BM20A3001 Statistical Analysis in Modelling

Heikki Haario, Antti Solonen

Course Organization

- Lectures 3h/week: theory and demonstrations.
- Computer lab sessions 2h/week: practical work.
- MATLAB is used throughout the course (make sure you know the basics!).
- Several homeworks that need to be returned in paper.
  - Homeworks are graded (0-3 pts each).
  - 20% of the points are required to pass the course.
  - Extra points awarded to the first exam based on the homeworks.
- Lab sessions are reserved for going through the homeworks and preparing for the next ones.
- All homeworks and material will be posted on the NOPPA page.
- Lecture notes also available at the AALEF bookstore.

Homeworks

- Return preferably on paper in person (at lectures or lab sessions). In extreme cases, return by email (as PDF) to `heikki.haario@lut.fi` or the exercise assistants `gasper.mwanga@lut.fi` and `isambi.mbawalata@lut.fi`
- Write your documents preferably in `LaTex` (Word or something else is OK too).
- Points are awarded based on the homeworks according to the table below:

% of points	Extra pts
20-29	0 (obligatory)
30-39	1
40-49	2
50-59	3
60-69	4
70-79	5
80-89	6
90-100	7

Note: exam will be 5 tasks, each worth 6 points.

Course Outline

1. **Intro:** mathematical models and their uncertainties
2. **Prequisities:** some statistics, methods for generating random numbers
3. **Linear models** and their statistical analysis, design of experiments
4. **Nonlinear models**, approximative error analysis
5. **Classical Monte Carlo methods:** bootstrapping
6. **Bayesian analysis:** MCMC, the Metropolis algorithm and variants
7. **Dynamical state estimation:** Kalman filtering

Mathematical Models

Mathematical modeling is a central tool in most fields of science and engineering. Mathematical models can be either

- **Mechanistic:** based on principles of natural sciences. Also known as 'physiochemical' or 'physics-based' or 'hard' models.
- **Empirical:** inferring relationships between variables directly from the available data. Also known as 'data-driven' or 'soft' models.

Empirical models try to learn the relationship between input variables $\mathbf{X}$ and output variables $\mathbf{Y}$ using the empirical data alone.

$$\mathbf{X} = \begin{array}{c} \begin{matrix} \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_p \end{matrix} \\ \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{pmatrix} \end{array} \quad \mathbf{Y} = \begin{array}{c} \begin{matrix} \mathbf{y}_1 & \mathbf{y}_2 & \dots & \mathbf{y}_q \end{matrix} \\ \begin{pmatrix} y_{11} & y_{12} & \dots & y_{1q} \\ y_{21} & y_{22} & \dots & y_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \dots & y_{nq} \end{pmatrix} \end{array}.$$

Empirical Models

+ Often result in easier computation.
- Possibly problems in interpreting the modeling results.
- Usually poorly extendable to new situations from which measurements are not available (extrapolation).
+ Often the preferred choice, if a 'local' description (interpolation) is enough or if the mechanism is not known or too complicated to be modeled in detail.

The methodology used in analyzing empirical model is often called *regression analysis*.

Mechanistic Models

Built on first principles of natural sciences, and are often formulated as differential equations. For instance, the elementary chemical reaction $A \rightarrow B \rightarrow C$ can be modeled as an ODE system

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

- Model building is often more demanding as with empirical models.
- Require knowledges of numerical methods.
+ Often extends well to new situations.
+ Chosen if the phenomenon is understood well enough and if the model is needed outside the experimental region.

Combining Models and Data

- Mathematical models must be verified against measured data.
- The models usually contain some unknown *parameters* that need to be calibrated from the measurements.
- In this course, we will use the following notation:

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

  where $\mathbf{y}$ are the obtained measurements and $f(\mathbf{x}, \theta)$ is the mathematical model, $\mathbf{x}$ are the control (input) varibles, θ the unknown parameters and ε the measurement error.
- The statistical analysis of the model happens at this 'model fitting' stage: uncertainty in the data $\mathbf{y}$ implies uncertainty in the parameter values θ.

Linear vs. Nonlinear Models

- Mathematical models are either linear or nonlinear.
- Here, we mean linearity with respect to the parameters θ.
- That is, for instance $y = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2$ is linear, whereas $y = \theta_1 \exp(\theta_2 x)$ is nonlinear.
- Most of the classical theory is for linear models
- Direct formulas can often be developed for linear models, which results in easier computation.

Parameter Estimation

- Traditionally, point estimates for the parameters are obtained by solving a least squares (LSQ) optimization problem.
- In LSQ, we search for parameter values $\hat{\theta}$ that minimize the sum of squared differences between the observations and the model:

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

- The LSQ method does not say anything about the uncertainty of the estimator, which this is the purpose of statistical analysis in modeling.
- The theory for statistical analysis of linear models is well established, and often used as an approximation for nonlinear models as well.
- Efficient numerical Monte Carlo techniques have been introduced lately to allow full statistical analysis for nonlinear models. This is the main topic of this course.

Prequisities: statistical measures

- The expected value for a random vector $\mathbf{x}$ with PDF $p(\mathbf{x})$ is

$$\mathrm{E}(\mathbf{x}) = \int \mathbf{x} p(\mathbf{x}) dx.$$

- The variance of a one-dimensional random variable is

$$\mathrm{Var}(x) = \int (x - \mathrm{E}(x))^2 p(x) dx.$$

- The standard deviation is $\mathrm{Std}(x) = \sqrt{\mathrm{Var}(x)}$.
- The *covariance* of two random variables x and y measures how much the two variables change together:

$$\mathrm{Cov}(x, y) = \mathrm{E}((x - \mathrm{E}(x))(y - E(y))).$$

Prequisities: statistical measures

- The *covariance matrix* of two random vectors $\mathbf{x}$ and $\mathbf{y}$, with dimensions n and m, gives the covariances of different combinations of elements in the $\mathbf{x}$ and $\mathbf{y}$ vectors. The covariances are collected into an $n \times m$ covariance matrix, defined as

  $$\mathrm{Cov}(\mathbf{x}, \mathbf{y}) = \mathrm{E}((\mathbf{x} - \mathrm{E}(\mathbf{x}))(\mathbf{y} - E(\mathbf{y}))^T).$$

  In this course, the covariance matrix is used to measure the covariances within the elements of a single random vector $\mathbf{x}$, and we denote the covariance matrix by $\mathrm{Cov}(\mathbf{x}) = \mathrm{Cov}(\mathbf{x}, \mathbf{x})$.

- The correlation coefficient between two random variables x and y is the normalized version of the covariance:

  $$\mathrm{Cor}(x, y) = \frac{\mathrm{Cov}(x, y)}{\mathrm{Std}(x)\mathrm{Std}(y)}.$$

  For vector quantities, one can form a *correlation matrix* (similarly as in the covariance matrix).

Prequisities: sample statistics

To be able to use the statistical measures defined above, we need ways to estimate the statistics using measured data. Here, we give the *sample statistics*, which are estimators of the above measures.

Let us consider a $n \times p$ matrix of data, where each column contains n measurements for variables $\mathbf{x}_1, ..., \mathbf{x}_p$:

$$\mathbf{X} = \begin{array}{c} \begin{array}{cccc} \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_p \end{array} \\ \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{pmatrix} \end{array}.$$

Prequisities: sample statistics

- the sample mean for the kth variable: $\hat{x}_k = \frac{1}{n}\sum_{i=1}^{n} x_{ik}$
- the sample variance of the kth variable:

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

- the sample STD: $\mathrm{Std}(x_k) = \sqrt{(\mathrm{Var}(x_k))} = \sigma_k$
- the sample covariance between two variables:

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

- the sample correlation coefficient between two variables:

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

Prequisities: sample statistics

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

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

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

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

Useful MATLAB commands: `mean`, `var`, `std`, `cov`, `corr`, `cov2cor`.

Prequisities: some distributions

- The PDF of the univariate normal (or Gaussian) distribution with mean x_0 and variance σ^2 is

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2} \left(\frac{x - x_0}{\sigma} \right)^2 \right).$$

  For a random variable x that follows the normal distribution, we write $x \sim \mathrm{N}(x_0, \sigma^2)$.

- The sum of n univariate Gaussian random variables, $s = \sum_{i=1}^{n} x_i^2$, follows the *chi-square* distribution with n degrees of freedom, which we denote by $s \sim \chi_n^2$.

- The PDF of the d-dimensional Gaussian distribution with mean vector $\mathbf{x}_0$ and covariance matrix Σ is

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \Sigma^{-1} (\mathbf{x} - \mathbf{x}_0) \right),$$

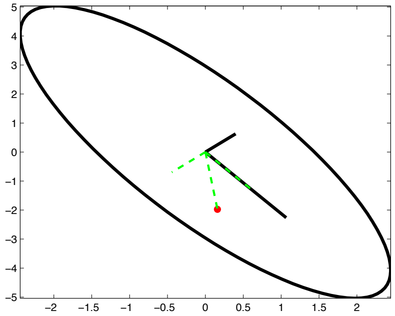
  where $|\Sigma|$ denotes the determinant of Σ.

Visualizing Gaussian Densities as Ellipses

- The contours of multivariate Gaussian densities are ellipsoids, where the principal axes are given by the eigenvectors of the covariance matrix, and the eigenvalues correspond to the lengths of the axes.
- Therefore, two-dimensional covariance matrices can be visualized as ellipses.
- In practice, this can be done with the `ellipse` function given in the code package.
- Check out the demo program `ellipse_demo.m`.

Gaussian Random Numbers: Geometric Interpretation

- Assume that the covariance matrix can be written as a sum of outer products of n vectors, $\Sigma = \sum_{i=1}^{n} \mathbf{a}_i \mathbf{a}_i^T$.
- Then, the sum $\mathbf{y} = \sum_{i=1}^{n} \omega_i \mathbf{a}_i$, where $\omega_i \sim N(0,1)$, follows the Gaussian distribution with covariance matrix Σ.
- That is, Gaussian random variables are randomly weighted combinations of 'basis vectors' $(\mathbf{a}_1, ..., \mathbf{a}_n)$.



Generating Gaussian Random Numbers

- Recall that in the univariate case, normal random numbers $y \sim N(\mu, \sigma^2)$ are generated by $y = \mu + \sigma z$, where $z \sim N(0, 1)$.
- Let us consider a covariance matrix Σ, which we can decompose into a 'square-root' form $\Sigma = \mathbf{A}\mathbf{A}^T$.
- Now, if we take a random variable with $\mathrm{cov}(\mathbf{x}) = \mathbf{I}$, the covariance matrix of the transformed variable $\mathbf{Ax}$ is

$$\mathrm{cov}(\mathbf{Ax}) = \mathbf{A}\mathrm{cov}(\mathbf{x})\mathbf{A}^T = \mathbf{A}\mathbf{A}^T = \Sigma.$$

- This gives us a recipe for generating Gaussian random variables with a given covariance matrix:

  1. Generate standard normal random numbers $\mathbf{x} \sim \mathrm{N}(\mathbf{0}, \mathbf{I})$.
  2. Compute decomposition $\Sigma = \mathbf{A}\mathbf{A}^T$.
  3. Transform the standard normal numbers with $\mathbf{y} = \mathbf{Ax}$.

- Code examples: `chol_demo.m` and `chi2_demo.m`

Prequisities: checking if samples are Gaussian

- Let us consider a zero-mean multivariate Gaussian with covariance matrix Σ.
- The term in the exponential of the PDF can be written as $\mathbf{x}^T\Sigma^{-1}\mathbf{x} = \mathbf{y}^T\mathbf{y}$, where $\mathbf{y} = \Sigma^{-1/2}\mathbf{x}$. Here $\Sigma^{-1/2}$ denotes the 'square root' (e.g. Cholesky decomposition) of Σ^{-1}.
- Using the formula $\mathrm{Cov}(\mathbf{A}\mathbf{x}) = \mathbf{A}\mathrm{Cov}(\mathbf{x})\mathbf{A}^T$, the covariance of the transformed variable $\mathbf{y}$ can be written as

$$\mathrm{Cov}(\Sigma^{-1/2}\mathbf{x}) = \Sigma^{-1/2}\mathrm{Cov}(\mathbf{x})(\Sigma^{-1/2})^T = \Sigma^{-1/2}\Sigma(\Sigma^{-1/2})^T = \mathbf{I}.$$

- That is, the term $\mathbf{x}^T\Sigma^{-1}\mathbf{x} = \mathbf{y}^T\mathbf{y}$ is a squared sum of d normal variables, and therefore $\mathbf{x}^T\Sigma^{-1}\mathbf{x} \sim \chi^2_d$.
- One can check how well sampled vectors follow a Gaussian distribution: compute the values $\mathbf{x}^T\Sigma^{-1}\mathbf{x}$ and see how well they follow the χ^2_d distribution (see exercises).

General Methods: Inverse CDF

- Let us consider a continuous random variable whose cumulative distribution function (CDF) is F.
- The inverse CDF method is based on the fact that a random variable $x = F^{-1}(u)$ where u is sampled from $U[0, 1]$, has the distribution with CDF F.

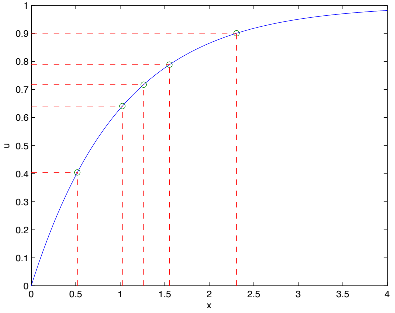


Figure: Producing samples from the exponential distribution using inverse CDF, $F(x) = 1 - \exp(-x)$ and $F^{-1}(u) = -\ln(1-u)$.

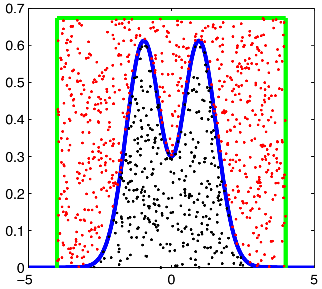
General Methods: Accept-Reject

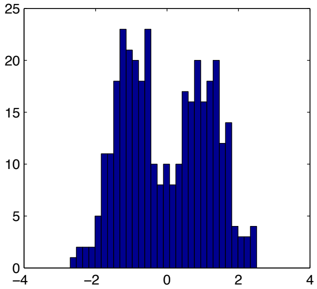
- Suppose that f is a positive (but non-normalized) function on the interval $[a, b]$, bounded by M.
- Consider uniform random points (x_i, u_i) in the 'box' $[a, b] \times [0, M]$
- The points that satisfy $u_i < f(x_i)$ form a uniform sample under the graph of f.
- The area of any slice $\{(x, y)|x_l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

General Methods: Accept-Reject

A (very) straightforward algorithm:

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





Linear Models

- Let us consider a linear model
  $f(\mathbf{x}, \theta) = \theta_0 + \theta_1 \mathbf{x}_1 + \theta_2 \mathbf{x}_2 + \ldots + \theta_p \mathbf{x}_p$.
- Assume that we have noisy measurements $\mathbf{y} = (y_1, y_2, ..., y_n)$ obtained at points $\mathbf{x}_i = (x_{1i}, x_{2i}, ..., x_{ni})$ where $i = 1, ..., p$.
- Now, we can write the model in matrix notation:

$$\mathbf{y} = \mathbf{X}\theta + \varepsilon,$$

  where $\mathbf{X}$ is the design matrix that contains the measured values for the control variables, augmented with a column of ones to account for the intercept term θ_0:

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1p} \\ 1 & x_{21} & x_{22} & \ldots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \ldots & x_{np} \end{pmatrix}.$$

Linear Models

- For linear models, we can derive a direct formula for the LSQ estimator.
- The LSQ estimate, that minimizes $SS(\theta) = ||\mathbf{y} - \mathbf{X}\theta||_2^2$, is obtained as the solution to the *normal equations* $\mathbf{X}^T\mathbf{X}\theta = \mathbf{X}^T\mathbf{y}$:

$$\hat{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

- To obtain the statistics, we can compute the covariance matrix $\mathrm{Cov}(\hat{\theta})$.
- Assume i.i.d. measurement error: $\mathrm{Cov}(\mathbf{y}) = \sigma^2\mathbf{I}$.
- Then, we can show (exercise) that

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

Linear Models

- Let us further assume that the measurement errors are Gaussian.
- Then, we can also conclude that the distribution of $\hat{\theta}$ is Gaussian, since $\hat{\theta}$ is simply a linear transformation of a Gaussian random variable $\mathbf{y}$.
- That is, the unknown parameter follows the normal distribution with mean and covariance matrix given by the above formulae: $\theta \sim \mathrm{N}(\hat{\theta}, \sigma^2(\mathbf{X}^T\mathbf{X})^{-1})$.
- The probability density of the unknown parameter can be written as

$$p(\theta) = K \exp\left(-\frac{1}{2}(\theta - \hat{\theta})^T \mathbf{C}^{-1}(\theta - \hat{\theta})\right),$$

where $K = ((2\pi)^{d/2}|\Sigma|^{1/2})^{-1}$ is the normalization constant and $\mathbf{C} = \mathrm{Cov}(\hat{\theta})$.

Analyzing Modeling Results

- In regression models, usually not all possible terms are needed to obtain a good fit between the model and the data $\rightarrow$ *model selection* problem.
- It can be dangerous to employ a 'too fine' model, too complex models often result in 'over-fitting'.
- **A. Einstein:** *"a model should be as simple as possible, but not simpler than that"*.
- A natural criterion for selecting terms in a regression model is to drop terms that are poorly known (not well identified by the data).

Analyzing Modeling Results: t-values

- When we have fitted a linear model, we can compute the covariance matrix of the unknown parameters.
- Then, we can compute the ’signal-to-noise’ ratios for the parameters in the model, also known as *t-values*.
- For parameter θ_i, the t-value is

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

  where the standard deviations of the estimates can be read from the diagonal of the covariance matrix.
- The higher the t-value is, the smaller the relative uncertainty is in the estimate, and the more sure we are that the term is relevant in the model and should be included.
- We can select terms for which the t-values are clearly separated from zero. As a rule of thumb, we can require that, for instance, $|t_i| > 3$.

Analyzing Modeling Results: the R^2-value

- The t-values do not tell how good the model fits the observations in general.
- A classical way to measure the goodness of the fit is the *coefficient of determination*, or R^2 value, which is written as

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

- Consider the simplest possible model $\mathbf{y} = \theta_0$, that is, the data are modeled by just a constant. In this case, the LSQ estimate is just the empirical mean of the data, $\hat{\theta}_0 = \overline{\mathbf{y}}$ (exercise).
- That is, the R^2 value measures how much better the model fits to the data than a constant model. The better the model fit is, the closer the R^2 value is to 1.

Analyzing Modeling Results: Cross-validation

- The R^2 value does not tell about the predictive power of the model, that is, how well the model is able to generalize to new situations.
- The predictive power of the model can be assessed, for instance, via cross-validation:
  1. Leave out a part of the data from $\mathbf{X}$ and $\mathbf{y}$.
  2. Fit the model using the remaining data.
  3. Using the fitted model, predict the data that were left out.
  4. Repeat steps 1-3 so that each response value in $\mathbf{y}$ is predicted.
- The goodness of the predictions at each iteration can be assessed using the R^2 formula. The obtained number is called the Q^2 value.

Analyzing Modeling Results: Cross-validation

- Cross-validation is commonly used for model selection. The idea is to test different models with different terms included and choose the model that gives the best Q^2 value.
- If the model is either too complex ('over-parameterized') or too crude, the model will predict poorly and give a low Q^2 value.
- In practice, the cross-validation procedure can be carried out by stepwise regression, which can be implemented in many ways, for instance by:
    - *Forward stepping*: test all terms individually, choose the one with highest Q^2 value. Continue by testing all remaining terms, and choose the one with highest Q^2 value at each step.
    - *Backward stepping*: similarly, but starting with the full model, and dropping, one by one, the worst term on each iteration step.

Code example: fitting linear models

Let us consider fitting the model
$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_{11} x_1^2 + \theta_{12} x_1 x_2 + \theta_{22} x_2^2$ the to data given below:

$$\left[\begin{array}{ccccccccccccc} x_1 : & 100.0 & 220.0 & 100.0 & 220.0 & 75.1 & 244.8 & 160.0 & 160.0 & 160.0 & 75.1 & 75.1 \\ x_2 : & 2.0 & 2.0 & 4.0 & 4.0 & 3.0 & 3.0 & 1.5 & 4.4 & 3.0 & 3.0 & 3.0 \\ y : & 25.0 & 14.0 & 6.9 & 5.9 & 14.1 & 9.3 & 18.2 & 5.6 & 9.6 & 14.9 & 14.8 \end{array}\right]$$

The model fitting and analyzing the modeling results is given in the demo program `lin_fit.m`.

Design of Experiments

- So far, we have assumed that the measurements for the input and output variables $\mathbf{X}$ and $\mathbf{y}$ are given.
- The task of Design of Experiments (DOE) is to figure out how to choose $\mathbf{X}$ so that maximal information about θ is obtained with minimal experimental effort.
- In linear models, the covariance matrix of the parameters is $\mathrm{Cov}(\hat{\theta}) = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}$.
- That is, the uncertainty in $\hat{\theta}$ depends on the noise level σ^2 and on the design matrix $\mathbf{X}$ (not on $\hat{\theta}$).
- This suggests that, for linear models, one can derive general, 'case-independent' theory about how to choose $\mathbf{X}$.

DOE: 2^N and CCD design plans

- Different design plans allow the fitting of different kinds of models:
  - 2^N **design** enables the estimation of a model with first order and interaction terms. For instance, the model $y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_{12} x_1 x_2$ can be fitted using a 2^2 design.
  - **Central composite design (CCD)** allows the estimation of a model that contains the quadratic terms. In a two-parameter example, we can fit $y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_{11} x_1^2 + \theta_{12} x_1 x_2 + \theta_{22} x_2^2$.
- Mathematically, the design plans are typically expressed in *coded units*, where the center point of the design is moved to the origin and the minimum and maximum are scaled to the values ± 1.

DOE: 2^N and CCD design plans

- If $\overline{x}_i$ is the mean value of factor i and Δ_i is the difference between the maximum and minimum values of the experimental region, the transformation from original units x_i to coded units X_i is given by

$$X_i = \frac{x_i - \overline{x}_i}{\Delta_i/2}.$$

- Coded units give a generic way to present various design plans. In addition, scaling all variables to the same unit interval often results in more numerically stable computations.
- The 2^N design puts measurements in the corners of the experimental region, containing all combinations of levels ±1.
- The CCD design extends the 2^N plan by adding 'One Variable at a Time' (OVAT) measurements, where only the value of one variable is changed at a time while the others are fixed to the center point value.

DOE: 2^N and CCD design plans

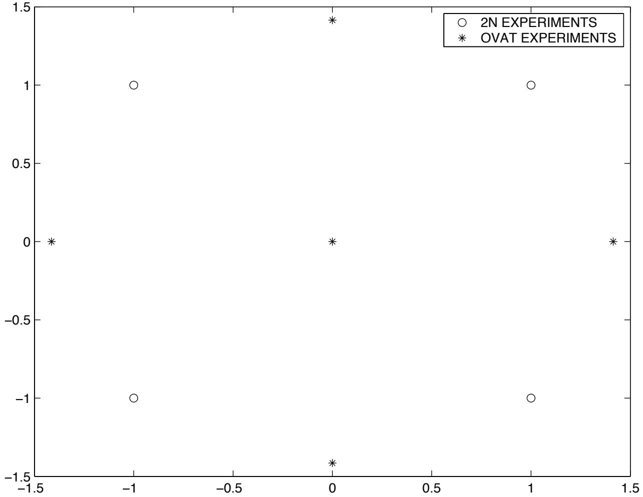


Figure: 2^N and CCD design plans in coded units.

DOE: 2^N and CCD design plans

The design plans for $N = 2$ (without replicated measurements):

$$X_{2^N} = \begin{pmatrix} +1 & -1 \\ +1 & +1 \\ -1 & -1 \\ -1 & +1 \end{pmatrix} \qquad X_{CCD} = \begin{pmatrix} +1 & -1 \\ +1 & +1 \\ -1 & -1 \\ -1 & +1 \\ \sqrt{2} & 0 \\ -\sqrt{2} & 0 \\ 0 & \sqrt{2} \\ 0 & -\sqrt{2} \end{pmatrix}$$

Check the code example `doe_demo.m` for examples of creating designs and moving between original and coded units.

Nonlinear Models

- For nonlinear models, no direct formulas are available, and one has to resort to numerical methods and different approximations.
- Let us consider a nonlinear model $\mathbf{y} = f(\mathbf{x}, \theta) + \varepsilon$
- To compute the LSQ estimate, one has to numerically minimize the sum of squares

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

- For simple models, one can use standard optimization routines in computational software packages. For this course, the MATLAB gradient-free nonlinear simplex optimizer `fminsearch` is enough.
- Next, let us see how approximative error analysis can be performed for the parameters of a nonlinear model.

Nonlinear Models

- The first three terms of the Taylor series expansion for $l(\theta)$ at a point $\hat{\theta}$ can be written as

$$l(\theta) \approx l(\hat{\theta}) + \nabla l(\hat{\theta})^T(\theta - \hat{\theta}) + \frac{1}{2}(\theta - \hat{\theta})^T \mathbf{H}(\theta - \hat{\theta}),$$

  where ∇ denotes the gradient and $\mathbf{H}$ is the Hessian matrix.

- The 2nd derivatives, or elements $[\mathbf{H}]_{pq}$ of the Hessian matrix, are

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

- Assuming that the residuals $f(x_i,\hat{\theta}) - y_i$ are small, the Hessian matrix can be approximated using only first derivatives by dropping the residual terms:

$$[\mathbf{H}]_{pq} = \frac{\partial^2 \ell(\hat{\theta})}{\partial\theta_p \partial\theta_q} \approx 2\sum_{i=1}^{n} \frac{\partial f(x_i,\hat{\theta})}{\partial\theta_p} \frac{\partial f(x_i,\hat{\theta})}{\partial\theta_q}.$$

Nonlinear Models

- The first derivatives can be collected into a Jacobian matrix $\mathbf{J}$, which has elements

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

- Now, the Hessian approximation can be written in a matrix form:

$$\mathbf{H} \approx 2\mathbf{J}^T\mathbf{J}.$$

- Inserting this into the Taylor expansion, and noting that $\nabla l(\hat{\theta}) = \mathbf{0}$, we obtain

$$l(\theta) \approx l(\hat{\theta}) + (\theta - \hat{\theta})^T \mathbf{J}^T \mathbf{J} (\theta - \hat{\theta}).$$

Nonlinear Models

- For linear models, the least squares expression is

  $$l(\theta) = ||\mathbf{y} - \mathbf{X}\theta||^2 = (\mathbf{y} - \mathbf{X}\theta)^T(\mathbf{y} - \mathbf{X}\theta) = \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\theta + \theta^T\mathbf{X}^T\mathbf{X}\theta.$$

- Differentiating the function twice gives the Hessian matrix $\mathbf{H} = \mathbf{X}^T\mathbf{X}$, and the Taylor expansion is

  $$l(\theta) = l(\hat{\theta}) + (\theta - \hat{\theta})^T\mathbf{X}^T\mathbf{X}(\theta - \hat{\theta}).$$

- Now, compare the nonlinear and linear expressions.
- We observe that **the Jacobian matrix J assumes the role of the design matrix X in the linear case**.
- That is, the covariance matrix of $\hat{\theta}$ can be approximated with

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

Estimating σ^2

- The measurement error σ^2 can be estimated using repeated measurements.
- Often, however, replicated measurements are not available.
- In this case, the measurement noise can be estimated using the residuals of the fit, using the assumption that *residuals* $\approx$ *measurement error*.
- An estimate for the measurement error can be obtained using the mean square error (MSE):

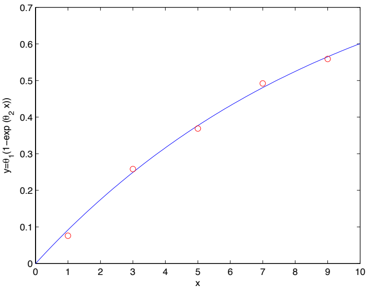
  $$\sigma^2 \approx MSE = RSS/(n-p),$$

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

Code Example: Nonlinear LSQ Fitting and Approximative Error Analysis

- Let us consider estimating the parameters in a model $\mathbf{y} = \theta_1(1 - \exp(-\theta_2\mathbf{x}))$ using the data $\mathbf{x} = (1, 3, 5, 7, 9)$ and $\mathbf{y} = (0.076, 0.258, 0.369, 0.492, 0.559)$.
- We create two files, the main program `bod_fit.m` and the function `bod_ss.m` that computes the sum of squares objective function that is minimized.
- We use the `fminsearch` optimizer, and compute the Jacobian matrix analytically.
- Check the demo program `bod_fit.m`.

Code Example: Nonlinear LSQ Fitting and Approximative Error Analysis



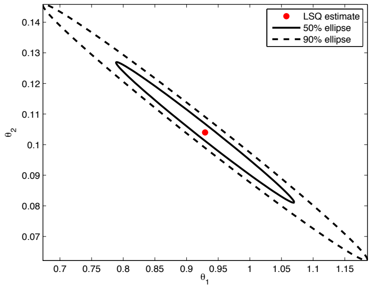


Figure: Left: the model (blue line) fitted to the data (red circles). Right: the LSQ estimates and two confidence ellipses.

Code Example: LSQ for Dynamical Models

- Let us consider the chemical reactions $A \to B \to C$, which can be modeled as an ODE system:

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

- The parameters are the reaction rates, $\theta = (k_1, k_2)$. The data $\mathbf{y}$ consists of the values of A and B:

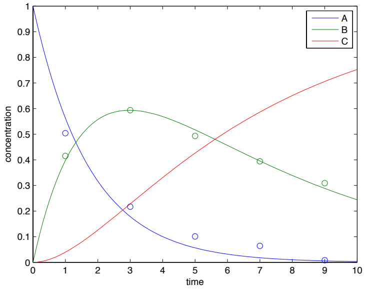
$$\begin{pmatrix} \text{time} & A & B \\ 1.0 & .504 & .415 \\ 3.0 & .217 & .594 \\ 5.0 & .101 & .493 \\ 7.0 & .064 & .394 \\ 9.0 & .008 & .309 \end{pmatrix}.$$

- The initial values for the concentrations are $A(0) = 1$ and $B(0) = C(0) = 0$.

Code Example: LSQ for Dynamical Models

- Now, the above ODE system needs to be solved inside the least squares function.
- This can be done with numerical ODE solvers. In MATBAL, we can use built-in solvers such as `ode45`. The system is solved in the function `ABCmodel.m`.
- The ODE solver needs an *ODE function* that defines the system. Here, the ODE function is given in `ABCode.m`.
- The sum of squares is computed in `ABCss.m` and the main program is `ABCrun.m`.
- Here, the Jacobian must be computed numerically. This is demonstrated in `jacob_demo.m`.

Code Example: LSQ for Dynamical Models



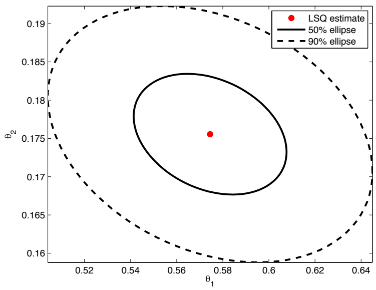


Figure: Left: the model (lines) fitted to the data (circles). Right: the LSQ estimates and two confidence ellipses.

Monte Carlo Methods for Parameter Estimation

- The approximative error analysis for nonlinear models described above can sometimes given misleading results.
- An alternative way to obtain statistics for parameter estimates is to use various Monte Carlo (MC) random sampling methods.
- Before proceeding to Bayesian estimation and MCMC topics, which is the main focus of this course, we briefly present several 'classical' Monte Carlo methods that one can use to evaluate the uncertainty in $\hat{\theta}$.

Adding Noise to Data

- Uncertainty in the model parameters θ in a model

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

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

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

- Often the structure of the noise is not properly known.

- An iterative optimization needs to be performed after every time new data is generated, which can be time consuming.

- The results are dependent on the optimizer settings.

Bootstrapping

- A very popular statistical analysis method, in spirit similar to the above 'adding noise to data' approach.
- No new data is generated, but new random combinations of the existing data are used.
- The basic idea can be written as a pseudo-algorithm as follows:
  1. From the existing data $\mathbf{x} = (x_1, ..., x_n)$, $\mathbf{y} = (y_1, ..., y_n)$, sample new data $\tilde{\mathbf{x}}$, $\tilde{\mathbf{y}}$ with replacement. In practice, select n indices randomly from $1, ..., n$ and choose the data points corresponding to the chosen indices.
  2. Compute the fit using the resampled data $\tilde{\mathbf{x}}$, $\tilde{\mathbf{y}}$.
  3. Go to step 1, until a desired number of θ samples are obtained.
- Suffers from the same problems as the 'adding noise to data' approach: depends on the success of the optimization step and is CPU intensive (repeated calls to an optimization routine).

Bayesian Estimation and MCMC

- In Bayesian parameter estimation, θ is interpreted as a random variable and the goal is to find the *posterior distribution* $\pi(\theta|\mathbf{y})$ of the parameters.
- The posterior distribution gives the probability density for values of θ, given measurements $\mathbf{y}$.
- Using the Bayes' formula, the posterior density is

$$\pi(\theta|\mathbf{y}) = \frac{l(\mathbf{y}|\theta)p(\theta)}{\int l(\mathbf{y}|\theta)p(\theta)d\theta},$$

  where $l(\mathbf{y}|\theta)$ is the *likelihood* and $p(\theta)$ is the *prior* distribution.
- The likelihood gives the probability density of observing $\mathbf{y}$ given the parameter value θ, and the prior contains all existing information about the parameters, such as bound constraints.
- The integral in the denominator is the normalization constant.

Likelihood

- The likelihood function contains the measurement error model.
- Let us consider the model $\mathbf{y} = f(\mathbf{x}, \theta) + \epsilon$ and employ a Gaussian i.i.d. error model, $\epsilon \sim N(0, \sigma^2 \mathbf{I})$
- Noting that $\epsilon = \mathbf{y} - f(\mathbf{x}, \theta)$ gives the likelihood

$$l(\mathbf{y}|\theta) \propto \prod_{i=1}^{n} l(y_i|\theta) \propto \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}[y_i - f(x_i, \theta)]^2\right).$$

- That is, the likelihood contains the familiar sum of squares term $SS(\theta) = \sum_{i=1}^{n}[y_i - f(x_i, \theta)]^2$.

Point Estimates

- Different point estimates can be derived from the posterior distribution:
  - *Maximum a Posteriori* (MAP) estimator maximizes $\pi(\theta|\mathbf{y})$.
  - *Maximum Likelihood* (ML) estimator maximizes $l(\mathbf{y}|\theta)$.
- If the prior distribution is uniform within some bounds, ML and MAP coincide.
- With the Gaussian i.i.d. error assumption, ML coincides also with the classical Least Squares (LSQ) estimate.

Bayesian Computation: MCMC

- In principle, the posterior distribution gives the solution to the parameter estimation problem in a fully probabilistic sense.
- We can find the peak of the probability density, and determine, for instance, the 95% credibility regions for the parameters.
- However, working with the posterior density directly is challenging, since we need to compute the normalization constant $\int l(\mathbf{y}|\theta)p(\theta)d\theta$.
- This often cannot be computed analytically, and classical numerical integration methods also become infeasible, if the number of parameters is larger than a few.
- With the *Markov chain Monte Carlo* (MCMC) methods, statistical inference for the model parameters can be done without explicitly computing this difficult integral.

Bayesian Computation: MCMC

- MCMC methods aim at generating a sequence of random samples $(\theta_1, \theta_2, ..., \theta_N)$, whose distribution asymptotically approaches the posterior distribution as N increases.
- That is, the posterior density is not used directly, but samples from the posterior distribution are produced instead.
- The *Monte Carlo* term is used to describe methods that are based on random number generation.
- The samples are generated so that each new point θ_{i+1} only depends on the previous point θ_i, and the samples therefore form a *Markov Chain*.
- Markov Chain theory can be used to show that the samples approach the correct target (posterior).

The Metropolis Algorithm

- One of the most widely used MCMC algorithms.
- Works by generating candidate parameter values from a *proposal distribution* and then either accepting or rejecting the proposed value according to a simple rule.
- The Metropolis algorithm can be written as follows:
  - 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 step 2.

The Metropolis Algorithm

- One can see that the candidate points that give a higher posterior density value than the previous point (points where $\pi(\hat{\theta}) > \pi(\theta_n)$) are always accepted.
- However, moves 'downward' may also be accepted, with probability given by the ratio of the posterior density values.
- That is, the algorithm randomly jumps around the peak of the target, without converging to a single point.
- In code level, the accept-reject step can be implemented by generating a uniform random number $u \sim U(0,1)$ and accepting if $u \leq \pi(\hat{\theta})/\pi(\theta_i)$.
- Note that we only need to compute ratios of posterior densities, and the normalization constant (nasty integral) cancels out!

The Metropolis Algorithm

- The problem remaining in the implementation of the Metropolis algorithm is defining the proposal distribution q.
- The proposal should be chosen so that it is easy to sample from and 'close' to the underlying target distribution.
- An unsuitable proposal can lead to inefficient implementation:
  - if the proposal is too large, the new candidates mostly miss the essential support π and are only rarely accepted.
  - if the proposal is too small, the new candidates are mostly accepted, but from a small neighborhood of the previous point, and the chain moves slowly.

The Metropolis Algorithm

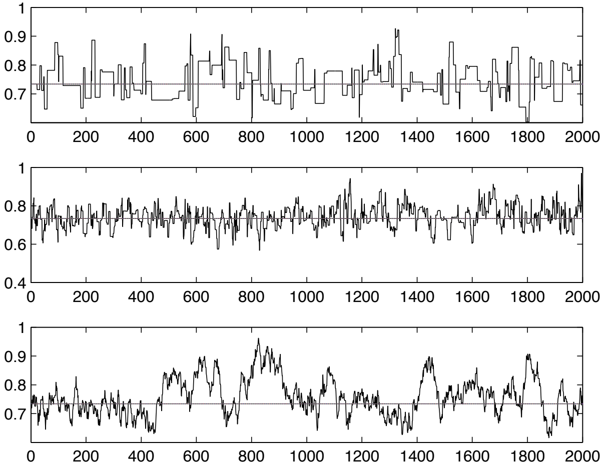


Figure: Top: too wide, bottom: too narrow, middle: good 'mixing'.

The Metropolis Algorithm

- In this course, we typically deal with multidimensional continuous parameter spaces.
- In such a setting, a multivariate Gaussian distribution is a common choice for a proposal distribution.
- In the commonly used *random walk Metropolis* algorithm, the current point in the chain is taken as the center point of the Gaussian proposal.
- The problem then is to find a suitable proposal covariance matrix so that its size and the shape (orientation) match well with the target density.
- The covariance matrix selection problem is discussed in detail a bit later.

The Metropolis Algorithm

- Our most typical application is MCMC for standard nonlinear parameter estimation.
- Typically, the prior information we have are some bound constraints for the parameters, and within the bounds we use a uniform, 'flat' prior, $p(\theta) \propto 1$.
- Assuming independent measurement error with a known constant variance σ^2, the posterior density can be written as

$$\pi(\theta|\mathbf{y}) \propto l(\mathbf{y}|\theta)p(\theta) \propto \exp\left(-\frac{1}{2\sigma^2}SS(\theta)\right),$$

  where $SS(\theta) = \sum_{i=1}^{n}[y_i - f(x_i, \theta)]^2$ is the LSQ function.
- Using this notation, the acceptance ratio reduces to

$$\min\left(1, \frac{\pi(\hat{\theta})}{\pi(\theta_n)}\right) = \min\left(1, \exp\left(-\frac{1}{2\sigma^2}\left(SS(\hat{\theta}) - SS(\theta_n)\right)\right)\right).$$

The Metropolis Algorithm

- Using these assumptions and this notation, the Metropolis algorithm with a Gaussian proposal, with covariance matrix $\mathbf{C}$ and initial point $\theta_{old} = \theta_0$, can be written as follows:

  1. Generate a candidate value $\theta_{new} \sim N(\theta_{old}, \mathbf{C})$ and compute $SS(\theta_{new})$.
  2. Accept the candidate if $u < \exp\left(-\frac{1}{2\sigma^2}\left(SS(\theta_{new}) - SS(\theta_{old})\right)\right)$ where $u \sim U(0,1)$.
     - 2.1 If accepted, add θ_{new} to the chain and set $\theta_{old} := \theta_{new}$ and $SS(\theta_{old}) := SS(\theta_{new})$.
     - 2.2 If rejected, repeat θ_{old} in the chain.
  3. Go to step 1 until a desired chain length is achieved.

- This will be the version of the Metropolis algorithm that is mostly used in this course.

The Metropolis Algorithm

- Note that although we assume here a flat prior, it is easy to add possible prior information about the parameters.
- In this course, we will mostly use bound constraints as prior information.
- Implementing simple bound constraints is easy: if the proposed parameter is out of bounds, it is simply rejected.
- The progress of the Metropolis algorithm is animated in the `mcmcmovie` demo program.

The Metropolis Algorithm

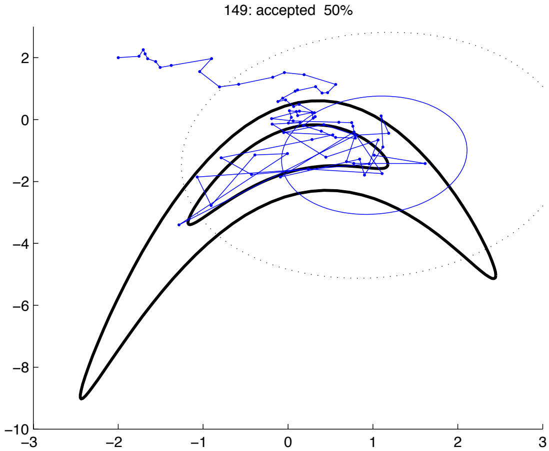
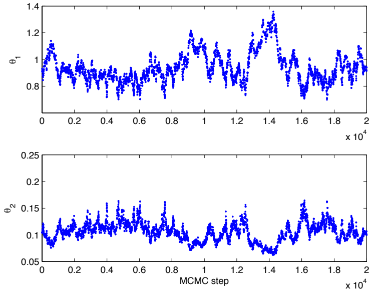


Figure: The path of the Metropolis sampler (blue line), when sampling a non-Gaussian target with a Gaussian proposal distribution (ellipses).

Code example: implementing the Metropolis algorithm

- Let us consider again model $\mathbf{y} = \theta_1(1 - \exp(-\theta_2 \mathbf{x}))$ with data $\mathbf{x} = (1, 3, 5, 7, 9)$, $\mathbf{y} = (0.076, 0.258, 0.369, 0.492, 0.559)$.
- We first compute the LSQ estimate by minimizing the sum of squares, and use that as the starting point for MCMC.
- A spherical proposal covariance $\mathbf{C} = \alpha \mathbf{I}$ is used, where α controls the size of the proposal distribution.
- The measurement error variance is estimated from the residuals using the MSE formula.
- The code is given in two files, the main program `bod_mcmc.m` and the sum of squares objective function `bod_ss.m`.

Code example: implementing the Metropolis algorithm



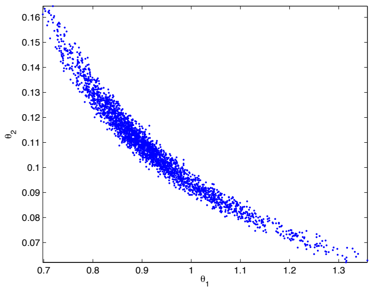
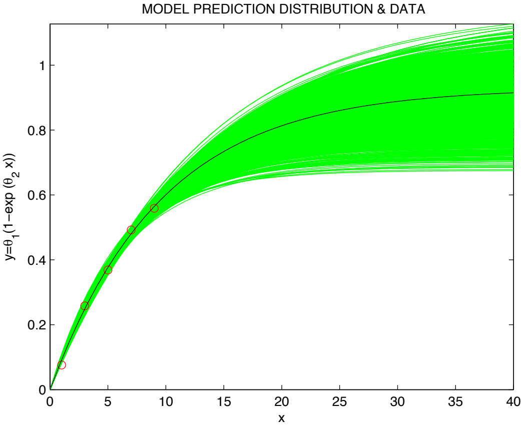


Figure: Left: the path of the MCMC sampler for both parameters. Right: the posterior distribution of the parameters.

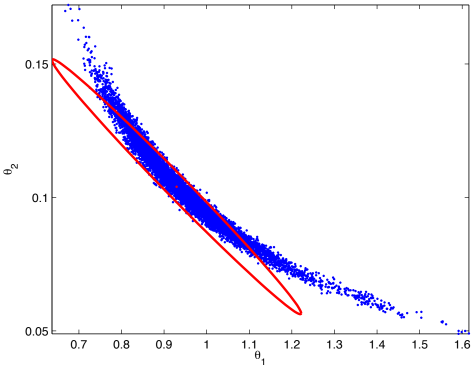
Predictive Distributions

In addition to finding the distribution of the model parameters, the MCMC results can be used to simulate the distribution the model predictions, the *predictive distribution*.



Selecting the Proposal Covariance Matrix

- A good starting point for selecting $\mathbf{C}$ is to use the approximation obtained via linearization: $\mathbf{C} = \sigma^2(\mathbf{J}^T\mathbf{J})^{-1}$.
- This proposal can better match with the orientation of the target distribution:



Selecting the Proposal Covariance Matrix

- In addition to orientation, scale of the proposal distribution is important.
- Theory has been developed for the optimal scaling of the proposal covariance matrix for Metropolis algorithm.
- For Gaussian targets, an efficient scaling factor is $s_d = 2.4^2/d$, where d is the dimension of the problem.
- This result can be used as a rule of thumb also for non-Gaussian targets.
- That is, utilizing the Jacobian-based covariance matrix, we can use proposal covariance matrix $\mathbf{C} = s_d \sigma^2 (\mathbf{J}^T \mathbf{J})^{-1}$.
- A code example of the Jacobian-based proposal is given in `bod_mcmc2.m`.

Selecting the Proposal Covariance Matrix

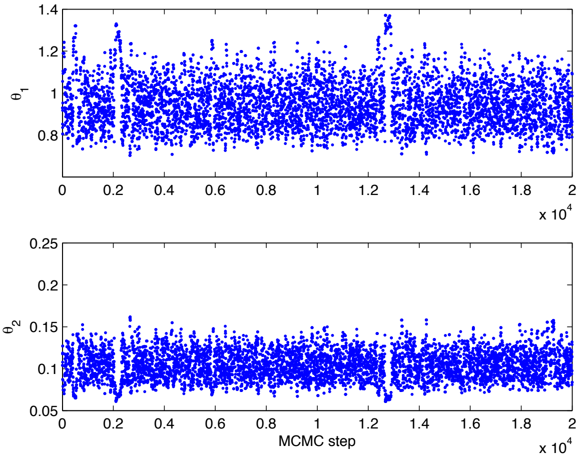


Figure: The path of the MCMC sampler for the two parameters with proposal covariance matrix $\mathbf{C} = s_d \sigma^2 (\mathbf{J}^T \mathbf{J})^{-1}$.

On MCMC Theory: Ergodicity

- A central concept in MCMC theory is *ergodicity*, which guarantees the correctness of an MCMC algorithm in the sense that the Law of Large Numbers (LLN) holds.
- LLN: the averages computed from the MCMC samples approach the correct expected value as the number of samples increases.
- Formally: let π be the target density. An MCMC algorithm is said to be ergodic if, for an arbitrary function $f : \mathbb{R}^d \to \mathbb{R}$ and initial parameter value θ_0, it holds that

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

  where $(\theta_0, ..., \theta_n)$ are the samples produced by the MCMC algorithm.
- It can be shown, for instance, that the Metropolis algorithm described above is ergodic.

On MCMC Theory: Ergodicity

- 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 the model prediction $f(\mathbf{x}, \theta)$. The theorem then states that the values calculated at the sampled parameters correctly gives the distribution of the model predictions, also known as the *predictive distribution*.

Adaptive MCMC

- The bottleneck in MCMC computations is usually selecting a proposal distribution.
- The covariance matrix using the linearization of the model is a good starting point, but does not always lead to efficient sampling.
- The purpose of adaptive MCMC methods is to tune the proposal 'on the run' as the sampling proceeds, using the information of the previously sampled points.
- In the Adaptive Metropolis (AM) algorithm, the proposal covariance matrix is taken to be the empirical covariance matrix computed from the history.
- Note that now the sampled points depend on the earlier history of the chain, and the chain is therefore no longer Markovian.
- However, it can be shown that the AM algorithm gives correct (ergodic) results.

AM Algorithm

- In AM, if we have sampled points $(\theta_0, ..., \theta_{n-1})$, we propose the next candidate using $\mathbf{C}_n = s_d \mathrm{Cov}(\theta_0, \dots, \theta_{n-1}) + \varepsilon \mathbf{I}_d$.
- Here s_d is the scaling factor and $\varepsilon > 0$ is a regularization parameter that ensures that the proposal covariance matrix is positive definite.
- In practice, ε can often be chosen to be very small or set to zero.
- In the beginning of the sampling, a positive definite initial covariance $\mathbf{C}_0$ is chosen.
- A time index $n_0 > 0$ defines the length of the initial non–adaptation period, after which the empirical covariance matrix is used:

$$\mathbf{C}_n = \begin{cases} \mathbf{C}_0, & n \le n_0 \\ s_d \mathrm{Cov}(\theta_0, \dots, \theta_{n-1}) + s_d \varepsilon \mathbf{I}_d, & n > n_0. \end{cases}$$

AM Algorithm

- The initial proposal covariance $\mathbf{C}_0$ needs to be specified.
- A good starting point is often the approximative error analysis given by the linearization of the model.
- That is, we can use the scaled Jacobian-based covariance matrix as the initial proposal: $\mathbf{C}_0 = s_d \sigma^2 (\mathbf{J}^T \mathbf{J})^{-1}$.
- However, in most simple cases, it is enough to use a simple (e.g. diagonal) $\mathbf{C}_0$ that is small enough so that the sampler gets moving, and let the adaptation tune the proposal.
- The empirical covariance matrix does not have to be recomputed every time, since recursive formulas exist.

AM Algorithm

- The covariance matrix $\mathbf{C}_n$ satisfies the recursive formula

$$\mathbf{C}_{n+1} = \frac{n-1}{n}\mathbf{C}_n + \frac{s_d}{n}\left(n\overline{\theta}_{n-1}\overline{\theta}_{n-1}^T - (n+1)\overline{\theta}_n\overline{\theta}_n^T + \theta_n\theta_n^T + \varepsilon\mathbf{I}_d\right),$$

- The mean $\overline{\theta}_n$ also has an obvious recursive formula.
- Only the expression $\theta_n\theta_n^T/n$ is 'new' in the update formula, all the rest depends on previous mean values.
- So the effect of adaptation goes down as $1/n$; this is often called *diminishing adaptation*: in the long run, AM goes back to usual non-adaptive sampling.
- This form of adaptation can proved to be ergodic, but the same adaptation with a *fixed* update length for the covariance is *not ergodic*.

AM Algorithm

- The choice of the length of the *burn-in period* n_0 is free.
- The adaptation might not be efficient if done at each time step, and one should adapt only at given time intervals.
- Finally, the AM algorithm as a pseudocode:

```
- Choose the lenght of the chain $N$ and initialize $\theta_1$ and $\mathbf{C}_1$.
- For $k = 1, 2, ..., N$
    * Perform the Metropolis step, using proposal $N(\theta_k, \mathbf{C}_k)$.
    * Update $\mathbf{C}_{k+1} = \mathrm{Cov}(\theta_1, ..., \theta_k)$.
```

- One may compute the covariance by the whole chain $(\theta_1, ..., \theta_k)$ or by an increasing part of it, for instance $(\theta_{k/2}, ..., \theta_k)$.

Delayed Rejection Adaptive Metropolis

- In the Metropolis algorithm, a candidate move $\tilde{\theta}_k$ is generated from a proposal distribution $q_1(\cdot|\theta_k)$.
- In the *Delayed Rejection* (DR) algorithm, upon rejection, a second stage move $\tilde{\theta}_k^{(2)}$ is proposed from a proposal distribution q_2.
- The second stage proposal is allowed to depend on what we have just proposed and rejected: $q_2(\cdot|\theta_k, \tilde{\theta}_k)$.
- An ergodic chain is created, if the second stage proposal is accepted with suitably modified acceptance probability (details skipped here).
- The process of delaying rejection can be iterated to try sampling from further proposals.
- We often use only a 2-stage version, where the 2nd stage proposal is a downscaled version of the first stage proposal (upon rejection, we try a new candidate value closer to the current point).

Delayed Rejection Adaptive Metropolis

- The delayed rejection method can be combined with the adapting proposal covariance matrix.
- This algorithm is called Delayed Rejection Adaptive Metropolis (DRAM).
- The algorithm can be implemented in various ways, but we often use a the following simple implementation:
  - The proposal at the 1st stage of DR is adapted just as in AM: the covariance matrix $\mathbf{C}_n^1$ 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 $\mathbf{C}_n^i$ of the proposal for the i-th stage $(i = 2, ..., m)$ is computed as a scaled version of the proposal of the first stage, $\mathbf{C}_n^i = \gamma_i \mathbf{C}_n^1$, with fixed scaling factors γ_i.

MCMC in practice: the `mcmcrun` tool

- In this course we use a MATLAB code package that makes it easy to run MCMC analyses.
- The code is written by Marko Laine, and it can be downloaded from `http://helios.fmi.fi/ lainema/mcmc/`.
- The toolbox provides a unified interface for specifying models, and implements the AM and DRAM methods.
- Let us demonstrate the toolbox by considering again the simple BOD model $\mathbf{y} = \theta_1(1 - \exp(-\theta_2 \mathbf{x}))$ with data $\mathbf{x} = (1, 3, 5, 7, 9)$, $\mathbf{y} = (0.076, 0.258, 0.369, 0.492, 0.559)$.
- The main program for running the MCMC analysis is `bod_mcmcrun.m`. The code uses the same sum of squares function `bod_ss.m` as before.
- The MCMC is run with the `mcmcrun` function, and the results are visualized with `mcmcplot`, see lecture notes for details.

Visualizing MCMC Output

- Previously we have plotted the output as two dimensional distribution plots and one dimensional 'chain' plots that give the sample paths for each parameter.
- In addition to these, one can obviously approximate the one dimensional marginal distributions, for instance, by histograms.
- In the `mcmcplot` function, one can give the format of the visualization: `'chain'` gives the chain plot, `'pairs'` draws the 2D marginal distributions and `'hist'` gives histograms.
- The target density can be approximated based on the obtained samples also by the *kernel density estimation* technique.

Visualizing MCMC Output

- In kernel density estimation, the density is approximated by a sum of certain kernel functions, which are centered at the sampled parameter values.
- The kernel function can be, for instance, the density function of the normal distribution.
- The width and the orientation of the Gaussian kernel functions can be controlled via the covariance matrix.
- Kernel density estimation is implemented in the `mcmcplot` function.
- One can add density lines to the two dimensional marginal plots by using the function as `mcmcplot(chain,inds,names,'pairs',smo,rho)`, where `smo` gives the width of the kernel and `rho` gives the orientation (correlation).

Visualizing MCMC Output

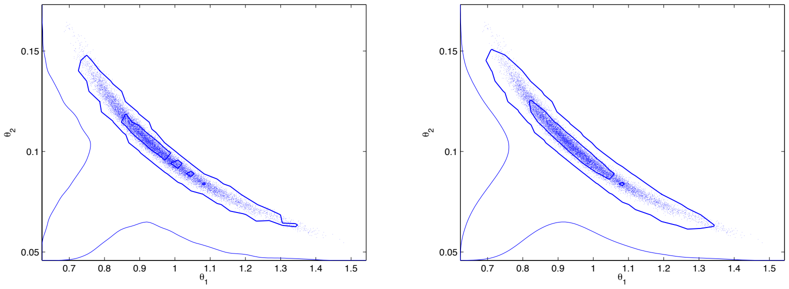


Figure: Pairs plots with two different width parameters for kernel density estimation.

Visualizing MCMC Output

- In addition to visualizing the parameter posterior, we are often interested in the predictive distribution.
- Predictive distributions can be visualized simply by simulating the prediction model with different parameter values and drawing the prediction curves.
- Another way is to compute, for instance, 50% and 95% envelopes for the predictions.
- The MCMC package contains functions for visualizing predictive distributions:
  - `mcmcpred` simulates the model responses and computes different confidence envelopes.
  - `mcmcpredplot` draws the envelopes
- Check the demo program `bod_mcmcrun_pred.m`.

Visualizing MCMC Output

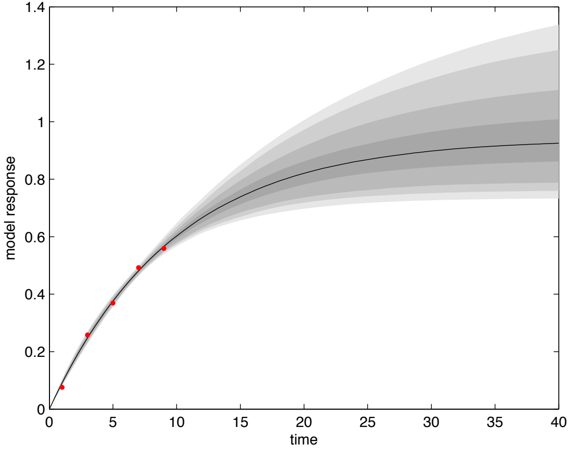


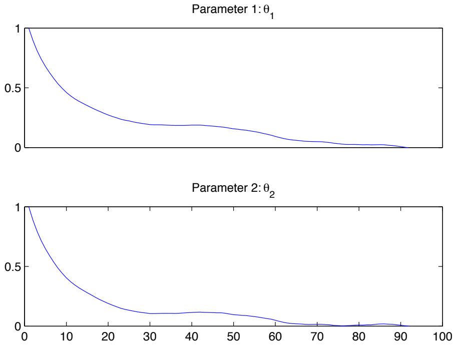
Figure: The data (red dots), the median of the predictions (black line) and the 50%, 95% and 99% confidence envelopes for the predictions (grey areas).

MCMC Convergence Diagnostics

- Usually, one can simply visually inspect the chains to see how well the chain is mixing and if the sampler has reached its stationary distribution.
- However, various formal diagnostic methods have been developed to study if the sampler has converged.
- In code package, the `chainstats` function can be used to compute some basic statistics of the chain.
- A useful way to visualize how well the chain is mixing is to plot the *autocorrelation function* (ACF) of the parameter chains.
- The ACF tells how much, on average, samples that are k steps apart correlate with each other.

MCMC Convergence Diagnostics

- In MCMC methods, subsequent points correlate with each other since the next point depends on the previous point.
- The further apart the samples are in the chain, the less they correlate.
- The ACF can be visualized with the `mcmcplot` using the plotting mode `'acf'`.



Gibbs Sampling

- In the Metropolis algorithm presented above, candidate values for all parameters are proposed at the same time.
- Sometimes, especially in high-dimensional problems, it may be difficult to find a good multivariate proposal distribution for all parameters simultaneously.
- The idea in Gibbs sampling is to reduce the sampling to one dimensional distributions.
- Each parameter is sampled in turn, while the other parameters are kept fixed.

Gibbs Sampling

- In more detail, the parameter vector $(\theta_1, \theta_2, ..., \theta_p)$ is updated in sweeps, by updating one coordinate at a time.
- This may be done if the 1D *conditional distributions* $\pi(\theta_i|\theta_1, ..., \theta_{i-1}, \theta_{i+1}, ..., \theta_p)$ are known.
- In many cases these reduce to simple known densities which are easy to sample from.
- But often the conditional distributions are not known, and they must be approximated by computing 'sufficiently' many values in the 1D directions.
- Gibbs pseudocode:

```
- for k=1,...,N
    - for i=1,...,p
      sample $\theta_i^k$ from the 1D conditional distribution
      $\pi(\theta_i|\theta_1, ..., \theta_{i-1}, \theta_{i+1}, ..., \theta_p)$.
```

Gibbs Sampling

- Note that there is no accept-reject procedure here and 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 $\pi(\theta_i|\theta_1, ..., \theta_{i-1}, \theta_{i+1}, ..., \theta_p)$ with respect to the coordinate i a given number of times.
- The computed values can be used to create an empirical CDF.
- The new value for θ_i^k can then be sampled from the empirical distribution by using the inverse CDF method.

Component-wise Metropolis

- Instead of sampling directly from the one-dimensional conditional distributions, as in Gibbs sampling, one can perform component-wise Metropolis sampling.
- The proposal for each component is, for instance, a normal distribution with a given variance, separate for each coordinate.
- When sampling 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 1D proposal $q_t^i \sim N(\theta_{t-1}^i, v^i)$ 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 $\theta_t^i = \theta_{t-1}^i$.

Importance Sampling

- Importance sampling is a Monte Carlo method for approximating integrals of form

$$\mathrm{E}_p(f) = \int f(x)p(x)dx.$$

- A straightforward way would be to sample points $x_1, \ldots, x_m$ from $p(x)$ and approximate the expectation by

$$\mathrm{E}_p(f) \approx \bar{f}_m = \frac{1}{m}\sum_{j=1}^{m} f(x_j).$$

- Suppose we cannot sample directly from $p(x)$, but we do know $g(x)$ (such that $g(x) > 0$ if $p(x) > 0$) from which we can sample.

Importance Sampling

- Let us write the expectation as

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

- We can sample from $g(x)$ and approximate the expected value 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* and w as the *importance weight*.
- The function g should be chosen so that it mimics the distribution p, and is easy to sample from.

Conjugate Priors and MCMC Estimation of σ^2

- In our typical applications, the likelihood distribution reads as

$$l(\mathbf{y}|\theta) = (2\pi\sigma^2)^{-n/2} \exp\left(-\frac{1}{2\sigma^2} SS(\theta)\right).$$

- We need to specify the value for the measurement error variance σ^2.
- Previously, we have given a fixed value for σ^2, estimated from the residuals of the model fit or from repeated measurements.
- We can also regard σ^2 as a random variable and treat it in a Bayesian way by sampling it along with the model parameters in the MCMC algorithm.

Conjugate Priors and MCMC Estimation of σ^2

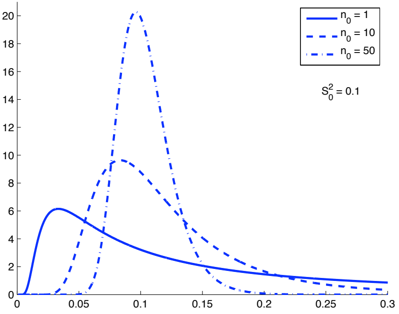
- We often have a rather good idea about the level of the measurement error, and we therefore would like to specify a prior distribution for it.
- A computationally convenient choice for the prior is obtained using the *conjugacy* property.
- If we set the prior so that the posterior is of the same form as the prior, the prior is called a *conjugate prior*.
- Looking at the Gaussian likelihood and considering it as a function of σ^2 (with fixed θ), we see that the inverse variance $1/\sigma^2$ has a Gamma type distribution.
- The conjugate prior for the Gamma distribution is also Gamma. That is, if we specify a Gamma prior for $1/\sigma^2$, the conditional posterior $p(\sigma^{-2}|\mathbf{y}, \theta)$ will also be Gamma distributed.

Conjugate Priors and MCMC Estimation of σ^2

- More specifically, the prior for σ^{-2} can be defined as

$$\sigma^{-2} \sim \Gamma(\frac{n_0}{2}, \frac{n_0}{2} S_0^2),$$

where S_0^2 gives the mean value for σ^2 and n_0 defines how accurate we think the value S_0^2 is. The higher n_0 is, the more peaked the prior distribution is around S_0^2.



Conjugate Priors and MCMC Estimation of σ^2

- With the conjugate prior, we can derive a Gamma posterior for the conditional posterior for the error variance, $p(\sigma^{-2}|\mathbf{y},\theta)$.
- Skipping the details, the conditional posterior for σ^{-2} is

$$p(\sigma^{-2}|\mathbf{y},\theta) = \Gamma(\frac{n_0+n}{2}, \frac{n_0 S_0^2 + SS(\theta)}{2}).$$

- Now we have an analytical expression for the conditional distribution of σ^{-2}, and we can build a Gibbs sampler that samples θ and σ^2 in turn:
  1. Sample a new θ value from $p(\theta|\sigma^{-2},\mathbf{y})$
  2. Sample a new σ^2 value from $p(\sigma^{-2}|\mathbf{y},\theta)$
- In the `mcmcrun` code, σ sampling can be turned on by setting `options.updatesigma = 1`. See `bod_mcmcrun_sig.m` for a demo.

Dynamical State Estimation

- Besides estimating the static parameters θ, one is often interested in estimating the dynamically changing *state* of the system.
- As time proceeds, new measurements become available, that can be used to update the state estimates.
- The state estimation problem: at time k, estimate the system state $\mathbf{x}_k$ using previous observations $\mathbf{y}_{1:k} = (\mathbf{y}_1, ..., \mathbf{y}_k)$, when the model and observation equations are given as

$$\begin{aligned} \mathbf{x}_k &= \mathcal{M}(\mathbf{x}_{k-1}) + \varepsilon_k^p \\ \mathbf{y}_k &= \mathcal{K}(\mathbf{x}_k) + \varepsilon_k^o, \end{aligned}$$

  where $\mathcal{M}$ is the *evolution model* and $\mathcal{K}$ is the *observation model*. The terms ε_k^p and ε_k^o represent the model error and the observation error.

Dynamical State Estimation

- In dynamical state estimation problems, measurements are obtained in real-time and the state estimate needs to be updated after the measurements are obtained.
- This can be achieved by applying the Bayes' formula sequentially:
    1. The prior is given by evolving the state from the previous time step using the model $\mathcal{M}$ (prediction step)...
    2. ... the prior is updated with the likelihood of the measurement (update step) to get the posterior ...
    3. ... which is evolved with the model and used as the prior in the next time step.
- Repeating this procedure allows 'on-line' estimation of model states.

Dynamical State Estimation

Dynamical state estimation techniques are needed in many applications:

- Target tracking: estimate the position and velocity of an object. For instance, the Global Positioning System (GPS) uses state estimation techniques (extended Kalman filtering).
- Combining accelerometer and gyroscope data to compute the orientation, position and velocity of a gaming device.
- Process tomography: estimate the dynamically changing concentrations of different compounds in a pipe by combining fluid dynamics and chemistry models with tomographic measurements.
- Numerical Weather Prediction (NWP): estimate the state of the weather by correcting the previous prediction with different kinds of observations (ground based, satellite etc.) to allow real-time weather predictions.
- ...

Dynamical State Estimation: General Formulas

- The filtering methods aim at estimating the marginal distribution of the states $p(\mathbf{x}_k|\mathbf{y}_{1:k})$.
- In the prediction step, the distribution of the state is moved with the dynamical model to the next time step:

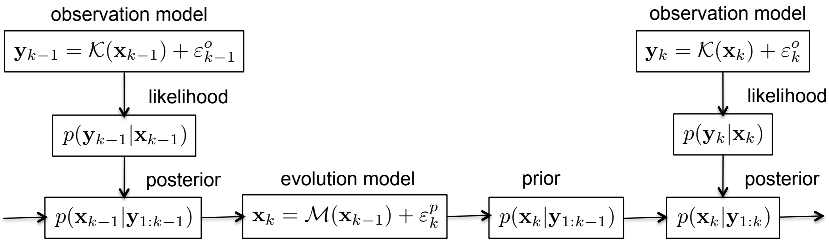
$$p(\mathbf{x}_k|\mathbf{y}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{y}_{1:k-1})d\mathbf{x}_{k-1}.$$

- When the new observation $\mathbf{y}_k$ is obtained, the model state is updated using the Bayes' rule:

$$p(\mathbf{x}_k|\mathbf{y}_{1:k}) \propto p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{y}_{1:k-1}).$$

- The above calculation is repeated for all time steps.

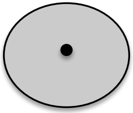
Dynamical State Estimation: General Formulas


observation model
$\mathbf{y}_{k-1} = \mathcal{K}(\mathbf{x}_{k-1}) + \varepsilon^o_{k-1}$
likelihood
$p(\mathbf{y}_{k-1}|\mathbf{x}_{k-1})$
posterior
$p(\mathbf{x}_{k-1}|\mathbf{y}_{1:k-1})$
evolution model
$\mathbf{x}_k = \mathcal{M}(\mathbf{x}_{k-1}) + \varepsilon^p_k$
prior
$p(\mathbf{x}_k|\mathbf{y}_{1:k-1})$
observation model
$\mathbf{y}_k = \mathcal{K}(\mathbf{x}_k) + \varepsilon^o_k$
likelihood
$p(\mathbf{y}_k|\mathbf{x}_k)$
posterior
$p(\mathbf{x}_k|\mathbf{y}_{1:k})$

Different state estimation methods can be derived, depending on the assumptions of the form of the state distribution, the likelihood and the techniques used to evolve the uncertainty of the state in time.

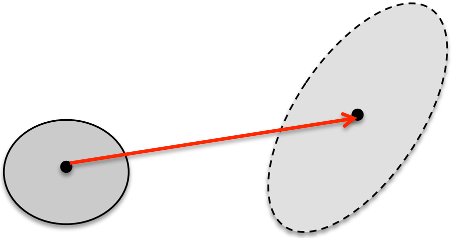
Dynamical State Estimation: General Formulas

1) Current state distribution



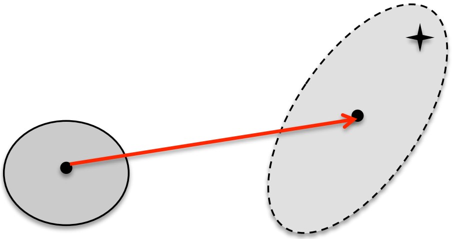
Dynamical State Estimation: General Formulas

2) Predict: move the state distribution with the model



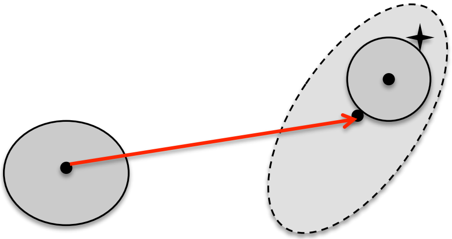
Dynamical State Estimation: General Formulas

3) New observations become available



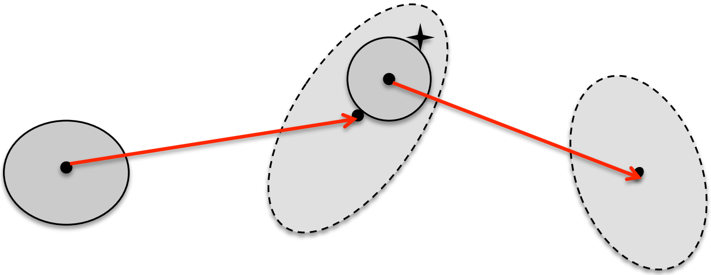
Dynamical State Estimation: General Formulas

4) Update the state estimate with the new observations



Dynamical State Estimation: General Formulas

5) Predict forward



Kalman Filter and Extended Kalman Filter

- The Kalman filter (KF) is meant for situations, where the model is linear and the model and observation errors are assumed to be zero mean Gaussians.
- The extended Kalman filter (EKF) is its extension to nonlinear situations, where the model is linearized and the KF formulas are applied.
- To derive the KF formulas, let us first consider the linear model $\mathbf{y} = \mathbf{A}\mathbf{x}$ with Gaussian prior and Gaussian likelihood.
- Note that now the unknown is denoted by $\mathbf{x}$ instead of θ.

Linear LSQ with Gaussian Prior and Likelihood

- The posterior distribution is $\pi(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$, where the likelihood and prior are

$$\begin{aligned} p(\mathbf{x}) &\propto \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{x}_p)^T\mathbf{P}^{-1}(\mathbf{x}-\mathbf{x}_p)\right) \\ p(\mathbf{y}|\mathbf{x}) &\propto \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{A}\mathbf{x})^T\mathbf{R}^{-1}(\mathbf{y}-\mathbf{A}\mathbf{x})\right), \end{aligned}$$

  where $\mathbf{x}_p$ is the prior mean and $\mathbf{P}$ and $\mathbf{R}$ are model error and measurement error covariance matrices.

- Maximizing the posterior is equivalent to minimizing the negative log-likelihood

$$-2\log(\pi(\mathbf{x}|\mathbf{y})) = (\mathbf{y}-\mathbf{A}\mathbf{x})^T\mathbf{R}^{-1}(\mathbf{y}-\mathbf{A}\mathbf{x})+(\mathbf{x}-\mathbf{x}_p)^T\mathbf{P}^{-1}(\mathbf{x}-\mathbf{x}_p).$$

Linear LSQ with Gaussian Prior and Likelihood

- Let decomposed the covariance matrices into $\mathbf{P}^{-1} = \mathbf{K}^T\mathbf{K}$ and $\mathbf{R}^{-1} = \mathbf{L}^T\mathbf{L}$.
- Then, we can write the least squares expression as

$$\begin{aligned}&(\mathbf{y}-\mathbf{A}\mathbf{x})^T\mathbf{L}^T\mathbf{L}(\mathbf{y}-\mathbf{A}\mathbf{x}) + (\mathbf{x}-\mathbf{x}_p)^T\mathbf{K}^T\mathbf{K}(\mathbf{x}-\mathbf{x}_p)\\ &= (\mathbf{L}\mathbf{y}-\mathbf{L}\mathbf{A}\mathbf{x})^T(\mathbf{L}\mathbf{y}-\mathbf{L}\mathbf{A}\mathbf{x}) + (\mathbf{K}\mathbf{x}-\mathbf{K}\mathbf{x}_p)^T(\mathbf{K}\mathbf{x}-\mathbf{K}\mathbf{x}_p).\end{aligned}$$

- We continue by combining both terms into one expression:

$$-2\log(\pi(\mathbf{x}|\mathbf{y})) = (\tilde{\mathbf{y}}-\tilde{\mathbf{A}}\mathbf{x})^T(\tilde{\mathbf{y}}-\tilde{\mathbf{A}}\mathbf{x}),$$

where

$$\tilde{\mathbf{y}} = \begin{pmatrix}\mathbf{L}\mathbf{y}\\ \mathbf{K}\mathbf{x}_p\end{pmatrix} \quad \tilde{\mathbf{A}} = \begin{pmatrix}\mathbf{L}\mathbf{A}\\ \mathbf{K}\end{pmatrix}.$$

Linear LSQ with Gaussian Prior and Likelihood

- That is, we have transformed the problem into a least squares problem with identity $\mathbf{I}$ as the error covariance matrix.
- This can be solved with the LSQ formulas: $\hat{\mathbf{x}} = (\tilde{\mathbf{A}}^T\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^T\tilde{\mathbf{y}}$ and $\mathrm{Cov}(\hat{\mathbf{x}}) = (\tilde{\mathbf{A}}^T\tilde{\mathbf{A}})^{-1}$.
- Switching back to the original notation, theLSQ solution $\hat{\theta}$ and its covariance matrix $\mathbf{C}$ are

$$\begin{aligned}\hat{\mathbf{x}} &= (\mathbf{A}^T\mathbf{R}^{-1}\mathbf{A} + \mathbf{P}^{-1})^{-1}(\mathbf{A}^T\mathbf{R}^{-1}\mathbf{y} + \mathbf{P}^{-1}\mathbf{x}_p)\\ \mathbf{C} &= (\mathbf{A}^T\mathbf{R}^{-1}\mathbf{A} + \mathbf{P}^{-1})^{-1}.\end{aligned}$$

- These formulas are similar to the previously obtained LSQ formulas, but here we have the prior center point $\mathbf{x}_p$ and covariance matrix $\mathbf{P}$ present.

The Kalman Filter

- Let us now consider the linear state space model

$$\begin{aligned} \mathbf{x}_k &= \mathbf{M}_k \mathbf{x}_{k-1} + \varepsilon_k^p \\ \mathbf{y}_k &= \mathbf{K}_k \mathbf{x}_k + \varepsilon_k^o, \end{aligned}$$

  where the model error and observation error are assumed to be zero mean Gaussian with covariance matrices $\mathbf{Q}_k$ and $\mathbf{R}_k$.
- The idea of filtering is to estimate the state vector $\mathbf{x}_k$ for time steps $k = 1, 2....$
- Let us assume that we have at time step $k-1$ obtained a state estimate $\mathbf{x}_{k-1}^{\text{est}}$ with covariance matrix $\mathbf{C}_{k-1}^{\text{est}}$.
- The prior center point for the next time step k is given by the model prediction:

$$\mathbf{x}_k^p = \mathbf{M}_k \mathbf{x}_{k-1}^{\text{est}}.$$

The Kalman Filter

- The covariance of the prediction is computed using the assumption that the state vector and model error are statistically independent:

$$\mathbf{C}_k^p = \mathrm{Cov}(\mathbf{M}_k \mathbf{x}_{k-1}^{\mathrm{est}} + \varepsilon_k^p) = \mathbf{M}_k^T \mathbf{C}_{k-1}^{\mathrm{est}} \mathbf{M}_k + \mathbf{Q}_k.$$

- This Gaussian with mean $\mathbf{x}_k^p$ and covariance matrix $\mathbf{C}_k^p$ is used as the prior, which is updated with the new measurement vector $\mathbf{y}_k$.
- The negative log-posterior for the state vector at time step k is

$$(\mathbf{x}_k - \mathbf{x}_k^p)^T (\mathbf{C}_k^p)^{-1} (\mathbf{x}_k - \mathbf{x}_k^p) + (\mathbf{y}_k - \mathbf{K}_k \mathbf{x}_k)^T \mathbf{R}_k^{-1} (\mathbf{y}_k - \mathbf{K}_k \mathbf{x}_k),$$

  which is just the linear least squares problem with Gaussian prior and Gaussian likelihood discussed in the previous section.

The Kalman Filter

- For computational reasons, the KF formulas are usually written in the following form, which can be obtained via direct but somewhat non-trivial matrix manipulations (exercise):

$$\begin{aligned}\mathbf{G}_k &= \mathbf{C}_k^p \mathbf{K}_k^{\mathrm{T}} (\mathbf{K}_k \mathbf{C}_k^p \mathbf{K}_k^{\mathrm{T}} + \mathbf{R}_k)^{-1} \\ \mathbf{x}_k^{\mathrm{est}} &= \mathbf{x}_k^p + \mathbf{G}_k (\mathbf{y}_k - \mathbf{K}_k \mathbf{x}_k^p) \\ \mathbf{C}_k^{\mathrm{est}} &= \mathbf{C}_k^p - \mathbf{G}_k \mathbf{K}_k \mathbf{C}_k^p.\end{aligned}$$

- In the above formulas, $\mathbf{G}_k$ is called the *Kalman gain* matrix.

The Kalman Filter

To sum up, we write the Kalman filter as an algorithm:

1. **Prediction**: move the state estimate $\mathbf{x}_{k-1}^{\text{est}}$ and its covariance $\mathbf{C}_{k-1}^{\text{est}}$ in time
   1.1 Compute $\mathbf{x}_k^p = \mathbf{M}_k \mathbf{x}_{k-1}^{\text{est}}$.
   1.2 Compute $\mathbf{C}_k^p = \mathbf{M}_k \mathbf{C}_{k-1}^{\text{est}} \mathbf{M}_k^{\text{T}} + \mathbf{Q}_k$.
2. **Update**: combine the prior $\mathbf{x}_k^p$ with observations $\mathbf{y}_k$
   2.1 Compute the Kalman gain $\mathbf{G}_k = \mathbf{C}_k^p \mathbf{K}_k^{\text{T}} (\mathbf{K}_k \mathbf{C}_k^p \mathbf{K}_k^{\text{T}} + \mathbf{R}_k)^{-1}$.
   2.2 Compute the state estimate $\mathbf{x}_k^{\text{est}} = \mathbf{x}_k^p + \mathbf{G}_k (\mathbf{y}_k - \mathbf{K}_k \mathbf{x}_k^p)$.
   2.3 Compute the covariance estimate $\mathbf{C}_k^{\text{est}} = \mathbf{C}_k^p - \mathbf{G}_k \mathbf{K}_k \mathbf{C}_k^p$.
3. Increase k and go to step 1.

The Extended Kalman Filter

- The extended Kalman filter (EKF) is an extension of the Kalman filter to the case where the evolution and/or observation model are nonlinear.

- The EKF directly uses the Kalman filter formulas by replacing the model and operators with linearizations: $\mathbf{M}_k = \partial\mathcal{M}(\mathbf{x}_{k-1}^{\text{est}})/\partial\mathbf{x}$ and $\mathbf{K}_k = \partial\mathcal{K}(\mathbf{x}_k^p)/\partial\mathbf{x}$.

- For small scale models, the linearizations can be computed numerically, using finite differences.

Ensemble Kalman Filtering

- In ensemble filtering, the uncertainty in the state $\mathbf{x}_k$ is represented as N samples, here denoted as $\mathbf{s}_k = (\mathbf{s}_{k,1}, \mathbf{s}_{k,2}, ..., \mathbf{s}_{k,N})$.
- In the ensemble Kalman filter (EnKF), the covariances in EKF are replaced with the sample covariances calculated from the ensemble.
- The sample covariance can be written as $\mathrm{Cov}(\mathbf{s}_k) = \mathbf{X}_k \mathbf{X}_k^{\mathrm{T}}$, where

$$\mathbf{X}_k = \left((\mathbf{s}_{k,1} - \overline{\mathbf{s}_k}), (\mathbf{s}_{k,2} - \overline{\mathbf{s}_k}), ..., (\mathbf{s}_{k,N} - \overline{\mathbf{s}_k})\right) / \sqrt{N-1}.$$

- The sample mean is denoted by $\overline{\mathbf{s}_k}$.

Ensemble Kalman Filtering

Using this notation, the EnKF algorithm can be formulated as follows:

1. **Prediction**: move the state estimate and covariance in time
   1.1 Move ensemble forward and perturb members with model error:
   $\mathbf{s}^p_{k,i} = \mathcal{M}(\mathbf{s}^{\text{est}}_{(k-1),i}) + \mathbf{e}^p_{k,i}$, $i = 1, ..., N$.
   1.2 Calculate sample mean $\overline{\mathbf{s}^p_k}$ and covariance $\mathbf{C}^p_k = \mathbf{X}_k \mathbf{X}_k^{\mathrm{T}}$.
2. **Update**: combine the prior with observations
   2.1 Compute the Kalman gain $\mathbf{G}_k = \mathbf{C}^p_k \mathbf{K}_k^{\mathrm{T}} (\mathbf{K}_k \mathbf{C}^p_k \mathbf{K}_k^{\mathrm{T}} + \mathbf{C}_{\varepsilon^o_k})^{-1}$.
   2.2 Update ensemble members
   $\mathbf{s}^{\text{est}}_{k,i} = \mathbf{s}^p_{k,i} + \mathbf{G}_k(\mathbf{y}_k - \mathbf{K}_k \mathbf{s}^p_{k,i} + \mathbf{e}^o_{k,i})$.
   2.3 Calculate state estimate as the sample mean: $\mathbf{x}^{\text{est}}_k = \overline{\mathbf{s}^{\text{est}}_k}$.
3. Increase k and go to step 1.

In the above algorithm, vectors $\mathbf{e}^p_{k,i}$ and $\mathbf{e}^o_{k,i}$ are realizations of the model and observation errors ε^p_k and ε^o_k.

Particle Filtering

- The Kalman filtering methods described above assume a Gaussian form for the filtering distributions.
- Particle filtering (PF) methods do not rely on any assumptions about the form of the target.
- PF methods, often also called *sequential Monte Carlo* (SMC) methods, are based on sequential application of importance sampling.
- Let us describe the state estimate at time $k-1$ with samples $\mathbf{s}^{\mathrm{est}}_{(k-1,i)}$, where $i = 1, ..., N$.
- In PF, the forward model $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ is used to move the particles forward to obtain the predicted particles $\mathbf{s}^{\mathrm{p}}_{k,i}$.

Particle Filtering

- The predicted particles are considered to be samples from the prior distribution at time step k
- The prior distribution is considered as the importance function for sampling from the posterior.
- That is, the importance weight for particle i at time k becomes

$$w_{k,i} = \frac{\pi(\mathbf{s}^{\mathrm{p}}_{k,i}|\mathbf{y}_k)}{p(\mathbf{s}^{\mathrm{p}}_{k,i})} = \frac{p(\mathbf{y}_k|\mathbf{s}^{\mathrm{p}}_{k,i})p(\mathbf{s}^{\mathrm{p}}_{k,i})}{p(\mathbf{s}^{\mathrm{p}}_{k,i})} = p(\mathbf{y}_k|\mathbf{s}^{\mathrm{p}}_{k,i}).$$

- Thus, the weights for the particles can be computed directly using the likelihood.
- The predicted particles that hit closer to the observation get a larger weight.
- Once the importance weights are obtained, the posterior particles $\mathbf{s}^{\mathrm{est}}_{k,i}$ are sampled with replacement from the prior particles in proportion to the importance weights.

Particle Filtering

The particle filter implemented in this way is called the *Sequential Importance Resampling* (SIR) algorithm:

1. Move the particles forward: $\mathbf{s}^{\mathrm{p}}_{k,i} \sim p(\mathbf{s}_{k,i}|\mathbf{s}^{\mathrm{est}}_{(k-1,i)})$, with $i = 1, ..., N$.
2. Compute the importance weights $w_{k,i} = p(\mathbf{y}_k|\mathbf{s}^{\mathrm{p}}_{k,i})$, with $i = 1, ..., N$
3. Resample with replacement in proportion to the importance weights to obtain the posterior particles $\mathbf{s}^{\mathrm{est}}_{k,i}$.
4. Increase k and go to step 1.