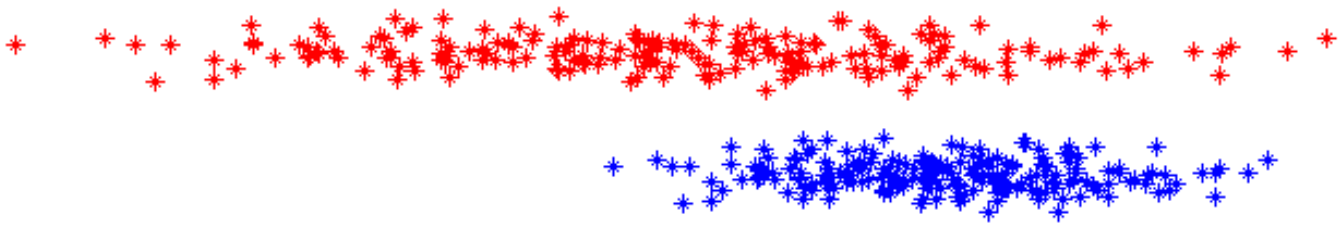
Helsinki University of Technology Control Engineering Laboratory

Espoo 2001 Report 125

# MULTIVARIATE REGRESSION

## Techniques and tools

**Heikki Hyötyniemi**



TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D´HELSINKI

Helsinki University of Technology Control Engineering Laboratory

Espoo July 2001 Report 125

# MULTIVARIATE REGRESSION

## Techniques and tools

**Heikki Hyötyniemi**

**Abstract:** Multivariate statistical methods are powerful tools for analysis and manipulation of large data sets. This report introduces the most important statistical tools that can be used for multivariate regression in a general framework.

**Keywords:** data analysis, multivariate statistical methods, chemometrics; cluster analysis, linear regression, orthogonal least squares, ridge regression, principal component analysis and regression, partial least squares, canonical correlation analysis and regression, independent component analysis, factor analysis; neural networks; subspace identification

Helsinki University of Technology
Department of Automation and Systems Technology
Control Engineering Laboratory

Distribution:
Helsinki University of Technology
Control Engineering Laboratory
P.O. Box 5400
FIN-02015 HUT, Finland
Tel. +358-9-451 5201
Fax. +358-9-451 5208
E-mail: control.engineering@hut.fi
http://www.control.hut.fi/

ISBN 951-22-5587-1
ISSN 0356-0872

Picaset Oy
Helsinki 2001

HELSINKI UNIVERSITY OF TECHNOLOGY CONTROL ENGINEERING LABORATORY

Editor: H. Koivo

Report 112 Hyötyniemi, H., Koivo, H. (eds.),
Multivariate Statistical Methods in Systems Engineering. December 1998.

Report 113 Robyr, S.,
FEM Modelling of a Bellows and a Bellows-Based Micromanipulator. February 1999.

Report 114 Hasu, V.,
Design of Experiments in Analysis of Flotation Froth Appearance. April 1999.

Report 115 Nissinen, A. S., Hyötyniemi, H.,
Analysis Of Evolutionary Self-Organizing Map. September 1999.

Report 116 Hätönen, J.,
Image Analysis in Mineral Flotation. September 1999.

Report 117 Hyötyniemi, H.,
GGHA Toolbox for Matlab. November 1999.

Report 118 Nissinen, A. S.
Neural and Evolutionary Computing in Modeling of Complex Systems. November 1999.

Report 119 Gadoura, I. A.
Design of Intelligent Controllers for Switching-Mode Power Supplies. November 1999.

Report 120 Ylöstalo, T., Salonen, K., Siika-aho, M., Suni, S., Hyötyniemi, H., Rauhala, H., Koivo, H.
Paperikoneen kiertovesien konsentroitumisen vaikutus mikrobien kasvuun. September 2000.

Report 121 Cavazzutti, M.
Fuzzy Gain Scheduling of Multivariable Processes. September 2000.

Report 122 Uykan, Z.
Intelligent Control of DC/DC Switching Buck Converter. December 2000.

Report 123 Jäntti, R.
Power Control and Transmission Rate Management in Cellular Radio Systems - A snapshot analysis approach. May 2001.

Report 124 Uykan, Z.
Clustering-Based Algorithms For Radial Basis Function and Sigmoid Perceptron Networks. June 2001.

Report 125 Hyötyniemi, H.
Multivariate Regression - Techniques and tools. July 2001.

ISBN 951-22-5587-1

ISSN 0356-0872

Picaset Oy, Helsinki 2001

# Preface

In psychology, *regression* means degeneration, or return to a prior, lower level of development. And, indeed, speaking of statistical methods sounds somewhat outdated today — all those neural networks and fuzzy systems being now available.

However, it has been recognized that the new soft computing methods are no panacea. These methods cannot find models any better than the more conventional ones if there is not enough information available in the data to start with. On the other hand, many celebrated soft computing applications could have been solved with age-old methods — assuming that somebody were familiar with them. After all, neural networks can only operate on the statistical properties visible in the data. Why not concentrate on these statistical properties directly?

The starting point here is that *when analysing complex systems, simple methods are needed.* When the system under study is very complicated, one needs data analysis methods that are reliable and fast and that give possibility for closer analysis.

Unfortunately, the statistical literature seems to be mathematically rather unpenetrable for normal engineers (for those that are not too ashamed to admit that). Researchers seem to represent the results in such sophisticated forms that the very simple and elegant ideas remain hidden — the problem is that the powerful methods may not be applied in practice. What is more, different methods have been developed in different research communities: It is difficult to see the connections between the methods when the approaches and notations differ. New methods are constantly being developed; there exists no one-volume book that would cover the whole field in sufficient detail.

This text tries to show how simple the basic ideas are and how closely related different methods turn out to be. The approach is rather pragmatic, many proofs being omitted for readability. All the methods are presented in a homogeneous framework. It is crucial that a student recognizes that all formulas and algorithms are based on simple underlying principles; everything can be questioned and nothing needs to be believed as some kind of unpenetrable "secret wisdom". One objective is to show that there is still room for new ideas and innovations. As the reader can see, there exist plenty of ideas waiting to be explored and exploited — indeeed, one is very near to frontier science, maybe able to cross the boundary towards new discoveries (examples of such exploratory experiments are indicated by stars "*").

The theoretical methods are supported by Matlab routines. The implemented "vanilla" algorithms are by no means optimized; their main purpose is also to show that the methods are by no means unpenetrable.

In addition to the printed version, this report is available in public domain in PostScript format. The Matlab Toolbox and the textual material can be accessed through the HTML page at the Internet address

`http://saato014.hut.fi/hyotyniemi/publications/01_reportXXX.htm.`

The earlier version of this text, with the name "Multivariate Regression — Techniques and Tools" was published in 2001. Despite its various shortcomings, it received a positive acceptance. The printed version was "sold out" a long time ago. This interest was the motivation to try and fix some of the holes that were left open.

In the previous version, applications were not discussed: Now there is the last chapter that tries to present examples of not so evident but potential ways of applying the new tools. Second, the fields of physical first-principles modeling and data-oriented modeling are not so distinct that no connections could be found; the new appendices concentrates on this kind of discussions, fitting the structural considerations into the domain of numeric manipulations in a more or less seamless way.

National Technology Agency of Finland (TEKES) has provided funding during the research under several project frames, and this support is gratefully acknowledged.

Heikki Hyötyniemi

# List of symbols

The same variable names are used consistently in the theoretical discussion and in the accompanying `Regression Toolbox` for `Matlab`, if possible.

- $A$, $B$, $C$, $D$: Matrices determining a state-space system
- $c$: Arbitrary constant, scalar or vector
- $i$, $j$: Matrix and vector indices
- $e$, $E$: Measurement error vector and matrix, dimensions $m \times 1$ and $k \times m$, respectively
- $\epsilon$: State error, dimension $n \times 1$
- $f$, $F$: Vector and matrix defining a linear mapping
- $g(\cdot)$: Any function (scalar or vector-valued)
- $\gamma$, $\Gamma$: Constraint vector and matrix, respectively
- $J(\cdot)$: Cost criterion
- $I$, $I_n$: Identity matrix (of dimension $n \times n$)
- $k$: Time index, sample number (given in parentheses)
- $K$: Kalman gain
- $m$: Dimension of the output space
- $M$: Arbitrary matrix (or vector)
- $n$: Dimension of the input space / Non-compressed feature space
- $N$: Dimension of the latent space / Number of levels or substructures
- $P$: Error covariance matrix, dimension $d \times d$
- $p(\cdot)$: Probability density
- $R$: Covariance matrix (or, more generally, *association matrix*)
- $u$, $U$: Input vector and matrix, dimensions $\nu \times 1$ and $k \times \nu$, respectively
- $v$, $V$: Combined data vector and matrix ($x$ and $y$ together), dimensions $m + n \times 1$ and $k \times m + n$, respectively
- $\nu$: Unprocessed measurement vector / Dimension of the vector $u$ in dynamic systems
- $w$, $W$: Weight vector and matrix, respectively
- $x$, $X$: Data vector and matrix, dimensions $n \times 1$ and $k \times n$, respectively. In the case of a dynamic system, state vector and matrix, dimensions $d \times 1$ and $k \times d$, respectively.

- $y$, $Y$: Output data vector and matrix, dimensions $m \times 1$ and $k \times m$, respectively
- $z$, $Z$: Latent data vector and matrix, dimensions $N \times 1$ and $k \times N$, respectively
- $\xi$, $\zeta$: Arbitrary scalars
- $\lambda$, $\Lambda$: Vector of eigenvalues and eigenvalue matrix, respectively
- $\mu$, $\eta$: Lagrange multipliers
- $\theta$, $\phi$: Reduced base, dimensions $n \times N$ and $m \times N$, respectively
- $\Theta$, $\Phi$: Matrices of data basis vectors, dimensions $n \times n$ and $m \times m$, respectively

# Notations

- $\mathbf{M}$: Unprocessed data
- $\bar{M}$: Mean value of $M$ (columnwise mean matrix if $M$ is matrix)
- $\hat{M}$: Estimate of $M$
- $\tilde{M}$: Error in $M$ / Erroneous $M$
- $M'$: $M$ modified (somehow)
- $M^i$: Power of $M$ / Level $i$ data structure
- $M_i$: Column $i$ for matrix $M$ / Element $i$ for vector-form $M$
- $M^T$: Transpose of $M$
- $M^\dagger$: Psedoinverse of $M$ (for definition, see page 21)
- $\mathrm{E}\{M\}$: Expectation value of $M$
- $M_{\text{test}}$, $M_{\text{est}}$: Independent testing data or run-time data
- $\left( \begin{array}{c|c} M_1 & M_2 \end{array} \right)$: Partitioning of a matrix
- $\underset{\xi \times \zeta}{M}$: Matrix dimensions ($\xi$ rows, $\zeta$ columns)

# Abbreviations

- CCA/CCR: Canonical Correlation Analysis/Regression (page 102)
- CR: Continuum Regression (page 98)
- CA: Cluster Analysis (page 205)
- CLR: Constrained Linear Regression (page **??**)

- DA or FDA: (Fisher) Discriminant Analysis (page 208)
- EIV: Error In Variables model (page 68)
- FOBI: Fourth-Order Blind Identification (page 115)
- GHA: Generalized Hebbian Algorithm (page 138)
- ICA/ICR: Independent Component Analysis/Regression (page 109)
- MLR: Multi-Linear Regression (page 63)
- NNR: Neural Networks based Regression (page 129)
- OLS: Orthogonal Least Squareas (page 72)
- PCA/PCR: Principal Component Analysis/Regression (page 77)
- PLS: Partial Least Squares regression (page 95)
- RR: Ridge Regression (page 73)
- SOM: Self-Organizing Map (page 126)
- SPC: Statistical Process Control (page 90)
- SSI: SubSpace Identification (page 143)
- TLS: Total Least Squares (page 185)

# Contents

[1]The simulations were carried out by Mr. Hans-Christian Pfisterer
[2]The simulations were carried out by Mr. Olli Haavisto, M.Sc.

# A Good Book Must Have Obscure Quotes

A theory has only the alternative of being wrong. A model has a third possibility — it might be right but irrelevant.

— M. Eigen

Statistics in the hands of an engineer are like a lamppost to a drunk — they're used more for support than illumination.

— B. Sangster

Like other occult techniques of divination, the statistical method has a private jargon deliberately contrived to obscure its methods from non-practitioners.

— G. O. Ashley

Make the model as simple as possible — but not simpler!

— A. Einstein

There are three kinds of lies: lies, damned lies, and statistics.

— B. Disraeli

> In earlier times, they had no statistics, and so they had to fall back on lies. — Stephen Leacock

Torture the data long enough and they will confess to anything.

— unknown

> If at first it doesn't fit, fit, fit again.
>
> — J. McPhee

Data! Data! Data! I can't make bricks without clay!

— S. Holmes

...

Why's the bell-shaped curve called normal?
Is it normal to be so formal?
There's nothing mean about the mean.
Its just average, as is clearly seen.
And what's so standard about that deviation?
It's a really malicious creation.
Confusing students is its only function.
It frustrates and mystifies, in conjunction.
...

— *"On statistical terminology"* by C. Lation

# Lesson 1

# Introduction to Multivariate Modeling

Statistics tell the biggest lies, everybody knows that ...! — However, this is not *necessarily* true. It all depends on the user who is interpreting the results: If the statistical methods are applied appropriately, by somebody who understands their properties, excellent results are often reached. Gaining this understanding is the objective of this report.

## 1.1 About systems and models

The role of a *model* is to organize information about a given system. There are various questions that need to be answered when a model is being constructed — the selection of the modeling principle being perhaps the most significant, affecting all subsequent analysis. First, one can start from the physical first principles, constructing the model in the bottom-up fashion. However, this modeling style is very knowledge-intensive, and the scarcity of the domain-area experts is an ever increasing problem. And, what is more, the larger the system is, the more probable it is that such a model will *not really be suited for real use,* for analysis of the system behavior or for control design. To understand this, see Fig. 1.1: What is the dimension of this constant-volume system, what is the number of independent system state variables?

The intuition tells us that there are three independent state variables — the concentrations in each tank needs to be represented in the system model. The problem with qualitative analyses is that the relevance of different constructs is not at all judged. For example, assume that one of the tanks is negligible, so that its volume is small as compared to the other two tanks: This tank does not contribute in the system behavior, and the nominal three-state model is unnecessarily complex for practical use. On the extreme, all the system time constants may be negligible as compared to the sampling interval, and, seemingly, the system becomes static with no dynamics whatsoever. On the other hand, assume that the mixing is not perfect: If the tanks are not ideal mixers, the one-state-per-tank approach is no more sufficient, and one ends in

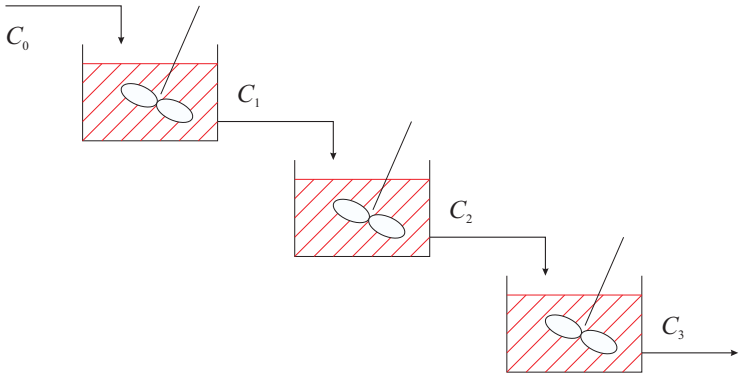



Figure 1.1: What is the dimension of the dynamic system?

a partial differential equation model. How many state variables are needed to reach sufficient accuracy is dependent of the actual system. The *relevant* dimension of the system can be anything between zero and infinity — and, indeed, for partial differential equation models, one can even speak of non-integer dimensions! Appropriate model structure is dependent of the intended use of the model.

As compared to bottom-up methods, the multivariate statistical methods operate in the top-down fashion: Plenty of data is collected and one tries to find the relevant system properties underlying the measurements. Instead of being knowledge-intensive, multivariate methods are *data-intensive*. The field of systems engineering is getting wider and wider, and the systems to be modeled are getting more and more complicated and less structured (see Fig. 1.2). All these trends necessitate data-oriented approaches in both ends of the systems continuum.

Another fact is that totally new branches of research — for example, *data mining* and *microactuators* must be based on massively parallel analysis and modeling methods.

The field of modern multivariate methods is wide and heterogeneous. Researchers in different disciplines typically have different objectives and application fields, and certainly they do have differing terminology and notations. What is more, the methods are still in turmoil and their overall relevance has not yet been generally understood. Some examples:

- Since the Gaussian times, least-squares mapping has been the standard technique in all fields of science. This methodology matured well before any data-orientation became a hot topic, and it seems that it is not typically seen in the wider perspective, in connection to other multivariate methods. It can be assumed that a great number of scientists and engineers suffer from its deficiencies, having to force their data into an unnatural least-squares-conditioned model structure, never getting acquainted with alternatives.

- In chemical engineering, for example, where calibration of devices is of utmost importance, a method called Partial Least Squares is routinely ap-

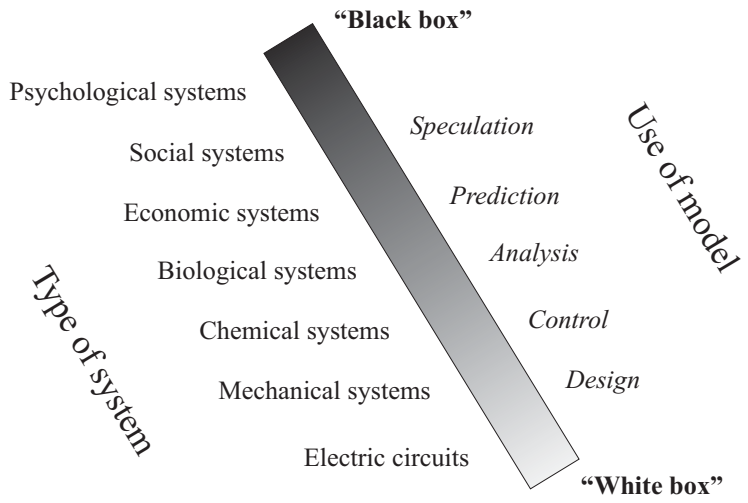



Figure 1.2: Spectrum of systems to be modeled

plied. However, being based on simple matching of correlations, employing unpenetrable algorithms, the uncompromising and ambitious mathematicians and statisticians are not very impressed or interested. Their answer to similar problems is Canonical Correlation Regression — a method that seems to be inaccessible for a practicing engineer. The unfortunate fact is that these mental barriers are caused simply by different terminologies and practices in these communities: The underlying ideas turn out to be closely related.

- The neural networks have become popular in almost all complex data modeling applications. In these research circles there seem to exist prejudices against the "outdated" statistical methods — but it is not only the "postmodernists" to blame: In the traditional school, there exist similar scornful attitudes towards the "heuristic" neural network methods. Again, beyond the surface, there is very much in common.

- Finally, in control engineering, dynamic models are often regarded as the only "interesting" models. However, the dynamic models can often better be understood when the simpler, static models are studied — indeed, dynamic modeling is static modeling with appropriately chosen data. On the other hand, static data samples are seldom independent of each other and dynamic understanding can reveal additional structure beyond that data, so that, again, it would be nice if these two approaches, static and dynamic, could be presented in a consistent setting.

It is difficult to see the underlying relationships among different approaches. An illustration of this heterogeneity in the field of data-oriented modeling is the fact that there seems not to exist an engineering level treatment of all relevant methods in a common framework. This report tries to do that: To offer a homogeneous view over the various disciplines in multivariate analysis.

## 1.2 About mathematical tools

Its is *mathematics* that is the natural language of nature. To fluently "think" using the syntactical structures defined in that language, the appropriate concepts need to be, not only familiar, but they have to belong to one's *active vocabulary.*

### 1.2.1 Challenge of high dimensions

In multivariate framework the structural complexity of the system is changed into a problem of high dimensionality: It is assumed that when one includes enough measurement data in the model, arbitrary accuracy concerning the system behavior can be achieved.

One needs to keep in mind the lesson learned in "Flatland" [1]:

> Assume that there existed *two-dimensional* creatures, being only able to recognize their environment in two dimensions. Now, assume that a three-dimensional ball goes through this two-dimensional world — what the creatures perceive, is a point coming from nowhere, expanding into a circle, and finally again contracting into a dot before vanishing. This circle can emerge anywhere in the two-dimensional space. How can the creatures understand what happened?

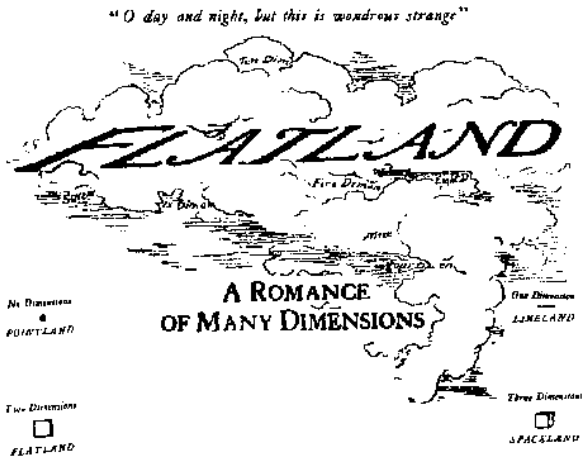



Figure 1.3: Cover page of E. Abbott's "Flatland"

The answer is that this behavior exceeds the capacity of those creatures, there is no way they could really understand it. The point is that *we* are bound to the three-dimensional world — the behavior of higher-dimensional objects similarly exceeds our capacity. Or, as J. Hadamard put it:

> *Give me 100 parameters and I will construct an elephant; give me 101 parameters, and I will make it wag its tail.*

This incomprehensibility is the basic problem when studying multivariate methods: The phenomena emerging in higher dimensions cannot really be visualized as two-dimensional plots (or even three-dimensional wire models). When more sophisticated methods are analyzed, illustrations do not help much, and common sense intuitions are of no use.

How the above problems with high data dimensionality are reflected in practice is perhaps best illustrated by an example: Assume that behaviors of a scalar (one-dimensional) function can be captured along a line; a two-parameter function spans the whole plane, and a three-parameter function spans the three-dimensional space. This means that to reach the same accuracy in each case, to cover the space equelly, the claim for data grows *exponentially.* To master the dimensional complexity, it is evident that one has to make strong assumptions to constrain the model structures.

Mathematics is a robust tool to attack the above challenges, offering stronger concepts and grammar for discussing multivariate phenomena.

The way to reach reasonable restrictions on the model structures, is to assume *linearity.* For linear models, essentially the same methodologies work no matter what is the dimension of the problem. This means that it is *linear algebra* that is the theoretical framework for studying multivariate statistics, and it is *matrix calculus* that is the practical language for implementing models for high-dimensional phenomena. Good understanding of these conceptual tools is vital.

### 1.2.2 About matrices

When doing multivariate modeling, data is (hopefully) received in huge numbers, and some kind of standardization of representations is necessary. It is assumed here that the only data structure that is employed is a *data matrix,* following the original `Matlab` style course of operation. The matrices will then have different roles: They are used as data storages, but also as frames for vector systems, and as linear operators representing linear mappings. In each case, it is *matrix operations* that are applied to manipulate the data structures.

The principles of matrix calculus are *not* repeated here (for more information, see, for example, [2] or [11]). It is assumed that *matrix inverses,* etc., are familiar; however, let us repeat what are *eigenvalues* and *eigenvectors,* what are *singular values,* and how matrix-form expressions are differentiated and how their extrema can be found. The discussion is restricted to real-valued matrices.

#### Eigenvalues and eigenvectors

It turns out that, rather astonishingly, most of the major regression methods can be presented in a homogeneous framework; this is the framework of the so called *eigenproblem.* A square matrix $M$ of dimension $n \times n$ generally fulfills the following formula for some $\xi$ and $\lambda$:

$$M \cdot \xi = \lambda \cdot \xi. \tag{1.1}$$

Here, $\lambda$ is a scalar called *eigenvalue* and $\xi$ is a vector called *eigenvector.* This means that the eigenvector directions are, in a sense, "natural" to the matrix $M$: Vectors in this direction, when multiplied by $M$, are only scaled, so that their direction is not changed (later, when speaking of linear mappings, this property will be elaborated on further). From the construction, it is clear that if $\xi$ is eigenvecter, then $\alpha\xi$ also is, where $\alpha$ is an arbitrary scalar. For uniqueness, from now on, it will be assumed that the eigenvectors are always normalized, so that $\|\xi\| = \sqrt{\xi^T\xi} = 1$ (this constraint is automatically fulfilled by the eigenvectors that are returned by the `eig` function of `Matlab`, for example).

In those cases that we will study later the matrices will be *non-defective* by construction (however, see the exercise); this means that there will exist $n$ distinct eigenvectors fulfilling the expression (1.1). These eigenvectors $\xi_i$ and corresponding eigenvalues $\theta_i$, where $1 \leq i \leq n$, can be compactly presented as matrices $\Xi$ and $\Lambda$, respectively:

$$\Xi = \left( \begin{array}{c|c|c} \xi_1 & \cdots & \xi_n \end{array} \right) \qquad \text{and} \qquad \Lambda = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix}, \tag{1.2}$$

where the dimension of matrices $\Xi$ and $\Lambda$ is $n \times n$. Using these notations, it is easy to verify that the $n$ solutions to the eigenproblem (1.1) can now be expressed simulateneously in a compact matrix form as

$$M \cdot \Xi = \Xi \cdot \Lambda. \tag{1.3}$$

In those cases that we will be later studying, the eigenvectors are linearly independent, $\Xi$ has full rank and the matrix $M$ is *diagonalizable:* The above expression equals

$$\Xi^{-1} \cdot M \cdot \Xi = \Lambda, \tag{1.4}$$

or

$$M = \Xi \cdot \Lambda \cdot \Xi^{-1}, \tag{1.5}$$

so that $M$ is *similar* to a diagonal matrix consisting of its eigenvalues. One of the specially useful properties of the above *eigenvalue decomposition* is due to the following:

$$\begin{aligned} M^i &= \left(\Xi \cdot \Lambda \cdot \Xi^{-1}\right)^i \\ &= \underbrace{\Xi \cdot \Lambda \cdot \Xi^{-1} \cdots \Xi \cdot \Lambda \cdot \Xi^{-1}}_{i \text{ times}} \\ &= \Xi \cdot \Lambda^i \cdot \Xi^{-1} \\ &= \Xi \cdot \begin{pmatrix} \lambda_1^i & & 0 \\ & \ddots & \\ 0 & & \lambda_n^i \end{pmatrix} \cdot \Xi^{-1}. \end{aligned} \tag{1.6}$$

That is, calculation of matrix powers reduces to a set of scalar powers. From this it follows that all matrix functions determined as power series can be calculated using their scalar counterparts after the matrix eigenstructure has been

determined. On the other hand, it is interesting to note that matrix functions *cannot* affect the matrix eigenvectors.

The eigenvalues can be determined also as the roots of the *characteristic equation*

$$\det\{\lambda \cdot I - M\} = 0. \tag{1.7}$$

Even though the eigenvalues should not be calculated this way, the roots of high-order polynomials being numerically badly behaving, some theoretical properties can easily be proven in this framework. For example, if a matrix of a form $q \cdot I$, where $q$ is scalar, is added to the matrix $M$, all of the eigenvalues are shifted by that amount:

$$\det\{(\lambda - q) \cdot I - M\} = 0. \tag{1.8}$$

The properties of the eigenvalues and eigenvectors will be discussed more when we know more about the properties of the matrix $M$.

### Singular value decomposition

The eigenvalue decomposition is defined only for square matrices (and not even all square matrices matrices can be decomposed in such a way). The generalization, the *singular value decomposition (SVD),* on the other hand, is defined[1] for all matrices $M$:

$$M = \Xi \cdot \Sigma \cdot \Psi^T. \tag{1.9}$$

Here $\Xi$ and $\Psi$ are orthogonal square matrices, so that $\Xi^T\Xi = I$ and $\Psi^T\Psi = I$, and $\Sigma$ is a diagonal matrix of *singular values.* Note that $\Sigma$ does not need to be square; if $M$ has dimension $\xi$ times $\zeta$, where $\xi > \zeta$ (and analogous results are found if $\xi < \zeta$), the decomposition looks like

$$\underset{\xi\times\zeta}{M} = \underset{\xi\times\xi}{\Xi} \cdot \left(\begin{array}{ccc} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_\zeta \\ \hline & \mathbf{0} & \end{array}\right) \cdot \underset{\zeta\times\zeta}{\Psi}^T. \tag{1.10}$$

The singular values $\sigma_i$ are characteristic to a matrix; they are positive real numbers, and it is customary to construct $\Sigma$ so that they are ordered in descending order. The singular values are close relatives of eigenvalues: Note that, because of the orthogonality of $\Xi$ and $\Psi$ there holds

$$M^T M = \Psi \cdot \Sigma^T\Sigma \cdot \Psi^T = \Psi \cdot \left(\begin{array}{ccc} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_\zeta^2 \end{array}\right) \cdot \Psi^T \tag{1.11}$$

[1] Here, the `Matlab` convention is followed: Matrices $\Xi$ and $Psi$ are kept invertible (square), meaning that generally $\Sigma$ is non-square. Other ways to define SVD can be found in other contexts

and

$$MM^T = \Xi \cdot \Sigma\Sigma^T \cdot \Xi^T = \Xi \cdot \left(\begin{array}{ccc|c} \sigma_1^2 & & 0 & \\ & \ddots & & \mathbf{0} \\ 0 & & \sigma_\zeta^2 & \\ \hline & \mathbf{0} & & \mathbf{0} \end{array}\right) \cdot \Xi^T. \tag{1.12}$$

These are eigenvalue decompositions of $M^TM$ and $MM^T$, respectively; this means that the (non-zero) eigenvalues of $M^TM$ (or $MM^T$) are squares of the singular values of $M$, the corresponding eigenvectors being collected in $\Psi$ (or $\Xi$, respectively).

Generalization of functions to non-square matrices can be based in the following matrix power definition, following the idea of (1.6):

$$M^i = \Xi \cdot \left(\begin{array}{ccc} \sigma_1^i & & 0 \\ & \ddots & \\ 0 & & \sigma_\zeta^i \\ \hline & \mathbf{0} & \end{array}\right) \cdot \Psi^T. \tag{1.13}$$

**Matrix differentiation**

Corresponding to the differentiation with respect to a scalar, we can differentiate a scalar-valued function $f(\cdot)$ with respect to a vector; the result is a vector called *gradient.* Assume that the function $f : \mathcal{R}^\zeta \to \mathcal{R}$ is being differentiated:

$$\frac{d}{d\,z} f(z) = \left(\begin{array}{c} \frac{d}{d\,z_1} f(z) \\ \vdots \\ \frac{d}{d\,z_\zeta} f(z) \end{array}\right). \tag{1.14}$$

Note that now we choose that gradients to be column vectors (in literature, this is not always the case). Assuming that the matrix $M$ has dimension $\zeta \times \xi$, its row dimension being compatible with $z$, so that there exists

$$z^TM = \left(\begin{array}{ccc} \sum_{i=1}^\zeta z_i M_{i1} & \cdots & \sum_{i=1}^\zeta z_i M_{i\xi} \end{array}\right), \tag{1.15}$$

the differentiation can be carried out columnwise:

$$\begin{array}{rcl} \frac{d}{d\,z}\left(z^TM\right) & = & \left(\begin{array}{ccc} \frac{d}{d\,z_1}\sum_{i=1}^\zeta z_i M_{i1} & \cdots & \frac{d}{d\,z_1}\sum_{i=1}^\zeta z_i M_{i\xi} \\ \vdots & \ddots & \vdots \\ \frac{d}{d\,z_\zeta}\sum_{i=1}^\zeta z_i M_{i1} & \cdots & \frac{d}{d\,z_\zeta}\sum_{i=1}^\zeta z_i M_{i\xi} \end{array}\right) \\ & = & \left(\begin{array}{ccc} M_{11} & \cdots & M_{1\xi} \\ \vdots & \ddots & \vdots \\ M_{\zeta 1} & \cdots & M_{\zeta\xi} \end{array}\right) \\ & = & M. \end{array} \tag{1.16}$$

This is how one would expect a linear matrix function to behave. On the other hand, it turns out that if $z$ is multiplied from the other side, the matrix has to be transposed:

$$\frac{d}{d\,z}\left(M^T z\right) = M. \tag{1.17}$$

Thus, using the product differentiation rule,

$$\begin{array}{rcl} \frac{d}{d\,z}\left(z^T M z\right) & = & \left(\frac{d}{d\,z}\left(z^T M \bar{z}\right) + \frac{d}{d\,z}\left(\bar{z}^T M z\right)\right)\Big|_{\bar{z}=z} \\ & = & \left(M + M^T\right) z. \end{array} \tag{1.18}$$

Here, $M$ must have dimension $\zeta \times \zeta$; $\bar{z}$ is assumed to be a (dummy) constant with respect to $z$. For symmetric $M$ the above coincides with $2Mz$, something that looks familiar from scalar calculus.

It turns out that more sophisticated differentiation formulas are not at all needed later in this report.

### 1.2.3 Optimization

Matrix expressions can be optimized (minimized or maximized) similarly as in the scalar case — set the derivative to zero (this time, "zero" is a vector consisting of only zeros):

$$\frac{d}{d\,z} J(z) = \mathbf{0}. \tag{1.19}$$

For matrix functions having quadratic form (like $x^T Ax$) the minima (maxima) are unique; this is the case in all optimization tasks encountered here. For an extremum point to be maximum, for example, the (hemo) *Hessian matrix* must be negative semidefinite:

$$\frac{d^2}{d\,z^2} J(z) = \frac{d}{d\,z}\left(\frac{d}{d\,z} J(z)\right)^T \leq \mathbf{0}. \tag{1.20}$$

Note the transpose; the other gradient must be written as a row vector, so that the final result would be a square matrix. Here "$<$" has to be interpreted (both sides being matrices) so that

$$\xi^T \cdot \left(\frac{d^2}{d\,z^2} J(z)\right) \cdot \xi \leq 0 \tag{1.21}$$

for any (compatible) vector $\xi$. However, the above approach only works if there are no constraints to be taken care of in the optimization.

#### Pseudoinverse (case I)

As an example, study the least squares solution when there does not exist any exact solution.

Assume that there holds $x = \theta z + e$, $x$ and $\theta$ being fixed, and one wants to solve $z$ so that the norm of the error vector $e$ is minimized. It is now assumed that the dimension of $z$ is *lower* that that of $x$; this means that the solution generally has no exact solution. The criterion to be minimized, the square of the norm of $e$ is

$$\begin{aligned} J(z) &= e^T e = (x - \theta z)^T (x - \theta z) \\ &= x^T x - x^T \theta z - z^T \theta^T x + z^T \theta^T \theta z. \end{aligned} \tag{1.22}$$

Differentiating one has

$$\frac{d}{d\,z} J(z) = -\theta^T x - \theta^T x + 2\theta^T \theta z = 0, \tag{1.23}$$

so that one can solve

$$\theta^T \theta z = \theta^T x. \tag{1.24}$$

This is called the *normal equation* — it can further be solved if explicit formula for $z$ is needed:

$$z = \left(\theta^T \theta\right)^{-1} \theta^T x. \tag{1.25}$$

### 1.2.4 Lagrange multipliers

The method of *Lagrange multipliers* is a generic method for solving constrained optimization problems, assuming that the functions involved are continuously differentiable. It must be recognized that this method gives necessary, not sufficient conditions for optimality.

Assume that one should find the maximum (or minimum) of the function $f(z)$, so that there holds $g(z) = 0$. The idea of the Lagrangian method is visualized in Fig. 1.4: at the optimum point, the gradients of $f$ and $g$ must be parallel. In the case of scalar $z$ this means that the gradient of $g$ at $z$ must be a scalar multiple of the gradient of $f$. For vector $z$, analogously, there must exist constant vector $\lambda$ so that

$$\frac{df(z)}{dz} = \lambda^T \cdot \frac{dg(z)}{dz}, \tag{1.26}$$

or, written in the standard form,

$$\frac{d}{dz}\left(f(z) - \lambda^T \cdot g(z)\right) = \mathbf{0}. \tag{1.27}$$

#### Pseudoinverse (case II)

As an example, study the least squares solution when there is a multitude of possible solutions available.

Assume that there holds $x = \theta z$, and we want to solve $z$ when $x$ and $\theta$ are fixed. Further, assume that the dimension of $z$ is now *higher* that that of $x$; this

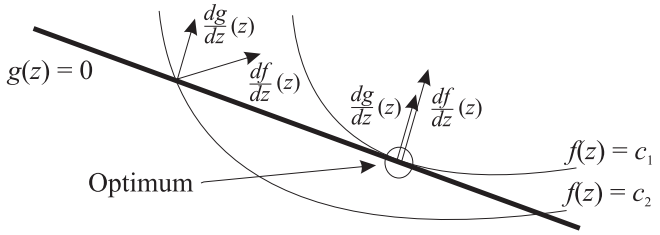



Figure 1.4: Illustration of the optimality condition: in the extremal point the gradients of $f$ and $g$ are parallel

means that the solution is not generally unique. To formulate a well-defined problem, assume that out of the set of candidate solutions, we want to select the "simplest" in the sense of vector size; that is, the criterion to be minimized is $z^T z$.

Now we have the optimality defined in terms of $f(z) = z^T z$ and the constraint function is $g(z) = x - \theta z$, so that

$$\frac{d}{dz}\left(z^T z - \lambda^T \cdot (x - \theta z)\right) = \mathbf{0} \tag{1.28}$$

gives

$$2z + \theta^T \lambda = \mathbf{0}, \tag{1.29}$$

and, further, $z = -\frac{1}{2} \cdot \theta^T \lambda$. From $x = \theta z$ we now get $x = -\frac{1}{2} \cdot \theta\theta^T \lambda$ or $\lambda = (-\frac{1}{2} \cdot \theta\theta^T)^{-1} \cdot x$. Finally, combining this and $z = -\frac{1}{2} \cdot \theta^T \lambda$, one has the least squares solution

$$z = \theta^T \left(\theta\theta^T\right)^{-1} \cdot x. \tag{1.30}$$

In this section, we have found two expressions for the solution of the least-squares problem in different cases. These can be expressed using the so called *pseudoinverse:*

$$\theta^\dagger = \theta^T \left(\theta\theta^T\right)^{-1}, \tag{1.31}$$

or

$$\theta^\dagger = \left(\theta^T\theta\right)^{-1} \theta^T, \tag{1.32}$$

whichever is appropriate; anyway the least-squares solution is given as $z = \theta^\dagger x$.

Note that it is possible that *neither* of the above forms of pseudoinverse is defined, both $\theta^T\theta$ and $\theta\theta^T$ being rank deficient; in such case more general approach to defining pseudoinverse is needed. The *general* pseudoinverse can be calculated utilizing the singular value decomposition (for example, see "`help pinv`" in `Matlab`).

## Computer exercises

1. Study the `Matlab` commands `eig` and `svd` (command "`help eig`", etc.). Define the matrix

$$M = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix},$$

and construct the eigenvalue and singular value decompositions

```
[XiEIG,Lambda] = eig(M);
[XiSVD,Sigma,PsiSVD] = svd(M);
```

Study the matrices and explain the results you have when you try

```
XiEIG*Lambda*inv(XiEIG)
XiSVD*Sigma*inv(PsiSVD)
```

Repeat the above with the matrices $M^T M$ and $MM^T$. Comment the results in the case where the matrix is defined as

$$M = \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

2. Download the `Regression Toolbox` for `Matlab` from the Internet address `http://saato014.hut.fi/hyotyniemi/publications/01_report125.htm`, and install it following the guidelines presented on page 229. This Toolbox will be used later for illustrating the theoretical derivations.

# Lesson 2

# About Distributions

When modeling large amounts of data, individual samples are of no special relevance. To construct *relevant* models, one has to reach the "big picture" beyond the surface. The relevance is captured in the statistical general properties of the whole data set; these statistical properties are represented by *probability distributions.* After all, it is distributions that are being modeled by the data-oriented methods.

To find appropriate methods for data modeling, it is also necessary to first study the properties of data distributions. Later, however, the statistical considerations can be ignored: Assumptions concerning the data-generating procss make it possible just to concentrate on some emergent distribution characteristics, like variance and covariance. The existence of the underlying distributions is reflected in the modeling methods and resulting model structures. In this chapter, the basic model structures are motivated in statistical terms, and their correspondence with real data is discussed.

## 2.1 Data mining

When facing something new, it is clever to first look it from a distance, from different points of view. It is the same with data: Before starting any harder labor, it is clever to gain insight. There are efficient and innovative data analysis tools available where the computing power available today is utilized to reveal different ways to see the data.

The diversity of data mining approaches and tools is not surveyed here. Only as an example, in Fig. 2.1 industrial data is visualized applying the *Self-Organizing Map* or *SOM* (see [**?**]; also see Sec. 8.1.1). SOM efficiently utilizes the human pattern recognition capability: The data is typically projected onto a two-dimensional surface, so that the dependencies among data are visually manifested. However, *computer is notoriously bad in such pattern recognition tasks;* SOM is a good front-end for humans, but not for implementing some machine-to-machine (or "algorithm-to-algorithm") interaction.

Such SOM models can directly be used, and they have been used, also directly

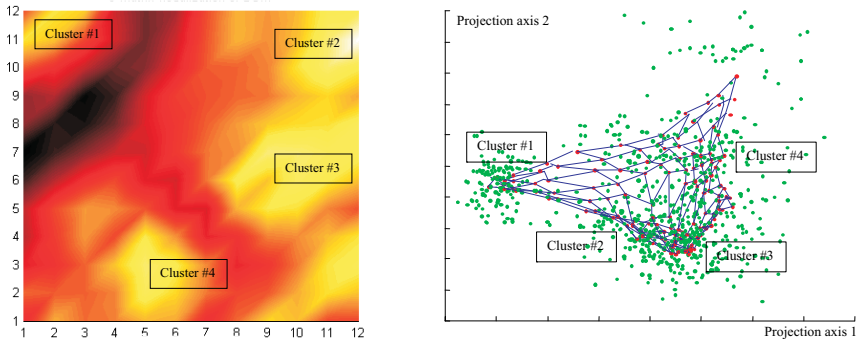



Figure 2.1: How the character of the data can be visualized: The SOM approach. High-dimensional industrial data has been projedcted onto a two-dimensional manifold or "hypersurface", so that the *topology* among the data has been maximally preserved. On the left, the $12 \times 12$ SOM grid is shown: The regions of many "hits" have been printed with lighter color. On the right, the converged SOM map itself has been projected into the original variable space, showing its curved nature. It seems that there are perhaps four (or more?) separate concentrations of data, or *clusters,* perhaps revealing something about the variability in the operating regimes in the process

for process monitoring purposes, etc., but when implementing prediction or control, SOM should be seen as a pre-analysis tool only. SOM implements extreme compression, mapping data from high-dimensional continuous-valued variable space onto a discrete set of map nodes, so that unavoidably very much of the available information is lost. Better regression can be implemented if the intuition offered by SOM is exploited for adjusting the more traditional modeling methods.

It also needs to be mentioned that when the data is high-dimensional, the wonders of high dimensions can look too fancy for the inhabitants of Flatlands. The higher the dimension, the more there exist alternative explanations for the observations, at least if the evidence is interpreted in an appropriate way ... One should remember the "Barnum effect" and recognize that *You see what you want to see.*

## 2.2 Normal distribution

The *Gaussian* or *normal distribution* is the most important abstraction for more or less stochastic measurement data. The famous *central limit theorem* states that if a large number of independent random variables are added together, the sum is normally distributed under very general conditions, no matter what is the distribution of the original variables. Usually, when making process measurements, it can be assumed that underlying the actual measurement values there is a large number of minor phenomena that cannot be separately analyzed

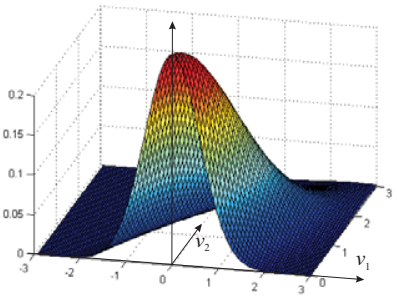



Figure 2.2: Visualizing the density of the two-dimensional Gaussian distribution: In surface form ...

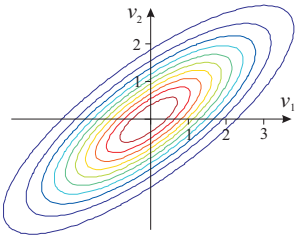



Figure 2.3: ... and as a contour map. Here the covariance matrix was such that $r_{11} = r_{22} = 1$ and $r_{12} = r_{21} = 0.8$

— their net effect, according to the central limit theorem, is that the overall distribution becomes normal.

Similarly as in the one-dimensional case, *multinormality* holds for multivariate data (see Figs. 2.2 and 2.3). Let $v$ stand for the measurement vector of length $\dim\{v\}$. Assuming multinormal distribution, the density function value (corresponding to the probability; note that finite probabilities are found only when the density function is integrated within some region in $v$ space) for a data sample $v$ can be calculated as

$$p(v) = \frac{1}{\sqrt{(2\pi)^{\dim\{v\}} \det\{R\}}} \mathrm{e}^{-\frac{1}{2}\cdot(v-\bar{v})^T R^{-1}(v-\bar{v})}. \tag{2.1}$$

Here, $\bar{v}$ stands for the center of the distribution and $R$ is the covariance matrix, $\det\{R\}$ being its determinant. This prototypical distribution can be compactly denoted as $\mathcal{N}\{\bar{v}, R\}$. The distribution formula consists essentially of a (decaying) exponential function, making the "bell-shaped" distribution extend to infinity in all directions; due to the normalization factor, its integral over the whole space equals 1. The statistical properties of multinormal distributions are not elaborated on in this context; it suffices to note that all projections of a normal distribution are also normal, and, generally, linear functions of normally distributed data result in normal distributions (see Figs. 2.2 and 2.3).

It needs to be noted that above it is all variables that are assumed similarly stochastic. Traditionally when doing modeling and identification, there is a distinction between deterministic and stochastic variables; there are inputs and outputs; there is noise and there is information. Now, the framework is homogeneous: All variables have the same stochastic nature to begin with. This means that methodologies for analysing the variables also remain uniform. The information is assumed to be buried in correlations among the variables.

### 2.2.1 About distribution parameters

Multinormal distribution is uniquely determined by its mean and covariance. If there are measurements $v(1)$ to $v(k)$ taken from the distribution, the unbiased mean, $\bar{v} = \mathrm{E}\{v(\kappa)\}$, can be approximated as the *sample mean*

$$\bar{v} = \frac{1}{k} \cdot \sum_{\kappa=1}^{k} v(\kappa). \tag{2.2}$$

The covariance matrix, $R = \mathrm{E}\{v(\kappa)v^T(\kappa)\}$, can be approximated as *sample covariance*

$$R = \frac{1}{k} \cdot \sum_{\kappa=1}^{k} (v(\kappa) - \bar{v})(v(\kappa) - \bar{v})^T, \tag{2.3}$$

or, if the individual sample vectors $v$ are collected as rows in the $k \times \dim\{v\}$ matrix $V$,

$$R = \frac{1}{k} \cdot (V - \bar{V})^T (V - \bar{V}). \tag{2.4}$$

The matrix $\bar{V}$ now consists of $k$ copies of $\bar{v}^T$. It is assumed that the covariance matrix has full rank and it is invertible; this means that necessarily there must hold $k \geq \dim\{v\}$ (there are at least as many data vectors as there are separate measurements in the measurement vector) and the measurements $V_1$ to $V_{\dim\{v\}}$ are *linearly independent.*

Note that the presented estimate for covariance that is based on the estimate of the sample mean is *biased;* one should take into account the reduced degrees of freedom to find the unbiased estimate (that is, the denominator should read $k - \dim\{v\}$). However, it is not always clear what is the theoretically appropriate normalizing factor (for example, if calculating the cross-correlation $X^T Y$, where $X$ is a $k \times n$ matrix and $Y$ is a $k \times m$ matrix). In what follows, it is assumed that the number of measurements is so high that this bias can be neglected (that is, $k \gg \dim\{v\}$). Later, it turns out that it is the covariance matrix that plays a central role when determining the model structure — and it is the *structure* of the covariance matrix that is of relevance, ratios between elements, revealing the interconnections among variables, not its *scaling.*

The covariance matrix is such a central data construcy in subsequent analyses that it deserves a still closer look — it is still *intuition* that plays a central role when constructing good models. Understanding the structure of the covariance matrices, and understanding how this structure is related to data properties, is fundamental knowledge when trying to understand multivariate statistical methods.

As visualized in Fig. 2.3, the covariance structure can be visualized in terms of (hyper)ellipsoids in the data space: The ellipsoids represent the “equi-probability” surfaces in the data space. The projections of Gaussians onto lower dimensions (also having Gaussian distribution) can be visualized es ellipses.

| | About mean | About origin |
|---|---|---|
| **Unnormalized** | Covariance matrix | Inner product matrix |
| **Normalized** | Correlation matrix | Cosine matrix |

Figure 2.4: Some association matrices

It is instructive to interpret the covariances in terms of concrete ellipses, end here are some guidelines to interpret them. The variances of the individual variables that are collected on the diagonal of the covariance matrix dictate how far the ellipsoid extends in that variable direction; zero variance means that the ellipsoid "collapses" into a (hyper)planar structure. The non-diagonal elements in the covariance matrix reveal how much the ellipsoid is "tilted" as compared to the variable axes. This "tiltedness" connects the variables together, variables becoming dependent, and it is indeed these dependency structures, cross correlations, that make it possible to estimate the values of some variables when some other variables only are known, making regression analysis feasible. However, the properties of such tilted ellipsoids cannot be seen in the original coordinate frame, and more closer analyses have to be postponed to the eigenvalue/eigenvector analysis of the covariance matrix in Chapter 5. It turns out that the extent of the ellipsoid in different directions (as determined by the eigenvectors of the covariance matrix) is revealed by the square roots of the corresponding eigenvalues; the "volume" of the ellipsoid is proportional to the product of the eigenvalues.

### 2.2.2 Association matrices

The covariance matrix reveals the second-order properties of the data (variances and co-variances) in a compact form, and it turns out that it is these second-order properties that one concentrates on in multivariate modeling. It turns out that *determination of the model structure is based on the analysis of the data covariances.* However, there also exist other ways to capture the second-order properties.

Covariance measures *similarity* between variables, and it makes it possible to define *associations* among them. Generalizing slightly, rather than speaking merely of covariance matrices, we can speak of *association matrices.* The idea is the same: the second-order "nearness" properties between variables should be captured in a compact form so that the assumedly relevant phenomena would become tractable. Fig. 2.4 shows some common selections that are found when the data either *is* centralized or it is not, and when the data either *is* normalized to unit variance or it is not (there will be more about data preprocessing in the next chapter). In all of the above cases, the association matrices are constructed as

$$R = \frac{1}{k} \cdot \sum_{\kappa=1}^{k} x'(\kappa) x'^{T}(\kappa) = \frac{1}{k} \cdot X'^{T} X' \tag{2.5}$$

where $x'$ is the correspondingly scaled (and perhaps centered) data sample. Again, if being theoretically orthodox, one would have problems with the nor-

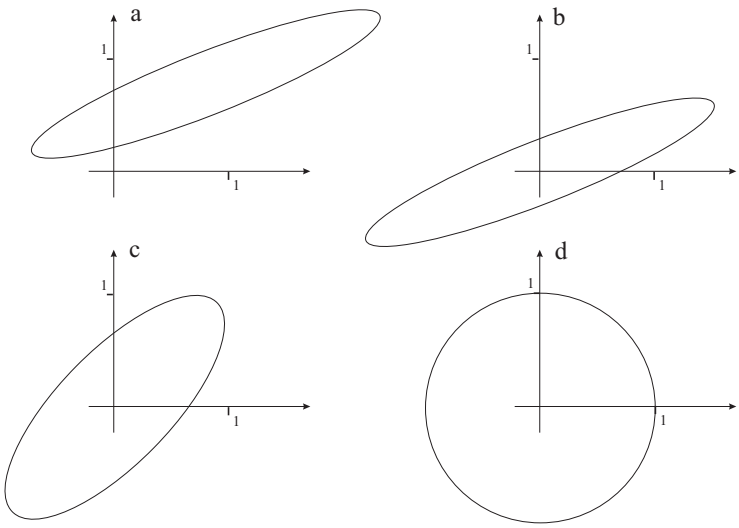



Figure 2.5: Data distributions after different preprocessing operations. First, in **a**, the assumed original data distribution is shown, and in **b** data is *centered.* In **c** data is additionally *normalized* to unit variance, and in **d**, it is *whitened* (in addition to being centered)

malization factor: If the origin is now assumed to be the "center" of data the degrees of freedom are not reduced?

All of the above association matrices are positive semi-definite, that is, $\xi^T R \xi \geq 0$ for any vector $\xi$:

$$\xi^T R \xi = \frac{1}{k} \cdot \sum_{\kappa=1}^{k} \xi^T x'(\kappa) x'^T(\kappa) \xi = \frac{1}{k} \cdot \sum_{\kappa=1}^{k} \left(\xi^T x'(\kappa)\right)^2 \geq 0. \tag{2.6}$$

This means that all eigenvalues are non-negative. Note that when discussing general association matrices one is violating the basic assumptions concerning covariance matrices on purpose: One is no more analyzing the properties of the original Gaussian distribution but some *virtual* distribution. "Forgetting" the centering, for example, has major effect on the data distribution as seen by the algorithms. In Fig. 2.5, the effects of different preprocessing operations (see Chapter 3) on the data distribution are shown.

There are also other possibilities for constructing matrices that are related to similarity matrices — for example, the *distance matrix,* where the element $R_{ij}$ is the distance (Euclidean or other) between vectors $X_i$ and $X_j$, can be used for structuring the relationship between variables (note that the diagonal contains zeros, making this matrix to be *not* positive definite); also see Sec. 7.3.3. The *Kernel matrices* are yet another of representing (nonlinear) relationships between variables (see Appendix 2). When similarity is measured in some feature space, so that one applies similarity matrices of the form $\mathrm{E}\{f(x)f(x)^T\}$ for analysis, where the features are determined through the nonlinear vector-valued function $f$, one sometimes speaks of *nonlinear component analysis* or *kernel PCA* (compare to Chapter 5). Indeed, determination of the function $f$,

or feature extraction, is discussed in the next chapter.

Depending on the situation, it can be motivated to study the connections among *samples* rather than among *variables,* that is, rather than finding the structure for $X^T X$, one can search for the structure of the matrix $XX^T$. Note that the non-zero eigenvalues are the same in both cases.

The data can also be scaled samplewise; If there holds $y(\kappa) = F^T \cdot x(\kappa)$, then, for some scalar function $g(x(\kappa), y(\kappa), \kappa)$, there must also hold

$$g\, y(\kappa) = F^T\, g\, x(\kappa), \tag{2.7}$$

and these scaled variables can be just as well be used for determining $F$. Even though the expression above looks like an identity, the statistical properties of the data may be changed remarkably when the samples are individually scaled (see Sec. 7.3.2): This kind of "samplewise" scaling can also be justified if one knows that different samples have different levels of reliability — or if the noise variance level varies along the sampling; this is sometimes called *heterosedasticity.* Specially, assume that $g(\kappa)$ is a function of time index $\kappa$ alone, and study the properties of the correlation matrix:

$$R = \frac{1}{\sum_\kappa} \cdot \sum_{\kappa=1}^{k} g(\kappa)\, v(\kappa) v^T(\kappa). \tag{2.8}$$

Here, the normalizing factor compensates for the scaling effect of the sequence of the weighting factors. Further, assuming that one wants to apply *exponential forgetting,* so that the "memory" gradually fades away, one can select $g(\kappa) = \lambda^{k-\kappa}$, where $0 \ll \lambda < 1$ is the forgetting factor, one can write the recursive adaptation rule for the covariance estimate in the familiar-looking form (mathematical interpretations ranging from weighted-average to convex-combination):

$$R(k) = \lambda\, R(k-1) + (1-\lambda)\, v(k) v^T(k). \tag{2.9}$$

### 2.2.3 $\chi^2$ distribution

Multivariate normal distribution (2.1) gives a probability of any point to belong to a Gaussiann distribution. However, in a high-dimensional space the probability of *any location* becomes very low — one would like to have a scalar measure for easily studying whether an observation is characteristic to a distribution or not, regardless of the data dimension. It turns out that the $\chi^2$ *distribution* is a practical tool for this purpose.

If there are $n$ independent, normally distributed normalized variables $v_i$, where $1 \le i \le n$, *the sum of the squares, or* $v^T v$, *has* $\chi^2$ *distribution with degrees of freedom* $n$. This sounds like a rare special case, but this is not so. Study the case where Gaussian variable vectors $\nu$ are normalized so that

$$v(\kappa) = R_\nu^{-1/2} \nu(\kappa), \tag{2.10}$$

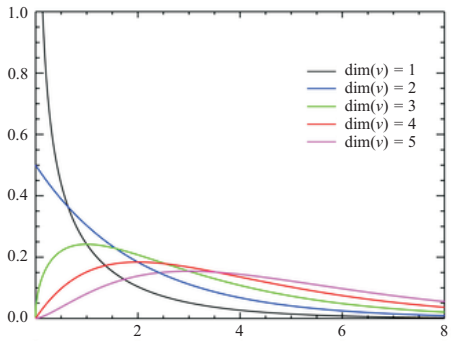



Figure 2.6: Note that in higher dimensions the maximum of $\chi^2$ distribution is not in zero!

where

$$R_\nu = \frac{1}{k}\sum_{\kappa=1}^{k} \nu(\kappa)\nu^T(\kappa). \tag{2.11}$$

Then the new variables $v$ are Gaussian and there holds

$$\begin{aligned} R_v &= \tfrac{1}{k}\sum_{\kappa=1}^{k} v(\kappa)v^T(\kappa) = R_\nu^{-1/2}\tfrac{1}{k}\sum_{\kappa=1}^{k} \nu(\kappa)\nu^T(\kappa)\; R_\nu^{-T/2} \\ &= R_\nu^{-1/2}R_\nu R_\nu^{-T/2} = I. \end{aligned} \tag{2.12}$$

This means that the familiar expression has $\chi^2$ distribution:

$$v^T(\kappa)v(\kappa) = \nu^T(\kappa)R_\nu^{-T/2}R_\nu^{-1/2}\nu(\kappa) = \nu^T(\kappa)R_\nu^{-1}\nu(\kappa). \tag{2.13}$$

This $\nu^T R_\nu^{-1}\nu$ is a quantity that is routinely computed in multivariate analysis, and it makes it possible to reduce the high-dimensional distribution into one dimension. The $\chi^2$ distribution can be found, for example, in `Matlab`; to use the functions there, one needs to determine the degrees of freedom, or the number $n$ (see Fig. 2.6). In the Regression Toolbox, there is the function `regrP` that is tailored for course usage.

## 2.3 Motivation of modeling approaches

After this chapter, the distributions are abstracted away — one only concentrates on a single distribution parameter, (co)variance, forgetting about the other distribution properties. Calculation of variance and covariance can be carried out for any set of data, regardless of the actual distribution, and, similarly, the regression models that are presented later being based on covariance properties can be constructed. However, the Gaussianity assumption is implicitly buried in the model structures: As it turns out, the adopted modeling principles are not only pragmatically motivated, they are *optimal* for the Gaussian distribution.

### 2.3.1 Why linear models?

Let us study some special properties of the Gaussian distribution. The "most probable" regions in the data space are determined by the formula (2.1). For simplicity, assume that data is zero-mean, $\bar{v} = 0$. Because the exponent function is a monotonously increasing function, the maximum of probability is reached when the following expression reaches minimum:

$$J = v^T R^{-1} v \tag{2.14}$$

To proceed, one has to distinguish between the roles of individual variables in $v$. As explained in the next chapter, it is reasonable to separate the *input variables* and *output variables* from each other. If it is assumed that some of the variables in $v$, collected in the vector $x$, are known, and some, collected in $y$, are unknown, so that

$$v = \begin{pmatrix} x \\ \hline y \end{pmatrix}, \tag{2.15}$$

expression (2.14) can be divided in parts:

$$J = \begin{pmatrix} x \\ \hline y \end{pmatrix}^T \left(\begin{array}{c|c} (R^{-1})_{xx} & (R^{-1})_{xy} \\ \hline (R^{-1})_{yx} & (R^{-1})_{yy} \end{array}\right) \cdot \begin{pmatrix} x \\ \hline y \end{pmatrix}, \tag{2.16}$$

or, written explicitly,

$$J = x^T (R^{-1})_{xx} x + x^T (R^{-1})_{xy} y + y^T (R^{-1})_{yx} x + y^T (R^{-1})_{yy} y. \tag{2.17}$$

Here the matrices $(R^{-1})_{xx}$, etc., are formally used to denote the blocks of the inverse covariance matrix; how they should actually be constructed is not of interest here. Minimization with respect to $y$ means solving

$$\begin{aligned} \frac{dJ}{dy} &= \frac{d}{dy}\left(x^T (R^{-1})_{xx} x + x^T (R^{-1})_{xy} y + y^T (R^{-1})_{yx} x + y^T (R^{-1})_{yy} y\right) \\ &= \mathbf{0}, \end{aligned}$$

giving a unique solution:

$$((R^{-1})_{xy})^T x + (R^{-1})_{yx} x + ((R^{-1})_{yy})^T y + (R^{-1})_{yy} y = \mathbf{0}, \tag{2.18}$$

or

$$y = \left(((R^{-1})_{yy})^T + (R^{-1})_{yy}\right)^{-1} \left(((R^{-1})_{xy})^T + (R^{-1})_{yx}\right) x. \tag{2.19}$$

This can be expressed in a very simple form

$$y = Mx. \tag{2.20}$$

It is not of interest here to study any closer the matrices that constitute the solution, or what is the structure of the matrix $M$; these issues will be concentrated on later in detail. What is crucial is the basic outlook of the maximum

likelihood (ML) solution for the regression problem: *The unknown variables are linear functions of the known ones.* Within a Gaussian distribution, linear estimates are optimal — this is a very useful result, justifying the simple model structures that will be applied later.

To be exact, assuming that the distribution is not zero-mean, the general maximum likelihood relationship between variables becomes *affine:*

$$y = Mx + c, \tag{2.21}$$

where $c$ is a constant vector. However, models will be assumed strictly linear later — the techniques to avoid problems that are faced because of this assumption will be discussed in the next chapter.

### 2.3.2 Why sum-of-error-squared criteria?

Continuing from the above linear model structure, assume that there exists such a matrix $M$ that maps $x$ onto $y$, so that (2.20) is assumed to apply, and one's task is to determine this mapping matrix. Typically (if $k > n$) exact matching cannot be reached, so that for each sample $\kappa$ there remains a residual error

$$e(\kappa) = y(\kappa) - Mx(\kappa). \tag{2.22}$$

If the data is Gaussian, also this error has Gaussian distribution. Further, assume that the errors in the sequence $e(\kappa)$ are independent of each other, and have identical Gaussian distribution with mean $\bar{e} = 0$ and covariance $R_e$. The best choice for the matrix $M$ maximizes the probability that the observed sequence of samples has been obtained — that is, the probabilities of observing the sequence $e(\kappa)$ should be maximized. Because the individual errors were assumed independent, the overall probability is the product of individual probabilities, so that the *likelihood function* now becomes

$$\begin{aligned} L &= \prod_{\kappa} p(e(\kappa)) \\ &= \frac{1}{\sqrt{2\pi \det\{R_e\}}} \mathrm{e}^{-\frac{1}{2}\sum_{\kappa}(y(\kappa)-Mx(\kappa))^T R_e^{-1}(y(\kappa)-Mx(\kappa))} . \end{aligned} \tag{2.23}$$

Because the logarithm function is monotonously increasing, the maximum of the above criterion equals the minimum of the following:

$$J = -\log L = c + \sum_{\kappa=1}^{k} \left(y(\kappa) - Mx(\kappa)\right)^T R_e^{-1} \left(y(\kappa) - Mx(\kappa)\right). \tag{2.24}$$

for scalar $y$, this reduces essentially to a sum of squared errors, $J$ is proportional to $\sum_\kappa e^2(\kappa)$. This all means that the criterion that makes it possible to find solutions in a mathematically closed form, is again *optimal* for Gaussian data.

There are also many other reasons for selecting the error-squared criterion: From the theoretical point of view, it is nice that the minimum of the quadratic criterion is unique, so that no closer analyses of candidate solutions is needed;

from the practical point of view, it is nice that this criterion has rather natural interpretations in terms of signal powers, error squares are related to noise variances, capturing the essence of the noise distributions, etc. However, in some cases the emphasis on the error squares is clearly a disadvantage: This is the case specially if there exist large spurious variations in the data (perhaps caused by undetected outliers, etc.) — such samples are emphasized excessively in the model construction because of the error-squared criterion.

Often errors in different variables are more critical than in others; however, the error-squared criterion assumes that all errors are equally significant. It is the user's task to assure that this equality assumption is justified; this can be carried out by appropriate scaling of the variables during the preprocessing. If the variables are scaled up, also the errors in those variables are emphasized accordingly.

## 2.4 Tackling with real-world data

Gaussianity assumption is well-motivated, due to the Central Limit Theorem. However, despite the above optimistim, the things are not so simple in practice.

The real measurement data seldom is purely Gaussian. There are various reasons for this: First, normally distributed data that goes through a nonlinear element is no more Gaussian; second, the measurement samples may be generated by different underlying processes, constituting no single distribution at all. All these phenomena can be explained as different kinds of nonlinearities in the system. If the Gaussianity assumption has to be abandoned, what kind of model structure to adopt instead?

The selection of the model structure is always a compromise between two things: The model should fit the data well, but, at the same time, the model should suit the user's needs, being easily applicable and analyzable. The first of the objectives — matching the data — generally means that complex models should be used, but, on the other hand, the second objective favors overall simplicity. There are no final truths available here, but it turns out that a nice conceptual compromise between real data properties and theoretical preferences is given by the *Gaussian mixture model* of data.

### 2.4.1 Gaussian mixture models

Non-Gaussianity of a distribution is a symptom of nonlinearity somewhere along the data generation processes. As it was observed in the previous chapter, nonlinearity in high dimensions is a problem defying analyses. But assuming that the nonlinearities can be locally linearized, the function can approximately be substituted with a set of linear functions — and the complex distribution can approximately be substituted with a set of appropriately located Gaussians. Such a collection of Gaussian subdistributions is called *Gaussian mixture model.* Assuming smoothness of functions, nearby samples are related; but the farther apart in the data space the samples are, the less they are assumed to be related, or assumed to contribute to the same model: Thus, there are different (linear) submodels for different clusters. In Fig. 2.1, there exist, say, 4 or 5 data clus-

ters; It is also assumed here that these subdistributions can be approximately characterized by Gaussians.

It seems that the typical nonlinearities can be attacked using a two-level strategy: First, find the set of appropriate clusters $\Gamma$, whatever phenomenon has given rise to such clustering, and, after that, apply linear methods for modeling within each of the clusters $c \in \Gamma$ separately.

However, when constructing regression models, one is not interested in the clusters, but one would like to have (continuous) mappings between variables. Gaussianity (or, indeed, any compact distribution model) as the model for subdistributions gives a consistent way of getting back from discretized (clustered) coding of data to smooth and continuous (nonlinear) input/output functions. Assume that $p_c(\kappa)$ is the probability of sample number $\kappa$ to belong in the subdistribution $\kappa$, as revealed by (2.1), with mean $\bar{v}_c$ and covariance $R_c$ determined using the samples belonging to that distribution, and assume that $\hat{y}_c$ is the output estimate determined by the cluster $c$. Then, the maximum likelihood estimate that combines the clusterwise sub-estimates in a probabilistically reasonable way, weighting the individual estimates by the appropriate probability, is given by

$$\hat{y}(\kappa) = \sum_{c \in \Gamma} \frac{p_c(\kappa)}{\sum_{c' \in \Gamma} p_{c'}} \hat{y}_c(\kappa) \tag{2.25}$$

The normalization factor in the denominator is needed to assure that the total probability of the sample to belong to some of the clusters is 1.

It is clear that if data only is available, determination of the cluster structure is a difficult task ... In Appendix A, some (more or less heuristic) approaches to determining the cluster structure are presented.

### 2.4.2 Example: Types of "Natural Data"

The class of nonlinear functions in real processes is hopelessly large, and capturing all alternative behavioral patterns within a single model structure is not possible. However, it turns out that just a few special types of nonlinearities usually exist in measurement data, and these classes of nonlinearity can nicely be captured by the Gaussian mixture model (see 23]). Let us study little closer those nonlinearities that we would assume to detect in a typical system to be modeled. The first type of structural nonlinearities is reflected as *separate clusters* (see Fig. 2.7). During different periods, different conditions in the process apply (sometimes a pump is on, sometimes it is off; sometimes ore is coming from one mine, sometimes from another mine, etc.), and the qualitatively differing process conditions are typically seen in the data in a specific way, the samples being clustered around the cluster centers. Within the operating regimes, however, no structural changes take place, meaning that within the clusters the Gaussianity assumption holds. This means that linear analysis can be carried out for each cluster separately.

The second typical source of distribution non-Gaussianity are the *continuous nonlinearities* (see Fig. 2.8). It is common in practice that this kind of behavior is approximated using piecewise linearized models around the operating points;

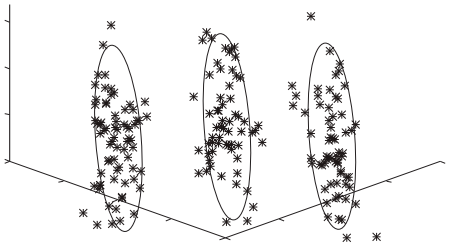


Figure 2.7: Clusters of Type I: Different operating regimes

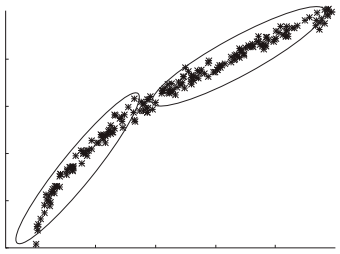


Figure 2.8: Clusters of Type II: Continuous nonlinearities

this means that separating data in clusters and modeling each operating point separately, useful models are again reached. It is a nice coincidence that this piecewise linearity approach is also well compatible with current engineering practices: Smooth nonlinearities are typically linearized around the operating points in control engineering models.

### 2.4.3 Outliers

A rather special reason giving rise to separate degenerate data clusters is the existence of *outliers* in the data. Outliers are more or less "lonely" samples, being typically not caused by real process originated phenomena but by spurious measurement errors, device or communication failures, etc. Often outliers are located alone far from other samples. However, the normal distribution extends to infinity — there exist no straightforward criteria for distinguishing between valid and outlier samples, and it is more or less visual inspection by a domain-area expert that is needed.

Because it is the error squared criterion that is typically used in modeling, samples far from the more typical ones have a considerable effect on the subsequent modeling. There are two opposite risks:

1. Including outliers among the modeling data may totally ruin the model, the far-away sample dominating in the final model.
2. On the other hand, too cautious selection of samples, neglecting too many samples, also affects the final model: It is those samples that are far from others that carry the most of the fresh information — of course, *assuming* that these samples carry information rather than disinformation.

As all clusters seemingly existing in the data should be checked separately to assess their overall validity, this is specially true in the case of outliers. Detecting outliers is knowledge-intensive, and special expertise on the domain-area, measurement devices, etc., is needed. Often a missing measurement variable is replaced by the measurement machinery by zero (or some other predetermined value), and such outliers can easily be detected, but this is not always the case.

Typically, if there is no scarcity of data, sample vectors with missing values can be simply ignored and eliminated from the data set. If only some of the

measurements are missing, all other measurements within a sample being valid, however, it may be reasonable to utilize that sample anyway: Then, the missing values have to be somehow fixed before the sample is used (see "missing values" in Appendix B).

## 2.5 Excursion: Networks and *power law*

Some of the "hottest" areas of research — like *chaos* and *complexity theory* — seem to be very far from the age-old statistical approaches. Specially, linearity seems to be completely out of the question: Interesting behaviors emerge only in nonlinear environments. However, looking the applications in more detail, it seems that there are connections.

It has been observed that there exist peculiar similarities among very different kinds of complex systems. For example, it has been claimed [**?**] that distributions in self-organized complex networks follow the *power law,* that is, there generally holds

$$y = cx^f \tag{2.26}$$

for scalars $y$, $x$, and constant $f$. Here, $x$ stands for the free variable, and $y$ is some emergent phenomenon related to the probability distribution of $x$; for example, if $x$ is the "ranking of an Internet page", and $y$ represents "number of visits per time instant", the dependency between these variables follows power law: There are some very popular pages, whereas there are huge numbers of seldom visited pages. As compared to Gaussian distribution, the power law distribution has "long tails"; the distribution does not decay so fast[1].

In the multivariate spirit, one can extend the single-variable formula (2.26) by including more variables; if there is only one variable $x_i$ changing at a time, the new formula corresponds to a set of $n$ simultaneous power laws:

$$y = x_1^{f_1} \cdot \cdots \cdot x_n^{f_n}. \tag{2.28}$$

Now, if one takes logarithm on both sides of the formula, one has

$$\log y = f_1 \log x_1 + \cdots + f_n \log x_n, \tag{2.29}$$

[1]It is interesting to note that the power law distribution is closely related to another modern concept, namely *fractal dimension.* Assuming that the variable $x$ represents some kind of "yardstick", determining the scale factor, and $y$ represents the level of *self-similarity,* so that when one zooms the original pattern by the factor of $1/x$, there exist $y$ copies of the original pattern (and this zooming process can be repeated infinitely), the fractal dimension of that pattern can be defined as

$$\dim = \frac{\log y}{\log x}. \tag{2.27}$$

When the pattern is simple, this definition coincides with the traditional ideas concerning dimension, but for complex patterns, non-integer dimensions can exist. Now, it is easy to see that, after taking logarithms, the parameter $f$ in (2.26) closely corresponds to the fractal dimension for the networked system

or

$$y' = f_1\, x'_1 + \cdots + f_n\, x'_n + c, \tag{2.30}$$

where $x'_i = \log x_i$, etc. It turns out that the multiplicative dependency has become globally linear — by only preprocessing the variables appropriately. There are also other approaches towards reaching a linear (local) model structure: Differentiate (2.29) around the nominal values $\bar{x}_i$, so that there holds

$$\left(\frac{\Delta y}{\bar{y}}\right) = f_1 \left(\frac{\Delta x_1}{\bar{x}_1}\right) \cdot \quad \cdots \quad \cdot f_n \left(\frac{\Delta x_n}{\bar{x}_n}\right). \tag{2.31}$$

Now the variables $\Delta x_i/\bar{x}_i$ are the relative deviations from the nominal state. This kind of variables are assumedly more robust that the log-variables. It is evident that very much can be done by appropriately conditioning the data; these issues are studied closer in the next chapter.

As a final note here, study the outlook of the *multivariate fractal distribution.* variable $y'$ in (2.30) is a sum of assumedly large number of assumedly independent stochastic variables $f_1 x'_1$. Because nothing more accurately about these variables is known, it can be assumed (again according to the Central Limit Theorem) that $y' = \log y$ has normal distribution:

$$p(\log y) = \frac{1}{\sqrt{2\pi}\sigma}\ \exp\left(-\left(\log y - \mu\right)^2 / 2\sigma^2\right). \tag{2.32}$$

Taking logarithms,

$$\log(p(\log y)) = c - \left(\log y - \mu\right)^2 / 2\sigma^2. \tag{2.33}$$

This means that the multivariate fractal distribution is *parabolic* rather than linear on the log/log axis, the three parameters being $c$, $\mu$, and $\sigma^2$. Indeed, this is in conflict with "traditional modern" network intuition!

## Computer exercises

1. Try the `dataClust` command in the `Regression Toolbox`. Define one-dimensional data of two Gaussian clusters, both containing 1000 samples and centers being 10 units apart, with the command

   ```
   [X] = dataClust(1,2,1000,10);
   hist(X,50);
   ```

   Modify data, summing variables that have this same distribution:

   ```
   X = X + X(randperm(length(X)));
   hist(X,50);
   ```

   Repeat the above steps sufficiently many times. What happens with the data clusters? Why natural data still typically is clustered — what is the difference in the data production processes?

2. Search for examples of observed distributions in complex networks that have been published in Internet. Applying some search engine, use keywords like

   ```
   power law distribution
   fractal dimension
   ```

   Study the distributions; observe how the claimed linear dependencies on the log/log scales (as resulting from the single-variable fractal dependency) can often indeed better be matched against a parabola (as resulting from the multivariate fractal dependency assumption).

# Lesson 3

# Understanding Data

As it will turn out, the multivariate modeling procedures are extremely mathematical, powerful, but rather mechanical procedures: There is not very much left for the user to do after the machinery has been started. But there is one thing that is common to statistics and computer programs: if you put *trash in,* you get *trash out* — and when these two, statistics and computers, are combined, as is the case in multivariate modeling, the risks only cumulate. The value of the modeling results is *completely* dependent of the quality of the modeling data; this data validity has to be ascertained by the user of the modeling tools. Whether the quality really *was* good, can only be assessed afterwards, when the constructed model is tested. It is *preprocessing of data* and *postprocessing of models* where expertise is truly needed (see Fig. 3.1).

## 3.1 From intuition to information

Statistical analysis methods can only do *data modeling,* not actual *system modeling.* The statistical analysis only looks for and utilizes the observed correlations between measurements. On the other hand, the mathematical tools always operate only within some selected model structure. It is the *user's* responsibility to connect this data crunching to real system modeling. When aiming at useful models, the user has to utilize his understanding in all levels of statistical modeling.

The data preprocessing and model postprocessing tasks — to be discussed later in this Lesson — are more or less *quantitative.* Before there are any numbers to be processed, however, some *qualitative* analyses are needed: The chaos of data has to structured. In this section, these preliminary analysis tasks are briefly discussed.

### 3.1.1 Some philosophy

It is perhaps interesting to recognize that systems modeling is closely related to those activities that have been studied by the philosophers since the dawn of history. The age-old questions of what the world is *really* like, and what we can

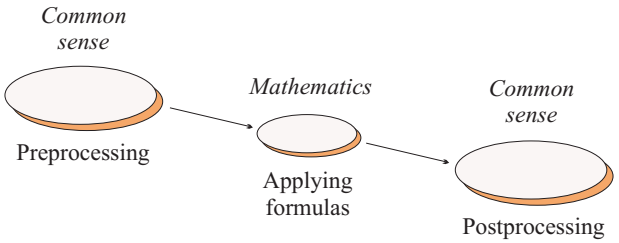



Figure 3.1: Role of knowledge in model construction

possibly know about it, are studied within two branches of philosophy, namely *ontology* and *epistemology,* respectively.

The Platonian *idealism,* where it is assumed that fundamentally there are some *ideas,* perfect objects underlying our observations, has become outdated — it has turned out that the *empiristic* approach is more fruitful. According to the Kantian view, it is assumed that it is only through our senses that we can receive information from our environment, and from these observations we construct our subjective world view, trying to find a coherent (sub-conscious) explanation for all of it. We can only hope that our mental machinery and senses are constructed so that they are capable of perceiving the essential phenomena in our environment and drawing relevant conclusions.

Loosely speaking, these two opposite views, idealistic and empiristic, correspond to the qualitative, first principles approach and the multivariate statistical approach to system modeling, respectively. It is now the mathematical machinery that is in the role of the human: Its subjective "world" is determined by those sensor signals that it is let to receive. It is not the human that is put in the center of this chaos of sensations, but it is the computer, and it is *we* that are like Gods in the universe of measurements giving the computer its senses and all those tools it has for making some sense in the chaos.

The questions of "applied ontology and epistemology" become to questions of *what are the system's real properties,* and *how can one get information about them.* What is valuable information in the measurements, and what is only noise? It is our task to make the (assumed) real system structure as visible as possible to the modeling machinery. The algorithms start from "tabula rasa", the only hardwired structures determining the construction of the data model being fixed by the organization of the measurements and the selected modeling method.

The problems of prior data analysis and manipulation, and those of model validation, are always knowledge-intensive. These tasks cannot be automated; there are just good practices that often seem to work. Because these tasks are based on expert intuition, there are no methodologies that would always work — that is why, this chapter gives various examples, hopefully visualizing the questions from comprehensible points of view.

### 3.1.2 Implementing structure on the data

One of the disadvantages when using data-oriented techniques is that it is difficult to integrate such models with expert understanding. However, to find the best possible model, one should utilize the available knowledge somehow. The only way to do this is to first partition the complex modeling problem into subtasks, in an engineering-like reductionistic way, exploiting the domain-area expertise in this partitioning task, and apply appropriate methods to the subproblems. If some parts of the process are known beforehand, their contribution can hopefully be eliminated from the remaining unknown behavior (see Appendix B).

To reach practical models, the data needs to be structured. This structuring should reflect the intended use of the model, but it should also support the human ways of perceiving the system.

*Causality* is one of the basic mechanisms that characterizes human cognition; on the other hand, statistical methods cannot see causalities (or any kinds of dependency structures) from data. This is the first, crucial task of the expert doing modeling: Determine the *inputs* and *outputs* of the system. The whole idea of regression models concentrates on modeling the relation between *action* and *reaction.*

The determination of the causal structure must be done by a human having some "common sense" — mathematical machinery can analyze data, revealing co-occurrences, but these dependencies are *correlation,* not *causation.* Constructing a causal structure that is based on false premises can result in a useless (and fallacious) model[1]. Study the following example:

> It has been recognized that *taller* children outperform smaller ones in almost all tasks, not only in physical contests, but also in cognitive tests. And this observation is *true,* however unjust it may sound. The correlation, however, vanishes, if only children of the *same age* are studied! There is no causation between size and mental capacity; rather, there is causal relation from age to both child size *and* capacity.

In concrete terms, to achieve causal structuring among data, the roles of different variables in the data vector $v$ need to be studied. From now on, we assume that the input variables are denoted $x_i$, where $1 \leq i \leq n$, $n$ being the number of input variables. The measurements are assumed to be linearly independent, so that none of the variables can be expressed as a weighted sum of the other ones. An input measurement sample can be presented as a data vector

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}. \tag{3.1}$$

Correspondingly, the output variables $y_j$, where $1 \geq j \geq m$, are collected in the

[1]However, this is again very much dependent of the intended use of the model: Non-causal correlations can be useful if aiming at simple prediction models, but they are not suited for control design purposes

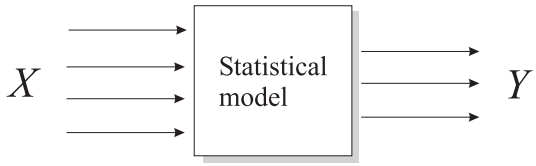



Figure 3.2: The assumed system outlook

output vector

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}. \tag{3.2}$$

The assumption is that the process output variables can be calculated as $y = f(x)$, where $f(\cdot)$ is a linear, vector-valued function, so that $f : \mathcal{R}^n \to \mathcal{R}^m$ (see Fig. 3.2). The linearity assumption (motivated in the previous chapter) means that the mapping from input to output can be represented using the matrix formulation:

$$y = F^T \cdot x, \tag{3.3}$$

where the dimension of $F$ is $n \times m$. Further, assume that one has measured sets of $k$ data vectors, from $x(1)$ to $x(k)$ and from $y(1)$ to $y(k)$, respectively. These observations are written in matrices (following `Matlab` practices) as

$$\underset{k\times n}{X} = \begin{pmatrix} x^T(1) \\ \vdots \\ x^T(k) \end{pmatrix} \quad \text{and} \quad \underset{k\times m}{Y} = \begin{pmatrix} y^T(1) \\ \vdots \\ y^T(k) \end{pmatrix}. \tag{3.4}$$

Again, the mapping between these matrices can be written compactly (note that changes in ordering and the transpositions are necessary to make the dimensions match) as

$$Y = X \cdot F. \tag{3.5}$$

No structure can also be seen in the data — the structure is imposed on the data by the domain area expert. The modeling machinery will match the data against this structure, finding the best possible parameters within that framework. For pragmatic reasons, depending on the application, it can sometimes be reasonable to apply some physically non-meaningful structure for the data. For example, it can be motivated to apply a *causally incorrect* structure: If there is need for a model for estimating some quantity based on measurements of other variables, the observed correlations can be exploited regardless of the actual causality structures. The model structure should follow this intended model usage, so that the variables that are used for estimation are collected in $x$, and the estimated variables in $y$. However, it is necessary to stick to the real causality structures, if one wants to apply the models not only for prediction but also for control, etc.

### 3.1.3 Experiment design

*Experiment design* studies how maximum information can be extracted by carrying out minimum number of (expensive) experiments. If data can be measured under optimal conditions, many of the problems that will be discussed later are automatically solved. However, a more challenging case is such that one can only *observe* the system, without being able to dictate the process inputs. At least if the system is large, this assumption typically holds.

No matter whether one can carry out an explicit experimenting procedure or not, there are some necessary requirements to be taken care of before data acquisition. It needs to be noted that the algorithms only see the data that is delivered to them; in this sense, one must trust on the "benevolence of nature", so that it is not explicitly trying to fool the observer! It is, of course, quite possible that all samples happen to be located in just a narrow region of the whole operating range of the system, thus misleading the analyses, giving incorrect mean values, etc.; but it is the *law of large numbers* promising that in the long run the observed mean value, for example, should be unbiased, assuming that lots of representative data containing fresh information is collected. But this optimism is justified only if there is not some agent acting (more or less consciously) in the opposite direction, explicitly ruining the quality of data! One of such mechanisms efficiently deteriorating the quality of data is *feedback.*

In the traditional modeling, one of the most important guidelines is that *if there are some feedback loops, they should be opened* before data acquisition. Otherwise, the causality chains become blurred: The physical output, if used for feedback control, effectively becomes an input signal. Study the following example:

> Assume that the process acidity is being controlled. Because of some unmodeled disturbance, the pH value tends to fluctuate. Proportional control law is used, that is, when the acidity is too low, more acid is added, and *vice versa.* If the process acidity is analyzed statistically, it seems that *high amounts of added acid correlate with low acidity.* In this case, of course, this paradox is easily explained: The actual process dynamics being slow and noisy, it is essentially the *inverse process,* the feedback loop that is being modeled; when the process is non-acidic, the acid flow is increased. Against intuition, the "cause" now is the process acidity, the acid flow being the "effect".

However, the above view ("open all loops!") is becoming challenged in the multivariate real world. First, there are the pragmatic reasons: During on-line operation of the plant the feedbacks simply cannot be opened — and, specially when complex systems are to be analyzed, not all feedback structures can even be detected, not all dependency structures are known. And, after all, one would like to know the *typical* behaviors in the process, not the artificially induced experiments. It is the undisturbed operation of the whole plant that is actually of interest — one should model the working plant with appropriate feedbacks closed. Indeed, the system should be seen as a "pancausal" network, where all variables are tightly interconnected. The consequences of this new kind of thinking are studied closer in Chapter 11.

## 3.2 Selection of variables

It is assumed that all relevant information that is assumed to contribute in the model construction can be stored in the *sample vectors.* No matter what is the origin of that data, it is, after all, static mappings among sample variables in $v(\kappa)$ that are constructed. One sample, or one unit of information, is assumed to be isolated from the other pieces of information, with no memory whatsoever. Model construction tries to combine such (contradictory) pieces of information $v(\kappa)$ for different values of $\kappa$ to reach a representation that can cover them all *as well as possible* — what this means is studied in detail in later chapters.

### 3.2.1 Feature extraction

As a basic rule, all information that is relevant for a model must be available at the same time, in a single vector $v(\kappa)$. All process phenomena should be captured as a set of stati(sti)c quantities. What is more, this data has to fulfill the structural assumptions, like linearity. Here, some guidelines are presented: How to select variables so that they would characterize the system appropriately. These variables are not necessarily the measurements directly: They are functions of the measurements that represent *features* characterizing the system appropriately. There are no unambiguous variable combinations to choose. Fortunately, the more sophisticated regression methods to be presented after Chapter 4 will efficiently solve this problem of high dimensionality, and then we can say that it is clever strategy to *include all available information there exists* and different kinds of features characterizing the model in the beginning (the excessive variables can be pruned later).

As an example, study how nonlinearities can be avoided by (formally) introducing approriate features.

Often it is so that a nonlinear function can be modeled in a linear form when the input dimension is augmented — that is, when additional features are included. This is the idea beyond, for example, *basis functions* (see later). For example, if one knows the functional form of the nonlinearity, this nonlinearity can be included among the input data: If there holds $y_i = F^T f(x)$, where $f$ is the known classa of nonlinearity, it is possible to introduce the new input vector

$$x' = \left( \frac{x}{f(x)} \right). \tag{3.6}$$

However, when doing data-based modeling, such *a priori* knowledge often cannot be assumed to exist. A generic way to extend the linear framework is to apply some kind of parameterized family of prototypical nonlinearities: For example, it is known that smooth functions can be approximated by their *Taylor expansions* that consist of a power series. Unknown nonlinearity forms can also be approximated using truncated power series expansions. If the linear term of variable $x_i$ does not suffice, arbitrary number of higher-order terms can be

included among the data:

$$\begin{pmatrix} x_i \\ x_i^2 \\ \vdots \\ x_i^\xi \end{pmatrix}. \tag{3.7}$$

When the nonlinear prototypical features are included among the input data, it is the task of the modeling machinery to select among the relevant components and determine their weights. More complex multivariate nonlinearities can be handled in the similar manner, applying the multiple-variable Taylor expansion, so that if one wants to be prepared for quadratic dependencies among variables $x_i$ and $x_j$, the input data vector can be augmented by the following three variables: $x_i^2$, $x_j^2$, and $x_i x_j$.

### 3.2.2 Special challenge: Dynamic systems

In systems engineering applications, it is often the dynamic properties that are of special interest. However, if dynamic phenomena are to be modeled, one is facing a problem: Static, instantaneous features are not enough to capture the dynamics that is characterized by memory, or inertia, coupling variables together also along the time axis. The basic trick is to use time series data, that is, prior variable values, $x_i(\kappa)$, $x_i(\kappa-1)$, etc. all have separate entries in the data vector — this is the standard approch, for example, in system identification, where the dynamic system also needs to be expressed in a static form (see Sec. 10.4). System theory assures that if the system memory extends back $n$ time steps, behaviors of a $n$'th order dynamic system can be captured. However, this definition of data vectors means that successive samples are overlapping, there are copies of the same variable values in different samples; this overlap and redundancy among samples can cause numerical problems (see Lesson 10).

If the dynamic system is infinite dimensional, sometimes the data representation can be simplified: For example, if there are delays, etc., the data can first be synchronized.

The above time series approach can be applied for capturing the fast dynamics in the system. However, it is not necessarily these high-frequency phenomena that are always of special interest; sometimes it is the stationary behaviors rather than the immediate transients of more or less random signal realizations that should be captured to reach some statistical relevance of features. The emphasis can be put on different frequency ranges, for example, by low-pass/band-pass filtering of the signals. Filtering of signals affects the weighting among frequency bands; this idea can be extended when one closer studies how information is not only distributed among frequencies but buried in the observations.

Signal smoothing is still not the only alternative to enhance the data: When filtering, it may be that relevant information is inevitably lost. Statistically relevant phenomena can be captured, for example, by matching the signals against some basis function families. The frequency-domain studies can be motivated also in this framework: If the signals are matched against the orthogonal set of harmonic functions, one receives spectra corresponding to the frequency content

in the signals. These spectral components can be used for characterizing the dynamic properties of the system, and the data vector $x$ can be constructed correspondingly. Whereas the spectra represent long-term phenomena, other function families can be applied to capture short-term ones: For example, *wavelets* have been proposed for this purpose. Wavelets also span a family of orthogonal functions, but having a rather limited range, they can be applied to characterize spurious peaks in signals, etc.

Powerful features can be constructed when observations are matched against assumed model structures, nonlinear or dynamic, and when it is the fitted model parameters that are used to characterize the system. For example, in cellular phones voice coding applies this strategy: The formants characterizing the voiced phonemes can efficiently be captured in the auto-regressive (AR) model parameters, and when only the dynamic model parameters are employed the amount of transmitted data can be radically reduced. The overall system behavior can be represented as a collection of local behaviors, as characterized by lower-level local model parameters within some structural framework. To implement more complicated feature extraction strategies, different approaches can be further combined: For example, time-domain (time series) structures can be combined with temporally local feature extraction techniques for modeling variability of behavioral properties between time windows.

In short, when defining features to characterize dynamic phenomena, one should avoid trusting some individual phenomena, absolute time points, etc., and use some invariant quantities or perhaps statistical cumulants characterizing signal properties. The features should be valid for different sets of signals with different noise realizations: No minimum/maximum values, etc., but averages or probabilities. Integral-based criteria (ISE, ITSE, etc.) are typically smoother-behaving than some perhaps visually well-motivated criteria[2].

Sometimes it is possible to abstract the time axis away altogether, consentrating exclusively on the higher-level *quality measures* directly [**??**]. The higher-level measures one extracts from the data, the farther the quantities are from the original measurements, and the nearer they are *qualities* characterizing the system. There is also no clear distinction between the "quantifying variables" and the "qualifying variables" — also this structure is not determined by the system itself, but by the model designer.

There are some intuitions offered also by the studies concerning *cybernetic systems:* The key point in such systems is balance, and the cybernetic model essentially is a model over the spectrum of balances. To capture this essence, one has to code these balances already in the data; that is, the tensions keeping the system in balance need to be represented by the data. In concrete terms, this means that not only the process state but also the balancing forces, or control signals, need to be included in data.

[2]For example, the *settling time,* traditional measure characterizing system responses, revealing when the oscillation after a transient has decayed below the level of, say, 5% of the original, turns out to behave in a curious, non-continuous manner: It is either after the first oscillation cycle, or after the second (or third, ...), when the criterion level is no more crossed — it turns out that as system parameters are varied the possible locations of these time points are not continuously distributed but more or less clustered along the time axis

## 3.3 Data preprocessing

After the set of variables has been selected, they need to be conditioned to reveal the information they carry in an optimal way. The most typical tasks here are *centering* and *scaling*. First, however, more challenging situations are studied.

### 3.3.1 Reaching "well-behavedness" of data

Often, the distribution of the variables is more or less peculiar. Sometimes it can be motivated to normalize the distribution — remember that it was assumed that data being modeled is Gaussian.

#### Qualitative data

Qualitative data here means data that does not have continuous distribution: For example, there can be binary data concerning process operating mode, etc. A single status bit can have crucial effects on the interpretations: The roles of the variables can change altogether depending of the operating mode. In principle, such qualitative data gives rise to clustering, so that each combination of qualitative variables defines a cluster of its own. However, the number of clusters explodes exponentially if there exist various qualitative variables, and this should not be done without closer analysis of data. As was discussed in Sec. 2.4.3, introducing new clusters too hastily may weaken the overall information content that is available for modeling the individual clusters. Further, there are the problems of mastering the "model library": There is a separate model for each cluster.

Assume that one allocates a separate (binary) variable for each of the qualitative values. Very often it turns out that the effects of the individual binary data become abstracted away, so that they sum up to a more or less continuous distributions.

There are different types of qualitative variables, not all are binary. Some qualitative variables can rather naturally be quantified: For example, alternatives along a continuum (like "hot" — "medium" — "cold") can be *fuzzified* by a domain-area expert, so that those variables can be coded in numeric form after all (and, as will be seen also when discussing *neural networks,* the "modern" methods should not be seen as alternatives of statistical methods, but as complementary techniques) .

#### Logistic regression

Sometimes one has variables that are limited to a certain range. For example, assume that the variable $p_i$ (being interpreted as some "probability") ranges originally between 0 and 1. The problem here is that linear models cannot easily be constrained to only deliver results obeying such range limitations. It turns out that by appropriately modifying the variables, such problems can be (virtually) avoided.

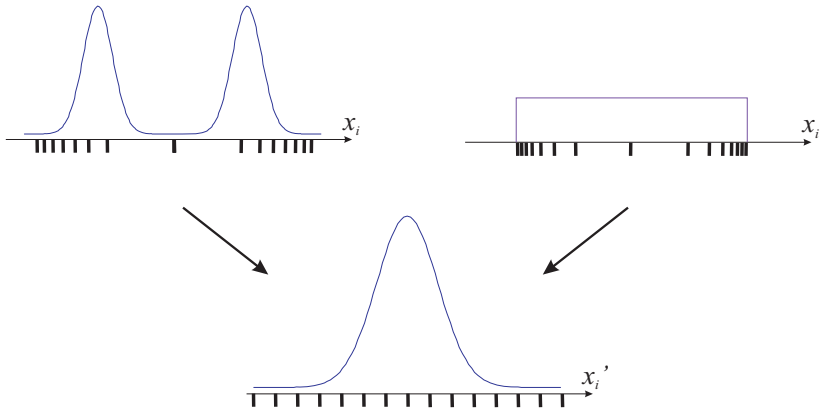



Figure 3.3: The idea of histogram deformation: The original data densities (on top) can be changed by stretching and contracting the data axis appropriately. This modification is applied to all of the variables separately

Assume that one computes $p_i/(1-p_i)$ — this way, the range can be extended from 0 to infinity. Additionally, if one defines

$$v_i = \log \frac{p_i}{1-p_i}, \tag{3.8}$$

there holds for the new variable $-\infty < v_i < \infty$. If such a data data deformation is carried out before modeling, one sometimes speaks of *logistic regression.* Note that extreme values $p_i = 0$ and $p_i = 1$ are equally illegal when applying the deformation.

#### Histogram equalization

In some cases there is no real physical reason to assume that the data should be non-Gaussian in the first place. It may be that the anomalies in the distribution are caused by some external factors, whereas the original distribution really was normal (see Sec. 3.6.1). In such cases one can equalize or "renormalize" the virtual distribution by nonlinear modifications of the variable scale: Data density is deformed when the samples are distributed on a differently scaled axis (see Fig. 3.3).

This deformation of the data axis must be remembered when applying the model that is constructed for renormalized data: All variables have to be deformed correspondingly. Indeed, this need for restoring the original data properties applies to all data preprocessing.

### 3.3.2 "Operating point"

Look at the regression formula (3.3): It is clear that the regression hyperplane always goes through the origin of the data space, so that $x = \mathbf{0}$ means $y = \mathbf{0}$. This must be taken into account during the data preprocessing: One has to

select one point (explicitly or implicitly) where the regression hyperplane will be anchored. This is simple if there is some physical knowledge available: For example, if the point $\bar{\mathbf{x}}$ is known to correspond to $\bar{\mathbf{y}}$, one has to apply such a transformation that this point becomes the origin of the modified data; that is, $\mathbf{x} \leftarrow \mathbf{x} - \bar{\mathbf{x}}$ and $\mathbf{y} \leftarrow \mathbf{y} - \bar{\mathbf{y}}$. This transformation must, of course, be remembered every time when the model is used, eliminating $\bar{\mathbf{x}}$ from $\mathbf{x}_{\text{est}}$ during run-time application, and after the $\mathbf{y}_{\text{est}}$ has been found, the transformation must be inverted by adding $\bar{\mathbf{y}}$ to the result to receive the answer in original coordinates.

Often, there is no *a priori* knowledge of the values $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$. The normal approach in such cases is to *assume* that the regression line goes through the data center, that is, one selects $\bar{\mathbf{x}} = \frac{1}{k} \cdot \sum_{\kappa=1}^{k} x(\kappa)$ and $\bar{\mathbf{y}} = \frac{1}{k} \cdot \sum_{\kappa=1}^{k} y(\kappa)$, and eliminates this mean from the data. This procedure is called *mean centering* of data. Note that, even though this approach is commonly used, it is still quite heuristic[3].

In principle, there is another way to avoid this affinity problem caused by unknown operating point: One can include some constant among the measurement signals, so that, say, $x_0(\kappa) \equiv 1$; the resulting mapping $y = F^T x + F_0$ does not have the above constraints. Here this approach is not recommended, though: The problem is that the signal covariance matrix would become singular, one of the variables being constant, and some of the methods that will be discussed later could not be applied at all.

In some cases one can eliminate the effects of biases by *differentiation,* that is, one can define $x'(\kappa) = x(\kappa) - x(\kappa - 1)$ and $y'(\kappa) = y(\kappa) - y(\kappa - 1)$. It turns out that when using $x'(\kappa)$ and $y'(\kappa)$ rather than the original variables, the constant term vanishes from the model:

$$\begin{array}{rl} & y(\kappa) = F^T x(\kappa) + F_0 \\ - & y(\kappa - 1) = F^T x(\kappa - 1) + F_0 \\ \hline & y'(\kappa) = F^T x'(\kappa). \end{array} \tag{3.9}$$

It may also be so that the values $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ change continuously. For example, linear trends are common in practice. Elimination of the trends (or other deterministic components) is more difficult than compensating constant biases, because the behaviors rarely remain constant *ad infinitum.*

Finally, study an example: If mean centering is forgotten, and no other appropriate method is applied, the results can be catastrophic, specially if the data mean dominates over the variance: The center of the virtual distribution lies always in the origin, extending symmetrically in the "negative" direction (see Fig. 3.4). The resulting model is intuitively incorrect: If $x$ goes up, also $y$ goes up according to the model — even though this is evidently incorrect.

### 3.3.3 Data scaling

The role of variable scaling is to make the relevant features optimally visible in the data. Study an example:

[3]If the center is determined blindly from the data, the degrees of freedom become lower; when calculating covariance matrices, for instance, this should be taken into account (by dividing with $k - n$ rather than with $k$), but here it is assumed that the number of samples $k$ is so large that ignoring this does not matter too much

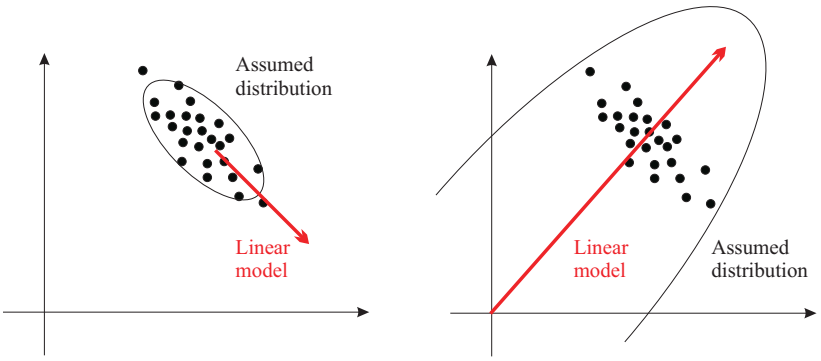



Figure 3.4: The original data distribution, on the left, and the virtual distribution if the centering is "forgotten", on the right

> Assume that there are temperature values in the range from 100°C to 200°C and pressure values in the range from 100000 Pa to 200000 Pa among the measurement data. The variation range in temperature (ignoring units) is 100 and in pressure it is 100000. It turns out that in the mathematical analysis the role of the temperature will be neglected because the variation range is so narrow: The error-square criterion concentrates on minimizing the more significant variations, emphasizing the pressure measurements exclusively.

The "equalization" of variable visibility can be carried out using data scaling. Scaling can be formally expressed using weighting matrices $W_{\mathbf{X}}$ and $W_{\mathbf{Y}}$: If data $\mathbf{X}$ is to be scaled, for example, one has $X = \mathbf{X}W_{\mathbf{X}}$. Often, $W_{\mathbf{X}}$ and $W_{\mathbf{Y}}$ are diagonal matrices with the corresponding elementwise scaling factors on the diagonal.

It is customary to assume (if there is no additional knowledge) that data given in different units should carry the same level of information. This heuristic means that all variables should have about the same variation range to be equally emphasized in the error-squared based algorithms (see Sec. **??**). To accomplish this normalization, the weighting matrix should be selected as

$$W_{\mathbf{X}} = \begin{pmatrix} \frac{1}{\sqrt{\mathrm{var}\{\mathbf{x}_1\}}} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{\sqrt{\mathrm{var}\{\mathbf{x}_n\}}} \end{pmatrix}, \tag{3.10}$$

and similarly for the output data. For each variable there then holds $\frac{1}{k} \cdot X_i^T X_i = 1/(k \cdot \mathrm{var}\{\mathbf{x}_i\}) \cdot \mathbf{X}_i^T \mathbf{X}_i = 1$. However, there are also other ways to define the scaling (see Appendix B), and there are no general guidelines for scaling that would always give optimal results exist.

However, it seems that if the data comes from a strictly cybernetic system [9.11], the rigid model structure proposes some guidelines. If the variables represent "deformations" in the cybernetic system as defined by the interaction between a system and its environment, variance normalization is explicitly motivated;

further, in such systems mean values are not necessarily eliminated in the preprocessing phase.

After the above steps it is now assumed that the data in $X$ and $Y$ are valid, and the system properties are (more or less) optimally visible in the data. In the subsequent chapters, the construction of the matrices $X$ and $Y$ is no more concentrated on.

## 3.4 Model construction and beyond

In the beginning of the chapter it was claimed that there is no room for expertise during the actual regression model construction phase. However, this is not exactly true. First, the *selection of the modeling method* is a question of what one expects there to be found in the data. Second, many of the more sophisticated methods are more or less *interactive,* so that the final model refinement (like determining the model order) is left to the user.

### 3.4.1 Analysis and synthesis

When using the more sophisticated methods, to be discussed in later chapters, the modeling procedure can be roughly divided in two parts, in *analysis* and in *synthesis.* In analysis, the mathematical machinery is used to reveal some kind of *latent structure* hidden among the observed data dependencies. The data being numeric, there are typically no clear-cut absolutely correct answers to the question of the underlying structure; the machinery just makes the data better comprehensible, so that the final decisions can easier be made by the user. This structure visualization is typically carried out so that the most fundamental data properties of the high-dimensional data are compressed into sequences of scalars measuring the relevance between the structural constructs. Generally, it is the model dimension selection that is left to the user.

After the analysis, in the synthesis phase, the analyzed structure is reconstructed in another form; when discussing regression models, this new structure emphasizes the mapping from input to output.

Note that in synthesis and in analysis the data preprocessing can be carried out in different ways — that is, the selection of the association matrix form studied in the previous chapter is an independent task from final mapping model construction. Generally, in analysis, when only the latent structure is searched for, there is more freedom; on the other hand, in synthesis, one has to be able to somehow "invert" the data deformations to construct the output mapping.

It turns out that optimizing the model is often rather simple (at least if the optimality criteria are selected in a sensible way). However, it also turns out that sometimes *the "best" is an enemy of "good".* It is not only the accuracy but also the *robustness* or *generalization capability* of the model that should be taken into account: How the model behaves when the data is somewhat different as compared to the training data?

### 3.4.2 Validating the model

After a model has been constructed, the knowledge is needed in *interpreting* the results: What has actually been carried out, does the model do what it was intended to do. In principle, the model validity should be checked using statistical significance tests, etc. However, these methods often turn out to have more theoretical than practical value and they are not studied in this context; a more pragmatic approach to model validation is taken.

It is fair to apply the same criterion for evaluating the model as was used when the model was constructed, that is, the sum-of-squared-errors criterion[4]. However, there is a catch: A good measure for checking the model robustness, applicable to all models, is to see what is the average prediction error size for *independent* data that has not been used for training. If the error matrix is defined as $E = Y_{\text{test}} - \hat{Y}_{\text{test}}$, where $Y_{\text{test}}$ is the correct output and $\hat{Y}_{\text{test}}$ is the estimate given by the model, the following *Mean-Square Error (MSE)* measure can be applied for each output separately:

$$\frac{1}{k} \cdot E_i^T E_i = \frac{1}{k} \sum_{\kappa=1}^{k} e_i^2(\kappa) = \frac{1}{k} \sum_{\kappa=1}^{k} \left( y_{\text{test},i}(\kappa) - F_i^T x_{\text{test}}(\kappa) \right)^2 . \tag{3.11}$$

Without independent data, if only the data fit is measured, one can only speak of modeling *data.* If the model works fine for independent *validation data,* one can assume that the model also captures the actual *behaviors.* A still more challenging goal is to find a model that would represent the *system.* Whereas validation data is typically collected from the system in the same environmental conditions as the training data was, the *testing data* is collected in different conditions, in different time. If the correspondence between the model and the real system still is good, one can be satisfied — at least for some time: The properties of the systems typically change over time.

Because the properties of the models are determined in the preprocessing phase, but the model validity can be seen only afterwards, it is evident that the cycle between preprocessing and model construction becomes iterative. Indeed, it is clever to construct different types of models, using different kinds of preprocessings for different sets of input data, and compare the results. Rather than employing the computing capacity for complex parameter fitting for complex models once and for all, the repetitive approach is here preferred: The designs become more transparent and analyzable in this way. Because of this iterative nature of model design, it is important that the model construction can be carried out in an efficient way — and the toolbox of methods to be presented later all share this efficiency (linearity) goal.

[4]Remember that the selection of the validation criterion is not quite straightforward: For example, note that the (otherwise clever) information theoretic criteria like Final Prediction Error criterion (FPE), Akaike Information Criterion (AIC), or Minimum Description Length (MDL) criterion are *not* suited here. They only employ training data in the formulas, measuring the model applying *a priori* structural assumptions; robustness, being due to *unanticipated* disturbances, cannot be captured. Typically, it would be the basic least-squares method that would win

### 3.4.3 Cross-validation

A practical way to evaluate the model validity, at least if there is scarcity with data, is *cross-validation.* The basic idea is to leave one sample (or sequence of samples) out at a time from the data set, construct regression model using other remaining training samples, and check how well the missing sample can be predicted using this truncated model. When this procedure is repeated for all samples (or all sequences), a rather reliable view of how well the modeling method can abstract the data is found. On the other hand, large cross-validation errors may also reveal outliers that can be eliminated during the next modeling cycle.

## 3.5 Summary: Modeling procedures

The above discussion can be summarized as follows (note that the steps below illustrate the typical procedure, not always being implemented exactly in that way):

**Constructing the model**

1. Construction of the features, classification of data, search of primary Gaussian distributions, outlier detection, determination of training sets $\mathbf{X}$ and $\mathbf{Y}$, and the corresponding test sets $\mathbf{X}_{\text{test}}$ and $\mathbf{Y}_{\text{test}}$.
2. Determination of the data scaling matrices $W_{\mathbf{X}}$ and $W_{\mathbf{Y}}$ using data in $\mathbf{X}$ and $\mathbf{Y}$.
3. Preprocessing, data transformations, centering and scaling, giving $X = (\mathbf{X} - \bar{\mathbf{X}})W_{\mathbf{X}}$ and $Y = (\mathbf{Y} - \bar{\mathbf{Y}})W_{\mathbf{Y}}$, and $X_{\text{test}} = (\mathbf{X}_{\text{test}} - \bar{\mathbf{X}})W_{\mathbf{X}}$ and $Y_{\text{test}} = (\mathbf{Y}_{\text{test}} - \bar{\mathbf{Y}})W_{\mathbf{Y}}$.
4. Model structure refinement, if appropriate, giving $\theta = g_\theta(X', Y')$.
5. Model construction, giving $F = g_F(X, Y, \theta)$.
6. Model validation, comparing $X_{\text{test}}F$ against $Y_{\text{test}}$.

Note that in Steps 4 and 5 different preprocessing procedures may be used, so that the data $X'$ and $Y'$ need not be the same as $X$ and $Y$. The semantics of "functions" $g_\theta$ and $g_F$ will be concentrated on in subsequent chapters (see page 80).

**Using the model**

1. Construction of the features, classification of data, selection of the primary Gaussian distribution, outlier detection, giving $\mathbf{X}_{\text{est}}$.
2. Preprocessing, data transformations, centering and scaling, giving the final data $X_{\text{est}} = (\mathbf{X}_{\text{est}} - \bar{\mathbf{X}})W_{\mathbf{X}}$.
3. Model use, giving $\hat{Y}_{\text{est}} = X_{\text{est}}F$.
4. Inverse transformations, denormalization and decentering; reconstruction of the final estimate by inverting all preprocessing operations that were carried out for the output data: $\hat{\mathbf{Y}}_{\text{est}} = (\hat{Y}_{\text{est}}W_{\mathbf{Y}}^{-1}) + \bar{\mathbf{Y}}$.

## 3.6 Case studies

In this section two examples of modern process data preprocessing are presented. In both cases the information is acquired in the form of digital camera images. One reason for increased general interest in machine vision is the intuition: Humans looking at complex processes often can recognize valuable information — why not automate such a measuring process? However, whereas perceiving images is easy for humans, but pattern recognition is very difficult for computers. Assuming that there are, say, $512 \times 512$ pixels of raw data in these images, multivariate methods are clearly needed for *sensor fusion* — but without appropriate preprocessing of the data, the relevant information would still remain hidden.

These examples illustrate how difficult it is to give any general guidelines on how the data should be preprocessed; it is always a matter of domain area expertise. In 2005, both of these process analyses are still being carried out and further modeling is still continuing.

### 3.6.1 Analysis of the paper machine dry line

The first example illustrates the modern development work at a paper mill. This modeling effort is currently taking place at Stora-Enso Kaukopää plant in Imatra, Finland. The discussion here is somewhat streamlined, simplifying the problem to some extent; more detailed discussions can be found, for example, in [4].

The paper machine consists of the "wet end", where the liquid-form pulp is processed, and the "dry end", where the more or less solid-form paper (or cardboard) is received. The connection point between these two processing phases is the *wire,* where the pulp is spread from the *headbox.* The wire being a sparse fabric, excessive water is filtered through it, whereas the fibres remain on the wire, constituting the final paper formation. The wire runs continuously, taking the moist paper to the drying section.

The drying section consists of dozens of steam-heated cylinders; the important processes governing the final paper properties take place on the wire, but the results can today be measured only in the end of the dry end, causing a considerable delay in the control loop. It would be excellent if the properties of the final paper could be estimated already on the wire; this would make the control loops much faster. And, indeed, there seems to be room for improvement: The location on wire where the slush pulp turns from thick liquid into moist paper affects the paper formation and thus the final paper properties; in this transition region the mirror reflectance of the pulp surface turns into diffuse. This *dry line* has traditionally been utilized by the operators for more or less intuitive manual process control. Installing a camera beside the wire and determining the dry line from the digital image, one could perhaps mimic the expert actions (see Figs. 3.5 and 3.6).

Before some kind of control based on the dry line measurements can perhaps be implemented, the problem that remains is that the connection between the dry line measurements and quality properties at the dry end (mainly *final humidity*

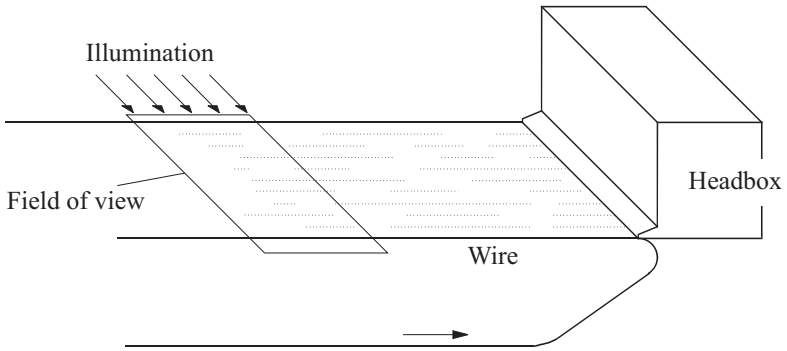



Figure 3.5: Paper machine headbox and wire

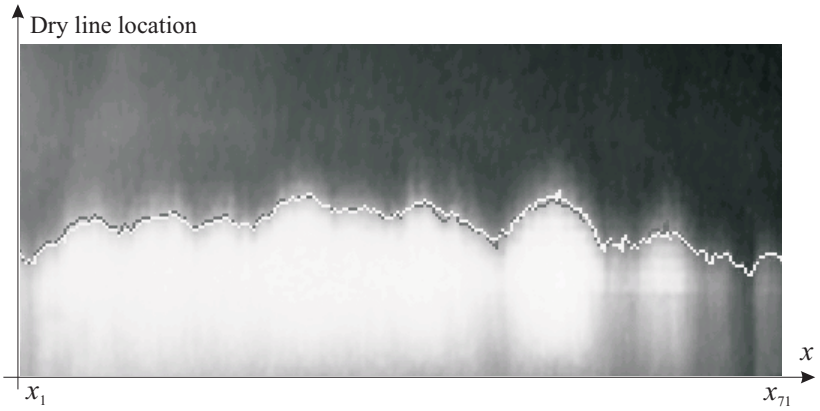



Figure 3.6: Paper machine dry line as seen by the camera. The edge detection algorithm has determined the "most probable" location of the wet/dry transition

and *total mass*) should be determined. It is the *profile* that is the most relevant now: The distribution of the fibres is determined on the wire. This means that the variations in "cross direction" (CD) in the dry and wet ends of the machine should be connected using statistical methods.

There are many technical problems in the camera imaging, concerning illumination, etc.; here we assume that these problems are solved and a high-quality digitized image is available. The pixel matrix first has to be deformed to compensate for the perspective distortions caused by the nonoptimal camera installation. An edge detection algorithm is applied to find the locations where the gradient of the pixel intensity is maximum: When these maximum gradient points are connected, an estimate for the dry line is achieved.

The dry line is analyzed every 18 seconds, extracting 71 dry line measurement values along the width of the wire, constituting the original data vector $\mathbf{x}(\kappa)$. When data is collected during several hours, the data matrix $\mathbf{X}$ can be constructed. Similarly, in the dry end, a traversing sensor measures the quality properties, constituting the output data $\mathbf{Y}$.

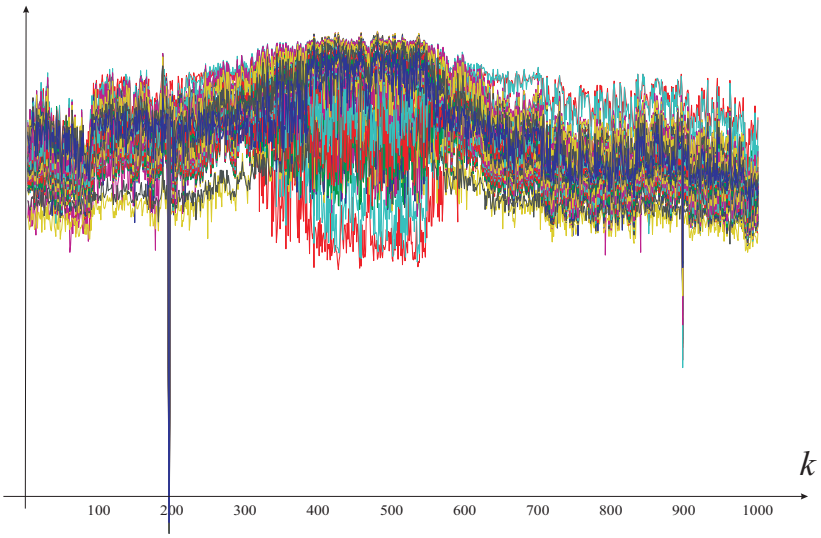



Figure 3.7: Dry line behavior in time. Some 1000 samples of all profile points (71) are plotted; samples #194 and #898 seem to be outliers

The first task in preprocessing is to make the input data and output data compatible. Because it takes (for the cardboard type being produced) 1.5 minutes to proceed from the wire to the dry end sensors, the output block must be *shifted* so that corresponding data values are aligned: The number of shifts in this case is five (because 90 sec / 18 sec = 5).

The outliers have to be eliminated from the data (see Fig. 3.7. These outliers are due to failures in pattern recognition — these problems may be caused by, for example, an operator walking by the wire, confusing the edge detection algorithm! Of course, the outlier data rows have to be eliminated in input and output blocks simultaneously to keep the data structures aligned. After this, some 1000 valid data points were left in data.

Next, the distribution of dry line points is analyzed (see Fig. 3.8). It seems that the measured dry line points seem to be distributed rather strangely, wide "unactive" regions existing between areas of frequent hits. Closer analysis reveals that this is (partly) caused by the *suction boxes:* a negative pressure under the wire is applied to increase the efficiency of pulp drainage. There is no real physical reason why the dry line should have non-Gaussian distribution, and it can be assumed that these anomalies are caused by the external suction. That is why, the normality of the distribution is restored by using histogram equalization techniques. Note that histogram equalization has to be carried out for the data still in absolute coordinates (because the effects of the suction boxes, of course, are always visible in the same coordinates), so that necessarily equalization has to precede any other manipulation affecting the numerical measurement values.

Because only profile shapes are now of interest, the changes in the dry line average have to be eliminated. This means that the instantaneous dry line mean (in cross-direction) is eliminated from the measurement sample: $\mathbf{x}'_i(\kappa) =$

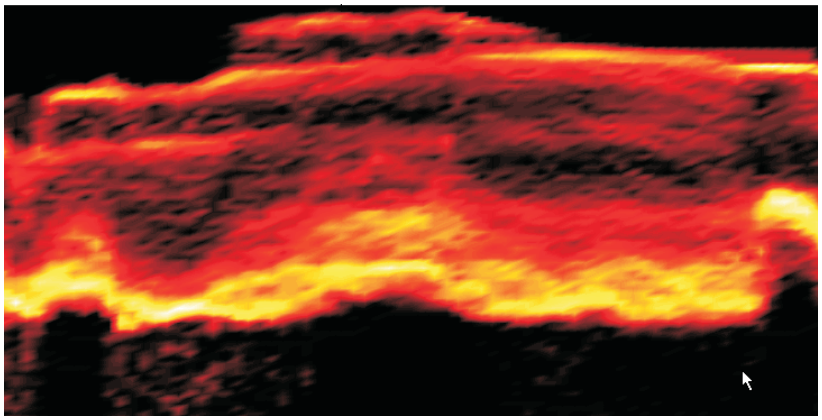


Figure 3.8: Paper machine dry line histograms as a contour plot (wire running upwards). The lighter a point is in this image, the more hits there are, meaning that the measured dry line is most probably located there in the long run

$\mathbf{x}_i(\kappa) - \frac{1}{71} \cdot \sum_{i=1}^{71} \mathbf{x}_i(\kappa)$. However, this seemingly straightforward data modification procedure introduces surprising additional problems: Note that after this modification the variables become linearly dependent, the last variable (or any of the variables) being negative of the sum of the other ones, so that, for example, $\mathbf{x}'_{71}(\kappa) = -\sum_{i=1}^{70} \mathbf{x}'_i(\kappa)$ — otherwise they would not add to zero! The easiest way to circumvent this new problem is to simply *ignore* the last, redundant variable, so that effectively we have $n = 70$. The linear independence of the variables (or the invertibility of the covariance matrix) was assumed before, and it is the prerequisite for many of the regression methods.

Only after these steps, the standard operations of mean centering and scaling are carried out (now in MD, or "machine direction", that is, $\kappa$ running from 1 to 1000). Now, because all input variables have the same interpretation, it is natural to simply normalize the variances to unity before the model construction phase.

### 3.6.2 Modeling of flotation froth

*Flotation* is used in mineral processing industries for separation of grains of valuable minerals from those of side minerals. In the continuous flow flotation cell (see Fig. 3.9), air is pumped into a suspension of ore and water, and the desired mineral tends to adhere to air bubbles and rises to the froth layer where the concentrate floats over the edge of the cell; the main part of other minerals remains in the slurry. The separation of minerals requires that the desired mineral is water-repellent: In zinc flotation, this can be reached by conditioning chemicals like copper sulphate $CuSO_4$.

Flotation is one of the most difficult and challenging processes in mineral processing industry. This characteristic of the process mainly arises from the inherently chaotic nature of the underlying microscopic phenomena; there are no

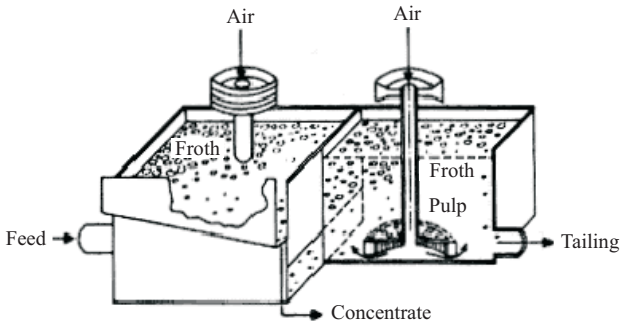



Figure 3.9: An array of two flotation cells in series

good models available that would capture the behaviour of the particles. Additional problems are caused by the fact that todays measurement technology is not able to provide with a description of the current state of the process that would be accurate and reliable enough. It is the froth surface that dictates the quality of the outflowing concentrate; the properties of the froth are reflected in its texture, movement, and colour. No standard measurement devices, however, can capture the outlook of the froth. Thus, most of the chemical reagents that are used to increase the efficiency of flotation are controlled by the human operators. The operators usually determine the suitable levels of the reagents by analysing the visual appearance of the froth; the control strategies they apply are expert knowledge.

Perhaps the limited capacity of the operator to monitor cells continuously (the operator is usually responsible for various circuits consisting of several cells) could be increased by machine vision (see Fig. 3.10)? This idea was studied during the Esprit long term research project ChaCo (Characterization of Flotation Froth Structure and Colour by Machine Vision); for example, see [25]. There were various lines in this research project; however, in this context only those studies are explained where the machine vision system was used to help the human operators in their task, analyzing the froths for process monitoring and supervision purposes.

The status of the flotation froth cannot be uniquely characterized; there are just a few measurements that can be explicitly defined and determined (like the froth level and thickness), whereas most of the factors that characterize the froth properties are more or less conceptual having no explicit definition. To construct "soft sensors" for the unmeasurable quantities, operator interviews were first carried out to find out what are the most relevant phenomena to be studied in the froth. It turned out that the trivial features — like "big bubbles", "small bubbles", etc. — are only used by novices, not the real experts. The operator interviews revealed that the most interesting (more or less fuzzy) froth types in zinc flotation are the following[5]:

[5]Note that *classification* can be seen as a special case of regression. Each of the classes has an output of its own in the regression model; this variable has value "1" if the sample belongs to that class. Using regression, the calassification results are not binary — the output values reveal how certain the classifications are

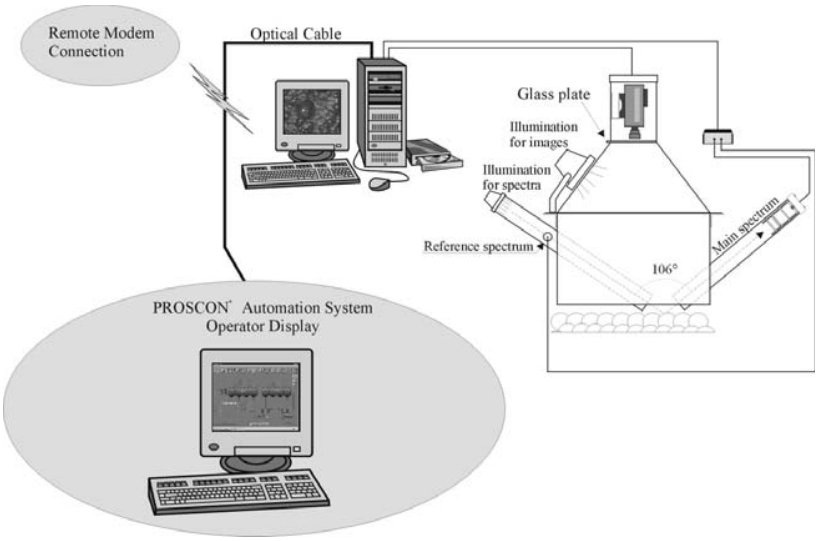



Figure 3.10: The automatic froth monitoring system helping the operator

1. **"Wet" froth** is characterized by "empty bubbles", meaning that not all bubbles are covered with the concentrate; it also seems that most of the bubbles are tiny whereas some of them may grow excessively. The bubbles do have a rather high tendency to burst.

2. **"Dry" froth** has a rather even tessellation, bubbles being of equal size and all being covered by concentrate; because of the uniformity, the bubbles are often hexagonal. This seems to be the category characterizing optimal production conditions, both froth speed and quality of concentrate being high.

3. **"Stiff" froth** is "porridge-like", the bubble forms becoming distorted and finally being substituted for layered concentrate rafts; this kind of froth floats rather unevenly, sometimes stopping altogether. In the extreme, stiffness can make the froth collapse, so that no concentrate floats out; these pathological cases should be avoided at any cost.

These characterizations are conceptual and there are no exact mathematical definitions for them (see Fig. 3.11). The role of image data preprocessing is to somehow make the classes distinguishable. First, it was noticed that static images are not enough: Many of the characterizations involve dynamic phenomena. Second, there is need for both frequency-domain and spatial segmentation approaches, as well as for pixel-wise analyses:

- **Dynamic phenomena** were captured by analysing image pairs having 0.2 sec time interval; this way, the changes between the images revealed information about the *bubble collapse rate* (how well the aligned images match each other) and *froth speed* (the average speed being determined as the maximum point in the image pair cross-correlation matrix — this can be calculated in the *frequency space,* that is, the tgwo-dimensional

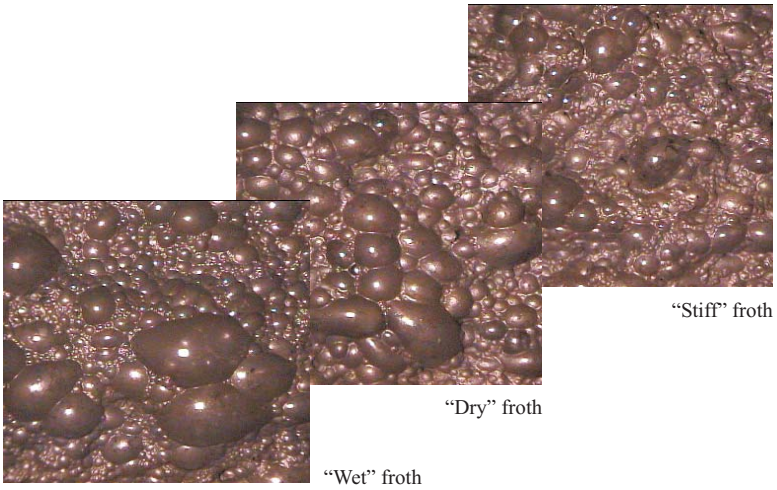



Figure 3.11: Different "conceptual categories" as seen by experts

FFT transform is applied to both images, and these transformed images are multiplied pixelwise, revealing the matches among shifted images). The *variation in speed* was measured by how high the average maximum cross-correlation peak was as compared to neighboring values.

- **Frequency domain methods** (in practice, based on the two-dimensional fast Fourier transform) were used to analyze the directional orientedness and non-sphericity of the bubbles; also the above cross-correlation matrices were calculated using FFT.
- **Segmentation techniques** (based on the so called "watershed technique") were used to extract the properties of individual bubbles; the bubble size distributions were determined this way, as well as average "roundness" of the bubbles.
- **Pixel-wise analyses** were carried out, for example, to determine the "emptiness" or transparency of the bubbles. The bubbles covered with concentrate only reflect light in a diffuse manner, whereas uncovered bubbles typically have bright total reflectance points on top; the number of maximum intensity pixels in the image can be used as a measure for the number of empty bubbles.

Finally, there are some few dozen variables characterizing the froth state, a new set of variables being calculated after every twenty seconds; this data is then mean-centered and normalized.

After all the above steps the data is ready for further model building — whatever that happens to mean. These questions will be concentrated on in subsequent chapters.

## Computer exercises

1. You can check how deformed, non-Gaussian data looks like after the "equalization" of the histogram:

   ```
   DATA = regrDataClust(1,2,100,5,3);
   hist(DATA);     % Matlab histogram command
   defmatrix = regrForm(DATA);
   X = regrDeForm(DATA,defmatrix);
   hist(X,10);
   hist(X,30);     % Note the changed resolution!
   ```

2. Load data by running the `m`-file `dataEmotion`. There are five different signal sources (or, actually, five different "modes" of the same source!) collected in the columns of the matrix `DATA`. To have some intuition into the signals, you can try, for example

   ```
   sound(DATA(:,1),16384);
   ```

   You are now asked to search for such features that these signal sources could be distinguished from each other. First divide the signals in shorter sequences of, say, a few hundred samples each, and analyze these sequences separately. Are there some invariances between sequences coming from the same source as compared to the other sources?

   In addition to the time-domain features, you should experiment with, for example, frequency domain features (use `fft`), AR-type (auto-regressive) model parameters, etc. — and you can combine these, defining new features that are based on how fast the "first-order" features change. How robust do you think your features are — would the same phenomena be detected in other samples from the same sources?

# Lesson 4

# "Quick and Dirty"

Since its introduction by C. F. Gauss in the early 1800's, the least-squares parameter matching technique has penetrated to all fields of research and practical engineering work, and it still seems to be among the only ones that are routinely used. However, there are some probloms that are not easily detected — these problems become evident only in the complex modeling tasks, where there is plenty of data that is necessarily not optimally conditioned. In this chapter, the least-squares regression method is first derived, and modifications are presented; finally, the fundamental problem (so called *multicollinearity*) plaguing this method is explained, giving motivation to search for more sophisticated methods.

## 4.1 Linear regression model

As presented in the previous chapter, assume that the measurement data is collected in the matrices $X$ of dimension $k \times n$ and $Y$ of dimension $k \times m$. It is assumed that there are (much) more measurement samples than what is the dimension of the data, that is, $k \gg n$. One would like to find the matrix $F$ so that

$$Y = X \cdot F \tag{4.1}$$

would hold. Finding a good matrix $F$ is the main emphasis from now on. Even though the modeling problem can be formulated in such a simple way, in a multivariate system the task is far from trivial. There are $n \cdot m$ free parameters in the model, and the optimum is searched for in this parameter space.

### 4.1.1 Least-squares solution

To start with, first study a model of just one output signal $Y_i$, so that $m = 1$. The parameter matrix reduces to a vector $F_i$:

$$Y_i = X \cdot F_i. \tag{4.2}$$

Solving for $F_i$ in (4.2) means that somehow $X$ should be inverted; however, $X$ is not invertible, and because $k > n$, generally no exact solutions can be found. To find the best approximation, the model needs to be extended to include the modeling errors as

$$\tilde{Y}_i = X \cdot F_i + E_i, \tag{4.3}$$

where $E_i$ is a $k \times 1$ vector containing the reconstruction error for each measurement sample $k$. It is only these noisy measurements $\tilde{y}$ that are assumed to be available for modeling; in what follows, the sloppy notation $y$ will for brevity still be used to denote the noisy data. Now there are more unknowns than there are constraints, and the problem can be transformed into a form where optimization is being carried out. It needs to be recognized that the formulation in (4.3) is just a model representing the coupling of uncertainty in the system. The variables $e_i(\kappa)$ do not represent any real noise signals in the system, they only stand for the match between the model and the data. In this sense, minimizing this uncertainty is a justified objective.

The errors can be solved from (4.3) as $E_i = Y_i - XF_i$; these errors should be somehow simultaneously minimized. It turns out that the easiest way to proceed — and also theoretically well motivated, as shown in the previous chapter — is to *minimize the sum of error squares.* The sum of the squared errors can be expressed as

$$\begin{aligned} E_i^T E_i &= (Y_i - XF_i)^T (Y_i - XF_i) \\ &= Y_i^T Y_i - Y_i^T X F_i - F_i^T X^T Y_i + F_i^T X^T X F_i. \end{aligned} \tag{4.4}$$

This (scalar) can be differentiated with respect to the parameter vector $F_i$:

$$\frac{d\left(E_i^T E_i\right)}{d\,F_i} = \mathbf{0} - X^T Y_i - X^T Y_i + 2X^T X F_i. \tag{4.5}$$

Because

$$\frac{d^2\left(E_i^T E_i\right)}{d\,F_i^2} = 2X^T X > 0, \tag{4.6}$$

this extremum is minimum, and because the extremum of a quadratic function is unique, setting the derivative to zero (vector) gives the unique optimum parameters:

$$-2X^T Y_i + 2X^T X F_i = \mathbf{0}, \tag{4.7}$$

resulting in

$$F_i = \left(X^T X\right)^{-1} X^T Y_i. \tag{4.8}$$

The estimate for $y_i$ is found as

$$\hat{y}_{\text{est},i} = F_i^T x_{\text{est}} = Y_i^T X \left(X^T X\right)^{-1} x_{\text{est}}. \tag{4.9}$$

This result can be intuitively interpreted also in terms of correlation matrices: First, covariance structure in $x$ is eliminated (multiplication by $\left(\frac{1}{k}X^TX\right)^{-1}$), and after that the "whitened" data is mapped onto $y$ utilizing the cross-correlation structure (as revealed by $\frac{1}{k}X^TY$).

### 4.1.2 Piece of analysis

In what follows, some theoretical analyses that can be used to evaluate the above least squares model are presented. More analysis can be found in various sources, for example, in [35].

#### Model consistency

Because the model construction was an optimization process based on stochastic data, the model parameters cannot be assumed to be quite accurate. Indeed, for the parameter estimates one can write

$$\begin{aligned}\hat{F}_i &= \left(X^TX\right)^{-1}X^TY_i \\ &= \left(X^TX\right)^{-1}X^T\left(XF_i+E_i\right) \\ &= F_i+\left(X^TX\right)^{-1}X^T\cdot E_i.\end{aligned} \tag{4.10}$$

Here, $\hat{F}_i$ are the estimates, whereas $F_i$ is assumed to contain the "true" noiseless parameter values. From this one can write the expression for parameter errors:

$$\tilde{F}_i = F_i-\hat{F}_i = \left(X^TX\right)^{-1}X^T\cdot E_i. \tag{4.11}$$

The expected parameter error is zero, *assuming* that $X$ and $E_i$ do not correlate (this issue is studied later):

$$\mathrm{E}\{\tilde{F}_i\} = \left(X^TX\right)^{-1}X^T\,\mathrm{E}\{E_i\} = 0. \tag{4.12}$$

If this uncorrelatedness assumption does not hold, there will be *bias.* If $X$ is deterministic and $E$ has zero mean, as was assumed, there will be no problem; however, these assumptions cannot always be fulfilled (see Section 4.2.1).

#### Parameter sensitivity

The reliability of the regression model (4.8) can be approximated, for example, by checking how much the parameters vary as there are stochastic variations in $E_i$. The parameter vector covariance matrix becomes, applying (4.11)

$$\begin{aligned}\mathrm{E}\{\tilde{F}_i\tilde{F}_i^T\} &= \mathrm{E}\left\{\left(\left(X^TX\right)^{-1}X^TE_i\right)\left(\left(X^TX\right)^{-1}X^TE_i\right)^T\right\} \\ &= \left(X^TX\right)^{-1}X^T\cdot\mathrm{E}\left\{E_iE_i^T\right\}\cdot X\left(X^TX\right)^{-1} \\ &= \left(X^TX\right)^{-1}X^T\sigma_e^2IX\left(X^TX\right)^{-1} \\ &= \sigma_e^2\left(X^TX\right)^{-1}.\end{aligned} \tag{4.13}$$

The noise variance $\sigma_e^2$ can be approximated as the variance of the reconstruction error $\tilde{Y}_i = Y_i - \hat{Y}_i = Y_i - X\hat{F}_i$. The parameter variance is also intimately related, not only to the noise properties determined by the error variance $\sigma_e^2$, but also to the properties of the matrix $X^T X$ — see Sec. 4.3 for more analysis.

The estimate for the parameter error can be applied, for example, when assessing the relevance of the input variables $x_j$. For example, assume that a least-squares model is constructed, and the corresponding diagonal element in the model parameter covariance matrix is $\mathrm{E}\{\tilde{F}_{jj}^2\} = \sigma_{jj}^2$. Now, assuming that the probability density function form for the error is known (Gaussian?), one can approximate the probability that the parameter $F_{jj}$, rather than being the estimated $\tilde{F}_{ii}$, actually has zero value. This would mean that there is no contribution of that variable in the model, and it could be ignored.

Without going into details, it turns out that the expression for parameter covariance (4.13) reaches the *Cramer-Rao lower bound,* meaning that for Gaussian data the least-squares model implements the best possible, or *efficient,* estimator for the parameters.

### Measures of fit

To evaluate how well the regression model matches the training data, the so called $R$ *squared* criterion can be applied: how much of the real output variance can be explained by the model. That is, one can calculate the quantity

$$R^2 = 1 - \frac{SS_{\mathrm{E}}}{SS_{\mathrm{T}}}, \tag{4.14}$$

where, for the $i$'th output,

- The "error sum of squares" is defined as

$$SS_{\mathrm{E}} = (Y_i - \hat{Y}_i)^T (Y_i - \hat{Y}_i) = (Y_i - XF_i)^T (Y_i - XF_i). \tag{4.15}$$

- The "total sum of squares" (for zero-mean data) is

$$SS_{\mathrm{T}} = Y_i^T Y_i. \tag{4.16}$$

So, $R^2$ measures how much of the total variation in the output can be explained by the model. This quantity has value 1 if all the variation in the output can be exactly predicted, and lower value otherwise.

This $R^2$ is a traditional measure for characterizing least-squares fitting. However, it needs to be emphasized here that it is *not* a good approach for evaluating model goodness. It simply measures data fit, not model goodness: It uses the same data for evaluation that was used for model construction. Applying this criterion for comparing model structures, the least-squares model would always outperform the other structures (to be studied later), no matter how sensitive the model is to noise!

### 4.1.3 Multivariate case

If there are various output signals, so that $m > 1$, the above analysis can be carried out for each of them separately. When collected together, there holds

$$\begin{cases} F_1 &= \left(X^T X\right)^{-1} X^T Y_1 \\ &\vdots \\ F_m &= \left(X^T X\right)^{-1} X^T Y_m. \end{cases} \tag{4.17}$$

It turns out that this set of formulas can simultaneously be rewritten in a compact matrix form, so that

$$F = \left( \begin{array}{c|c|c} F_1 & \cdots & F_m \end{array} \right) = \left(X^T X\right)^{-1} X^T \cdot \left( \begin{array}{c|c|c} Y_1 & \cdots & Y_m \end{array} \right). \tag{4.18}$$

This means that the *multilinear regression (MLR)* model from $X$ to estimated $Y$ can be written as

$$F_{\text{MLR}} = \left(X^T X\right)^{-1} X^T Y. \tag{4.19}$$

The MLR solution to modeling relationships between variables is exact and optimal in the sense of the least squares criterion, implementing the *pseudoinverse* of the matrix $X$. However, in Sec. 4.3 it will be shown that one has to be careful when using this regression method: In practical applications and in nonideal environments this MLR approach *may collapse altogether.* The problem is that trying to explain noisy data too exactly may make the model sensitive to individual noise realizations. In any case, in later chapters, the above MLR model is used as the basic engine to reach mappings between variables; the deficiencies of the basic approach are taken care of separately.

The basic MLR solution can be extended and modified in many ways. For example, assuming that not all samples are assumed to be equally informative, one can define the weighted cost criterion for output $i$ as

$$J_i = \sum_{\kappa=1}^{k} w(\kappa) \cdot e_i^2(\kappa) = E_i^T W E_i, \tag{4.20}$$

where the $k \times k$ matrix $W$ contains the weighting factors on the diagonal. then the solution (as expanded to multiple outputs) as

$$F = \left(X^T W X\right)^{-1} X^T W Y. \tag{4.21}$$

The parallel structure of the multivariate problems can be utilized also more generally for extending formulas to multivariate cases. For example, the exponentially weighted (so that $w(\kappa) = \lambda^{k-\kappa}$ with the forgetting factor $0 \ll \lambda$¡1) recursive least-squares algorithm [**?**] corresponding to (4.21) can be extended to multiple outputs, so that $m > 1$, as

$$\begin{array}{rcl} F(k) &=& F(k-1) + \left(R(k)\right)^{-1} x(k) \left(y(k) - F^T(k) x(k)\right)^T \\ R(k) &=& \lambda R(k-1) + x(k) x^T(k). \end{array} \tag{4.22}$$

## 4.2 "Colored noise"

The above MLR formula will be the standard approach to implementing the mapping between two sets of variables in later chapters. As was observed, it is optimal and efficient — but only if the mapping problem is appropriately conditioned. There are two basic problems that will be discussed during the rest of this chapter. Both of the problems become acute in multivariate cases where the quality of the high-dimensional data cannot be assured.

From the practical point of view, the first problem is caused by the *incompatible model structure* assumption; this issue is studied in this section. In Section 4.3, it is *the robustness problem* that is studied.

### 4.2.1 Error in variables

In the beginning, it was assumed that the nature of the variables is heterogeneous: It was assumed that $Y$ only is stochastic, noise $E$ being added to it, and $X$ was assumed to be deterministic. To understand the problem of deterministic vs. stochastic variables, study an example.

Now, study a familiar-looking case: Assume that system dynamics is to be modeled:

$$y(k) = F^T \cdot x(k) + e(k), \tag{4.23}$$

where

$$x(k) = \begin{pmatrix} \tilde{y}(k-1) \\ \vdots \\ \tilde{y}(k-n) \end{pmatrix} = \begin{pmatrix} y(k-1) + e(k-1) \\ \vdots \\ y(k-n) + e(k-n) \end{pmatrix}. \tag{4.24}$$

When the input $x$ is defined so that the former outputs are "recirculated" into input, one is identifying the *auto-regressive* (AR) model structure. It is clear that the assumption of $x$ being deterministic collapses; what is more, $X$ becomes correlated with $E$, so that the model in (4.11) becomes biased.

Clearly, it has to be assumed that $X$ measurements can also contain uncertainty. The model matching problem becomes very different if both $X$ and $Y$ blocks are regarded as equally stochastic data values and errors in all variables should be taken into account (see Fig. 4.1). This assumption results in the so called Error In Variables (EIV) model.

There exists a wide variety of different ways to implement the data homogeneity in the models. As an example, below, one approach is presented for circumventing the problems of correlated noise. A more concise treatment is carried out in the next chapter, where the problem is attacked from a fresh point of view[1].

[1] In chapter 11, a method called Total Least Squares is presented that also addresses Errors in Variables

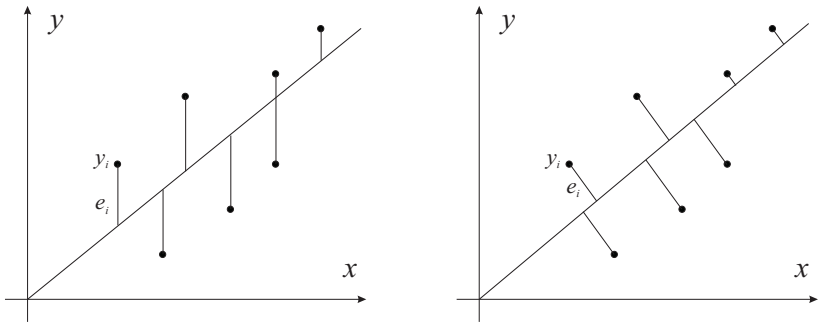



Figure 4.1: Normal least-squares matching principle, on the left, assuming that only the $y$ variables contain noise, and the *total least squares* principle, assuming that errors in all variables are equally meaningful, on the right. For visualization purposes only one $x$ and $y$ variable is employed

### 4.2.2 Instrumental variables

When constructring linear regression models, after all, it is all about inverting the mapping: Starting from $XF = Y$ solve the mapping matrix $F$. The challenge is caused by the uninvertibility of the matrix $X$. However, if the original model holds, there must also hold $\mathcal{X}^T Y = \mathcal{X}^T XF$, where $\mathcal{X}$ is some $k \times n$ matrix. Now, assuming that the matrix $\mathcal{X}^T X$ is invertible, one can solve

$$F = \left(\mathcal{X}^T X\right)^{-1} \mathcal{X}^T Y. \tag{4.25}$$

As in (4.10), one can find the correspondence between the noise and the parameter matrix

$$\hat{F} = F + \left(\mathcal{X}^T X\right)^{-1} \mathcal{X}^T E, \tag{4.26}$$

and, further, for the parameter error one has

$$\tilde{F} = \left(\mathcal{X}^T X\right)^{-1} \mathcal{X}^T E. \tag{4.27}$$

It is interesting here that it is no more the correlation between $X$ and $E$ that determines the model bias: To minimize the model error, there should be high correlation between $\mathcal{X}$ and $X$, and low correlation between $\mathcal{X}$ and $E$. Naturally, the first objective is reached for $\mathcal{X} = X$, resulting in the nominal MLR, but when the other objective is also emphasized, non-trivial alternatives can be proposed. The variables in § are called *instrumental variables.*

How to reach good properties for the instruments, is dependent of the situation. For example, if in $y(\kappa) = f^T x(\kappa)$ the $x(\kappa)$ data vector consists of the past values of (scalar) $y(\kappa)$, as shown in (4.24), meaning that AR modeling of a dynamic system is being carried out, different choices have been studied a lot. A good choice for instruments in such case would be to use the correct (noiseless) values of $y(\kappa)$ as collected in §$(\kappa)$: This would be the optimal choice, and, indeed,

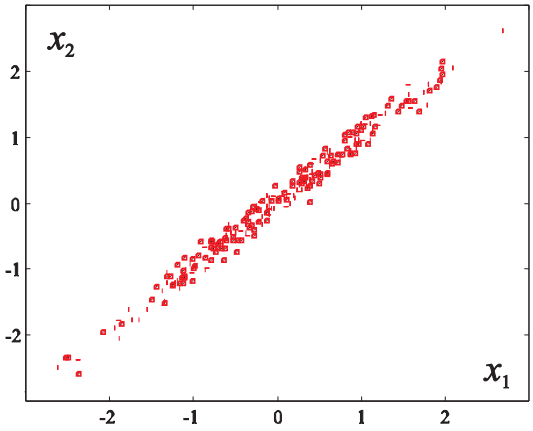



Figure 4.2: Collinearity visualized in two dimensions

this can be approximately implemented. When using the model, in the reconstructed values of $y$ the noise realization has (hopefully) been abstracted away, and these estimates can be used as instruments: Select $\S(\kappa) = \hat{x}(\kappa)$, where $\hat{x}(\kappa)$ vector consists of the past values of $\hat{y}(\kappa)$. When the cycle of first determining a preliminary model and thereafter refining the instruments is repeated, the model parameters finally converge to unbiased values.

## 4.3 Collinearity

In the previous section, the problem of data heterogeneity was discussed. The deficiencies of MLR become even more painstaking when dimensional complexity is faced.

The MLR regression model is optimal[2]. In simple cases it is difficult to see why *optimality* is in contrast with *usability.* Today, when the problems to be modeled involve large amounts of poor data, the problems of MLR have become evident. The main problem plaguing MLR is caused by *(multi)collinearity.* What this means can best be explained using an example.

### 4.3.1 Example: When variables are redundant

Assume that one can observe two variables $x_1$ and $x_2$, so that $x = (\ x_1 \quad x_2\ )^T$. Further, assume that these variables are not strictly independent; they can be written as $x_1(\kappa) = \xi(\kappa) + \epsilon_1(\kappa)$ and $x_2(\kappa) = \xi(\kappa) + \epsilon_2(\kappa)$, where the sequences $\epsilon_1(\kappa)$ and $\epsilon_2(\kappa)$ are mutually uncorrelated, both having the same variance $\sigma^2$. This can be interpreted so that we have two noisy measurements of the same underlying variable $\xi$, and together these measurements should give a more reliable estimate for it.

[2]of course, only in the least-squares sense; but, because of the mathematical benefits, the same criterion will be applied later, too

Let us check what this collinearity of $x_1$ and $x_2$ means in practice. First, calculate the matrix $X^TX$ that has an essential role in the regression formula

$$\begin{aligned} X^TX &= \begin{pmatrix} \sum_\kappa x_1^2(\kappa) & \sum_\kappa x_1(\kappa)x_2(\kappa) \\ \sum_\kappa x_1(\kappa)x_2(\kappa) & \sum_\kappa x_2^2(\kappa) \end{pmatrix} \\ &\approx k \cdot \begin{pmatrix} \mathrm{E}\{\xi^2\} + \sigma^2 & \mathrm{E}\{\xi^2\} \\ \mathrm{E}\{\xi^2\} & \mathrm{E}\{\xi^2\} + \sigma^2 \end{pmatrix}. \end{aligned} \tag{4.28}$$

To understand the properties of the regression formula, let us study the eigenvalues of the above matrix. It turns out that the solutions to the eigenvalue equation

$$\det\left\{\lambda \cdot I_2 - k \cdot \begin{pmatrix} \mathrm{E}\{\xi^2(\kappa)\} + \sigma^2 & \mathrm{E}\{\xi^2(\kappa)\} \\ \mathrm{E}\{\xi^2(\kappa)\} & \mathrm{E}\{\xi^2(\kappa)\} + \sigma^2 \end{pmatrix}\right\} = 0 \tag{4.29}$$

are

$$\begin{cases} \lambda_1 = 2k \cdot \mathrm{E}\{\xi^2(\kappa)\} + k\sigma^2, & \text{and} \\ \lambda_2 = k\sigma^2. \end{cases} \tag{4.30}$$

The theory of matrices reveals that the *condition number* of a matrix determines its numerical properties — that is, the ratio between its largest and smallest eigenvalue dictates how vulnerable the formulas containing it are to unmodeled noise. As the condition number grows towards infinity the matrix becomes gradually uninvertible. In this case, the matrix $X^TX$ has the condition number

$$\mathrm{cond}\{X^TX\} = 1 + 2 \cdot \frac{\mathrm{E}\{\xi^2(\kappa)\}}{\sigma^2}, \tag{4.31}$$

telling us that the smaller the difference between the variables $x_1$ and $x_2$ is ($\sigma^2$ being small), the higher the sensitivity of the regression formula becomes.

The above result reveals that when using regression analysis, one has to be careful: It is the matrix $X^TX$ that has to be inverted, and problems with invertibility are reflected in the model behavior. There only need to exist two linearly dependent measurements among the variables in $x$, and the problem instantly becomes ill-conditioned. In practice, it may be extremely difficult to avoid this kind of "almost" collinear variables — as an example, take a system that has to be modeled using partial differential equation (PDE) model (say, a rod that is being heated). PDE models are often called "infinite-dimensional"; that is, one needs very high number (in principle, infinitely many) measurements to uniquely determine the process state. It is not a surprise that temperature readings along the rod do not change rapidly, or nearby measurements deliver almost identical values, variables becoming linearly dependent; a regression model trying to utilize all the available information becomes badly behaving. When aiming towards accuracy, the model robustness is ruined!

To see an example of what collinear data looks like in a two-dimensional space, see Fig. 4.2: the data points in the figures are created using the above model, where $\mathrm{E}\{\xi^2(\kappa)\} = 1.0$ and $\sigma^2 = 0.01$, the sequences being normally distributed random processes. The data points seem to be located along a line; they do not really seem to "fill" the whole plane. Intuitively, this is the key to understanding the ideas of further analyses in later chapters.

The TLS approach by no means solves the above collinearity problem — on the contrary, even more severe problems emerge. Note that the last principal component essentially spans the null space of the covariance matrix, that is, if there is linear dependency among the variables, this dependency dominates in $f'$. Assuming that the linear dependency is between, say, input variables $x_i$ and $x_j$, the parameters $f'_i$ and $f'_j$ have high values, all other coefficients being near zero. Now, if (11.21) is applied, the parameter $f'_y$ (having negligible numerical value) in the denominator makes the model badly conditioned. The main problem with TLS is that while solving a minor problem (error in variables), it may introduce more pathological problems in the model.

### 4.3.2 Patch fixes

Because of the practical problems caused by collinearity, various ways to overcome the problems have been proposed. In what follows, two of such propositions are briefly presented — more sophisticated analyses are concentrated on in next chapters.

#### Orthogonal least squares

Because the basic source of problems in linear regression is related to inversion of the matrix $X^T X$, one can try to avoid the problem by enhancing the numerical properties of this matrix. Intuitively, it is clear that if the input variables were mutually orthogonal, so that $X^T X = I$, the numerical properties would be nice. Indeed, one can construct new variables $Z$ so that this orthogonality holds using the so called *Gram-Schmidt procedure:* Corresponding to all indices $1 \leq i \leq n$, define $Z_i$ by

$$Z'_i = X_i - \sum_{j=1}^{i-1} X_i^T Z_j \cdot Z_j, \tag{4.32}$$

and normalize it,

$$Z_i = Z'_i / \sqrt{Z_i'^T Z'_i}, \tag{4.33}$$

starting from $Z_1 = X_1 / \sqrt{X_1^T X_1}$. These data manipulation operations can be presented in a matrix form

$$Z = X \cdot M, \tag{4.34}$$

where $M$ is an upper-triangular matrix[3]. It is easy to see that there holds

$$Z_i^T Z_j = \begin{cases} 1, & \text{if } i = j, \text{ and} \\ 0, & \text{otherwise,} \end{cases} \tag{4.35}$$

[3]Actually, the so called *QR factorization* of $X$ that is readily available, for example, in `Matlab`, gives the same result (note that the resulting `R` matrix is the inverse of our $M$. The inversions of the triangular matrix are, however, nicely conditioned)

so that $Z^T Z = I$. Using these intermediate variables one has the mapping matrix from $Z$ to $Y$ as

$$F = \left(Z^T Z\right)^{-1} Z^T Y = Z^T Y, \tag{4.36}$$

or returning to the original variables $X$, the *Orthogonal Least Squares (OLS)* formula becomes

$$F_{\text{OLS}} = M Z^T Y. \tag{4.37}$$

Of course, reformatting formulas does not solve the fundamental problems — the inversion of the matrix is implicitly included in the construction of $M$. However, it turns out that reorganizing the calculations still often enhances the numerical properties of the problem.

### Ridge regression

Ridge Regression (RR) is another (ad hoc) method of avoiding the collinearity problem — the basic idea is to explicitly prevent the covariance matrix from becoming singular. Ridge regression belongs to a large class of *regularization* methods where the numerical properties of the data — as seen by the algorithms — are somehow enhanced. The idea here is not to minimize exclusively the squared error, but to include weighting for *parameter size* in the optimization criterion: The badly-behaving nature of models is reflected in excessive parameter values. Instead of (4.4), the criterion that is really minimized is

$$\begin{aligned} E_i^T E_i + F_i^T Q_i F_i = \\ Y_i^T Y_i - Y_i^T X F_i - F_i^T X^T Y_i + F_i^T X^T X F_i + F_i^T Q_i F_i, \end{aligned} \tag{4.38}$$

where $Q_i$ is a positive definite weighting matrix. Differentiation yields

$$\frac{d\left(E_i^T E_i\right)}{dF_i} = \mathbf{0} - X^T Y_i - X^T Y_i + 2X^T X F_i + 2Q_i F_i. \tag{4.39}$$

Setting the derivative to zero again gives the optimum:

$$-2X^T Y_i + 2X^T X F_i + 2Q_i F_i = \mathbf{0}, \tag{4.40}$$

resulting in

$$F_i = \left(X^T X + Q_i\right)^{-1} X^T Y_i. \tag{4.41}$$

In the multi-output case, assuming that $Q_i = Q$ is the same for all outputs, one can compactly write

$$F_{\text{RR}} = \left(X^T X + Q\right)^{-1} X^T Y. \tag{4.42}$$

Usually there is no a priori information about the parameter values and the weighting matrix $Q$ cannot be uniquely determined. The normal procedure is

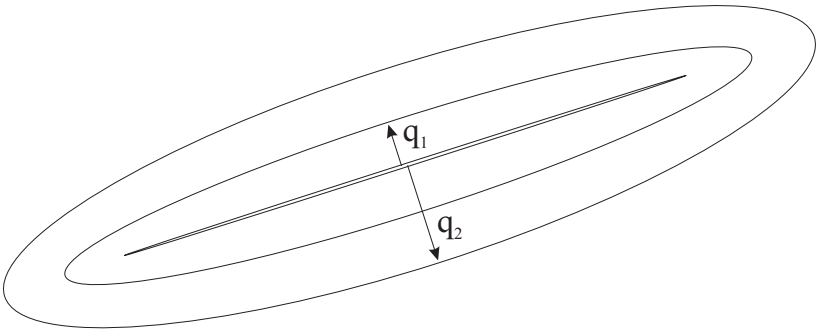



Figure 4.3: The "virtual" distribution of collinear data as seen by the ridge regression algorithm for different values of $q$

to let $Q$ be diagonal; what is more, it is often chosen as $Q = q \cdot I$, where $q > 0$ is a small number. This approach efficiently prevents the matrix being inverted from becoming singular.

The key point here is that the matrix $Q$ is added to the (unscaled) data covariance matrix. Study the eigenvalues; another way to determine the eigenvalues is to solve the determinant expression

$$\left|\left(X^T X + q \cdot I\right) - \lambda I\right| = \left|X^T X - (\lambda - q) \cdot I\right|. \tag{4.43}$$

When adding $q\,I$ to the matrix $X^T X$, its all eigenvalues are shifted up by the amount $q$, so that originally zero eigenvalues will have numerical value $q > 0$. The condition number also goes down. The model parameters are typically more conservative than in the nominal MLR case.

Note that the same ridge regression behavior in standard MLR is achieved also if white noise with covariance $\frac{1}{k}\,q\,I$ is added to data: If this added noise does not correlate with $X$ — this assumption is easily fulfilled because the noise is artificial, being added in the algorithm — the noise-corrupted data covariance matrix is $\frac{1}{k}\left(X^T X + q\,I\right)$. This regularization approach is often explicitly used, for example, when training neural networks.

It seems that there are essentially two ways to enhance the invertibility of the matrix $X^T X$, and thus the MLR regression model properties:

1. Either, one can *ignore information* by leaving the "redundant" variables out. The problem here is that there are typically no variables with no information at all, even though this information can be highly redundant, and such variable elimination necessarily makes the model ignore available information.

2. Or, one can *introduce disinformation* by adding noise in the variables. This is effectively done when implementing regularization.

Just think of it: Either information is ignored, or noise is deliberately added to data just to make the model better behaving! There is an uneasy feeling of heuristics here, and something more sophisticated is clearly needed — the

modeling method should be matched with the data, not *vice versa.* Alternatives to MLR are presented in the following chapters.

## Computer exercises

1. Check how the MLR sensitivity is affected when the data properties are changed; that is, try different values for the parameters $k$ (number of samples), $n$ (data dimension), $dof_x$ (true degrees of freedom), and $\sigma_x$ (deviation of the noise) below, and calculate the covariance matrix condition number:

```
k = 20;
n = 10;
dofx = 5;
sigmax = 0.001;
X = dataXY(k,n,NaN,dofx,NaN,sigmax);
Lambda = eig(X'*X/k);
max(Lambda)/min(Lambda)
```

2. Study how robust the different regression algorithms are. First generate data, and test the methods using cross-validation (try this several times for fresh data):

```
[X,Y] = dataXY(20,10,5,5,3,0.001,1.0);
E = regrCrossVal(X,Y,'regrMLR(X,Y)');
errorMLR = sum(sum(E.*E))/(20*5)
E = regrCrossVal(X,Y,'regrTLS(X,Y)');
errorTLS = sum(sum(E.*E))/(20*5)
E = regrCrossVal(X,Y,'regrOLS(X,Y)');
errorOLS = sum(sum(E.*E))/(20*5)
E = regrCrossVal(X,Y,'regrRR(X,Y,0.001)');   % Change this!
errorRR = sum(sum(E.*E))/(20*5)
```

# Lesson 5

# Tackling with Redundancy

The collinearity problem is essentially caused by redundancy in the data: Measurements are more or less dependent of each other. However, none of the measurements is *completely* useless, each of them typically delivers *some* fresh information. Qualitative analyses cannot help here — on the other hand, when the quantitative approach is adopted, powerful methods turn out to be readily available.

## 5.1 Some linear algebra

*Linear algebra* is a highly abstract field of systems theory. In this context, it suffices to concentrate on just a few central ideas, and theoretical discussions are kept in minimum; these issues are studied in more detail, for example, in [15] or [33].

### 5.1.1 On spaces and bases

To have a deeper understanding of how the mapping from the "space" of input variables into the "space" of output variables can be analyzed, basic knowledge of linear algebra is needed. The main concepts are *space, subspace,* and *basis.* The definitions are briefly summarized below:

> The set of all possible real-valued vectors $x$ of dimension $n$ constitutes the *linear space* $\mathcal{R}^n$. If $\mathcal{S} \in \mathcal{R}^n$ is a set of vectors, a *subspace* spanned by $\mathcal{S}$, or $\mathcal{L}(\mathcal{S})$, is the set of all linear combinations of the vectors in $\mathcal{S}$. An (ordered) set of linearly independent vectors $\theta_i$ spanning a subspace is called a *basis* for that subspace.

Geometrically speaking, subspaces in the $n$ dimensional space are hyperplanes (lines, planes, etc.) that go through the origin. The number of linearly independent vectors in the subspace basis determines the *dimension* of the subspace. The basis vectors $\theta_1$ to $\theta_N$ can conveniently be represented in a matrix form:

$$\theta = \left( \begin{array}{c|c|c} \theta_1 & \cdots & \theta_N \end{array} \right). \tag{5.1}$$

This basis matrix has dimension $n \times N$, assuming that the dimension of the subspace in the $n$ dimensional space is $N$. Given a basis, all points $x$ in that subspace have a unique representation; the basis vectors $\theta_i$ can be interpreted as coordinate axes in the subspace, and the "weights" of the basis vectors, denoted now $z_i$, determine the corresponding "coordinate values" (or *scores*) of the point:

$$x = \sum_{i=1}^{N} z_i \cdot \theta_i. \tag{5.2}$$

The elements in $\theta_i$ are called *loadings* of the corresponding variables. In matrix form, the above expression can be written as

$$x = \theta \cdot z. \tag{5.3}$$

Further, if there are various data vectors, the matrix formulation can be written as

$$X = Z \cdot \theta^T. \tag{5.4}$$

There is an infinite number of ways of choosing the basis vectors for a (sub)space. One basis of special relevance is the so called "natural" basis: fundamentally, all other bases are defined with respect to this natural basis. For the space of $n$ measurements the natural basis vector directions are determined directly by the measurement variables; formally speaking, each entry in the data vector can be interpreted as a coordinate value, the basis vectors constituting an identity matrix, $\theta = I_n$.

However, even though this trivial basis is easy to use, it is not necessarily mathematically the best representation for the data (as was shown in the example about collinearity above). Next we see how to change the basis.

### 5.1.2 About linear mappings

The matrix data structure has been adopted here for various purposes — this is partly duw to the role of `Matlab` as the assumed basic tool: There (at least originally) the matrix was the only one data structure available. The matrix can have various roles. It can be used as a collection of data values (as $X$ and $Y$ above, for example), or it can be used as a frame for a vector system (as in the case of basis vectors); but perhaps the most important role of a matrix is its use as a means of accomplishing *linear transformations* between different bases of (sub)spaces.

Whereas all matrix operations can be interpreted as linear transformations, now we are specially interested in mappings between different bases. The transformations from a given basis to the natural basis are straightforward: applying (5.3) gives the transformed coordinates directly. The question that arises is how one can find the coordinate values $z$ for a given $x$ when the new basis $\theta$ is given. There are three possibilities depending on the dimensions $n$ and $N$:

- If $n \equiv N$, matrix $\theta$ is square and invertible (because the linear independence of the basis vectors was assumed). Then one can directly solve

$$z = \theta^{-1} \cdot x. \tag{5.5}$$

- If $n > N$, the data point cannot necessarily be represented in the new basis. Using the least squares technique (see the first lesson) results in an approximation

$$\hat{z} = \left(\theta^T \theta\right)^{-1} \theta^T \cdot x. \tag{5.6}$$

- If $n < N$, there are an infinite number of exact ways of representing the data point in the new basis. Again, the least squares method offers the solution, now in the sense of minimizing $z^T z$, that is, finding the minimum numerical values of the coordinates (see page 20):

$$z = \theta^T \left(\theta\theta^T\right)^{-1} \cdot x. \tag{5.7}$$

All of the above cases can be conveniently presented using the *pseudoinverse* notation:

$$z = \theta^{\dagger} \cdot x. \tag{5.8}$$

If the basis vectors are *orthonormal* (orthogonal and normalized at the same time, meaning that $\theta_i^T \theta_j = 0$, if $i \neq j$, and $\theta_i^T \theta_j = 1$, if $i = j$) there holds $\theta^T \theta = I_N$ (or $\theta\theta^T = I_n$, whichever is appropriate). Thus, all the above formulas (5.5), (5.6), and (5.7) give a very simple solution:

$$z = \theta^T \cdot x, \tag{5.9}$$

or, corresponding to (5.4),

$$Z = X \cdot \theta. \tag{5.10}$$

The above result visualizes the benefits of basis orthonormality; there are additional advantages that are related to the numerical properties of orthogonal transformation matrices (manipulations in an orthonormal basis are optimally conditioned)[1].

### 5.1.3 Data model revisited

To enhance the basic regression method, a more sophisticated scheme is now adopted (see Fig. 5.1). Speaking informally, we search for an "internal structure" that would capture the system behavior optimally; this internal structure is assumed to be implemented as a linear subspace. The data samples are first

[1]Note that the orthogonality condition is always fulfilled by the basis vectors that are generated by the PCR and PLS approaches that will be presented later. Furthermore, when using `Matlab`, say, for calculating the eigenvectors, they will be automatically normalized; this means that the practical calculations are rather straightforward

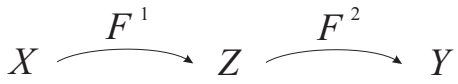



Figure 5.1: The dependency model $y = f(x)$ refined

projected onto the internal basis, and from there they are further projected onto the output space, the final projection step being based on MLR regression. Note that because all of the mappings are linear, they can always be combined so that, seen from outside, the original "one-level" model structure is still valid: $Y = (XF^1)F^2 = X(F^1F^2) = XF$.

Now there are approximate mappings instead of only one, as in the MLR case. Is it not so that the regression model will become even more sensitive to noise? However, it is not so. It is not the number of mappings, it is the properties of these mappings that matter — and now, as it turns out, the mapping from input to the latent variables and the mapping from latent variables to output can be made well-conditioned.

The overall regression model construction becomes a two-phase process, so that there are the following tasks:

1. Determine the basis $\theta$.
2. Construct the mapping $F^1 = \theta \left(\theta^T\theta\right)^{-1}$.
3. Calculate the "latent variables" $Z = XF^1$.
4. Construct the second-level mapping $F^2 = \left(Z^TZ\right)^{-1} Z^TY$.
5. Finally, estimate $\hat{Y}_{\text{est}} = X_{\text{est}}F = X_{\text{est}}F^1F^2$.

Here $Z$ stands for the internal coordinates corresponding to the training data $X$ and $Y$. In special cases (for example, for orthonormal $\theta$) some of the above steps may be simplified. The remaining problem is to determine the basis $\theta$ so that the regression capability would be enhanced.

How the internal structure should be chosen so that some benefits would be reached? When the rank of the basis is the same as the number of degrees of freedom in the data (normally meaning that there are $n$ basis vectors representing the $n$ dimensional data), the data can be exactly reconstructed, or the mapping between data and the transformed representation can be inverted. This means that also the random noise that is present in the samples will always remain there. A good model, however, should only represent the *relevant* things, ignoring something, hopefully implementing this compression of data in a clever way. In concrete terms, this data compression means dimension reduction, so that there are *fewer* basis vectors than what is the dimension of the data, or $N < n$.

Let us study this a bit closer — assume that the dimension of input is $n$,

the dimension of output is $m$, the dimension of the latent basis is $N$, and the number of samples is $k$. The nominal regression model, matrix $F$ mapping input to output contains $n \cdot m$ free parameters; there are $k \cdot m$ constraint equations. This means that on average, there are

$$\frac{k \cdot m}{n \cdot m} = \frac{k}{n} \tag{5.11}$$

constraints for each parameter. The higher this figure is, the better the estimate becomes in statistical sense, random noise having smaller effect. On the other hand, if the latent basis is used in between the input and output, there is first the mapping from input to the latent basis ($n \cdot N$ parameters) and additionally the mapping from the latent basis to the output ($N \cdot m$ parameters). Altogether the average number of constraints for each parameter is

$$\frac{k \cdot m}{n \cdot N + N \cdot m} = \frac{k}{N\left(1 + \frac{n}{m}\right)}. \tag{5.12}$$

Clearly, if $N \ll n$, benefits can be achieved, or the model sensitivity against random noise can be minimized — of course, assuming that these $N$ latent variables can carry all the relevant information.

How an automatic modeling machinery can accomplish such a clever thing of compression, or "abstracting" the data? There are different views of how the relevant phenomena are demonstrated in the data properties. Speaking philosophically, it is the *ontological assumption* that is left to the user: The user has to decide what are the most interesting features carrying most of the information about the system behavior. Concentrating on different aspects and utilizing the statistical properties of the data accordingly results in different regression methods.

## 5.2 Principal components

The hypothesis that will now be concentrated on is that *data variance carries information.* This is the assumption underlying *Principal Component Analysis (PCA),* also known as *Karhunen–Loeve decomposition,* and the corresponding regression method PCR. In brief, one searches for the directions in the data space where the data variation is maximum, and uses these directions as basis axes for the internal data model. Whereas noise is (assumed to be) purely random, consistent correlations between variables hopefully reveal something about the real system structure.

Assume that $\theta_i$ is the maximum variance direction we are searching for. Data points in $X$ can be projected onto this one-dimensional subspace determined by $\theta_i$ simply by calculating $Z_i = X\theta_i$; this gives a vector with one scalar number for each of the $k$ measurement samples in $X$. The (scalar) variance of the projections can be calculated[2] as $E\{z_i^2(k)\} = \frac{1}{k} \cdot Z_i^T Z_i = \frac{1}{k} \cdot \theta_i^T X^T X \theta_i$. Of

[2]Here, again, maximum degrees of freedom existent in the data is assumed; for example, if the centering for the data is carried out using the sample mean, the denominator should be $k - 1$. However, this scaling does not affect the final result, the directions of the eigenvectors

course, there can only exist a solution if the growth of the vector $\theta_i$ is restricted somehow; the length of this vector can be fixed, so that, for example, there always holds $\theta_i^T \theta_i = 1$. This means that we are facing a constrained optimization problem (see Sec. 1.2.4) with

$$\begin{cases} f(\theta_i) &= \frac{1}{k} \cdot \theta_i^T X^T X \theta_i, \quad \text{and} \\ g(\theta_i) &= 1 - \theta_i^T \theta_i. \end{cases} \tag{5.13}$$

Using the the method of Lagrange multipliers, the optimum solution $\theta_i$ has to obey

$$\frac{d\, J(\theta_i)}{d\theta_i} = \frac{d}{d\theta_i} \left( f(\theta_i) - \lambda_i \cdot g(\theta_i) \right) = \mathbf{0} \tag{5.14}$$

or

$$2\frac{1}{k} \cdot X^T X \theta_i - 2\lambda_i \theta_i = \mathbf{0}, \tag{5.15}$$

giving

$$\frac{1}{k} X^T X \cdot \theta_i = \lambda_i \cdot \theta_i. \tag{5.16}$$

Now, the variance maximization has become an *eigenvalue problem* with the searched basis vector $\theta_i$ being an eigenvector of the matrix $R = \frac{1}{k} \cdot X^T X$. The eigenvectors of the data covariance matrix are called *principal components.*

Because of the eigenproblem structure, if $\theta_i$ fulfills the equation (5.16), so does $\alpha\theta_i$, where $\alpha$ is an arbitrary scalar; it will be assumed that the eigenvectors are always normalized to unit length, so that $\theta_i^T \theta_i = 1$.

The solution to the variance maximization problem is also given by some of the eigenvectors — but there are $n$ of them, which one to choose? Look at the second derivative:

$$\frac{d^2\, J(\theta_i)}{d\theta_i^2} = \frac{2}{k} \cdot X^T X - 2\lambda_i \cdot I. \tag{5.17}$$

To reach the maximum of $J(\theta_i)$, there must hold $d^2\, J(\theta_i)/d\theta_i^2 \leq \mathbf{0}$, that is, the second derivative matrix (Hessian) must be semi-negative definite: For any vector $\xi$ there must hold

$$\xi^T \cdot \left( \frac{2}{k} \cdot X^T X - 2\lambda_i \cdot I \right) \cdot \xi \leq 0. \tag{5.18}$$

For example, one can select $\xi$ as being any of the eigenvectors, $\xi = \theta_j$:

$$\begin{aligned} &\theta_j^T \cdot \left( \tfrac{2}{k} \cdot X^T X - 2\lambda_i \cdot I \right) \cdot \theta_j \\ &\qquad = \tfrac{2}{k} \cdot \theta_j^T \cdot X^T X \cdot \theta_j - 2\lambda_i \cdot \theta_j^T \theta_j \\ &\qquad = 2\lambda_j \cdot \theta_j^T \theta_j - 2\lambda_i \cdot \theta_j^T \theta_j \\ &\qquad = 2\lambda_j - 2\lambda_i \leq 0. \end{aligned} \tag{5.19}$$

This always holds regardless of the value of $1 \leq j \leq n$ only for the eigenvector $\theta_i$ corresponding to the largest eigenvalue.

### 5.2.1 Eigenproblem properties

Let us study closer the eigenvalue problem formulation (5.16).

#### Symmetricity and non-negativity of eigenvalues

It seems that the matrix $R = \frac{1}{k} \cdot X^T X$ (or the data covariance matrix) determines the properties of the PCA basis vectors, and, indeed, these properties turn out to be very useful. First, it can be noted that $R$ is *symmetric,* because there holds

$$R^T = \left(\frac{1}{k} \cdot X^T X\right)^T = \frac{1}{k} \cdot X^T X = R. \tag{5.20}$$

Next, let us multiply (5.16) from left by the vector $\theta_i^T$ (note that, of course, this vector is rank deficient, and only "one-way" implication can be assumed):

$$\frac{1}{k} \cdot \theta_i^T X^T \cdot X\theta_i = \lambda_i \cdot \theta_i^T \theta_i. \tag{5.21}$$

This expression consists essentially of two dot products ($\theta_i^T X^T \cdot X\theta_i$ on the left, and $\theta_i^T \cdot \theta_i$ on the right) that can be interpreted as squares of vector lengths. Because these quantities must be real and non-negative, and because $k$ is positive integer, it is clear that the eigenvalue $\lambda_i$ is always *real* and *non-negative.*

#### Orthogonality of eigenvectors

Let us again multiply (5.16) from left; this time by *another* eigenvector $\theta_j^T$:

$$\theta_j^T R\theta_i = \lambda_i \cdot \theta_j^T \theta_i. \tag{5.22}$$

Noticing that because $R$ is symmetric (or $R = R^T$), there must hold $\theta_j^T R = (R^T\theta_j)^T = (R\theta_j)^T = \lambda_j \theta_j^T$, so that we have an equation

$$\lambda_j \cdot \theta_j^T \theta_i = \lambda_i \cdot \theta_j^T \theta_i, \tag{5.23}$$

or

$$(\lambda_i - \lambda_j) \cdot \theta_j^T \theta_i = 0. \tag{5.24}$$

For $\lambda_i \neq \lambda_j$ this can only hold if $\theta_j^T \theta_i = 0$. This means that for a symmetric matrix $R$, eigenvectors are *orthogonal* (at least if the corresponding eigenvalues are different; for simplicity, this assumption is here made). Further, because of the assumed normalization, the eigenvectors are *orthonormal.*

The above orthogonality property is crucial. Because of orthogonality, the eigenvectors are uncorrelated; that is why, the basis vectors corresponding to the maximum variance directions can, at least in principle, be extracted one at a time without disturbing the analysis in other directions.

### 5.2.2 Analysis of the PCA model

Let us study the properties of the variables in the new basis. There are $n$ eigenvectors $\theta_i$ corresponding to eigenvalues $\lambda_i$; from now on, assume that they are ordered in descending order according to their numerical values, so that $\lambda_i \geq \lambda_j$ for $i < j$. This is possible because it was shown that the eigenvalues are real and positive (note that the `eig` function in `Matlab` does *not* implement this ordering automatically). When the eigenvectors and eigenvalues are presented in the matrix form

$$\Theta = \left( \begin{array}{c|c|c} \theta_1 & \cdots & \theta_n \end{array} \right) \qquad \text{and} \qquad \Lambda = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix}, \tag{5.25}$$

where the dimension of $\Theta$ and $\Lambda$ is $n \times n$, the eigenproblem can be expressed compactly as

$$\frac{1}{k} X^T X \cdot \Theta = \Theta \cdot \Lambda. \tag{5.26}$$

It was shown that the vectors constituting $\Theta$ are orthonormal; this means that the whole matrix $\Theta$ also is, so that $\Theta^T = \Theta^{-1}$. Noticing that $X\Theta = Z$ is the sequence of variables as presented in the new latent basis, one can write

$$\frac{1}{k} \cdot Z^T Z = \frac{1}{k} \cdot \Theta^T X^T X \Theta = \Theta^T \Theta \cdot \Lambda = \Lambda. \tag{5.27}$$

What this means is that the new variables are mutually uncorrelated (because their covariance matrix $\Lambda$ is diagonal); what is more, the eigenvalues $\lambda_i$ directly reveal the variances of the new variables. Let us elaborate on this a bit closer.

$$\begin{array}{ll} \mathrm{var}\{z_1\} + \cdots + \mathrm{var}\{z_n\} & \\ \quad = \lambda_1 + \cdots + \lambda_n & \\ \quad = \mathrm{tr}\{\Lambda\} & \text{Definition of } \textit{matrix trace} \\ \quad = \mathrm{tr}\{\frac{1}{k} \cdot \Theta^T X^T \cdot X\Theta\} & \\ \quad = \mathrm{tr}\{\frac{1}{k} \cdot X^T X \cdot \Theta\Theta^T\} & \text{(See below)} \\ \quad = \mathrm{tr}\{\frac{1}{k} \cdot X^T X\} & \text{Orthonormality of } \Theta \\ \quad = \frac{1}{k} x_1^2 + \cdots + \frac{1}{k} x_n^2 & \\ \quad = \mathrm{var}\{x_1\} + \cdots + \mathrm{var}\{x_n\}. & \end{array} \tag{5.28}$$

The *matrix trace* used above returns the sum of the diagonal elements of a square matrix. The change of the multiplication order above is motivated by the trace properties: Note that for all square matrices $A$ and $B$ there must hold

$$\mathrm{tr}\{AB\} = \sum_{i=1}^{n_A} \sum_{j=1}^{n_B} A_{ij} B_{ji} = \sum_{j=1}^{n_B} \sum_{i=1}^{n_A} B_{ji} A_{ij} = \mathrm{tr}\{BA\}. \tag{5.29}$$

The above result (5.28) means that the total variability in $x$ is redistributed in $z$. It was assumed that variance directly carries information — the information content is then redistributed, too. If the dimension is to be reduced, the optimal

approach is to drop out those variables that carry least information: If an $N < n$ dimensional basis is to be used instead of the full $n$ dimensional one, it should be constructed as

$$\theta = \left( \begin{array}{c|c|c} \theta_1 & \cdots & \theta_N \end{array} \right), \tag{5.30}$$

where the vectors $\theta_1$ to $\theta_N$ are the directions of the most variation in the data space. If one tries to reconstruct the original vector $x$ using the reduced basis variables, so that $\hat{x} = \theta z$, the error

$$\tilde{x} = x - \hat{x} = x - \sum_{i=1}^{N} z_i \cdot \theta_i = \sum_{i=N+1}^{n} z_i \cdot \theta_i \tag{5.31}$$

has the variance

$$E\{\tilde{x}^T(k)\tilde{x}(k)\} = \sum_{i=N+1}^{n} \lambda_i. \tag{5.32}$$

This reveals that the the eigenvalues of $R = \frac{1}{k} \cdot X^T X$ give a straightforward method for estimating the significance of PCA basis vectors; the amount of data variance that will be neglected when basis vector $\theta_i$ is dropped is $\lambda_i$.

As an example, study the case of Sec. 4.3 again. The eigenvalues of the data covariance matrix are

$$\left\{ \begin{array}{lcl} \lambda_1 & = & 2 \cdot E\{\xi^2(\kappa)\} + \sigma^2 \\ \lambda_2 & = & \sigma^2, \end{array} \right. \tag{5.33}$$

and the corresponding eigenvectors are

$$\theta_1 = \frac{1}{\sqrt{2}} \cdot \left( \begin{array}{c} 1 \\ 1 \end{array} \right) \quad \text{and} \quad \theta_2 = \frac{1}{\sqrt{2}} \cdot \left( \begin{array}{c} -1 \\ 1 \end{array} \right). \tag{5.34}$$

These basis vectors are shown in Fig. 5.2 (on the right); in this example, the data variance was $E\{\xi^2(k)\} = 1$ and the noise variance was $\sigma^2 = 0.01$. In this case, the ratio between the eigenvalues becomes very large, $\lambda_1/\lambda_2 \approx 200$; the basis vector $\theta_1$ is much more important as compared to $\theta_2$. When a reduced basis with only the vector $\theta_1$ is applied, all deviations from the line $x_2 = x_1$ are assumed to be noise and are neglected in the lower-dimensional basis. The *data collinearity problem is avoided altogether.*

### 5.2.3 Another view of "information"

In the beginning of the chapter it was claimed that it is *variance maximization* that is the means of reaching good data models. But *why* does this seemingly arbitrary assumption really seem to do a good job?

It must be recognized that the main goal in the data compression is to enhance the *signal-to-noise ratio,* so that the amount of misleading disinformation would be minimized as compared to the valuable real information. And it is here that

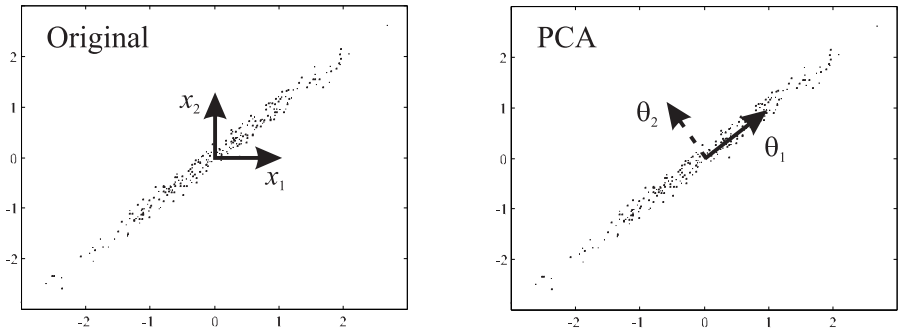



Figure 5.2: Illustration of the "natural" and the PCA bases for the collinear data

the assumptions about "noise ontology" are utilized: The distribution of the noise hopefully differs from that of real information. Typically the underlying basic assumption is that the noise is "more random" than the real signal is; this assumption can have different manifestations:

1. Truly random signals fulfill the assumptions of *central limit theorem,* so that noise distribution is *more Gaussian* than that of real information (this starting point is elaborated on in Chapter 7).

2. If one assumes that noise signals are *uncorrelated* with other signals, the noise is distributed approximately evenly in different directions in the $n$ dimensional space.

The second assumption is utilized in PCA: It is assumed that the same information is visible in various variables, so that the information introduces correlation in the data, whereas noise has no correlations or preferred directions in the data space (see Figs. 5.3 and 5.4). Specially if the data is normalized to unit variance, the *variance pursuit* of PCA changes to *covariance pursuit,* trying to capture the dependencies among variables. The noise variation remaining constant regardless of the direction, the maximum signal-to-noise ratio is reached in the direction where the signal variation is maximum — that is, in the direction of the first principal component. PCR is strongest when MLR is weakest — in large-scale systems with high number of redundant measurements.

Note that PCA gives tools also for further data analysis: For example, if one of the variables varies *alone* (just one variable dominating in the loadings), this variable seemingly does not correlate with other variables — one could consider leaving that variable out from the model altogether (however, see the next section).

### 5.2.4 Selection of basis vectors

How to determine the dimension of the latent basis? For normalized data $\sum_{i=1}^{n} \lambda_i = n$; a crude approximation is to include only those latent vectors $\theta_i$ in the model for which there holds $\lambda_i > 1$ — those directions carry "more

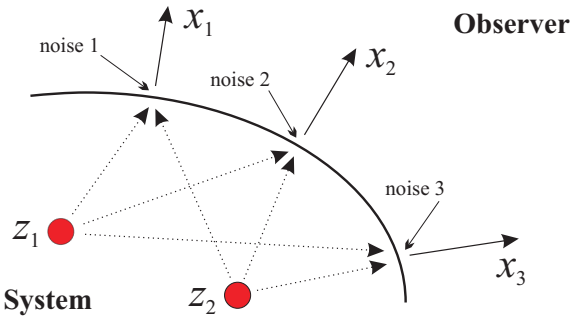



Figure 5.3: Why PCA works: It is assumed that *covariation* reveals some underlying phenomena, whereas noncorrelating variation is measurement noise

that average amount" of the total information (being manifested in the variances). However, the overall behavior of the eigenvalue envelope should be taken into account: That is, plot the eigenvalues in descending order; if there is a significant drop between some of them, this may suggest where to put the model order.

As a rule, it can be argued that the directions of largest eigenvalues are the most important, the dependency relations between variables being concentrated there, whereas the effects of noise are pushed to the later principal components. However, analysis of the components may also reveal some pecularities in the system operation, like outlier data, etc., and the basis selection should not be completely automated. Often the first few eigenvectors represent general dependencies within data, but they may start representing individual disturbances out from the nominal behaviors if these outliers are dominant enough; this all is dependent of the numerical ratios between different phenomena.

If the first principal component dominates excessively, it may be reasonable to check whether the data preprocessing has been successfull: If the data is not mean-centered, it is this mean that dominates in the model rather than the true data variation, specially if the numerical data values are far from origin. The absolute minimum eigenvalue is zero, meaning that the set of measurements is linearly dependent; this can happen also if there are too few measurements, so that $k < n$; note, however, that PCA type data modeling can still be carried out in such case, whereas MLR would collapse. In general, the more there are good-quality samples as compared to the problem dimension, that is, if $k \gg n$, MLR often given good results, whereas the latent basis methods outperform MLR if the number of samples is low (and random variations are visible in data).

If there exist eigenvectors with exactly equal eigenvalues in the covariance matrix, the selection of the eigenvectors is not unique; any linear combination of such eigenvectors also fulfills the eigenvalue equation (5.16). This is specially true for *whitened data,* where the data is preprocessed so that the covariance matrix becomes identity matrix; PCA can find no structure in whitened data (however, see Chapter 7).

It needs to be noted that the PCA results are very dependent of scaling: The principal components can be turned arbitrarily by defining an appropriate or-

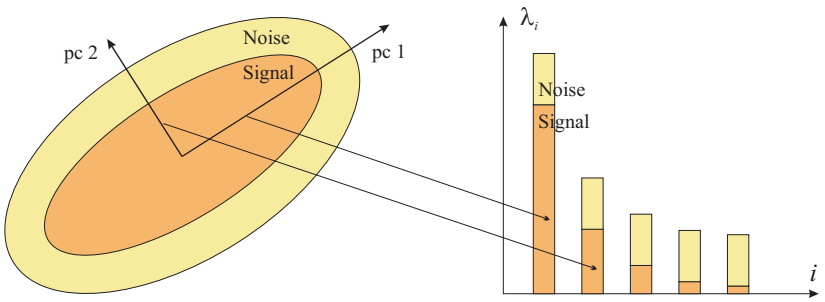



Figure 5.4: Two views of the "directional" information vs. the "undirectional" noise: Five-dimensional data projected onto the first two principal components, on the left, and the corresponding PCA eigenvalues on the right (note that adding a matrix $q \cdot I$, the noise covariance, to the data covariance matrix shifts all eigenvalues up by the amount $q$). Relatively the most of the noise seems to be concentrated in the directions of lowest overall variation

thogonal transformation matrix $D$. Assume that $X' = XD$; if there holds $\Lambda = \frac{1}{k} \cdot \Theta^T X^T X \Theta$, then

$$\Lambda = \frac{1}{k} \cdot \Theta^T D \cdot X'^T X' \cdot D^T \Theta, \tag{5.35}$$

so that the new set of eigenvactors is $D^T\Theta$ — directions being freely adjustable.

figure "InfoNoise"

## 5.3 Practical aspects

Below, some practical remarks concerning the PCA method are presented. For more theoretical discussions, for the validity of the principal components model, etc., the reader should study, for example, [3].

### 5.3.1 Regression based on PCA

The PCA approach has been used a long time for data compression and classification tasks. In all applications the basic idea is redundancy elimination — this is the case also in regression applications.

Summarizing, it turns out that the eigenvector corresponding to the largest eigenvalue explains most of the data covariance. The numeric value of the eigenvalue directly determines how much of the data variation is contained in that eigenvector direction. This gives a very concrete way of evaluating the importance of the PCA basis vectors: One simply neglects those basis vectors that have minor visibility in the data. Using this reduced set of vectors as the internal model subspace basis $\theta_{\text{PCA}}$, principal component regression (PCR) is

directly implemented[3]. Because of the orthogonality of the basis vectors there holds $Z = XF^1 = X\theta_{\text{PCA}}$, and the general modeling procedure (see page 80) reduces into the expression

$$\begin{aligned} F_{\text{PCR}} &= F^1 F^2 \\ &= \theta_{\text{PCA}} \left(\theta_{\text{PCA}}^T X^T X \theta_{\text{PCA}}\right)^{-1} \theta_{\text{PCA}}^T X^T Y \\ &= \theta_{\text{PCA}} \left(k\Lambda_N\right)^{-1} \theta_{\text{PCA}}^T X^T Y. \end{aligned} \tag{5.36}$$

### 5.3.2 Other applications

Principal component analysis has routinely been used for data compression tasks, in all kinds of applications where huge amounts of data are being processed. For example, in *neural networks* the input data is often preprocessed in this way to reach manageable adaptation in the network weights — no matter how "outdated" the statistical methods are claimed to be in that community.

PCA has also been applied in more ambitious tasks, hoping that the compression of data would reveal some underlying hidden phenomena. For example, there exist plenty of applications in *fault diagnosis* and *process monitoring.* A rather new solution to these problems is called *multivariate statistical process control (SPC),* where the traditional approach of observing individual variables at a time is extended to analysis of variation structures of multiple variables (see Fig. 5.5).

### 5.3.3 Analysis tools

The numerical values of the principal component loadings reveal the dependencies (covariances) between different variables, and they also give information about the relevances of different input variables in the regression model. Assuming that $\theta_{i,j}$ is the $j$'th element in the basis vector $i$, the contribution of variable $z_i$ when explaining variance in $x_j$ is $\lambda_i \theta_{i,j}^2$, and the overall relevance of this variable is $\hat{\text{E}}\{x_j^2\} = \sum_{i=1}^{N} \lambda_i \theta_{i,j}^2$, expressing the total amount of variance in $x_j$ that can be reconstructed by the selected latent variables; for normalized $x_j$ this gives a measure for estimating the "value" of that input variable. This kind of analysis is important specially when the model structure is iteratively refined: Non-existent weighting of some of the inputs in all of the latent variables suggests that these inputs could perhaps be excluded from the model altogether.

The PCA model can be analyzed against data in various ways in practice. One can for example calculate the measure for *lack of fit,* the parameter called $Q$. This is simply the sum of error squares when a data sample is fitted against the reduced basis, and then reconstructed. Because $z(\kappa) = \theta^T x(\kappa)$ and $\hat{x}(\kappa) = \theta z(\kappa)$, there holds $\hat{x}(\kappa) = \theta\theta^T x(\kappa)$, so that the reconstruction error becomes

[3]Even if the basis would not be reduced, the orthogonality of the basis vectors already enhances the numeric properties of the regression model: in a non-orthogonal basis, the different coordinates have to "compete" against each other (heuristically speaking), often resulting in excessive numeric values

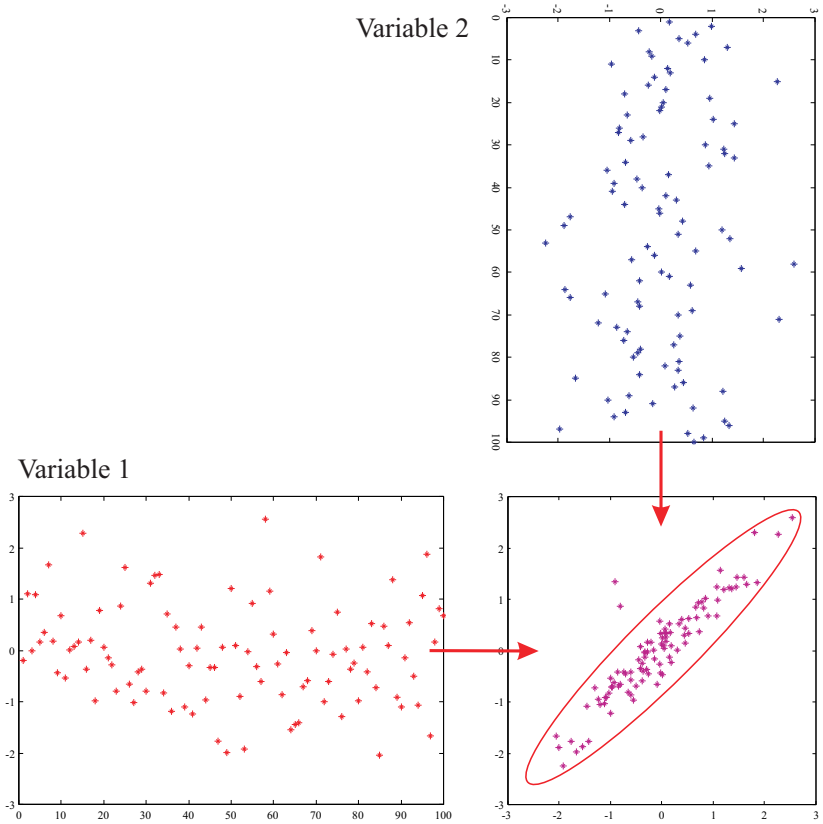



Figure 5.5: Idea of process monitoring using multivariate SPC: It is not always the measurements that are farthest away from the nominal values of the variables that indicate problems in the process

$\tilde{x}(\kappa) = (I_n - \theta\theta^T) \cdot x(\kappa)$. The sum of error squares is then

$$\begin{aligned} Q(\kappa) &= \tilde{x}^T(\kappa)\tilde{x}(\kappa) \\ &= x^T(\kappa) \cdot \left(I_n - \theta\theta^T\right)^T \left(I_n - \theta\theta^T\right) \cdot x(\kappa) \\ &= x^T(\kappa) \cdot \left(I_n - 2\theta\theta^T + \theta\theta^T\theta\theta^T\right) \cdot x(\kappa) \\ &= x^T(\kappa) \cdot \left(I_n - \theta\theta^T\right) \cdot x(\kappa), \end{aligned} \tag{5.37}$$

because due to orthonormality of $\theta$ there holds $\theta \cdot \theta^T\theta \cdot \theta^T = \theta\theta^T$. The $Q$ statistic indicates how well each sample conforms to the PCA model telling how much of the sample remains unexplained.

Another measure, the *sum of normalized squared scores,* known as *Hotellings* $T^2$ *statistic,* reveals how well the data fits the data in another way: It measures the variation in each sample *within* the PCA model. In practice, this is revealed by the scores $z(\kappa)$; the $T^2(\kappa)$ is calculated as a sum of the squared normalized scores. Because the standard deviation of $z_i$ to be normalized is known to be $\sqrt{\lambda_i}$, there holds

$$T^2(\kappa) = z^T(\kappa) \cdot \Lambda_N^{-1} \cdot z(\kappa) = x^T(\kappa) \cdot \theta\Lambda_N^{-1}\theta^T \cdot x(\kappa). \tag{5.38}$$

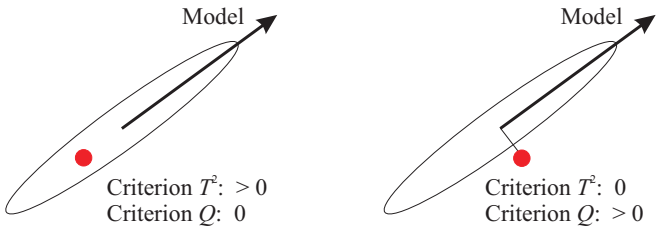



Figure 5.6: The difference between the $T^2$ and $Q$ criteria: The former data point can be represented within the assumed model, whereas the latter one resides in the subspace that is orthogonal to that model

Roughly speaking, the smaller both of these $Q(\kappa)$ and $T^2(\kappa)$ turn out to be, the better the data fits the model. There are essential differences, though: For example, inflating the basis, or letting $N$ grow, typically increases the value of $T^2(\kappa)$, whereas $Q(\kappa)$ decreases (see Fig. 5.6). Closer analyses could be carried out to find exact statistical confidence intervals for these measures; however, these analyses are skipped here.

### 5.3.4 Calculating eigenvectors in practice

There exist very efficient methods for calculating eigenvalues and eigenvectors, available, for example, in `Matlab`. However, let us study such a case where the dimension $n$ is very high, and only few of the eigenvectors are needed.

Assuming that the measurement signals are linearly independent, the (unknown) eigenvectors of the covariance matrix span the $n$ dimensional space, that is, any vector $\xi$ can be expressed as a weighted sum of them:

$$\xi = w_1\theta_1 + w_2\theta_2 + \cdots + w_n\theta_n. \tag{5.39}$$

If this vector is multiplied by the covariance matrix, each of the eigenvectors behaves in a characteristic way:

$$R\xi = \lambda_1 \cdot w_1\theta_1 + \lambda_2 \cdot w_2\theta_2 + \cdots + \lambda_n \cdot w_n\theta_n. \tag{5.40}$$

Further, if this is repeated $k$ times:

$$R^k\xi = \lambda_1^k \cdot w_1\theta_1 + \lambda_2^k \cdot w_2\theta_2 + \cdots + \lambda_n^k \cdot w_n\theta_n. \tag{5.41}$$

If some of the eigenvalues is bigger than the others, say, $\lambda_1$, finally it starts dominating, no matter what was the original vector $\xi$; that is, the normalized result equals the most significant principal component $\theta$:

$$\lim_{k\to\infty} \left\{ \frac{R^k\xi}{\|R^k\xi\|} \right\} = \theta_1. \tag{5.42}$$

Assuming that the eigenvalues are distinct, this *power method* generally converges towards the eigenvector $\theta_1$ corresponding to the highest eigenvalue $\lambda_1$ —

but only if $w_1 \neq 0$. Starting from a random initial vector $\xi$ this typically holds. However, one can explicitly eliminate $\theta_1$ from $\xi$, so that

$$\xi' = \xi - \theta_1^T \xi \cdot \theta_1. \tag{5.43}$$

Now there holds

$$\theta_1^T \xi' = \theta_1^T \xi - \theta_1^T \xi \cdot \theta_1^T \theta_1 = 0, \tag{5.44}$$

meaning that $\theta_1$ does not contribute in $\xi'$, and necessarily $w_i = 0$. If the power method is applied starting from this $\xi'$ as the initial guess, the iteration converges towards the eigenvector direction corresponding to the *next highest* eigenvalue $\lambda_2$. Further, after the second principal component $\theta_2$ is found, the procedure can be continued starting from $\xi''$ were both $\theta_1$ and $\theta_2$ are eliminated, resulting in the third eigenvector, etc. If only the most significant eigenvectors are needed, and if the dimension $n$ is high, the power method offers a useful way to iteratively find them in practice (in still more complex cases, where the matrix $R$ itself would be too large, other methods may be needed; see Sec. 8.3.1).

Of course, numerical errors cumulate, but the elimination of the contribution of the prior eigenvectors (5.43) can be repeated every now and then. The elimination of basis vectors can be accomplished also by applying so called *deflation methods* for manipulating the matrix $R$ explicitly.

## 5.4 New problems

The PCR approach to avoiding the collinearity problem is, however, not a panacea that would always work. To see this, let us study another simple example.

Again, assume that we can observe two variables $x_1$ and $x_2$, so that $x = \begin{pmatrix} x_1 & x_2 \end{pmatrix}^T$. This time, however, these variables are independent; and to simplify the analysis further, assume that no noise is present. This means that the covariance matrix becomes

$$\begin{aligned} \tfrac{1}{k} \cdot X^T X &= \tfrac{1}{k} \cdot \begin{pmatrix} \sum_{\kappa=1}^{k} x_1^2(\kappa) & \sum_{\kappa=1}^{k} x_1(\kappa)x_2(\kappa) \\ \sum_{\kappa=1}^{k} x_1(\kappa)x_2(\kappa) & \sum_{\kappa=1}^{k} x_2^2(\kappa) \end{pmatrix} \\ &\approx \begin{pmatrix} E\{x_1^2(\kappa)\} & 0 \\ 0 & E\{x_2^2(\kappa)\} \end{pmatrix}. \end{aligned} \tag{5.45}$$

The eigenvalues are now trivially $\lambda_1 = E\{x_1^2(\kappa)\}$ and $\lambda_2 = E\{x_2^2(\kappa)\}$, and the eigenvectors are $\theta_1 = \begin{pmatrix} 1 & 0 \end{pmatrix}^T$ and $\theta_2 = \begin{pmatrix} 0 & 1 \end{pmatrix}^T$, respectively. If either of the eigenvalues has much smaller numerical value, one is tempted to drop it out (as was done in the previous PCA example). So, assume that $\theta_2$ is left out. What happens if the underlying relationship between $x$ and $y$ can be expressed as $y = f(x_2)$, so that $x_1$ (or $\theta_1$) is not involved at all? This means that a regression model that uses the reduced PCA basis will *fail completely.*

### 5.4.1 Experiment: "Associative regression"*

It is evident that one has to take the output into account when constructing the latent variables — so, what if we define

$$v(\kappa) = \left( \frac{x(\kappa)}{y(\kappa)} \right), \tag{5.46}$$

and construct a PCA model for this — then the input and output variables should be equally taken into account in the construction of the latent variables. The corresponding covariance matrix becomes

$$\frac{1}{k} \cdot V^T V = \frac{1}{k} \cdot \left( \begin{array}{c|c} X^T X & X^T Y \\ \hline Y^T X & Y^T Y \end{array} \right), \tag{5.47}$$

so that the eigenproblem can be written as

$$\frac{1}{k} \cdot \left( \begin{array}{c|c} X^T X & X^T Y \\ \hline Y^T X & Y^T Y \end{array} \right) \cdot \left( \frac{\theta_i}{\phi_i} \right) = \lambda_i \cdot \left( \frac{\theta_i}{\phi_i} \right). \tag{5.48}$$

Here, the eigenvectors are divided in two parts: First, $\theta_i$ corresponds to the input variables and $\phi_i$ to outputs. The selection of the most important eigenvectors proceeds as in standard PCA, resulting in the set of $N$ selected eigenvectors

$$\left( \frac{\theta}{\phi} \right). \tag{5.49}$$

The eigenvectors now constitute the mapping between the $x$ and $y$ variables, and the matrices $\theta$ and $\phi$ can be used for estimating $y$ in an "associative way". During regression, only the input variables are known; these $x$ variables are fitted against the "input basis" determined by $\theta$, giving the projected $z$ variables[4]:

$$Z = X \cdot \theta^T (\theta^T \theta)^{-1}. \tag{5.50}$$

The output mapping is then determined by the "output basis" $\phi$: Because the coordinates $z$ are known, the estimate is simply

$$\hat{Y} = Z \cdot \phi. \tag{5.51}$$

Combining these gives the regression model

$$F_{\text{ASS}} = \theta^T (\theta^T \theta)^{-1} \phi. \tag{5.52}$$

This should work, at least if the dimension of input $n$ is much higher than that of output $m$. The problem of loosely connected input and output variables still does not vanish: The correlated variables dominating in the eigenvectors can be in the same block, that is, they may both be input variables or they may both be output variables. Modeling their mutual dependency exclusively may

[4]Note that, whereas the eigenvectors of the whole system are orthogonal, the truncated vectors in $\theta$ are not

ruin the value of the regression model altogether. What one needs is a more structured view of the data; the roles of inputs and outputs need to be kept clear during the analysis, and it is the regression models duty to bind them together. This objective is fulfilled when applying the methods that are presented in the following chapter.

It needs to be noted that when concentrating on specific details, something always remains ignored. Now we have seen two methods (MLR and PCA) that offer the best possible solutions to well-defined compact problems. In what follows, MLR will routinely be used when it is justified, and PCA will be used for data compression tasks, understanding their deficiencies; the problems they may possibly ignore are then solved separately. It is expert knowledge to have such a mental "theoretical toolbox" for attacking different problems using appropriate combinations of basic methods depending on the situation at hand.

## Computer exercises

1. Study how the data properties affect the principal component analysis; that is, change the degrees of data freedom and noise level (parameters `dofx` and `sigmax`, respectively):

   ```
   dofx = 5;
   sigmax = 0.5;
   X = dataXY(100,10,NaN,dofx,NaN,sigmax);
   regrPCA(X);
   ```

2. Compare the eigenvectors and eigenvalues of the matrix $R = \frac{1}{k} \cdot X^T X$ when the data preprocessing is done in different ways; that is, create data as

   ```
   DATA = dataClust(3,1,100,50,5);
   ```

   and analyze the results of

   ```
   regrShowClust(X,ones(size(X))); hold on; plot(0,0,'o');
   regrPCA(X)
   ```

   when the following approaches are used:

   ```
   X = DATA;
   X = regrCenter(DATA);
   X = regrScale(DATA);
   X = regrScale(regrCenter(DATA));
   X = regrCenter(regrScale(DATA));
   X = regrWhiten(DATA);
   X = regrWhiten(regrCenter(DATA));
   ```

   Explain the qualitative differences in the eigenvalue distributions. Which of the alternatives is recommended for PCR modeling?

# Lesson 6

# Bridging Input and Output

In the previous chapter it was shown that (one) thing plaguing PCA is its exclusive emphasis on the input variables. The next step to take is then to connect the output variables in the analysis. But, indeed, there are various ways to combine the inputs and outputs. In this chapter, two strategies from the other ends of the scientific community are studied — the first of them, *Partial Least Squares,* seems to be very popular today among chemical engineers. This approach is pragmatic, usually presented in an algorithmic form[1]. The second one, *Canonical Correlation Analysis,* has been extensively studied among statisticians, but it seems to be almost unknown among practicing engineers. However, both of these methods share very similar ideas and structure — even though the properties of the resulting models can be very different.

## 6.1 Partial least squares

The *Partial Least Squares (PLS)*[2] regression method has been used a lot lately, specially for *calibration* tasks in chemometrics [31],[38]. In this section, a *different* approach to PLS is taken as compared to usual practices, only honoring the very basic ideas. The reason for this is to keep the discussion better comprehensible, sticking to the already familiar eigenproblem-oriented framework.

### 6.1.1 Maximizing correlation

The problem with PCA approach is that it concentrates exclusively on the input data $X$, not taking into account the output data $Y$. It is not actually the data variance one wants to capture, it is the *correlation between* $X$ *and* $Y$ that should be maximized.

The derivation of the PLS basis vectors can be carried out as in the PCA case,

[1] PLS is sometimes characterized as being one of those "try and pray" methods; the reason for this is — it can be claimed — that a practicing engineer simply cannot grasp the unpenetrable algorithmic presentation of the PLS ideas. He/she can just use the available toolboxes and hope for the best

[2] Sometimes called also *Projection onto Latent Structure*

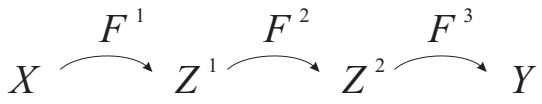



Figure 6.1: The dependency model $y = f(x)$ refined

now concentrating on correlation rather than variance. The procedure becomes slightly more complex than in the PCA case: It is not only the input X block that is restructured, but the internal structure of the output $Y$ block is also searched for. The regression procedure becomes such that the $X$ data is first projected onto a lower dimensional $X$ oriented subspace spanned by the basis vectors $\theta_i$; after that, data is projected onto the $Y$ oriented subspace spanned by the basis vectors $\phi_i$, and only after that, the final projection onto the $Y$ space is carried out.

The objective now is to find the basis vectors $\theta_i$ and $\phi_i$ so that the correlation between the projected data vectors $X\theta_i$ and $Y\phi_i$ is maximized while the lengths of the basis vectors are kept constant. This objective results in the constrained optimization problem (1.27) where

$$\left\{ \begin{array}{rcll} f(\theta_i, \phi_i) & = & \frac{1}{k} \cdot \theta_i^T X^T \cdot Y\phi_i, & \text{when} \\ g_1(\theta_i) & = & 1 - \theta_i^T \theta_i & \text{and} \\ g_2(\phi_i) & = & 1 - \phi_i^T \phi_i. & \end{array} \right. \tag{6.1}$$

There are now two separate constraints, $g_1$ and $g_2$; defining the corresponding Lagrange multipliers $\eta_i$ and $\mu_i$ gives the Hamiltonian

$$\frac{1}{k} \cdot \theta_i^T X^T \cdot Y\phi_i - \eta_i \left(1 - \theta_i^T \theta_i\right) - \mu_i \left(1 - \phi_i^T \phi_i\right), \tag{6.2}$$

and differentiation gives

$$\left\{ \begin{array}{l} \frac{d}{d\theta_i} \left( \frac{1}{k} \cdot \theta_i^T X^T \cdot Y\phi_i - \eta_i(1 - \theta_i^T \theta_i) - \mu_i(1 - \phi_i^T \phi_i) \right) = 0 \\ \frac{d}{d\phi_i} \left( \frac{1}{k} \cdot \theta_i^T X^T \cdot Y\phi_i - \eta_i(1 - \theta_i^T \theta_i) - \mu_i(1 - \phi_i^T \phi_i) \right) = 0, \end{array} \right. \tag{6.3}$$

resulting in a pair of equations

$$\left\{ \begin{array}{l} \frac{1}{k} \cdot X^T Y\phi_i - 2\eta_i \theta_i = 0 \\ \frac{1}{k} \cdot Y^T X\theta_i - 2\mu_i \phi_i = 0. \end{array} \right. \tag{6.4}$$

Solving the first of these for $\theta_i$ and the second for $\phi_i$, the following equations can be written:

$$\left\{ \begin{array}{l} \frac{1}{k^2} \cdot X^T Y Y^T X\theta_i = 4\eta_i \mu_i \cdot \theta_i \\ \frac{1}{k^2} \cdot Y^T X X^T Y\phi_i = 4\eta_i \mu_i \cdot \phi_i. \end{array} \right. \tag{6.5}$$

This means that, again, as in Sec. 5.2, the best basis vectors are given as solutions to eigenvalue problems; the significance of the vectors $\theta_i$ (for the $X$ block) and $\phi_i$ (for the $Y$ block) is revealed by the corresponding eigenvalues $\lambda_i = 4\eta_i \mu_i$.

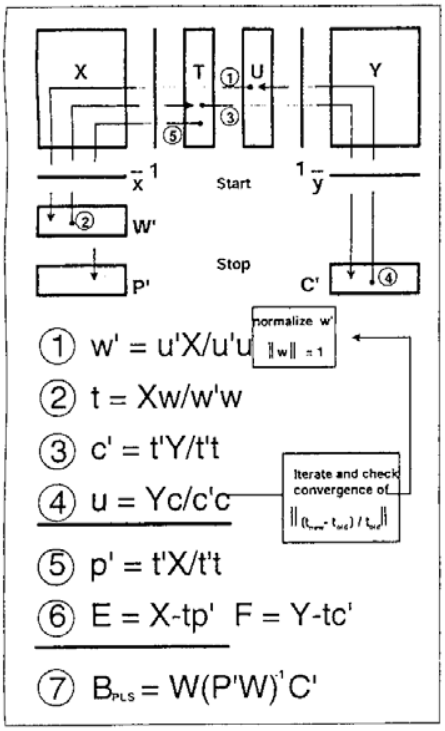



Figure 6.2: What is usually meant by "PLS": The *Algorithm*

Because the matrices $X^TYY^TX$ and $Y^TXX^TY$ are symmetric, the orthogonality properties again apply to their eigenvectors. The expression (5.36) can directly be utilized; the internal basis $\theta_{\text{PLS}}$ consists of a subset of eigenvectors, selection of these basis vectors being again based on the numeric values of the corresponding eigenvalues. In practice, the basis vectors $\phi_i$ are redundant and they need not be explicitly calculated (see Sec. 6.3.3). Because the rank of a product of matrices cannot exceed the ranks of the multiplied matrices, there will be only $\min\{n, m\}$ non-zero eigenvalues; that is why, the PCR approach may give higher dimensional models than PLS (when applying this eigenproblem oriented approach).

It should be recognized that the PLS model is usually constructed in another way (for example, see [31]); this "other way" may sometimes result in better models, but it is extremely uninstructive and implicit, being defined through an iterative algorithm (see Fig. 6.2). It can be shown that the two approaches exactly coincide only what comes to the *most significant* basis vector; other basis vectors can differ. For example, applying the approach based on the eigenvectors, the number of non-zero eigenvalues cannot exceed the number of variables in either input or the output — this means that the latent basis dimension is restricted so that $N \leq m$. Such constraint does not apply to the iterative PLS approach.

Let us study the example that was presented in the previous chapter, now in the PLS framework. The output is scalar; it is assumed that it is linearly dependent of the second input variable, so that $y(\kappa) = f \cdot x_2(\kappa)$, where $f$ is a constant.

The matrix in (5.45) becomes

$$\begin{aligned}
&\tfrac{1}{k^2} \cdot X^T Y Y^T X \\
&= \tfrac{1}{k^2} \cdot \begin{pmatrix} \sum_\kappa x_1(\kappa)y(\kappa) \sum_\kappa x_1(\kappa)y(\kappa) & \sum_\kappa x_1(\kappa)y(\kappa) \sum_\kappa x_2(\kappa)y(\kappa) \\ \sum_\kappa x_1(\kappa)y(\kappa) \sum_\kappa x_2(\kappa)y(\kappa) & \sum_\kappa x_2(\kappa)y(\kappa) \sum_\kappa x_2(\kappa)y(\kappa) \end{pmatrix} \\
&\approx \begin{pmatrix} \mathrm{E}^2\{x_1(\kappa)y(\kappa)\} & \mathrm{E}\{x_1(\kappa)y(\kappa)\} \cdot \mathrm{E}\{x_2(\kappa)y(\kappa)\} \\ \mathrm{E}\{x_1(\kappa)y(\kappa)\} \cdot \mathrm{E}\{x_2(\kappa)y(\kappa)\} & \mathrm{E}^2\{x_2(\kappa)y(\kappa)\} \end{pmatrix} \\
&= \begin{pmatrix} 0 & 0 \\ 0 & f^2 \cdot E^2\{x_2^2(\kappa)\} \end{pmatrix},
\end{aligned}$$

because $x_1$ and $y$ are not assumed to correlate. This result reveals that the maximum eigenvalue is $f^2 \cdot \mathrm{E}^2\{x_2^2(\kappa)\}$ corresponding to the *second* input variable — no matter what is the ratio between the variances of $x_1$ and $x_2$. This means that the basis always includes the vector $(\ 0 \quad 1\ )^T$ — and according to the assumed dependency structure, this is exactly what is needed to construct a working regression model. As a matter of fact, it can be seen that the eigenvalue corresponding to the first input variable is zero, reflecting the fact that $x_1$ has no effect on $y$ whatsoever.

## 6.2 Continuum regression

### 6.2.1 On the correlation structure

Let us study the correlation between input and output from yet another point of view. The correlation structure is captured by the (unnormalized) cross-correlation matrix

$$X^T Y. \tag{6.6}$$

The eigenvalues and eigenvectors are already familiar to us, and it has been shown how useful they are in the analysis of matrix structures. Perhaps one could use the same approaches to analysis of this correlation matrix? However, this matrix is generally not square and the eigenstructure cannot be determined; but the *singular value decomposition,* the geralization of the eigenvalue decomposition exists (see Sec. 1.2.2)

$$X^T Y = \Theta_{XY} \Sigma_{XY} \Phi_{XY}^T. \tag{6.7}$$

Here $\Theta_{XY}$ and $\Phi_{XY}$ are orthogonal matrices, the first being compatible with $X$ and the other being compatible with $Y$; $\Sigma_{XY}$ is a diagonal matrix, but if the input and output dimensions do not match, it is not square. Multiplying (6.7) by its transpose either from left or right, the orthonormality of $\Theta_{XY}$ and $\Phi_{XY}$ (so that $\Theta_{XY}^T = \Theta_{XY}^{-1}$ and $\Phi_{XY}^T = \Phi_{XY}^{-1}$) means that there holds

$$X^T Y Y^T X = \Theta_{XY} \Sigma_{XY} \Sigma_{XY}^T \Theta_{XY}^{-1} \tag{6.8}$$

and

$$Y^T X X^T Y = \Phi_{XY} \Sigma_{XY}^T \Sigma_{XY} \Phi_{XY}^{-1}. \tag{6.9}$$

Because $\Sigma_{XY}\Sigma_{XY}^T$ and $\Sigma_{XY}^T\Sigma_{XY}$ are diagonal square matrices, these two expressions are eigenvalue decompositions (1.5) of the matrices $X^TYY^TX$ and $Y^TXX^TY$, respectively. This means that there is a connection between the singular value decomposition and the above PLS basis vectors: The matrices $\Theta_{XY}$ and $\Phi_{XY}$ consist of the (full sets) of PLS basis vectors $\theta_i$ and $\phi_i$. The diagonal elements of $\Sigma_{XY}$, the singular values, are related to the PLS eigenvalues in such a way that $\sigma_i = k \cdot \sqrt{\lambda_i}$.

What is more, one can see that the SVD of the input data block $X$ alone is similarly closely related to the PCA constructs:

$$X^T = \Theta_X \Sigma_X \Phi_X^T, \tag{6.10}$$

so that

$$X^TX = \Theta_X \Sigma_X \Sigma_X^T \Theta_X^{-1}, \tag{6.11}$$

meaning that, again, the singular value decomposition does the trick, principal components being collected in $\Theta_X$ and singular values being related to the eigenvalues through $\sigma_i = \sqrt{k \cdot \lambda_i}$.

### 6.2.2 Filling the gaps

What if one defines the matrix[3]

$$(X^T)^{\alpha_X}(Y)^{\alpha_Y}, \tag{6.12}$$

so that both of the analysis methods, PCA and PLS, would be received by selecting the parameters $\alpha_X$ and $\alpha_Y$ appropriately (for PCA, select $\alpha_X = 1$ and $\alpha_Y = 0$, and for PLS, select $\alpha_X = 1$ and $\alpha_Y = 1$), and applying SVD? And, further, why not try other values for $\alpha_X$ and $\alpha_Y$ for emphasizing the input and output data in different ways in the model construction? Indeed, there is a continuum between PCA and PLS — and this is not the whole story: Letting the ratio $\alpha_X/\alpha_Y$ go towards zero, we go beyond PLS, towards models where the role of the output is emphasized more and more as compared to the input, finally constructing an singular value decomposition for $Y$ alone (or eigenvalue decomposition for $Y^TY$).

It is only the ratio between $\alpha_X$ and $\alpha_Y$ that is relevant; we can eliminate the other of them, for example, by fixing $\alpha_X = 1$. Representing the problem in the familiar eigenproblem framework, multiplying (6.12) from left by its transpose and compensating the number of samples appropriately one has the eigenproblem formulation for the *Continuum Regression (CR)* basis vectors defined as[4]

$$\frac{1}{k^{1+\alpha}} \cdot X^T \left(YY^T\right)^\alpha X \cdot \theta_i = \lambda_i \cdot \theta_i. \tag{6.13}$$

[3]The powers of non-square matrices being defined as shown in Sec. 1.2.2

[4]These eigenproblems should not be solved directly in this form: The matrix $XXT$ has dimension $k \times k$, even though there are only $n$ non-zero eigenvalues (or singular values)

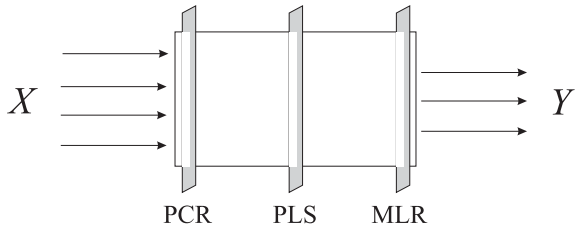



Figure 6.3: Schematic illustration of the relation between regression approaches

Correspondingly, the "dual" formulation becomes

$$\frac{1}{k^{1+1/\alpha}} \cdot Y^T \left(XX^T\right)^{\frac{1}{\alpha}} Y \cdot \phi_i = \lambda_i' \cdot \phi_i. \tag{6.14}$$

When $\alpha$ grows from 0 towards $\infty$, the modeling emphasis is first put exclusively on the input data, and finally exclusively on the output data (see Fig. 6.3); some special values of $\alpha$ do have familiar interpretations:

- If $\alpha = 0$, the PCA model results, only input being emphasized.
- If $\alpha = 1$, the PLS model results, input and output being in balance.
- If $\alpha \to \infty$, an "MLR type" model results, only output being emphasized[5].

Which of the regression approaches, MLR, PCR, or PLS, is the best, cannot be determined beforehand; it depends on the application and available data. All of these methods have only mathematical justification; from the physical point of view, none of them can be said to always outperform the others. It may even be so that the ultimate optimum model lies somewhere on the continuum between PCR, PLS, and MLR (it may also lie somewhere else outside the continuum).

In Figs. 6.4 and 6.5, the CR performance is visualized: There were 30 machine-generated data samples with 20 input and 20 output variables; the number of independent input variables was 10 and the "correct" dimension of the output was 5; relatively high level of noise was added. And, indeed, it seems that when the cross-validation error is plotted as the function of latent variables $N$ and continuum parameter $\alpha$ as a two-dimensional map, interesting behavior is revealed: Starting from $\alpha = 0$, the minimum error is reached for about $N = 12$ whereas the overall optimum is found near MLR with $N = 6$.

### 6.2.3 Further explorations*

It needs to be emphasized again that there are typically no absolutely correct methods for determininf physically optimal latent basis vectors. As in the whole report, the goal here is to show that there is plenty of room for experimenting

[5]Note that MLR is not based on basis vectors; that is why, the correspondence is somewhat artificial (the first basis vector of the CR model explaining the first principal component of the output data, thus explaining maximum amount of the output variance)

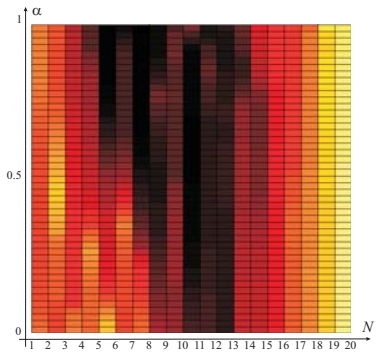



Figure 6.4: Continuum regression performance for different parameter values $N$ and $\alpha$

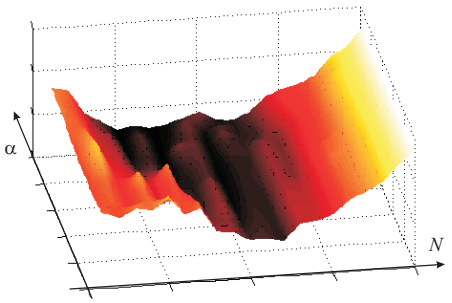



Figure 6.5: Continuum regression performance as a "mountain view"

and research (after all, the history of CR is less than ten years long; by no means one should assume that the final word has been said). For example, a whole class of methods can be defined that share the idea of continuum regression. Let us study a slightly different approach.

MLR can be interpreted as modeling the covariance structure of the estimated output $Y$. The problem that emerges is that the output space usually does not have the same dimension as the input space has; that is why, the output variations need to be somehow presented in the input space to make this approach compatible with the other ones, PCR and PLS. The outputs can be projected into the input space by applying MLR in the "inverse direction", that is, $\hat{X} = Y \cdot (Y^T Y)^{-1} Y^T X$, so that the covariance to be modeled has the form

$$\begin{aligned} \frac{1}{k} \cdot \hat{X}^T \hat{X} &= \frac{1}{k} \cdot X^T Y (Y^T Y)^{-1} \cdot Y^T Y \cdot (Y^T Y)^{-1} Y^T X \\ &= \frac{1}{k} \cdot X^T Y (Y^T Y)^{-1} Y^T X. \end{aligned} \tag{6.15}$$

Actually, this formulation gives a new "latent structure" oriented view of MLR. Assuming that all eigenvectors are utilized, the normal MLR results (of course, this is true for all latent variables based methods if all latent variables are employed), but if a lower dimensional internal model is constructed, the output properties are preserved based on their "visibility" in $Y$. It turns ot that if one defines the latent vectors $\theta_i$ as

$$\frac{1}{k^{1+\alpha_1(1+\alpha_2)}} \cdot X^T \left( Y \left( Y^T Y \right)^{\alpha_2} Y^T \right)^{\alpha_1} X \cdot \theta_i = \lambda_i \cdot \theta_i, \tag{6.16}$$

all of the above regression methods can be simulated by appropriately selecting the parameters $\alpha_1$ and $\alpha_2$:

- PCR is given by $\alpha_1 = 0$, whereas parameter $\alpha_2$ can have any value;
- PLS results if $\alpha_1 = 1$ and $\alpha_2 = 0$; and
- MLR is found if $\alpha_1 = 1$ and $\alpha_2 = -1$.

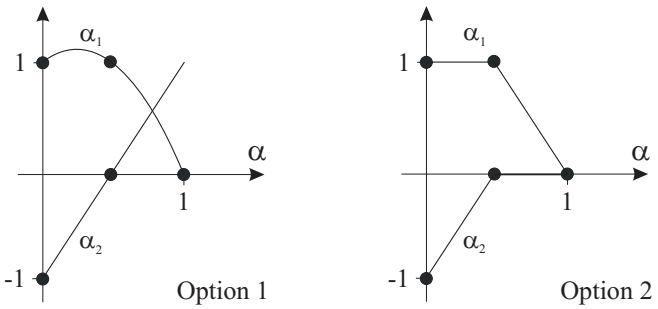



Figure 6.6: Two alternative feasible function forms (see text)

We would like to have a single parameter $\alpha$ spanning the continuum between the approaches, so that $\alpha = 0$ would give MLR, $\alpha = 1/2$ would give PLS, and $\alpha = 1$ would give PCR (note that the range of $\alpha$ is now from 0 to 1). There is an infinity of alternative options — for example, the following definitions fulfill our needs:

1. $\alpha_1 = -2\alpha^2 + \alpha + 1$ and $\alpha_2 = 2\alpha - 1$, or
2. $\alpha_1 = \frac{3}{2} - \alpha - |\alpha - \frac{1}{2}|$ and $\alpha_2 = -\frac{1}{2} + \alpha - |\alpha - \frac{1}{2}|$.

The outlooks of these functions are presented in Fig. 6.6. As an example, selecting the option 1 above, the latent vectors of CR can be calculated as solutions to the following eigenproblem:

$$\frac{1}{k^\beta} \cdot X^T \left( Y \left( Y^T Y \right)^{2\alpha-1} Y^T \right)^{-2\alpha^2+\alpha+1} X \cdot \theta = \lambda \cdot \theta. \tag{6.17}$$

Here, the parameter $\beta$ can be selected as $\beta = -4\alpha^3 + 2\alpha^2 + 2\alpha + 1$ to compensate for the changes in the number of samples. It needs to be noted that the outer matrix that one has to calculate the power function of may be very large (dimension being $k \times k$); however, there are only $m$ eigenvalues different from zero, meaning that (in principle) only $m$ power functions have to be calculated. The matrix power is best to calculate using the singular value decomposition.

The basis $\theta_{\mathrm{CR}}$ is again constructed from the selected eigenvectors; because of the symmetricity of the matrix in (6.17), the basis is orthonormal.

## 6.3 Canonical correlations

Another approach to modeling the dependency structure between the input and the output is offered by *Canonical Correlation Analysis (CCA)* [32].

### 6.3.1 Problem formulation

Again, one would like to find the latent basis vectors $\theta_i$ and $\phi_i$ so that the correlation between the input and output blocks would be maximized. The

criterion to be maximized is again

$$f(\theta_i, \phi_i) = \frac{1}{k} \cdot {\theta_i}^T X^T Y \phi_i, \tag{6.18}$$

but the constraints are modified slightly:

$$\begin{cases} g_1(\theta_i) = \frac{1}{k} \cdot {\theta_i}^T X^T X \theta_i = 1 \\ g_2(\phi_i) = \frac{1}{k} \cdot {\phi_i}^T Y^T Y \phi_i = 1. \end{cases} \tag{6.19}$$

Note the difference as compared to the PLS derivation: It is not the basis vector $\theta_i$ itself that is kept constant size; it is the projected data vector size $Z_i = X\theta_i$ that is regulated, ${\theta_i}^T X^T \cdot X\theta_i$ being kept constant. The same applies also in the output block: The size of ${\phi_i}^T Y^T \cdot Y\phi_i$ is limited.

Again using the Lagrangian technique the following expression is to be maximized:

$$\frac{1}{k} \cdot {\theta_i}^T X^T Y \phi_i + \eta_i \cdot (1 - \frac{1}{k} \cdot {\theta_i}^T X^T X \theta_i) + \mu_i \cdot (1 - \frac{1}{k} \cdot {\phi_i}^T Y^T Y \phi_i). \tag{6.20}$$

This expression can be minimized with respect to both $\theta_i$ and $\phi_i$ separately:

$$\begin{cases} \frac{1}{k} \cdot \frac{d}{d\theta_i} \left( \theta_i^T X^T Y \phi_i - \eta_i (1 - \theta_i^T X^T X \theta_i) - \mu_i (1 - \phi_i^T Y^T Y \phi_i) \right) = 0 \\ \frac{1}{k} \cdot \frac{d}{d\phi_i} \left( \theta_i^T X^T Y \phi_i - \eta_i (1 - \theta_i^T X^T X \theta_i) - \mu_i (1 - \phi_i^T Y^T Y \phi_i) \right) = 0, \end{cases}$$

resulting in a pair of equations

$$\begin{cases} X^T Y \phi_i - 2\eta_i X^T X \theta_i = 0 \\ Y^T X \theta_i - 2\mu_i Y^T Y \phi_i = 0. \end{cases} \tag{6.21}$$

Solving the first of these for $\theta_i$ and the second for $\phi_i$, the following equations can be written (assuming invertibility of the matrices):

$$\begin{cases} X^T Y (Y^T Y)^{-1} Y^T X \theta_i = 4\eta_i \mu_i \cdot X^T X \theta_i \\ Y^T X (X^T X)^{-1} X^T Y \phi_i = 4\eta_i \mu_i \cdot Y^T Y \phi_i, \end{cases} \tag{6.22}$$

or

$$\begin{cases} (X^T X)^{-1} X^T Y (Y^T Y)^{-1} Y^T X \cdot \theta_i = 4\eta_i \mu_i \cdot \theta_i \\ (Y^T Y)^{-1} Y^T X (X^T X)^{-1} X^T Y \cdot \phi_i = 4\eta_i \mu_i \cdot \phi_i. \end{cases} \tag{6.23}$$

This means that, again, the best basis vectors are given as solutions to eigenvalue problems; the significance of the vectors $\theta_i$ (for the $X$ block) and $\phi_i$ (for the $Y$ block) is revealed by the corresponding eigenvalues $\lambda_i = 4\eta_i \mu_i$ (note the equivalences of the corresponding eigenvalues in different blocks). If either $X^T X$ or $Y^T Y$ is not invertible, either one of the *generalized eigenvalue problems* in (6.22) can directly be solved.

It needs to be recognized that data must be explicitly scaled in the CCA case[6]: The property $\frac{1}{k} \cdot \theta^T X^T X \theta = I$ is not automatically guaranteed by the eigen-

[6]This kind of extra scaling is not needed in the above PCA and PLS approaches: By construction, the eigenvectors were assumed to be normalized to unit length

problem formulation. The matrix is diagonal (see the next section), but the diagonal elements are ones only after appropriate scalings:

$$\theta_i \leftarrow \theta_i / \sqrt{\frac{1}{k} \cdot \theta_i^T X^T X \theta_i}. \tag{6.24}$$

### 6.3.2 Analysis of CCA

If the former equation in (6.22) is multiplied from left by $\theta_j^T$, one has

$$\theta_j^T X^T \cdot Y(Y^T Y)^{-1} Y^T \cdot X\theta_i - \lambda_i \cdot \theta_j^T X^T \cdot X\theta_i = 0. \tag{6.25}$$

When rearranged in the above way, one can see that the matrix $Y(Y^T Y)^{-1} Y^T$ is symmetric — meaning that (as in Chapter 5) the eigenproblem can be read in the "inverse" direction, and the following must hold

$$\begin{aligned} &\left(\theta_j^T X^T \cdot Y(Y^T Y)^{-1} Y^T\right) \cdot X\theta_i - \lambda_i \cdot \theta_j^T X^T \cdot X\theta_i \\ &\quad = \lambda_j \cdot \theta_j^T X^T X\theta_i - \lambda_i \cdot \theta_j^T X^T X\theta_i \\ &\quad = (\lambda_j - \lambda_i) \cdot \theta_j^T X^T \cdot X\theta_i \\ &\quad = 0, \end{aligned} \tag{6.26}$$

meaning that $X\theta_i$ and $X\theta_j$ must be orthogonal if $i \neq j$ so that $\theta_i^T X^T X\theta_j = 0$ (remember that for $i = j$ it was assumed that $\theta_i^T X^T X\theta_j = 1$). The same result can be derived for the output block: The projected variables are mutually uncorrelated. Further, if the equations in (6.21) are multiplied from left by $\theta_j^T$ and $\phi_j^T$, respectively, one has

$$\begin{cases} \theta_j^T X^T Y\phi_i = 2\eta_i \cdot \theta_j^T X^T X\theta_i \\ \phi_j^T Y^T X\theta_i = 2\mu_i \cdot \phi_j^T Y^T Y\phi_i. \end{cases} \tag{6.27}$$

Observing the above uncorrelatedness result, one can conclude that also for the cross-correlations between the projected input and output blocks the same structure has emerged: Only for $j = i$ there is correlation, otherwise not; this correlation coefficient is $2\eta_i = 2\mu_i = \sqrt{\lambda_i}$. The above results can be summarized by showing the correlation structure between the latent input and output bases:

$$\begin{aligned} &\left(\begin{array}{c|c} X\Theta & Y\Phi \end{array}\right)^T \left(\begin{array}{c|c} X\Theta & Y\Phi \end{array}\right) \\ &\quad = \left(\begin{array}{ccc|ccc} 1 & & & \sqrt{\lambda_1} & & \\ & \ddots & & & \ddots & \\ & & 1 & & & \sqrt{\lambda_n} \\ \hline \sqrt{\lambda_1} & & & 1 & & \\ & \ddots & & & \ddots & \\ & & \sqrt{\lambda_n} & & & 1 \end{array}\right). \end{aligned} \tag{6.28}$$

For notational simplicity, it is assumed here that $n = m$ (otherwise, the non-diagonal blocks are padded with zeros). The basis vectors $\theta_i$ and $\phi_i$ are called *canonical variates* corresponding to the *canonical correlations* $\sqrt{\lambda_i}$. The very

elegant structure of (6.28) suggests that there must be going on something more important — the dependencies between the input and output blocks are channelled exclusively through these variates. Indeed, it has been recognized that the canonical variates typically reveal some kind of real physical structure underlying the observations, and they have been used for "exploratory data analysis" already in the 1960's. The underlying *real structure* will be concentrated on more in the next chapter.

Note that, because of the non-symmetricity of the eigenproblem matrices, the bases are now generally *not orthogonal!* This is one concrete difference between CCA and PCA/PLS. It can be claimed that whereas PCA and PLS are mathematically better conditioned, CCA is often physically better motivated — the underlying real structures seldom represent orthogonality.

Despite the very similar starting points, PLS and CCA bases are truly very different. For example, if $Y$ is substituted with $X$ in the formulas, it turns out that PLS equals PCA (because the eigenvectors of $X^X$ are the same as those of $\left(X^TX\right)^2$, and the eigenvalues become squared), whereas CCA cannot at all distinguish between directions in the data space — check this by substituting $Y$ with $X$ in (6.23).

### 6.3.3 Regression based on PLS and CCA

In Fig. 6.1, it was explained that regression is a three-step procedure with two latent bases. However, it needs to be noted that this cumulating complexity is only illusion, presented in this form only to reach conceptual comprehensibility. In practice, it is only the first mapping from $X$ to $Z^1$ where the data compression takes place, the step between $Z^1$ to $Z^2$ introducing no additional information loss — thus, the same functionality as in the "stepwise" procedure is reached if one maps the data directly from $Z^1$ to $Y$, discarding the level $Z^2$. With PLS, the structure of the regression model reduces into the same expression as with PCR (see page 80):

$$F_{\text{PLS}} = \theta_{\text{PLS}} \left(\theta_{\text{PLS}}^T X^T X \theta_{\text{PLS}}\right)^{-1} \theta_{\text{PLS}}^T X^T Y. \tag{6.29}$$

With CCR, however, the basis vectors are not orthogonal but the projected data score vectors are — see (6.28). That is why, there is again reduction to the algorithm presented on page 80, but the result looks very different[7]:

$$F_{\text{CCR}} = \theta_{\text{CCA}} \theta_{\text{CCA}}^T X^T Y. \tag{6.30}$$

### 6.3.4 Further ideas*

There are various benefits when all methods are presented in the same eigenproblem-oriented framework — one of the advantages being that one can fluently

[7]Note the similarity between these regression formulas and the expressions (4.19) and (5.36): It is always the correlation between $X$ and $Y$, or $X^TY$, being the basis for the mapping between input and output; how this basic structure is modified by the additional matrix multiplier is only dependent of the method

combine different approaches. For example, it turns out that if one defines

$$R_{\text{CR2}} = \frac{1}{k^\beta} \cdot \left(X^T X\right)^{2\alpha-1} \left(X^T Y \left(Y^T Y\right)^{2\alpha-1} Y^T X\right)^{1-\alpha}, \tag{6.31}$$

the methods from CCR to PLS and PCR are found for $\alpha = 0$, $\alpha = \frac{1}{2}$, and $\alpha = 1$, respectively![8] Parameter $\beta$ can be selected as $\beta = 2\alpha - 1 + (1 - \alpha)(2\alpha - 1) = -2\alpha^2 + 5\alpha - 2$. MLR could also easily be included somewhere along the continuum when using another choice of expressions for the exponents There is one drawback, though — only for the distinct values $\alpha = \frac{1}{2}$ and $\alpha = 1$ the eigenvectors are orthogonal, as compared with the standard continuum regression.

Study yet another idea: Observe the combination of matrices in the CCA solution

$$(X^T X)^{-1} X^T Y (Y^T Y)^{-1} Y^T X. \tag{6.32}$$

Note that this can be divided in two parts: The first part can be interpreted as a mapping $X$ from input to $\hat{Y}$, and the second part maps $\hat{Y}$ to $\hat{X}$:

$$\hat{X} = X \cdot F^1 F^2, \tag{6.33}$$

where

$$\begin{array}{rcl} F^1 & = & (X^T X)^{-1} X^T Y, \quad \text{and} \\ F^2 & = & (Y^T Y)^{-1} Y^T X. \end{array} \tag{6.34}$$

That is, CCA can be interpreted as modeling the behaviors of the mappings when data $X$ is first projected onto output $Y$ and from there back to input. This introduces yet another (CCA oriented) way of constructing the latent basis: One can study what are the statistical properties of this "twice projected" data in the PCA way, that is, the *orthogonal* basis vectors can be defined trough the eigenproblem

$$\hat{X}^T \hat{X} \cdot \theta_i = \lambda_i \cdot \theta_i, \tag{6.35}$$

or

$$X^T Y (Y^T Y)^{-1} Y^T X (X^T X)^{-1} X^T Y (Y^T Y)^{-1} Y^T X \cdot \theta_i = \lambda_i \cdot \theta_i. \tag{6.36}$$

[8] In this case, all the matrices that are involved are low-dimensional and the powers are easily calculated; also note that in the PLS case the square root of the nominal formulation is used for notational simplicity — the eigenvectors, however, remain invariant in both cases

## Computer exercises

1. Study the robustness of the different regression methods trying different values for parameter $k$ (number of samples):

```
k = 20;
[X,Y] = dataXY(k,5,4,3,2,0.001,1.0);

E = regrCrossVal(X,Y,'mlr(X,Y)');
errorMLR = sum(sum(E.*E))/(k*4)
E = regrCrossVal(X,Y,'mlr(X,Y,pca(X,3))');     % Try different
errorPCR = sum(sum(E.*E))/(k*4)
E = regrCrossVal(X,Y,'mlr(X,Y,pls(X,Y,2))');   % Try different
errorPLS = sum(sum(E.*E))/(k*4)
E = regrCrossVal(X,Y,'mlr(X,Y,cca(X,Y,2))');   % Try different
errorCCR = sum(sum(E.*E))/(k*4)
```

2. If installed on your computer, get acquainted with the `Chemometrics Toolbox` for `Matlab`, and `PLS Toolbox`. Try the following demos:

```
plsdemo;
crdemo;
```

# Lesson 7

# Towards the Structure

During the previous discussions, the role of the latent structure has become more and more emphasized. And, indeed, now we are taking yet another leap in that direction: It will be assumed that there really exists some underlying structure behind the observations (see Fig. 7.1)[1]. The observations $x$ are used to determine the internal phenomena taking place within the system; the output variables are calculated only after that. Truly knowing what happens within the system no doubt helps to pinpoint the essential behavioral patterns, thus promising to enhance the accuracy of the regression model. In the earlier chapters the latent structure was just a conceptual tool for compressing the existing data, now it takes a central role in explaining the data.

As has been noticed, the methods presented this far do not offer us intuitively appealing ways to find the real structure: If simple scaling can essentially change the PCA model, for example (see (5.35), it cannot be the physical structure that is being revealed. On the other hand, somehow the idea of continuity between the methods (as utilized in CR) does not promise that a uniquely correct structure would be found. The mathematically motivated structure is not necessarily physically meaningful.

It is an undeniable truth that the underlying primary structure cannot be determined when only observations of the behavior are available. We can only make optimistic guesses — if we trust the benevolence of Nature these guesses are perhaps not all incorrect. However, remember Thomas Aquinas and his theories of "First Cause":

> "... And so we must reach a First Mover which is not moved by anything; and this all men think of as God."

[1]Note that the causal structure is now assumedly different as it was before: If both $X$ and $Y$ are only reflections of some internal system structure, so that no causal dependence is assumed between them, the applications of the final models should also recognize this fact. This means that control applications are somewhat questionable: If $x$ values are altered in order to affect the $y$ values according to the correlations as revealed by the model, it may be that the intended effects are not reached. On the other hand, different kinds of *soft sensor* applications are quite all right: The observed correlations justify us to make assumptions about $y$ variables when only $x$ has been observed (assuming invariant process conditions)

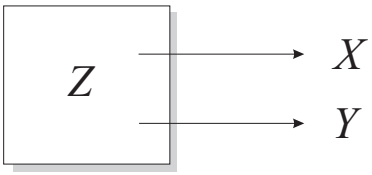



Figure 7.1: Yet another view of signal dependency structure

## 7.1 Factor analysis

An age-old method for feature extraction, or finding the underlying explanations beyond the observations is *Factor Analysis.* It has been applied widely in social sciences, etc. The basic model is familiar:

$$x(\kappa) = \theta z(\kappa), \tag{7.1}$$

or

$$X = Z\theta^T. \tag{7.2}$$

The goal is to find the basis $\theta$ and the scores $Z$ (factors) so that the residual errors $E = X - Z\theta^T$ would be minimized. Nothing strange here — actually the goal sounds identical with the PCA problem formulation. However, now we have an additional *uncorrelatedness* constraint for the residual: The residual errors $E_i$ should be uncorrelated[2]:

$$\mathrm{E}\{e(\kappa)e^T(\kappa)\} = \frac{1}{k} \cdot E^T E = \begin{pmatrix} \mathrm{var}\{e_1(\kappa)\} & & 0 \\ & \ddots & \\ 0 & & \mathrm{var}\{e_n(\kappa)\} \end{pmatrix}. \tag{7.3}$$

All dependencies between data should be explained by the factors alone. Assuming that the residual errors and factors are uncorrelated, the data covariance matrix can be written as

$$\begin{array}{rcl} \frac{1}{k} \cdot X^T X & = & \frac{1}{k} \cdot \left(\theta Z^T Z \theta^T + \theta Z^T E + E^T Z \theta^T + E^T E\right) \\ & & \frac{1}{k} \cdot \theta Z^T Z \theta^T + \frac{1}{k} \cdot E^T E. \end{array} \tag{7.4}$$

From this it follows that, if one defines

$$\begin{array}{rcl} \theta' & = & \theta M \\ Z'^T Z' & = & M^{-1} Z^T Z (M^T)^{-1}, \end{array} \tag{7.5}$$

the same residual errors are received for different factor structure; the new model is also equally valid factor model as the original one was for any invertible matrix $M$. This means that the results are not unique. Factor analysis is more like art than science; there are more or less heuristic basis *rotations* that can be applied to enhance the model. These algorithms will not be studied here.

[2]Note that this uncorrelatedness property is *not* fulfilled by the PCA basis

Note that the uniqueness of the PCA model (at least if the eigenvalues are distinct) is caused by the assumed ordering of the basis vectors according to their relevance in terms of explained variance; in the factor analysis model, this kind of ordering is not assumed and uniqueness is not reached in the same manner. As long as the rotations just operate in the same subspace, the selection of the factors does not affect the accuracy if regression model is to be implemented.

## 7.2 Independent components

Above, factor analysis tried to find the *original sources* by emphasizing uncorrelatedness — but the results were not quite satisfactory, uniqueness of the results remaining lost. Could we define more restrictive objectives that would fix the problems of traditional factor analysis? The key question here, again, is that of *ontological assumptions:* Just as in the case of information vs. noise (chapter 5), now one has to determine how the structure is manifested in the data.

And, indeed, the answer to the question whether structure can be characterized in a reasonable way or not is *yes:* During the last decade, it has turned out that the *independence* of sources is a good starting point. This approach is called *Independent Component Analysis (ICA),* and it has lately been studied specially in the neural networks community. It has been successfully applied for blind source separation, image coding, etc. (see [16], [28]).

### 7.2.1 Why independence?

Intuituively, the original sources are those that are independent of other sources. Finding the underlying structure can be based on this idea: Search for data that is maximally independent. In mathematical terms, two variables $x_1$ and $x_2$ can be said to be independent if there holds[3]

$$\mathrm{E}\{f_1(x_1(\kappa))f_2(x_2(\kappa)\} = \mathrm{E}\{f_1(x_1(\kappa))\} \cdot \mathrm{E}\{f_2(x_2(\kappa)\}. \tag{7.6}$$

According to the above formulation, it can be said that maximizing independence between signals simultaneously minimizes the *mutual information* between them.

In a way, the idea of ICA is to *invert the central limit theorem:* When various independent variables are mixed, the net distribution more or less approximates normal distribution. So, when searching for the original, unmixed signals, one can search for *maximally non-normal* projections of the data distribution!

### 7.2.2 Measures for independence

Probability distributions can uniquely be determined in terms of *moments* or *cumulants.* Gaussian distribution is determined by the first order cumulant

[3]Note that independence is much more than simple uncorrelatedness, where the formula (7.6) holds only when both of the functions are identities, $f_1(x_1) = x_1$ and $f_2(x_2) = x_2$. Because independence is so much more restricting condition than what uncorrelatedness is, one is capable of finding more unique solutions than what is the case with traditional factor analysis

(mean value) and the second order cumulant (variance) alone; for this distribution, all higher order cumulants vanish. This means that the "non-normality" of a distribution can be measured (in some sense) by selecting *any* of the higher order cumulants; the farther this cumulant value is from zero (in positive or negative direction), the more the distribution differs from Gaussian. For example, non-normality in the sense of "non-symmetricity" can be measured using the third-order cumulant *skewness*. In ICA, the standard selection is the fourth-order cumulant called *kurtosis* that measures the "peakedness" of the distribution:

$$\mathrm{kurt}\{x_i(\kappa)\} = E\{x_i^4(\kappa)\} - 3 \cdot E^2\{x_i^2(\kappa)\}. \tag{7.7}$$

For normalized data this becomes

$$\mathrm{kurt}\{x_i(\kappa)\} = E\{x_i^4(\kappa)\} - 3. \tag{7.8}$$

If the data is appropriately normalized, the essence of kurtosis is captured in the fourth power properties of the data; this fact will be utilized later.

After the ICA basis has been determined somehow, regression based on the independent components can be implemented (this method could be called "ICR"). Note that the expressions are somewhat involved because the basis vectors are non-orthogonal.

### 7.2.3 ICA vs. PCA

Figs. 7.3 and 7.2 illustrate the difference between the principal components and the independent components in a two-dimensional case. The data is assumed to have uniform distribution within the diamond-shaped region, and in these figures, ICA and PCA bases for this data are shown, respectively. It really seems that independence means non-Gaussianity: Note that the trapetzoidal marginal distributions in the non-independent PCA case are much more Gaussian than the "flat", negatively kurtotic uniform distributions in the ICA case. The "mixing matrix" (using the ICA terminology) in the case of Fig. 7.3 is

$$\theta = \begin{pmatrix} 1/\sqrt{2} & 1 \\ 1/\sqrt{2} & 0 \end{pmatrix}, \tag{7.9}$$

meaning that $x = \theta z$. Note that, as compared to the Gaussian distribution, uniform distribution is rather "flat"; in this case the kurtosis is maximally negative in the directions of the original sources, other projections having smoother, more Gaussian distributions.

## 7.3 Eigenproblem-oriented ICA algorithms

Normally independent component analysis is carried out in an algorithmic, iterative framework [16]; there are good reasons for this, but in this context we would like to bring ICA into the same eigenproblem-oriented framework as all the other approaches before.

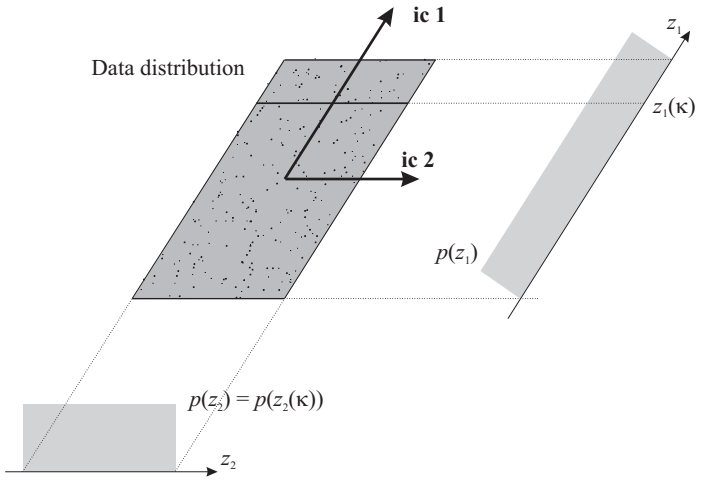



Figure 7.2: The ICA basis vectors, or "independent components". Knowing the value of $z_1(\kappa)$, say, nothing about the value of $z_2(\kappa)$ can be said. The distribution remains intact, or $p(z_2(\kappa)) = p(z_2(\kappa)|z_1(\kappa))$, and the two projected variables really are independent (compare to the PCA case below: information about $z_1(\kappa)$ affects the posteriori probabilities of $z_2(\kappa)$)

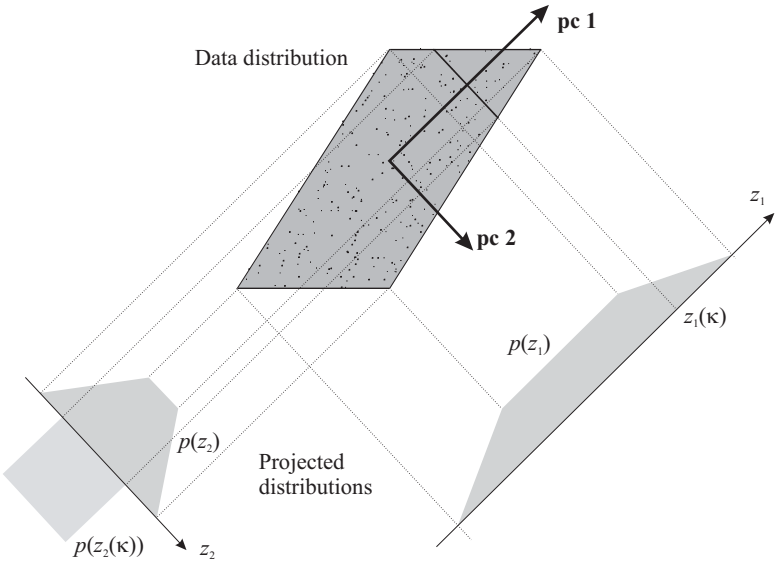



Figure 7.3: The PCA basis vectors, or "principal components": the first of them captures the maximum variance direction, and the other one is perpendicular to it. Variables are not independent

In what follows, kurtosis (or, equally, the fourth moment of data, as shown in (7.8)) as a measure of independence is concentrated on (even though other *contrast functions* can also be defined). The problem with the eigenproblem framework is that it naturally emerges only when the second-order data properties, covariances and correlations, are studied. It is now asked whether the higher-order statistical properties like kurtosis could somehow be captured in the same way. And, indeed, the *tensor methods* for ICA have been found[4]. In principle, the tensors are linear operators just as normal matrices are, and the eigenstructure can be defined also for the four-dimensional tensors; however, the procedures are computationally involved, tensors consisting of $n \cdot n \cdot n \cdot n$ elements, and also the mathematical theory is cumbersome (the "eigenvectors" now being $n \times n$ matrices!). Here the excessive growth of search space (and the sophisticated mathematics) is avoided and some alternative approaches are studied.

### 7.3.1 Data whitening

The key point is to modify the data distribution so that the structural features — as assumedly being revealed by the fourth-order properties — become visible. To reach this, the lower-order properties have to be compensated, because they typically outweight the higher-order properties:

- First-order properties are eliminated by only studying mean-centered data, that is, $\mathrm{E}\{x_i(\kappa)\} = 0$ for all $i$;
- Third-order properties (or "skewness") vanish if one *assumes* that the distributions are symmetric, so that $\mathrm{E}\{x_i(\kappa)x_j(\kappa)x_l(\kappa)\} = 0$ for all $i, j, l$; and
- Second-order properties are eliminated if the data is *whitened.*

The data whitening means that the data is preprocessed so that its covariance matrix becomes an identity matrix. This can be accomplished by

$$x(\kappa) = \left(\sqrt{\mathrm{E}\{\mathbf{x}(\kappa)\mathbf{x}^T(\kappa)\}}\right)^{-1} \cdot \mathbf{x}(\kappa), \tag{7.10}$$

where the square root of a matrix is here defined so that $M = \sqrt{M}^T\sqrt{M}$. After this modification there holds $\mathrm{E}\{x(\kappa)x^T(\kappa)\} = I$. No matter what kind of additional preprocessing is needed, the above elimination of lower-order statistics is assumed in what follows[5].

We are again searching for a basis $\theta$ so that $x(\kappa) = \theta z(\kappa)$, signals $z_i$ now hopefully being independent; and, again, we assume that in the whitened data space

[4]Note that the first-order statistical properties of a distribution are captured by the one-dimensional mean value vector, and the second-order properties are captured by the two-dimensional covariance matrix — similarly, the fourth-order properties can be captured by the four-dimensional *tensor*

[5]If one is capable of finding some structure in the data after this prewhitening, this structure cannot be dependent of the measurement scaling, thus reflecting the *real* structure in a more plausible way — this dependency of the scaling was one of the arguments against the PCA model

the basis is orthogonal (of course, when expressed in the original coordinates, the orthogonality does not hold — see Fig. 7.2).

### 7.3.2 Deformation of the distribution

One way to reduce the fourth-order properties to second-order properties is to explicitly change the distribution. In *Fourth-Order Blind Identification (FOBI)* the data is preprocessed (after first being whitened) so that the samples are either stretched or contracted about the origin. This can be accomplished as

$$x'(\kappa) = f(\|x(\kappa)\|) \cdot x(\kappa), \tag{7.11}$$

where $f$ is some function. For example, selecting $f(\|x\|) = \|x\|$ means that analyzing the variance properties of $x'$ the fourth order properties of the original $x$ are modeled. This can be seen when the new covariance matrix is studied:

$$\begin{aligned} \mathrm{E}\{x'(\kappa)x'^T(\kappa)\} &= \mathrm{E}\{x(\kappa)x^T(\kappa) \cdot \|x(\kappa)\|^2\} \\ &= \mathrm{E}\{\Theta z(\kappa) z^T(\kappa)\Theta^T \cdot z^T(\kappa)\Theta^T\Theta z(\kappa)\} \\ &= \Theta \cdot \mathrm{E}\{z(\kappa)z^T(\kappa) \cdot z^T(\kappa)z(\kappa)\} \cdot \Theta^T. \end{aligned} \tag{7.12}$$

This formulation is justified because one assumes that there exists an orthogonal basis $\Theta$ and independent signals $z_i$. Let us study the matrix $\mathrm{E}\{z(\kappa)z^T(\kappa) \cdot z^T(\kappa)z(\kappa)\}$ closer. The element $i, j$ has the form

$$\begin{aligned} &\mathrm{E}\{z_i(\kappa)z_j(\kappa) \cdot z^T(\kappa)z(\kappa)\} \\ &\quad = \mathrm{E}\{z_i(\kappa)z_j(\kappa) \cdot (z_1^2(\kappa) + \cdots + z_n^2(\kappa))\} \\ &\quad = \mathrm{E}\{z_i(\kappa)z_j(\kappa) \cdot (z_1^2(\kappa)\} + \cdots + \mathrm{E}\{z_i(\kappa)z_j(\kappa)(z_n^2(\kappa)\} \\ &\quad = \begin{cases} \mathrm{E}\{z_i^4(\kappa)\} + \mathrm{E}\{z_i^2(\kappa)\} \cdot \sum_{l\neq i} \mathrm{E}\{z_l^2(\kappa)\} = \mathrm{E}\{z_i^4(\kappa)\} + n - 1, \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } i = j, \text{ and} \\ \mathrm{E}\{z_i^3(\kappa)z_j(\kappa)\} + \mathrm{E}\{z_i(\kappa)z_j^3(\kappa)\} + \\ \qquad \mathrm{E}\{z_i(\kappa)z_j(\kappa)\} \cdot \sum_{l\neq i, l\neq j} \mathrm{E}\{z_l^2(\kappa)\} = 0, \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise.} \end{cases} \end{aligned}$$

The above simplifications are justified because of the assumed independence of the signals $z_i$ — for example, $\mathrm{E}\{z_i^\xi(\kappa)z_j^\zeta(\kappa)\} = \mathrm{E}\{z_i^\xi(\kappa)\} \cdot \mathrm{E}\{z_j^\zeta(\kappa)\}$ for $i \neq j$. Also, because of centering, $\mathrm{E}\{z_i(\kappa)\} = 0$, and because of whitening, $\mathrm{E}\{z_i^2(\kappa)\} = 1$. Additionally, taking into account the assumed orthogonality of $\Theta$ (in the whitened data space), there holds $\Theta^T = \Theta^{-1}$, and

$$\begin{aligned} &\mathrm{E}\{x'(\kappa)x'^T(\kappa)\} \\ &\quad = \Theta \cdot \begin{pmatrix} \mathrm{E}\{z_1^4(\kappa)\} + n - 1 & & 0 \\ & \ddots & \\ 0 & & \mathrm{E}\{z_n^4(\kappa)\} + n - 1 \end{pmatrix} \cdot \Theta^T \\ &\quad = \Theta \cdot \Lambda \cdot \Theta^{-1}. \end{aligned} \tag{7.13}$$

This means that the right hand side can be interpreted as the eigenvalue decomposition of the covariance matrix of the modified data. The diagonal elements in the eigenvalue matrix are directly related to the fourth-order properties of the (assumed) independent components. The cumulant maximization/minimization task (for whitened data) is also transformed into the variance

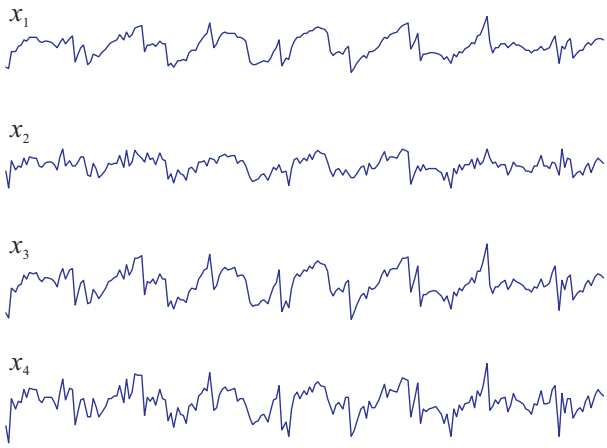



Figure 7.4: The original mixture of signals (200 of the original 1000 samples are shown as time series). Can you see any structure here?

maximization/minimization task for the modified variable[6]. This means that the standard PCA approach can be applied; simultaneously as the covariance structure of $x'$ is analyzed, so is the kurtosis structure of the original variables $x$. However, contrary to the standard PCA, now the principal components carrying the least of the variance may be equally interesting as the first ones are — depending on whether one is searching for the latent basis of maximal or minimal kurtosis. The eigenvalues reveal the kurtoses of the signals $z_i$ so that $\mathrm{kurt}\{z_i(\kappa)\} = \mathrm{E}\{z_i^4(\kappa)\} - 3 = \lambda_i - n - 2$.

As an example, study the four-dimensional data samples as shown in Fig. 7.4. Here, the data sequence is interpreted as constituting a continuous signal; however, note that this signal interpretation is only for visualization purposes. Using the above scheme, the underlying signals can be extracted — with no additional information, just based on the statistical properties of the samples (see Fig. 7.5)!

The exclusively input-oriented approach for determining the latent structure can again be extended: Note that the regression structure $y = F^T x$ remains formally intact if both sides are multiplied by the same factor, so that there holds $yf(x, y) = F^T xf(x, y)$. This means that extensions towards the directions of PLS and CCR, for example, can be proposed where both input and output data are preprocessed prior to determination of the latent structure; it seems that such possibilities have never been explored.

It is important to note that the curve continuity and periodicity, properties that are intuitively used as criteria for "interesting" signals, are not at all utilized by the ICA algorithms — indeed, the samples could be freely rearranged, the continuity and periodicity vanishing, but the analysis results would still remain the same. In fact, the traditional methods like some kind of harmonic analysis could reveal the underlying periodic signal structure, but ICA is specially

[6]Note that if the signals $z_i$ are independent, kurtosis can be maximized/minimized using this algorithm even if the distributions are skewed, that is, $\mathrm{E}\{z_i^3(\kappa)\} \neq 0$

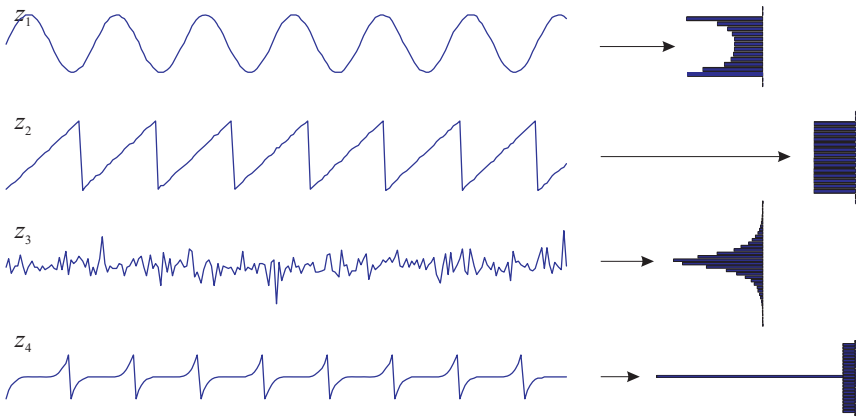



Figure 7.5: The extracted sources and their distributions

powerful when such periodicity or continuity properties cannot be assumed.

However, even though the above approach seems promising, it has to be recognized that in many cases the distribution properties are only visible on the local scale, they cannot be attacked applying global methods like PCA. For example, see Fig. 7.6: there it is shown how the peculiar data distribution is deformed in the data processing. The key observation here is that even after the data deformation (last image), the covariance properties remain identical in orthogonal directions, meaning that none of the directions can be selected by the PCA-based approaches. Typically, algorithmic approaches to ICA are superior, because locally there still exist gradients in the kurtosis-oriented design criterion.

### 7.3.3 Further explorations*

One of the disadvantages of the above algorithm is that it cannot distinguish between independent components that have equal kurtosis[7]. Let us try to find another approach offering more flexibility.

First, study the basic properties of the fourth power of the data point norm:

$$\begin{aligned} \|x\|^4 &= \left(\sqrt{x_1^2 + \cdots + x_n^2}\right)^4 \\ &= \left(x_1^2 + \cdots + x_n^2\right)^2 \\ &= x_1^4 + \cdots + x_n^4 + 2x_1^2x_2^2 + 2x_1^2x_3^2 + \cdots + 2x_{n-1}^2x_n^2. \end{aligned} \tag{7.14}$$

[7]Note that the non-uniqueness problem is the same with PCA if there are equal eigenvalues; however, in this case when we are searching for the real explanations beneath the observations, not only some compression of information, this problem is more acute

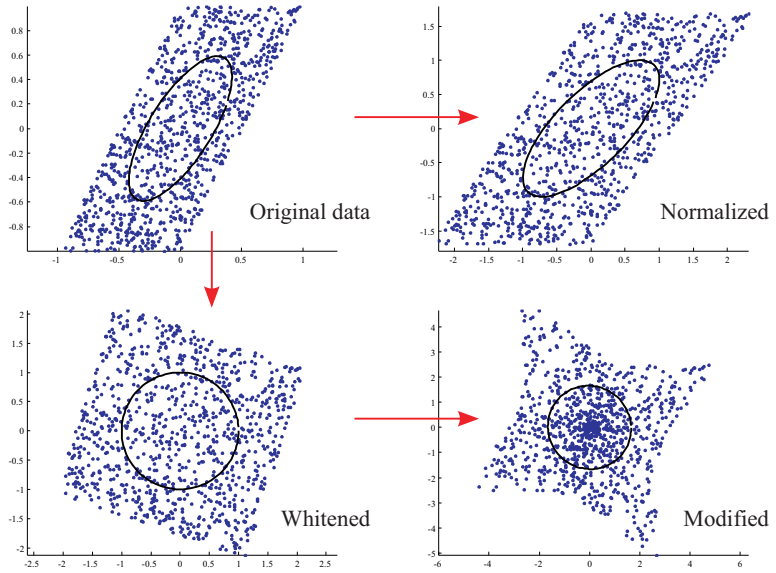



Figure 7.6: Modifications of the data distribution. The covariance structures are shown as ellipses (circles) in the figures

Let us define a modified $(n^2+n)/2$ dimensional data vector as follows:

$$x' = \begin{pmatrix} x_1^2 \\ \vdots \\ x_n^2 \\ \sqrt{2} \cdot x_1 x_2 \\ \vdots \\ \sqrt{2} \cdot x_{n-1} x_n \end{pmatrix}, \tag{7.15}$$

containing all possible second order cross-products between the $x$ elements. Using this kind of modified data vector one can express the fourth moment of the original data vector simply as

$$\|x\|^4 = \|x'\|^2 = x'^T \cdot x'. \tag{7.16}$$

Now in the $x'$ space one can project the point onto an axis $l$ as $x'^T l$, and, further, it is possible to express the fourth moment of this projected data as

$$l^T \cdot x' x'^T \cdot l. \tag{7.17}$$

One should find such an axis $l$ that the average of this quantity over all the modified samples $x'(\kappa)$, where $1 \leq \kappa \leq k$, would be maximized (or minimized). First, construct the expression for the average of projected fourth moment values:

$$\frac{1}{k} \cdot l^T \cdot \sum_{\kappa=1}^{k} x'(\kappa) x^T(\kappa) \cdot l = \frac{1}{k} \cdot l^T \cdot X'^T X' \cdot l, \tag{7.18}$$

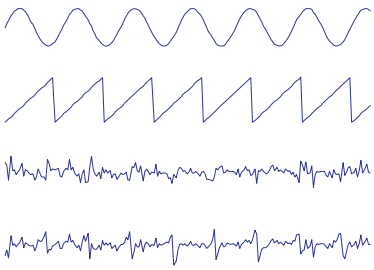


Figure 7.7: The basis vectors corresponding to $l$ of lowest kurtosis. Note that only first two are "correct" signals, these sources being sub-Gaussian

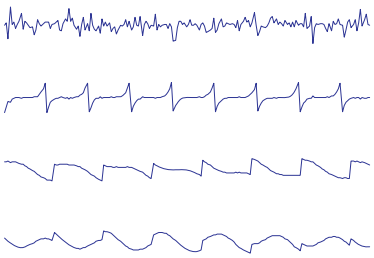


Figure 7.8: The basis vectors corresponding to $l$ of highest kurtosis. Only the first two are correct, these sources being super-Gaussian

where the modified data vectors are written in the matrix form. Letting $\|l\| = 1$, Lagrangian constrained optimization problem results:

$$J(l) = \frac{1}{k} \cdot l^T \cdot X'^T X' \cdot l + \lambda \cdot \left(1 - l^T l\right), \tag{7.19}$$

so that

$$\frac{d\,J(l)}{d\,l} = \frac{2}{k} \cdot X'^T X' \cdot l - 2\lambda \cdot l = 0, \tag{7.20}$$

again resulting in an eigenproblem:

$$\frac{1}{k} \cdot X'^T X' \cdot l = \lambda \cdot l. \tag{7.21}$$

Substituting (7.21) in (7.19) one can see that the eigenvalue equals the cost criterion value, that is, $\lambda$ is the average of the projected fourth moments of the samples. Note that here the least significant eigenvector can be more important than the most significant one, depending whether one is searching for sub-Gaussian or super-Gaussian distribution. This principal component is now presented in the high-dimensional $x'$ space, and to make it useful as a basis vector, one needs to approximate it in the lower-dimensional space of $x$ vectors. For this purpose, remember what is the interpretation of each of the elements in $l$:

$$\begin{cases} l_1 \sim x_1^2 \\ \quad \vdots \\ l_n \sim x_n^2 \\ l_{n+1} \sim \sqrt{2} x_1 x_2 \\ \quad \vdots \\ l_{(n^2+n)/2} \sim \sqrt{2} x_{n-1} x_n, \end{cases} \quad \rightarrow \quad \begin{cases} x_1^2 \sim l_1 \\ \quad \vdots \\ x_n^2 \sim l_n \\ x_1 x_2 = x_2 x_1 \sim \frac{1}{\sqrt{2}} \cdot l_{n+1} \\ \quad \vdots \\ x_{n-1} x_n = x_n x_{n-1} \sim \frac{1}{\sqrt{2}} \cdot l_{(n^2+n)/2}, \end{cases}$$

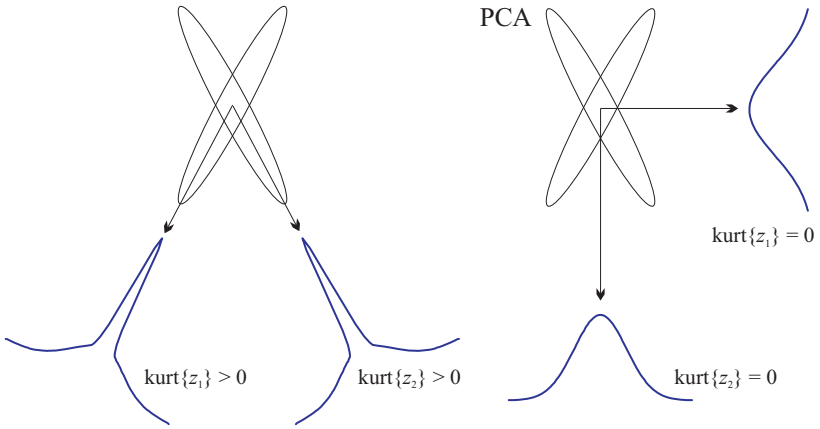



Figure 7.9: Another (more difficult) example of underlying structure: Positive kurtosis in the basis that is suggested by the structure of the distribution (on the left), and zero kurtosis using PCA (two equal projections of normal distributions summed together). In this case, for example, the PCA analysis hides the underlying structure altogether — all samples belonging to different distribution regions are mixed up. However, for this distribution the independence assumption also collapses (see text)

These are not expectation values, but they still tell something about the connections between the variables for some hypothetical data; from the elements of $l$ one can construct an association matrix

$$R = \begin{pmatrix} l_1 & \frac{1}{\sqrt{2}} \cdot l_{n+1} & \cdots & \frac{1}{\sqrt{2}} \cdot l_{2n-1} \\ \frac{1}{\sqrt{2}} \cdot l_{n+1} & l_2 & & \\ \vdots & & \ddots & \\ \frac{1}{\sqrt{2}} \cdot l_{2n-1} & & & l_n \end{pmatrix}. \tag{7.22}$$

Using this matrix, one can determine the $n$ dimensional basis vectors $\theta_i$ that best can span the higher-dimensional space; the answers must be given by the principal components of $R$. Note that the eigenvalues may now be negative, as well as the diagonal elements; this could be explained assuming that data is *complex-valued.* However, because the matrix is symmetric (in this case, actually, Hermitian) the eigenvalues and vectors are real-valued.

## 7.4 Beyond independence

Study the distributions in Fig. 7.9: The intuitively correct basis vectors fulfill the non-Gaussianity goal, the marginal distributions being peaked, or positively kurtotic. However, note that the variables $z_1$ and $z_2$ are in this case *not independent:* knowing, for example, that $z_1$ has high value, one immediately knows that $z_2$ must be near zero, whereas low values of $z_1$ leave much more freedom for $z_2$; in a way, these variables are rather *mutually exclusive* than indepen-

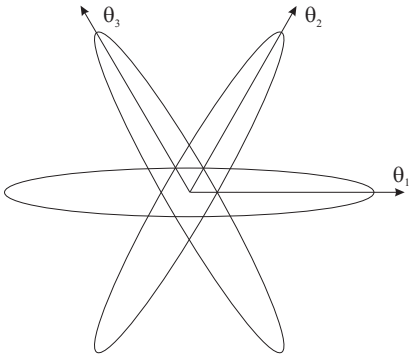



Figure 7.10: Three basis vectors in a two-dimensional space: The basis set is *overcomplete*

dent, either of them telling very much about the other one. The independence objective does not seem to always work when searching for good basis vectors. What kind of alternatives do exist?

### 7.4.1 Sparse coding

It turns out that in those types of distributions that seem to be characteristic to measurement data and that we are specially interested in, meaning mixture models as studied in Sec. 2.4, this mutual exclusiveness is more like a rule rather than exception: If a sample belongs to some specific subdistribution, the other subdistributions do have no role in explaining it. And there are more surprises: It may be so that the correct number of latent structures is higher than what is the dimension of the data space (see Fig. 7.10). Perhaps it is this *exclusiveness* that could be taken as starting point? And, indeed, this approach results in a framework that could be called *Sparse Component Analysis (SCA).*

In sparse coding it is assumed that a sample $x$ is represented in latent basis so that most of the scores are zeros. Sparse coding is a rather general framework: For example, the various submodels constituting a mixture model can be presented within the same sparse structure. But sparse models are *more general* than the mixture models are: Whereas the constructs in mixture models strictly belong to one submodel only, in the sparse framework the *components may be shared,* so that the submodels can have common substructures. This exchange of substructures is the key to the expressional power of sparse models. Unfortunately, this power also suggests that there exist no explicit algortihms for constructing sparse models[8]. Also the *varimax, quartimax,* and *infomax* rotation algorithms resemble sparse coding; these approaches are commonly used within the factor analysis community for maximizing the *score variance,* thus distributing the activity in more specialized factors).

As compared to the modeling methods discussed before, ICA is typically not seen as a compression technique; rather, it carries out data reorganization, so that the $z$ vectors often do have the same dimension as $x$. In the sparse coding

[8]However, various iterative approaches exist; for example, see next chapter

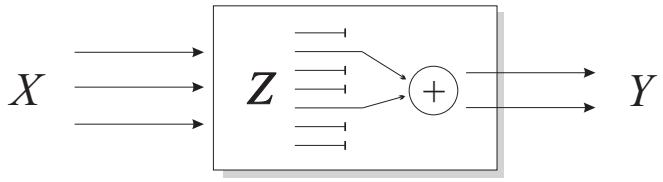



Figure 7.11: The structure of a sparse model

case, the goal is still different: The model may even be *inflated,* so that the number of constructs is higher than the data dimension, hoping that the structural phenomena become better visible when the data is less compactly packed.

One of the implicit general objectives in modeling is *simplicity,* in the spirit of *Occam's razor.* Now this simplicity objective has to be interpreted in another way: Usually, "model of minimum size" means minimum number of substructures; in sparse coding it means minimum number of simultaneously *active* units (see Fig. 7.11).

Regression based on a sparse model is nonlinear; however, the nonlinearity is concentrated on the selection of appropriate latent vectors among the candidates — after they are selected, the model *is* linear. The latent variables can be selected so that together they can explain the data sample as well as possible. The abrupt switching between latent structures means that the model behavior is discontinuous if no additional measures are applied.

When the most specialized constructs are only used, it seems that sparse representations often seem to be "easily interpreted", being sometimes connected to intuitive mental (subsymbolic) constructs. There is some evidence that the human brain organizes at least sensory information in this way: In visual cortex, there are areas, groups of neurons that have specialized in very narrow tasks, like detecting tilted lines in the visual image. The observed image is mentally reconstructed using the low-level visual features — and what is more, it seems that similar principles may be governing the higher level perception, too. There is perhaps room for fruitful cooperation between cognitive science and multivariate statistics.

## Computer exercises

1. Study the robustness of the eigenproblem-formulated ICA by iteratively running the two commands below with different selections of parameter `alpha` (here $\alpha$ denotes the power used in data preprocessing: $x' = \|x\|^\alpha \cdot x$. Note that the default value $\alpha = 1$ resulting in the nominal strictly kurtosis-oriented algorithm is *not* necessarily the best choice — for example, try $\alpha = -1$ for this data):

   ```
   X1 = dataIndep;
   regrICA(X1,alpha);
   ```

   Define data as

   ```
   X2 = dataIndep(1000,...
         'randn(1000,1)',...
         'sign(randn(1000,1)).*(rand(1000,1)<1/3)');
   regrICA(X2);
   ```

   and analyze the independent components. Change the threshold value (the peak probability; value "1/3" above) between 0 and 1, and explain the results.

2. Download the `FastICA Toolbox` for `Matlab` through the Internet address `http://www.cis.hut.fi/projects/ica/fastica/`, and install it. Apply the FastICA algorithm to the above data sets.

# Lesson 8

# Regression vs. Progression

In the earlier chapters, methods were discussed that are based on "traditional" statistical approaches. On the other hand, lately the so called *soft computing* methods have become popular, seemingly shadowing the more traditional modeling methods. In this chapter, the new methodologies are briefly discussed exclusively focusing on *neural networks.* This paradigm consists of a wide variety of different approaches and methods, but there are some common features — the methods are data-based, iterative, and massively parallel. And, what is more, the intended applications are typically extremely complex and nonlinear (see [5], [13], [7], and [36]).

However, neural networks and statistical methods are not competing methodologies for data analysis, and the aim of this chapter is to discuss the connections between these two seemingly very different approaches. It is shown how understanding the (linear) statistical phenomena help in understanding the operation of the more complex algorithms — it is statistical properties between data that there only exist, no matter what is the analysis method, after all.

Neural networks research is a quite diverse field of methods originating from different kinds of intuitions. Three different branches of neural networks research are discussed here separately: The first branch is unsupervised neural clustering methods and regression based on them, and the second branch is perceptron networks and regression. Finally, it is shown how the statistical methods are not only related to artificial neural networks but also to natural neuron structures.

## 8.1 Neural clustering

For practically any statistical data processing method, there exist a neural counterpart. When clustering, for example, is done using the neural networks algorithms, there are typically some benefits: The robustness of the clustering process can be enhanced, and as a bonus, some kind of "topology" can be found between clusteres, making it easier to gain intuition about the data properties. But, on the other hand, there are drawbacks, like the longer execution time of the algorithms.

### 8.1.1 Self-organizing maps

The celebrated *Self-Organizing Map (SOM)* algorithm by Teuvo Kohonen [27] performs nonlinear dimensonality reduction using *competitive learning.* It accomplishes data clustering in such a way that the topology of the input space is somehow preserved, that is, nearby data in the input space are mapped into clusters (now called "nodes") that are near each other also in the low-dimensional grid[1].

The self-organizing map consists of $N$ nodes, each of which is characterized by an $n$ dimensional prototype vector $\bar{x}^c$, where $1 \leq c \leq N$, standing for the cluster centers. The batch SOM algorithm that iteratively organizes the map can be expressed as follows:

1. Choose a set of original node prototypes $\bar{x}^1, \ldots, \bar{x}^N$ arbitrarily.
2. Assign the $k$ samples to the $N$ nodes using the minimum distance rule, that is, sample $x(\kappa)$ belongs to node $c(\kappa)$ such that
$$\bar{c}(\kappa) = \underset{c \in [1,N]}{\operatorname{argmin}} (x(\kappa) - \bar{x}^c)^T (x(\kappa) - \bar{x}^c). \tag{8.1}$$
3. For all pairs of center $\bar{c}(\kappa)$ and node $c$ calculate the "distance measure" (for explanation of the parameters, see below):
$$h_c(\kappa) = \exp\left( -\frac{d^2(c, \bar{c}(\kappa))}{2\sigma^2(\kappa)} \right). \tag{8.2}$$
4. Compute new node prototypes, that is, for all $1 \leq c \leq N$:
$$\bar{x}^c \leftarrow \sum_{\kappa=1}^{k} h_c(\kappa) \cdot x(\kappa) / \sum_{\kappa=1}^{k} h_c(\kappa). \tag{8.3}$$
5. If any of the node prototypes changes, return to step 2, otherwise, stop.

Above, in Step 4, the value of the *neighborhood parameter* $h_c(\kappa)$ determines the behavior of the adaptation process. This parameter determines the net topology, giving large values if the node $c$ and the best matching node $\bar{c}(\kappa)$ are "near" each other in the net, and smaller values otherwise. In (8.2) the parameter $d(c, \bar{c}(\kappa))$ gives the distance between nodes $c$ and $\bar{c}(\kappa)$ in the grid of network neurons, and $\sigma(\kappa)$ determines the "width" of the neighborhood. This parameter can be time-varying, starting from a relatively large value, but getting smaller (the neighborhood "shrinking") as the adaptation continues, making the adaptation more local. When the algorithm has converged, the node prototypes $\bar{x}^c$ contain the cluster centers being arranged within a grid structure.

The selection (8.2) for $h_c(\kappa)$ gives a Gaussian form for the neighborhood effect. As shown in [18], this parameter can be interpreted as probability for a sample to belong to a specific node; that is, the net can be interpreted again as a Gaussian mixture model for the data, and the algorithm tries to adjust the Gaussian centers to best match the data.

[1] Now we are not specially interested in the mapping, or the visual properties of the data, but on the clusters in the input space as generated by the algorithm. The topological ordering is reached as a side-effect

### 8.1.2 "Expectation Maximizing SOM"

The SOM algorithm is related to the K-means clustering, because in both cases the Gaussian distributions are assumed to have identity covariance matrix. As in the case of the EM algorithm, the above algorithm can also be extended to "EMSOM":

1. Choose a set of original node prototypes $\bar{x}^1, \ldots, \bar{x}^N$ arbitrarily, for example, using the SOM algorithm; the cluster covariances are originally identity matrices, that is, $R^c = I$ for all $1 \leq c \leq N$.
2. Assign the $k$ samples to the $N$ nodes using the minimum (balanced) Mahalanobis distance rule:
$$\bar{c}(\kappa) = \underset{c \in [1,N]}{\operatorname{argmin}} \ln(\det\{R^c\}) + (x(\kappa) - \bar{x}^c)^T (R^c)^{-1} (x(\kappa) - \bar{x}^c). \quad (8.4)$$
3. Calculate the neighborhoods for all node/sample pairs according to
$$h_c(\kappa) = \exp\left(-\frac{d^2(c, \bar{c}(\kappa))}{2\sigma^2(\kappa)}\right). \quad (8.5)$$
4. Compute new node prototypes, that is, for all $1 \leq c \leq N$:
$$\bar{x}^c \leftarrow \sum_{\kappa=1}^{k} h_c(\kappa) \cdot x(\kappa) / \sum_{\kappa=1}^{k} h_c(\kappa). \quad (8.6)$$
5. Correspondingly, update covariance estimates
$$R^c \leftarrow \sum_{\kappa=1}^{k} h_c(\kappa) \cdot (x(\kappa) - \bar{x}^c)(x(\kappa) - \bar{x}^c)^T / \sum_{\kappa=1}^{k} h_c(\kappa). \quad (8.7)$$
6. If any node prototypes changes, return to step 2, otherwise, stop.

The algorithm can be stabilized by introducing some gradual forgetting in (8.7). Note that if the number of clusters $N$ is high, this algorithm typically behaves better than the original EM algorithm: The $R^c$ matrices do not become singular as easily.

When doing neural networks modeling, it is typical that very little is assumed about the data. In Chapter 2, it was (optimistically) assumed that the data distributions can be expressed as combinations of Gaussian subdistributions. When doing data modeling with SOM-type algorithms, the philosophy is very different: All assumptions about the underlying distribution structure are discarded, one just tries to capture the data density as exactly as possible. The Gaussian formulas that are used in the self-organization algorithms are used only as *basis functions* for spanning the observed data density, they do not stand for separately distinguishable clusters. That is why the number of nodes in SOM does not usually match the number of real clusters in data (if there exists some) but is considerably higher, each condensation of data being represented possibly by various nodes. This generity makes it possible to model very complex data distributions having no *a priori* information about the nature of the distributions.

### 8.1.3 Radial basis function regression

It turns out that nonlinear regression can be realized directly based on the clustered data model[2]. This kind of regression methods are studied under the name of *radial basis function networks.*

For example, one can assume that if the sample is explained exclusively by some specific subdistribution $c$, the correct (constant) output vector is $\bar{Y}^c$. Limiting the $y$ values to one constant in a distribution sounds to be a harsh limitation — but, as can be seen later, this is not true for a mixture model (see Fig. 8.1). When the distributions are weighted in a reasonable way, according to their probabilities of explaining the measured sample, continuity is achieved.

To apply basis function regression, one first has to find the probabilities for a sample $x$ being represented by a specific subdistribution. The estimated value for the output is (in a maximum likelihood sense) a weighted sum of the candidate outputs; these weighting parameters are the probabilities of the corresponding subdistributions, so that

$$\hat{y} = \sum_{c=1}^{N} \bar{p}_c(x) \cdot \bar{Y}^c, \tag{8.8}$$

where the normalized probabilities are calculated as

$$\bar{p}_c(x) = \frac{p_c(x)}{\sum_{c'=1}^{N} p_{c'}(x)} \tag{8.9}$$

with the individual densities being determined as Gaussian distributions

$$p_c(x) = \frac{1}{\sqrt{(2\pi)^n \det\{R^c\}}} \cdot \mathrm{e}^{-\frac{1}{2}\cdot(x-\bar{x}^c)^T (R^c)^{-1}(x-\bar{x}^c)}. \tag{8.10}$$

If using K-means or SOM for clustering, the covariance is $R^c = \sigma^2 \cdot I$ for all $1 \le c \le N$. If there are $k$ samples $x(\kappa)$ and $y(\kappa)$ available for constructing the model, one has to optimize the values $\bar{Y}^c$ to fit the regression curve with the observations $X$ and $Y$. The above normalized probabilities can be collected (in the familiar way) into the $k \times N$ matrix $\bar{P}(X)$, and the optimal prototype outputs for the clusters can be calculated in the MLR style as

$$\bar{Y} = \left(\bar{P}^T(X)\bar{P}(X)\right)^{-1} \bar{P}^T(X) \cdot Y, \tag{8.11}$$

so that the final nonlinear regression model becomes

$$\begin{aligned} \hat{Y}_{\text{est}} &= \bar{P}(X_{\text{est}}) \cdot \bar{Y} \\ &= \bar{P}(X_{\text{est}}) \cdot \left(\bar{P}^T(X)\bar{P}(X)\right)^{-1} \bar{P}^T(X) Y. \end{aligned} \tag{8.12}$$

[2]Note that we are assuming that the data density is now represented by (ovelapping) Gaussian distributions, no matter whether there really exist separate clusters or not

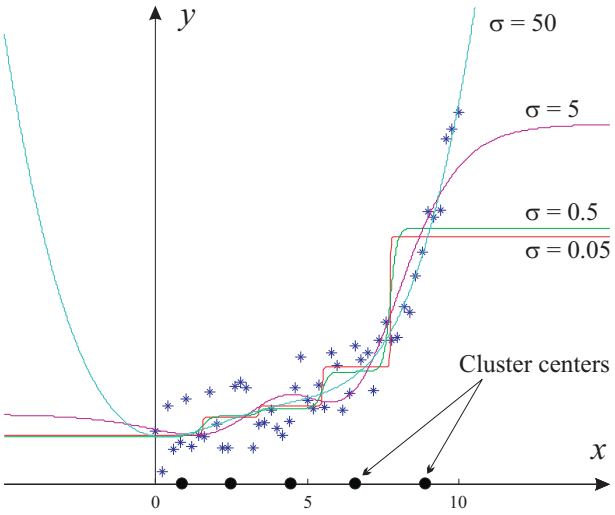



Figure 8.1: The effects of different basis distribution sizes in one dimension. For very narrow distributions, the regression curve becomes stepwise, representing only $N$ distinct values, one for each cluster; when the distribution becomes wider, the curve becomes smoother, but abrupt changes cannot any more be modeled


Figure 8.2: Introducing more and more latent variables ...!

## 8.2 Feedforward networks

This far, we have been mainly interested in linear models, assuming that nonlinearities can be circumvented by appropriate preprocessing of data, clustering or variable selection. However, this is not always enough, specially if the system structure is unknown, and more general methods may sometimes be needed. Nonlinear regression

$$y = g(x) \tag{8.13}$$

can be accomplished in a variety of ways. The key question is: How to parameterize the function $g$?

As compared to linear models, the nonlinear regression problem is much more complex. First, there is the model structure selection problem, and even if the type of the nonlinearity has been determined, the algorithms for finding the best parameters are complicated. Only iterative methods exist. The *multilayer feedforward perceptron networks (MLP's)* are taken here as a prototype of nonlinear regression models. The overall "layered" MLP regression structure is depicted in Fig. 8.2.

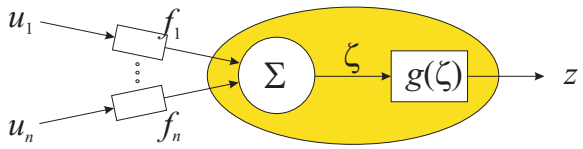



Figure 8.3: A single perceptron: $z = g(\zeta) = g(\sum_{j=1}^{n} f_j u_j)$

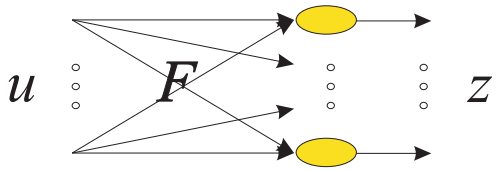



Figure 8.4: A layer of perceptrons: $z = g(F^T u)$, now $g : \mathcal{R}^{n_1} \rightarrow \mathcal{R}^{n_2}$

### 8.2.1 Perceptron networks

Within the neural networks paradigm, it is customary to construct the overall nonlinearity from simple nonlinear units called *perceptrons* (see Fig. 8.3). These perceptrons are independent computing elements trying to mimic the information processing of the biological nerves. Various outputs can be realized when more perceptrons are used in a layer; more sophisticated functions can be implemented by connecting several layers after each other (Figs. 8.4 and 8.5).

As shown in Fig. 8.6, MLP's are general-purpose, flexible, nonlinear models that, given enough hidden neurons and enough data, can approximate virtually any function to any desired degree of accuracy. In other words, MLP's are universal approximators. MLP's can be used when there exists little *a priori* knowledge about the form of the relationship between the input and output variables. MLP's are especially useful because the complexity of the model can be varied easily.

However, if no assumptions are made about the structure of nonlinearity, there are too many degrees of freedom to fix the model using some training data (remember the "Flatland"!). That is why, in practice, the assumption about function *smoothness* is made. This assumption is not well suited for modeling functions with abrupt changes.

### 8.2.2 Back-propagation of errors

The original multilayer perceptron training method was *back-propagation of errors*. The basic backpropagation algorithm is a gradient method, where the weights are adapted in the negative error gradient direction, thus being relatively inefficient. Later, various enhancements have been proposed, but in this context only the original version is presented. The training algorithm can be divided in two parts, forward regression and backward adaptation:

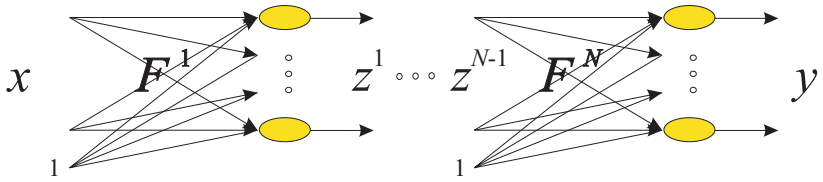



Figure 8.5: The complete perceptron network (note the additional "bias input" in each layer

1. **Neural regression:** For each neuron layer $i$, where $1 \leq i \leq N$, apply the following:

   (a) Augment the input (note that $\hat{z}^0(\kappa) = x(\kappa)$):

   $$u^i(\kappa) = \begin{pmatrix} \hat{z}_{i-1}(\kappa) \\ 1 \end{pmatrix}. \tag{8.14}$$

   (b) Calculate the weighted sum of the inputs:

   $$\zeta^i(\kappa) = (F^i)^T \cdot u^i(\kappa). \tag{8.15}$$

   (c) Apply the ouput function for all $1 \leq j \leq N_i$:

   $$\hat{z}^i_j(\kappa) = g^i\left(\zeta^i_j(\kappa)\right). \tag{8.16}$$

2. **Error back-propagation:** If the weights are to be adapted, assuming that the "correct" output $y(\kappa)$ is available, do the following:

   (a) Calculate the error $e^i(\kappa) = z^i(\kappa) - \hat{z}^i(\kappa)$. Only for the last layer this can be carried out explicitly, because $z^N(\kappa) = y(\kappa)$ is known. All other errors can only be approximated; a heuristic approach is to forget about the nonlinearities, etc., and back-propagate the error from the outer level, assuming that the appropriate "error distribution" between the inner level neurons is determined by the weighting matrix $F^i$:

   $$e^{i-1}(\kappa) = (F^i)^T \cdot e^i(\kappa). \tag{8.17}$$

   (b) Calculate the error gradients for all $1 \leq i \leq N$ and $1 \leq j \leq N_i$ (that is, $i$ is the layer index, and $j$ is the neuron index within a layer). The idea is to calculate the effect of one output at a time on all of the inputs:

   $$\begin{aligned}
   \tfrac{d(e^i_j)^2}{dF^i_j}(\kappa) &= \tfrac{d(z^i_j - \hat{z}^i_j)^2}{dF^i_j}(\kappa) \\
   &= -2e^i_j(\kappa) \cdot \tfrac{d\hat{z}^i_j}{dF^i_j}(\kappa) \\
   &= -2e^i_j(\kappa) \cdot \tfrac{dg^i(\zeta^i_j)}{dF^i_j}(\kappa) \\
   &= -2e^i_j(\kappa) \cdot \tfrac{dg^i}{d\zeta}(\zeta^i_j(\kappa)) \cdot \tfrac{d\zeta^i_j}{dF^i_j}(\kappa) \\
   &= -2e^i_j(\kappa) \cdot \tfrac{dg^i}{d\zeta}(\zeta^i_j(\kappa)) \cdot \tfrac{d((F^i_j)^T u^i)}{dF^i_j}(\kappa) \\
   &= -2e^i_j(\kappa) \cdot \tfrac{dg^i}{d\zeta}(\zeta^i_j(\kappa)) \cdot u^i(\kappa).
   \end{aligned} \tag{8.18}$$

   (c) Update the parameters applying the gradient descent algorithm ($\gamma$ being the step size). The whole column is updated simultaneously:

   $$F^i_j \quad \leftarrow \quad F^i_j - \gamma \cdot \frac{d(e^i_j)^2}{dF^i_j}(\kappa). \tag{8.19}$$

   This process of forward and backward propagation is repeated for different training samples until the network converges.

It needs to be noted that training a nonlinear neural network, or finding the values of the parameters in the matrices $F^i$, is a much more complicated thing than it is in the linear case. First, the results are very much dependent of the initialization of the parameters; and during training, there are the problems caused by local minima and neuron saturation effects, not to mention overfitting problems, etc.

Typically, the nonlinear perceptron activation function $g^i$ is selected as hyperbolic tangent (or "tansig"):

$$g^i_{\text{tansig}}(\zeta) = \frac{2}{1 + \mathrm{e}^{-a\zeta}} - 1, \tag{8.20}$$

so that the derivative becomes

$$\frac{dg^i_{\text{tansig}}}{d\zeta}(\zeta) = \frac{2a\mathrm{e}^{-a\zeta}}{\left(1 + \mathrm{e}^{-a\zeta}\right)^2}, \tag{8.21}$$

where $a$ is some constant (see Figs. 8.8 and 8.9). In some cases, linear neurons may be used[3]. It is reasonable to let at least the outermost layer have linear activation function, otherwise (in the case of $g_{\text{tansig}}$) the network output would be limited between $-1 < \hat{y}_j < 1$:

$$g^i_{\text{linear}}(\zeta) = a\zeta, \tag{8.22}$$

so that

$$\frac{dg^i_{\text{linear}}}{d\zeta}(\zeta) = a. \tag{8.23}$$

The selection of the number of hidden layer neurons is a delicate matter. Note that if there are $N_i$ neurons on the previous level and $N_j$ neurons on the next level, $N_i \cdot N_j$ free parameters are introduced (plus the additional bias term). Too many degrees of freedom may make the model useless (see Fig. 8.7); however, the results are very dependent of the training method, too.

### 8.2.3 Relations to subspace methods

Nonlinear versions of PCA and PLS can also be constructed using the feedforward perceptron network (see Fig. 8.10). The key point is that there is a layer $i$ of relatively low dimension $N_i$, no matter how complicated layers there are before and after this layer. If the dependency between input and output can be compressed, these lower-dimensional hidden layer activations can be interpreted as latent variables.

[3]Note that successive linear layers can be "collapsed"; various linear layers do not expand the expressional power as compared to a single linear layer. If all $g^i$ are linear, the whole network, no matter how complex, can be presented as a single matrix multiplication — and because the cost criterion is identical (minimization of squared error average) the result of training must be the same as in the case of linear regression!

It must be recognized that this kind of results are seldom unique, and the results are often difficult to interpret: It is no more only linear subspaces that are spanned. For example, see Fig. 8.11: It turns out that the functions being modeled are symmetric, so that $y_i = f_i(x) = f_i(\|x\|)$ for $i = 1, 2$, and this can efficiently be utilized in compression — see Fig. 8.12. Only the positive part needs to be modeled, the negative region being taken care of automatically because of the latent variable construction. The network for reaching appropriate behavior was as simple as $N_1 = 2$, $N_2 = 1$ (the latent layer), $N_3 = 2$, and $N_4 = 2$ (the output layer). It is no wonder that the algorithms often fail in this kind of complex tasks (see Exercise 2); typically the training becomes more and more complex as the number of the layers grows.

Again, if all neurons are linear, variables $z$ span the same linear subspace as in normal PCA/PLS; this gives us a new, iterative way to determine the latent basis. In the linear case, again, only one layer of neurons is needed to achieve the required mappings, so that $z = (F^1)^T x$ and $y = (F^2)^T z$. However, it has to be recognized that the same ordering as in standard PCA/PLS cannot generally be reached: None of the hidden layer neurons start uniquely representing the first principal component, etc., and the latent variables are linear combinations of the actual PCA/PLS latent variables that would be derived using the methods presented in earlier chapters.

## 8.3 "Anthropomorphic models"

Of course, all artificial neural networks do have their underlying ideas in neurophysiology, the power of brains having boosted the interest. But typically it is only the network structure that is copied, the functional characteristics being simplified to extreme. However, long before the research on artificial neural networks started, some fundamental phenomena of the neural functions were noticed by Donald O. Hebb [14]:

> Neurons seem to adapt so that the synaptic connections become stronger if the neuronal activation and its input signals correlate.

This general idea of *Hebbian learning* has later become one of the basic paradigms in unsupervised learning where there is no external training.

### 8.3.1 Hebbian algorithms

In unsupervised learning the only thing that the adaptation algorithm can do is to try to find some statistical structure within the input signals. According to the Hebbian principle, it is correlation that should be maximized — this goal sounds distantly familiar, and as it will turn out, the results will also look familiar.

In the simplistic technical implementation, as compared to the earlier discussions, the Hebbian neuron is a *linear perceptron without bias,* that is, the activation (output) of the neuron can be expressed as

$$z(\kappa) = f^T \cdot x(\kappa). \tag{8.24}$$

The parameters in $f$ are interpreted as synaptic weights connecting the neuron to other neurons. According to the Hebbian rule the change in the weights can be expressed as

$$\Delta f = \gamma \cdot z(\kappa)x(\kappa), \tag{8.25}$$

where $z(\kappa)x(\kappa)$ denotes the correlation between the neuronal activation and input, so that

$$f(\kappa+1) = f(\kappa) + \gamma \cdot z(\kappa)x(\kappa). \tag{8.26}$$

Here $\gamma$ is a small adaptation factor. The change in the vector $f$ is determined essentially by the match between input $x(\kappa)$ and the contents of the Hebbian neuron $f$.

One thing plaguing the extremely simplified linear Hebbian neuron is that above learning law (8.26) is not stable but boosts the parameters in $f$ without limits. To enhance the basic Hebbian model, let us prevent $f$ from exploding. A simple solution to this (as proposed by Erkki Oja) is to normalize the length of $f$ to unity after each adaptation step. Assume that, to begin with, $\|f(\kappa)\| = 1$, and this normality is returned after each iteration as follows:

$$\begin{aligned} f(\kappa+1) &= \frac{f(\kappa) + \gamma \cdot z(\kappa)x(\kappa)}{\|f(\kappa) + \gamma \cdot z(\kappa)x(\kappa)\|} \\ &= \frac{f(\kappa) + \gamma \cdot z(\kappa)x(\kappa)}{\sqrt{\left(f(\kappa) + \gamma \cdot z(\kappa)x(\kappa)\right)^T \left(f(\kappa) + \gamma \cdot z(\kappa)x(\kappa)\right)}} \\ &= \frac{f(\kappa) + \gamma \cdot z(\kappa)x(\kappa)}{\sqrt{f^T(\kappa)f(\kappa) - 2\gamma \cdot z(\kappa)x^T(\kappa)f(\kappa) + \mathcal{O}\{\gamma^2\}}} \\ &= \frac{f(\kappa) + \gamma \cdot z(\kappa)x(\kappa)}{\sqrt{1 + 2\gamma \cdot z^2(\kappa) + \mathcal{O}\{\gamma^2\}}}. \end{aligned} \tag{8.27}$$

Assuming that $\gamma$ is small, terms including powers of $\gamma$ higher than two can be ignored. Here one can further approximate the square root by noticing that for small values of $\alpha$ there holds

$$\frac{1}{\sqrt{1+\alpha}} \approx 1 - \frac{1}{2} \cdot \alpha, \tag{8.28}$$

giving (again forgetting terms containing $\gamma^2$)

$$\begin{aligned} f(\kappa+1) &= \left(f(\kappa) + \gamma \cdot z(\kappa)x(\kappa)\right) \cdot \left(1 - \gamma \cdot z^2(\kappa)\right) \\ &\approx f(\kappa) + \gamma \cdot z(\kappa)x(\kappa) - \gamma \cdot z^2(\kappa)f(\kappa). \end{aligned} \tag{8.29}$$

The "stabilized" Hebbian algorithm also becomes

$$f(\kappa+1) = f(\kappa) + \gamma \cdot \left(z(\kappa)x(\kappa) - z^2(\kappa) \cdot f(\kappa)\right). \tag{8.30}$$

In addition to the nominal correlation-motivated factor $z(\kappa)x(\kappa)$, another non-linear term has emerged, preventing the algorithm from growing excessively.

Assuming that the algorithm converges to some fixed $f$ (indeed, it does; for example, see [13]), this parameter change trend $\Delta f(\kappa) = f(\kappa+1) - f(\kappa)$ should

vanish. One can study the properties of this fixed state by taking expectation values on both sides:

$$\begin{aligned} \mathrm{E}\{\Delta f(\kappa)\} &= \gamma \cdot \big(\mathrm{E}\{x(\kappa)z(\kappa)\} - \mathrm{E}\{z^2(\kappa)\} \cdot f\big) \\ &= \gamma \cdot \big(\mathrm{E}\{x(\kappa)x^T(\kappa)\} \cdot f - \mathrm{E}\{z^2(\kappa)\} \cdot f\big)\,. \end{aligned} \tag{8.31}$$

It turns out that in the fixed state there must hold

$$\mathrm{E}\{x(\kappa)x^T(\kappa)\} \cdot f = \mathrm{E}\{z^2(\kappa)\} \cdot f, \tag{8.32}$$

where $\mathrm{E}\{x(\kappa)x^T(\kappa)\}$ is the data covariance matrix. This means that the above formula has the same structure as the PCA problem has, the eigenvector of the data covariance matrix being $f$ and the eigenvalue being $\lambda_1 = \mathrm{E}\{z^2(\kappa)\} \neq 0$. It also turns out that the Hebbian algorithm converges to the principal component (the most significant one; see [13]), so that one can redefine $\theta_1 = f$. Further,

$$\begin{aligned} \lambda_1 &= \mathrm{E}\{z^2(\kappa)\} \\ &= \mathrm{E}\{\theta_1^T x(\kappa)x^T(\kappa)\theta_1\} \\ &= \theta_1^T \cdot \big(\mathrm{E}\{x(\kappa)x^T(\kappa)\} \cdot \theta_1\big) \\ &= \theta_1^T \cdot \lambda_1 \cdot \theta_1 \\ &= \lambda_1 \cdot \|\theta_1\|. \end{aligned} \tag{8.33}$$

When $\lambda_1$ is eliminated on both sides, it turns out that the eigenvector is automatically normalized by this Hebbian algorithm:

$$\|\theta_1\| = 1. \tag{8.34}$$

So, it is no wonder that correlation maximization results in the principal components of the data. What *is* interesting, is that this seems to be happening also in the brain!

### 8.3.2 Generalized Hebbian algorithms

Assume that the contribution of $\theta_1$ is eliminated from the data, so that

$$x'(\kappa) \leftarrow z(\kappa) \cdot \theta_1. \tag{8.35}$$

Note that after this operation the modified input vector is orthogonal to $\theta_1$:

$$\theta_1^T \cdot x'(\kappa) = \theta_1^T \cdot (x(\kappa) - z(\kappa) \cdot \theta_1) = z(\kappa) - z(\kappa) \cdot 1 = 0. \tag{8.36}$$

Applying the Hebbian algorithm using this modified $x'(\kappa)$ as input extracts the most significant principal component that is left after the elimination of the first principal component — that is, now $f$ converges towards $\theta_2$. This procedure can be continued, and the resulting *Generalized Hebbian Algorithm (GHA)* can be used to iteratively extract as many principal components that is needed; if the procedure is formalized as a $N$ layer network, the algorithm to be applied for all $1 \leq i \leq N$ (note that $x^1(\kappa) = x(\kappa)$) becomes

$$\begin{aligned} x^i(\kappa) &= x^{i-1}(\kappa) - z^{i-1}(\kappa) \cdot f^{i-1}(\kappa) \qquad \text{for layers } i > 1 \\ z^i(\kappa) &= (x^i)^T(\kappa) \cdot f^i(\kappa) \\ f^i(\kappa+1) &= f^i(\kappa) + \gamma \cdot z^i(\kappa) \cdot \big(x^i(\kappa) - z^i(\kappa) \cdot f^i(\kappa)\big)\,. \end{aligned} \tag{8.37}$$

After the iteration has converged, the principal components are

$$\left( \begin{array}{c|c|c} \theta_1 & \cdots & \theta_N \end{array} \right) = \left( \begin{array}{c|c|c} f^1 & \cdots & f^N \end{array} \right). \tag{8.38}$$

Note that the algorithm (8.37) does probably no more have any connection to operations taking place in the brain; it is just an extension of the basic Hebbian idea for easily extracting the principal component structure from the data. The structures of the Hebbian algorithms are so simple that they can be easily implemented; there is no explicit reference to their neural background, and this is an example of how the new paradigms can give important contribution to other branches of research.

Note that when using the Hebbian algorithms, the covariance matrix never needs to be explicitly constructed — that is why, the Hebbian approach may be useful in specially high-dimensional data analysis tasks.

### 8.3.3 Further extensions

The generalized Hebbian algorithm can still be extended. For example, take the *anti-Hebbian learning,* where, in addition to maximizing the correlation with the inputs, the correlations with *other* outputs is minimized. The goal is to make the neurons as independent from each other as possible; as we have seen, this kind of independence often reveals underlying structure in the data. The explicit decorrelation between outputs results in sparse coding as shown in [8]. However, the correlation maximization/minimization structure is recursive, and the training algorithms are rather inefficient.

Another extension towards multiple mixture models is the *Generalized Generalized Hebbian Algorithm (GGHA)* [24]. The idea is to explicitly assume sparsity in the data; that is, there are various trains of candidate principal components, and only one of these candidate sequences is selected at a time (using the "best match" principle). When the selected components are eliminated from data, as in GHA, the rest is explained by the remaining components. This algorithm has been applied to a number of high-dimensional feature extraction problems.

## 8.4 Cybernetic neurons*

Cybernetics is a branch of complex systems research, where it is assumed that the observed complex functionalities can be explained in terms of interactions and feedbacks among the underlying local "agents". Specially, in *neocybernetics* the approaches are made very concrete: In the spirit of multivariate models, it is assumed that understanding of high dimensionality and dynamic structures can help in explaining the emergent functionalities. Furthermore, there are some very stringent assumptions: First, it is assumed that there is dynamic balance on all levels in a complex hierarchical system; second, model structures are kept as simple as possible — one could speak of *linearity pursuit.* Despite the constraints, it seems that non-trivial systems can be modeled in this way (see [**?**]).

These guidelines can be exploited on different levels of modeling the neuronal system, and analysis of a Hebbian neuron grid is carried out here. First, one can study the synaptic level: The stabilization of the synaptic weight can be accomplished not only applying nonlinearity, as is done when following Oja's rule, but also applying linear feedback. So, assume that instead of (14) one defines

$$\Delta f = \gamma \cdot z(\kappa)x(\kappa) - \alpha f \tag{8.39}$$

for some scalar $\alpha$. It turns out that, assuming stationarity of the input, the synapse finds a stable value that is proportional to the correlation between the input and the neuronal activity. In the matrix form, the steady state of all synaptic weights can be expressed ($\beta$ being some scalar) using an (unnormalized) correlation matrix

$$W = \beta \mathrm{E}\{zx^T\}. \tag{8.40}$$

To reach some added value, the neocybernetic intuitions can be applied also on the next level, or to the analysis of the whole neuron grid. Assume that the behavior of the grid of individual Hebbian neurons is orchestrated again by linear feedback, so that some of the synapses are between neurons — this means that one has a dynamic structure of the form

$$\frac{dz}{dt} = -Az + Bx. \tag{8.41}$$

Here, the matrices $A$ and $B$ contain the synaptic weights of $W$ as divided according to their roles: synapses between inputs and neurons are collected in $B$, whereas the inter-neuronal connections are represented by the matrix $A$. In front of $A$ there is the "$-$" sign to explicitly emphasize the negative feedback nature of these "anti-Hebbian" connections. The adaptation of the neuronal activities is presented here in the continuous-time form, and it is assumed that dynamics of this internal loop is much faster than the dynamics of the input $x$, so that one can solve for the steady state

$$\bar{z} = F^T x = A^{-1} B\, x, \tag{8.42}$$

where now

$$A = \beta \mathrm{E}\{\bar{z}\bar{z}^T\}, \qquad \text{and} \qquad B = \beta \mathrm{E}\{\bar{z}x^T\}. \tag{8.43}$$

Note that because $A$ represents the covariance matrix of the state vector, all eigenvalues being non-negative, dynamics determined by the matrix $-A$ always remains stable. The covariance matrix estimates can be adapted, for example, using a continuous-time algorithm for some time constant $\tau \gg 1/\beta$ as

$$\frac{d\,\hat{\mathrm{E}}\{\bar{z}x^T\}}{dt} = -\frac{1}{\tau}\hat{\mathrm{E}}\{\bar{z}x^T\} + \frac{1}{\tau}\bar{z}x^T. \tag{8.44}$$

Because the neocybernetic studies concentrate on balances on all levels, it is of interest to see what are the stationary properties of $\bar{x}$. One has

$$\begin{aligned} \mathrm{E}\{\bar{z}\bar{z}^T\} &= A^{-1} B\, \mathrm{E}\{xx^T\}\, B^T A^{-1} \\ &= \mathrm{E}\{\bar{z}\bar{z}^T\}^{-1} \mathrm{E}\{\bar{z}x^T\}\, \mathrm{E}\{xx^T\}\, \mathrm{E}\{\bar{z}x^T\}^T \mathrm{E}\{\bar{z}\bar{z}^T\}^{-1}, \end{aligned} \tag{8.45}$$

or, when simplified

$$\mathrm{E}\{\bar{z}\bar{z}^T\}^3 = \mathrm{E}\{\bar{z}x^T\}\mathrm{E}\{xx^T\}\mathrm{E}\{\bar{z}x^T\}^T. \tag{8.46}$$

Taking the linearity of the model into account, there holds

$$\left(F\mathrm{E}\{xx^T\}F^T\right)^3 = F\left(\mathrm{E}\{xx^T\}\right)^3 F^T. \tag{8.47}$$

The solution for the mapping matrix is non-trivial, if the dimension of the input $x$ is higher than that of the state $\bar{z}$, that is, $N < n$. It turns out that the columns of $F$ span the *principal subspace* of the input data, that is, they are linear combinations of the $N$ most significant principal components. It also turns out that the neocybernetic principles are enough to implement self-regulation and self-organization (in the sense of PCA), even though the local synapses only are capable of reacting to their immediate environment, knowing nothing about the global situation. From the technical point of view, it is nice that explicit covariance matrices in the assumedly high-dimensional space of $x$ vectors is not needed — one essentially operates in the low-dimensional space of $z$ vectors. However, the process is necessarily highly iterative as the final balances $\bar{z}$ are not known before adaptation.

Looking at the structure of the mapping matrix $F$, it is evident that one can implement reconstruction of the input in the least-squares sense in a straightforward way:

$$\hat{x} = \mathrm{E}\{\bar{z}x^T\}^T\mathrm{E}\{\bar{z}\bar{z}^T\}^{-1}\,\bar{z} = F\,\bar{z}. \tag{8.48}$$

This means that the internal state $\bar{z}$ can be interpreted as some kind of "mirror image" of the environment as represented by $x$.

Further, one can also implement normal principal component regression exploiting the principal subspace (see Fig. 8.13):

$$\hat{y} = \mathrm{E}\{\bar{z}y^T\}^T\mathrm{E}\{\bar{z}\bar{z}^T\}^{-1}\,\bar{z} = \mathrm{E}\{\bar{z}y^T\}^T\mathrm{E}\{\bar{z}\bar{z}^T\}^{-2}\mathrm{E}\{\bar{z}x^T\}\,x. \tag{8.49}$$

It seems that the dynamic systems understanding can give new intuitions for studying regression models. Also, as it turns out in what follows, understanding of the static regression models can help to better exploit the dynamic models.

## Computer exercises

1. Construct data as follows:

```
X = 20*rand(100,1)-10;
Y = [cos(X/2),abs(X)/5-1];
Xtest = [-15:0.1:15]';
```

   Study the behavior of the radial basis regression for different values of $N$ (number of clusters) and $\sigma$ (width of the distributions):

```
[clusters] = regrKM(X,N);
[rbfmodel] = regrRBFN(X,Y,clusters,sigma);
Ytest = regrRBFR(Xtest,rbfmodel);
```

2. Assuming that you have the `Neural Network Toolbox` for `Matlab` available (version 3.0.1, for example) study the robustness of training the multilayer feed-forward perceptron network. Using the same data as above, construct the model as

```
structure = [2 1 2 size(Y,2)];
outfunc = {'tansig', 'tansig', 'tansig', 'purelin'};
net = newff([min(X);max(X)]',structure,outfunc);
net = train(net,X',Y');
```

   In principle, these commands reproduce the example presented in Figs. 8.12 and 8.11. What can you say about reliability? For model simulation use the commands

```
Ytest = sim(net,Xtest');
net.outputConnect = [0 1 0 0]; % Second layer output
Ztest = sim(net,Xtest');
```

   Try also different network structures (that is, change the `structure` and `outfunc` parameters). For example, using only one hidden layer, what is the minimum number of hidden layer units that can accomplish the mapping?

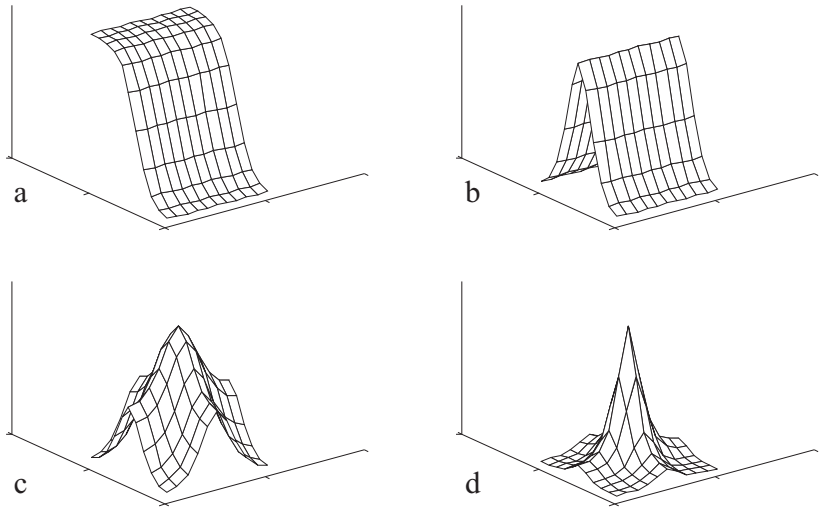



Figure 8.6: Visualization of the generality of feedforward perceptron networks as approximators of smooth functions. In this example, only two input signals is assumed ($x_1$ and $x_2$), and in the figures, the outputs of neurons are plotted as functions of these inputs in a two-dimensional $(x_1, x_2)$ plane. First, see **a**: this kind of output function is characteristic to the nonlinear neuron; adjusting the weights of the inputs and the bias, the location, orientation, and depth of the "transition barrier" can be freely adjusted (note that $z = g(w_1x_1 + w_2x_2 + b)$). Similarly, if yet another neuron is connected to the same input using the same ratio between the input weights, the new transition barrier is parallel to the previous one, yet shifted. Figure **b** results if the outputs of these two neurons are added together — the height of the bump can be freely adjusted by changing the weights. Using another set of two neurons, another (not parallel) bump can be constructed, and if the outputs of these four neurons are added together, the result looks something like the surface in **c**. The peak can be emphasized (see **d**) if this signal is connected to a second-layer neuron. It turns out that using four first-layer neurons, a peak can be created anywhere in the plane; if there are enough neurons, a large number of such peaks can be constructed. These peaks can be applied as basis functions (compare to radial basis function networks), and any continuous function can be approximated to arbitrary accuracy. To summarize, a two-layer network with enough hidden layer neurons can approximate any function

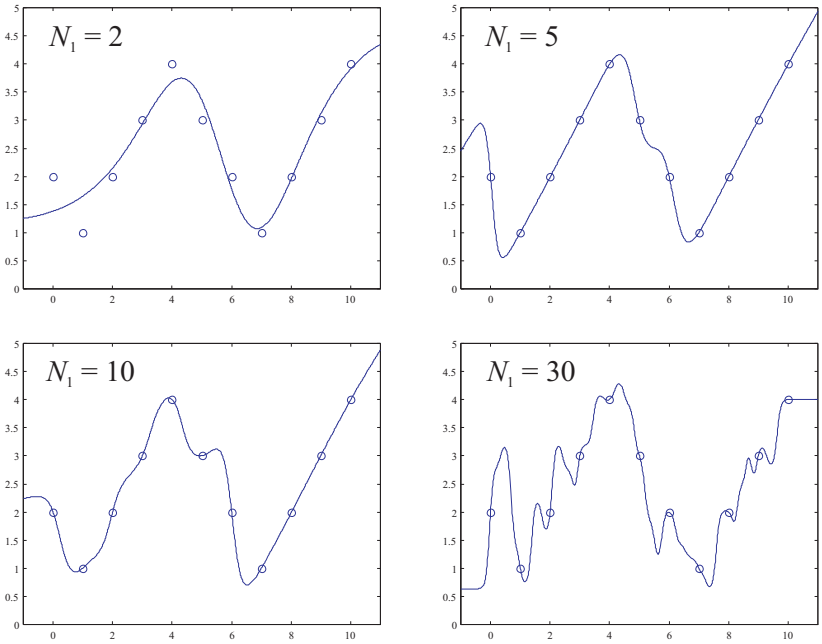



Figure 8.7: Number of neurons — matter of expertise. A two-level feed-forward perceptron network (hyperbolic tangent / pure linearity) with different numbers $N_1$ of hidden layer neurons has been trained using the dotted points as training samples. As the number of free parameters grows, the matching error becomes smaller, but, at the same time, the curve outlook becomes less predictable. Note that the results can vary from simulation to simulation

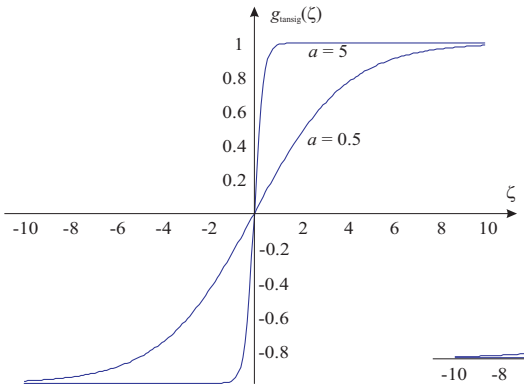



Figure 8.8: Hyperbolic tangent

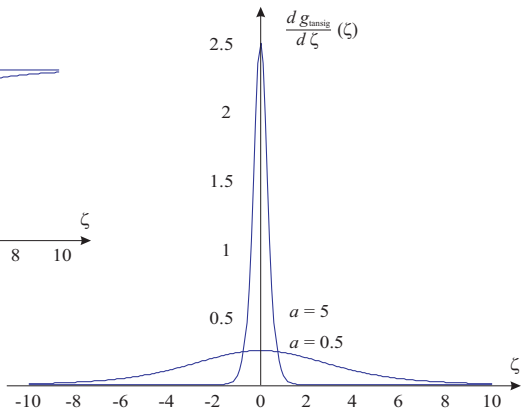



Figure 8.9: ... and its derivative

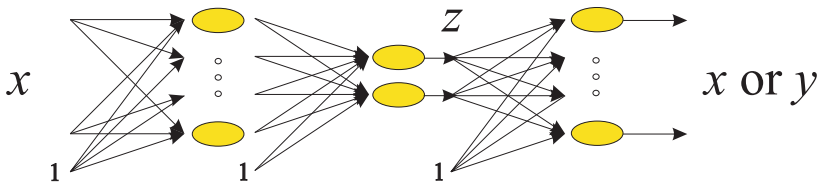



Figure 8.10: Neural network based "nonlinear PCA" (output $x$) and "nonlinear PLS" (output $y$). After training, the impact from the input to the output gets channelled through the variables $z$ of lesser dimension. If the mapping from $x$ to $x$ or $y$ still can be done, it must be so that the information has been successfully compressed

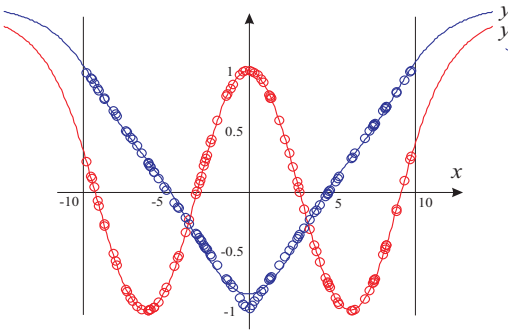



Figure 8.11: Two functions to be modeled (shown as circles), approximations shown using solid line type (see Fig. 8.12)

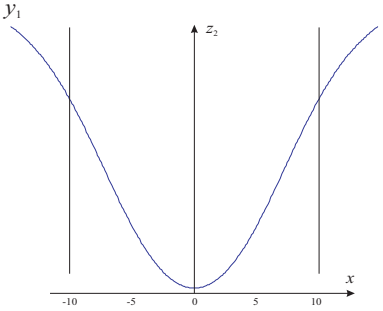



Figure 8.12: An intuitively reasonable nonlinear latent variable behavior, recognizing the symmetry of the signals in Fig. 8.11

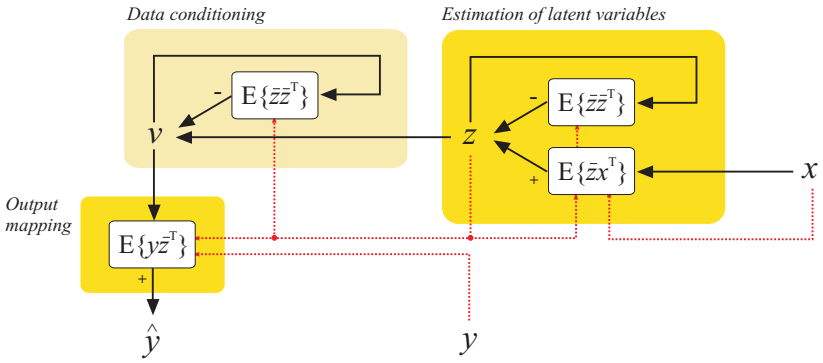



Figure 8.13: "Structure of "cybernetic regression". Note that the notations differ from those employed in [**?**]

# Lesson 9

# Application to Dynamic Models

All the above discussions assumed static dependency between the input and the output, so that the value of $y$ should be directly determined by the value $x$. In systems engineering, the models are usually dynamic — there is some "inertia" in the system, or "memory" between inputs and responses. Past affects the future, and successive measurements become interlinked. In principle, this makes the models considerably more complicated. However, the derived models can readily be extended to dynamic domains.

## 9.1 Representing dynamics

It turns out that basically static multivariate methods can be extended also to the determination of dynamic model parameters. The procedure is as follows: First, determine the state sequence, then compress the state space (using the familiar methods), and, finally, solve for the system matrices. To accomplish this, some basic theory is needed.

### 9.1.1 Capturing history

In the chapter 3, it was explained how structural complexity can be changed into dimensional complexity. A specially interesting form of structural complexity is caused by dynamic nature in the system being studied. How to determine the features to represent the dynamic phenomena as data?

In dynamic systems, there is *inertia* — it is not only the input but also the history that affects the current and future behaviors of the system. This infinite history can be captured in a finite set of time-series samples: System theory says that behaviors of a $d$'th order discrete-time dynamic system can be represented in terms of $d$ past samples:

$$y(\kappa) = f\left(y(\kappa-1), \ldots, y(\kappa-d_{\max}), u(\kappa), \ldots, u(\kappa-d_{\max})\right). \tag{9.1}$$

This means that the dynamic features in the data vector should be more or less delayed measurements of the signals. Assuming that the system input $u$ goes through the system causing the output $y$, one can express the information in the input sample vectors and in the output sample vectors collectively as the *state vector*

$$x(\kappa) = \begin{pmatrix} \frac{u(\kappa)}{\vdots} \\ \frac{u(\kappa - d_{\max})}{y(\kappa - 1)} \\ \frac{\vdots}{y(\kappa - d_{\max})} \end{pmatrix}. \tag{9.2}$$

That is, there are overlapping windows over the past time-series data. If the dynamics is assumed linear, there holds

$$y(\kappa) = F^T x(\kappa) \tag{9.3}$$

for some parameter matrix $F$. Here, it is assumed that the system dimension is not known beforehand, it is only assumed that dimension cannot exceed $d_{\max}$. Note that high dimensionality and excessive redundant variables is not assumed to be a problem — now it can be assumed that (preliminary) state $x$ contains all information back to the maximum length of system memory.

Indeed, high dimensionality is not a problem if appropriate latent variable methods are applied when the regression model between $x$ and $y$ is constructed. Again, assume that the latent variable, or the reduced *minimum state,* is denoted $z(\kappa)$. The preliminary state $x(\kappa)$ is first projected onto this minimum state $z(\kappa)$ and from there to output $y(\kappa)$.

So, nothing new here. In principle, the above scheme is already a working solution for implementing determination of dynamic models. The problem here is that $x(\kappa)$ needs to be reconstructed at each time point $\kappa$ separately — a more streamlined approach would be beneficial. What is more, more tailored formulations make it possible to employ the very powerful theory of linear dynamic systems. The rest of this chapter devoted to the challenges of writing the above model in the standard dynamic model form.

### 9.1.2 State-space models

There are various ways to represent dynamic systems in a mathematically compact way. In this context we only study *state-space models.* The basic idea here is that the history of the system is coded in the state vector $z$, as explained above, and the time-domain behavior of this state is coded in the model. Together with the future inputs the state determines the future outputs. The linear discrete-time state-space model can be written in the following general form:

$$\begin{cases} z(\kappa + 1) = Az(\kappa) + Bu(\kappa) + \epsilon(\kappa) \\ y(\kappa) = Cz(\kappa) + Du(\kappa) + e(\kappa). \end{cases} \tag{9.4}$$

Vector $u(\kappa)$ is the system input (dimension $n_u \times 1$), $z(\kappa)$ is the state vector (dimension $N \times 1$), and $y(\kappa)$ is the output (dimension $m \times 1$); matrices $A$, $B$, $C$, and $D$ have compatible dimensions. Even if the model is written here using the minimum-dimensional state vector, the state sequence is by no means unique, and the state model representation is by no means optimal in terms of available parameters. Vectors $e(\kappa)$ and $\epsilon(\kappa)$ represent white noise sequences so that

$$\mathrm{E}\{\epsilon(\kappa_1)\epsilon^T(\kappa_2)\} = \begin{cases} R_{zz}, & \text{if } \kappa_1 = \kappa_2 \\ 0, & \text{otherwise,} \end{cases} \tag{9.5}$$

and

$$\mathrm{E}\{e(\kappa_1)e^T(\kappa_2)\} = \begin{cases} R_{yy}, & \text{if } \kappa_1 = \kappa_2 \\ 0, & \text{otherwise,} \end{cases} \tag{9.6}$$

are the *state noise* and the *measurement noise* covariances matrices, respectively. Additionally, assume that $\epsilon(\kappa)$ and $e(\kappa)$ are mutually correlated:

$$\mathrm{E}\{\epsilon(\kappa_1)e^T(\kappa_2)\} = \begin{cases} R_{zy}, & \text{if } \kappa_1 = \kappa_2 \\ 0, & \text{otherwise.} \end{cases} \tag{9.7}$$

Note that this system model is a generalization of what has been studied this far: If matrices $A$, $B$, and $C$ are ignored, matrix $D$ directly corresponds to the static regression model $F^T$. From this it is easy to see that there are much more degrees of freedom in the dynamic model as compared to the static one.

The above state-space system structure is used, for example, by the Kalman filter, and a wide variety of analysis and control design methods are readily available for models that are presented in this form. If one were able to find all the system matrices directly from data, such method would be very useful — and finding such a technique is our objective now.

## 9.2 Subspace identification

The solution to the above problem is given by *subspace identification* (complete coverage can be found in [**?**]). This branch of research is rather new, developed mainly in the 1990's (yet having its roots in *realization theory* dating back to 1960's), and the texts discussing this material are typically difficult to follow for non-experts. However, the ideas are again very simple. It needs to be kept in mind that this presentation only explains the bare bones, and various enhancements could be (and have been) implemented.

### 9.2.1 Stochastic models

There exist various modifications of the basic model (9.4) that can be useful in different applications. For example, discarding the input $u$, assuming that the process is run exclusively by the noise, one faces the so called *stochastic realization problem:* What is the underlying succession of unmeasurable states $z$, together with the model structure, that best can explain the observed outputs $y$? This problem setting is characteristic to filtering problems, where measurements are corrupted by noise and the original variables should be recovered.

**State sequence**

The simplified system model that is exclusively driven by the noise processes becomes

$$\begin{cases} z(\kappa+1) = Az(\kappa) + \epsilon(\kappa) \\ y(\kappa) = Cz(\kappa) + e(\kappa). \end{cases} \tag{9.8}$$

Now the available data only consists of output time series samples. Assume that the block of $k$ successive data samples extends from time point $k_0 = 1$ to time $k$. Define vector of past outputs as follows:

$$\underset{d_{\max}m\times 1}{y_{\text{past}}}(\kappa) = \begin{pmatrix} y(\kappa-1) \\ \hline \vdots \\ \hline y(\kappa-d_{\max}) \end{pmatrix}. \tag{9.9}$$

The future signals are, correspondingly,

$$\underset{d_{\max}m\times 1}{y_{\text{future}}}(\kappa) = \begin{pmatrix} y(\kappa+d_{\max}-1) \\ \hline \vdots \\ \hline y(\kappa) \end{pmatrix}.$$

The data matrices $Y_{\text{past}}$ and $Y_{\text{future}}$ are constructed as before, stacking transposed sample vectors on top of each other, the indexes running from $d_{\max}+1$ to $k-d_{\max}+1$. Note that the matrices have $k-2d_{\max}+1$ rows rather than the original $k$. The preliminary state variable can be constructed immediately:

$$X = \begin{pmatrix} Y_{\text{past}} \end{pmatrix}. \tag{9.10}$$

The dimension of the preliminary system state is high and the states are highly redundant — but the main thing here is that the originally dynamic problem has been transformed into a static one, and all of the tools that have been presented in the previous chapters for dealing with static models can be utilized for dimension reduction, or for determining the latent vector matrix $Z$ containing the $N$ dimensional minimum state vectors. According to system theoretic understanding, one should select $N = d$.

**Compressing the state**

There is still plenty of freedom. One can compress the state space utilizing the state sequence $X$ exclusively, or one can take into account the fact that there should be a mapping from the state to the future states and outputs. These alternative viewpoints give rise to approaches of PCA/PLS/CCA type. Further, applying some independent component approach to the state selection should be something new in this field (see Exercises). Here, the PLS option is briefly studied.

One should determine the compressed state so that mapping from this state to next state and to the output could be accomplished.

Define $X^-$ as a submatrix of $X$, where the *last* row is eliminated (the newest state vector), and, similarly, define $X^+$ as a submatrix, where the *first* row is eliminated (the oldest state vector):

$$X^- = \begin{pmatrix} x^T(1) \\ \vdots \\ x^T(k-2d_{\max}) \end{pmatrix} \quad \text{and} \quad X^+ = \begin{pmatrix} x^T(2) \\ \vdots \\ x^T(k-2d_{\max}+1) \end{pmatrix}. \tag{9.11}$$

These matrices stand for the succession of states, that is, elements in $X^+$ are the next state variables corresponding to the state variables in $X^-$. Further, define (dimensionally and causally matching) output matrix

$$Y = \begin{pmatrix} y^T(d_{\max}) \\ \vdots \\ y^T(k-d_{\max}-1) \end{pmatrix}, \tag{9.12}$$

consisting of altogether $k-2d_{\max}$ rows. Now the intended mapping can be expressed so that the virtual input data and the virtual output data, respectively, are

$$\mathcal{X} = \begin{pmatrix} X^- \end{pmatrix} \quad \text{and} \quad \mathcal{Y} = \left( \begin{array}{c|c} X^+ & Y \end{array} \right). \tag{9.13}$$

When the PLS model is constructed for this problem, and the latent variables are determined, it is this sequence of latent variable vectors (in the input-oriented subspace) that can be selected as the compressed state sequence, so that $Z = \mathcal{Z}$.

Typically, the selection of the state dimension will not be unique, so that the system order is more or less uncertain. Note that this uncertainty is an inherent property of real distributed parameter systems — the exact system dimension is just a mathematical abstraction that has been approved as there are no alternatives for representing the system behavior in such a compact form.

### System matrices

To proceed, the resolved compressed state matrix $Z$ needs to be restructured. This is done as in (9.11): Define $Z^-$ as a submatrix of $Z$, where the *last* row is eliminated (the newest compresswed state vector), and, similarly, define $Z^+$ as a submatrix, where the *first* row is eliminated (the oldest compressed state vector):

$$Z^- = \begin{pmatrix} z^T(1) \\ \vdots \\ z^T(k-2d_{\max}) \end{pmatrix} \quad \text{and} \quad Z^+ = \begin{pmatrix} z^T(2) \\ \vdots \\ z^T(k-2d_{\max}+1) \end{pmatrix}. \tag{9.14}$$

These matrices stand for the succession of states. Also, define the output matrix as in (9.12).

When the state sequence is now known, the subsequent steps of subspace identification nicely illustrate the power of linear machinery — the final model (9.8)

can be constructed by matching the parameters against the now known state variables. Because the system model can be written as

$$\left(\frac{z(\kappa+1)}{y(\kappa)}\right) = \left(\frac{A}{C}\right)\left(\ z(\kappa)\ \right), \tag{9.15}$$

there holds

$$\left(\ Z^+ \ \middle|\ Y\ \right) = \left(\ Z^-\ \right)\cdot\left(\ A^T \ \middle|\ C^T\ \right). \tag{9.16}$$

The matrices $A$ and $C$ can be solved in the least squares sense:

$$\left(\frac{A}{C}\right) = \left(\ Z^+ \ \middle|\ Y\ \right)^T \cdot \left(\ Z^-\ \right)\left(\left(\ Z^-\ \right)^T \cdot \left(\ Z^-\ \right)\right)^{-1}, \tag{9.17}$$

where the individual system matrices can be identified as partitions of the resulting matrix the first $N \times N$ elements being allocated for $A$.

### Noise covariances

The mismatch between data and the model gives the estimates for the noise. Because the state and output sequences estimated by the model can be written as

$$\left(\ \hat{Z}^+ \ \middle|\ \hat{Y}\ \right) = \left(\ Z^-\ \right)\cdot\left(\ A^T \ \middle|\ C^T\ \right), \tag{9.18}$$

the estimation error becomes

$$\begin{aligned} E &= \left(\ Z^+ \ \middle|\ Y\ \right) - \left(\ \hat{Z}^+ \ \middle|\ \hat{Y}\ \right) \\ &= \left(\ Z^+ \ \middle|\ Y\ \right) - \left(\ Z^-\ \right)\cdot\left(\ A^T \ \middle|\ C^T\ \right), \end{aligned} \tag{9.19}$$

and the noise covariances can be mechanically solved from this. The approximations for the covariances are again found as partitions of

$$\left(\frac{R_{zz} \ \middle|\ R_{zy}}{R_{zy}^T \ \middle|\ R_{yy}}\right) = \frac{1}{k'}\cdot E^T E. \tag{9.20}$$

Here, the normalization factor $k' = (k - 2d_{\max} + 1) - (N + m)$ is the number of data samples minus the overall data dimension. Again, in typical applications (like when doing Kalman filtering, see below), it is the ratio between $R_{zz}$ and $R_{yy}$ that is the most important information, not the absolute scalings.

Assuming that the input signal exactly can explain the output, the error covariances are zero matrices. In such cases one is facing the *deterministic realization problem;* it turns out that the realization problem is solved as a special case of the more general subspace identification problem. The above discussion can be further extended.

### 9.2.2 Stochastic-deterministic models

A more complicated case is faced when the complete model (9.4) is employed. Now define sequences of past inputs and outputs as follows:

$$u_{\text{past}}(\kappa) = \left(\begin{array}{c} u(\kappa-1) \\ \hline \vdots \\ \hline u(\kappa-d_{\max}) \end{array}\right) \quad \text{and} \quad y_{\text{past}}(\kappa) = \left(\begin{array}{c} y(\kappa-1) \\ \hline \vdots \\ \hline y(\kappa-d_{\max}) \end{array}\right) \tag{9.21}$$

The future signals are, correspondingly,

$$u_{\text{future}}(\kappa) = \left(\begin{array}{c} u(\kappa+d_{\max}-1) \\ \hline \vdots \\ \hline u(\kappa) \end{array}\right) \quad \text{and} \quad y_{\text{future}}(\kappa) = \left(\begin{array}{c} y(\kappa+d_{\max}-1) \\ \hline \vdots \\ \hline y(\kappa) \end{array}\right).$$

The data matrices $U_{\text{past}}$, $U_{\text{future}}$, $Y_{\text{past}}$, and $Y_{\text{future}}$ are again constructed in the same way as above.

To find a good model for the data in this more complicated case, one should distribute the burden of explaining the future behaviors appropriately among the two sources of information — the known history, and the unknown future inputs. One has to apply the mathematical theory of *oblique projections.* To give an idea of what this means, construct matrices

$$\mathcal{X} = \left(\begin{array}{c|c|c} Y_{\text{past}} & U_{\text{past}} & U_{\text{future}} \end{array}\right) \tag{9.22}$$

and

$$\mathcal{Y} = \left(\begin{array}{c} Y_{\text{future}} \end{array}\right), \tag{9.23}$$

so that there should exist a mapping from $\mathcal{X}$ to $\mathcal{Y}$. Indeed, this can be constructed using least squares matching, giving a prediction model for the future:

$$\begin{array}{rcl} \mathcal{Y}_{\text{est}} & = & \mathcal{X}_{\text{est}} \cdot \left(\mathcal{X}^T\mathcal{X}\right)^{-1}\mathcal{X}^T\mathcal{Y} \\ & = & \left(\begin{array}{c|c|c} Y_{\text{past,est}} & U_{\text{past,est}} & U_{\text{future,est}} \end{array}\right) \cdot \left(\mathcal{X}^T\mathcal{X}\right)^{-1}\mathcal{X}^T\mathcal{Y}. \end{array} \tag{9.24}$$

In practice, the invertibility of $\mathcal{X}^T\mathcal{X}$ may be poor, specially id $d_{\max}$ has too high value as compared to the system dimension, and some dimension reduction technique may again be needed.

The values of $U_{\text{future,est}}$ are not known at time $\kappa$, and to make this model useful, it has to be divided in parts:

$$\begin{array}{rcl} \mathcal{Y}_{\text{past}} + \mathcal{Y}_{\text{future}} & = & \left(\begin{array}{c|c|c} Y_{\text{past,est}} & U_{\text{past,est}} & \mathbf{0} \end{array}\right) \cdot \left(\mathcal{X}^T\mathcal{X}\right)^{-1}\mathcal{X}^T\mathcal{Y} \\ & & + \left(\begin{array}{c|c|c} \mathbf{0} & \mathbf{0} & U_{\text{future,est}} \end{array}\right) \cdot \left(\mathcal{X}^T\mathcal{X}\right)^{-1}\mathcal{X}^T\mathcal{Y}. \end{array} \tag{9.25}$$

The variables in $\mathcal{Y}_{\text{past}}$ can be interpreted as containing all available information about the system past. Based on this, one can define the "refined" data matrix where the contribution of the future inputs is eliminated:

$$X = \mathcal{Y}_{\text{past}} = \left(\begin{array}{c|c|c} Y_{\text{past,est}} & U_{\text{past,est}} & \mathbf{0} \end{array}\right) \cdot \left(\mathcal{X}^T\mathcal{X}\right)^{-1}\mathcal{X}^T\mathcal{Y}. \tag{9.26}$$

It is possible to choose this data as constituting the preliminary system states. The determination of the reduced state matrix $Z$ can be accomplished exactly as was done before.

Again, the resolved state matrix $Z$ is restructured into $Z^-$ and $Z^+$ parts. Further, define (dimensionally matching) input and output matrices

$$U = \begin{pmatrix} u^T(d_{\max}) \\ \vdots \\ u^T(k - d_{\max} - 1) \end{pmatrix} \quad \text{and} \quad Y = \begin{pmatrix} y^T(d_{\max}) \\ \vdots \\ y^T(k - d_{\max} - 1) \end{pmatrix}, \tag{9.27}$$

consisting of altogether $k - 2d_{\max}$ rows. Now, because the system model can be written as

$$\left(\frac{z(\kappa+1)}{y(\kappa)}\right) = \left(\frac{A \mid B}{C \mid D}\right)\left(\frac{z(\kappa)}{u(\kappa)}\right), \tag{9.28}$$

there holds

$$\left(\begin{array}{c|c} Z^+ & Y \end{array}\right) = \left(\begin{array}{c|c} Z^- & U \end{array}\right) \cdot \left(\frac{A^T \mid C^T}{B^T \mid D^T}\right). \tag{9.29}$$

The matrices $A$, $B$, $C$, and $D$ can be solved in the least squares sense:

$$\begin{aligned} \left(\frac{A \mid B}{C \mid D}\right) \;=\; & \left(\begin{array}{c|c} Z^+ & Y \end{array}\right)^T \cdot \left(\begin{array}{c|c} Z^- & U \end{array}\right) \\ & \cdot \left(\left(\begin{array}{c|c} Z^- & U \end{array}\right)^T \cdot \left(\begin{array}{c|c} Z^- & U \end{array}\right)\right)^{-1}, \end{aligned} \tag{9.30}$$

where the individual system matrices can be identified as partitions of the resulting matrix (again, note the possible invertibility problems). The reconstruction errors give the estimates for the noise. Because the state and output sequences estimated by the model can be written as

$$\left(\begin{array}{c|c} \hat{Z}^+ & \hat{Y} \end{array}\right) = \left(\begin{array}{c|c} Z^- & U \end{array}\right) \cdot \left(\frac{A^T \mid C^T}{B^T \mid D^T}\right), \tag{9.31}$$

the estimation error becomes

$$\begin{aligned} E \;&=\; \left(\begin{array}{c|c} Z^+ & Y \end{array}\right) - \left(\begin{array}{c|c} \hat{Z}^+ & \hat{Y} \end{array}\right) \\ &=\; \left(\begin{array}{c|c} Z^+ & Y \end{array}\right) - \left(\begin{array}{c|c} Z^- & U \end{array}\right) \cdot \left(\frac{A^T \mid C^T}{B^T \mid D^T}\right), \end{aligned} \tag{9.32}$$

and one has

$$\left(\frac{R_{zz} \mid R_{zy}}{R_{zy}^T \mid R_{yy}}\right) = \frac{1}{k'} \cdot E^T E. \tag{9.33}$$

## 9.3 Practical aspects

Subspace identification differs much from traditional identification methods, and its properties are also different from what one is familiar with. Understanding these boundary conditions is necessary to effriciently exploit the techniques.

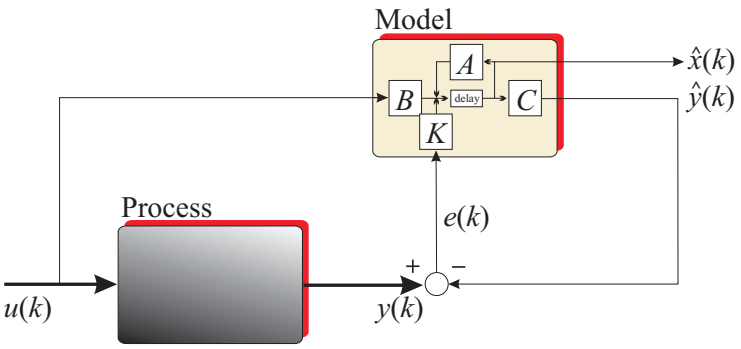



Figure 9.1: Making the process behaviors transparent

### 9.3.1 Comparisons

As this subspace identification approach is compared to the standard black-box identification procedures, some clear benefits can be seen:

- The selection of the system dimension needs not be done beforehand as in traditional black-box identification; the algorithms give tools for appropriate on-line determination of the dimension.
- The models are naturally applicable to multi-input/multi-output (MIMO) systems, whereas traditional identification only studies systems of one input and one output.
- Coloured noise becomes modeled without additional efforts, whereas standard techniques have to rely on complicated nonlinear, iterative methods.

The dynamical phenomena that are visible in the data are all integrated in the model structure itself, no matter where those phenomena originate from. Indeed, the properties of observed noise are also characteristic to the system and it is natural that this information is also coded in the model in a consistent way. Some state variables may be allocated for presenting the noise — if this dynamics is relevant enough.

However, there are also some drawbacks as the dynamic modeling approach is compared to the static approach. The model is more sophisticated, and there are more quality requirements concerning the data: Longer sequences of valid data are needed, as no outlier or missing data samples can be dropped from within the sequence. For the same reason, also model validation becomes problematic, because longer continuous sequences of validation data are needed; the leave-one-out cross-validation (with one sample being dropped at a time) cannot be implemented.

### 9.3.2 Emulating the process

A natural way to exploit the models given by subspace identification is reached through *state estimation:* When the process state is known, all state-based analysis and design methods are available. Typically, one can only measure the

inputs and outputs of a complex process; now, one can implement a *process simulator,* and this simulator, rather than being a black-box system, is a *white-box system* with explicitly known structure and measurable state. When simulator has the same dynamics as the process, and when it is given the same inputs as the actual process has, the behaviors of the simulator and the process itself should be equal (see Fig. 9.1). The feedback matrix $K$ is used for correcting the state according to the observed error $e(k) = y(k) - \hat{y}(k)$, or difference between process and model outputs.

The *Kalman filter* is a natural companion of models derived by subspace identification: The feedback matrix $K$ can be optimally determined based on the observed system and noise properties [**?**]. The model structure with the matrices $A$, $B$, $C$, and $D$ are explicitly given by subspace identification, and even the noise properties $R_{xx}$, $R_{yy}$, and $R_{xy}$, are solved, making the determination of the Kalman filter a straightforward task.

Often, specially when the system dimension is high, determining all the system matrices is a difficult task using traditional techniques — and the matrices $R_{zz}$ and $R_{yy}$ are typically used as tuning parameters only, typically chosen to be simply diagonal. Now, using subspace identification, all these data structures are determined to optimally match the data. However, as it turns out in the next chapter, optimality and robustness are different things also in this case. Understanding the nature of the problem — again caused by the multivariate nature and collinearity — makes it possible to attack the problems.

### 9.3.3 Preprocessing and postprocessing

When applying subspace identification, model construction can be affected again by appropriately scaling of the data samples. In addition to the normal scalings, etc., there also exist other ways to affect the modeling results. Specially, because of the dynamic nature of the problem, or because there is a time-domain succession of the samples, one can apply *frequency weighting* to emphasize (or attenuate) variations in some frequency ranges. Here, one would not like to affect the model construction process itself (compare to [**?**]) — the goal is to implement the frequency weighting directly in the data.

Assume that there exists some linear dynamic model between inputs and outputs, so that the mapping matrix $F(q^{-1})$ contains delay operator polynomials

$$y(k) = F^T(q^{-1})\, u(k). \tag{9.34}$$

Now, the model remains valid if the left-hand side and the right-hand side are further filtered by the operator polynomial $h(q^{-1})$; because of the linearity assumption, the operators are commutative:

$$h(q^{-1})y(k) = h(q^{-1})F^T(q^{-1})\, u(k) = F^T(q^{-1})\, h(q^{-1})u(k). \tag{9.35}$$

If one defines the new data as $u'(\kappa) = h(q^{-1})u(\kappa)$ and $y'(\kappa) = h(q^{-1})y(\kappa)$, for this new data the original model still applies:

$$y'(k) = F^T(q) \cdot u'(k). \tag{9.36}$$

The system structure, and also the state-space model, can be determined for the modified data as well as for the original data. Nominally there is no change in the system — however, as always when modeling is based on the variance properties, the results change as errors are weighted in different ways in different frequency bands, depending on the ratios between information and noise along the frequency axis. In practice, one can apply low-pass, band-pass, band-stop, or high-pass filtering, as studied below.

It is very common in real processes that the levels of the signals can vary, even though the dynamic properties essentially remain intact. If one applies the standard mean-centering for training data, this mean does not necessarily remain constant, and the biased model can become invalid. A simple way to avoid this problem is to apply "on-line centering", or high-pass filtering, so that the zero-frequency properties are eliminated altogether:

$$h(q^{-1}) = 1 - q^{-1}, \tag{9.37}$$

meaning that the differentiated variables (variable vectors) in time domain are defined as

$$\Delta y(\kappa) = y(\kappa) - y(\kappa - 1). \tag{9.38}$$

However, assuming that there exists high-frequency noise in the system, differentiation emphasizes such noise signals. It can be reasonable to define some upper limit frequency for the filter, for example, by further filtering

$$y'(\kappa) = \lambda\, y'(\kappa - 1) + (1 - \lambda)\ \Delta y(\kappa). \tag{9.39}$$

This lead-lag compensator can also be applied for inputs and outputs alike, and subspace identification is applied for those signals. However, when the model is used for estimation, data preprocessing has to be ripped off by applying inverse postprocessing to reach the actual signal estimates. First, the differentiated signal estimates are found by inverting (9.39):

$$\Delta\hat{y}(\kappa) = \frac{1}{1-\lambda}\,\hat{y}'(\kappa) - \frac{\lambda}{1-\lambda}\,\hat{y}'(\kappa - 1). \tag{9.40}$$

In principle, the differentiation within these signals can be eliminated by integration — however, no pure integrators should be applied, as biases in signals would increase during this process. It is motivated to introduce a "leaking integrator" that eventually tends towards the actual measurements:

$$\hat{y}(\kappa) = \mu\ (\hat{y}(\kappa - 1) + \Delta\hat{y}(\kappa)) + (1 - \mu)\ y(\kappa). \tag{9.41}$$

Here, the parameters $\lambda$ and $\mu$ are forgetting factors to be adjusted appropriately to fit the signal and process properties. The pure discrete-time derivator in (**??**) should also be modified to match the inverse operation (9.41).

## 9.4 Case study: Towards "smart devices"

Here, an example is presented where the above tools are being experimented and exploited. This research and development work is currently being carried

out in the industrial scale. It is clear that the full potential of the new methods cannot yet be seen.

### 9.4.1 Data based data reconciliation

In process industries knowing the contents of processed materials is of utmost importance. An *X-ray fluorescence analyzer* is an efficient tool when doing such analyses in mineral processing. The X-ray fluorescence analyzer excites the atoms and measures the emitted photons: The emission spectra are unique to the atoms, and, in principle, the atom contents can be solved by analyzing the spectral peaks. However, because of the environmental conditions and because of the physical reasons, the results are corrupted by noise and other stochastic phenomena. To enhance the estimates, one needs *calibration models* to map between the measured spectra to actual concentration estimates. The superposition of individual spectra is a linear process, and it is plausible that linear multivartiate methods can efficiently be used for this calibration purpose. And, indeed, such developments have been taking place recently (for example, see [**?**]).

Even though the static mapping models from the intensities to concentrations could be appropriately constructed, there is need for closer studies. It does not help if the sample could be analyzed exactly, if that sample is not representative. The random samples do not necessarily reflect the overall contents of the slurry — this is due to the changes in slurry densities and grain size distributions. To dampen the variability in the measurements, different kinds of filtering techniques have been applied. The traditional approach is exponential filtering with some forgetting factor: That is, one only partly trusts the measurements, keeping the prior average level of estimates as the starting point. If there are some consistent changes in the slurry properties, such filtered estimate becomes delayed, as the average level only slowly follows the changes. There is a trade-off between accuracy and smoothness of estimates. It is not the variability that should be dampened, if it reclects the reality; but there is no way to locally determine whether some change in the measurement values is relevant or not. Are there other ways to enhance the estimates?

*Data reconciliation* is a bunch of techniques for employing the system structure for enhancing the noisy measurements. Typically, data reconciliation is based on mass or volume balances: If the inputs and outputs of some process structure are recorded, the mass/volume flows should compensate each other. For example, a flotation cell (or a bang of cells) is such a vessel: The two outgoing flows (concentrate and gangue flows) have to balance the incoming slurry flow. Because of the practical challenges, data reconciliation is often carried out statically, for steady-state levels, so that the flows compensate each other only in the long run.

However, often the structures or dependencies within a complex process are not known, and there exist no explicit constraint equations. And even if the constraints were known, there may be no measurements: For example, an X-ray analyzer is such an expensive device that measurements are carried out only in the most informative locations in the process. But, after all, data reconciliation is based on the redundancy among data, and if the measurements are cut to

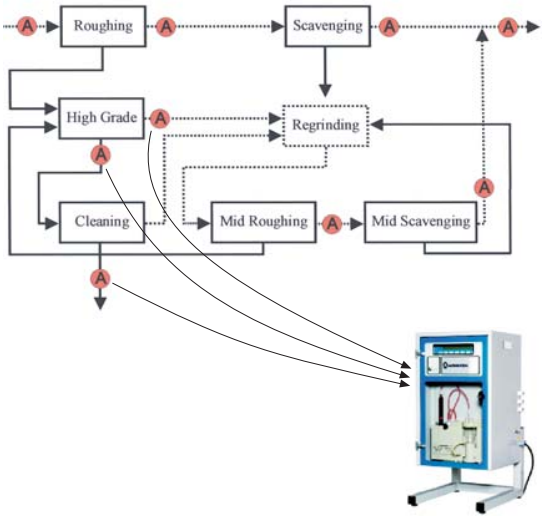



Figure 9.2: The analyzer is already the "heart" of the process — make it also the "brain"

minimum, the crucial measurements closing the logical loops may be missing. Still, there most probably exist some interdependencies between signals. How to implement the idea of data reconciliation as a robust enough scheme, applicable in such a complex environment — and, what is more, how to extend data reconciliation to dynamic cases?

The presented ideas of statistical multivariate modeling, and specially the ideas of subspace identification, make it possible to implement "data based data reconciliation". As an example case, again study the Pyhäsalmi concentrator plant (see chapter 3).

The zinc circuit alone is a rather complicated network (see Fig. 9.2), different recirculations having been implemented to enhance the recovery rate and purity of the concentrate. The current structures in the process are a result of an evolutionary process, and they are characteristic to this unique process plant. In addition to the physical feedforward and feedback flows, there is yet another level of information flows, being caused by the control structures, making the overall system fully connected and "pancausal". Indeed, the degrees of freedom in the system are reduced by the interconnected variables, and the remaining variation assumedly takes place only in a rather low-dimensional subspace. Even if all the constraints cannot be explicitly tracked, it can be assumed that in the cybernetic system the balance around the steady-state average is maintained, and it suffices to capture the properties of the "emergent model" (see chapter 11). Because of the balance and assumed minimization of signal variations, local linearity can be assumed, at least if no structural changes take place in the process.

The concentrations in the most relevant flows are measured: Slurry samples are taken to the centralized analyzer unit, where the samples are analyzed one after the other in a sequence, the whole cycle lasting some 20 minutes. However, significant changes can take place in the process during these time intervals, and to implement efficient control and monitoring applications, it would be of utmost importance to be capable of reliably estimating the signal behaviors during the

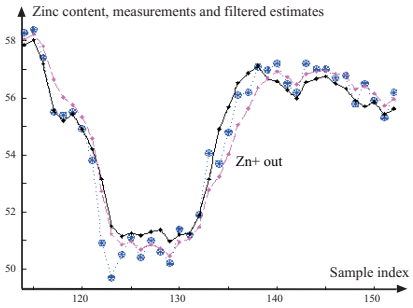



Figure 9.3: Original unfiltered measurements are shown as dots; the "50% smoothened", exponentially filtered estimates are shown along dashed lines, and the model-based estimates are along the solid lines. The new filtering scheme is considerably faster

periods of "blindness". One should implement *soft sensors* based on process models; to construct the models that utilize the available measurements, one has to implement *sensor fusion.* It is clear the slurry concentrations in different parts of the process are interrelated, and utilizing these relationships, behaviors "downstream" can be estimated. Rather than concentrating on the individual signals, one should find the overall dependencies — this global-level state of the whole zinc circuit can be exploited for estimation. The correlations between measurements are rather low, but if many pieces of evidence are combined in a clever way, useful models can still be found.

What is this *clever way,* then? It is evident that subspace identification, as accompanied by the appropriate Kalman filter, is the method of choice: As the high dimensionality is no problem, the various measurent channels can be efficiently exploited, and also the minor pieces of correlation information can be extracted. Depending on the available information, different model structures are possible:

1. **Stochastic model.** The measurements constitute the output vector $y$, and the task is to determine the system state vector $z$, assuming that the changes in the system state are driven by noise. The internal system model makes it possible to enhance the "downstream" estimates by exploiting the "upstream" observations.

2. **Stochastic-deterministic model.** As explained in chapter 3, there are also cameras installed on top of the flotation cells: The features extracted from the camera images can be employed as causally preceding information, giving delayless data of the froth outlook (color, "thickness", etc.) that is assumedly related with the concentrate properties.

It is a closed-loop system that is being implicitly identified. The causalities are blurred, one cannot distinguish between the process and the controls. If further feedbacks or process re-design is to be implemented, the derived models have to be interpreted by a domain-area expert.

In Fig. 9.3, results are shown when the zinc concentrate from the "Cleaning" cell is estimated utilizing also the measurement information from the "High Grade" cell (see Fig. 9.2). The preliminary experiments with the subspace identification schemes are promising, and faster reactions to changes in concetrate properties can be reached. When the estimates are "calibrated" after each measurement, the signals can be estimated during the inter-sample periods using the model. However, it turns out that the data quality needs to be

emphasized: Just as global information is exploited for local estimation, local problems become global, perhaps ruining the whole plant operation. The outlier values need to be detected and fixed reliably on-line, so that new challenges are faced.

### 9.4.2 Connections to AI*

The discussion of "clever" data analyses can be extended.

In Artificial intelligence (AI) one tries to implement "intelligent-looking" functionalities in computers. Traditionally, AI approaches are symbolic, meaning that the constructs, declarative or procedural, need to be explicitly programmed: There is then no connection to data, and there is no possibility of adaptation. Human is needed as an interface between the system and its environment. Real intelligence can be defined as ability of appropriate reacting to its environment and adapting according to it, and fixed structures are necessarily deficient.

The neural netwroks, etc., have been exploited to reach data-orientedness and associative reasoning, and they can be applied to accomplish complicated pattern recognition tasks. However, something essential seems to be missing: The structures themselves are still fixed, non-adaptive. How to find a good compromise, a structural framework where adaptation is possible?

Structures that characterize human cognition are causal, meaning that human mind naturally organizes observations in cause/effect hierarchies. Such causality structures cannot be seen in the data — but, again, one can do assumptions of how the structures are reflected in data. If it is assumed that correlation structures can be used to represent causalities, temporal ordering among data can be found applying the techniques presented above. When the data dimension is high and redundant, the challenge of the modeling method is to cope with the high dimensionality, and detect the appropriate connections — getting "wiser" is about ignoring irrelevant connections. After adaptation the dynamic state models become "numeric inference models" — when given the current state, the system can predict what happens afterwards. Many philosophical problem settings become very concrete: Questions concerning *ontologies* and *semantics* are wiped away, as everything is based on *contextual semantics* defined in terms of similarities, or correlation structures among variables. As the "numeric concepts", or the state variables, have continuous values, the problems of "hermeneutic circles", or the convergence properties of infinite recursions can be studied mathematically. In this way, it seems that multivariate modeling can offer new possibilities for research in AI.

On the other hand, models derived in the field of AI can perhaps give new tools for extending the dynamic model structure beyond linearity. For example, the sparse coding schemes derived for cognitive tasks can perhaps be extended to dynamic applications.

## Computer exercises

1. Study the properties of the subspace identification algorithm; try different parameter values in the following:

```
[U,Y] = dataDyn(3,2,1,100,0.001);
regrSSI(U,Y,5);
```

2. Analyze the different compression techniques in state dimension reduction. First, define time series data as

```
[u,y] = dataDyn(1,0,1,100,0.1);
X = [y(1:96),y(2:97),y(3:98),y(4:99),y(5:100)];
Xminus = X(1:size(X,1)-1,:);
Xplus = X(2:size(X,1),:);
```

Compare the first principal component basis vector to the system transient behavior:

```
thetaPCA = regrPCA(X,1)
```

Compare PLS and CCA. How can you explain the differences in the eigenvalue behavior? How about the latent vectors?

```
[thetaPLS,phiPLS] = regrPLS(Xminus,Xplus)
[thetaCCA,phiCCA] = regrCCA(Xminus,Xplus)
```

What happens to the independent components in a dynamic system? Can you explain the results?

```
U = dataIndep(1000,'f1','f2');
[U,Y] = dataDyn(2,U,3);
X = [Y(1:999,:),Y(2:1000,:)];
thetaICA = regrICA(regrWhiten(X),-1)
```

# Lesson 10

# Relations to Systems Engineering

This far we have been exclusively interested on data, and on models that are constructed directly from that data. It seems that the traditional control engineering practices that are usually based on first principles models would not have very much to do with these discussions. However, finally it is numeric data that flows through these physical models, too, even if the model construction had been based on qualitative studies; the structure of the model determines how the data is manipulated in the system, directly dictating how nice behavioral properties that system has. In this chapter we study how the multivariate ideas and approaches can be applied for analysis and manipulation of existing system models so that the *expected behavior* of the hypothetical data being processed in the system is taken into account.

In this context, some specially important aspects of modern control engineering will be concentrated on, among them *balanced model reduction* and *state estimation.*

## 10.1 MIMO vs. SISO systems

Traditional control design has been based on SISO (single input, single output) models, meaning that complex plants are reduced to simple control loops. This is natural, because the SISO systems are easily comprehensible for the system designers as well as for field operators. Yet, it is clear that local optimization of single control loops does not result in optimal behavior of the larger plant, and there is need to be able to design and analyze more complex models of multiple interconnected inputs and outputs (MIMO systems).

Because of the intuitive understandability and long tradition of SISO systems, the main paradigm in MIMO control design has been to somehow extend the SISO design ideas to the multivariate cases, or, rather, to make the MIMO systems *look like* sets of SISO systems.

One usually starts from the transfer function matrix

$$G(s) = \begin{pmatrix} G_{11}(s) & \cdots & G_{1,n}(s) \\ \vdots & \ddots & \\ G_{m,1}(s) & & G_{m,n}(s) \end{pmatrix}, \tag{10.1}$$

that characterizes a linear, dynamic multivariate system of $n_u$ inputs and $m$ outputs in Laplace (frequency) domain; the algebraic dependency $\mathcal{L}\{y(t)\}(s) = G(s) \cdot \mathcal{L}\{x(t)\}(s)$ governs the behavior of the Laplace transformed input and output signal vectors $\mathcal{L}\{u(t)\}(s)$ and $\mathcal{L}\{y(t)\}(s)$. Sustituting $s \to j\omega$ in (10.1), the transfer properties of sinusoidals of angular frequency $\omega$ are directly given, the absolute value revealing the amplitude and the angle of the complex-valued number revealing the phase of the output sinusoidal. Now, it is immediately clear that the interconnections between different inputs and outputs can be minimized if the matrix (10.1) is somehow *diagonalized,* and, indeed, constructing the singular-value decomposition, this can be done:

$$G(j\omega) = \Xi(\omega) \cdot \Sigma(\omega) \cdot \Psi^H(\omega), \tag{10.2}$$

the matrix $\Sigma(\omega)$ containing the Hankel singular values $\sigma_i(j\omega)$ (to be discussed more later) on the diagonal. This means that if the measured Laplace-domain output signals are multiplied by $\Psi^H(\omega)$ and the constructed control signals by $\Xi(\omega)$, the system looks like a set of separate SISO systems, and direct SISO design is possible for all input-output pairs. However, note that the matrices $\Xi(\omega)$ and $\Psi(\omega)$ are complex valued, and they are functions of $\omega$; using constant (real) matrices, this diagonalization can only be approximate. To easily get rid of complex factors, the diagonalization is typically optimized for zero frequency.

The frequency domain stability and sensitivity analysis techniques (stability margins, etc.) are also originally developed for SISO systems. These techniques can easily be extended to the multivariate case, if one is only interested in the worst-case behavior: The above Hankel singular values $\sigma_i(j\omega)$ reveal the system gains, and studying the largest of them, the *principal gain* $\bar{\sigma}_i$ for all frequencies, gives information about the maximum possible system response, given the most pathological input signal. Constructing controllers concentrating on the open-loop principal gains, it is possible to assure system stability in all situations and for all inputs (for example, see [30]).

## 10.2 Dimension reduction

The traditional way to apply multivariate techniques (as discussed above) are somewhat crude, not really taking into account the dynamic nature beneath the numbers, but forcing the complex systems into the SISO framework. In what follows, more sophisticated approaches discussed.

### 10.2.1 About state-space systems

The basis of modern systems engineering is the state-space model; as compared to the models in the previous chapter, now we start with its deterministic version

of it:

$$\begin{cases} x(\kappa+1) = Ax(\kappa) + Bu(\kappa) \\ y(\kappa) = Cx(\kappa) + Du(\kappa). \end{cases} \tag{10.3}$$

Here, $u$ and $y$ are the system input and output of dimension $n_u$ and $m$, respectively, and $x$ is the state of dimension $d$. It needs to be noted that, from the input/output behavior point of view, the state realization is not unique: For example, if there is an invertible matrix $L$ of size $d \times d$, one can define another state vector $x' = Lx$ so that the following state-space model fulfills the same relationship between $u(\kappa)$ and $y(\kappa)$:

$$\begin{cases} x'(\kappa+1) = LAL^{-1}x'(\kappa) + LBu(\kappa) \\ y(\kappa) = CL^{-1}x'(\kappa) + Du(\kappa). \end{cases} \tag{10.4}$$

Intuitively, the strength of the state-space model is that the real internal phenomena within the system can also be included in the model. In such cases, when the model has been constructed starting from first principles, the state representation for the system may be unique, and state transformations in the above way are meaningless. However, when the modern controller construction approaches like robust or optimal control theory are applied, the resulting controller is also given in the state-space form — but now the states are constructed by the mathematical machinery and they do no more have any physical relevance. The only thing that counts is the input-output behavior of the controller.

The question that arises is: How should we define the state vector of the model so that some of the model properties would be enhanced? Or, more specifically, how should the matrix $L$ be selected so that the *essence* of the state components would be revealed[1]?

When the controller is constructed using robust or optimal control theory, the resulting controller is usually high-dimensional. There are practical difficulties when using this kind of controllers — it may take long time to calculate the control actions and it may take a long time before the controller reaches the correct operating state after startup. In many cases, the models need to be simplified for practical purposes.

It is typical that some of the states are more relevant than the others. Could we reduce the number of states without losing too much, without essentially changing the dynamic behavior of the model? It sounds plausible that the methodologies presented in the previous chapters could be applied for this purpose, and, indeed they can. Now it is the most relevant dynamic modes that we would like to retain, while ignoring the less significant ones. We would like to determine the transformation matrix $L$ so that the relevance of different variables would be easily assessed in the resulting state vector[2].

[1] What is this "essence" in the data — again, it can be noted that mathematical machinery cannot judge the real relevance of the phenomena. But if we are less ambitious, useful results can be reached

[2] A traditional way to solve this model reduction problem was to concentrate on the most significant modes, those that are nearest to the unit circle, thus being slowest and decaying last, and simply "forget" about the less significant modes altogether. Often this approach works — the slowest decaying modes are visible longest and determine the overall system behavior. However, pole-zero studies miss one crucial point: The gains of the modes cannot

### 10.2.2 Preliminary experiments

For a moment, study the following simplified system model:

$$\left\{ \begin{array}{rcl} x(\kappa+1) & = & Ax(\kappa) \\ y(\kappa) & = & Cx(\kappa). \end{array} \right. \tag{10.5}$$

Assume that $A$ can be diagonalized, so that there exists a matrix of eigenvecors $\Theta$ so that there holds

$$A = \Theta \cdot \Lambda \cdot \Theta^{-1}. \tag{10.6}$$

Defining $x' = \Theta^{-1}x$, the model (10.5) can be written as

$$\left\{ \begin{array}{rcl} x'(\kappa+1) & = & \Lambda x'(\kappa) = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_d \end{pmatrix} \cdot x'(\kappa) \\ y(\kappa) & = & C\Theta \cdot x'(\kappa). \end{array} \right. \tag{10.7}$$

Now it is evident that all modes have been separated — each of the state elements $x_i(\kappa)$ only affects itself; furthermore, the eigenvalues $\lambda_i$ determine their decay rates. However, there are problems. First, the matrix $A$ may not be diagonalizable; what is more, when the general model (10.3) is studied, this diagonalization of the matrix $A$ alone cannot guarantee independence of the modes: It is the matrix $B$ that routes the exogenous input $u(\kappa)$ into the system, and depending on the connections between the state elements as determined by $B$, the modal structure gets blurred, state elements becoming mutually correlated; this approach clearly does not solve the problem.

Of course, one possibility would be to analyze again the state covariance matrix $\frac{1}{k} \cdot \sum_{\kappa=1}^{k} x'(\kappa)x'^T(\kappa) = L \cdot \frac{1}{k} \cdot \sum_{\kappa=1}^{k} x(\kappa)x^T(\kappa) \cdot L^T$ in the same way as in the previous chapters, defining the state transformation matrix $L$ so that the covariance becomes identity matrix. This procedure separates the states — but there are various problems: First, the *output* is not taken care of in the construction of $x'$; second, this kind of transformation is not determined exclusively by the system structure but also by the input signal properties — this kind of dependency of the external conditions is not what is needed.

Something more sophisticated needs to be done. What one would like to have is a state reduction technique that would find a realization that is only dependent of the system structure, and that would be somehow *balanced* between input and output.

---

be seen if only the locations of the transfer function roots are studied. There may be a mode that is located far from the unit circle but having so high gain that — even though decaying fast — its transients start from such high values that they are still most significant. On the other hand, the qualitative pole-zero analyses cannot reveal the role of interactions between the modes: A nearby zero can essentially shadow the effects of some otherwise relevant pole. What is needed is a numerical methodology for reducing the system order

### 10.2.3 Balanced realizations

It turns out to be a good strategy to study how the signal *energy* is transferred through the system; how the *past inputs* affect the *future outputs* through the system state variables [10].

Let us study how the state $x$ at time $\kappa$ relays the signal *energy* from input to output in the model (10.3). First, calculate the contributions of the past inputs on the current state (assuming system stability):

$$x(\kappa) = Bu(\kappa-1) + ABu(\kappa-2) + A^2Bu(\kappa-3) + \cdots$$

This can be expressed as

$$\begin{aligned} x(\kappa) &= \left( \begin{array}{c|c|c|c} B & AB & A^2B & \cdots \end{array} \right) \cdot \begin{pmatrix} u(\kappa-1) \\ u(\kappa-2) \\ \vdots \end{pmatrix} \\ &= M_C \cdot \begin{pmatrix} u(\kappa-1) \\ u(\kappa-2) \\ \vdots \end{pmatrix}, \end{aligned} \tag{10.8}$$

where $M_C$ is the (extended) *controllability matrix.* On the other hand, the effect of the current state $x(\kappa)$ on the future outputs can be written as

$$\begin{cases} y(\kappa) = Cx(\kappa) & \text{Output at time } \kappa \\ y(\kappa+1) = CAx(\kappa) & \text{Output at time } \kappa+1 \\ y(\kappa+2) = CA^2x(\kappa) & \text{Output at time } \kappa+2 \\ \quad \vdots \end{cases} \tag{10.9}$$

This can be written in a compact form using the (extended) *observability matrix* $M_O$:

$$\begin{pmatrix} y(\kappa) \\ y(\kappa+1) \\ y(\kappa+2) \\ \vdots \end{pmatrix} = \begin{pmatrix} C \\ \hline CA \\ \hline CA^2 \\ \hline \vdots \end{pmatrix} \cdot x(\kappa) = M_O \cdot x(\kappa). \tag{10.10}$$

These two expressions (10.8) and (10.10) can now be combined, resulting in the expression for signal that goes from input to output through the single state $x(\kappa)$:

$$\begin{pmatrix} y(\kappa) \\ y(\kappa+1) \\ y(\kappa+2) \\ \vdots \end{pmatrix} = M_O M_C \cdot \begin{pmatrix} u(\kappa-1) \\ u(\kappa-2) \\ u(\kappa-3) \\ \vdots \end{pmatrix}. \tag{10.11}$$

The matrix $H = M_O M_C$ is known as the *Hankel matrix.* For a discrete system, the Hankel matrix has a close connection to the system properties, and it can

be constructed simply from its pulse response: The element on the $i$'th row and $j$'th column in the Hankel matrix is the $i+j-1$'th element of the system pulse response.

The total power of the output (summing the powers of all output signals together) can be expressed as

$$\begin{aligned} & y^T(\kappa)y(\kappa) + y^T(\kappa+1)y(\kappa+1) + y^T(\kappa+2)y(\kappa+2) + \cdots \\ & \quad = \begin{pmatrix} y(\kappa) \\ y(\kappa+1) \\ y(\kappa+2) \\ \vdots \end{pmatrix}^T \begin{pmatrix} y(\kappa) \\ y(\kappa+1) \\ y(\kappa+2) \\ \vdots \end{pmatrix} \\ & \quad = \begin{pmatrix} u(\kappa-1) \\ u(\kappa-2) \\ u(\kappa-3) \\ \vdots \end{pmatrix}^T \cdot M_C^T M_O^T M_O M_C \cdot \begin{pmatrix} u(\kappa-1) \\ u(\kappa-2) \\ u(\kappa-3) \\ \vdots \end{pmatrix}. \end{aligned} \tag{10.12}$$

In the static case, structuring of data was reached through maximization of (co)variance; similarly, it turns out that the power transfer properties can be structured through an optimization procedure: Now the goal is to maximize the power in output when the total input power is fixed (but the power in input may be arbitrarily distributed):

$$\begin{aligned} & u^T(\kappa-1)u(\kappa-1) + u^T(\kappa-2)u(\kappa-2) + \cdots \\ & \quad = \begin{pmatrix} u(\kappa-1) \\ u(\kappa-2) \\ u(\kappa-3) \\ \vdots \end{pmatrix}^T \begin{pmatrix} u(\kappa-1) \\ u(\kappa-2) \\ u(\kappa-3) \\ \vdots \end{pmatrix} \\ & \quad = 1. \end{aligned} \tag{10.13}$$

The criterion to be maximized also becomes

$$\begin{aligned} \text{Maximize} \quad & \begin{pmatrix} u(\kappa-1) \\ u(\kappa-2) \\ u(\kappa-3) \\ \vdots \end{pmatrix}^T \cdot M_C^T M_O^T M_O M_C \cdot \begin{pmatrix} u(\kappa-1) \\ u(\kappa-2) \\ u(\kappa-3) \\ \vdots \end{pmatrix}, \\ \text{when} \quad & \begin{pmatrix} u(\kappa-1) \\ u(\kappa-2) \\ u(\kappa-3) \\ \vdots \end{pmatrix}^T \begin{pmatrix} u(\kappa-1) \\ u(\kappa-2) \\ u(\kappa-3) \\ \vdots \end{pmatrix} = 1. \end{aligned} \tag{10.14}$$

The method of Lagrangian multipliers can again be applied, and it turns out that, again, an eigenproblem is found:

$$M_C^T M_O^T M_O M_C \cdot u_i = \lambda_i \cdot u_i. \tag{10.15}$$

Elements of vector $u_i$ have the same interpretation as the elements in the infinite dimensional input signal vectors above; the problem now is that the infinite

dimensional eigenproblem is slightly questionable! However, there is a nice trick available here: Multiply (10.15) from left by the matrix $M_C$, so that

$$M_C M_C^T M_O^T M_O \cdot M_C u_i = \lambda_i \cdot M_C u_i. \tag{10.16}$$

It turns out that the vector $M_C u_i$ must now be the eigenvector of the finite-dimensional matrix $M_C M_C^T M_O^T M_O$, the eigenvalues remaining the same as in the original eigenproblem. Note that this equality of eigenvalues holds only for the nonzero ones; the higher-dimensional problem of course has high number of additional eigenvalues, but they are all zeros. The matrix $M_C M_C^T \cdot M_O^T M_O$ consists of two low-dimensional parts:

- The *Controllability Gramian* contains only the input-related factors:

$$\begin{aligned} P_C &= M_C M_C^T = \sum_{\kappa=0}^{\infty} A^\kappa B B^T (A^T)^\kappa \\ &= BB^T + ABB^T A^T + A^2 BB^T A^{2T} + \cdots \end{aligned} \tag{10.17}$$

- The *Observability Gramian* contains only the output-related factors:

$$\begin{aligned} P_O &= M_O^T M_O = \sum_{\kappa=0}^{\infty} (A^T)^\kappa C^T C A^\kappa \\ &= C^T C + A^T C^T C A + A^{2T} C^T C A^2 + \cdots \end{aligned} \tag{10.18}$$

It is easy to show that the Gramians satisfy the linear matrix equations

$$\begin{aligned} A P_C A^T - P_C &= -BB^T \\ A^T P_O A - P_O &= -C^T C. \end{aligned} \tag{10.19}$$

Gramians are closely related to the controllability and observability properties of a system: If $P_C$ is positive definite, the system is controllable, and if $P_O$ is positive definite, the system is observable. Compared to the standard controllability and observability matrices, the Gramians offer a more quantitative method to studying the system properties.

If some similarity transform is applied to the system states, so that $A' = LAL^{-1}$, $B' = LB$, and $C' = CL^{-1}$, the Gramians will be modified as

$$\begin{aligned} P'_C &= L P_C L^T \qquad \text{and} \\ P'_O &= L^{-T} P_O L^{-1}. \end{aligned} \tag{10.20}$$

It is also possible to change the Gramians by selecting the state transformation matrix $L$ appropriately. However, it turns out that the product of the Gramians

$$P'_C P'_O = L \cdot P_C P_O \cdot L^{-1} \tag{10.21}$$

is a similarity transtorm of the original Gramian product $P_C P_O$; regardless of the state transformation the eigenvalues of

$$P'_C P'_O \cdot M_C u_i = \lambda_i \cdot M_C u_i \tag{10.22}$$

remain invariant. It is possible to select $L$ so that the new Gramian product $P'_C P'_O$ becomes diagonal by diagonalizing it using eigenvalue decomposition:

$$P'_C P'_O = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_n^2 \end{pmatrix}. \tag{10.23}$$

The parameters $\sigma_i$ are very important system characterizing constants, and they are called *Hankel singular values,* being the singular values of the corresponding Hankel matrix. As was shown, Hankel singular values are invariant in a system under state-space representation. The Hankel singular values also determine, in a way, the "maximum gain" and the "minimum gain" of the system; the largest of the Hankel singular values is called the *Hankel norm* of the system.

The system realization where the state transformation has been selected in this way is said to be in the *balanced* form (the signals should also be appropriately normalized). In the balanced realization each of the state components is independent of the others; what is more, each state component is as well "visible" in the output as it is "excitable" from the input.

### 10.2.4 Eliminating states

The above discussion gives us concrete tools for state reduction: Drop those state components from $x'$ that have the least importance in signal power transfer between input and output; these state components are exposed by the lowest Hankel singular values.

Noticing that $x = L^{-1}x'$, it can be recognized that the most of the input-output mapping is transferred through those states in $x'$ that correspond to those rows of $L^{-1}$ standing for the largest Hankel singular values. If dimension reduction is to be carried out using the balanced realization, the state mapping matrix $\theta^T$ is constructed from these rows of $L^{-1}$. Note that because $P_C P_O$ is not generally symmetric, the eigensystem is not orthogonal; that is why, the reduced system matrices cannot be constructed as $A' = \theta^T A\theta$, etc., but one first has to calculate the full matrices $A' = L^{-1}AL$, $B' = L^{-1}B$, and $C' = CL$, and only after that eliminate the rows and columns that correspond to the eliminated state elements in $z = x'$. The matrix $D$ is not affected in the reduction process.

It needs to be recognized that there are some practical limitations what comes to balanced system truncation. First, the system has to be asymptotically stable (otherwise the Gramians do not remain bounded). Second, again, one fact needs to be kept in mind: Mathematical optimality does not always mean good design[3].

[3]For example, for physical reasons, we may know that the overall system gain should be unity; state truncation in the above way does not assure that this system property is maintained — see Exercises

## 10.3 State estimation

Another example of the surprises that one can attack using "multivariate thinking" is taken from the field of *state estimation:* Given the system structure and the measurement signals, the problem is to determine the system state. Now, it is assumed that the system model has the form of (9.4), also containing stochastic components. Further, assume that only the output $y(\kappa)$ can be measured, and, of course, $u(\kappa)$ is known. The goal is to find out $x(\kappa)$ using only these past system inputs and outputs:

$$\begin{cases} x(\kappa+1) = Ax(\kappa) + Bu(\kappa) + \epsilon(\kappa) \\ y(\kappa) = Cx(\kappa) + Du(\kappa) + e(\kappa). \end{cases} \tag{10.24}$$

The state estimators generally has the (recursive) form

$$\hat{x}(\kappa+1) = A\hat{x}(\kappa) + Bu(\kappa) + K(\kappa) \cdot (y(\kappa) - \hat{y}(\kappa)), \tag{10.25}$$

where

$$\hat{y}(\kappa) = C\hat{x}(\kappa) + Du(\kappa). \tag{10.26}$$

The expression $y(\kappa) - \hat{y}(\kappa)$ represents the error in the model output as compared to the real system output. The state estimate follows the assumed system model, but if there is error in the model output, the estimate is corrected appropriately. Our goal is to determine the matrix $K(\kappa)$ so that the actual state would be recovered as well as possible using the observed system behavior. It is reasonable to define this "goodness" in terms of the state estimation error

$$\tilde{x}(\kappa) = x(\kappa) - \hat{x}(\kappa). \tag{10.27}$$

The goal is now to minimize the covariance matrix $\mathrm{E}\{\tilde{x}(\kappa)\tilde{x}^T(\kappa)\}$. This is accomplished by the so called *Kalman filter.*

### 10.3.1 Kalman filter

The solution to the state estimation problem is based on induction: Assume that $P(\kappa)$ is the minimum error covariance having been found using the measurements that were available before the time instant $\kappa$. The matrix $K(\kappa)$ is now determined so that the covariance at the next time point also is minimal.

Subtract the state estimator, as given by (10.25), from the system state, as defined in (10.24):

$$\begin{aligned} \tilde{x}(\kappa+1) &= x(\kappa+1) - \hat{x}(\kappa+1) \\ &= (A - K(\kappa)C)\,\tilde{x}(\kappa) + \epsilon(\kappa) - K(\kappa)e(\kappa). \end{aligned} \tag{10.28}$$

Multiply both sides by their transposes and take the expectation values — on the left hand side, one has the next step estimation error covariance matrix to

be minimized:

$$\begin{aligned}
&P(\kappa+1)\\
&\quad= \mathrm{E}\{\tilde{x}(\kappa+1)\tilde{x}^T(\kappa+1)\}\\
&\quad= (A-K(\kappa)C)P(\kappa)(A-K(\kappa)C)^T\\
&\qquad + R_{xx} + K(\kappa)R_{xy}^T + R_{xy}K^T(\kappa) + K(\kappa)R_{yy}K^T(\kappa)\\
&\quad= AP(\kappa)A^T + R_{xx}\\
&\qquad - \left(AP(\kappa)C^T + R_{xy}\right)\left(CP(\kappa)C^T + R_{yy}\right)^{-1}\left(AP(\kappa)C^T + R_{xy}\right)^T\\
&\qquad + \Big(K(\kappa) - \left(AP(\kappa)C^T + R_{xy}\right)\left(CP(\kappa)C^T + R_{yy}\right)^{-1}\Big)\cdot\\
&\qquad\quad \left(CP(\kappa)C^T + R_{yy}\right)\cdot\\
&\qquad\qquad \Big(K(\kappa) - \left(AP(\kappa)C^T + R_{xy}\right)\left(CP(\kappa)C^T + R_{yy}\right)^{-1}\Big)^T.
\end{aligned}$$

The last part of the equation above (last three rows) is a quadratic form and the matrix in the middle $CP(\kappa)C^T + R_{yy}$ is positive semidefinite. This means that the minimum for the overall expression is reached if this last part is made zero, or if one selects

$$K(\kappa) = \left(AP(\kappa)C^T + R_{xy}\right)\left(CP(\kappa)C^T + R_{yy}\right)^{-1}. \tag{10.29}$$

In this case the minimum covariance becomes

$$\begin{aligned}
P(\kappa+1) &= AP(\kappa)A^T + R_{xx}\\
&\qquad - \left(AP(\kappa)C^T + R_{xy}\right)\left(CP(\kappa)C^T + R_{yy}\right)^{-1}\cdot\\
&\qquad\quad \left(AP(\kappa)C^T + R_{xy}\right)^T\\
&= AP(\kappa)A^T + R_{xx} - K(\kappa)\cdot\left(AP(\kappa)C^T + R_{xy}\right)^T.
\end{aligned} \tag{10.30}$$

Often in time-invariant environments a constant gain matrix is used instead:

$$\bar{K} = \left(A\bar{P}C^T + R_{xy}\right)\left(C\bar{P}C^T + R_{yy}\right)^{-1}, \tag{10.31}$$

where $\bar{P}$ is the positive semidefinite solution to the Riccati equation

$$\begin{aligned}
\bar{P} &= A\bar{P}A^T + R_{xx}\\
&\qquad - \left(A\bar{P}C^T + R_{xy}\right)\left(C\bar{P}C^T + R_{yy}\right)^{-1}\left(A\bar{P}C^T + R_{xy}\right)^T.
\end{aligned} \tag{10.32}$$

### 10.3.2 Optimality vs. reality

The above solution to the state estimation problem is also optimal. However, let us study what may happen in practice — assume that the system is one-dimensional, with only one input and two outputs as

$$\left\{\begin{array}{l}
x(\kappa+1) = ax(\kappa) + bu(\kappa) + \epsilon(\kappa)\\
\begin{pmatrix} y_1(\kappa)\\ y_2(\kappa)\end{pmatrix} = \begin{pmatrix} 1\\ 1\end{pmatrix}\cdot x(\kappa) + \begin{pmatrix} e_1(\kappa)\\ e_2(\kappa)\end{pmatrix},
\end{array}\right. \tag{10.33}$$

so that essentially the scalar state is measured two times. Intuitively, this should of course enhance the estimate, or, at least, it should not have any catastrophic effects. However, study the resulting steady-state gain matrix:

$$\bar{K} = \left(a\bar{p} \cdot \begin{pmatrix} 1 & 1 \end{pmatrix} + R_{xy}\right) \cdot \left(\bar{p} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} \mathrm{E}\{e_1^2(\kappa)\} & \mathrm{E}\{e_1(\kappa)e_2(\kappa)\} \\ \mathrm{E}\{e_1(\kappa)e_2(\kappa)\} & \mathrm{E}\{e_2^2(\kappa)\} \end{pmatrix}\right)^{-1}, \tag{10.34}$$

so that the properties of the estimator are essentially dictated by the invertibility of the matrix

$$\bar{p} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} \mathrm{E}\{e_1^2(\kappa)\} & \mathrm{E}\{e_1(\kappa)e_2(\kappa)\} \\ \mathrm{E}\{e_1(\kappa)e_2(\kappa)\} & \mathrm{E}\{e_2^2(\kappa)\} \end{pmatrix}. \tag{10.35}$$

Clearly, the first term is singular regardless of the numeric value of the scalar $\bar{p}$ — the whole sum becomes uninvertible, at least, if there holds $e_1(\kappa) = e_2(\kappa)$. If the same variable is measured, most probably the different measurements are correlated — if the measurements happen to be exactly identical, the whole estimator explodes, and even if they are not, the matrix may still become arbitrarily badly conditioned.

Consider some sensor fusion tasks, for example: The Kalman filter is often used for combining more or less reliable measurements, and often the number of measurements is very high — for example, take the weather models, where thousands of measurements are used to gain information about atmospheric phenomena. Blindly trusting the Kalman filter is dangerous: Even though it is optimal it may sometimes work against intuition[4].

### 10.3.3 Reducing the number of measurements

The above uninvertibility problem was caused again by the collinearity of the measurements. It is not a surprise that the multivariate analysis techniques turn out to offer valuable tools for attacking this kind of problems. So, assume that we want to reduce the output dimension so that redundancies are eliminated. We search for the directions where the measurements are most informative as determined by the symmetric, positive semidefinite matrix

$$C\bar{P}C^T + R_{yy}. \tag{10.36}$$

What is "informative" is again a matter of taste; if the PCA type approach is chosen, the task is to find the eigenvectors corresponding to the largest eigenvalues in

$$\left(C\bar{P}C^T + R_{yy}\right) \cdot \theta_i = \lambda_i \cdot \theta_i. \tag{10.37}$$

Assume that the dimension is reduced by, say, the PCA technique. The reduced basis is assumed to be $\theta$ and the corresponding eigenvalues are on the diagonal of $\Lambda$; then one can write

$$C\bar{P}C^T + R_{yy} \approx \theta \cdot \Lambda \cdot \theta^T. \tag{10.38}$$

[4]In practice, there is no exact information about the noise properties, and to avoid problems, the covariance matrices are usually assumed diagonal ... but the optimality of the estimator is of course ruined when the system model is incorrect!

Now $\Lambda$ is low-dimensional and well-conditioned, so that its inverse is easily calculated; approximately there holds

$$\left(C\bar{P}C^T + R_{yy}\right)^{-1} \approx \theta \cdot \Lambda^{-1} \cdot \theta^T. \tag{10.39}$$

Substituting this in (10.31) gives

$$\bar{K} \approx \left(A\bar{P}C^T + R_{xy}\right) \cdot \theta\Lambda^{-1}\theta^T \left(A\bar{P}C^T\theta + R_{xy}\theta\right) \Lambda^{-1}\theta^T, \tag{10.40}$$

so that the estimator becomes

$$\begin{aligned}&\hat{x}(\kappa+1) = \\ &\quad A\hat{x}(\kappa) + Bu(\kappa) + \left(A\bar{P}C^T\theta + R_{xy}\theta\right) \Lambda^{-1}\theta^T \cdot \left(y(\kappa) - C\hat{x}(\kappa) - Du(\kappa)\right).\end{aligned}$$

This formulation efficiently helps to avoid anomalies caused by the measurement redundancy.

## 10.4 SISO identification

Finally, yet another systems engineering application field is studied where the multivariate problem setting becomes relevant. We will study the prediction error methods for black-box identification (see [29], [26]). The traditional approaches that are still the mainstream technology (for example, see the `System Identification Toolbox` for `Matlab`) suffer from the problems that have been demonstrated in previous chapters, and analogous solutions to the problems can be proposed. Note that for practical parameter estimation purposes in dynamic systems, subspace identification (as explained in Chapter 9) is recommended.

### 10.4.1 Black-box model

The behavior of a linear, strictly proper, $d$'th order discrete time system can be expressed as a difference equation

$$\begin{aligned}y(\kappa) = a_1 y(\kappa-1) + \cdots + a_d y(\kappa-d) \\ + b_1 u(\kappa-1) + \cdots + b_d u(\kappa-d),\end{aligned} \tag{10.41}$$

where $u(\kappa)$ denotes the (centered) scalar process input and $y(\kappa)$ the scalar output at time instant $\kappa$. Using vector formulation, this can be written (following the earlier notations) as

$$y(\kappa) = x^T(\kappa) \cdot f, \tag{10.42}$$

where

$$x(\kappa) = \begin{pmatrix} y(\kappa-1) \\ \vdots \\ y(\kappa-d) \\ u(\kappa-1) \\ \vdots \\ u(\kappa-d) \end{pmatrix} \quad \text{and} \quad f = \begin{pmatrix} a_1 \\ \vdots \\ a_d \\ b_1 \\ \vdots \\ b_d \end{pmatrix}. \tag{10.43}$$

This means that the dynamic nature of the process has been transformed into a representation where virtually static time series samples are used; the dynamic complexity has been changed to dimensional complexity.

The structure of the linear dynamic system is assumed to be extremely simple, consisting of one input and one output signals; further, it is assumed that the dynamic dimension of the system is exactly known. If it is still assumed that the signals are persistently exciting, and no unmodeled noise is present, identifying the parameters of the model should be a trivial task. This is what standard theory says; however, in practice, problems often emerge. These problems can again be studied in the framework of statistical data analysis.

### 10.4.2 Recursive least-squares algorithm

The parameter vector $f$ can be solved off-line, as a batch for some set of data; in this way, the methods presented in earlier chapters can directly be utilized (in practice, it seems to be customary to stick to the basic MLR or its derivations). However, in many cases measurements are obtained one at a time, and it is reasonable to rearrange the calculations so that the computational load would be minimized. To derive the *on-line recursive identification algorithm,* define the exponentially weighted cost criterion as

$$J(k) = \sum_{\kappa=0}^{k} \lambda^{k-\kappa} \cdot e^2(\kappa), \tag{10.44}$$

where the prediction error is defined as

$$e(\kappa) = y(\kappa) - x^T(\kappa) f. \tag{10.45}$$

The exponential weighting emphasizes the newest measurements, that is, if the *forgetting factor* $\lambda$ has value less than one, old measurements are gradually forgotten. Note that the so called ARX system structure is chosen, again essentially assuming that the error is summed only to the output; otherwise the noise becomes *colored* and algorithms give biased estimates. Minimizing the criterion can be carried out as follows:

$$\begin{aligned} \tfrac{d\,J(k)}{d\,f} &= -2 \cdot \textstyle\sum_{\kappa=0}^{k} \lambda^{k-\kappa} \cdot x(\kappa) \cdot \big(y(\kappa) - x^T(\kappa) f\big) \\ &= -2 \cdot \textstyle\sum_{\kappa=0}^{k} \lambda^{k-\kappa} x(\kappa) \cdot y(\kappa) + 2 \cdot \sum_{\kappa=0}^{k} \lambda^{k-\kappa} x(\kappa) x^T(\kappa) \cdot f \\ &= \mathbf{0}, \end{aligned}$$

or

$$\sum_{\kappa=0}^{k} \lambda^{k-\kappa} x(\kappa) x^T(\kappa) \cdot f = \sum_{\kappa=0}^{k} \lambda^{k-\kappa} x(\kappa) \cdot y(\kappa), \tag{10.46}$$

so that the parameter estimate can be solved as

$$\hat{f} = \left( \sum_{\kappa=0}^{k} \lambda^{k-\kappa} x(\kappa) x^T(\kappa) \right)^{-1} \cdot \sum_{\kappa=0}^{k} \lambda^{k-\kappa} x(\kappa) \cdot y(\kappa). \tag{10.47}$$

However, this is not yet in the recursive form, so that the new parameter estimate $\hat{f}(k)$ would be received from the old estimate $\hat{f}(k-1)$ by updating it using some fresh information. To reach such a formulation, define

$$\begin{aligned} R_{xx}(k) &= \sum_{\kappa=0}^{k} \lambda^{k-\kappa} x(\kappa) x^T(\kappa) \\ &= x(k)x^T(k) + \lambda \cdot \sum_{\kappa=0}^{k-1} \lambda^{k-\kappa} x(\kappa) x^T(\kappa) \\ &= x(k)x^T(k) + \lambda \cdot R_{xx}(k-1) \end{aligned} \tag{10.48}$$

and

$$\begin{aligned} R_{xy}(k) &= \sum_{\kappa=0}^{k} \lambda^{k-\kappa} x(\kappa) y(\kappa) \\ &= x(k)y(k) + \lambda \cdot \sum_{\kappa=0}^{k-1} \lambda^{k-\kappa} x(\kappa) y(\kappa) \\ &= x(k)y(k) + \lambda \cdot R_{xy}(k-1). \end{aligned} \tag{10.49}$$

Formula (10.46) can be expressed using these matrices as $R_{xx}(k) \cdot \hat{f}(k) = R_{xy}(k)$, so that the new parameter estimate can be written as

$$\begin{aligned} \hat{f}(k) &= R_{xx}^{-1}(k) \cdot R_{xy}(k) \\ &= R_{xx}^{-1}(k) \cdot (x(k)y(k) + \lambda \cdot R_{xy}(k-1)) \\ &= R_{xx}^{-1}(k) \cdot \Big( x(k)y(k) + \lambda \cdot R_{xx}(k-1) \cdot \hat{f}(k-1) \Big) \\ &= R_{xx}^{-1}(k) \cdot \Big( x(k)y(k) + \big( R_{xx}(k) - x(k)x^T(k) \big) \cdot \hat{f}(k-1) \Big) \\ &= \hat{f}(k-1) + R_{xx}^{-1}(k) \cdot \Big( x(k)y(k) - x(k)x^T(k) \cdot \hat{f}(k-1) \Big) \\ &= \hat{f}(k-1) + R_{xx}^{-1}(k) \cdot x(k) \cdot \Big( y(k) - x^T(k) \cdot \hat{f}(k-1) \Big). \end{aligned}$$

Rewriting this and collecting the results together (and defining $R = R_{xx}$), the final Gauss-Newton type identification algorithm becomes

$$\begin{aligned} \hat{f}(k) &= \hat{f}(k-1) + R^{-1}(k) x(k) \cdot \Big( y(k) - x^T(k) \hat{f}(k-1) \Big) \\ R(k) &= \lambda R(k-1) + x(k)x^T(k). \end{aligned} \tag{10.50}$$

The *matrix inversion lemma* could be applied here to make the algorithm more efficient in practice; however, in this context overall comprehensibility of the algorithm is preferred. On the first line, the parameter estimate vector is updated; the size of the update step is determined by the prediction error, whereas the update direction is determined by the matrix $R(k)$. What is this matrix, then — this can be seen if one studies the expectation values:

$$E\{R(k)\} = \lambda \cdot E\{R(k-1)\} + E\{x(k)x^T(k)\}, \tag{10.51}$$

where $E\{R(k)\} = E\{R(k-1)\} = E\{R\}$, so that

$$E\{R\} = \frac{1}{1-\lambda} \cdot E\left\{ xx^T \right\}. \tag{10.52}$$

This means that $R$ is the (scaled) data covariance matrix estimate — sounds familiar ...!

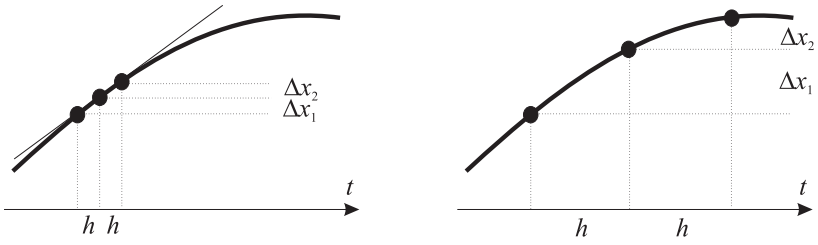



Figure 10.1: A figure illustrating the fact that short sampling intervals (shown on the left) make the successive samples mutually dependent: They have almost equal values, or, at least, assuming smooth signal behavior, they are almost on the same line, thus being collinear

### 10.4.3 Structure of dynamic data

The data $x(\kappa)$ was constructed from successive signal measurements. However, there are peculiar dependencies between the delayed signals that are not taken into account by the standard SISO identification algorithms. The problems are (as in the MLR case) concentrated on the invertibility of the data covariance matrix $R$.

The successive samples are most probably highly correlated because they should represent continuous dynamic evolution. This redundancy between the signal samples is the main reason for the structural identifiability problems — the data is collinear, and this linear dependency between variables becomes more and more dominating when the sampling interval is made smaller (see Fig. 10.1).

The numerical problems inherent in the data are emphasized by the recursive "forgetting" of the algorithms: Older information gets ignored as time evolves, and it may be that the numerical properties of the data covariance matrix are gradually ruined, the identification process becoming badly behaving.

Because of the relevance of the robustness issues in practical approaches, various more or less heuristic approaches have been proposed, including different kinds of *constant trace, regularization,* or *variable forgetting* algorithms (see [19] and [21], for example). A close relative of Ridge Regression (as written in recursive form) is the so called *Levenberg-Marquardt* identification algorithm that keeps $R$ invertible by adding a minor positive definite factor to it during each step:

$$\begin{aligned} \hat{f}(k) &= \hat{f}(k-1) + R^{-1}(k)x(k) \cdot \Big(y(k) - x^T(k)\hat{f}(k-1)\Big) \\ R(k) &= \lambda R(k-1) + x(k)x^T(k) + q \cdot I_{2d}. \end{aligned} \tag{10.53}$$

It can be shown that as $q$ varies from zero to higher values, the continuum from Gauss-Newton and simple gradient method is spanned.

Another family of identification methods (or, actually, data preprocessing methods) is found when the system *parameterizations* are studied. The dynamic nature of a system can be captured in an infinite number of ways; even though the above time series approach is rather natural, it is only one alternative — and not a very good choice, as it has turned out. As an example, study Fig. 10.1:

Using the traditional shift operator $q$ formalism (or, equivalently, using delay operators $q^{-1}$) all systems finally become integrators as the sampling period $h$ goes towards zero! A very simple alternative to the standard parameterization is the $\delta$ parameter formalism [12]; the idea is that rather than taking the measurements themselves as a basis, the *differentiated* signals are used: Differences between successive samples are analyzed rather than the original signal values. It has been shown that this parameterization enhances the numerical properties of many algorithms (not only identification algorithms). More sophisticated parameterizations are studied, for example, in [9] and in [20][5].

### 10.4.4 Further analysis: System identifiability*

The properties of the data covariance matrix are, of course, determined by the data properties themselves — but not exclusively. As was seen above, the system dynamics dictates what is the relation between successive measurements, and this is reflected also in the data covariance. Now study the *structural identifiability properties* of a dynamic system. This analysis was carried out originally in [22].

Differentiating (10.42) with respect to $f$, one has

$$\frac{d\,y}{d\,f}(\kappa) = x(\kappa), \tag{10.54}$$

so that the data covariance matrix can be written as

$$\mathrm{E}\left\{x(\kappa)x^T(\kappa)\right\} = \mathrm{E}\left\{\left(\frac{d\,y}{d\,f}(\kappa)\right)\left(\frac{d\,y}{d\,f}(\kappa)\right)^T\right\}. \tag{10.55}$$

What are these signal derivatives? One can differentiate the response $y(\kappa)$ in (10.41) with respect to all the parameters, so that a set of difference equations is found:

$$\left\{\begin{array}{rcl} \dfrac{\partial y}{\partial a_i}(\kappa) & = & \displaystyle\sum_{j=1}^{d} a_j q^{-j} \cdot \frac{\partial y}{\partial a_i}(\kappa) + q^{-i} \cdot y(\kappa) \\ \dfrac{\partial y}{\partial b_i}(\kappa) & = & \displaystyle\sum_{j=1}^{d} a_j q^{-j} \cdot \frac{\partial y}{\partial b_i}(\kappa) + q^{-i} \cdot u(\kappa). \end{array}\right. \tag{10.56}$$

Neglecting the initial conditions, these $2d$ difference equations of order $d$ can be written in a $2d$-dimensional state space form

$$x'(\kappa+1) = A'x'(\kappa) + B'u(\kappa) \tag{10.57}$$

[5] In the multivariate framework, one straightforward approach to enhancing the matrix invertibility properties would be to reduce the dimension of $R$, just as has been done so many times this far. However, now it cannot be assumed that the properties of data remain invariant — the original reason to use recursive algorithms was to be able to react to changing system properties — and the matrix $R$ does not remain constant: The eigenvalue decomposition should be computed, in principle, during each step, and the computational burden would become excessive

defined by the matrices

$$A' = \left(\begin{array}{cccc|cccc} a_1 & a_2 & \cdots & a_d & & & & \\ 1 & & & & & & 0 & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ \hline b_1 & b_2 & \cdots & b_d & a_1 & a_2 & \cdots & a_d \\ & & & & 1 & & & \\ & & & & & \ddots & & \\ & & & & & & 1 & \end{array}\right) \quad \text{and} \quad B' = \left(\begin{array}{c} 1 \\ 0 \\ \vdots \\ 0 \\ \hline 0 \\ 0 \\ \vdots \\ 0 \end{array}\right).$$

This state-space system will here be called the "sensitivity system" corresponding to the model (10.41). The control signal in this system is the original input $u(k)$, but the state vector is

$$x'(\kappa) = \left(\begin{array}{c} \frac{\partial y}{\partial b_1}(\kappa) \\ \vdots \\ \frac{\partial y}{\partial b_d}(\kappa) \\ \hline \frac{\partial y}{\partial a_1}(\kappa) \\ \vdots \\ \frac{\partial y}{\partial a_d}(\kappa) \end{array}\right) = \frac{d\,y}{d\,f}(\kappa). \tag{10.58}$$

The motivation for these definitions is that the covariance structures for the data vector $x$ and the state vector $x'$ must be identical; and the behavior of the dynamic system state can be easily analyzed. The data covariance matrix (10.55) can also be written as

$$\begin{aligned} &\mathrm{E}\left\{\left(\frac{dy}{df}(\kappa)\right)\left(\frac{dy}{df}(\kappa)\right)^T\right\} \\ &\quad = \mathrm{E}\Big\{(B'u(\kappa-1) + A'B'u(\kappa-2) + \cdots)\cdot \\ &\qquad\qquad (B'u(\kappa-1) + A'B'u(\kappa-2) + \cdots)^T\Big\} \\ &\quad = r_u(0)\cdot\left(B'B'^T + A'B'B'^TA'^T + A'^2B'B'^TA'^{2T} + \cdots\right) \\ &\qquad + r_u(1)\cdot\left(A'B'B'^T + B'B'^TA'^T + \cdots\right) \\ &\qquad\quad + r_u(2)\cdot\left(A'^2B'B'^T + B'B'^TA'^{2T} + \cdots\right) \\ &\qquad\qquad + \cdots \\ &\quad = r_u(0)\cdot M'_C \\ &\qquad + r_u(1)\cdot\left(A'M'_C + M'_CA'^T\right) \\ &\qquad\quad + r_u(2)\cdot\left(A'^2M'_C + M'_CA'^{2T}\right) \\ &\qquad\qquad + \cdots \end{aligned} \tag{10.59}$$

It turns out that the matrix $M'_C$ equals the Controllability Gramian for the sensitivity system; additionally, the input signal autocorrelation function values

are involved here:

$$\begin{aligned}
r_u(0) &= \mathrm{E}\{u^2(\kappa-1)\} = \mathrm{E}\{u^2(\kappa-2)\} = \cdots \\
r_u(1) &= \mathrm{E}\{u(\kappa-1)u(\kappa-2)\} = \mathrm{E}\{u(\kappa-2)u(\kappa-3)\} = \cdots \\
&= \mathrm{E}\{u(\kappa-2)u(\kappa-1)\} = \mathrm{E}\{u(\kappa-3)u(\kappa-2)\} = \cdots \\
r_u(2) &= \mathrm{E}\{u(\kappa-1)u(\kappa-3)\} = \mathrm{E}\{u(\kappa-2)u(\kappa-4)\} = \cdots \\
&= \mathrm{E}\{u(\kappa-3)u(\kappa-1)\} = \mathrm{E}\{u(\kappa-4)u(\kappa-2)\} = \cdots \\
&\vdots
\end{aligned}$$

This gives us a possibility of estimating the "efficiency" of the input signal what comes to its capability of helping in the parameter identification. On the other hand, optimization of the input can also be carried out: One can determine the autocorrelation function values so that (10.59) becomes easily invertible, and after that construct realizations of such a signal applying spectral factorization. However, note that there are physical constraints what comes to the autocorrelation function — for example, $r_u(0)$ must always be the largest of all $r_u(i)$ for the signal to be realizable.

More anqalysis is needed here: Formal identifiability differs from actual identifiability. To avoid the problems that are inherent to traditional model structures, new model structures need to be introduced. Such efforts are illustrated in the last chapter.

## Computer exercises

1. Study the power of the dimension reduction technique; run the following command sequence various times. Select (interactively) the number of states in different ways — what kind of non-physicalities are seen in the approximations?

   ```
   d = 10;
   A = randn(d,d); A = A/(1.1*norm(A));
   B = randn(d,1);
   C = randn(1,d);
   regrBal(A,B,C);
   ```

   Construct a discrete-time system for implementing a *pure delay* of $d$ time steps, and try to reduce the model:

   ```
   d = 10;
   A = zeros(d,d); A(2:d,1:d-1) = eye(d-1);
   B = zeros(d,1); B(1,1) = 1;
   C = zeros(1,d); C(1,d) = 1;
   [Ared,Bred,Cred] = regrBal(A,B,C);
   ```

   What is the problem? In *this* special case, that specific problem can be circumvented by the following modifications without altering the input/output behavior. However, what can you say about all possible "optimal" model reductions now?

   ```
   A = 0.9*A;
   B = B/0.9^d;
   ```

2. Study the robustness of recursive identification: Check how much the behavior of the parameter estimates changes as the underlying system structure varies by running the following commands various times. Also try the effects of the forgetting factor $\lambda$.

   ```
   lambda = 0.99;
   u = randn(100,1);
   [u,y] = dataDyn(3,u,1);
   regrIdent(u,y,3,lambda);
   ```

# Lesson 11

# Conclusion:* About "Emergent Models"

Statistical methods seem to be efficient tools for data analysis. But will these methods always be inferior to first-principles models — are they only describing *surface-level* reflections of internal phenomena, can they ever capture the true *essence* of systems?

What is this "essence", then? Modeling is about hiding details and concentrating information, one has to abstract away irrelevant details. Again, when determining what *is* irrelevant, one is facing ontological assumptions. We already know how to model simple systems, but when studying *complex systems,* new ways of thinking are needed.

This final chapter tries to illustrate the possibilities that may someday come true. It is shown here how the multivariate statistical methods can perhaps offer new conceptual tools for mastering the complexity in systems. It is the *differences that make a difference:* If there exist phenomena that cannot be seen in observations, they can be ignored. And it is the multivariate methods that can capture such phenomena — if the way of looking at systems is adjusted in an approprite way. The traditional methods only capture narrow projections of the behavioral wealth, whereas the multivariate methods can give a more *holistic* view. This view is presented in closer detail in [**?**]; here, only excerpts from there are reviewed.

## 11.1 Capturing semantics in data

To make it possible to apply multivariate methods for capturing the system essence, the data needs to be defined so that the phenomena of relevance are represented there. The key question is: How to capture the essential information, or *domain-area semantics* in the data? To define data so that the important features are available there for further modeling, one needs a concrete application area. Here, the application area throughout this chapter is the *realm of chemical systems.*

### 11.1.1 What is "semantics"?

The model should be an interface between the system and outside world, providing best possible information transfer. The model structure should be a compromise between the properties of the system and the properties of the applications. What are the model structures like that support the new tools and new ways of thinking, simultaneously taking into account the system itself?

When searching for *good models,* philosophical questions cannot be avoided: It is such modeling issues that have been studied for millennia — what is the nature of systems, and and how they should be represented. Indeed, what there is, what one can we know about them, these problem fields are called *ontology* and *epistemology,* respectively. Earlier in this report, ontological questions have been discussed in simple terms — now these discussions need to be extended slightly. Here all these mutually related issues are collected under the common concept of *semantics:* What is the essence of a system, and how this essence should be interpreted?

Semantics conveys *meaning.* Traditionally, it is thought that semantics cannot exist outside human brain. However, to reach "smart models" that can adapt in new environments, one needs to make this meaning machine-readable and machine-understandable. Otherwise, no abstraction of relevant vs. irrelevant phenomena can be automatically carried out. Indeed, one is facing a huge challenge here, but something *can* be done.

Just as was done earlier when ontologies were studied, now this semantics is formalized: This very abstract concept is given here very concrete contents, compromizing between intuitions (what would be nice) and reality (what can be implemented in reality). It can even be said that a *good model formalizes the semantics of the domain field, making it visible.* Now there are two levels of semantics to be captured:

1. **Low-level semantics.** The formless complexity of the underlying system has to be captured in concrete homogeneous data. The "atoms" of semantics constitute the connection between the numeric representations and the physical realm, so that the properties of the system are appropriately coded and made visible to the higher-level machineries. In concrete terms, one has to define "probes" and put them in the system appropriately. The measurements delivered by the probes still need to be interpreted, or features need to be extracted from the measurements by applying appropriate data preprocessing.

2. **Higher-level semantics.** The high number of structureless low-level features have to be connected into *structures* of semantic atoms. Assuming that the semantic atoms are available, this higher-level task is *simpler,* being more generic. In our numbers-based environments, a practical approach towards such *contextual semantics,* where relevant lower-level structures are to be appropriately combined, is again offered by correlations-based measures. As has been shown before, assuming that information is conveyed in co-variations among data, structuring of lower-level data can be implemented by the mathematical machinery without need of outside expert guidance.

Indeed, analyses of this higher-level semantics processing have been carried out already a lot in this report, and they can be implemented implicitly by the presented multivariate statistical tools. But representation of the low-level domain-area features is domain-area specific, and needs to be studied separately in each case. To have a solid grounding, one somehow needs to limit the overwhelming diversity of available measurements by applying some assumptions concerning the nature of systems being studied.

### 11.1.2 Neocybernetic starting points

The traditional models need to be explicitly controlled by the domain area expert, and the structure needs to be determined before the machinery (identification algorithms, etc.) take over. When modeling complex systems, the structure is hidden, it is not known beforehand. The objective is *automatic abstraction,* letting the structures automatically emerge. And the statistical tools naturally carry out abstraction: Individual observations are not assumed to be significant, only phenomena that remain consistent over the long-term observation periods.

To use statistical methods in a plausible way, the observations need to have statistical relevance. To reach this, the observations need to be *stationary,* that is, there need to exist some consistent statistical structure in the data. To make this possible, to be able to collect stationary data from a complex process, there has to be *balance,* at least as seen in the wider scale.

To find general ways of modeling, something has to be assumed. It turns out that such a rigid enough structural modeling framework where there is possibility of individual structures to emerge is that of *neocybernetics:* One assumes dynamic balance in the system where the internal interactions and feedbacks implement tensions that maintain the system integrity. One can forget the underlying interaction structures if they are just capable of providing appropriate stabilizing internal controls.

In the neocybernetic framework, one does not study all mathematically possible systems — only the physically reasonable ones that are in balance with their environment. Natural systems typically fulfill this assumption, and one would like the industrial systems to fulfill this assumption. What is more, good controls, however they are implemented, keep the system near its setpoint, regardless of the environmental disturbances: This means that linearity of the models can reasonably be assumed.

So, to apply multivariate methods, one has to concentrate on such (thermo)dynamic balances. The data needs to be selected so that is reflects this framework to make it possible to later determine appropriate models. As it turns out, the domain of *chemical systems* offers a compact framework for such studies.

### 11.1.3 Modeling chemical systems

Study a hypothetical example reaction, where there are $\alpha$ reactants on the left hand side, being denoted as $\mathrm{A}_i$, $1 \leq i \leq \alpha$, and the $\beta$ products on the right

hand side are $\mathrm{B}_j$, $1 \leq j \leq \beta$:

$$a_1\mathrm{A}_1 + \cdots + a_\alpha\mathrm{A}_\alpha \quad \underset{k_\mathrm{A}}{\overset{k_\mathrm{B}}{\Leftrightarrow}} \quad b_1\mathrm{B}_1 + \cdots + b_\beta\mathrm{B}_\beta, \quad \Delta H. \tag{11.1}$$

Processes are typically reversible, so that the reaction can take place in both directions ($k_\mathrm{B}$ being the reaction speed in forward and $k_\mathrm{A}$ in backward direction). Symbol $\Delta H$ denotes the change in enthalpy, or inner energy, when the reaction takes place.

One needs a mathematically more compact representation for chemical reactions. How to "cybernetize" chemical reaction models applying the neocybernetic principles?

### Information representation

The first problem is to represent such a chemical reaction formula in a practical numeric form. It seems that a practical way to code the reactions in a mathematically applicable form is to employ the vector formulation: Define a vector $C$ containing all chemical concentrations so that all $\mathrm{A}_i$ and $\mathrm{B}_j$ are represented there among the elements. The "chemical state" can assumedly be captured in this vector, and individual reactions determine equations in that chemical space: If the coefficients $-a_i$ and $b_j$ from (11.1) corresponding to the chemicals are collected in the vector $G$, one can express the total concentration changes in the system as

$$\Delta C = G\,\zeta. \tag{11.2}$$

Here, $\zeta$ is a scalar that reveals "how much" (and in which direction) that reaction has proceeded. When there are many simultaneous reactions taking place, there are various vectors $G_i$; the weighted sum of reaction vectors $\zeta_i G_i$ reveals the total changes in chemical contents, the weighting factors being collected in the vector $\zeta$.

Using the above framework, metabolic systems can in principle be modeled: If one knows the rates of reactions, or the scalars $\zeta_i$, the changes in the chemical contents can be determined. This idea of *invariances* within a chemical system have been widely applied for metabolic modeling; the key term here is *flux balance analysis (FBA)* (for example, see [**?**]). However, the rates $x$ are not known beforehand, and, what is more, the reactions are typically not exactly known.

In many ways, the model structure (11.2) is not yet what one is looking for. The main problem there is that the flux balances only capture the *stoichiometric,* more or less *formal balance* among chemicals. It does not capture the *dynamic balance,* whether or not the reactions actually take place or not. Luckily, there exist also other ways to represent the chemical realm.

### Thermodynamic balance

There is a big difference between what is *possible* and what is *probable,* that is, even though something may happen in principle, it will not actually happen. To

understand the dynamic balance, the reaction mechanisms need to be studied closer.

Assume that it takes $a_1$ molecules of $\mathrm{A}_1$, $a_2$ molecules of $\mathrm{A}_2$, etc., according to (11.1), for one unit reaction to take place. This means that all these molecules have to be located sufficiently near to each other at some time instant for the forward reaction to take place. The probability for one molecule to be within the required range is proportional to the number of such molecules in a volume unit; this molecular density is revealed by consentration (when the unit is mole/liter; by definition one mole always contains $6.022 \cdot 10^{23}$ particles). Assuming that the locations of the molecules are independent of each other, the probability for several of them being found within the range is proportional to the product of their concentrations. On the other hand, the reverse reaction probability is proportional to the concentrations of the right-hand-side molecules. Collected together, the rate of change for the concentration of the chemical $\mathrm{A}_1$, for example, can be expressed as a difference between the backward reaction and forward reaction rates:

$$\frac{d\,C_{\mathrm{A}_1}}{d\,t} = -k_{\mathrm{B}} C_{\mathrm{A}_1}^{a_1} \cdots C_{\mathrm{A}_\alpha}^{a_\alpha} + k_{\mathrm{A}} C_{\mathrm{B}_1}^{b_1} \cdots C_{\mathrm{B}_\beta}^{b_\beta}. \tag{11.3}$$

In equilibrium state there holds $\frac{d\,C_{\mathrm{A}_1}}{d\,t} = 0$, etc., and one can define the constant characterizing the thermodynamic equilibrium (for example, see [**?**]):

$$K = \frac{k_{\mathrm{B}}}{k_{\mathrm{A}}} = \frac{C_{\mathrm{B}_1}^{b_1} \cdots C_{\mathrm{B}_\beta}^{b_\beta}}{C_{\mathrm{A}_1}^{a_1} \cdots C_{\mathrm{A}_\alpha}^{a_\alpha}}. \tag{11.4}$$

### Linearity objective

One of the neocybernetic objectives is that of linearity. Clearly, the expression (11.4) is far from being linear — indeed, it is purely multiplicative. It turns out that applying a purely syntactic trick, linearity of the structures can be reached: Taking logarithms on both sides there holds

$$\log K' = b_1 \log C_{\mathrm{B}_1} + \cdots + b_\beta \log C_{\mathrm{B}_\beta} - a_1 \log C_{\mathrm{A}_1} + \cdots - a_\alpha \log C_{\mathrm{A}_\alpha}. \tag{11.5}$$

To get rid of constants and logarithms, it is also possible to differentiate the expression:

$$0 = b_1 \frac{\Delta C_{\mathrm{B}1}}{\bar{C}_{\mathrm{B}_1}} + \cdots + b_\beta \frac{\Delta C_{\mathrm{B}_\beta}}{\bar{C}_{\mathrm{B}_\beta}} - a_1 \frac{\Delta C_{\mathrm{A}_1}}{\bar{C}_{\mathrm{A}_1}} + \cdots - a_\alpha \frac{\Delta C_{\mathrm{A}_\alpha}}{\bar{C}_{\mathrm{A}_\alpha}}, \tag{11.6}$$

where the variables $\Delta C_i / \bar{C}_i$ are deviations from the nominal values, divided by those nominal values, meaning that it is *relative changes* that are of interest. The differentiated model is only locally applicable, valid in the vicinity of the nominal value.

### Multivariate representation

A single reaction formula can also be expressed in a linear form when the variables are appropriately selected. However, to model complex systems consisting

of various reactions, the data representation needs to be extended: The differing data vectors containing different sets of variables (the reactions employing different chemicals) have to be embedded in the same vector space to make them compatible.

Assume that the vector $v$ is a vector containing all relevant variables capturing the state of the environment and the system itself, including, for example, relative changes in all chemical concentrations. This means that the vector $\Gamma_i$ representing a single reaction can contain various zeros, assuming that the corresponding chemicals are not contributing in the reaction $i$. If the vectors $\Gamma_i$ are collected as columns in the matrix $\Gamma$, one can write the individual expressions in (11.6) in the matrix form where one row is allocated to each of the reactions:

$$0 = \Gamma^T \, v. \tag{11.7}$$

This expression needs to be compared to flux balance analysis: Now one only needs to study levels of concentrations, not changes in them. This is indeed essential in complex chemical systems, where the energy and matter flows cannot be exactly managed. The key point to observe here is that analysis of complicated reaction networks can be avoided: No matter what has caused the observed chemical levels, only the prevailing tensions in the system are of essence. The underlying assumption is that the system is robust and redundant: Individual pathways are of no special importance as there exist various alternative routes in the network.

It turns out that reactions can in principle be characterized applying linear algebra in the space of chemical concentrations, being compatible with the multivariate methods. However, the results still need to be interpreted appropriately. Nothing mathematically very special is being done — as there seldom is in the field of linear theory! — but when seen from the appropriate point of view, new conceptual tools for modeling of complex systems can be available.

## 11.2 From constraints to degrees of freedom

As shown above, the domain-area information can be captured in data. However, this representation feels somewhat hollow, and it is difficult to believe that domain-area *knowledge* could ever be captured this way. However, it can be claimed that *freedoms-oriented* way of modeling is just as natural as the *constraints-oriented* approach is. To understand the meaning of this claim, closer analyses are needed.

### 11.2.1 Constraint-based models

Traditional models are typically based on *constraints.* This means that system properties are captured by formulas of the general form

$$0 = f(v), \tag{11.8}$$

where $f$ is some scalar or vector-valued function of the variable vector $v$. For example, the chemical model in (11.7) is a special (linear) case consisting of

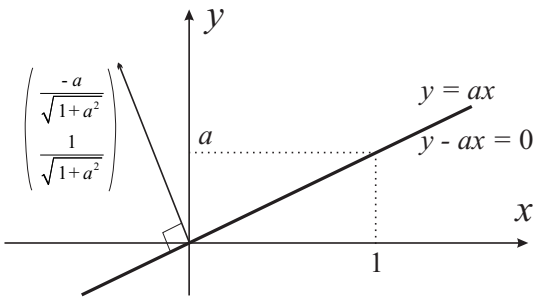



Figure 11.1: How the parameter vector represents the normal vector

various independent equations or constraints in a matrix form. What is more, the linear multivariate models that have been studied in previous chapters, or the models of the form $y = F^T x$, can be written as $0 = F^T x - y$, so that when one defines

$$\Gamma = \left( \frac{F}{-I} \right), \qquad \text{and} \qquad v = \left( \frac{x}{y} \right), \tag{11.9}$$

this is again of the form (11.7), and simultaneously a special case of (**??**). Note that such models are not unique — the vectors $\Gamma_i$ can be freely scaled without affecting the validity of the equations. So, to make such a presentation less ambiguous, from now on assume that the vectors in $\Gamma$ are normalized to unit length, so that $\Gamma_i^T \Gamma_i = 1$.

To better understand the structure of models that are presented in such constraints-oriented form, study a single-output case, so that $y_i$ is scalar, and $\Gamma_i$ is a vector. Whereas $y_i = F_i^T x$ defines a *one-dimensional null-space* in the high-dimensional variable space of $v$, and because the inner product $\Gamma_i^T v$ between the data and the vector $\Gamma_i$ is zero, this vector defines a unit vector that is *orthogonal to this subspace.*

Further, to illustrate the above fact, for a moment study a case where the input data also is scalar, so that there holds $y = ax$ for some scalar $a$. This case is shown in Fig. 11.1: As the variable $x$ varies, the variable $y$ follows it following the linear dependency. When the $x$–$y$ pairs are projected onto the normal vector, the projection length for variable pairs that fulfill the constraint is always zero. However, because of noise, this seldom exactly holds, and one has $e = \Gamma^T v$ for some non-vanishing $e$. Because of the orthonormal nature of $\Gamma_i$, the dot product $\Gamma_i^T v$ directly tells the distance between the data point $v$ and the model. This gives an explicit solution to the error-in-variables problem presented in chapter 4: All variables have similar roles, all containing noise. Indeed, cleverly minimizing this model error gives yet another regression strategy, and this will be briefly studied in what follows.

### 11.2.2 Total Least Squares

One approach to implementing the EIV model (see Sec. 4.2.1) is the *Total Least Squares (TLS)* algorithm [11]. Following the idea presented above, search for such a regression hyperplane that when data points are orthogonally projected onto this plane, the (squared) distances reach minimum.

Here we continue with the single-output study for output $y_i$, so that

$$y_i = F_i^T x = F_{i,1}x_1 + \cdots + F_{i,n}x_n, \tag{11.10}$$

equalling

$$0 = \Gamma_i^T v \tag{11.11}$$

for

$$\Gamma_i = \begin{pmatrix} F_{i,1} \\ \vdots \\ F_{i_n} \\ -1 \end{pmatrix}, \qquad \text{and} \qquad v = \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ y_i \end{pmatrix}. \tag{11.12}$$

The dimension of the "augmented" data space of $v$, and the length of the vector $\Gamma_i$, is $n+1$. As was observed above, $\Gamma_i$ is orthogonal to the subspace that is "allowed" by the model. Further assuming that $\Gamma_i$ is normalized, so that $\|\Gamma_i\| = 1$, the dot product $e = \Gamma_i^T v$ directly tells the shortest distance (positive or negative) from the point $v$ to the regression hyperplane (for points lying exactly on the plane this measure, of course, giving 0, according to the model). the average of squared distances for a set of points $v(1)$ to $v(k)$ can be expressed as

$$\frac{1}{k} \cdot \sum_{\kappa=1}^{k} e^2(\kappa) = \frac{1}{k} \cdot \sum_{\kappa=1}^{k} \left(\Gamma_i^T v(\kappa) \cdot v^T(\kappa)\Gamma_i\right) = \frac{1}{k} \cdot \Gamma_i^T \cdot V^T V \cdot \Gamma_i, \tag{11.13}$$

where

$$\underset{k\times n+1}{V} = \left( \begin{array}{c|c} X & Y_i \end{array} \right). \tag{11.14}$$

To minimize this with the requirement that the normal vector must be normalized,

$$\begin{array}{ll} \text{Minimize} & \frac{1}{k} \cdot \Gamma_i^T \cdot V^T V \cdot \Gamma_i \\ \text{when} & \Gamma_i^T \Gamma_i = 1, \end{array} \tag{11.15}$$

leads to the Lagrangian formulation (see page 20) where one has

$$\left\{ \begin{array}{rcl} f(\Gamma_i) & = & \frac{1}{k} \cdot \Gamma_i^T \cdot V^T V \cdot \Gamma_i, \quad \text{when} \\ g(\Gamma_i) & = & 1 - \Gamma_i^T \Gamma_i. \end{array} \right. \tag{11.16}$$

The cost criterion becomes

$$J(\Gamma_i) = \frac{1}{k} \cdot \Gamma_i^T V^T V \Gamma_i + \lambda_i(1 - \Gamma_i^T \Gamma_i). \tag{11.17}$$

This results in

$$\frac{d}{d\,\Gamma_i} \left( \frac{1}{k} \cdot \Gamma_i^T V^T V \Gamma_i + \lambda_i(1 - \Gamma_i^T \Gamma_i) \right) = 0, \tag{11.18}$$

giving

$$\frac{1}{k} \cdot 2V^T V \cdot \Gamma_i - 2\lambda_i \cdot \Gamma_i = 0, \tag{11.19}$$

or

$$\frac{1}{k} \cdot V^T V \cdot \Gamma_i = \lambda_i \cdot \Gamma_i. \tag{11.20}$$

The distance minimization has become an *eigenvalue problem* with the searched normal vector $\Gamma_i$ being an eigenvector of the data covariance matrix $R = \frac{1}{k} \cdot V^T V$. However, as compared to principal component analysis, the searched normal vector is given by the principal component corresponding to the *least significant* eigenvalue — zero eigenvalue meaning exact match with the assumed model structure: In such a case, there must exist an exact linear dependency between the variables, and this dependency can be extracted as the model. Remembering the definition of the vector $\Gamma_i$, the final regression formula solved as

$$y_i = \frac{\Gamma_{i,1}}{\Gamma_{i,n+1}} \cdot x_1 + \cdots + \frac{\Gamma_{i,n}}{\Gamma_{i,n+1}} \cdot x_n. \tag{11.21}$$

For a multivariate system, the same analysis can be repeated for all outputs $y_i$ separately; note that the eigenproblem is generally different for all outputs. However, one needs to be careful: In the derivation $y_i$ was interpreted as any of the other input variables, meaning that it is not the output that was explicitly being explained (as is the case with MLR). This means that the TLS model not necessarily gives a good regression model for estimating the output.

This TLS method can also be called "last principal component analysis", as compared to PCA, where the solution (to the problem of maximizing variance rather than minimizing variation) is given in terms of the *most significant* principal components. This is an indication of the need for new thinking, indeed, *inverse thinking:* Rather than concentrating on the null space, or the *constraints,* one concentrates on *freedoms,* what is left outside, where there still exists non-nullified information.

TLS is an example of experiments when trying to rehabilitate the old way of thinking. However, the problems of very high dimensions are not solved. If there is a high number of redundant variables, many of the eigenvalues are practically zero. Which of the minor eigenvectors to select, then? This selection becomes very sensitive: With another data with another noise realization the ordering can become very different — giving a completely different model. This means that the noise sensitivity of the TLS model is increased unreasonably. And, as observed before, it is this noise sensitivitity that is a crucial matter when constructing good regression models.

### 11.2.3 Emergent models

Mathematically speaking, if there are $n$ separate variables, there are $n$ degrees of freedom in the data space, but each (linear) constraint decreases the number

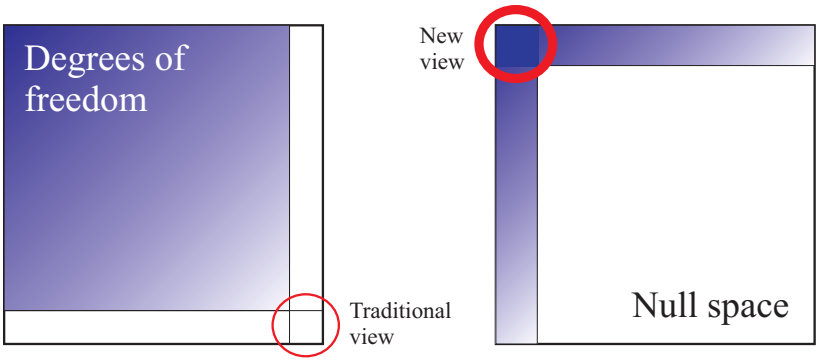



Figure 11.2: Schematic illustration of the covariance structure among data when there are few constraints (on the left), and when there are many constraints (on the right). The simplest presentation for the system properties changes as the number of constraints increases, or when the remaining degrees of freedom accordingly decrease

of degrees of freedom by one — specially, if there are $\nu$ linearly independent constraints, the number of remaining degrees of freedom is only $N = n - \nu$. Summarizing: The linear constraints constitute a null space within the data space: This means that in these directions there is no variability. The remaining $N$ directions in the data space constitute a linear subspace where all variation among variables is concentrated.

What do these degrees of freedom mean in practice? Originally, if there were completely separate unconnected variables (subsystems), there would be the maximum number of freedoms. When subsystems become connected, when interactions between them are established, the variables become coupled, thus reducing the number of free variables. Further, when feedbacks are introduced, the remaining inputs and outputs of the subsystems can still be connected. It is specially typical in cybernetic systems where this scenario holds: Ability to recover after disturbances is a manifestation of tightly interconnected system. In such systems it is only a few degrees of freedom that remain more or less loosely controlled.

The key point here is that essentially the same dependencies among variables can be captured in terms of degrees of freedom as with constraints. At some point, when the number of constraints increases, the *most economical* representation changes: The simplest model with the least parameters is no more the constraints-oriented model but the freedoms-oriented model (whatever it will be). According to the *Ockham's razor,* one needs to switch to *emergent models* when the system is cybernetic enough. In Fig. 11.2, the covariance structure of the data space is schematically depicted: When the null space of constraints is dead and dull, all interesting behaviors are concentrated in the directions of remaining freedoms.

It is difficult to escape the traditional ways of thinking: Traditional methods for analysis (modeling) and design (synthesis) are always based on models that are based on constraints.

it is the multivariate statistical methods that directly attack the degrees of freedom, abstracting away the structural details, that help to escape the constraints. Even though this opposite view of modeling sounds unintuitive, it turns out that the freedoms-oriented models are *more intuitive* than the constraints-oriented models, being based on the explicit time-domain features, as visualized below.

### 11.2.4 Examples

To visualize the freedoms-oriented model structures, exploit dynamic intuitions: Assume that the available variables are successive measurements of some signal $y$, so that samples are indexed as $y(\kappa)$, $y(\kappa-1)$, etc. Originally, it is assumed that these samples are independent of each other — it is the task of the (dynamic) model to connect the variables together. Assuming that the constraint-oriented model is

$$y(\kappa) = ay(\kappa - 1), \tag{11.22}$$

there is a direct connection to Fig. 11.1. Constructing the augmented data space as

$$v(\kappa) = \begin{pmatrix} y(\kappa - 1) \\ y(\kappa) \end{pmatrix}, \tag{11.23}$$

the whole data space $\mathcal{S}$ is spanned by the constraint vector and the freedom vector together:

$$\mathcal{S} = \left( \begin{array}{c|c} \Gamma & \theta \end{array} \right) = \left( \begin{array}{c|c} \frac{a}{\sqrt{1+a^2}} & \frac{1}{\sqrt{1+a^2}} \\ \frac{-1}{\sqrt{1+a^2}} & \frac{a}{\sqrt{1+a^2}} \end{array} \right). \tag{11.24}$$

The freedom-oriented way of describing the model is also

$$\theta = \begin{pmatrix} a \\ -1 \end{pmatrix} / \sqrt{1 + a^2}. \tag{11.25}$$

It is difficult to see here anything that would outperform the original model. However, now assume that there are three variables that are connected together by a model:

$$\begin{cases} y(\kappa) = ay(\kappa - 1) \\ y(\kappa + 1) = ay(\kappa). \end{cases} \tag{11.26}$$

This exactly corresponds to the model (11.22) where there are redundant variables. The key point here is that one does not know beforehand whether some of the variables are redundant — when modeling complex systems, this is typically the case. The data vectors are now

$$v(\kappa) = \begin{pmatrix} y(\kappa - 1) \\ y(\kappa) \\ y(\kappa + 1) \end{pmatrix}. \tag{11.27}$$

In this case, the constraint vectors without normalization are

$$\Gamma = \begin{pmatrix} a & 0 \\ -1 & a \\ 0 & -1 \end{pmatrix}. \tag{11.28}$$

The constraints span a two-dimensional subspace in the three-dimensional variable space – the remaining degree of freedom can be solved by orthogonalization, for example applying the *Gramm-Schmidt procedure.* To start with, one can take any linearly independent vector:

$$\begin{pmatrix} a & 0 & 1 \\ -1 & a & 0 \\ 0 & -1 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} a & \frac{a^2}{1+a^2} & \frac{1}{1+a^2} \\ -1 & \frac{a^3}{1+a^2} & \frac{a}{1+a^2} \\ 0 & -1 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} a & \frac{a^2}{1+a^2} & \frac{1}{1+a^2+a^4} \\ -1 & \frac{a^3}{1+a^2} & \frac{a}{1+a^2+a^4} \\ 0 & -1 & \frac{a^2}{1+a^2+a^4} \end{pmatrix}. \tag{11.29}$$

This means that the model becomes

$$\theta = \begin{pmatrix} 1 \\ a \\ a^2 \end{pmatrix} \Big/ \sqrt{1+a^2+a^4}. \tag{11.30}$$

The "axis of freedom" clearly has an *exponential outlook* in the data space. This is in exact correspondence with the actual time-domain behavior of a system that is characterized by a model of the form (11.22). Indeed, the degrees of freedom determine "behavioral fragments", so that the actual observations can be constructed as combinations of them. The patterns can be scaled arbitrarily to optimize the match — these scaling factors are the latent variables in $z$.

When working on simple cases, the approach is not crucial. But when new variables are introduced, each of them typically comes with an accompanying constraint, and it is only the degrees of freedom that truly reflect the essential dependency structures in the system. When modeling complex systems, it is assumed that the number of variables should not be limited artificially: Each of the new variables *can* contain some fresh information — the "accdompanying constraint" does not necessarily reduce the degrees of freedom in the augmented space exactly by one. Whereas the constraints-oriented modeling approach becomes a unmanageable mess, the freedoms-oriented models become clearer and clearer as the data dimension increases. The higher the number of variables is, the more appropriate is the pattern-based representation seems to become.

How about the interpretations when there is a higher number of remaining degrees of freedom? Study the model

$$y(\kappa) = a_1 y(\kappa-1) + a_2 y(\kappa-2), \tag{11.31}$$

or

$$0 = a_0 y(\kappa) - a_1 y(\kappa-1) - a_2 y(\kappa-2). \tag{11.32}$$

Now there is one constraint in the three-dimensional space, and two remaining degrees of freedom:

$$\Gamma = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} \quad \text{and} \quad v(\kappa) = \begin{pmatrix} y(\kappa) \\ y(\kappa-1) \\ y(\kappa-2) \end{pmatrix}. \tag{11.33}$$

The degrees of freedom for such a dynamic system have always the same interpretation: Typically, if there is considerable inertia in the system, the most significant principal component stands for a filter for finding the average momentary value of $y$, as being revealed by the latent variable $z_1(\kappa)$, and the second principal component stands for the trend prototype: The latent variable $z_2(\kappa)$ reveals the rate of change in the signal (see exercises). In this sense, there again exist very natural interpretations for the model structures.

In this kind of rather simple cases, there is a trade-off between approaches. The constraints-based model is stronger when it comes to analysis of dynamic phenomena (as the roots of the coefficient polynomial reveal the dynamic modes beyond the signal), whereas for the freedoms-oriented model such time-domain analyses need to be separately carried out (as presented in chapter 9), meaning that a heavier machinery needs to be employed.

The freedoms-oriented model is based on features taht constitute patterns that together explain the observations in the data space, assuming that there are some dependencies and redundancies in the behaviors. Determination of the system state becomes a *pattern recognition task.* Specially, when in the case of chemical systems, it is "chemical pattern matching" that is being carried out — and this is carried out automatically by the underlying thermodynamic processes.

## 11.3 Case studies

To illustrate the above approaches, two practical application examples are presented, where the "chemical semantics" is appropriate. Both of these complex processes are being currently studied at HUT Control Engineering Laboratory.

### 11.3.1 Characterizing the state in practical processes

To apply the ideas, the theoretical derivations still need to be extended towards practice. The data vector $v$ needs to be further studied to make it possible to capture all *internal tensions* in complex chemical systems. As it turns out, the following extensions can, for example, be implemented without ruining the linear structure among the variables:

- **Temperature.** According to the Arrhenius formula, the reaction coefficients are functions of the temperature, reactions becoming faster as the temperature rises, so that $k \propto \exp(c/T)$. This means that when this is substituted in the formulas, and when logarithms and differentiations are carried out, the model remains linear if the new variable is defined as $v_T = \Delta T/\bar{T}^2$.

- **Acidity.** The pH value of a solution is defined in terms of a nonlinear formula: $\mathrm{pH} = -\lg C_{\mathrm{H}^+}$. Because it is essentially logarithm taken of a concentration variable, one can directly include the changes in the pH value among the variables, $v_{\mathrm{pH}} = \Delta\mathrm{pH}$.

- **Voltage.** In electrochemical reactions, one should characterize the the "concentration of electrons". However, it turns out that acording to the Butler-Volmer theory [**?**], the amount of free electrons is exponentially proportional to the voltage. This means that, after taking the logarithms, the "electron pressure" can be characterized by the variable $v_{\mathrm{e}^-} = \Delta U$.

- **Dissipation.** It has been assumed that the systems being studied are in thermodynamic balance. This homeostasis can be extended, however: The steady state can be determined not only in terms of the variables, but also in terms of their derivatives. This means that one can study *dissipative systems,* where the rate of change remains constant, a constant flow of chemical flowing into or out from the system. Looking at the formula (11.3), it is clear that model linearity is not lost if one has variables like $v_{\dot{C}} = \Delta\dot{C}/\bar{\dot{C}}$.

- **Mass flows.** The concentration-oriented variables can be transformed into masses (molarities) when multiplied by volumes, meaning that after taking logarithms, the structure is linear. Similarly, the volumetric dissipation rates change into mass flows; further, surface phenomena (coating, etc.) are related to the surface area, so that if the volumes or areas change, one can include variables of the form $v_A = \Delta A/\bar{A}$ and $v_V = \Delta V/\bar{V}$.

- **Physical phenomena.** It is evident that structures that are originally linear, like phenomena that represent diffusion between compartments, etc., can directly be integrated in the model, assuming that appropriate variables (deviations from the nominal state) are included among the variables. What is more, smooth nonlinearities become affine when they are locally linearized, and, further, they become linear when developed around the nominal state.

In strong liquids one cannot always apply concentrations, but one has to employ *activities* instead, or actual activation probabilities. If it is assumed that these activities are some power functions of the concentration so that $\mathcal{A} = a_1 C^{a_2}$, after taking logarithms the model still remains linear in terms of the original concentrations. This means that — even though linearity is not compromized — the variables may become multiplied by some unknown factors, so that there is some scaling effect.

The vector $v$ selected here is the measurement vector, containing *all* possible quantities that can affect the system behavior — internal system variables and external environmental variables alike. This data presentation can capture the chemical domain semantics, and in different environments the models have different interpretations.

### 11.3.2 Case 1: Modeling an industrial nickel plating process[1]

In printed wiring boards, one needs a layer of nickel as an oxidation barrier between the copper electric circuitry and gold finishing (see Fig. 11.3). This nickel-phosphor layer can be created, for example, using electrochemical processes. The properties of the nickel layer can be affected by changing its phosphor content. It is clear that one should be capable of monitoring and controlling the layer thickness, and also its phosphor content so that the set values would be reached.

The chemical reactions taking place in the plating process are very complex, and not completely known. Four contradictory sets of reactions have been proposed to characterize the process, but none of them seems to satisfactorily explain observed behaviors. Not only is the exact process structure unknown — not all chemicals are either known, as the compositions of the commercial reagents are business secrets. However, the processes are slow, and it is evident that the appropriately operated coating process remains well in balance. All these observations are well in line with the assumptions beyond the freedoms-oriented modeling.

The process state can be characterized in terms of its acidity or pH (controlled using ammonia to be between 4.7 and 5.0), temperature (to be around 80 degrees centigrade), nickel concentration (controlled by adding nickel sulphate), and electrical potentials. The dynamics is also affected by the loading, or the total area to be plated simultaneously in the bath. In addition to these, additional chemicals are present, some of them are known, like the *reducers* (sodium hypophosphite), and some are not (different kinds of activators and inhibitors); the contribution of the residues of reaction chemicals is also estimated: The variable MTO (or "metal turn-over") descibes the aging of the process liquids, being supposedly proportional to the concentrations of the unspecified chemicals. All these state variables can be recorded or calculated in a practically continuous manner.

It is the properties of the final nickel surface that cannot be measured on-line: The layer thickness should be around 4 $\mu$m, and it should contain some 7 – 10 weight percent phosphor. Information of these is available only after laboratory analyses, once or twice a day, and a model is needed to estimate these quantities in a reliable way. To implement such soft sensors, the multivariate regression models were constructed.

As it is typically the case, the model (or data preprocessing) needs to be tailored to match the problem domain. The state variables were mean-centered and normalized in the traditional way — but, in addition to these variables, new ones could also be employed. This nicely illustrates the benefits of the simple linear model structure.

Because the relative changes in the momentary layer growth rate assumedly are linear functions of changes in the other state variables, the overall relative change is reached when one integrates the momentary rate over the bath time. And because of the linearity of this mapping model $F$, the integration can be

[1]The simulations were carried out by Mr. Hans-Christian Pfisterer

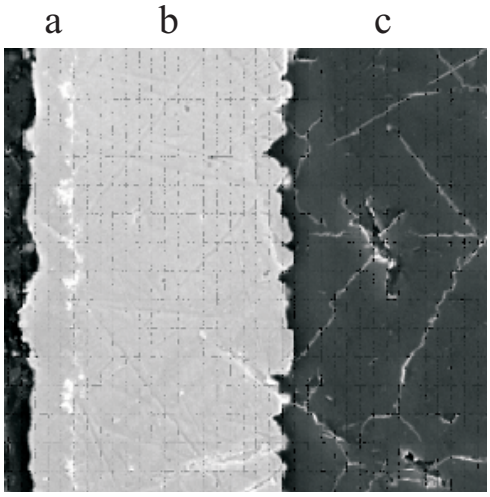



Figure 11.3: Cross-section of a test plate: **a** - the Ni-P layer (about 5 $\mu$m); **b** - copper layer; **c** - base (epoxy laminate)

moved "through" the model:

$$\Delta l(t) = \int_{t_0}^{t} \Delta \dot{l}(\tau)/\bar{\dot{l}} d\tau = \int_{t_0}^{t} F^T v(\tau)\, d\tau = F^T \int_{t_0}^{t} v(\tau)\, d\tau. \tag{11.34}$$

This means that if one includes the integrals of relative changes among the $x$ variables, a linear model should be capable of capturing the layer changes around the nominal cumulation rates. These nominal absolute values need to be separately modeled, or if the bath time of the board is also included among the input variables, it is the same model that suffices.

It is always difficult to evaluate the performance of the models in an unbiased way — however, in this case we are lucky: There is an explicit model derived specially for this process, starting from physico-chemical first principles, the free parameters being optimally tuned to match the observations. It can be assumed that this model is the best model one can construct for the process, as that modeling effort gained the the Best Diploma Thesis Prize of 2004 in Finland (as granted by TEK, the Finnish Association of Graduate Engineers). The results are shown in Figs. 11.4 and 11.5: Even though not all phenomena can be estimated by the model of four PCA-based latent variables (see Fig. 11.5), it seems that the same problems are faced by all models regardless of their construction. The data-oriented model where no process-specific knowledge is exploited is well comparable with the expert-tuned physical model that is based on a set of highly nonlinear differential equations: The validation errors for fresh data have the same orders of magnitude (results for two set of validation data shown in the figures).

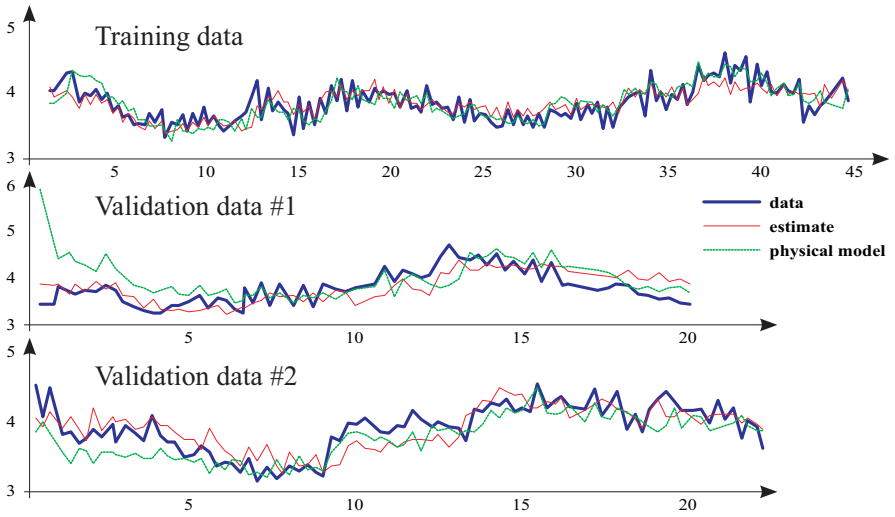



Figure 11.4: Estimates for nickel layer thickness

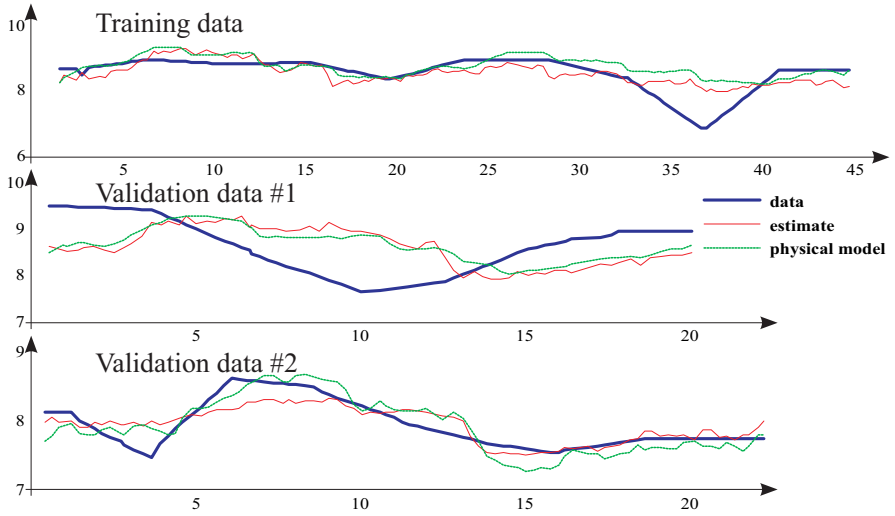



Figure 11.5: Estimates for phosphor content

### 11.3.3 Case 2: Modeling genetic networks and metabolic systems[2]

The previous example was a man-made system based on more or less designed chemical reactions, the reaction mechanisms being predetermined to explicitly implement intended behaviors, and a (more or less accurate) first-principles model could also be constructed. Now, study a natural system that is still much more complex, so that finding the explicit reaction mechanisms is even more complicated — perhaps the same principles of freedoms-based modeling apply?

When studying metabolic reactions, it is complex chains of reactions based on organic chemistry that should be mastered. What is more, these reactions are dictated by the genetic processes, where enzymes are produced. On the other hand, the chemical state affects the gene activities — this means that there are interacting genetic and metabolic networks that should be mastered. The closed control loops cannot be distinguished from each other, and the only realistic approach is to assume "pancausality", where the interactions and feedbacks constitute the tensions keeping the system in balance. As studied in chapter 2, genetic networks can be modeled applying the same model structures as the chemical processes — the metabolic processes are fast, whereas the genetic ones are slow (see Fig 11.6). Both of the levels can be combined in one model structure, making it perhaps possible to reach *systemic biology.* In the figure, the linear pattern recognition processes are expressed in terms of dynamic state-space models.

In the project SyMbolic (Systemic Models for Metabolic Dynamics and Gene Expression), funded by TEKES during 2004 – 2006, new kinds of models were derived for representing the cellular dynamics, and one of the approaches was the exploitation of the idea of emergent models [**?**].

There is plenty of data: The modern ChIP techniques, etc., provide huge amounts of measurements, as all gene activities can be simultaneously measured (for example, see [**?**]). Indeed, measuring gene activities (in terms of active messenger-RNA) is more straightforward than measuring the metabolites. Even though there is plenty of data, it is not optimally conditioned for dynamic identification purposes: The dimension of data (in thousands) is higher than what is the number of samples (in hundreds), and the excitation sequences are not persistently exciting (being step experiments). What is more, the data is very noisy — partly because of the uncertainties in the measurement process, and partly because measurements carried out in different laboratories seem not to be quite compatible. This means that the statistical multivariate methods, and specially the latent variable approaches, are well motivated also from the pragmatic point of view.

Implicitly, the latent variable methods assume that there is redundancy among genetic and cellular functionalities — and, indeed, it has been shown that there are typically groups of genes rather than individual genes that are responsible for the functionalities. And also on the metabolic level: Processes in the cytoplasm are well buffered, and typically there are negligible responses if one only considers a single input and a single output. The multivariate methods make it

[2]The simulations were carried out by Mr. Olli Haavisto, M.Sc.

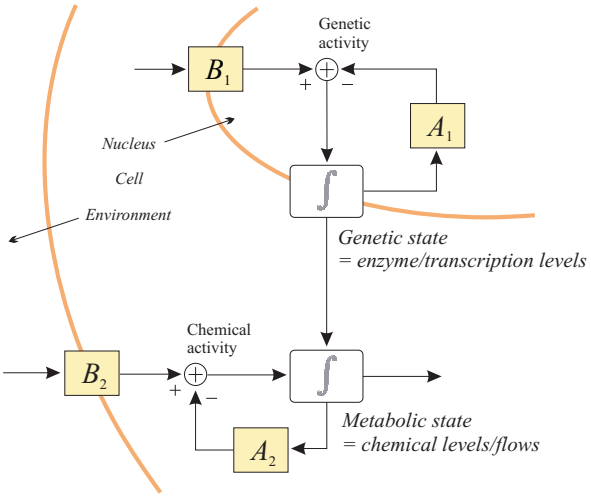



Figure 11.6: Two time scales in the cellular system

possible to study the whole grid of proteomic/metabolomic phenomena simultaneously — this means that one does not need to employ excessive excitation signals, or huge dosages, resulting in considerable disturbances in the cell behavior, or even death. The gentle approaches are necessary when one wants to study living cells rather than pathological, more or less irrelevant cases.

As an application example, modeling of data from yeast cell cultivations were used (see [**?**]). There were a few dozen experiments, where different kinds of step changes in the environment were executed, and the resulting gene activity transients were recorded. Modeling this data was quite a challenge, as there was not enough data. Even though the applied model structure was robust, no conclusive conclusions can be drawn.

As was observed above, metabolite concentrations and gene activities could be represented in the linear model structure, variaqbles being collected in a single vector. However, now the model was restructured so that dynamics was captured: The environmental variables (substrate properties, temperature, etc.) were collected in the input vector $u$, and the gene expression levels were collected in the output vector $y$. Mean-centering and normalization of data was carried out. The dimensions of the vectors were such that $n_u$ was about ten, and $m$ was about 4000; the number of latent variables $N$ was selected as 4, and stochastic-deterministic subspace identification was applied.

The assumption beyond the adopted modeling approach is that balances are more characteristic to cellular systems than the transients are. And, indeed, it seems that the steady states are nicely modeled, whereas the transient behaviors are not reproduced by the model (see Fig. 11.7). Still, it seems that the extreme compression of the variable space does not ruin the steady-state correspondence. There seem to exist only few degrees of freedom left in the behavioral data.

It can be claimed that the degrees of freedom in a cellular system characterize *metabolic behaviors* or *functions*. When the environment changes, the new balance is found along these axes in the chemical space whan "chemical pattern matching" is carried out. For example, assuming that available glucose

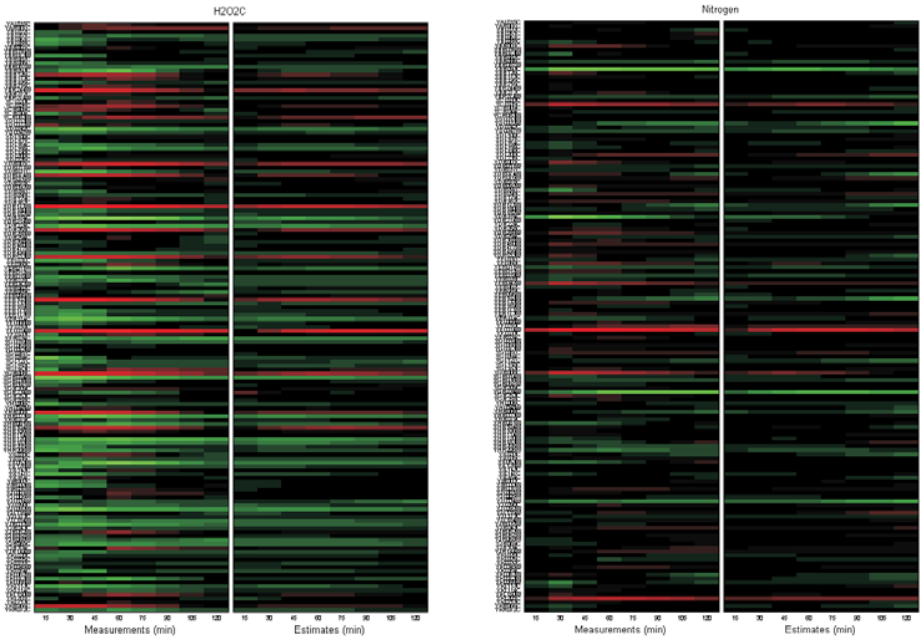



Figure 11.7: Two open-loop experiments with the model, showing 256 "stress genes" (red color meaning activity increase, green meaning activity decrease). In the leftmost figures, hydrogen peroxide step is being simulated for two hours, and in the rightmost ones, nitrogen step is simulated. In both cases, the actual behaviors in the genetic state are shown on the left, and the estimates given by the four-state model are shown on the right. Despite the transients, there is a good correspondence between the observations and the very low-dimensional model (see [**?**])

goes up, it is also mannose production that goes up, or some other processes that exploit glucose. There is only balance pursuit taking place: But after "anthropocentric", finalistically-loaded interpretations are employed, when some chemicals are interpreted as nutrients, some others as metabolic products, and the rest as waste, one reaches "emergent interpretations". When complexity cumulates, the balance reactions start looking goal-oriented, pre-planned, and "clever". Scarcity of some chemicals changes the balance appropriately, trying to compensate for the shortage.

## 11.4 Towards "artificial cells"

New conceptual tools become available as further interpretations are employed. In complex chemical systems, there seem to exist reserve mechanisms for compensating for the disturbances. This kind of buffering is characteristic not only to metabolic systems, but it seems to apply also in more general terms: *Le Chatelier principle* states that changes in environment are compensated by changes in the balance, so that the system tries to "escape" the changes. In

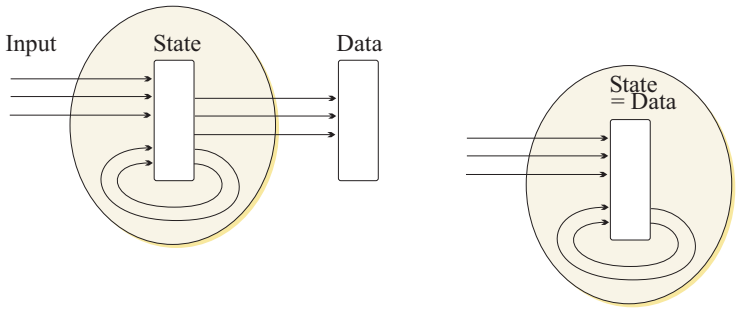



Figure 11.8: From *data modeling* (on the left) towards *system modeling* (on the right). The variables being measured are system variables: Because of pancausality, changing them also changes the system state

[**?**], the idea of "elastic systems" is proposed to characterize the reactions of cybernetic systems in general.

When the variables are selected appropriately, so that system semantics is captured, and if the pancausality assumption holds, the constructed modes are not only *data models* — they are *system models.* They can capture the fundamental essence of systems. They can be used not only for monitoring, but also for design and control construction: Changing variables appropriately also changes the resulting balance (see Fig. 11.8). The remaining degrees of freedom in the system reveal the possibilities of further controls to make the system still more balanced; in this sense, *process data mining* becomes possible, where information can be gathered directly from the behaviors, not from model-based assumptions. New kinds of models make it possible to implement new kinds of controls — higher-level controls. However, new challenges are faced: When new feedbacks are introduced, the set of freedoms changes. Control design becomes an iterative task, and new kinds of design tools are needed.

The ideas of biological cybernetic systems can be extended to technical (bio)processes: The still unbounded degrees of freedom can be regulated, new feedbacks can be constructed. Still better balanced "superorganisms" are constructed. The industrial systems are becoming like *artificial cells* themselves: Industrial plants also have *metabolism,* raw materials being exhausted and others being produced. Originally, the production can be far from optimum, but as soon as dependencies among variables are recognized, they can be used for constructing new feedback structures to implement more efficient and robust — better balanced — production. In both cases, in natural and man-made cells alike, it turns out that the goal of "evolution" is overall efficiency of production, no matter whether it is humans that are acting as agents for development or not. This can be reached by implementing mechanisms for reaching best possible production conditions; and this system integrity needs to be maintained without collapses. To maintain such balance, the system has to respond appropriately to the spectrum of disturbances coming from the environment.

It seems that the new approaches offer new possibilities for attacking the mysteries of evolutionary processes from a fresh point of view — such visions are studied closer in [**?**]).

## Computer exercises

1. Assume that data of an oscillating system is collected and its time-series is analyzed, that is, dynamics is being captured in data, and study the covariance structure:

   ```
   y = sin([1:100]/2)';
   V = [y(1:98),y(2:99),y(3:100)];
   theta = regrPCA(V)
   ```

   Interpret the distribution of the eigenvalues. Also interpret the first and second eigenvector as patterns characterizing the signal.

2. Applying the same data, study the eigenvector with the vanishing eigenvalue (carrying out the Total Least Squares regression analysis):

   ```
   G = regrPCA(V,-1)              % Also "regrTLS" available
   abs(roots(G))
   ```

   Interpret the result. What happens with the above analyses (freedoms vs. constraints) if the data is extended so that

   ```
   V = [y(1:97),y(2:98),y(3:99),y(4:100)];
   theta = regrPCA(V)
   ```

   Try to interpret the eigenvectors and eigenvalues now. What can you say about the extensibility and robustness of the two approaches?

# Bibliography


1. Abbott, E.A.: *Flatland: A Romance of Many Dimensions.* Originally printed in 1884.
   Available at `http://encyclopediaindex.com/c/flat10a.htm`.

2. Ayres, F.: *Theory and Problems of Matrices.* Schaum Publishing Company, New York, 1962.

3. Basilevsky, A.: *Statistical Factor Analysis and Related Methods.* John Wiley & Sons, New York, 1994.

4. Berndtson, J. and Niemi, A.J.: Automatic observation of the dry line in paper machine. In the *Proceedings of the 13th International Conference on Pattern Recognition,* Vienna, Austria, August 25–29, 1996, Vol. III, pp. 308–312.

5. Bishop, C. M.: *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, UK, 1995.

6. Buckingham, E.: On physically similar systems; illustrations of the use of dimensional equations. *Phys. Rev.* Vol 4, 1914, pp. 345–376.

7. Cheng B. and Titterington D. M.: *Neural Networks: A review from a statistical perspective.* Statistical science, vol. 9, No. 1, 1994, pp. 2–54.

8. Földiák, P.: Forming sparse representations by local anti-Hebbian learning. *Biological Cybernetics,* Vol. 64, No 2, 1990, pp. 165–170.

9. Gevers, M. and Li, G.: *Parametrizations in Control, Estimation and Filtering Problems: Accuracy Aspects.* Springer–Verlag, London, UK, 1993.

10. Glover, K.: *All optimal Hankel-norm approximations of linear multivariable systems and their $L^\infty$-error bounds.* International Journal of Control, Vol. 39, No. 6, pp. 1115–1193, 1984.

11. Golub, G.H. and van Loan, C.F.: *Matrix Computations* (2nd edition). The Johns Hopkins University Press, Baltimore, Maryland, 1989.

12. Goodwin, G.C.: *Some observations on robust estimation and control.* Proceedings of the 7th IFAC Symp. on Identification and System Parameter Estimation. Pergamon Press, Oxford, UK, 1985.

13. Haykin, S.: *Neural Networks. A Comprehensive Foundation.* Macmillan, 1994.

14. Hebb, D. *The Organization of Behavior.* Wiley, New York, 1949.

15. Hirsch, M.W. and Smale, S.: *Differential Equations, Dynamical Systems, and Linear Algebra.* Academic Press, San Diego, California, 1974.

16. Hyvärinen, A., Karhunen, J., and Oja, E.: *Independent Component Analysis.* John Wiley & Sons, New York, 2001.

17. Hyvärinen, A., Oja E.: *A fast Fixed-Point Algorithm for Independent Component Analysis.* Neural Computation, Vol. 9, 1997.

18. Hyötyniemi, H: *Self-Organizing Maps for Dynamic Systems Modeling and Control.* Helsinki University of Technology, Control Engineering Laboratory, 1994.

19. Hyötyniemi, H: *Information Content Weighting of Algorithms.* Preprints of the 10th IFAC Symposium on System Identification (SYSID'94), Copenhagen, Denmark, July 4–6, 1994, Vol. 3, pp. 417–422.

20. Hyötyniemi, H. and Ylinen, R.: *Improving Robustness of Parameter Estimation.* Proceedings of the First Asian Control Conference (ASCC'94), Tokyo, Japan, July 27–30, 1994, Vol. 1, pp. 415–418.

21. Hyötyniemi, H.: *Regularization of Parameter Estimation.* The 13th IFAC World Congress, July 1–5, 1996, San Francisco, California.

22. Hyötyniemi, H.: *On Structural Identifiability of Dynamic Models.* Preprints of the IFAC Symposium on System Identification (SYSID'97), Fukuoka, Japan, July 8–11, 1997, Vol. 1, pp. 243–248.

23. Hyötyniemi, H.: From intelligent models to smart ones. In *Proceedings of the 2nd Int. Conf. on Intelligent Processing and Manufacturing of Materials (IPMM'99),* Honolulu, Hawaii, July 10-15, 1999, Vol. 1, pp. 179–184.

24. Hyötyniemi, H.: *GGHA Toolbox for Matlab.* Helsinki University of Technology, Control Engineering Laboratory, Report 115, 1999.

25. Hyötyniemi, H. Ylinen, R., and Miettunen, J.: AI in Practice: Case study on a flotation plant. In the *Proceedings of the 9th Finnish Artificial Intelligence Conference (STeP 2000),* Aug. 28-30, 2000, Espoo, Finland, Vol. 2, pp. 159–166.

26. Johansson, R.: *System Modeling & Identification.* Prentice Hall, Englewood Cliffs, New Jersey, 1993.

27. Kohonen, T.: *Self-Organizing Maps.* Springer-Verlag, Berlin, Germany, 1995.

28. Lee, T.-W.: *Independent Component Analysis: Theory and Applications.* Kluwer Academic Publishers, Boston, 1998.

29. Ljung, L. and Söderström, T.: *Theory and Practice of Recursive Identification.* MIT Press, Cambridge, Massachusetts, 1983.

30. Maciejowski, J.: *Multivariable Feedback Design.* Addison-Wesley, Boston, 1989.

31. Martens, H. A.: *Multivariable Calibration — Quantitative Interpretation of Non-Selective Chemical Data.* Doctoral Thesis, Technical University of Norway, Trondheim, Norway, 1985.

32. Morrison, D.F.: *Multivariate Statistical Methods.* McGraw-Hill, New York, 1967.

33. Noble, B. and Daniel, J.W.: *Applied Linear Algebra.* Prentice–Hall, Englewood Cliffs, New Jersey, 1977.

34. van Overschee, P. and de Moor, B.: *Subspace Identification for Linear Systems.* Kluwer Academic Publishers, Boston, Massachusetts, 1996.

35. Pindyck, R.S. and Rubinfeld, D.L.: *Econometric Models and Econometric Forecasts* (3rd edition). McGraw-Hill, New York, 1991.

36. Sarle, W.S.: *Neural Networks and Statistical models.* Proceedings of the nineteenth annual SAS Users group international conference, April 1994.

37. Tabachnick, B.G. and Fidell, L.S.: *Using Multivariate Statistics* (4th edition). Allyn and Bacon, Boston, 2001.

38. Wold, H.: Soft modelling with latent variables: the nonlinear iterative partial least squares approach. In Gani, J. (ed.): *Perspectives in probability and Statistics: Papers in honour of M.S. Barlett,* Academic Press, London, 1975, pp. 114–142.

# Appendix A

# Structure *from Data*

The top-down (qualitative, structural) and bottom-up (quantitative, numeric) modeling approaches are fundamentally incompatible[1]. This report concentrates on the data-oriented modeling approach: The discussions were based exclusively on data. However, here in these Appendices we try to bridge the gap between the two extremes — or, at least, we try to bring the two approaches nearer to each other.

This first appendix concentrates on the approach *from data towards structure,* that is, it is assumed that analysis of data suggests some underlying structure explaining the observations. The data-suggested structure is typically seen as the clustered nature of the data. The latter appendix concentrates on the approach *from structure towards data,* that is, the data is explicitly modified to match the known structure. In both cases, these structure-oriented analyses are carried out *before* the actual data modeling, or regression analysis, is done.

## A.1 Cluster analysis

Determination of the system structure is, a complex and knowledge-intensive task. Typically, when doing multivariate modeling, no *a priori* information about the underlying subprocesses exists. This complexity is reflected in the data: The emergence of relevant clusters cannot be foreseen. There exist no unique solutions to the data clustering problem, and clustering is typically based on more or less heuristic algorithms. Because of the nonlinear and noncontinuous nature of the clustering problem, there exist only iterative, trial-and-error algorithms for this purpose.

In what follows, two prototypical clustering approaches will be studied a little closer. Both of them work well only for "nice" data, hoping that the clusters are more or less clearly distinguishable in the measurements.

[1]It is as in artificial intelligence (AI) — either you do it symbolically with expert systems, etc., or you do it numerically with neural networks, etc. — and there seem not to exist natural combinations in between

### A.1.1 K-means algorithm

The *K-means algorithm* is the basic approach that is used for clustering: Starting from some initial guesses, the clusters are refined by moving samples from a cluster to another depending on which of the clusters happens to be closest. When samples are redistributed, the cluster centers also evolve; iteration is needed to reach convergence.

The algorithm that searches for $N$ distinct clusters (parameter $N$ being fixed beforehand) can be written as follows (references to the clusters are now shown as superscript indices):

1. Choose a set of original cluster centers $\bar{v}^1, \ldots, \bar{v}^N$ arbitrarily, for example, let $\bar{v}^1 = v(1), \ldots, \bar{v}^N = v(N)$.

2. Assign the $k$ samples to the $N$ clusters using the minimum Euclidean distance rule: Sample $v(\kappa)$ belongs to cluster $c$, or $v(\kappa) \in \Gamma^c$, if $\|v(\kappa) - \bar{v}^c\| \leq \|v(\kappa) - \bar{v}^{c'}\|$ for all $c' \neq c$.

3. Compute new cluster center prototypes $\bar{v}^c \leftarrow \sum_{v(\kappa)\in\Gamma^c} v(\kappa)/\#\{\Gamma^c\}$, where $\#\{\Gamma^c\}$ denotes the number of vectors in cluster $\Gamma^c$.

4. If any of the cluster prototypes changes, return to step 2, otherwise, stop.

Note that it is assumed that $v$ contains now all available information, containing both the input variables (later denoted $x$) and the output variables (later $y$); in some sources this approach is called "input-output clustering". It is also possible to use $x$ exclusively[2].

If one determines some *topology* (indeed, a *metric*) among the clusters, so that some of the clusters are assumed to be "nearer" to each other than some others, one can easily extend the K-means algorithm towards *self-organizing map.* If it is not only the cluster itself whose center is moved towards the local data center, but also its "neighbors" are slightly adapted in the same direction, one has (a version of) the *Batch-SOM* algorithm [**?**]. In the converged cluster organization, the "neighboring" clusters will stand for nearby data samples, so that a "map" is constructed.

The K-means clustering method works reliably, and sometimes it gives useful results. However, there is a basic problem: The distances are calculated using the Euclidean norm. If searching for linear dependency models within the clusters, it is not pointwise but linear, "longish" data clusters that support linear model construction. How this can be achieved, is studied next.

### A.1.2 EM algorithm

The *Expectation Maximization (EM)* algorithm is a more sophisticated approach as compared to the basic K-means algorithm: Clustering is serched for in the maximum likelihood sense, fitting Gaussian distributions with data in a more

[2]Indeed, this is the normal approach: When the models are applied, it is only the input $x$ that is available for determining the cluster

complicated way ... needless to say that there is no guarantee about the convergence or the uniqueness of the solutions. It is not only the cluster centers, or means of the distributions, but also the "outlooks", or covariance matrices, that need to be determined here; this means that the number of free modl parameters is very high. As there exist various local minima, bootstrapping the algorithm is also complicated — typically the initial guesses for clusters are calculated using the K-means algorithm.

First, study the Gaussian distribution (2.1) a bit closer. Probability density reaches maximum simultaneously as its (natural) logarithm does; this means that one can define the *log-likelihood measure* for each cluster:

$$\begin{aligned} \ln(p(v)) \quad = \quad & -\tfrac{\dim\{v\}}{2} \cdot \ln(2\pi) \\ & -\tfrac{1}{2} \cdot \ln(\det\{R^c\}) \\ & -\tfrac{1}{2} \cdot (v-\bar{v}^c)^T (R^c)^{-1} (v-\bar{v}^c). \end{aligned} \tag{A.1}$$

This criterion can be applied for determining into which cluster a data sample should be put to maximize the overall model fit with the data. The first term above is constant for different clusters, and it can be neglected; the role of the second term, regulating the *a priori* cluster probability to fixed level, is to prevent the cluster from growing unboundedly. Finally, the third term matches the data points against the Gaussian models within clusters; essentially one calculates the *Mahalanobis distance* from the data point to the cluster $c$:

$$(v-\bar{v}^c)^T (R^c)^{-1} (v-\bar{v}^c). \tag{A.2}$$

This measure determines how "longish" the distribution is; searching for the locations of the equidistant points in the $v$ space using this distance measure, ellipsoids are found. Based on log-likelihood, the EM algorithm can be written as

1. Choose a set of original cluster centers $\bar{v}^1, \ldots, \bar{v}^N$ arbitrarily, for example, using the K-means algorithm; the cluster covariances are originally identity matrices, or $R^c = I$.
2. Assign the $k$ samples to the $N$ clusters using the minimum (balanced) Mahalanobis distance rule: Sample $v(\kappa)$ belongs to cluster $c$, or $v(\kappa) \in \Gamma^c$, if $\ln(\det\{R^c\}) + (v(\kappa)-\bar{v}^c)^T (R^c)^{-1} (v(\kappa)-\bar{v}^c)$ becomes minimum.
3. Compute new cluster center prototypes $\bar{v}^c \leftarrow \sum_{v(\kappa)\in\Gamma^c} v(\kappa)/\#\{\Gamma^c\}$ and covariance estimates $R^c \leftarrow \sum_{v(\kappa)\in\Gamma^c} (v(\kappa)-\bar{v}^c)(v(\kappa)-\bar{v}^c)^T/\#\{\Gamma^c\}$ where $\#\{\Gamma^c\}$ denotes the number of vectors in cluster $\Gamma^c$.
4. If any of the cluster prototypes changes, return to step 2, otherwise, stop.

Note that K-means algorithm results if it is explicitly assumed that in all clusters $R^c \equiv \sigma^2 \cdot I$ for some $\sigma^2$. The EM algorithm can be made more stable if some additional assumptions can be made. For example, if one can assume that the nonlinearity within the data is simple *affinity,* so that only the cluster centers vary while the internal structures within the clusters remains unchanged, there holds $R^c = R^{c'}$ for all $c$ and $c'$ — this effectively reduces the problem complexity.

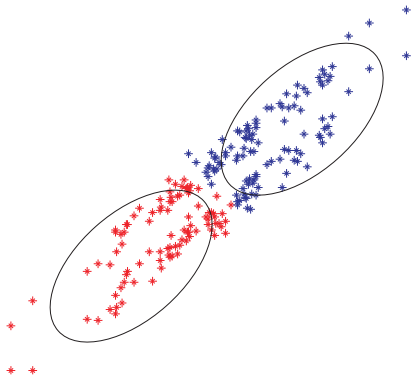


Figure A.1: Incorrect, K-means type clustering result

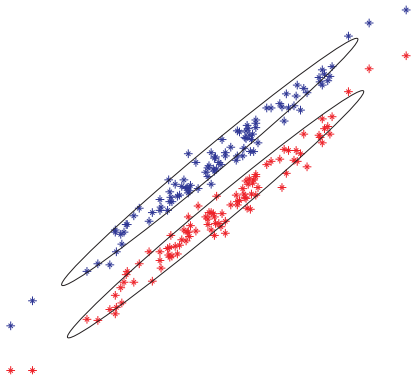


Figure A.2: Intuitively correct clustering result

The presented EM algorithm matches well with the (Gaussian) mixture model scheme that was discussed in Chapter 2: Data samples within the clusters have the direct probability interpretation, so that the combination of the submodels constructed for individual clusters can be carried out in the maximum likelihood sense.

## A.2 Classification

Sometimes the appropriate classes are already known — but they are known only by examples. Then one is facing a *classification problem*: How to determine the *decision boundary* between the classes, so that, when facing fresh data, the probability of false classifications would be minimized?

### A.2.1 Fisher discriminant analysis

Knowing the clusters, it would be good to know some structure among the clusters. Additionally, sometimes it would be nice to have a *linear* criterion for determining how well a new sample matches a cluster and how near it is to the neighboring clusters. One would like to find a projection axis so that data belonging to different clusters, as projected onto this axis, would be maximally distinguishable.

From the classification point of view, one can assume that the clusters carry the classification information, whereas the variation around the cluster centers can be interpreted as noise. One can try to find such a projection of the data that the *signal-to-noise ratio* is maximized. So, first define the "noise sequence" so that the cluster centers $\bar{v}^{c(\kappa)}$ corresponding to each of the classified sample $v(\kappa)$ is eliminated:

$$v_{\text{noise}}(\kappa) = v(\kappa) - \bar{v}^{c(\kappa)}. \tag{A.3}$$

The signal sequence, then, is the sequence of cluster centers:

$$v_{\text{signal}}(\kappa) = \bar{v}^{c(\kappa)}. \tag{A.4}$$

Assume that the projection axis that is being searched for is $\theta_i$. A data sample $v(\kappa)$ projected onto this axis is $v^T(\kappa)\theta_i$; its square thus is $\theta_i^T v(\kappa)v^T(\kappa)\theta_i$. The average of this, or the variance, can be written separately for the signal and the noise sequences, giving

$$\begin{aligned} &\tfrac{1}{k} \cdot \textstyle\sum_{\kappa=1}^{k} \theta_i^T v_{\text{signal}}(\kappa) v_{\text{signal}}^T(\kappa)\theta_i \\ &\quad = \theta_i^T \cdot \tfrac{1}{k} \cdot \textstyle\sum_{\kappa=1}^{k} v_{\text{signal}}(\kappa) v_{\text{signal}}^T(\kappa) \cdot \theta_i \\ &\quad = \theta_i^T \cdot R_{\text{between}} \cdot \theta_i, \end{aligned} \tag{A.5}$$

and

$$\begin{aligned} &\tfrac{1}{k} \cdot \textstyle\sum_{\kappa=1}^{k} \theta_i^T v_{\text{noise}}(\kappa) v_{\text{noise}}^T(\kappa)\theta_i \\ &\quad = \theta_i^T \cdot \tfrac{1}{k} \cdot \textstyle\sum_{\kappa=1}^{k} v_{\text{noise}}(\kappa) v_{\text{noise}}^T(\kappa) \cdot \theta_i \\ &\quad = \theta_i^T \cdot R_{\text{within}} \cdot \theta_i. \end{aligned} \tag{A.6}$$

Here, the matrices $R_{\text{between}}$ and $R_{\text{within}}$ denote the "between-classes" covariance and the "within-classes" covariance, respectively. Now the problem of maximizing the between-classes variance while keeping the within-classes variances constant can be expressed as a constrained optimization task

$$\begin{array}{ll} \text{Maximize} & \theta_i^T \cdot R_{\text{between}} \cdot \theta_i \\ \text{when} & \theta_i^T \cdot R_{\text{within}} \cdot \theta_i = 1. \end{array} \tag{A.7}$$

This can be formulated in the Lagrangian framework (see page 20) when selecting

$$\left\{ \begin{array}{rcl} f(\theta_i) & = & \theta_i^T \cdot R_{\text{between}} \cdot \theta_i \\ g(\theta_i) & = & 1 - \theta_i^T \cdot R_{\text{within}} \cdot \theta_i. \end{array} \right. \tag{A.8}$$

Using the Lagrange multipliers, the optimum solution $\theta_i$ has to obey

$$\frac{d\,J(\theta_i)}{d\theta_i} = \frac{d}{d\theta_i}\left(f(\theta_i) - \lambda_i \cdot g(\theta_i)\right) = \mathbf{0} \tag{A.9}$$

or

$$R_{\text{between}} \cdot \theta_i - \lambda_i \cdot R_{\text{within}} \cdot \theta_i = \mathbf{0}, \tag{A.10}$$

giving

$$R_{\text{between}} \cdot \theta_i = \lambda_i \cdot R_{\text{within}} \cdot \theta_i. \tag{A.11}$$

If the matrix $R_{\text{within}}$ is invertible, this can further be solved as

$$R_{\text{within}}^{-1} R_{\text{between}} \cdot \theta_i = \lambda_i \cdot \theta_i. \tag{A.12}$$

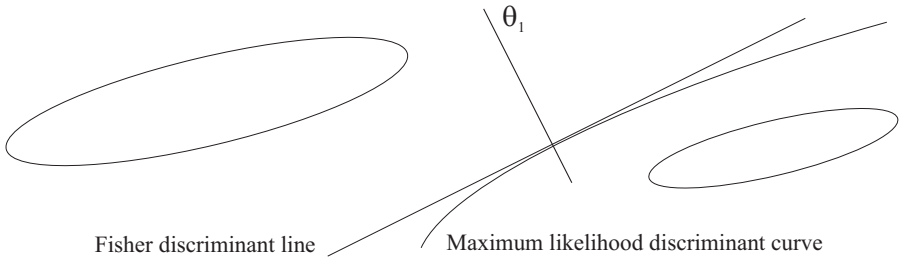



Figure A.3: Fisher discriminant axis $\theta_1$ in a two-cluster case with unequal covariances. Note two things: First, the discriminant axis is not pointed from one cluster center to the other, the covariance structure of the clusters affecting its orientation; second, the maximum likelihood discriminant surface between clusters is generally not a plane but an hyperellipsoid (or some other generalized conic section determined by the equi-distance points in the Mahalanobis sense)

This is an *eigenproblem* (and (A.11) is so called *generalized eigenproblem*)[3]. That is, the best projection axes are given as eigenvectors of the above problem. The eigenvector corresponding to the largest eigenvalue is the best in this sense. Note that if there are only two clusters, the axis is unique (the rank of $R_{\text{between}}$ being 1, all but one of the eigenvalues being zeros).

As an example, assume that there are only two clusters with equal covariances. In this case the *discriminant (hyper)plane* between the clusters is defined by those points that lie on the hyperplane going through the center point between the clusters and being perpendicular to the axis $\theta_1$. However, if the clusters do *not* have equal covariance structures, the linear Fisher discriminant no longer gives the theoretically correct separation between the clusters (see Fig. A.3).

If one is trying to find an appropriate way of scaling ones data, the FDA model can also give some intuition. The signal-to-noise ratio information can directly be used for weighting purposes according to the weighting scheme (B.31). This approach is generally used, more or less knowingly, for example, in *data mining:* The relevance of different words in textual documents is estimated by checking how often they are found in "interesting" documents and, on the other hand, in "non-interesting" ones. This information about frequency variations can be used for weighting the words appropriately.

### A.2.2 Support Vector Machines (SVM)

One of the most promising modern classification methods is called a *Support Vector Machine* or SVM for short [**?**]. It is a result of sophisticated mathematics, optimization, and statistical learning theory — and, surprisingly, the basic ideas are very well compatible with the ideas of multivariate regression as studied in this report:

- The structural complexity of the decision boundaries is substituted with

[3]Later we will see how often the same problem formulation pops up in multivariate analysis

dimensional complexity, that is, the original data space is augmented with a high number of feature variables.

- In the high-dimensional feature space, it is assumed that the patterns are linearly separable from each other, and very efficient linear classification methods are applied.
- The structural information about the samples can be expressed as *kernel matrices,* representing "similarities", being closely connected to association matrices that were studied before.

There are also some differences in interpretations. First, in pattern recognition applications it is assumed that the number of features is huge — typically the number of samples is lower than the number of features, $k \leq n$. Because of this, the kernel matrices, for example, are calculated "horizontally" for the *sample vectors.* Note that the covariance properties remain here essentially the same, only the matrix dimensions become lower.

It is also the objective — classification rather than regression — that means that there are some complications. After all, the adaptation is necessarily nonlinear and iterative: It is only those samples ("support vectors") that are located nearest to the decision boundary that are of essence in classifier training, the other samples are automatically correctly classified. The SVM algorithm maximizes the error margin; it maximizes the minimum distance between the support vectors and the separating hyperplane.

The SVM's are not concentrated on here in more detail. Perhaps it suffices to say that — despite its mathematically simple and elegant ideas, it often outperforms more sophisticated nonlinear approaches.

## Computer exercises

1. You can test the behaviors of different clustering algorithms by using the analysis and data construction routines in `Regression Toolbox` for `Matlab`. For example, you can try the following commands:

```
DATA = dataClust(3,5,50,20,100);   % See "help dataclust"
clustersKM = regrKM(DATA,5);
regrShowClust(DATA,clustersKM);
clustersEM = regrEM(DATA,5);
regrShowClust(DATA,clustersEM);
```

2. Extend the K-means algorithm `regrKM` in the `Regression Toolbox` so that it approximately implements the *Batch-SOM* algorithm. For simplicity, you can restrict to *one-dimensional* maps, so that the clusters $c-1$ and $c+1$ are the nearest neighbors of the cluster number $c$.

# Appendix B

# Structure *into Data*

Above, the data was analyzed to find structures, clusters or classes. In this appendix it is assumed that there already exists some knowledge about the system where the data is coming from, and this structural knowledge is applied to enhance the measurements, to get back to the actual behaviors beyond the noisy observations.

There are different kinds of structures that can be utilized: First, there is the *physical structure,* including not only the actual system structure but also the hierarchic structure determined by the instrumentation of the measurement devices; second, there is the *mathematical structure* as determined by theoretical dependencies among variables; and, third, the observed *a posteriori structure* among the measurements themselves can be utilized. Each of these alternatives is illustrated separately in what follows — what is possible and what is not is very much dependent of the practical system being modeled. Here, only a glimpse into these issues can be given, introducing the challenges and possibilities.

The data is manipulated so that the structural constraints are automatically taken care of when models are constructed for that data. Exploitation of the structures typically introduces new constraints among variables; these constraints become visible in the degrees of freedom in the data. In some cases this reduction in degrees of freedom is reflected as explicit reduction of the data vector dimension. However, if this is not done, explicitly reducing the degrees of dreedom in the data makes the data linearly dependent and collapses the analyses that are based on invertibility of covariance matrices — it turns out that multivariate methods are specially valuable when modeling that data.

## B.1 Data reconciliation

The system structure and and measurement variables are, of course, closely linked together. The system structure and its parameters determine what is being measured; on the other hand, these measurement realizations are used to determine the parameters. This report mainly concentrates on the issue of how to utilize the measurements to determine the system structure (or, at

least, its parameters). In this appendix, the known a priori structure is used to determine (or adjust) the measurements: Understanding of the physical system structure is utilized for trying to reconstruct the actual variables beneath the noisy measurement data — what the data values probably *should* have been. This kind of interference in the actual measurements is called *data reconciliation.*

Assume that vector $\nu$ represents the noisy measurements, and $v$ is the vector of polished variable values after the structural constraints have been taken into account. The problem of finding variable values $v$ near to measurements, subject to a set of (linear) constraints, can be written in the Lagrangian framework as

$$\begin{array}{ll} \text{Minimize} & \frac{1}{2} \cdot (\nu - v)^T R^{-1} (\nu - v) \\ \text{when} & \Gamma v = \gamma. \end{array} \tag{B.1}$$

Here it is assumed that the measurements $\nu$ are distributed normally around the (unknown) correct values $v$ having covariance matrix $R$; minimizing the above criterion gives the maximum likelihood estimates for $v$ (see Chapter 2). The linear constraints are expressed in the form $\Gamma v = \gamma$, where $\Gamma$ and $\gamma$ are a matrix and a vector of compatible sizes.

Note that even if the measurements were exactly correct, delivering the momentary variable values with no error at all, data reconciliation can still be motivated: The measurements only give information of temporary nature, they are not necessarily *representative,* they do not necessarily deliver *essential* information. It is the cumulative effect that is relevant; how the quantity has affected the system behavior over the longer sampling interval.

To apply the Lagrangian methodology, one can first construct the Hamiltonian as

$$J(v) = \frac{1}{2} \cdot (\nu - v)^T R^{-1} (\nu - v) + \mu^T \cdot (\Gamma v - \gamma). \tag{B.2}$$

Note that each constraint equation (as determined by individual rows $i$ in $\Gamma$ and $\gamma$) has a multiplier $\mu_i$ of its own; above, this set of constraints has been collected into a single matrix expression, $\mu_i$'s being collected in vector $\mu$. Minimizing the Hamiltonian gives the following expression for the gradient:

$$\frac{dJ(v)}{dv} = -R^{-1}(\nu - v) + \Gamma^T \mu = 0, \tag{B.3}$$

resulting in

$$v = \nu - R\Gamma^T \mu. \tag{B.4}$$

For eliminating the other unknown $\mu$ from the above expression, one needs to utilize the constraint equation $\Gamma v = \gamma$. Recognizing that when (B.3) is multiplied from the left by $\Gamma R$, the only term with $v$ can be substituted, resulting in

$$\mu = \left(\Gamma R \Gamma^T\right)^{-1} (\Gamma \nu - \gamma), \tag{B.5}$$

so that one finally has (assuming that $\Gamma R \Gamma^T$ is invertible)

$$v = \nu - R\Gamma^T \left(\Gamma R \Gamma^T\right)^{-1} (\Gamma \nu - \gamma). \tag{B.6}$$

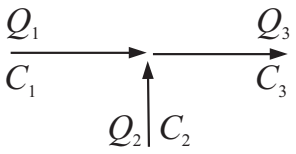



Figure B.1: Three interdependent flows

It is evident that if the measurements $\nu$ fulfill the constraints, so that $\Gamma\nu = \gamma$, these values are directly transferred to $v$, otherwise they are modified according to maximum credibility as expressed in the above formula. Note that the above derivation only modifies data, and constraints directly on the final model parameters cannot be given — see Section B.1.1.

As an example, look at Fig. B.1: Because no accumulation is possible in this subsystem, there must hold

$$Q_1 + Q_2 = Q_3. \tag{B.7}$$

Assuming that the flow values are the only available measurements, so that

$$\nu = \begin{pmatrix} \tilde{Q}_1 & \tilde{Q}_2 & \tilde{Q}_3 \end{pmatrix}^T, \tag{B.8}$$

one has a constraint that can be expressed as

$$\Gamma = \begin{pmatrix} 1 & 1 & -1 \end{pmatrix} \quad \text{with} \quad \gamma = \begin{pmatrix} 0 \end{pmatrix}. \tag{B.9}$$

If there are various measurements of the flows from different time instants, similar constraints have to be written for each time instant separately.

Often the variables are not linearly separable, and the above data reconciliation approaches cannot directly be applied. However, often such problems can still be (approximately) solved. For example, assume that also the concentration values are measured in Fig. B.1, and these values should also be polished. Now the dependencies between variables are highly nonlinear: When the mass balance equations are constructed, in addition to the above volume balance (B.7), one has another weighted average constraint for the solutions:

$$\frac{Q_1C_1 + Q_2C_2}{Q_1 + Q_2} = C_3. \tag{B.10}$$

This expression is nonlinear in variables, if all of the measurements are studied simultaneously; on the other hand, if the problem is divided in two separate (suboptimal) optimization tasks, both of these problems are linear. This means that one first solves for the new values for the flow variables as shown above, and when these values are regarded as fixed, one has in the second phase the measurement vector

$$\nu = \begin{pmatrix} \tilde{C}_1 & \tilde{C}_2 & \tilde{C}_3 \end{pmatrix}^T, \tag{B.11}$$

and the mass balance constraint can then be expressed (using the already fixed values for $Q_i$'s) as

$$\Gamma = \begin{pmatrix} \frac{Q_1}{Q_1+Q_2} & \frac{Q_1}{Q_1+Q_2} & -1 \end{pmatrix} \quad \text{with} \quad \gamma = \begin{pmatrix} 0 \end{pmatrix}. \tag{B.12}$$

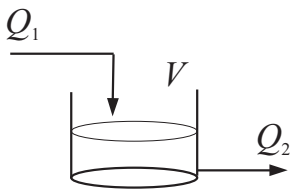



Figure B.2: Dynamics among variables

The final variable vector $v$ can be reconstructed from these data; this approach is not exactly optimal, because the concentration measurements cannot affect the values of the flow variables.

As a more complicated example, study the system in Fig. B.2 that is characterized by the flows $Q_1$ and $Q_2$ and the volume $V$. One knows that the volume at time $\kappa$ is dependent of the net flow, or, more accurately, the *change* in volume, $V(\kappa+1) - V(\kappa)$, is the same as the effective net flow, $Q_1(\kappa) - Q_2(\kappa)$ multiplied by the sampling interval $\Delta t$. If one wants to capture all information that concerns a specific time instant, the data vectors have to be of the form

$$\nu(\kappa) = \begin{pmatrix} \tilde{Q}_1(\kappa) & \tilde{V}(\kappa) & \tilde{Q}_2(\kappa) & \tilde{V}(\kappa+1) \end{pmatrix}^T. \tag{B.13}$$

It is now evident that successive measurement vectors $\nu(\kappa-1)$, $\nu(\kappa)$, $\nu(\kappa+1)$, etc., are linked together because of the shared variables $V$, and the constraint matrix (consisting of a band of non-zero entries) becomes huge:

$$\left(\begin{array}{ccc|ccc|c} & & \vdots & & & & \\ \cdots & \Delta t & 1 \quad -\Delta t & \Delta t & 1 \quad -\Delta t & & \cdots \\ & & \vdots & & & & \end{array}\right) \cdot \left(\begin{array}{c} \vdots \\ \hline Q_1(\kappa) \\ V(\kappa) \\ Q_2(\kappa) \\ \hline Q_1(\kappa+1) \\ V(\kappa+1) \\ Q_2(\kappa+1) \\ \hline \vdots \end{array}\right) = \begin{pmatrix} \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \end{pmatrix}.$$

Note that it is not necessary that all of the quantities needed to construct the structural constraints are measured. The elements in $R$ corresponding to such unmeasured variables can have high values, so that these dummy variables are not weighted. The process can be iterated to reach convergence.

### B.1.1 Explicit constraints on parameters

Above, it was the data that was assumed to have some internal structure; it is also possible that the structure exists among the parameters of the final model. For example, assume that the (dynamic single-output) system being modeled can be expressed in the form

$$y_i(k+1) = ay_i(k) + bu(k), \tag{B.14}$$

or

$$y_i(k+1) = \begin{pmatrix} a \\ b \end{pmatrix}^T \begin{pmatrix} y(k) \\ u(k) \end{pmatrix} = F_i^T x(k), \tag{B.15}$$

and, further,

$$Y_i = XF_i. \tag{B.16}$$

Assuming that we *know* that this system represents an ideal mixer, we know that the steady-state gain of the model must equal 1, meaning that there must hold

$$a + b = 1. \tag{B.17}$$

This can be expressed as

$$\begin{pmatrix} 1 & 1 \end{pmatrix} F_i = 1, \tag{B.18}$$

or, more generally, in the form

$$GF_i = g. \tag{B.19}$$

Here, $G$ can also be a matrix and $g$ can be a vector, assuming that there are various constraints to be matched simultaneously. However, $G$ must have more columns than there are rows — otherwise there are no degrees of freedom left for optimization.

To find a model for data, the same procedure as shown above can be applied for constrained optimization:

$$\begin{array}{ll} \text{Minimize} & \frac{1}{2}\left(Y_i - XF_i\right)^T \left(Y_i - XF_i\right) \\ \text{when} & GF_i = g. \end{array} \tag{B.20}$$

The difference here as compared to the above derivation is that the intended result cannot be assured by data manipulations alone; now the model construction has to be modified. In this sense, the results here differ from other examples in this chapter, but being closely related to data reconciliation, this case is also studied in this context. Differentiating the Hamiltonian

$$\begin{aligned} &\tfrac{d}{dF_i}\left(\tfrac{1}{2}\left(Y_i - XF_i\right)^T \left(Y_i - XF_i\right) - \mu^T\left(GF_i - g\right)\right) \\ &\qquad = X^T X F_i - X^T Y_i - G^T \mu = 0, \end{aligned} \tag{B.21}$$

giving

$$F_i = \left(X^T X\right)^{-1}\left(X^T Y_i + G^T \mu\right). \tag{B.22}$$

To solve for the vector of Lagrange multipliers $\mu$, one can first multiply the above expression from the left by $G$, and observe that according to the constraint this must equal $g$:

$$GF_i = G\left(X^T X\right)^{-1} X^T Y_i + G\left(X^T X\right)^{-1} G^T \mu = g, \tag{B.23}$$

so that

$$\mu = \left(G\left(X^T X\right)^{-1} G^T\right)^{-1}\left(g - G\left(X^T X\right)^{-1} X^T Y_i\right), \tag{B.24}$$

and, finally,

$$F_i = \left(X^T X\right)^{-1} \left( X^T Y_i + G^T \left( G \left(X^T X\right)^{-1} G^T \right)^{-1} \left( g - G \left(X^T X\right)^{-1} X^T Y_i \right) \right). \tag{B.25}$$

From the outlook of this expression one can see that the nominal solution of the least-squares minimization is modified by an additive factor that goes to zero if the nominal solution fulfills the given constraint.

What if a more sophisticated regression approach is to be applied, so that the mapping is to go through a subspace spanned by some matrix $\theta$? Assume that the data is first projected onto the latent basis by the mapping matrix $F^1 = \left(\theta^T \theta\right)^{-1} \theta^T$, just as have been done earlier, so that $Z = XF^1$, but the final mapping from the latent basis to the output, $F_i^2$, is modified from the nominal least-squares fitting so that the overall mapping $F_i = F^1 F_i^2$ fulfills the constraint $GF_i = GF^1 F_i^2 = g$. It is then evident that exactly the above formula (B.25) can be applied if one only selects

$$\begin{array}{lcl} X & \leftarrow & Z = X \left(\theta^T \theta\right)^{-1} \theta^T, \text{ and} \\ G & \leftarrow & GF^1 = G \left(\theta^T \theta\right)^{-1} \theta^T. \end{array} \tag{B.26}$$

The expression (B.25) now actually only gives the mapping $F_i^2$, so that the final result, or the mapping from input directly to output that fulfills the constraints, with $F^1 = \left(\theta^T \theta\right)^{-1} \theta^T$, is

$$F_i = F^1 \left((XF^1)^T XF^1\right)^{-1} \left( (XF^1)^T Y_i + (GF^1)^T \left( GF^1 \left((XF^1)^T XF^1\right)^{-1} (GF^1)^T \right)^{-1} \left( g - GF^1 \left((XF^1)^T XF^1\right)^{-1} (XF^1)^T Y_i \right) \right).$$

## B.2 Observing functional hierarchy

Different quantities are measured in different ways, using different kinds of devices. Typically, there is some measurement error present, and one can try to enhance the quality of the data by utilizing various independent (even though somewhat redundant) devices for measuring the same quantity. The number of measurements grows, but — as has been shown — special means are developed to make the models tolerate high dimensionality.

However, the more or less mechanical approaches that were recommended for preprocessing the wealth of measurements do not take into account the functional structure between individual measurements, and the results may be unoptimal. For example, in Chapter **??** it was said that a good approach to reach well-conditioned data is to make the measurements all have the same variance — then the information available from different channels is best balanced. This is a good rule of thumb — but it can be considerably enhanced if additional information is available.

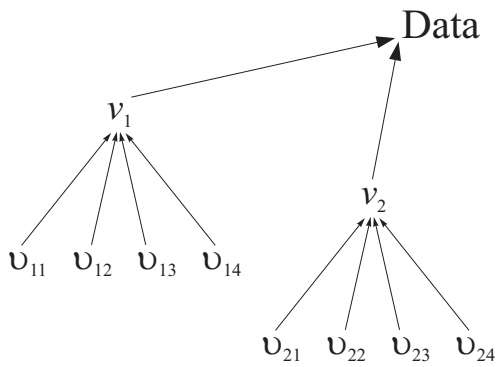



Figure B.3: Hierarchy among measurements

Look at Fig. **??**. It is assumed there that a subset of measurements together try to capture a single physical quantity; the reliability of different measurements (as characterized by the measurement variance) may be different. However, assuming that the devices are independent, all of them deliver some fresh information, and also the less reliable measurements should contribute in the determination of that quantity.

Study an example where $n$ zero-mean measurements of the same quantity $v$ are available, so that $v_i = \nu_i + e_i$ for all $1 \leq i \leq n$. All measurements have characteristic noise properties, so that the variances $\mathrm{var}\{e_i\}$ may vary between measurements; it is assumed that all measurements are unbiased. How should one scale these measurements to reach the best possible estimate $\hat{v} = \sum_{i=1}^{n} w_i v_i$ for $v$? Scaling of the variables means that the variances are also multiplied, so that $\mathrm{var}\{\hat{v}\} = \sum_i \mathrm{var}\{w_i v_i\} = \sum_i w_i^2 \cdot \mathrm{var}\{v_i\}$. Minimization of the overall variance when the sum of weights is fixed, results in the Lagrangian formulation (see page 20)

$$\left\{ \begin{array}{lcl} f(w_1, \ldots, w_n) & = & \sum_{i=1}^{n} w_i^2 \cdot \mathrm{var}\{v_i\}, \quad \text{and} \\ g(w_1, \ldots, w_n) & = & 1 - \sum_{i=1}^{n} w_i. \end{array} \right. \tag{B.27}$$

The former expression tries to minimize variance, whereas the second expression keeps the estimate $\hat{v}$ unbiased, the sum weights equalling unity. This problem formulation gives

$$J(w_1, \ldots, w_n) = \sum_{i=1}^{n} w_i^2 \cdot \mathrm{var}\{v_i\} + \lambda \cdot \left( 1 - \sum_{i=1}^{n} w_i \right), \tag{B.28}$$

so that

$$\left\{ \begin{array}{lcl} \frac{d\, J(w_1, \ldots, w_n)}{d\, w_1} & = & 2w_1 \cdot \mathrm{var}\{v_1\} - \lambda = 0 \\ & \vdots & \\ \frac{d\, J(w_1, \ldots, w_n)}{d\, w_n} & = & 2w_n \cdot \mathrm{var}\{v_n\} - \lambda = 0. \end{array} \right. \tag{B.29}$$

Because $\lambda$ is the same in all of the above equations, it turns out that in the

optimum there must hold for all $i$

$$w_i \cdot \mathrm{var}\{v_i\} = \textit{constant}. \tag{B.30}$$

This is only possible if the weight is inversely proportional to the corresponding error variance. The optimal weighting between the measurements can also be accomplished as

$$\hat{v} = \alpha \cdot \left( \begin{array}{ccc} \frac{1}{\mathrm{var}\{v_1\}} & \cdots & \frac{1}{\mathrm{var}\{v_n\}} \end{array} \right) \cdot \nu. \tag{B.31}$$

Essentially, forgetting the scalar normalization factor $\alpha$, the measurements are divided by the observed variances. It seems that division by the *standard deviations* $\sqrt{\mathrm{var}\{v_i\}}$, or normalization of the measurement variances to unity — as proposed later — is not the best way to determine the measurement scaling in this kind of a *sensor fusion* case, where the subset of measurements are tightly coupled together.

Note that still better estimates could be achieved, if not only the variances, but also the *covariances* between measuring devices were taken into account. In such case, the sensor fusion can best be carried out by applying *principal component analysis* (see Chapter 5) for the subset of measurements alone, or, if there is additionally some dynamics in the measurements, by applying *Kalman filter* (see Chapter 9.1). This means that it is not necessary (not even wise) to do all data processing in a centralized manner; if there are clearly independent data analysis subtasks that can be carried out separately, implementing this kind of hierarchical structure in the data processing enhances the overall system robustness and transparency.

## B.3 Dimensional analysis

In addition to the above considerations concerning physical structure among the measurements, *mathematical structure* can in some cases also be utilized in a (semi)automatic manner.

*Dimensional analysis* utilizes the theoretical compatibility properties among variables having different domains: The units have to match to result in mathematically valid expressions. All of the measurements do have some mathematical structure as expressed in terms of basic SI units — distances are written in meters (m), velocities are written as distances divided by time intervals (m/sec), etc. Only such multiplicative combinations of variables are allowed that make all dimensions among additive terms match with each other.

However, to utilize the above idea in practice, one has to make strong assumptions about structure among the variables. It is assumed that the dependency among the $n$ variables can be written in the following form:

$$\nu_1^{f_1} \cdot \cdots \cdot \nu_n^{f_n} = (\text{dimensionless}) \text{ constant}. \tag{B.32}$$

This means that the model structure has to be *multiplicative,* and no additive terms can be allowed in the model (see Sec. **??**). In this case it is only some

distinct combinations of the exponents (parameters) $f_i$ that make the units compatible in (B.32), and this fact is now extensively utilized: Determine a set of parameters so that their all combinations result in valid expressions.

The ideas of dimensional analysis are best explained through an example. Assume that the pressure drop in a tube, $\Delta p$, is a function of the tube length $l$, its diameter $d$, viscosity of the fluid $\mu$, average speed of the fluid $w$ and its density $\rho$, so that there exists some function

$$g(l, d, \mu, \rho, w, \Delta p) = 0. \tag{B.33}$$

How to find the functional dependency between the variables based on measurement data? To proceed in the spirit of dimensional analysis, one has to assume that — according to (B.32) — that there holds

$$l^{f_1'} \cdot d^{f_2'} \cdot \mu^{f_3'} \cdot \rho^{f_4'} \cdot w^{f_5'} \cdot \Delta p^{f_6'} = \text{(dimensionless) constant.} \tag{B.34}$$

In principle, even though the functional structure has already been considerably constrained, huge amounts of measurements would still be needed to find all six $f_i'$ parameters (see Sec. **??**). Each variable has the unit of its own, though, and they cannot be freely combined; this reduces the degrees of freedom in the search space. Let us study these units:

| **Variable** | **Unit** |
|---|---|
| Length $l$ | $[l] = \mathrm{m}$ |
| Diameter $d$ | $[d] = \mathrm{m}$ |
| Viscosity $\mu$ | $[\mu] = \mathrm{kg} \cdot \mathrm{m}^{-1} \cdot \mathrm{sec}^{-1}$ |
| Density $\rho$ | $[\rho] = \mathrm{kg} \cdot \mathrm{m}^{-3}$ |
| Velocity $w$ | $[w] = \mathrm{m} \cdot \mathrm{s}^{-1}$ |
| Pressure drop $\Delta p$ | $[\Delta p] = \mathrm{kg} \cdot \mathrm{m}^{-1} \cdot \mathrm{sec}^{-2}$. |

In these expressions, only three basic units are found: *meter* m, *kilogram* kg, and *second* sec. It can be shown (as originally shown in [6]) that if there exist $n_0$ basic units among the $n$ variables, the degrees of freedom of the formula can be spanned using only $n - n_0$ artificial *dimensionless* variables: The variables increase the degrees of freedom, whereas each basic unit introduces a constraint of its own, reducing the degrees of freedom by one.

Intuitively, the idea is that if all variables that are manipulated are dimensionless, they can be freely multiplied together without problems emerging due to compatibility; any values for exponents are valid (from the mathematical point of view). In this case, one should find those $6 - 3 = 3$ dimensionless variables $v_i$, so that the same functionality as in (B.33) can be reached in the form

$$g'(v_1, v_2, v_3) = 0, \tag{B.35}$$

or, more specifically, solving (B.34) for $v_3$, for example,

$$v_3 = f_0 \cdot v_1^{f_1} \cdot v_2^{f_2}. \tag{B.36}$$

When searching for the dimensionless variables $v_i$, there are various ways to proceed — the methods and also the results are not unique. One practice that

results in easily manageable expressions is to first select $n-n_0$ variables that one thinks are the most relevant, and have the dimensionless variables specifically reflect these (note that the rest of the variables have to contain all base units). For example, if one now wants to find a model for pressure drop, it is reasonable to select $\Delta p$ among the relevant variables; additionally, let us select $l$ and $w$. This means that, according to [6], the dimensionless variables are constructed as

$$\begin{cases} v_1 = \rho^{\phi_{11}} \mu^{\phi_{12}} d^{\phi_{13}} \cdot l \\ v_2 = \rho^{\phi_{21}} \mu^{\phi_{22}} d^{\phi_{23}} \cdot w \\ v_3 = \rho^{\phi_{31}} \mu^{\phi_{32}} d^{\phi_{33}} \cdot \Delta p. \end{cases} \tag{B.37}$$

The exponents $\phi_{ij}$ are now selected so that the units of $v_i$ become dimensionless:

$$\begin{cases} [v_1] = \left(\mathrm{kgm}^{-3}\right)^{\phi_{11}} \left(\mathrm{kgm}^{-1}\mathrm{sec}^{-1}\right)^{\phi_{12}} (\mathrm{m})^{\phi_{13}} \cdot \mathrm{m} = \mathrm{kg}^0\mathrm{m}^0\mathrm{sec}^0 \\ [v_2] = \left(\mathrm{kgm}^{-3}\right)^{\phi_{21}} \left(\mathrm{kgm}^{-1}\mathrm{sec}^{-1}\right)^{\phi_{22}} (\mathrm{m})^{\phi_{23}} \cdot \mathrm{msec}^{-1} = \mathrm{kg}^0\mathrm{m}^0\mathrm{sec}^0 \\ [v_3] = \left(\mathrm{kgm}^{-3}\right)^{\phi_{31}} \left(\mathrm{kgm}^{-1}\mathrm{sec}^{-1}\right)^{\phi_{32}} (\mathrm{m})^{\phi_{33}} \cdot \mathrm{kgm}^{-1}\mathrm{sec}^{-2} = \mathrm{kg}^0\mathrm{m}^0\mathrm{sec}^0, \end{cases}$$

or

$$\begin{cases} (\mathrm{kg})^{\phi_{11}+\phi_{12}} \cdot (\mathrm{m})^{-3\phi_{11}-\phi_{12}+\phi_{13}+1} \cdot (\mathrm{sec})^{-\phi_{12}} = \mathrm{kg}^0\mathrm{m}^0\mathrm{sec}^0 \\ (\mathrm{kg})^{\phi_{21}+\phi_{22}} \cdot (\mathrm{m})^{-3\phi_{21}-\phi_{22}+\phi_{23}+1} \cdot (\mathrm{sec})^{-\phi_{22}-1} = \mathrm{kg}^0\mathrm{m}^0\mathrm{sec}^0 \\ (\mathrm{kg})^{\phi_{31}+\phi_{32}+1} \cdot (\mathrm{m})^{-3\phi_{31}-\phi_{32}+\phi_{33}-1} \cdot (\mathrm{sec})^{-\phi_{32}-2} = \mathrm{kg}^0\mathrm{m}^0\mathrm{sec}^0. \end{cases}$$

From these one can construct a linear set of equations, and the solution for this set becomes

$$\begin{cases} \phi_{11} = 0 \\ \phi_{12} = 0 \\ \phi_{13} = -1, \end{cases} \quad \begin{cases} \phi_{21} = 1 \\ \phi_{22} = -1 \\ \phi_{23} = 1, \end{cases} \quad \text{and} \quad \begin{cases} \phi_{31} = 1 \\ \phi_{32} = -2 \\ \phi_{33} = 2. \end{cases} \tag{B.38}$$

The dimensionless variables also are

$$\begin{cases} v_1 = \rho^0\mu^0 d^{-1} \cdot l = \frac{l}{d} \\ v_2 = \rho^1\mu^{-1} d^1 \cdot w = \frac{wd\rho}{\mu} \\ v_3 = \rho^1\mu^{-2} d^2 \cdot \Delta p = \frac{\Delta p d^2 \rho}{\mu^2}. \end{cases} \tag{B.39}$$

Expression (B.36) can then be written as

$$\left(\frac{d^2\rho\Delta p}{\mu^2}\right) = f_0 \cdot \left(\frac{l}{d}\right)^{f_1'} \cdot \left(\frac{wd\rho}{\mu}\right)^{f_2'}. \tag{B.40}$$

The original problem has been considerably simplified. The expression can be further reduced if there is additional information available: For example, if it is known that the pressure drop is linearly proportional to the tube length, instead of having two separate variables, one can introduce a new independent variable

$$\xi = \Delta p / l \tag{B.41}$$

having the unit $\mathrm{kgm}^{-2}\mathrm{sec}^{-2}$. After this, there only exist $n = 5$ independent variables, and one only needs two dimensionless variables.

It seems that the dimensionless variable $v_2$ above (accidentally) has the definition of the *Reynold's number* that is familiar from fluid mechanics. This is typical: One often ends up having the same variables when using dimensional analysis — there exist much less freedom among dimensionless variables than there exist dimensioned variables.

It needs to be remembered that this astonishing reduction in the number of variables has its price: First, the assumed functional form (B.34) must be appropriate; second, note that the whole construction collapses if there are, in addition to the variables, some constants that *do have* some dimension. It seems, however, that in fluid mechanics, for example, this kind of assumptions hold, and dimensional analysis is a standard technique in those fields.

Note that in later phases in modeling (as in control engineering in general) the units of the variables are ignored altogether.

### B.3.1 Fixing missing data

The above data manipulations were (more or less) well motivated, because the information that was utilized for modifying the data was additional, received from independent external sources — from our *a priori* understanding of the physical or mathematical structure of the system being studied. This last section here, on the other hand, utilizes for modifying the data *a posteriori* structure, determined using a model that has been estimated (as was explained in earlier chapters) by using that *same data.* This means that the steps of data fixing and model construction become an iterative process with some kind of positive feedback[1].

When using the computer, all data structures have to be filled in, there must be no inhomogeneity in the data. In practice, one often has *missing values* among measurements, meaning that some of the variables $v_i$ are unknown; this may be caused, for example, by measurement problems. In the data such problems are often reflected as outliers (see next chapter), lone samples far from the nominal distribution, and it is reasonable to eliminate such erroneous values from the data. however, if there is scarcity with data, or if a contiguous sequence of data is needed for modeling purposes, it may be reasonable to try and fix the incorrect variable values, not to have a "hole" in the data set.

The traditional approach is to substitute the missing values by some kind of average values, either using the average over the whole data set, or calculating the average between the predecessor and the successor (assuming that the same quantity has been measured various times). However, it is clear that such approximations can be extremely crude, and the data distribution may become distorted, resulting in biased models.

The missing value estimate can also be refined iteratively, so that the model

[1]It needs to be kept in mind that fixing data in this way can be extremely dangerous: Using one's intuition about what the data *should* look like, and using such "tailored" data for modeling, makes the model follow this intuition — however incorrect the assumptions were; fresh data should be used where possible!

will be minimally affected by the errors in the fixed variables. One starts with a crude approximation, and step by step makes that value more appropriate, or less conflicting with the other measurements.

As will be shown later, the models that will be constructed later in this report essentially consist of one single matrix $F$ that tries to map a subset of variables $v_{\text{in}}$ onto another subset of variables $v_{\text{out}}$, so that there should hold $v_{\text{out}} = F^T v_{\text{in}}$. Assume that the regression model with the mapping matrix $\tilde{F}$ has been constructed using crudely fixed, incorrect data. Then the reconstruction error (the data vectors assumedly containing fixed data in some entries) can be written as

$$e = v_{\text{out}} - \tilde{F}^T v_{\text{in}} = \left( \begin{array}{c|c} I_m & -\tilde{F}^T \end{array} \right) \cdot \left( \frac{v_{\text{out}}}{v_{\text{in}}} \right) = Mv. \tag{B.42}$$

Now, one can rearrange the variables in $v$ so that all variables to be fixed are collected on top, no matter if they belong to the input or output variables. Note that the rows in the matrix $M$ also need to be reordered accordingly. This rearranged set of equations can be written as

$$e = \left( \begin{array}{c|c} M_{\text{NO}} & M_{\text{OK}} \end{array} \right) \cdot \left( \frac{v_{\text{NO}}}{v_{\text{OK}}} \right), \tag{B.43}$$

or

$$e = M_{\text{NO}} v_{\text{NO}} + M_{\text{OK}} v_{\text{OK}}. \tag{B.44}$$

The variables in $v_{\text{OK}}$ are assumed to be known *a priori*, whereas the variables in $v_{\text{NO}}$ should be modified to better match the model. The next approximation for the missing variables to be fixed can be found when such new values are selected that the matching error $e$ is minimized, so that one has

$$\begin{aligned} \frac{d\left(e^T e\right)}{d\, v_{\text{NO}}} &= \frac{d}{d\, v_{\text{NO}}} \left(M_{\text{NO}} v_{\text{NO}} + M_{\text{OK}} v_{\text{OK}}\right)^T \left(M_{\text{NO}} v_{\text{NO}} + M_{\text{OK}} v_{\text{OK}}\right) \\ &= \frac{d}{d\, v_{\text{NO}}} \left(v_{\text{NO}}^T M_{\text{NO}}^T M_{\text{NO}} v_{\text{NO}} + v_{\text{NO}}^T M_{\text{NO}}^T M_{\text{OK}} v_{\text{OK}} \right. \\ &\qquad \left. + v_{\text{OK}}^T M_{\text{OK}}^T M_{\text{NO}} v_{\text{NO}} + v_{\text{OK}}^T M_{\text{OK}}^T M_{\text{OK}} v_{\text{OK}}\right) \\ &= 2 M_{\text{NO}}^T M_{\text{NO}} v_{\text{NO}} + 2 M_{\text{NO}}^T M_{\text{OK}} v_{\text{OK}} \\ &= \mathbf{0}. \end{aligned}$$

This gives

$$v_{\text{NO}} = -\left(M_{\text{NO}}^T M_{\text{NO}}\right)^\dagger M_{\text{NO}}^T M_{\text{OK}} v_{\text{OK}}. \tag{B.45}$$

So, having found better approximates for the missing values, a new model can be constructed, where the error due to the missing data should be smaller; this refinement procedure can be continued until the parameters converge. The more there are missing values, the more probably the process converges in some local rather than global minimum. Note that the above pseudoinverse may be rank deficient, if too many variables are unknown in a single sample.

Above, it was assumed that one can concentrate on a single issue at a time — now, the data was fixed utilizing the knowledge of some existing structures,

and it is assumed that other issues, like constructing the actual model, can be concentrated on later, separately. The same step-by-step approach has been employed troughout this report, introducing new ideas only after the motivation for them can be understood; this understanding being engineering-like "hands-on" understanding rather than mathematically exhaustive mastering of details. Of course, some level of iteration cannot be avoided: It is clear that in engineering work iteration is necessary — on both conceptual and practical levels — because more can be found in the underlying issues if there is some understanding of the entity. The linear methods that have been elaborated on facilitate fast execution times, so that different kinds of iterative refinement schemes become feasible.

## Computer exercises

1. Define original data and constraints as follows:

```
v = [1,1,1]';
R = eye(3);
G = [1,1,1];
g = 1;
```

   Vary the covariance matrix changing the first variance between zero and very high values and study the results:

```
R(1,1) = input('Give variance of the first measurement: ');
RegrReconc(v,R,G,g)
```

2. Test the outlier detection using, for example, the following commands:

```
[X,Y] = dataxy(10,2,2);
X(1,1) = 10;
outl([X,Y]);
```

   Iteratively fix the missing value of the previous exercise by the following commands:

```
Wx = ones(size(X));
Wy = ones(size(Y));
Wx(1,1) = 0;
for i = 1:5
   F = mlr(X,Y);
   [X,Y] = fixval(X,Y,F,Wx,Wy)
end
```

   What may happen in the fixing process if there are too many degrees of freedom, say,

```
[X,Y] = dataxy(10,5,5);
X(1,1) = 10;
```

# Part II

# Practical Toolbox

# About the Regression Toolbox

One reason why data-oriented methods have become so popular during the last years is the availability of high-capacity computers and efficient software tools. There is a wealth of alternative tools available — different kinds of tools for different needs.

There exist many ways to categorize the software tools. One characterization goes along specialization: For example, general-purpose programming languages like `C++` and `Java` make it possible to implement any algorithm — if there is enough time available. Specialized program products (like `SIMCA`, etc.) make it easy to do the standard operations on data appropriately; however, one is bound to the ready-to-use routines. `Matlab` is there in between, offering a programming framework for implementing generic algorithms that can directly operate on high-level concepts from linear algebra. Based on this `Matlab` platform, various more or less sophisticated *toolboxes* have been implemented that are tailored for special purposes: For example, the following toolboxes are available, either in public domain or commercially:

- `Statistics Toolbox` (indeed, various versions exist) for statistical data analysis
- `PLS Toolbox` for PCA/PLS modeling, etc.
- `Chemometrics Toolbox` for model calibration, etc.
- `System Identification Toolbox` for analysis of dynamic systems.

Additionally, there exist dozens of toolboxes that concentrate on some specific approach, like `Neural Networks Toolbox`, `Optimization Toolbox`, `FastICA Toolbox`, etc. It seems that as these toolboxes have been deloped by professional domain-area experts, they often are rather unpenetrable: They are difficult to understand, meaning that they are difficult to modify or experiment with.

That is why, there is need for yet another toolbox. The implemented `Regression Toolbox` for `Matlab` specially supports the theoretical derivations discussed in Part 1. The routines are not polished or optimized. But because the used codes are so simple, they can be understood also by a non-expert, and they can be easily experimented with, and extended if needed.

Just as in the theoretical discussions before, the main goal in the Toolbox is to present all methods in a homogeneous framework, and to show how simple all the algorithms (in principle) are. The routines in this Toolbox are not intended for professional use.

# Installation

The `Regression Toolbox` version 1.1 can be downloaded through the address `http://saato14.hut.fi/hyotyniemi/publications/01_report125.htm`.

There are two compressed files, one for the commands, and one for the accompanying data for experimenting. The compressed files have to be uncompressed

using `WinZip`, for example, and the files should be expanded to a separate toolbox folder.

Within the `Matlab` environment, the search path has to be modified so that the toolbox folder is included (preferably in the beginning of the search path). After that, the commands should be operational. The toolbox was developed in the `Matlab` version `5.3` environment, and the compatibility with other versions is *not* guaranteed; however, only the very basic functionality of `Matlab` is used.

Standard `Matlab` style command line `help` scripts are supplied for all routines. No special user interface is available; the commands are run from command line (because of this, the Toolbox *may* be operational also in other versions of `Matlab`).

# Commands at a glance

The `Regression Toolbox` consists of the following commands (summarized here in not in alphabetical but in "logical" order). Each of the commands is explained in more detail in the attached Reference Guide. The `help` command of `Matlab` is available for on-line use.

## Preprocessing commands

- `regrCenter`: Mean centering of data
- `regrScale`: Normalization, variable variances getting scaled
- `regrWeight`: Weighting of data samples
- `regrWhiten`: "Whitening" of data: covariance becomes identity matrix
- `regrFixval`: Iterative fixing of missing data values.

## Cluster management

- `regrFDA`: Fisher Discriminant Analysis for distinguishing between clusters
- `regrForm`: Histogram equalization (or deformation) model construction
- `regrDeform`: Histogram equalization model application
- `regrEM`: Expectation Maximization clustering
- `regrKM`: K-Means clustering of data
- `regrOutl`: Visual outlier detection.

## Structure refinement

- `regrPCA`: Standard Principal Component Analysis
- `regrPLS`: Partial Least Squares analysis, formulated as an eigenproblem
- `regrCR`: Continuum regression basis determination
- `regrCCA`: Canonical Correlation Analysis
- `regrICA`: Independent Component Analysis
- `regrRBFN`: Radial Basis Function Network construction.

## Model construction and regression

- `regrMLR`: Multi-Linear Regression
- `regrMLRC`: Multi-Linear Regression with linear constraints
- `regrOLS`: Orthogonal Least Squares algorithm
- `regrTLS`: Total Least Squares regression
- `regrRR`: Ridge Regression
- `regrRBFR`: Radial Basis Function Regression.

### Functions for dynamic systems

- `regrBal`: Balancing and reducing a dynamic state-space system
- `regrCyb`: Iterative "cybernetic regression"
- `regrIdent`: Black-box identification of ARX models
- `regrSSI`: SubSpace Identification of dynamic systems
- `regrSSSI`: Stochastic SubSpace Identification of dynamic systems.

### Iterative demonstration algorithms

- `regrCYB`: "Cybernetic" adaptation of PCA
- `regrFACTOR`: Factor analysis applying the neocybernetic approach
- `regrHAH`: Hebbian - Anti-Hebbian regression
- `regrPPCA`: PCA using the power method
- `regrGHA`: PCA using the Generalized Hebbian Algorithm
- `regrIICA`: "Interactive" ICA.

### Analysis and visualization

- `regrP`: Fit data against a Gaussian distribution
- `regrCrossval`: Cross-validation of the model
- `regrShowClust`: Visualize the structure of clustered data
- `regrKalman`: Implement discrete-time Kalman filter
- `regrKalm`: Implement stochastic discrete-time Kalman filter
- `regrAskOrder`: Visual tool for model order determination.

### Test material

- `dataXY`: Generate random input-output data
- `dataClust`: Generate random clustered data
- `dataIndep`: Generate data consisting of independent signals
- `dataDigits`: Handwritten digits (Warning: Large file)
- `dataDyn`: Generate random dynamic data
- `dataHeatExch`: Heat exchanger data
- `dataEmotion`: Voice signal data (Warning: Large file).

# dataClust

Function generates random clustered data.

## Syntax

```
[X] = dataclust(n,N,kk,cm,cd)
[X] = dataclust(n,N,kk,cm)
[X] = dataclust(n,N,kk)
[X] = dataclust(n,N)
[X] = dataclust(n)
[X] = dataclust
```

## Input parameters

- `n`: Data dimension (default 3)
- `N`: Number of clusters (default 2)
- `kk`: Data samples in each cluster (default 100)
- `cm`: Deviation of the cluster centers (default 1)
- `cd`: Cluster spread, "longest" axis vs. "shortest" (default 1)

## Return parameter

- `X`: Data matrix (size $kk \cdot N \times n$)

## Comments

The cluster centers are normally distributed around origin, the centers having standard deviation `cm`.

The individual clusters have internal normal distributions determined by parameter `cd`: If this ratio between the distribution principal axes is 1, the clusters are circular. Otherwise, the standard deviations in randomly selected orthogonal directions are determined so that the deviation widths are equally spaced on the logarithmic scale, the ratio between widest and narrowest deviation being `cd`. The determinant of the covariance matrix is always 1.

# dataDigits

Challenging data: Handwritten digits (thanks to Jorma Laaksonen, Dr.Tech.)

## Syntax

```
datadigits
```

## Comments

Running the command defines the $500 \times 256$ matrix `DIGITS` containing 500 samples of handwritten digits in a $16 \times 16$ grid. Each row represents one sample, packed in a vector row by row; this means that the data can be visualized in the following way:

```
digit = DIGITS(index,:);
feature = zeros(16,16);
for j = 1:16
   feature(j,:) = digit((j-1)*16+1:j*16);
end
colormap(gray);
imagesc(feature);
```

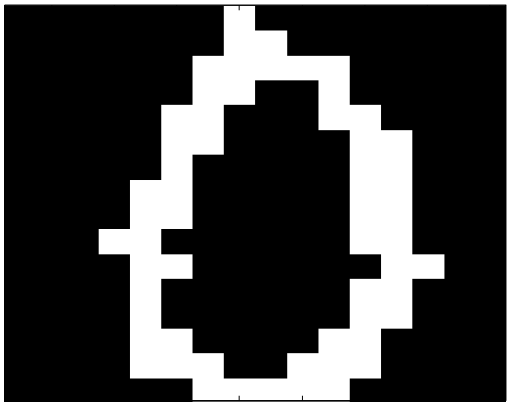


Figure B.4: Index 1: Example of the digit "0"

# dataDyn

Generate random data coming from a state-space system.

## Syntax

```
[U,Y] = datadyn(n,nu,m,k,sigma)
[U,Y] = datadyn(n,nu,m,k)
[U,Y] = datadyn(n,nu,m)
[U,Y] = datadyn(n,nu)
[U,Y] = datadyn(n)
[U,Y] = datadyn
```

## Input parameters

- `n`: State dimension (default 1)
- `nu`: Input dimension (default 1); see below
- `m`: Output dimension (default 1)
- `k`: Number of data samples (default 100)
- `sigma`: Standard deviation of the noise (default 0)

## Return parameters

- `U`: Input sequence (size $k \times \nu$)
- `Y`: Output sequence (size $k \times m$)

## Comments

Function generates sequences of random dynamical data, starting from zero initial condition. Parameter `sigma` determines all the noise processes: The standard deviation of the state noise and the measurement noise.

If the input parameter `nu` is zero, the system has no inputs, being driven by a stochastic process; if `nu` is vector or matrix, this data is directly interpreted as the input data.

# dataEmotion

Command file defines sound signal samples.

## Syntax

```
dataemotion
```

## Comments

There are no expicit inputs or outputs in this command file; running it constructs matrix `DATA` in the workspace. `DATA` contains five sequences of sound signals from different sources; these sources are presented in `DATA` as separate columns.

```
dataemotion;
sound(DATA(:,1),16000);
sound(DATA(:,2),16000);
sound(DATA(:,3),16000);
sound(DATA(:,4),16000);
sound(DATA(:,5),16000);
```

Because no compression of the signals has been carried out, this file is rather large.

# dataHeatExch

Command file defines heat exchanger data.

## Syntax

```
dataheatexch
```

## Comments

There are no expicit inputs or outputs in this command file; running it constructs two matrices `X` and `Y` in the workspace, where the input data `X` stands for temperature measurements along the heat exchanger (see Fig. B.5), and the matrix `Y` is interpreted as follows:

- $Y_1$: Temperature of the incoming cold flow
- $Y_2$: Temperature of the incoming hot flow
- $Y_3$: Temperature of the outgoing cold flow
- $Y_4$: Temperature of the outgoing hot flow.

This data is used, for example, as the training material for the *Regreswsion Course,* and different regression methods can be experimented with and their properties can be compared: A model should be constructed for estimating $y$ when $x$ is given. Note that the causality structure is here blurred — the values in $y$ cannot be interpreted as being functions of $x$, but prediction models can still be implemented.

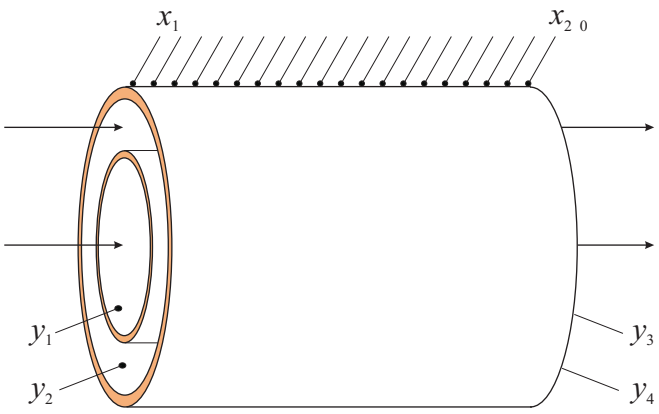



Figure B.5: "Instrumentation" of the heat exchanger

# dataIndep

Function mixes independent data sources.

## Syntax

```
[X] = dataindep(k,func1,func2,func3,func4,func5,func6)
[X] = dataindep(k,func1,func2,func3,func4,func5)
[X] = dataindep(k,func1,func2,func3,func4)
[X] = dataindep(k,func1,func2,func3)
[X] = dataindep(k,func1,func2)
[X] = dataindep(k,func1)
[X] = dataindep(k)
[X] = dataindep
```

## Input parameters

- `k`: Number of data samples (default 1000)
- `func`$i$: If `func`$i$ is string, it is evaluated as a function of time index $k$ (note that in the text this variable is called $\kappa$). There are some functions directly available:
  - `'f1'`: Harmonic, $x_i(k) = \sin(k/5)$
  - `'f2'`: Saw-tooth, $x_i(k) = (\mathrm{rem}(k, 27) - 13)/9$
  - `'f3'`: "Strange curve", $x_i(k) = ((\mathrm{rem}(k, 23) - 11)/9)^5$
  - `'f4'`: Impulsive noise, $x_i(k) = \mathrm{binrand}(k) \cdot \log(\mathrm{unifrand}(k))$, where "binrand" and "unifrand" denote random binary and uniform sequences, giving two alternative values -1 or 1 and continuum of values between 0 and 1, respectively

  Default is `func1='f1'`, `func2='f2'`, `func3='f3'`, and `func4='f4'`.

## Return parameter

- `X`: Data matrix (size $k \times n$, where $n$ is the number of selected functions)

## Comments

Linear mixing, mean centering, and whitening is applied to the set of signals automatrically.

# dataXY

Function generates random input-output data.

## Syntax

```
[X,Y] = dataxy(k,n,m,dofx,dofy,sn,sm)
[X,Y] = dataxy(k,n,m,dofx,dofy)
[X,Y] = dataxy(k,n,m)
[X,Y] = dataxy(k)
[X,Y] = dataxy
```

## Input parameters

- `k`: Number of samples (default 100)
- `n`: Input data dimension (default 5)
- `m`: Output data dimension (default 4)
- `dofx`: Non-redundant input data dimension (default 3)
- `dofy`: Non-redundant output data dimension (default 2)
- `sn`: Input noise level (default 0.001)
- `sm`: Output noise level (default 0.1)

## Return parameters

- `X`: Input data matrix (size $k \times n$)
- `Y`: Output data matrix (size $k \times m$)

## Comments

This data is specially intended for visualizing the differences between MLR, PCR, and PLS regression methods. There is redundancy in $X$, making problems of MLR visible; but not all of the input variation explains the output, so that the difference between PCR and PLS is also demonstrated.

# regrAskOrder

Interactively determine the model order.

## Syntax

```
[N] = regraskorder(LAMBDA)
```

## Input parameter

- `LAMBDA`: Vector of latent vector weights

## Return parameter

- `N`: Selected model order

## Comments

Function plots the values in `LAMBDA` and lets the user select how many of the latent variables will be used in the model construction.

This function is intended to be used only by other routines (`regrPCA`, `regrCCA`, `regrPLS`, `regrICA`, `regrTLS`, `regrBal`, `regrSSI`, and `regrSSSI`) if the model order is not explicitly determined.

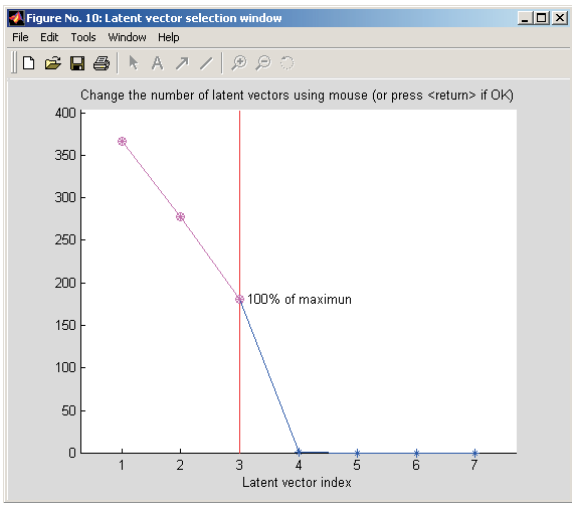



Figure B.6: Selection of the model order. Note that the figure "X% of maximum" only means how much of the absolute *achievable* maximum is captured

# regrBal

State-space dynamic system balancing and reduction.

## Syntax

```
[Ared,Bred,Cred,theta,sigma] = regrbal(A,B,C,N)
[Ared,Bred,Cred,theta,sigma] = regrbal(A,B,C)
```

## Input parameters

- `A`, `B`, `C`: System matrices
- `N`: Number of remaining states (optional)

## Return parameters

- `Ared`, `Bred`, `Cred`: Reduced system matrices
- `theta`: State transformation matrix, `z=theta'*x`
- `sigma`: Hankel singular values

## Comments

If the reduced model order is not given, graphical interaction with the user is used (see `regrAskorder`).

If none of the states is dropped, the model is just balanced.

# regrCCA

Canonical Correlation Analysis (CCA) model construction.

## Syntax

```
[theta,phi,lambda] = regrcca(X,Y,N)
[theta,phi,lambda] = regrcca(X,Y)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `Y`: Output data block (size $k \times m$)
- `N`: Number of latent variables (optional)

## Return parameters

- `theta`: Input block canonical variates
- `phi`: Output block canonical variates
- `lambda`: Canonical correlation coefficients

## Comments

If the number of latent vectors $N$ is not explicitly given, it is queried interactively (see `regrAskOrder`).

Based on this command, the Canonical Correlation Regression can easiest be implemented as

```
F = regrmlr(X,Y,regrcca(X,Y));
Ytest = Xtest*F;
```

# regrCenter

Function for mean centering the data.

## Syntax

```
[X,barX] = regrcenter(DATA,barX)
[X,barX] = regrcenter(DATA)
```

## Input parameter

- `DATA`: Data to be modeled
- `barX`: Point in space included in the model (optional)

## Return parameters

- `X`: Transformed data matrix
- `barX`: Center of DATA

## Comments

Returning to the original coordinates can be carried out as

```
Data = X + barX;
```

# regrCR

Continuum Regression basis determination.

## Syntax

```
[theta,lambda] = regrcr(X,Y,alpha,N)
[theta,lambda] = regrcr(X,Y,alpha)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `Y`: Output data block (size $k \times m$)
- `alpha`: Continuum parameter from $\approx 0$ (MLR) to 1 (PCR) through 0.5 (PLS)
- `N`: Dimension of the latent structure (optional)

## Return parameters

- `theta`: Latent basis vectors
- `lambda`: Corresponding eigenvalues

## Comments

The determination of the CR latent basis is *not* carried out exactly as explained on page 98; the reason is that the powers of an $k \times k$ matrix should be calculated, resulting in an huge eigenvalue problem; on the contrary, a shortcut using singular value decomposition is applied.

If the number of latent vectors $N$ is not explicitly given, it is queried interactively (see `regrAskOrder`).

Based on this command, the actual Continuum Regression can easiest be implemented as

```
F = regrmlr(X,Y,regrcr(X,Y,alpha));
Ytest = Xtest*F;
```

# regrCrossVal

Function for cross-validation of linear regression models.

## Syntax

```
[E] = regrcrossval(X,Y,expr,seqs)
[E] = regrcrossval(X,Y,expr)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `Y`: Output data block (size $k \times m$)
- `expr`: String form expression resulting in $F$, given $X$ and $Y$
- `seqs`: How many cross-validation rounds (default $k$, meaning "leave-one-out" cross-validation approach)

## Return parameter

- `E`: Validation error matrix (size $k \times m$)

## Comments

Cross-validation leaves a set of samples out from the training set, constructs the model using the remaining samples, and tries to estimate the left-out samples using the model.

Parameter `seqs` determines how many continuous separate validation sets are used; default is $k$, meaning that the model construction is carried out $k$ times, always with $k - 1$ samples in the training set.

In this routine, it is assumed that the string `expr`, when evaluated, returns the mapping matrix $F$, so that $Y = X \cdot F$. References within the string to input and output blocks must be `X` and `Y`, respectively, no matter what are the actual symbols.

Remember that validation of the constructed model with fresh samples is crucial — otherwise MLR would always give the seemingly best model, explicitly minimizing the cost criterion for the training data, even though its robustness is weak.

# regrCYB

Function that implements adaptation of data structures in the "neocybernetic model".

## Syntax

```
[A,B,Uhat,Xbar] = regrCYB(U,A,B,lambda,S,nonlin,maskA,maskB)
[A,B] = regrCYB(U,A,B)
```

## Input parameters

- `U`: Input data block (size $k \times \nu$)
- `A`: Feedback matrix A ($n \times n$)
- `B`: Feedforward matrix B ($n \times \nu$)
- `lambda`: Forgetting factor (scalar)
- `S`: Sparsity - how many variables used
- `nonlin`: If non-zero, nonlinear cut modeling
- `maskA`: Masking matrix for A
- `maskB`: Masking matrix for B

## Return parameters

- `A`: Modified feedback matrix A
- `B`: Modified feedforward matrix B
- `Uhat`: Estimate of the input
- `Xbar`: State variable sequence in balance

## Comments

This routine implements the "neocybernetic model" in a simplified form (for more information, see `http://www.control.hut.fi/cybernetics`). In the linear and non-sparse-coded form principal subspace analysis is carried out; if the masking matrix `maskA` in adaptation of `A` is triangular, principal component analysis results (accepting the default, the last feature will represent the most significant principal component direction). Principal subspace analysis is now iterative, no explicit data covariance matrix is constructed; in this sense, this method can even be useful for analysis of very high-dimensional data, for example, when doing image analysis:

```
dataDigits;         % Loading challenging data
n = 10;             % Dimension of latent basis
A = 0.01*eye(n);
B = 0.01*randn(n,size(DIGITS,2));
```

```
                    % Iteration repeated until convergence
[A,B] = regrCYB(DIGITS,A,B,0.8);
[A,B] = regrCYB(DIGITS,A,B,0.8);
...
```

After this, the pattern vectors are stored as columns in the data structure `(inv(A)*B)'`. In the algorithm, the internal dynamics of a neocybernetic model is abstracted away. What is more, the structure is streamline somewhat: Internal iterations are eliminated, making the `Matlab` implementation relatively fast. However, these modifications can result in problems if nonlinearity is employed. Specially, when employing the *cut* nonlinearity (`nonlin = 1`), it can be beneficial to "invert" some of the features; assume that feature number $i$ is stuck in negative weights, so that there never holds $x_i > 0$, one can make it active by writing

```
invert = zeros(n,1); invert(i) = 1;
B = B - 2*(invert*ones(1,size(B,2))).*B;
A = A - 2*(invert*ones(1,size(A,2))).*A;
A = A - 2*(ones(size(A,1),1)*invert').*A;
```

# regrDeForm

Function for equalizing data distributions.

## Syntax

```
[X,W] = regrdeform(DATA,defmatrix)
```

## Input parameters

- `DATA`: Data to be manipulated (size $k \times n$)
- `defmatrix`: Matrix containing deformation information (see `form`)

## Return parameters

- `X`: Data with (approximately) deformed distribution (size $k \times n$)
- `W`: Validity vector containing "1" if measurement within assumed distribution, "0" otherwise

## Comments

Note that the histogram of the deformed data follows the intended distribution only with the resolution that was determined by the number of *bins* in the equalization model construction function `form`. That is, if only 10 bins, for example, are used, histograms plotted with higher resolution will still be unevenly distributed (even for the training data).

# regrEM

Function for clustering using Expectation Maximization algorithm.

## Syntax

```
[clusters] = regrem(X,N,centers,equal)
[clusters] = regrem(X,N,centers)
[clusters] = regrem(X,N)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `N`: Number of clusters
- `centers`: Initial cluster center points (by default determined by K-means, see `km`)
- `equal`: "1" if distributions within clusters are assumed equal (default "0")

## Return parameter

- `clusters`: Vector (size $k \times 1$) showing the clusters (1 to $N$) for samples

## Comments

Sometimes, for difficult combinations of clusters, the procedure may get stuck in the clustering given by K-means algorithm; that is why, the initial centers can be given also explicitly. Also, setting `equal` to "1" makes the algorithm more robust; this can be utilized if there is only affinity difference between clusters.

Only the set membership information is given out; the actual properties for cluster 1, for example, can be calculated as

```
cl = regrem(X,N)
count1 = sum(find(cl==1));
center1 = mean(X(find(cl==1),:))';
covariance1 = X(find(cl==1),:)'*X(find(cl==1),:)/count1 ...
              - center1*center1';
```

# regrFACTOR

Function that implements adaptation of data structures in the "neocybernetic model".

## Syntax

```
[Exu,Uhat,Q] = regrFACTOR(U,Exu,Q,lambda,nonlin,XXref)
[Exu,Uhat,Q] = regrFACTOR(U,Exu,Q)
```

## Input parameters

- `U`: Input data block (size $k \times m$)
- `Exu`: Model matrix (format $n \times m$)
- `Q`: "Stiffnesses" (diagonal matrix $n \times n$)
- `lambda`: Forgetting factor (scalar, default no learning)
- `nonlin`: Nonlinearity applied (default "1"=true)
- `XXref`: Scalar/vector of $x_i$ variances (default levels at 1)

## Return parameters

- `Exu`: Modified model matrix
- `Uhat`: Balance data estimate outside the system
- `Q`: Related to error covariances (diagonal matrix $n \times n$)

## Comments

This routine implements the "neocybernetic model" in a simplified form (for more information, see `http://www.control.hut.fi/cybernetics`). Principal subspace analysis is carried out, and within that subspace, *sparse coding* is searched for. The code below shows this coding for the digit data:

```
% Load hand-written digits
dataDigit; U = DIGITS;

% Parameters
[k,m] = size(U);
n = 16; lambda = 0.9;

% Initialize model structures
Exu = 0.01*randn(n,m);
Q = 1*eye(n);

while 1 == 1
  [Exu,Uhat,Q] = regrFACTOR(U,Exu,Q,lambda,1);
```

```
  % Visualization
  for i = 1:n
    feature = zeros(16,16);
    for j = 1:16, feature(j,:) = Exu(i,(j-1)*16+1:j*16); end
    subplot(sqrt(n),sqrt(n),i); imagesc(feature); colormap('hot'); drawnow;
  end
end
```

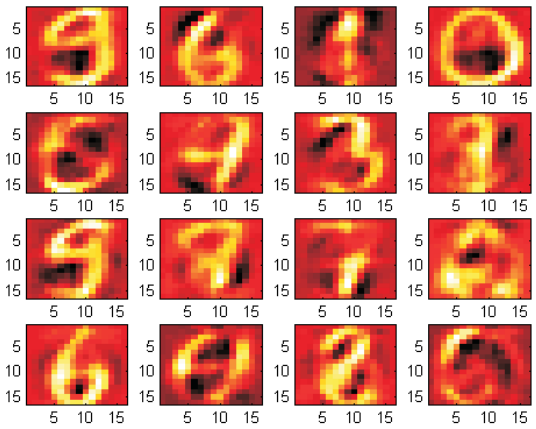


Figure B.7: 16 factors extracted from the handwritten digits

# regrFDA

Fisher Discriminant Analysis (FDA) for discriminating between classes.

## Syntax

```
[theta,lambda] = regrfda(X,clusters,N)
[theta,lambda] = regrfda(X,clusters)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `clusters`: Cluster index between 1 and $N$ for all $k$ samples (size $k \times 1$)
- `N`: Number of discriminant axes (optional)

## Return parameters

- `theta`: Discriminant axes (size $n \times N$)
- `lambda`: Corresponding eigenvalues

## Comments

If the number of discriminant axes $N$ is not explicitly given, it is queried interactively (see `regrAskOrder`).

The vector `clusters` can be constructed, for example, by the EM algorithm (see `em`):

```
X = dataclust(3,2,50,5,5);
clusters = regrem(X,2);
theta = (X,clusters,1);
```

# regrFixVal

Function tries to fix missing values matching data against model.

## Syntax

```
[Xhat,Yhat] = regrfixval(X,Y,F,Wx,Wy)
```

## Input parameters

- `X`: Data matrix to be fixed (size $k \times n$)
- `Y`: Output matrix to be fixed (size $k \times m$)
- `F`: Model matrix, assuming that `Y = X*F`
- `Wx`: Matrix containing "1" for each valid data in X (size $k \times n$)
- `Wy`: Matrix containing "1" for each valid data in Y (size $k \times m$)

## Return parameters

- `Xhat`: Fixed input data matrix (size $k \times n$)
- `Yhat`: Fixed output matrix (size $k \times m$)

## Comments

Function tries to fix uncertain elements in $X$ and $Y$ (as pointed out by zeros in the matrices `Wx` and `Wy`, respectively) to have more plaussible values, so that the reconstruction error $E = Y - XF$ would be minimized. This procedure can be repeated as meny times as needed:

```
for i = 1:10
   F = mlr(Xhat,Yhat);
   [Xhat,Yhat] = regrfixval(Xhat,Yhat,F,Wx,Wy);
end
```

# regrForm

Function for "equalizing", finding mapping between distributions.

## Syntax

```
[defmatrix] = regrform(DATA,normdist)
[defmatrix] = regrform(DATA,bins)
[defmatrix] = regrform(DATA)
```

## Input parameters

- `DATA`: Data to be manipulated (size $k \times n$)
- `normdist` for *vector argument*: Intended distribution form
- `bins` for *scalar argument*: Number of "bins", data sections (default distribution being Gaussian between $-3\sigma$ to $3\sigma$, number of bins being 10 if no argument is given)

## Return parameters

- `defmatrix`: Matrix containing deformation information. For each variable (column) of `DATA`, there are the starting points of the data bins, and between these there is the "steepness" of the distribution within the bin; there are this kind of bin/steepness pairs as many as there are bins (and, additionally, the end point of the last bin), altogether the matrix size being $2 \cdot \texttt{bins} + 1 \times n$.

## Comments

Too special distributions, or if there are too many bins as compared to the number of data, may result in difficulties in the analysis. These failure situations are stochastic, being caused by a random process dividing samples in bins: This means that retrial may help.

# regrGHA

Principal Component Analysis using Generalized Hebbian Algorithm.

## Syntax

```
[theta,lambda] = regrgha(X,N,gamma,epochs)
[theta,lambda] = regrgha(X,N,gamma)
[theta,lambda] = regrgha(X,N)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `N`: Number of latent variables to be extracted
- `gamma`: Step size (default $\gamma = 0.001$)
- `epochs`: Number of iterations (default 10)

## Return parameters

- `theta`: Extracted eigenvectors (size $n \times N$)
- `lambda`: Corresponding eigenvalues

## Comments

As compared to `regrPCA` or even `regrPPCA`, convergence here is slow; however, no explicit covariance matrix needs to be constructed.

# regrICA

Eigenproblem-form Independent Component Analysis (ICA).

## Syntax

```
[theta,lambda] = regrica(X,alpha,N)
[theta,lambda] = regrica(X,alpha)
[theta,lambda] = regrica(X)
```

## Input parameters

- `X`: Input (whitened) data block, mixture of signals (size $m \times n$)
- `alpha`: Type of signals being searched (default 1)
- `N`: Number of latent vectors (default all)

## Return parameters

- `theta`: Independent basis vectors
- `lambda`: Corresponding eigenvalues

## Comments

This function calculates independent components using an eigenproblem-oriented approach called FOBI. The (prewhitened) data is modified so that the anomalies in the distribution become visible in the second-order properties; this is accomplished in the following way:

$$x'(\kappa) = \|x(\kappa)\|^{\alpha} \cdot x(\kappa)$$

It can be shown that for $\alpha = 1$ the standard, kurtosis-based independent component analysis results. Different values of $\alpha$ may emphasize different statistical moments.

For $\alpha = 1$, the kurtosis in direction $\theta_i$ can be calculated as $\lambda_i - n - 2$.

# regrIdent

Identify SISO system recursively.

## Syntax

```
[f] = regrident(u,y,n,lambda,f)
[f] = regrident(u,y,n,lambda)
[f] = regrident(u,y,n)
```

## Input parameters

- `u`: Scalar input sequence (size $k \times 1$)
- `y`: Scalar output sequence (size $k \times 1$)
- `n`: System dimension
- `lambda`: Forgetting factor (default 1, no forgetting)
- `f`: Initial parameter vector (default zero)

## Return parameters

- `f`: Final parameter vector

## Comments

Very simple SISO identification algorithm based on the ARX model structure. The structure of the model is

$$y(\kappa) = a_1 y(\kappa - 1) + \cdots + a_n y(\kappa - n) + b_1 u(\kappa - 1) + \cdots + b_n u(\kappa - n),$$

and the parameter vector is

$$f = \begin{pmatrix} a_1 \\ \vdots \\ a_n \\ b_1 \\ \vdots \\ b_n \end{pmatrix}.$$

# regrIICA

"Interactive" Independent Component Analysis (ICA).

## Syntax

```
[theta] = regriica(X)
```

## Input parameter

- `X`: Input (whitened) data block, mixture of signals (size $k \times n$)

## Return parameters

- `theta`: Independent basis vectors

## Comments

The method presented in Sec. 7.3.3 results, in principle, in $n \cdot (n^2 + n)/2$ provisional independent components. This function lets the user interactively select those that are most interesting.

# regrKalm

Discrete time stochastic Kalman filter.

## Syntax

```
[Xhat,Yhat] = regrkalm(Y,A,C,Rxx,Ryy,Rxy,x0)
[Xhat,Yhat] = regrkalm(Y,A,C,Rxx,Ryy,Rxy)
```

## Input parameters

- `Y`: Output data block (size $k \times m$)
- `A`, `C`, `Rxx`, `Ryy`, and `Rxy`: System matrices
- `x0`: Initial state (default $\mathbf{0}$)

## Return parameters

- `Xhat`: State sequence estimate
- `Yhat`: Corresponding output estimate

## Comments

The assumed stochastic system structure is

$$\begin{cases} x(\kappa+1) = Ax(\kappa) + \epsilon(\kappa) \\ y(\kappa) = Cx(\kappa) + e(\kappa), \end{cases}$$

where the white noise sequences $\epsilon(\kappa)$ and $e(\kappa)$ are characterized by the covariance matrices $\mathrm{E}\{\epsilon(\kappa)\epsilon^T(\kappa)\} = R_{xx}$, $\mathrm{E}\{e(\kappa)e^T(\kappa)\} = R_{yy}$, and $\mathrm{E}\{\epsilon(\kappa)e^T(\kappa)\} = R_{xy}$.

This function augments the stochastic model and calls `regrKalman` function.

# regrKalman

Discrete time Kalman filter.

## Syntax

```
[Xhat,Yhat] = regrkalman(U,Y,A,B,C,D,Rxx,Ryy,Rxy,x0)
[Xhat,Yhat] = regrkalman(U,Y,A,B,C,D,Rxx,Ryy,Rxy)
```

## Input parameters

- `U`: Input data block (size $k \times n$)
- `Y`: Output data block (size $k \times m$)
- `A`, `B`, `C`, `D`, `Rxx`, `Ryy`, and `Rxy`: System matrices
- `x0`: Initial state (default $\mathbf{0}$)

## Return parameters

- `Xhat`: State sequence estimate
- `Yhat`: Corresponding output estimate

## Comments

The assumed stochastic system structure is

$$\begin{cases} x(\kappa+1) = Ax(\kappa) + Bu(\kappa) + \epsilon(\kappa) \\ y(\kappa) = Cx(\kappa) + Du(\kappa) + e(\kappa), \end{cases}$$

where the white noise sequences $\epsilon(\kappa)$ and $e(\kappa)$ are characterized by the covariance matrices $\mathrm{E}\{\epsilon(\kappa)\epsilon^T(\kappa)\} = R_{xx}$, $\mathrm{E}\{e(\kappa)e^T(\kappa)\} = R_{yy}$, and $\mathrm{E}\{\epsilon(\kappa)e^T(\kappa)\} = R_{xy}$. The implementation of the procedure is very elementary — for example, the Riccati equation is solved using a fixed-length iteration, meaning that for badly conditioned matrices the results can be inaccurate.

# regrKM

Function for determining the clusters using the K-means algorithm.

## Syntax

```
[clusters] = regrkm(X,N,centers)
[clusters] = regrkm(X,N)
```

## Input parameters

- `X`: Data to be modeled (size $k \times n$)
- `N`: Number of clusters
- `centers`: Initial cluster center points (optional)

## Return parameter

- `clusters`: Vector (size $k \times 1$) showing the clusters for samples

## Comments

If no initial cluster centers are given, the $N$ first samples are used as centers.

# regrMLR

Regression from X to Y, perhaps through a latent basis.

## Syntax

```
[F,error,R2,stds] = regrmlr(X,Y,theta)
[F,error,R2,stds] = regrmlr(X,Y)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `Y`: Output data block (size $k \times m$)
- `theta`: Latent basis, orthogonal or not (optional)

## Return parameters

- `F`: Mapping matrix, $\hat{Y} = X \cdot F$
- `error`: Prediction errors (size $k \times m$)
- `R2`: Data fitting criterion $R^2$
- `stds`: Estimated standard deviations of the parameters ($n \times m$)

## Comments

This is the basic regression function in the Toolbox, used by most of the other regression methods.

If no $\theta$ is given, least-squares mapping from $X$ to $Y$ is constructed; otherwise, the data is first projected onto the latent basis, no matter how it has been constructed.

The matrix `stds` has the same structure as the model matrix $F$, revealing the estimated accuracies of the parameters. Depending on the assumed probability distribution of the error, one can study whether the parameter value "0" is plausible. If there is the latent structure $\theta$ defined, standard deviations are not calculated.

# regrMLRC

Regression from X to Y, perhaps through a latent basis, when there are additional linear constraints for parameters.

## Syntax

```
[F,error,R2,stds] = regrMLRC(X,Y,G,g,theta)
[F,error,R2,stds] = regrMLRC(X,Y,G,g)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `Y`: Output data block (size $k \times m$)
- `G` and `g`: Linear constraints in the form $GF = g$
- `theta`: Latent basis, orthogonal or not (optional)

## Return parameters

- `F`: Mapping matrix, $\hat{Y} = X \cdot F$
- `error`: Prediction errors (size $k \times m$)
- `R2`: Data fitting criterion $R^2$
- `stds`: Estimated standard deviations of the parameters ($n \times m$)

## Comments

If no $\theta$ is given, least-squares mapping from $X$ to $Y$ is constructed; otherwise, the data is first projected onto the latent basis, no matter how it has been constructed. In both cases, the final mapping between the input and the output fulfills the given constraints.

# regrOLS

Orthogonal Least Squares (OLS).

## Syntax

```
[F,error] = regrols(X,Y)
```

## Input parameters

- X:Input data block (size $k \times n$)
- Y: Output data block (size $k \times m$)

## Return parameters

- F: Mapping matrix, $\hat{Y} = X \cdot F$
- error: Prediction errors

## Comments

Model construction is carried out using the QR factorization of $X$.

# regrOutl

Interactive elimination of outliers.

## Syntax

```
[W] = regroutl(X,W)
[W] = regroutl(X)
```

## Input parameters

- X: Data to be modeled (size $k \times n$)
- W: Old vector (size $k \times 1$) containing "1" for valid samples (optional)

## Return parameters

- W: New vector containing "1" for valid data (size $k \times 1$)

## Comments

Function eliminates outliers interactively. Feedback is given on the graphical screen — mouse clicks toggle the status of the nearest point ("1" for valid data and "0" for invalid), and the vector of these values W is returned.

Vector W can also be refined if it is given as input argument.

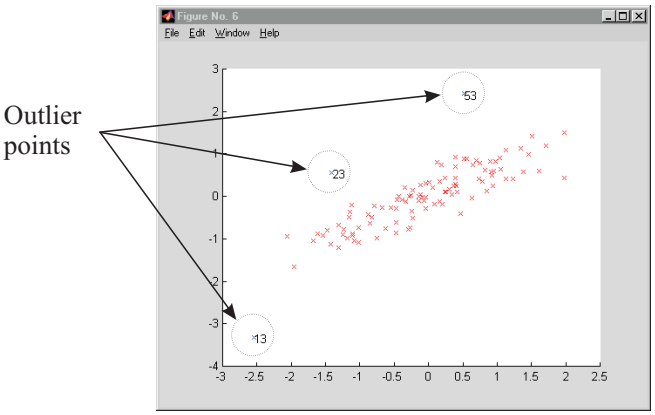



Figure B.8: Outliers, lone samples

# regrP

Fitting data against a Gaussian distribution.

## Syntax

```
[p] = regrP(X1,X2)
```

## Input parameters

- `X1`: Input data block determining the distribution (size $k_1 \times n$)
- `X2`: Data block to be fitted (size $k_2 \times n$)

## Return parameters

- `p`: Probabilities for data in `X2` to match `X1`

## Comments

For each vector in `X2`, a probability value is returned — how well that vector matches the assumedly Gaussian distributions determined by data in `X1`, or "how probably a vector truly belonging to that distribution is farther from the center than vector in `X2` is".

Assuming that $x$ has normal distribution, the measure $x^T \mathrm{E}\{xx^T\}^{-1}x$ has $\chi^2$ distribution, and this understanding is exploited.

# regrPCA

Principal Component Analysis (PCA) model construction.

## Syntax

```
[theta,lambda,Q,T2] = regrpca(X,N)
[theta,lambda,Q,T2] = regrpca(X)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `N`: Number of latent variables (optional)

## Return parameters

- `theta`: Latent basis
- `lambda`: Variances in latent basis directions
- `Q`,`T2`: Fitting criteria $Q$ and $T^2$ (sizes $k \times 1$)

## Comments

If the number of latent vectors $N$ is not explicitly given, it is queried interactively (see `regrAskOrder`).

Based on this command, the Principal Component Regression can easiest be implemented as

```
F = regrmlr(X,Y,regrpca(X));
Ytest = Xtest*F;
```

# regrPLS

Partial least Squares (PLS) model construction.

## Syntax

```
[theta,phi,lambda] = regrpls(X,Y,N)
[theta,phi,lambda] = regrpls(X,Y)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `Y`: Output data block (size $k \times m$)
- `N`: Number of latent variables (optional)

## Return parameters

- `theta`: Input block latent basis
- `phi`: Output block latent basis
- `lambda`: Correlation coefficients (unnormalized)

## Comments

If the number of latent vectors $N$ is not explicitly given, it is queried interactively (see `regrAskOrder`).

Based on this command, the actual regression can easiest be implemented as

```
F = regrmlr(X,Y,regrpls(X,Y));
Ytest = Xtest*F;
```

# regrPPCA

Principal Component Analysis using iterative power method.

## Syntax

```
[Z,L,iter] = regrppca(X,N,iterlimit)
[Z,L,iter] = regrppca(X,N)
[Z,L,iter] = regrppca(X)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `N`: Number of latent variables (optional)
- `iterlimit`: Maximum number of iterations (optional)

## Return parameters

- `theta`: Eigenvectors
- `lambda`: Corresponding eigenvalues
- `iter`: Number of iterations needed

## Comments

If there are eigenvectors with practically same eigenvalues, the convergence may be slow.

# regrRBFN

Radial Basis Function regression model construction.

## Syntax

```
[rbfmodel,error] = regrrbfn(X,Y,clusters,sigma)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `Y`: Output data block (size $k \times m$)
- `clusters`: Cluster index between 1 and $N$ for all $k$ samples, constructed, for example, by K-means (see `km`)
- `sigma`: Distribution of the Gaussians

## Return parameters

- `rbfmodel`: Matrix containing the model (see below)
- `error`: Prediction errors

## Comments

The structure of `rbfmodel` is the following:

```
rbfmodel = [centers;sigmas;weights]
```

where

- `centers`: Vector (size $n \times N$) containing cluster centers
- `sigmas`: Standard deviations in the clusters (size $1 \times N$)
- `weights`: Mappings (size $m \times N$) from clusters to outputs

This routine can be used as:

```
model = regrrbfn(X,Y,regrkm([X,Y],N));
Yhat = regrrbfr(X,model);
```

# regrRBFR

Radial Basis Function regression.

## Syntax

```
[Yhat] = regrrbfr(X,rbfmodel)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `rbfmodel`: Matrix containing the model

## Return parameters

- `Yhat`: Estimated output data block (size $k \times m$)

## Comments

See `regrRBFN`.

# regrReconc

Data reconciliation.

## Syntax

```
[v] = regrreconc(v,R,G,g)
```

## Input parameters

- v: Data vector (size $n \times 1$)
- R: Measurement error covariance matrix (size $n \times n$)
- G and g: Linear constraints in the form $Gv = g$

## Return parameter

- v: Modified data vector

## Comments

After this operation, the data vector fulfills the given set of linear constraints.

# regrRR

Ridge Regression.

## Syntax

```
[F,error] = regrrr(X,Y,q)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `Y`: Output data block (size $k \times m$)
- `q`: "Stabiliaztion factor"

## Return parameters

- `F`: Mapping matrix, so that $\hat{Y} = X \cdot F$
- `error`: Prediction errors

# regrScale

Function for normalizing data variances.

## Syntax

```
[X,backmap] = regrscale(DATA,w)
[X,backmap] = regrscale(DATA)
```

## Input parameters

- `DATA`: Data to be manipulated (size $k \times n$)
- `w`: Intended variable variances or "importances" (optional)

## Return parameters

- `X`: Scaled data matrix (size $k \times n$)
- `backmap`: Matrix for getting back to original coordinates

## Comments

The original coordinates are restored as

```
DATA = X*backmap;
```

# regrShowClust

Function for visualization of clusters.

## Syntax

```
regrshowclust(X,c)
regrshowclust(X)
```

## Input parameters

- `X`: Data matrix (size $k \times n$)
- `c`: Cluster indices for data (optional)

## Comments

Samples classified in different classes are shown in different colours. If only one argument is given, one cluster is assumed.

The user is interactively asked to enter (in `Matlab` list form) the principal axes onto which the data is projected. If only one number is given, the horizontal axis is time $k$; if two or three axes are given, a two-dimensional or three-dimensional plot is constructed. The three-dimensional view can be rotated using mouse.

# regrSSI

Combined stochastic-deterministic SubSpace Identification (simplified)

## Syntax

```
[A,B,C,D,Rxx,Ryy,Rxy] = regrssi(U,Y,maxdim,N)
[A,B,C,D,Rxx,Ryy,Rxy] = regrssi(U,Y,maxdim)
```

## Input parameters

- `U`: Input data block (size $k \times n$)
- `Y`: Output data block (size $k \times m$)
- `maxdim`: Assumed maximum possible system dimension
- `N`: System dimension (optional)

## Return parameters

- `A`, `B`, `C`, `D`: System matrices
- `Rxx`, `Ryy`, `Rxy`: Noise covariances

## Comments

If the number of states $N$ is not explicitly given, it is queried interactively (see `regrAskOrder`). The implementation of the ideas of subspace identification is very simple, so that internally it is simple PCA that is applied, and Ridge Regression is used for carrying out the internal mappings.

The assumed system structure is compatible with the following structure:

$$\begin{cases} x(\kappa+1) = Ax(\kappa) + Bu(\kappa) + \epsilon(\kappa) \\ y(\kappa) = Cx(\kappa) + Du(\kappa) + e(\kappa), \end{cases}$$

where the white noise sequences $\epsilon(\kappa)$ and $e(\kappa)$ are characterized by the covariance matrices $\mathrm{E}\{\epsilon(\kappa)\epsilon^T(\kappa)\} = R_{xx}$, $\mathrm{E}\{e(\kappa)e^T(\kappa)\} = R_{yy}$, and $\mathrm{E}\{\epsilon(\kappa)e^T(\kappa)\} = R_{xy}$.

# regrSSSI

Stochastic SubSpace Identification (simplified)

## Syntax

```
[A,C,Rxx,Ryy,Rxy] = regrsssi(Y,maxdim,N)
[A,C,Rxx,Ryy,Rxy] = regrsssi(Y,maxdim)
```

## Input parameters

- `Y`: Output data block (size $k \times m$)
- `maxdim`: Assumed maximum possible system dimension
- `N`: System dimension (optional)

## Return parameters

- `A`, `C`: System matrices
- `Rxx`, `Ryy`, `Rxy`: Noise covariances

## Comments

If the number of latent states $N$ is not explicitly given, it is queried interactively (see `regrAskOrder`). The implementation of the ideas of subspace identification is very simple, so that internally it is simple PCA that is applied, and Ridge Regression is used for carrying out the internal mappings.

The assumed system structure is compatible with the stochastic Kalman filter (see `regrKalman`):

$$\begin{cases} x(\kappa+1) = Ax(\kappa) + \epsilon(\kappa) \\ y(\kappa) = Cx(\kappa) + e(\kappa), \end{cases}$$

where the white noise sequences $\epsilon(\kappa)$ and $e(\kappa)$ are characterized by the covariance matrices $\mathrm{E}\{\epsilon(\kappa)\epsilon^T(\kappa)\} = R_{xx}$, $\mathrm{E}\{e(\kappa)e^T(\kappa)\} = R_{yy}$, and $\mathrm{E}\{\epsilon(\kappa)e^T(\kappa)\} = R_{xy}$.

# regrTLS

Total Least Squares regression.

## Syntax

```
[F,error] = regrtls(X,Y)
```

## Input parameters

- `X`: Input data block (size $k \times n$)
- `Y`: Output data block (size $k \times m$)

## Return parameters

- `F`: Mapping matrix, $\hat{Y} = X \cdot F$
- `error`: Prediction errors (size $k \times m$)

# regrWeight

Function for weighting the data.

## Syntax

```
[X] = regrweight(DATA,w)
```

## Input parameters

- `DATA`: Data to be manipulated (size $k \times n$)
- `w`: Data sample importances (size $k \times 1$)

## Return parameters

- `X`: Weighted data matrix (size $k \times n$)

## Comments

This function makes it possible to condition heteroscedastic data. Assuming that *invs* contains the inverses of the sample-wise a priori error variances, the following formulations can be employed:

```
theta = regrpca(regrweight(X,sqrt(invs)));
F = regrmlr(regrweight(X,sqrt(invs)),regrweight(Y,sqrt(invs)),theta);
```

# regrWhiten

Function for whitening the data.

## Syntax

```
[X,backmap] = regrwhiten(DATA)
```

## Input parameters

- `DATA`: Data to be manipulated (size $k \times n$)

## Return parameters

- `X`: Scaled data matrix (size $k \times n$)
- `backmap`: Matrix for getting back to original coordinates

## Comments

The original coordinates are restored as

```
DATA = X*backmap;
```