \epsilon$ we have seen that

$$cov\widehat{\theta} = \sigma^2(X'X)^{-1},$$

(so $\hat{\theta} \sim N(E\widehat{\theta}, (X'X)^{-1}\sigma^2)$), if the measurement error $\epsilon \sim N(0, \sigma^2)$.

For a nonlinear model $y = f(x, \theta) + \epsilon$ it is customary to employ an approximative approach by *linearizing* the model and then simply use the above formula.

Linearization

The Taylor expansion of a function $\ell(\theta)$ of several variables may in general be written as

$$\ell(\theta) = \ell(\hat{\theta}) + \nabla\ell(\hat{\theta})^T(\theta - \hat{\theta}) + \frac{1}{2}(\theta - \hat{\theta})^T H(\theta - \hat{\theta}) + \ldots$$

where H denotes the *Hessian* matrix of the second derivatives of ℓ.

Recalling the specific form of the LSQ function,

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

we easily see that the second derivatives assume the form

$$\frac{\partial^2\ell(\hat{\theta})}{\partial\theta_p\partial\theta_q} = 2\sum_{i=1}^{n}\frac{\partial f(x_i,\hat{\theta})}{\partial\theta_p}\frac{\partial f(x_i,\hat{\theta})}{\partial\theta_q} + 2\sum_{i=1}^{n}(f(x_i,\hat{\theta}) - y_i)\frac{\partial^2 f(x_i,\hat{\theta})}{\partial\theta_p\partial\theta_q}$$

The expression may be simplified if we assume that the residual terms $f(x_i, \hat{\theta}) - y_i$ are 'small' and thus the second sum on the right hand side may be omitted – the rather heuristic argument on behalf on this operation is that the residuals are just minimized. Using this approximation we arrive at the expression

$$H_{pq} = \frac{\partial^2 \ell(\hat{\theta})}{\partial\theta_p \partial\theta_q} \simeq 2\sum_{i=1}^{n} \frac{\partial f(x_i, \hat{\theta})}{\partial\theta_p} \frac{\partial f(x_i, \hat{\theta})}{\partial\theta_q}.$$

The *Jacobian matrix* J of the first derivatives is defined by

$$J_{ip} = \frac{\partial f(x_i; \theta)}{\partial\theta_p}|_{\theta=\widehat{\theta}} \, .$$

So we see that the Hessian may be approximated in matrix form as

$$H \simeq 2J^T J.$$

Substituting the above approximation to a truncated form of the Taylor expansion we get

$$\ell(\theta) \simeq \ell(\hat{\theta}) + (\theta - \hat{\theta})^T J^T J (\theta - \hat{\theta})$$

This expression may be compared to that we get for the linear case, where $\ell(\theta) = ||X\theta - y||^2 = \theta^T X^T X \theta - 2y^T X\theta + y^T y$ is a quadratic polynomial in θ. We may write it as a quadratic polynomial of $\theta - \hat{\theta}$, too. Comparing the second degree terms we see that the formula has to take the form

$$\ell(\theta) = (\theta - \hat{\theta})^T X^T X (\theta - \hat{\theta}) + D,$$

where D does not depend on θ. In fact, the substitution $\theta = \hat{\theta}$ yields $D = \ell(\hat{\theta}) = ||X(X^T X)^{-1} X y - y||$, so in the linear case we have the equality

$$\ell(\theta) = \ell(\hat{\theta}) + (\theta - \hat{\theta})^T X^T X (\theta - \hat{\theta})$$

So in the linear approximation the Jacobian matrix assumes the role of the design matrix X of a linear model, and we can write

$$cov(\widehat{\theta}) \simeq \sigma^2 (J'J)^{-1}$$

Here we may know the std of the measurement error, σ, by, for instance, replicate measurements. Often, however, σ must be estimated by the residuals of the fit. Note that then one has to *assume* that the fit fulfills the requirement $residuals \simeq measurement\ error$.

By using the standard notations, the estimate is obtained by the formula

$$\mathrm{MSE} = \mathrm{RSS}/(n-p)$$

where RSS (Residual Sum of Squares) is the fitted value of the least squares objective function, and MSE (Mean Square Error) is computed as an average value of residual squares, corrected by the 'degrees of freedom', the number of parameters p.

5 Bayesian modelling and Monte Carlo-methods

In the Bayesian approach the unknown vector θ is interpreted as a random variable. The aim of the analysis is to find the distribution of it. Before an experiment θ has a *prior distribution* $p(\theta)$. The observations y update the distribution $p(\theta)$ to the *posterior distribution* $\pi(\theta) = p(\theta|y)$:

$$\pi(\theta) = \frac{p(y|\theta)p(\theta)}{\int p(y|\theta)p(\theta)\,d\theta} \quad \text{The Bayes formula,} \tag{2}$$

Here

$p(\theta)$	the prior distribution
$\pi(\theta)$	the posterior distribution
$p(y\|\theta)$	the likelihood function, contains the model
$\int p(y\|\theta)p(\theta)\,d\theta$	the normalizing constant, due to which $\int_\theta \pi(\theta)\,d\theta = 1$

The likelihood function $p(y|\theta)$ is the probability distribution *of the observations* when the parameter values are given. The most 'likely' values of the parameters are those that give high values for the posterior $p(\theta|y) = p(y|\theta)p(\theta)$.

In MAP (*maximum a posteriori*) estimation one maximizes the posterior,

$$\max_{\theta} p(\theta|y) = \max_{\theta} p(y|\theta)p(\theta).$$

If we do not want to specify the prior – or we want to use the 'uninformative prior'– we may set $p(\theta) = 1$. The above task reduces to the ML, *maximum likelihood*, estimation:

$$\max_{\theta} p(y|\theta).$$

The least squares estimate $\widehat{\theta}$ often (but not always!) is the maximal value or *mode* of the distribution $\pi(\theta)$.

Example Let $y = f(x;\theta) + \epsilon$, where the experimental error $\epsilon \sim N(0, \sigma^2 I)$. So the error terms $\epsilon_i = y_i - f(x_i;\theta)$ are independent, and each normally distributed:

$$p(y_i|\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_i - f(x_i;\theta))^2/2\sigma^2}.$$

By independence, the combined probability $p(y|\theta)$ is obtained as the product

$$\begin{aligned} p(y|\theta) &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_i - f(x_i;\theta))^2/2\sigma^2} \\ &= \frac{1}{(2\pi\sigma^2)^{n/2}} e^{-\sum_{i=1}^{n} (y_i - f(x_i;\theta))^2/2\sigma^2}. \end{aligned}$$

So we arrive at the familiar LSQ function, and maximizing the likelihood function turns out to be equivalent to minimizing the LSQ function.

Problems with the Bayes formula

In principle, the Bayes formula solves the estimation problem in a fully probabilistic sense: we find the peak, the MAP point, of the parameter distribution and determine a required portion of the probability mass – typically some 95% or 99% of the mass – around it. However, we face the problems

- how to define the a prior distribution
- how to calculate the integral of the normalizing constant

The integration for the normalizing constant, especially, is a difficult task in high dimensional, nonlinear cases (dimension higher than 2 or 3!). Only recently the approach has become truly accessible, due to various Monte Carlo methods.

MCMC methods

Markov chain Monte Carlo (MCMC) algorithms generate a sequence of parameter values $\theta_1, \theta_2, ...$ whose empirical distribution, in the 'histogram sense', asymptotically approaches the posterior distribution.

Note the problem here: we can not directly sample form an *unknown* distribution. The way around the problem is to somehow generate 'candidate' points, and then suitably *accept* or *reject* them. Intuitively, a correct distribution of points θ is generated by favoring points with high values of $\pi(\theta)$.

The generation of the vectors in the *chain* $\theta_n, n = 1, 2, ...$ is done by random numbers - *Monte Carlo* generally refers to methods based on random number generation. Each new point θ_{n+1} may only depend on the previous point θ_n. This is the *Markov property*.

Accept–Reject methods

The idea of sampling from a distribution by an 'accept–reject' may be done by the following simple idea: if we generate a uniform sample on a set B and $A \subset B$, then the restriction of the sample on A is a uniform sample on A.

Suppose, for simplicity, that f is a positive (but non-normalized) function on the interval $[a, b]$, bounded by M. If we generate a uniform sample of points (x_i, u_i) on the box $[a, b] \times [0, M]$, the points that satisfy $u_i < f(x_i)$ form a uniform sample under the graph of f - the area of any slice $\{(x, y)|x_l < x < x_u, y \leq f(x)\}$ is proportional to the number of sampled points in it. So the (normalized) histogram of the points x_i gives an approximation of the (normalized) PDF given by f.

The method is not restricted to 1D, the interval $[a, b]$ may have any dimension. We arrive at the following algorithm:

- Sample $x \sim U([a, b])$, $u \sim U([0, M])$.
- Accept points x for which $u < f(x)$

So, we have a (very!) straightforward method that, in principle, solves 'all' sampling problems. However, difficulties arise in practice: how to choose M and the interval $[a, b]$. M it must be larger that the maximum of f, but a too large value leads to many rejected samples. The interval $[a, b]$ should contain the support of f – which generally is unknown. A too large interval again leads to many rejections. Moreover, uniform sampling in a high dimensional interval is inefficient in any case, if the support of f only is a small subset (e.g., a thin 'banana') of it.

A more effective way for accept/reject is to somehow choose an –as such artificial – auxiliary **proposal** distribution, sample from it, and then apply a suitable accept/reject rule. This is the core idea of MCMC (Markov chain Monte Carlo) methods. The most simple MCMC method is the *Metropolis algorithm*:

- 1 Initialize by choosing a starting point θ_1
- 2 Choose a new candidate $\hat{\theta}$ from a suitable **proposal distribution** $q(.|\theta_n)$, that may depend on the previous point of the chain.
- 3 **Accept** the candidate with probability

$$\alpha(\theta_n, \hat{\theta}) = \min\left(1, \frac{\pi(\hat{\theta})}{\pi(\theta_n)}\right).$$

  If rejected, repeat the previous point in the chain. Go back to item 2.

So points with $\pi(\hat{\theta}) > \pi(\theta_n)$ are always accepted (go 'upwards'). If $\pi(\hat{\theta}) < \pi(\theta_n)$ (go 'downwards'), the point may still be accepted, with probability that is given by the *ratio* of the π values. In practice, this is done by generating a uniformly distributed random number $u \in [0, 1]$ and accepting $\hat{\theta}$ if $u \leq \pi(\hat{\theta})/\pi(\theta_i)$.

Note that only the ratios of π at different points are needed – so the 'difficult' normalizing constant cancels out and is not needed!

Problem: the proposal distribution should be chosen so that the 'sizes' of the proposal q and target suitably match. This often may be difficult. An unsuitable proposal leads to inefficient sampling, typically due to

- the proposal is too large. Then the new candidates mostly miss the essential support π, they are chosen at points where $\pi \simeq 0$ and only rarely accepted.
- the proposal is too small. The new candidates mostly are accepted, but from a small neighborhood of the previous point. So the chain moves only slowly, and may, in finite number of steps, not cover the target π.

Naturally, the 'size' of the proposal distribution is not a sufficient specification. In higher dimensions, especially, the shape and orientation of the proposal are crucial. The most typical proposal is a multidimensional Gaussian (Normal) distribution. In the (most typical) *random walk* version, the center point of the Gaussian proposal is chosen to be the current point of the chain. The problem then is to find a covariance matrix that produces efficient sampling.

Standard nonlinear model

The above algorithm is valid for any distribution π, but typical examples deal with parameter estimation by least squares. We then mostly assume Gaussian measurement noise.

So consider a nonlinear model, with independent and Gaussian noise,

$$y = f(x, \theta) + \epsilon, \quad \epsilon \sim N(0, \sigma^2 I).$$

As was shown earlier, the (normalized) likelihood function assumes the form

$$p(y|\theta) = const \exp\left\{-\frac{1}{2\sigma^2} SS_\theta\right\},$$

where

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

and $const$ is the normalizing constant that cancels out.

If we employ the noninformative prior $p(\theta) = 1$ and a known constant for σ^2, the Metropolis-algorithm assumes the form

- Initializations. Choose θ_0, the chain length M and the proposal distribution q
- Generate θ_{new} from the distribution $q(\theta_{\text{old}}, \cdot)$ and compute SS_{new}. The new value is accepted, if $SS_{\text{new}} < SS_{\text{old}}$ or if

$$u \leq \exp\left\{-\frac{1}{2\sigma^2}(SS_{\text{new}} - SS_{\text{old}})\right\}$$

  where $u \sim U(0, 1)$
- Return to the previous item until M samples has been created.

Ergodicity

The theoretical correctness of MCMC methods may be expressed by the following 'ergodicity' theorem (we skip the theory so far, however, and only present a formal theorem as an example).

Let $\theta_0, \theta_1, ...\theta_n$ be the samples produced by a MCMC algorithm. The following should be valid (and indeed is, for example for the Metropolis algorithm with a fixed proposal):

Theorem Let π be the density function of a target distribution in the Euclidean space R^d. Then the MCMC algorithm simulates the distribution π correctly: for an arbitrary bounded and measurable function $f : R^d \to R$ it ('almost surely') holds that

$$\lim_{n\to\infty} \frac{1}{n+1}(f(\theta_0) + f(\theta_1) + \ldots + f(\theta_n)) = \int_{R^d} f(\theta)\pi(d\theta).$$

Intuitively, ergodicity (typically) means that, for a function that depends both on time and space, the time average at a fixed point in space equals the space average at a given time point.

Here, the 'time' average – the left hand side of the equation – is obtained via the discrete sampling by the algorithm, the 'space' average – the right hand side – by the integration over the probability distribution. The theorem simply states that the sampled values asymptotically approach the theoretically correct ones.

Note the role of the function f. If f is the characteristic function of a set A, i.e. $f(\theta) = 1$ if $\theta \in A$, $f(\theta) = 0$ otherwise, then the right hand side of the equation gives the probability measure of A, while the left hand side gives the frequency of 'hits' to A by the sampling.

But f might also be our model, $f(\theta)$ the model prediction at the parameter value θ. The theorem then states that the values calculated at the sampled parameters correctly gives the distribution of the model predictions.

Classical theorems on convergence of random samples are given in several text books – or may be easily found by, e.g., Google. Below are a few sites to recall the Law of Large Numbers and the Central Limit Theorem:

http://en.wikipedia.org/wiki/Law_of_large_numbers

http://en.wikipedia.org/wiki/Central_limit_theorem

http://mathworld.wolfram.com/WeakLawofLargeNumbers.html

http://mathworld.wolfram.com/CentralLimitTheorem.html

5.1 Metropolis-Hastings algorithm

So far we have (implicitly!) supposed that the proposal distribution is *symmetric*, $q(\theta, \hat{\theta}) = q(\hat{\theta}, \theta)$ - proposing $\hat{\theta}$ from θ is as likely as proposing θ from $\hat{\theta}$.

With the Metropolis acceptance probability

$$\alpha(\theta, \hat{\theta}) = \min(1, \frac{\pi(\hat{\theta})}{\pi(\theta)}),$$

the probability for the move will then be

$$p(\theta, \hat{\theta}) = q(\theta, \hat{\theta})\alpha(\theta, \hat{\theta})$$

Now it is straightforward to check that the so called *detailed balance* equation holds:

$$\pi(\theta)p(\theta, \hat{\theta}) = \pi(\hat{\theta})p(\hat{\theta}, \theta),$$

From here it follows that π is a *stationary (invariant)* distribution of the chain:

$$\int \pi(\theta)p(\theta,\hat{\theta})\,d\theta = \pi(\hat{\theta}).$$

This is a key ingredient (while more theory is needed!) for proving the ergodicity, that the sampling theoretically produces correct results.

The procedure can be generalized for nonsymmetric proposals. The acceptance probability proposed by Hastings reads as

$$\alpha(\theta,\hat{\theta}) = \min(1, \frac{\pi(\hat{\theta})q(\hat{\theta},\theta)}{\pi(\theta)q(\theta,\hat{\theta})}).$$

Again, is is easy to verify that the detailed balance equations holds. Various other MC algorithms can be constructed by 'weighting' the acceptance ratio so that the detailed balance holds.

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:

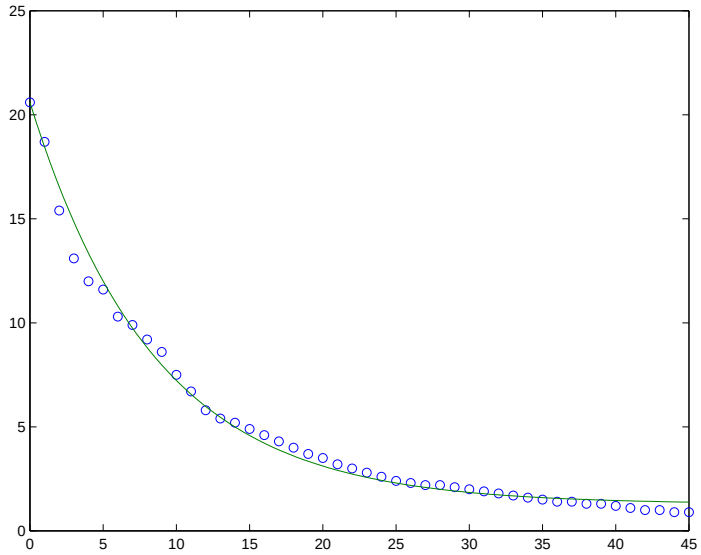
$$dT/dt = -k_1(T - T_{\text{water}}) - k_2(T - T_{\text{air}})$$

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

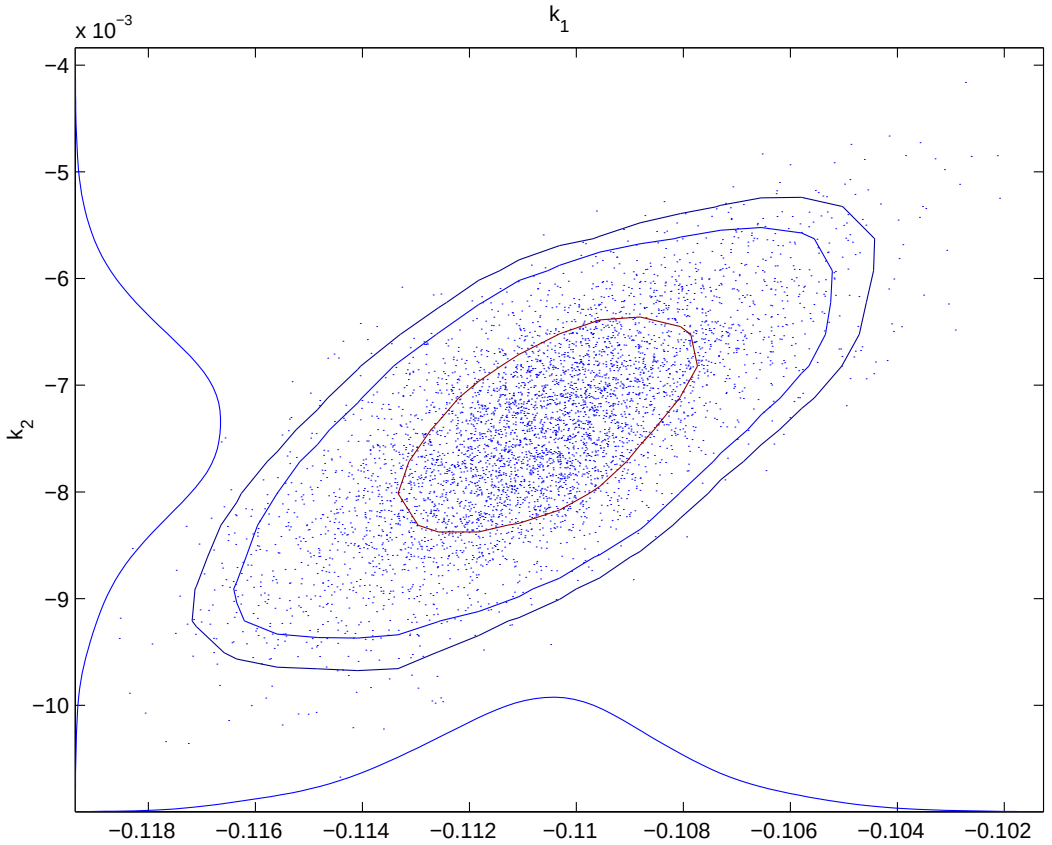
$$T(t) = (T_0 - T_{inf})e^{-(k_1+k_2)t} + T_{inf}$$

Here T_{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 as a nonlinear LSQ problem, solved by (for example) the FMINSEARCH optimizer in MATLAB:



(Note the non–ideality of the data!) For the LSQ fit we have coded an objective function 'lsqbeer'. The same function is now used in running the MCMC chain for the problem. We may start the chain at the LSQ fit point and might use approximative covariance $cov(\hat{\theta}) = \sigma^2(J'J)^{-1}$, or select some other proposal by trial and error.



2D posterior and 1D marginal posteriors for k_1 and k_2 by the MCMC-chain. The empirical contour curves at 50%, 90% ja 95% probability levels.

```
%%% Statistical analysis by MCMC simulation

nsimu  = 10000;                          % length of chain
sigma2 = 0.2;                            % variance of meas.error
% call of the optimizer:
par0   = fminsearch('lsqbeer',[0.1,0.1],[],t,T);
npar   = 2;                              %  n of parameters,k1 ja k2
chain  = zeros(nsimu,npar);              %  initialize the chain
qcov   = [1e-6 0;0 1e-6];                %  covariance for Gaussian proposal
R      = chol(qcov);                     %  Cholesky decomposition
xdata  = t; ydata  = T; oldpar = par0;
ss     = lsqbeer(oldpar,xdata,ydata);% first SS value
rej        = 0;                          % initialized count for rejections

chain(1,:) = oldpar;
for i=2:nsimu % simulation loop
   newpar = oldpar+randn(1,npar)*R;      % new candidate
   ss2    = ss;                          % old SS
   ss1    = lsqbeer(newpar,xdata,ydata);% new SS
   if (ss2<ss1 & (rand(1,1) > exp(-0.5*(ss1-ss2)/ sigma2)))
```

```
      chain(i,:) = oldpar;              % reject
      rej        = rej+1;
   else
      chain(i,:) = newpar;              % accept
      oldpar     = newpar;
      ss         = ss1;
   end
end

accept = 1-rej./nsimu;                  % acceptance rate
```

Suppose that the data has been sampled too late, in the sense that the reaction has already reached a steady–state equilibrium at the sampling times. It is clear that from such data the values of the parameters can not be separately determined, only the ratio k_1/k_2 may be identified, as well as lower bounds for k_1 and k_2. Without priors, the possible values for k_1 and k_2 would lie in a practically infinite "zone" in a direction where k_1/k_2 is constant.

5.2 Adaptive methods, AM

The bottleneck in MCMC (Metropolis) calculations often is to find a proposal that 'matches' the target distribution, so that the sampling will be effective. This may lead to a time-consuming trial-and-error 'tuning' of the proposal.

Various *adaptive* methods have been developed in order to improve the proposal during the run. One relatively simple way is to compute the covariance matrix of the chain and use it as the proposal, i.e., use a Gaussian proposal distribution whose covariance is the covariance matrix so computed.

Now the new point depends not just on the previous point, but on the earlier history of the chain. So the algorithm no more is Markovian. However, if the adaptation is based on an *increasing part* of the chain, one can indeed prove that the algorithm produces a correct (ergodic) result. Intuitively, the adaptation slows down with the chain length, and the process approaches the usual Metropolis algorithm.

The crucial point regarding the AM adaptation is how the covariance of the proposal distribution depends on the history of the chain. We take, possibly after an initial non–adaptation period, the Gaussian proposal to be centered at the current position of the Markov chain, X_n, and set its covariance to be: $C_n = s_d \mathrm{Cov}(X_0, \ldots, X_{n-1}) + s_d \varepsilon I_d$, where s_d is a parameter that depends only on the dimension d of the sampling space, and $\varepsilon > 0$ is a constant that we may choose very small.

In order to start the adaptation procedure an arbitrary strictly positive definite initial covariance, C_0, is chosen according to a priori knowledge (which may be quite poor). A time index, $n_0 > 0$, defines the length of the initial non–adaptation period and we let

$$C_n = \begin{cases} C_0, & n \leq n_0 \\ s_d \mathrm{Cov}(X_0, \ldots, X_{n-1}) + s_d \varepsilon I_d, & n > n_0. \end{cases}$$

Recall the definition of the empirical covariance matrix determined by points $X_0, \ldots, X_k \in R^d$:

$$\mathrm{Cov}(X_0, \ldots, X_k) = \frac{1}{k}\left(\sum_{i=0}^{k} X_i X_i^T - (k+1)\overline{X}_k\,\overline{X}_k^T\right),$$

where $\overline{X}_k = \frac{1}{k+1}\sum_{i=0}^{k} X_i$ and the elements $X_i \in R^d$ are considered as column vectors.

The covariance C_n satisfies the recursive formula:

$$C_{n+1} = \frac{n-1}{n}C_n + \frac{s_d}{n}\left(n\overline{X}_{n-1}\overline{X}_{n-1}^T - (n+1)\overline{X}_n\overline{X}_n^T + X_n X_n^T + \varepsilon I_d\right).$$

which permits the calculation of the covariance matrix without excessive computational cost (the mean, $\overline{X}_n$, also has an obvious recursive formula).

This form of adaptation can proved to be ergodic. Note that the same adaptation, but with a *fixed* update length for the covariance, is *not ergodic*.

The choice for the length of the initial non–adaptive portion of the simulation, n_0, is free. The larger it is, the longer it takes for adaptation to start. IT has been found that the adaptation might not be done at each time step, but only at given time intervals. This form of adaptation improves the mixing properties of the algorithm, especially for high dimensions. So the index n_0, in fact, can be used to define the length of non–adaptation periods during the whole chain.

The role of the parameter ε is just to ensure that, theoretically, C_n will not become singular. In most practical cases ε can be safely set to zero. The scaling parameter usually is taken as $s_d = 2.4^2/d$.

As a pseudocode:

- Choose the lenght of the chain, Nchain, and initialize θ^1, C_1 - for example, choose the θ_1 given by a LSQ fitting and take C_1 as the approximative covariance calculated at θ_1 by linearization.
- For k=1,2,...,Nchain
  - the Metropolis step, using proposal $N(\theta^k, C_k)$.
  - update $C_{k+1} = \mathrm{cov}[\theta_1, ..., \theta_k]$.

The algorithm may be implemented in several variations. One may compute the covariance by the whole chain $[\theta_1, ..., \theta_k]$ or by an increasing part of it, for instance $[\theta_k/2, ..., \theta_k]$. The covariance may also be updated only after a given number of steps k, instead of every step.

5.3 DR, the Delayed Rejection algorithm

Suppose the current position of a sampled chain is $\theta_n = \theta$. As in a regular MH, a candidate move, $\hat{\theta}_1$, is generated from a proposal $q_1(\theta, \cdot)$ and accepted with the usual probability

$$\alpha_1(\theta, \hat{\theta}_1) = 1 \wedge \frac{\pi(\hat{\theta}_1) q_1(\hat{\theta}_1, \theta)}{\pi(\theta) q_1(\theta, \hat{\theta}_1)} \tag{3}$$

Upon rejection, instead of retaining the same position, $\theta_{n+1} = \theta$, as we would do in a standard MH, a second stage move, $\hat{\theta}_2$, is proposed. The second stage proposal is allowed to depend not only on the current position of the chain but also on what we have just proposed and rejected: $q_2(\theta, \hat{\theta}_1, \cdot)$.

It can be shown that an ergodic chain is created, if the second stage proposal is accepted with probability

$$\alpha_2(\theta, \hat{\theta}_1, \hat{\theta}_2) = 1 \wedge \frac{\pi(\hat{\theta}_2) q_1(\hat{\theta}_2, \hat{\theta}_1) q_2(\hat{\theta}_2, \hat{\theta}_1, \theta)[1 - \alpha_1(\hat{\theta}_2, \hat{\theta}_1)]}{\pi(\theta) q_1(\theta, \hat{\theta}_1) q_2(\theta, \hat{\theta}_1, \hat{\theta}_2)[1 - \alpha_1(\theta, \hat{\theta}_1)]} \tag{4}$$

This process of delaying rejection can be iterated to try sampling from further proposals in case of rejection by the present one. However, in many cases the essential benefit – more accepted point in a situation where one (e.g., Gaussian) proposal does not seem to work properly – is already reached by the above '2 stage' DR algorithm. Moreover, more proposals introduce more 'tuning' work to get the algorithm working.

5.4 DR with adaptation: DRAM

It is possible to combine the ideas of adaptation and delayed rejection. To avoid complications, a direct way of implementing AM adaptation with an m–stage DR algorithm is suggested:

- The proposal at the first stage of DR is adapted just as in AM: the covariance C_n^1 for the Gaussian proposal is computed from the points of the sampled chain, no matter at which stage of DR these points have been accepted in the sample path.
- The covariance C_n^i of the proposal for the i-th stage ($i = 2, ..., m$) is always computed as a scaled version of the proposal of the first stage, $C_n^i = \gamma_i C_n^1$, with a fixed scaling factor γ_i.

The scale factors γ_i can be freely chosen: the proposals of the higher stages can have a smaller or larger variance than the proposal at earlier stages.

We have seen that AM alone typically recovers from an initial proposal that is too small, while the adaptation has difficulties if no or only a few accepted points are created in the start. So a good default is to use just a 2 stage version where the second proposal is scaled down from the (adapted) proposal of the 1. stage.

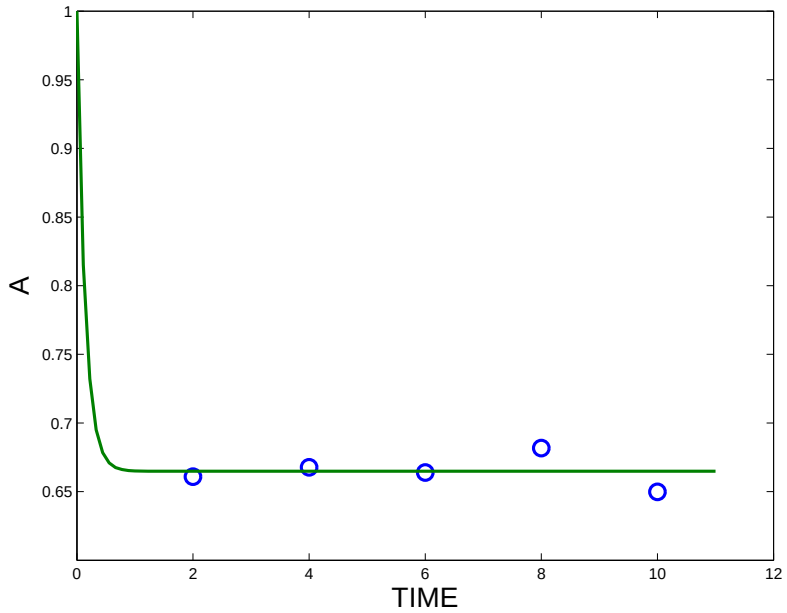
Example: singular covariance, physical bounds on parameters

Consider a simple chemical reaction $A \rightarrow^{k1} B$, $B \rightarrow^{k2} A$, where a component A goes to B in a reversible manner, with reaction rate coefficients k_1 and k_2. The dynamics are given by the ODE system:

$$\frac{dA}{dt} = -k_1 A + k_2 B, \quad \frac{dB}{dt} = k_1 A - k_2 B,$$

with initial values fixed as $A_0 = 1, B_0 = 0$ at $t = 0$.

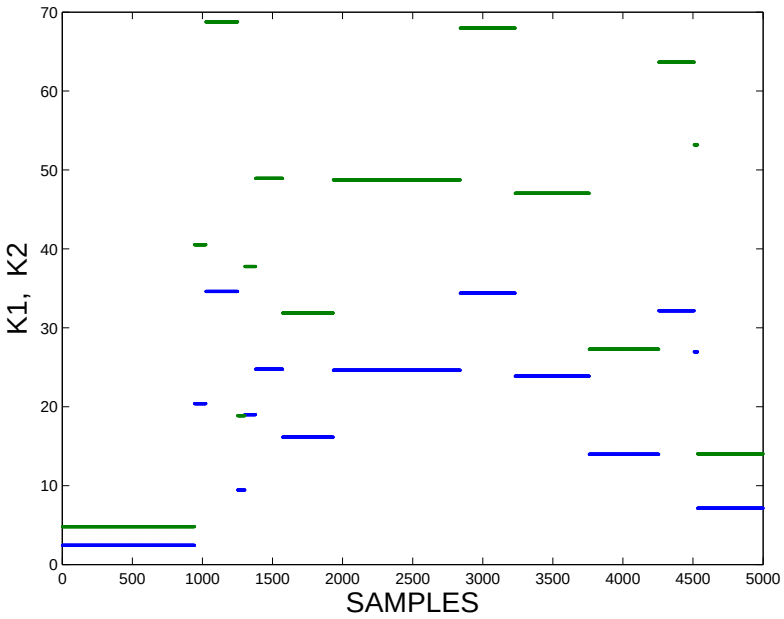
Estimate k_1 and k_2 when data for A has been obtained at given sampling times of t, but too late, at equilibrium.

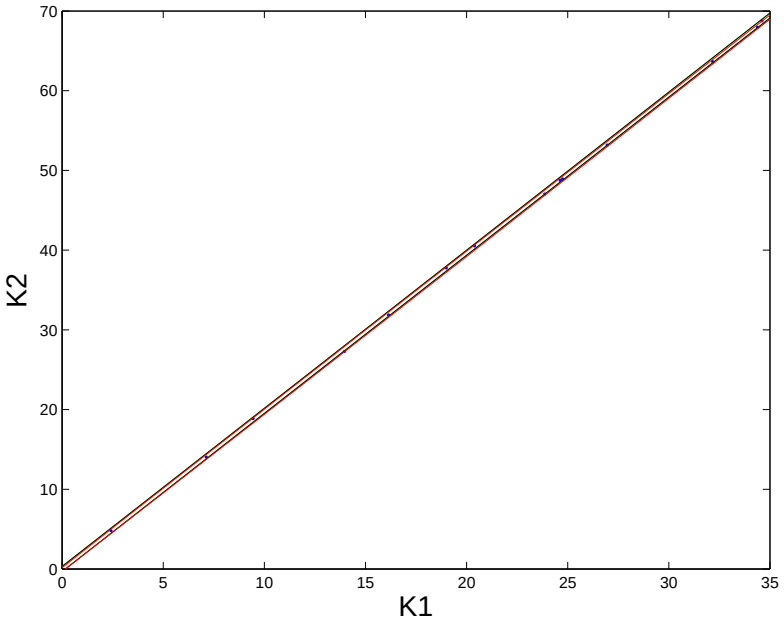


The values of the parameters can not be separately determined, only the ratio k_1/k_2 may be identified, as well as lower bounds for k_1 and k_2.

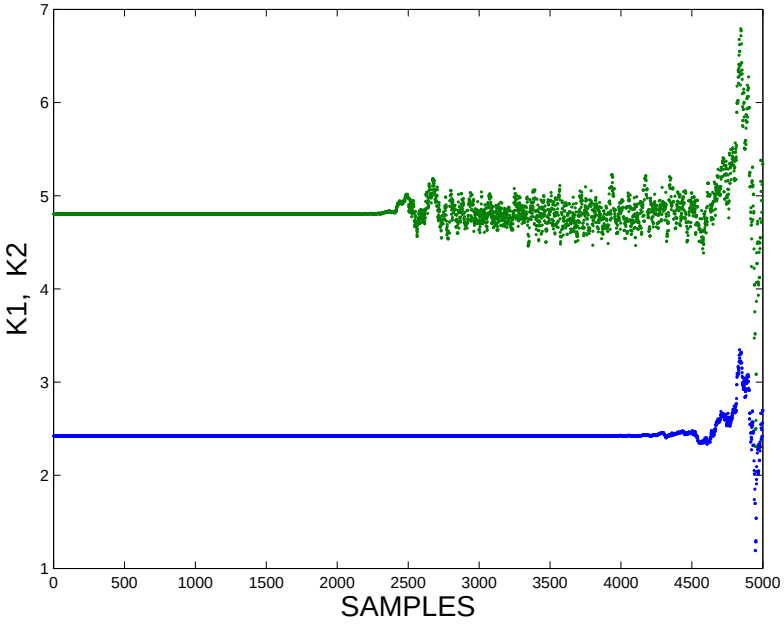
With given prior upper bounds for k_1 and k_2, start MCMC with the proposal given by the Jacobian approximation.

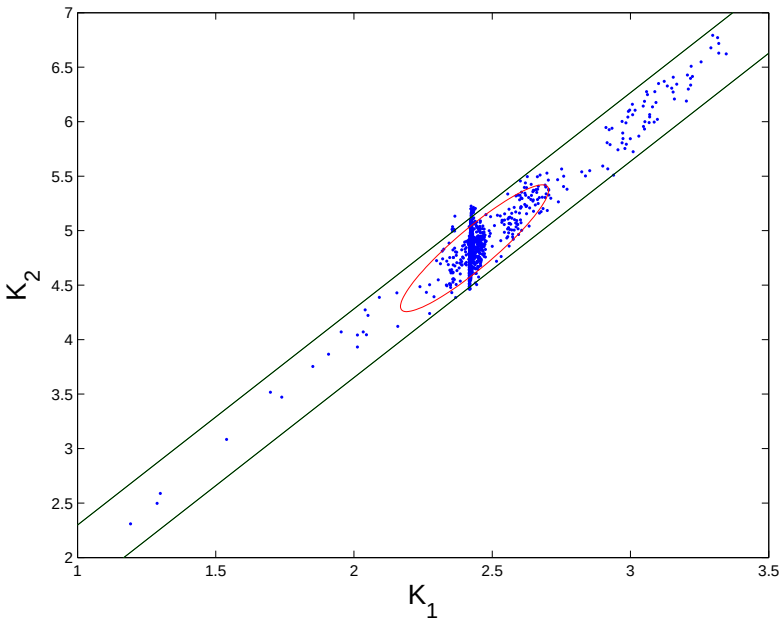
- MH: almost no accepted points, proposed points beyond the bounds (but without prior bounds all proposals accepted!) .



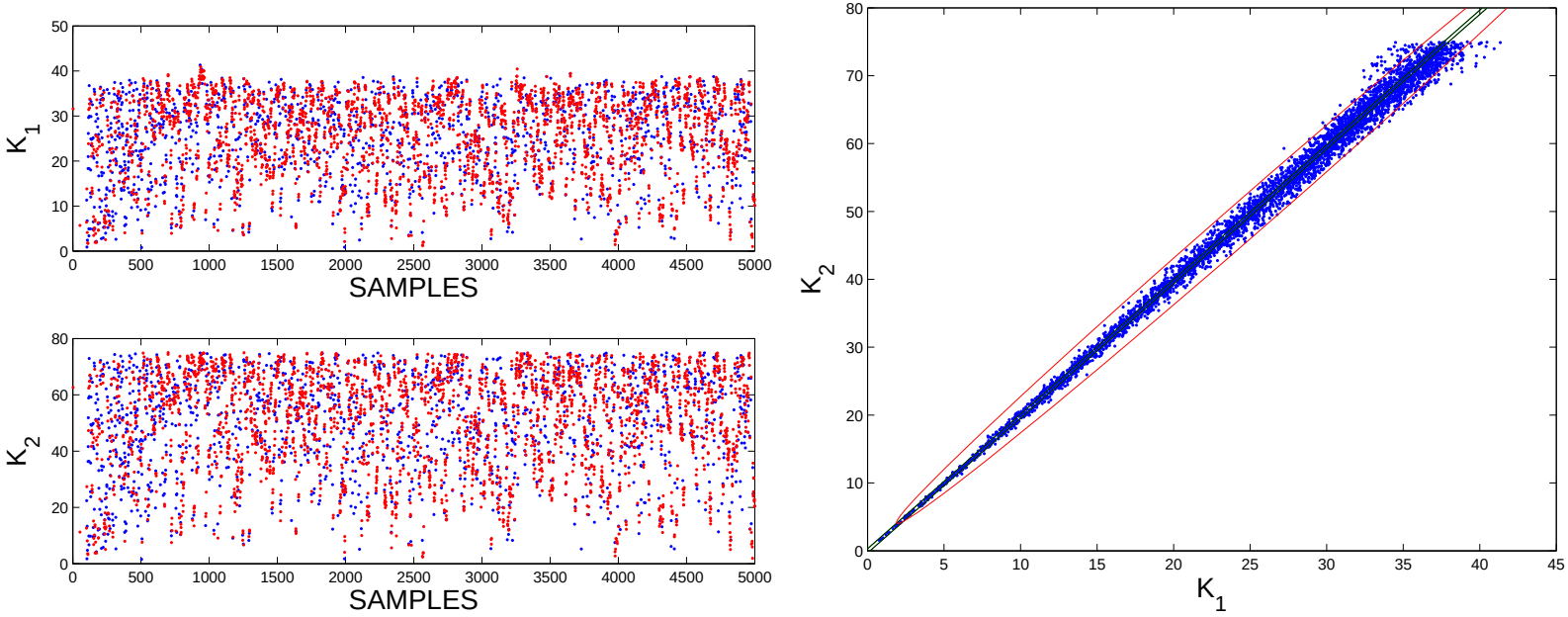


- AM: slow to start adaptation, since very few points from which to adapt.





Solution by DRAM



5.5 Gibbs algorithm

In high dimensional problems, especially, it may be difficult to find a good proposal distribution q. The idea of the Gibbs algorithm is to reduce the sampling to one dimensional distributions. The parameter vector $\theta_1, \theta_2, ..., \theta_p$ is updating in *sweeps*, by updating *one coordinate at a time*. This may be done, if the 1–dimensional or *conditional distributions* $\pi(\theta_i|\theta_1, ..., \theta_{i-1}, \theta_{i+1}, ..., \theta_p)$ are known: the distribution of any of the parameters is known if the values of the rest of the components of θ are fixed.

Often – in nonlinear problems – the conditional distributions are not known, and they must be approximated by sampling in the 1D directions 'sufficiently' many values.

Gibbs method as a pseudocode, that creates a chain of length $Nchain$:

- for k=1,...,Nchain
  - for i=1,2,...,p
  - sample θ_i^k from the distribution $\pi(\theta_i|\theta_1, ..., \theta_{i-1}, \theta_{i+1}, ..., \theta_p)$

Note that the point taken from the 1D distribution is *always accepted*, but the creation of the 1D (approximative) distribution may require several evaluations of the objective function.

If the 1D distribution for θ_k is not known, it must be approximatively created.

This may be done by evaluating $\pi(\theta)$ with respect to the coordinate i a given number of times. The computed values are used to create an *empirical distribution function*.

The new value for θ_k is then sampled from the empirical distribution by using the *inverse CDF* method: sample a random point uniformly on $[0, 1]$ and compute the inverse of the above empirical CDF at that point.

5.6 Single Component Metropolis

A 1D sampling may also be achieved by a single component Metropolis-Hastings (SCMH) algorithm. The proposal distribution of each component is, e.g., a normal distribution with the present point as the center point and with a given variance, separate for each coordinate. The coordinates are updated in loops, similarly as in Gibb's sampling.

Let again π denote the density of our target distribution in a d dimensional Euclidean space, typically a posterior density distribution which we can evaluate up to a normalizing constant. The sequence $\theta_0, \theta_1, \ldots$ denotes the full states of the process, that is, we consider a new state updated as soon as all the d coordinates (or components) of the state have been separately updated. When deciding the i:th coordinate θ_t^i ($i = 1, \ldots, d$) of the t:th state θ_t we apply the standard 1-dimensional Metropolis step:

1. Sample z^i from 1-dimensional normally distributed proposal distribution $q_t^i \sim N(\theta_{t-1}^i, v^i)$ centered at previous point with variance v^i.

2. Accept the candidate point z^i with probability

$$\min\left(1, \frac{\pi(\theta_t^1, \ldots, \theta_t^{i-1}, z^i, \theta_{t-1}^{i+1}, \ldots, \theta_{t-1}^d)}{\pi(\theta_t^1, \ldots, \theta_t^{i-1}, \theta_{t-1}^i, \theta_{t-1}^{i+1}, \ldots, \theta_{t-1}^d)}\right),$$

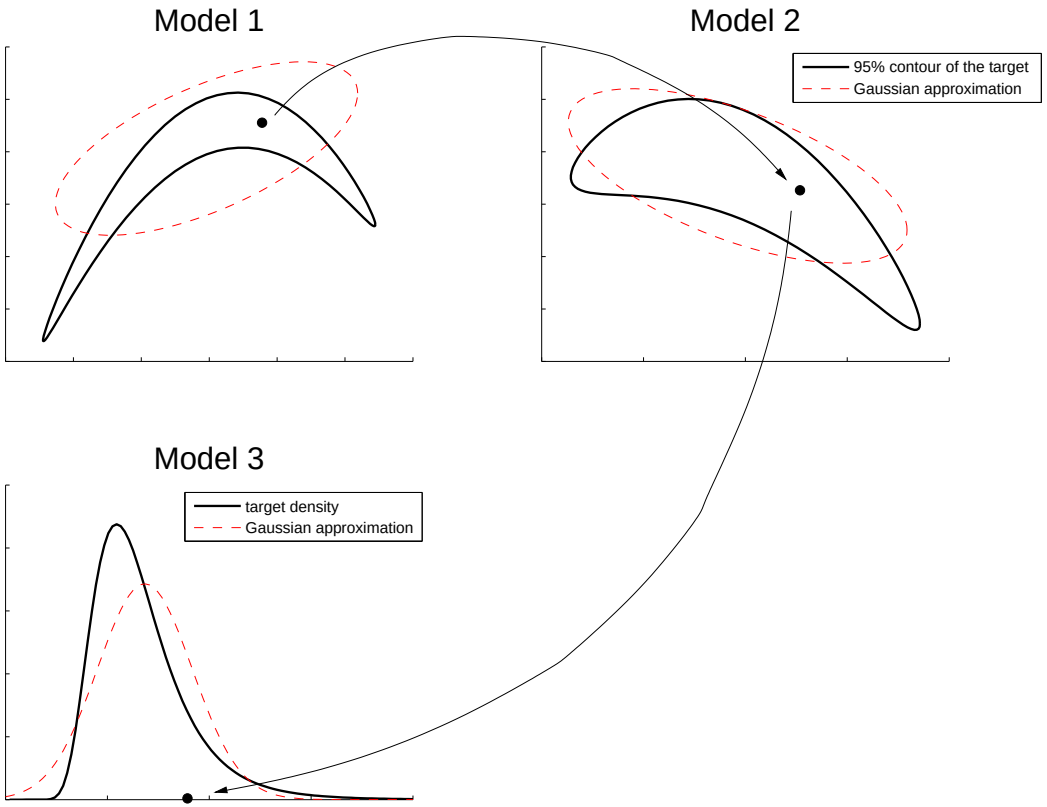
   in which case set $\theta_t^i = z^i$, and otherwise $\theta_t^i = \theta_{t-1}^i$.

Note that, after a full loop over the coordinates, acceptance of a changed value for θ_t typically is more likely than in standard Metropolis - since each coordinate may separately be accepted with a reasonable high probability. On the other hand, the CPU time needed for each coordinate loop increases with the dimension d. For certain high dimensional targets the function evaluation may be factorized, and so it will much quicker if only one coordinate is changed at time.

AARJ – automatic adaptive reversible jump

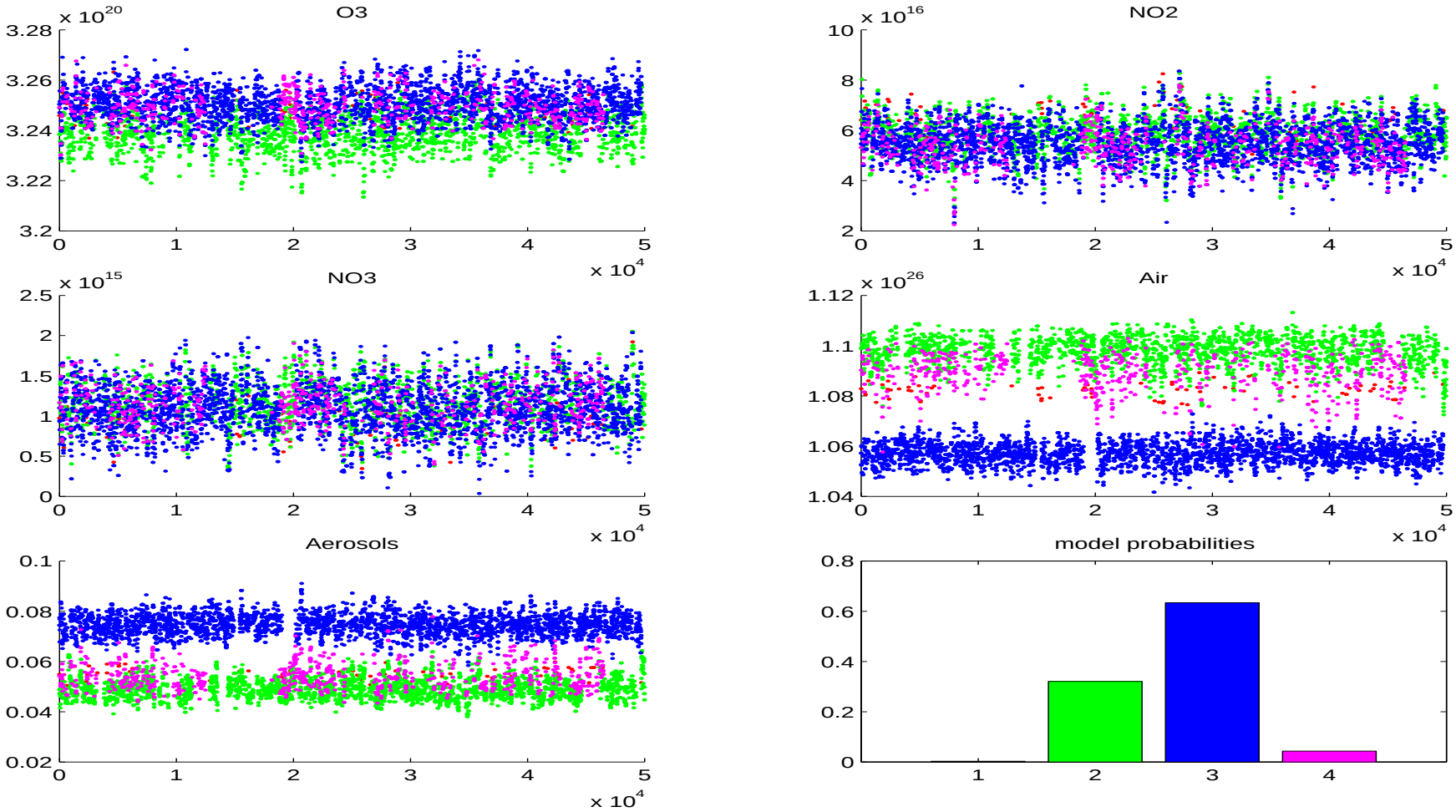
Model selection using adaptive Reversible Jump MCMC.

- AARJ = Automatic RJMCMC by Green (2003) with DRAM style adaptation.
- Uses Gaussian approximations of the target distributions to perform model to model parameter transformations.
- Both the target approximations and proposal covariances are adapted.
- First approximations by initial DRAM runs.



Suitable for model selection and model averaging problems with 2-10 competing models.

RJMCMC chains and model probabilities Four different aerosol models fitted with AARJ.



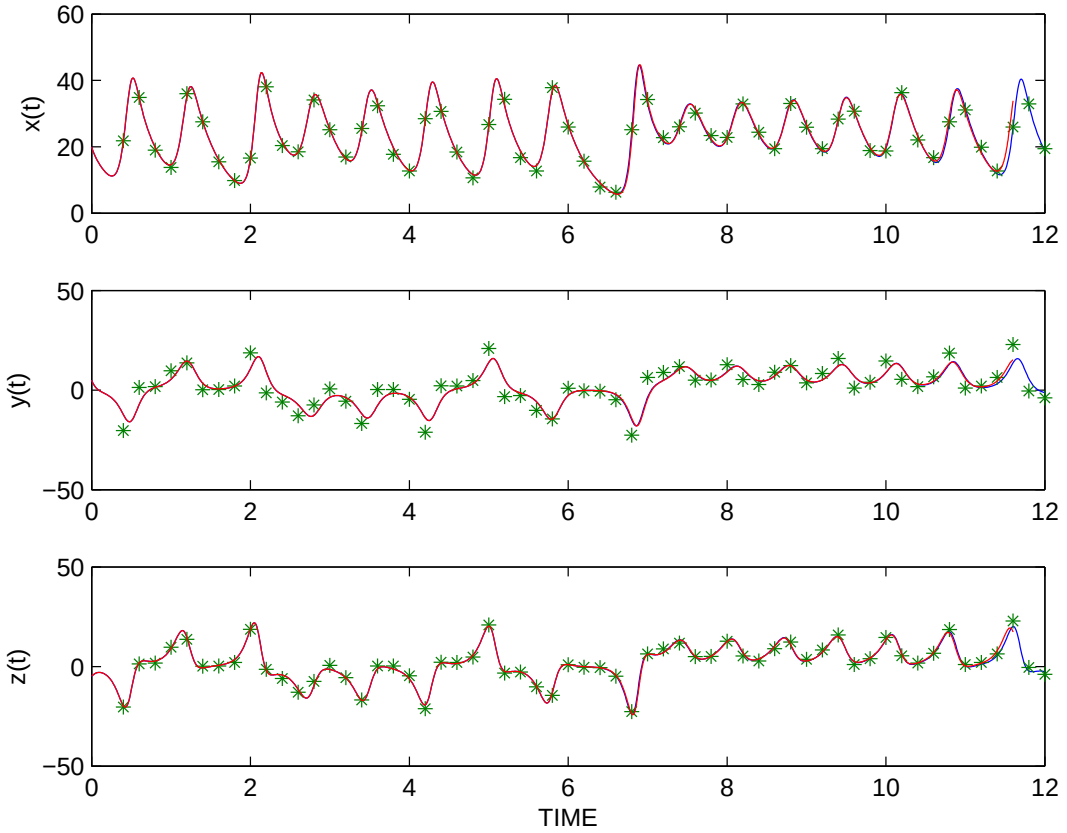
Reducing CPU: Early Rejection

Acceptance in Metropolis: Calculate likelihood ratio α, generate $r \sim U(0,1)$, reject if $r > \alpha$.

But if (as usually) the likelihood ratio α_n is *monotonic* with respect to the number n of calculated data points, just reverse the order: generate $r \sim U(0,1)$, monitor α_n, and reject as soon as $r > \alpha_n$.

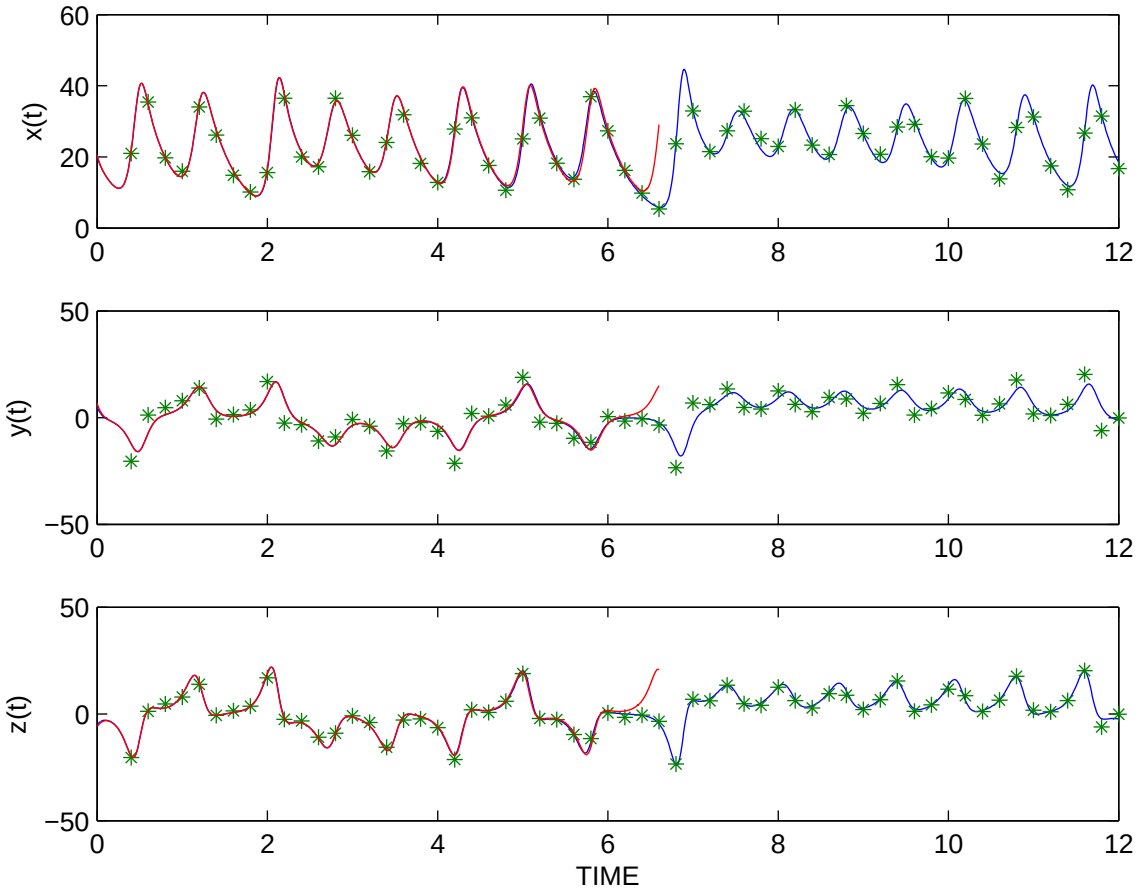
MCMC for Lorenz, Solution by ER

Sampling Lorenz model coefficients:



MCMC for Lorenz, Solution by ER

Sampling Lorenz model initial values:



Reducing CPU: Delayed Acceptance

Suppose a cheap approximative (coarse grid, etc) version π^* is available.
Evaluate it first, and the π only after passing the approximative acceptance:

- At θ_n, choose θ from a proposal distribution $q(.|\theta_n)$, and **Promote** it to $\hat{\theta}$ with probability

$$\min\left(1, \frac{\pi^*(\theta)}{\pi^*(\theta_n)}\right).$$

  otherwise set $\hat{\theta} = \theta_n$.

- **Accept** $\hat{\theta}$ with probability

$$\alpha(\theta_n, \hat{\theta}) = \min\left(1, \frac{\pi(\hat{\theta})}{\pi(\theta_n)}\right).$$

  (note that $\alpha = 1$ if not promoted)

5.7 Further topics

Importance sampling

Suppose we want to estimate the expectance of a function f with respect to a distribution given by the density function p:

$$E_p(f) = \int f(x)p(x)dx$$

This may be done by sampling points x_j and computing a numerical approximation for the integral. In crude Monte Carlo integration the samples are drawn from a uniform distribution. Then every point x_j is considered 'equally important' for the evaluation of the integral. By CLT we know that the average converges, but rather slowly, with the rate $1/\sqrt{n}$. To accelerate the convergence, the sampling should concentrate to 'important' regions of the target function, i.e., to those values where the functions f and p are large.

A better way to compute the integral would be to sample x_j from the distribution given by p and compute then the averages

$$\bar{f}_m = \frac{1}{m}\sum_{j=1}^{m} f(x_j)$$

Suppose we can not (or it is 'expensive') sample directly from p, but we do know a density function g such that $g(x) > 0$ if $p(x) > 0$, and sampling from the density g is easier. By the identity

$$E_f(p) = \int f(x)\frac{p(x)}{g(x)}g(x)dx$$

(where we take the integrand as zero if both p and g vanish) we may sample from g and approximate the expectance as

$$\bar{f}_m = \frac{1}{m}\sum_{j=1}^{m} f(x_j)\frac{p(x_j)}{g(x_j)} = \frac{1}{m}\sum_{j=1}^{m} f(x_j)w(x_j)$$

The function g is referred to as the importance function, w as the importance weight. The function g should be chosen so that it mimics the target distribution, and is easy to sample from. In that case, it may speed up the convergence of the sampling.

Naturally, the main problem here is how to find a proper importance function for each problem.

Population methods

So far the MC methods have been based on sampling *one parameter vector* at each step. But we might equally well sample *several* vectors at each step, either from the same proposal or from a set of different proposals.

The idea as called *Population Monte Carlo*, and may be implemented in various ways.

5.8 The Kalman Filter

Linear LSQ problem with general covariance and prior

Let us return to the Bayes formula

$$\pi(\theta) \simeq p(y|\theta)p(\theta) \tag{5}$$

where $p(y|\theta)$ is the likelihood function $p(\theta)$ is the prior distribution (we ignore the normalizing constant). Consider a linear model $y = Xb + \epsilon$ (using the notation $\theta = b$, as usual with linear models) and suppose that both the measurement error and the prior distribution are Gaussian with covariance matrixes S_ϵ and S_a, respectively. If the center point of the prior is denoted by b_a, we have

$$p(b) \simeq e - \frac{1}{2}(b - b_a)' S_a^{-1}(b - b_a)$$

and

$$p(y|b) \simeq e^{-\frac{1}{2}(y-Xb)' S_\epsilon^{-1}(y-Xb)}.$$

Thus we get (ignoring the normalizing constants again)

$$-2\log\pi(b) = (y - Xb)'S_\epsilon^{-1}(y - Xb) + (b - b_a)'S_a^{-1}(b - b_a)$$

Earlier, we have solved the linear LSQ problem with no priors and with a diagonal covariance. But the above expression can be reduced back to that simple form. Suppose that the inverses of the covariance matrixes may be decomposed by (for example) the Cholesky decomposition:

$$S_\epsilon^{-1} = K_\epsilon' K_\epsilon, \quad S_a^{-1} = K_a' K_a.$$

These substitutions yield

$$\begin{aligned} -2\log\pi(b) &= (K_\epsilon y - K_\epsilon X b)'(K_\epsilon y - K_e X b) + (K_a b - K_a b_a)'(K_a b - K_a b_a) \\ &= ||K_\epsilon X b - K_\epsilon y||^2 + ||K_a b - K_a b_a||^2. \end{aligned}$$

But the last expression is the norm of the LSQ problem for $\tilde{X}b = \tilde{y}$, where

$$\tilde{X} = \begin{pmatrix} K_\epsilon X \\ K_a \end{pmatrix}, \quad \tilde{y} = \begin{pmatrix} K_\epsilon y \\ K_a b_a \end{pmatrix}$$

with $N(0, I)$ as the covariance. So we may utilize the known formulas, $\hat{b} = (\tilde{X}'\tilde{X})^{-1}\tilde{X}'\tilde{y}$ and $S = \mathrm{cov}\hat{b} = (\tilde{X}'\tilde{X})^{-1}$, for the solution and the covariance. It remains to calculate the expressions:

$$\tilde{X}'\tilde{X} = X'K_\epsilon' K_e X + K_a' K_a = X' S_\epsilon^{-1} X + S_a^{-1},$$

$$\tilde{X}'\tilde{y} = X'K_\epsilon' K_\epsilon y + K_a' K_a b_a = X' S_\epsilon^{-1} y + S_a^{-1} b_a.$$

So, for the LSQ solution and the covariance we have the expressions

$$\hat{b} = (X'S_\epsilon^{-1}X + S_a^{-1})^{-1}(X'S_\epsilon^{-1}y + S_a^{-1}b_a), \quad (6)$$

$$S = (X'S_\epsilon^{-1}X + S_a^{-1})^{-1}. \quad (7)$$

In other words, the posterior distribution is Gaussian with center point $\hat{b}$ and covariance S:

$$-2\log\pi(b) = (b - \hat{b})'S^{-1}(b - \hat{b}) + c,$$

where c is a constant.

Linear Kalman Filter

Consider a time-dependent process, the state vector x_t of which is observed at time points t. Suppose further that the time evolution and the observation are given by the expressions

$$\begin{aligned} x_t &= M_t x_{t-1} + E_t \\ y_t &= K_t x_t + \epsilon_t. \end{aligned}$$

Here the matrix M_t gives the model for the state evolution, the observation is separately given as a function of x_t, by a matrix K_t. Note the *two* error terms: ϵ_t gives the 'usual' observation error at time t, while E_t denotes the *modelling error*. So we assume that the model, too, may contain error. In fact, by the choice of E_t and ϵ_t we can specify how much we trust the model or data.

In the basic form of Kalman filtering, we are interested in predicting new values of x, and correcting the state values by new measured observations y. This takes place iteratively, as the observations form a time series y_t for $t = 1, 2, 3, ...$

The process may be cast in the form of successive applications of the Bayes formula. The prior is given by the model prediction, the likelihood by the observation equation. In more detail, suppose the estimate $\hat{x}_{t-1}$ is obtained for the state of the previous time point, and the covariance matrix $\hat{S}_{t-1}$ has been estimated. Suppose further, that the errors are Gaussian, with covariance matrixes S_{Et} and $S_{\epsilon t}$.

The model equation then gives the prediction, that is used as the center point of the prior distribution,

$$x_{at} = M_t \hat{x}_{t-1}.$$

The covariance of the prior can then be computed (assuming that x_t and E_t are statistically independent):

$$S_{at} = \mathrm{cov}(M_t \hat{x}_{t-1} + E_t) = M_t \hat{S}_{t-1} M_t' + S_{Et}.$$

We now have all that is needed to employ the formulas of the general LSQ problem: a given Gaussian prior distribution and a given linear equation system for the observations. By substituting the present notations (M in place of X, x_t for b, x_{at} for b_a) in the equations (5) and (6) above, we have the solution.

However, for computational reasons the Kalman filter usually is written in the following form - (exercise: they may be obtained via direct but somewhat non-trivial matrix manipulations)

$$\begin{aligned} G_t &= S_{at}K_t'(K_tS_{at}K_t' + S_\epsilon)^{-1} \\ \hat{x}_t &= x_{at} + G_t(y_t - K_t x_{at}) \\ \hat{S}_t &= S_{at} - G_t K_t S_{at} \end{aligned}$$

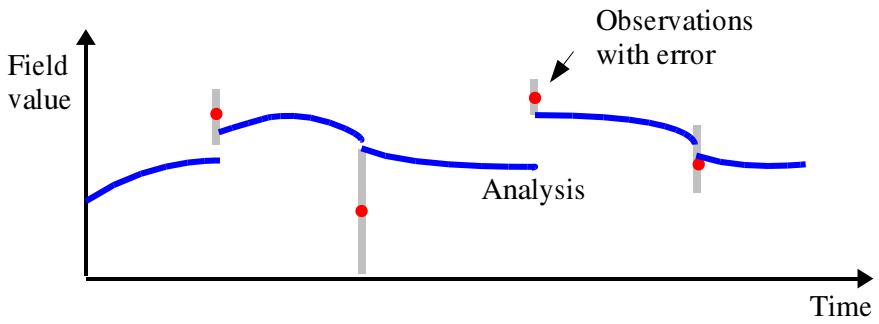
Here the matrix G_t is called the *Kalman gain*.

Note that in Kalman filtering we do not (typically!) estimate any parameters of the model, but the state vector itself. The uncertainty of the model is taken into account with the modelling error terms.

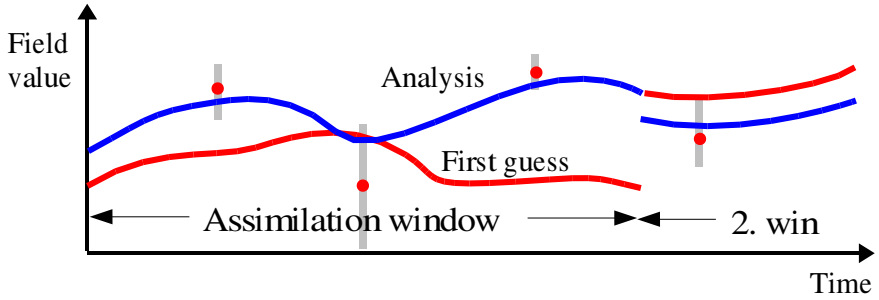
Assimilation: Extended Kalman Filter, Variational methods

Assimilation: calibrate time–series type data to a model.

- Kalman filter: update states of a linear model via the Bayes formula. The model prediction from the previous states used as a priori
- EKF, Extended Kalman Filter: model *nonlinear*, covariance matrixes by approximations. Computational problems with high dimensions.
- Variational methods (for example, 4D-VAR): previous state variables iterated to fit the new measurements. Computationally cheaper, routinely used, e.g., for weather forecasts. Requires tedious coding for the adjoint equations, no model error can be assumed.



EKF



4D-VAR

Kalman filter for nonlinear, high dimensional problems is subject to ongoing research work.