

Linear Models and Variance Analysis

Second version, work in progress

Peter H. Gruber

January 18, 2018

Contents

1	— Introduction	4
	1.1. The two topics of this course 1.2. Why study linear models? 1.3. About this script	
2	— A recap of matrix algebra	9
	2.1. Introduction 2.2. Matrices, vectors and scalars 2.3. Special matrices 2.4. Ma- trix calculation rules 2.5. Frequently used tricks with vectors 2.6. Solving matrix equations 2.7. Exercises	
3	— An interactive introduction to MATLAB	23
	3.1. Why MATLAB? 3.2. Getting started 3.3. The main elements of the MAT- LAB language 3.4. PC-Lab: An interactive introduction to MATLAB 3.5. Get- ting help 3.6. Exercises	
4	— Some multivariate probability theory and statistics	34
	4.1. *Partitions and the concept of a random variable 4.2. Random vectors and matrices 4.3. Expectations and Covariances 4.4. Exercises	
5	— Some multivariate statistics in MATLAB	45
	5.1. One-dimensional case 5.2. Multidimensional case 5.3. PC-Lab: Data han- dling and descriptive statistics 5.4. Some sources of data 5.5. Exercises	
6	— The linear model and OLS	52
	6.1. Derivatives of vectors and matrices 6.2. The linear model in matrix notation 6.3. Least squares 6.4. Estimators 6.5. Multiple linear regression model: Assumptions 6.6. Statistical inference and testing 6.7. Scale dependence of the β coefficients 6.8. PC-Lab: Linear equations and OLS 6.9. Exercises	
7	— Monte Carlo simulation	66
	7.1. An introduction to simulation-based methods 7.2. A first Monte Carlo example 7.3. The Monte Carlo Method 7.4. PC-Lab: Simulation and estimation 7.5. Exercises	

Contents

8	— Spectral theory	77
	8.1. Eigenvalues and eigenvectors 8.2. Matrix decompositions 8.3. Condition number 8.4. PC-Lab 8.5. Exercises	
9	— Worked problem: PCA and the term structure of interest rates	84
	9.1. Motivation 9.2. The method 9.3. Empirical results 9.4. PC Lab: PCA in MATLAB	

1 Introduction

The world is highly diverse and complex. For ages, man has tried to explain the world around him and to predict future events. While some of this research activity can be explained by pure curiosity, there is also a clear value in it: explaining the world helps to make better decisions; predicting it helps to prepare in time for future events.

Economics has started as a science for understanding economic activities. In the last 50 years, it has widened its field into studying all types of interactions between individuals and/or organizations. The aim of economics is to reduce the complexity of the world to explain a phenomenon with a few – the most important – determinants, e.g. in order to devise a policy. Economists have a clear picture as to how the world works and how it should be analyzed:

First of all, phenomena have to be observed and measured. Any quantity that is not observable or that cannot (at least in theory) be measured, is of no relevance to economics.¹

Second, an economic model of some relation between phenomena is formulated. In the most general case, this takes the form $Y = f(X)$, meaning that the phenomena Y and X occur in a certain relation. We usually call this relationship “ Y depends on X ”. Economists use the word “cause” with care. Although there are a few concepts to explain *causality*², most economic research today exploits only statistical *coincidence* and gives a rationale for causality in a separate step.

Note that the economic model $Y = f(X)$ is an *assumption*. As we do not know how the world works, there is no alternative to making assumptions. The fact that any economic model is an assumption, makes the next step – testing the model – very important.

Third, the model is estimated and hypotheses implied by it are tested. The beauty of econometrics lies in the fact that it knows its boundaries. Whenever we do econometrics, we do not only get an estimate for a certain parameter, we also get an estimate for the ac-

¹This explains why economists such as Frey/Stutzer have put great effort in quantifying such abstract concepts as happiness or well-being.

²Most noteworthy the concept of Granger causality in time series econometrics.

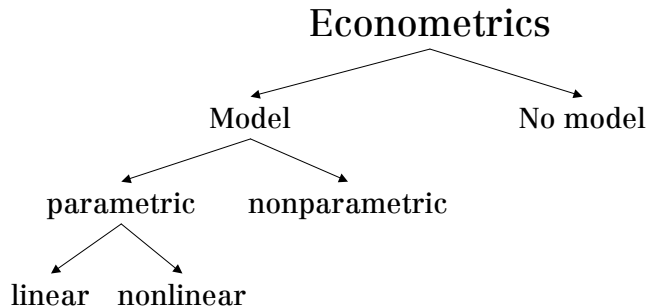


Figure 1.1: Overview of econometric methods. OLS and other linear models are on the leftmost; variance analysis as an exploratory method is on the rightmost.

curacy of this result. An econometrician’s weather forecast would look like this: “Tomorrow the sun is going to shine, and I am 90% sure about it.”

Economic theory follows essentially down the same lines, except that it only cares that the measurements, estimations and tests are theoretically feasible.

One aim of economic modelling is parsimony – explaining a phenomenon with the few most important variables. Today, we are sometimes confronted with the fact that we have too much data and that we have to identify the most important factors that are in the data. Variance analysis is one way to cope with the problem of too much data.

1.1 The two topics of this course

Econometric methods can roughly be divided as shown in Fig. 1.1. The first question is whether the analysis should be based on a model. Most econometric analysis is based on an economic model and tries to verify and quantify it. However, we have to keep in mind that any economic model is an assumption. If the model is misspecified, it may yield misleading results.

Among the models, we can discern parametric models, in which the functional form of $f(\cdot)$ in $Y = f(X)$ is explicitly given as a part of the model. Nonparametric models, on the other hand, still assume that Y depends *in some way* on X , but they leave the functional form of $f(\cdot)$ unspecified and produce results directly from the data. Parametric models, finally, can be divided in linear and non-linear models.

If there is no economic model or if it is not clear that the current model is correct, an alternative route may be pursued. This so-called model-free or exploratory approach aims

1 Introduction

at letting the data speak for itself. It has the clear advantage that it will not over-interpret the facts by forcing a misspecified model on the data. On the other hand, the insights that we can get with such an approach are limited. The principal component analysis (PCA) as one major model-free method delivers the number of factors that drive a certain phenomenon. The PCA therefore delivers a valuable hint for further modelling, but it does not explain the phenomenon itself.

This course aims at strengthening and widening the student's toolboxes, both on the analytical and numerical side. The unifying theme of this course is matrix algebra, used both in linear parametric models and in the model-free methods presented here.

1.2 Why study linear models?

The "linear" in linear model refers to the linearity in the parameters β_i . The explanatory variables need not to be linear. This allows for a great deal of flexibility to model the relationship $Y = f(X)$. All of the following models are linear models:

$$\begin{aligned}y &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + u \\y &= \beta_0 + \beta_1 x_1^2 + \beta_2 \ln x_2 + u \\y &= \beta_0 + \beta_1 x_1 + \beta_2 x_1 x_2 + u\end{aligned}\tag{1.1}$$

Many problems that seem non-linear at first sight actually fit into the class of linear models. In other cases, linear models are a good approximation, because any differentiable function can be approximated locally with a linear function.

The key advantage of linear models is their tractability. Matrix algebra is a very powerful tool that makes it possible to derive almost all properties of linear estimators analytically. This is an exceptional feature: there are many estimators in econometrics whose properties can only be obtained through numerical simulation. Linear models are also fast and easy to compute.

Actually, it is not so easy to conceive a non-linear model. One example is:

$$y = \beta_0 + x_1^{\beta_1} + u\tag{1.2}$$

1.3 About this script

Most undergraduate textbooks spend a considerable number of pages on explaining the mechanics of matrix algebra and have students manually calculate matrix manipulations. This text tries to shift the focus by recognizing that a computer can do these calculations

1 Introduction

much faster and more accurately. Therefore, I have chosen to concentrate on algebra with *whole matrices*. Wherever possible, derivations and proofs will be done in matrix notation. One may need a bit of training to get used to this notation, but it is worth the effort, as matrix notation is used nearly everywhere in later econometrics classes and in research. Detailed numerical calculations will be entirely left to computer programs such as MATLAB.

Notation. Matrices are written as capital letters (e.g. A), vectors and scalars as small caps (e.g. a). In cases where vectors and scalars could be mixed up, bold small caps (e.g. \mathbf{x}) are used for vectors. Greek letters (α, β, \dots) are always scalars. Where not stated otherwise, a vector is always a column vector. In order to save space and make the script more readable, column vectors are sometimes written as transposed row vectors:

$$\mathbf{x} \equiv \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \equiv (x_1, \dots, x_n)'$$

Row vectors are denoted as transposed column vectors: $\mathbf{x}' \equiv (x_1, \dots, x_n)$.

MATLAB commands are written in the font teletype (e.g. `help`). To avoid duplication, most MATLAB commands are not explained; the MATLAB help function or the MathWorks website are better at this task. MATLAB menu items are written in the font sans serif, with sub-menus separated by a forward slash (e.g. Help/Full Product Family Help).

MATLAB examples. These indented sections contain sample code that is closely linked to the text. The student is invited to try these few lines of code immediately when reading the script. Best use MATLAB side-by-side with this script.

TIP: at a few occasions, a practitioner's tip has been added. Observing these tips is not mandatory, but makes life a lot easier.

* Paragraphs marked with an asterisk represent additional material that need not be studied at the first round. Exercises with an asterisk are a bit more challenging.

PC labs. The PC labs give the student a guided tour through a new concept. These are an integral part of the course – a few concepts are only shown in the PC labs. Usage is simple: enter the commands one by one in the command window and watch the results.

Exercises. Studying any methodological subject without doing exercises is worthless. So the reader is more than invited to try at least a few exercises per chapter. As some of the exercises are used as problem sets for grading, no solution guide will be published.

1 Introduction

Key terms. A list of key terms is given at the end of each chapter. Terminology is important especially to undergraduates, because without the correct terminology it is very difficult to find help (i.e. ask a colleague, search the web, find relevant book chapters or papers. These lists should be used for the exam preparation – every student should be able to define and explain the key terms.

Comments and corrections to the script are highly welcomed. Please send all comments to peter.gruber@unisg.ch

Acknowledgements. I would like to thank Anna Cieslak for contributing chapter 8 on the principal component analysis and my students of the 2007 course in “Linear Models and Variance Analysis” for finding zillions of typos. I take the sole responsibility for all remaining errors.

2 A recap of matrix algebra

References: Gentle (very detailed), Greene Appendix A, Simon/Blume, The matrix cookbook, Wooldridge Appendix D, Verbeek Appendix A, Abadir/Magnus (very advanced),

2.1 Introduction

Short history¹ The study of matrices is quite old. Latin squares and magic squares have been studied since prehistoric times. Matrices have a long history of application in solving linear equations. An important Chinese text from between 300 BC and AD 200, Nine Chapters of the Mathematical Art (Chiu Chang Suan Shu), is the first example of the use of matrix methods to solve simultaneous equations. In the seventh chapter, "Too much and not enough," the concept of a determinant first appears, almost 2000 years before its invention by the Japanese mathematician Seki Kowa in 1683 or the German Gottfried Leibniz in 1693. The Swiss mathematician Gabriel Cramer developed the theory further, presenting Cramer's rule in 1750. Carl Friedrich Gauss and Wilhelm Jordan developed Gauss-Jordan elimination in the 1800s. The term "matrix" was first coined in 1848 by the English mathematician James Joseph Sylvester. Cayley, Hamilton, Grassmann, Frobenius and von Neumann are among the famous mathematicians who have worked on matrix theory.

Why matrix notation? Matrix notation has three advantages: it is concise, powerful and we have excellent computing tools for it. Matrices allow us to use just one symbol for a two-dimensional array of numbers. This is the perfect representation for most economic data. The alternative – double indices and sums – would be quite confusing. Furthermore, studying special properties of matrices such as the rank, the definiteness or their eigenvalues, allows us to get additional insights.

¹See also http://en.wikipedia.org/w/index.php?title=Matrix_%28mathematics%29

2.2 Matrices, vectors and scalars

Definition 1. A *matrix* is a rectangular array of scalar numbers. A matrix with m rows and n columns is called $m \times n$ -matrix. There are four different notations for matrices: a plain symbol, a symbol with subscript dimensions, one element with running indices and the full matrix:

$$A = A_{m,n} = (a_{ij}) = \begin{bmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & & \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mj} & \cdots & a_{mn} \end{bmatrix}$$

Note that it is not possible to have “holes” in a matrix. If a matrix represents a set of data, one has to be very careful about missing data entries. Missing data should never be replaced by a zero, as this would almost surely lead to a bias in the estimation.

MatLab note: MatLab has a special symbol for missing data: NaN (= “not a number”). MatLab can perform calculations with matrices that contain NaNs, however the results will be also NaN most of the time.

A $m \times n$ matrix can be decomposed in m row vectors of dimension $1 \times n$ or n column vectors of dimension $m \times 1$. Thus we can write:

$$\text{Row vector } A(i, :) = (a_{i1}, \dots, a_{ij}, \dots, a_{in}) \quad \text{Column vector } A(:, j) = \begin{bmatrix} a_{1j} \\ \vdots \\ a_{ij} \\ \vdots \\ a_{mj} \end{bmatrix}$$

MatLab note: In Matlab, the element a_{ij} can be accessed using this syntax: $A(i,j)$. The i -th row vector can be accessed using $A(i,:)$ and the j -th column vector using $A(:,j)$. MatLab has a special notation for row and column vectors: the colon $(:)$ operator, which stands for “all possible index values” in this dimension.

Interpretation of a matrix. There are different ways to interpret a matrix, depending on the underlying problem, a matrix can be seen as:

- Coefficients of a system of equations.
- Some data. A sample of explanatory variables (“X”) can be written as a matrix. The individuals are in rows, the different properties in columns. In other cases,

2 A recap of matrix algebra

matrix data is a number of column vectors, where each column vector is a time series of observations.

- A number of column vectors.
- A number of row vectors.
- A projection. A $m \times n$ matrix projects a vector from \mathbb{R}^m to \mathbb{R}^n , thus recombining the vector's elements and changing its dimension.

Conventions. The dimensions of a matrix A_{mn} are always quoted as **rows** \times **columns**. The same is valid for the index of the element (a_{ij}) : the first index is the row index, the second one the column index. This convention is also used by most software programs and programming languages, e.g. by MatLab.

MATLAB examples: The command `ones(5,3)` produces a 5×3 (rows \times columns) matrix with every element being 1; `A(2,4)` returns the element in row number 2, column number 4 of matrix A (equivalent to a_{24}).

In the case of *multivariate data*, one individual is usually represented in a row vector whereas one property is represented in a column vector. In the case of *panel data*, the first index is usually the time index (i.e. one column is one individual over time, one row is all individuals at one point in time.)

<div style="display: flex; align-items: center; justify-content: center;"> <div style="text-align: right; margin-right: 10px;"> Dimension of A_{mn} Index of (a_{ij}) </div> <div style="font-size: 3em; margin-right: 10px;">}</div> <div> rows \times columns </div> </div>

2.3 Special matrices

- *Square matrix*: equal number of columns and rows, $m = n$.
- *Diagonal matrix*: only the elements along the main diagonal a_{ii} are different from zero. Example: $\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$
- *Identity matrix* is a diagonal matrix with $a_{ii} = 1$. The identity matrix is the neutral element in matrix multiplication: any matrix multiplied with the identity matrix remains the same. There are several symbols for the $n \times n$ identity matrix: I , I_n and Id (for identity), the bold **1** and the German language E . Example: $I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
- *Scalar matrix*: a diagonal matrix with the same value in all diagonal elements. a_{ii} are different from zero. MatLab command: `eye(M,N)`. Example: $\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$

2 A recap of matrix algebra

- *Zero matrix*: a matrix with all elements equal to zero: $(a_{ij}) = 0$. Sometimes, the bold $\mathbf{0}$ is used as a symbol. The zero matrix is the neutral element of the matrix addition. MatLab command: `zeros(M,N)`. Example: $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
- *Symmetric matrix*: a square matrix is symmetric if $a_{ij} = a_{ji}$. Properties of these matrices are very important, because the covariance matrix is a symmetric, square (and positive definite) matrix. A symmetric matrix is equal to its transposed: $A = A'$. Example for a symmetric matrix: $\begin{bmatrix} 1 & -2 & 3 \\ -2 & 7 & 5 \\ 3 & 5 & 0 \end{bmatrix}$
- Upper (lower) *triangular matrix*: a matrix with all elements below (above) the main diagonal zero. Example for an upper triangular matrix: $\begin{bmatrix} 1 & 2 & 3 \\ 0 & -7 & 5 \\ 0 & 0 & 1 \end{bmatrix}$
- *Projection matrix*: a square matrix for which $B \cdot B = B$. These matrices are also called *idempotent matrices*. The matrix $X(X'X)^{-1}X'$, which plays an important role in OLS, is a projection matrix. Example: the identity matrix or $\begin{bmatrix} -2 & 3 \\ -2 & 3 \end{bmatrix}$
- *Orthogonal matrix*: every square matrix, which fulfills $A' = A^{-1}$. If A is orthogonal, then $AA' = Id$. Note that the row/column vectors are pairwise orthonormal. Example: $\begin{bmatrix} 0.6 & 0.8 \\ -0.8 & 0.6 \end{bmatrix}$

2.4 Matrix calculation rules

Note that many of these rules are also valid for vectors, as vectors can be seen as $n \times 1$ viz. $1 \times m$ matrices.

Addition and subtraction. Matrices are added (subtracted) element by element:

$$A + B = C = (c_{ij}) = (a_{ij} + b_{ij})$$

Only matrices of the same dimension can be added (subtracted). This implies that it is not possible to add a scalar to a matrix.

Properties:

$$\begin{aligned} A + B &= B + A \\ A + (B + C) &= (A + B) + C \\ A + \mathbf{0} &= A \end{aligned}$$

MatLab note: In MatLab, it is possible to add a scalar x to a matrix A with the command `A+x`. In this case, the scalar x is added to *every* element of the matrix A .

Scalar multiplication A matrix is multiplied by a scalar by multiplying each element of the matrix with this scalar:

2 A recap of matrix algebra

$$\alpha A = \begin{bmatrix} \alpha a_{11} & \alpha a_{12} & \dots & \alpha a_{1n} \\ \alpha a_{21} & \alpha a_{22} & \dots & \alpha a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha a_{n1} & \alpha a_{n2} & \dots & \alpha a_{nm} \end{bmatrix}$$

Properties: $\alpha(A + B) = \alpha A + \alpha B$
 $\alpha A = A\alpha$

Vector product. The vector product (or "dot" product) is only defined for one row vector times one column vector of same number of elements.

$$a'x = a_1x_1 + a_2x_2 + \dots + a_nx_n$$

An alternative notation with two column vectors is: $\langle ax \rangle = a'x$

Properties: $a'x = x'a$
 $a'x = 0$ for orthogonal vectors

MatLab note: It can be quite confusing which vector to transpose to obtain the vector product. Alternatively, use `dot(x,y)`, which works for any combination of row and column vectors.

Matrix multiplication. The product of the matrices $A_{m,n}$ and $B_{n,p}$ is a matrix $C_{m,p}$ with

$$c_{ij} = \sum_{k=1}^n a_{ik}b_{kj} \quad i = 1, \dots, m \quad j = 1, \dots, p$$

c_{ij} can be seen as the scalar product of the i^{th} row vector and the j^{th} column vector:
 $c_{i,j} = A(i,:)B(:,j)$.

Commuting matrices Note that matrices have to be of matching dimensions, i.e. the number of columns in A must be equal to the number of rows in B . Note that even if both exist, generally $AB \neq BA$. This is especially true for vectors, where $x' \cdot x$ is a scalar, whereas $x \cdot x'$ is a matrix.

A special case of matrix multiplication is multiplying a column vector with a row vector, which gives a matrix.

2 A recap of matrix algebra

Properties:

$$\begin{aligned}
 A \cdot Id &= Id \cdot A = A \\
 A(B + C) &= AB + AC \\
 (B + C)A &= BA + CA \\
 A(BC) &= (AB)C \\
 AB &\neq BA \text{ (generally)}
 \end{aligned}$$

The *vec* operator. This operator *vectorizes* a matrix into a vector by stacking the column-vectors. Thus one can apply operations that are only defined for vectors to matrices. In the 2×2 case, the *vec* operator looks like this:

$$vec \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{pmatrix} 1 \\ 3 \\ 2 \\ 4 \end{pmatrix}$$

MatLab note: The MatLab command for $vec(A_{m,n})$ is `reshape(A,n*m,1)`

Properties:

$$\begin{aligned}
 vec(A + B) &= vec(A) + vec(B) \\
 vec(\alpha A) &= \alpha vec(A) \\
 * \quad Tr(A'B) &= vec(A)' \cdot vec(B) \\
 * \quad vec(ABC) &= [C' \otimes A]vec(B)
 \end{aligned}$$

Equality. Equality is defined element-by-element. This implies that equality is only defined for matrices of the same dimension.

$$A_{m,n} = B_{m,n} \Leftrightarrow a_{ij} = b_{ij} \quad \forall i, j$$

MatLab note: The comparison operator for equality, the double equal sign (`==`), works exactly the same way. MatLab also defines all other comparison operators (`<`, `>`, `<=`, `>=`) element-by-element. One can e.g. test whether matrix A is “smaller” than B . This is true if all elements of A are smaller than the corresponding elements of B : $A_{m,n} < B_{m,n} \Leftrightarrow a_{ij} < b_{ij} \forall i, j$.

Rank of a matrix. In any matrix, the number of linear independent row vectors is the same as the number of linear independent column vectors. If the matrix is not square, this automatically implies that there must be some linear dependent vectors in the larger dimension.

2 A recap of matrix algebra

Definition 2. The rank of a matrix is the number of linear independent column/row vectors. As discussed above $rk(A_{m,n}) \leq \min(m, n)$. Furthermore, a matrix is called *regular* (or is said to have *full rank*), if $rk(A_{m,n}) = \min(m, n)$.

The rank of a matrix is crucial for the solvability of a system of equations, see Simon-Blume, section 7.4.

$$\begin{aligned} \text{Properties: } \quad & rk(A + B) \leq rk(A) + rk(B) \\ & rk(AB) = \min(rk(A), rk(B)) \\ & rk(A'A) = rk(A) \quad \text{(Useful for OLS regularity)} \end{aligned}$$

Example 1. Dummy variable trap. If there is a constant term (=1), we cannot have a dummy variable for “male” and for “female”, because $male + female = 1$ which is linear dependent from the constant

The rank criterion has one disadvantage: There are many other real-world cases with random errors, in which matrices are “close to singular”. In such a case, it is better to consider the condition number, see page 81

Inverse of a matrix. The inverse of a square matrix is defined in analogy to the inverse of a scalar. For any scalar number a , the inverse a^{-1} fulfills the equation $a \cdot a^{-1} = 1$. For a matrix A , this is generalized to

$$AA^{-1} = Id.$$

Theorem 1. *The inverse of a matrix exists if and only if the matrix is regular. Proof: Simon-Blume, p. 166.*

Calculation of the inverse: see determinants. In practice, one tries to avoid calculating the inverse of a matrix, because it is computationally intensive and can introduce imprecisions in some cases.

$$\begin{aligned} \text{Properties: } \quad & (A^{-1})^{-1} = A \\ & Id^{-1} = Id \\ & (AB)^{-1} = B^{-1}A^{-1} \\ & AA^{-1} = A^{-1}A = Id \quad \text{(Obtains by combining first three properties)} \\ & (\alpha A)^{-1} = \frac{1}{\alpha}A^{-1} \\ & (AB)(B^{-1}A^{-1}) = AIdA^{-1} = Id \\ & (A^{-1})' = (A')^{-1} \\ & A, B \text{ orthogonal} \rightarrow AB \text{ orthogonal, as well} \end{aligned}$$

Uses of the inverse: solve a system of equations. (Note: as the inverse exists only for square matrices with full rank, this approach works only for exactly identified systems.)

2 A recap of matrix algebra

$$\begin{aligned} y &= Ax \\ A^{-1}y &= x \end{aligned}$$

Transposed of a matrix The transposed of a matrix obtains by exchanging its rows and columns:

$$A_{m,n} = (a_{ij}) \Rightarrow A'_{n,m} = (a_{ji}) \quad .$$

Note that the number of rows and columns are exchanged between a matrix and its transposed. Transposing converts a row vector into a column vector and vice versa. This is often used to shorten the notation of a column vector: $x = (x_1, x_2, x_3)'$.

Properties:

$$\begin{aligned} (A')' &= A \\ Id' &= Id \\ (A + B)' &= A' + B' \\ (\alpha A)' &= \alpha A' \\ (AB)' &= B' A' \\ A \text{ symmetric} &\Rightarrow A = A' \\ rk(A' A) &= rk(A) \quad (A \text{ nonsquare, full rank} \rightarrow, A' A \text{ or } A A' \text{ full rank}) \\ \alpha' &= \alpha \quad (\text{May seem trivial, but gives rise to the following:}) \\ x' A y &= (x' A y)' = y' A' x \end{aligned}$$

Trace of a matrix The trace of a square matrix is the sum of the diagonal elements. It has a lot of convenient properties.

$$Tr(A_{n,n}) = \sum_{i=1}^n (a_{ii})$$

Properties:

$$\begin{aligned} Tr(Id_n) &= \sum_{i=1}^n 1 = n \\ Tr(A) &= Tr(A') \\ Tr(A + B) &= Tr(A) + Tr(B) \\ Tr(AB) &= Tr(BA) \quad (\text{this gives rise to:}) \\ Tr(ABC) &= Tr(CAB) = \dots \quad (\text{any rotation gives the same trace}) \\ Tr(\alpha) &= \alpha \quad (\text{This simple identity gives rise to:}) \\ x' C x &= Tr(x' C x) = Tr(C x x') \end{aligned}$$

2 A recap of matrix algebra

$$\begin{array}{ll}
 \text{More properties:} & Tr(\alpha A) = \alpha Tr(A) \\
 * & Tr(AA') = \sum_i \sum_j a_{ij}^2 \\
 * & Tr(A'B) = vec(A)' \cdot vec(B) \\
 * & vec(ABC) = [C' \otimes A]vec(B)
 \end{array}$$

Determinant of a matrix. The determinant is a scalar function of a square matrix A . It is nonzero if and only if the matrix is regular (Proof: Simon-Blume, p. 193). One can say that the determinant *determines* whether a square matrix is singular or not. Later we will see that the determinant is also the product of the characteristic roots of the matrix.

There are different ways how to compute the determinant. One way is the recursive definition. It starts with $n = 1$. A scalar is its own determinant: $\det \alpha = \alpha$. For larger matrices, the following rules apply: any row i can be chosen. For every element a_{ij} in this row, we can define a cofactor C_{ij} as

$$C_{ij} = (-1)^{(i+j)} M_{ij}$$

where M_{ij} is the so-called minor, the determinant of a matrix A_{ij} , which obtains if the i -th row and the j -th column of A are deleted. The determinant is then

$$\det A = \sum_{i=1}^n a_{ij} C_{ij}.$$

Note that if A is of dimension n , then A_{ij} is of dimension $(n - 1)$. To calculate $M_{ij} = \det A_{ij}$, one can re-apply this rule and obtain a matrix of dimension $(n - 2)$ and so on until one can use $\det \alpha = \alpha$.

The determinant of a 2×2 matrix is the product of the diagonal elements minus the product of the off-diagonal elements:

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{21}a_{12}$$

MatLab note: MatLab uses triangular matrices to calculate the determinant of A . First, A is decomposed into a lower and an upper triangular matrix $A = LU$. Next, MatLab uses the identity $\det(AB) = \det(A)\det(B)$ and the fact that the determinant of a triangular matrix is exactly the product of its diagonal elements: $\det L = \prod_{i=1}^n l_{ii}$.

2 A recap of matrix algebra

<u>Properties:</u>	$\det \alpha = \alpha$	
	$\det A' = \det A$	
	$\det(A \cdot B) = (\det A)(\det B)$	
	$\det(A + B) \neq \det A + \det B$	in general
	$\det(A^{-1}) = \frac{1}{\det A}$	
	$\det(\alpha A_{n \times n}) = \alpha^n \det A$	
	$\det A = \prod_{i=1}^n a_{ii}$	for A triangular or diagonal

Uses of the determinant:²

- Determine whether a matrix is singular or not.
- Calculate the inverse using the cofactors: $A_{ij}^{-1} = (\frac{1}{\det A} C_{ij})$
- Find analytical solutions to $Ax = b$ using Cramer's rule: $x_i = \frac{\det B_i}{\det A}$, where B_i is the matrix A with the right hand-side b replacing the i th column of A .

MatLab note: Cramer's rule has the advantage that it can deliver analytical solutions. For numerical purposes, it is quite slow, because it requires to calculate $(n + 1)$ determinants to solve a system of n equations. It is more efficient to solve systems of linear equations using Gaussian elimination. In MatLab, this is done using the left-divide operator (\backslash).

Similar, calculating the inverse as presented here requires to compute $(n^2 + 1)$ determinants. MatLab uses the LU-decomposition and Gaussian elimination to compute the inverse.

Quadratic forms. A matrix $A_{n \times n}$ can be multiplied with a vector $\mathbf{x}_{n \times 1}$ and its transposed to yield a scalar. This expression is called a quadratic form:

$$x'Ax$$

Positive definiteness. A matrix is called positive (semi-) definite if the quadratic form of every vector \mathbf{x} is positive (positive or zero):

$$x'Ax > (\geq) 0$$

Analogy to the scalar case: Take two real numbers a and x and construct the equivalent of a quadratic form:

$$xax = ax^2$$

This “quadratic form” is positive iff $a > 0$ and negative iff $a < 0$, regardless of the value of x (i.e. for all values of x). In the matrix world, there is an additional possibility: a matrix can be indefinite.

²See Simon-Blume Sect 9.2 and Sect 26.3 for more details.

2 A recap of matrix algebra

<u>Properties of pos. definite matrices</u>	$x_{ii} > 0$ (diagonal elements strictly positive) $\det A^{-1}$ exists and is also pos. def. $X'X$ and XX' are at least positive semidefinite $\det(A+B) \neq \det A + \det B$ in general for $X_{n \times k}$: $X'X$ is positive definite
---	--

2.5 Frequently used tricks with vectors

- Sum of elements: $\mathbf{x}'\mathbf{1} = \sum x_i$
Where $\mathbf{1}$ is a vector of ones.
- Sum of squares: $\mathbf{x}'\mathbf{x} = \sum x_i^2$
- Variance-Covariance matrix: $= X'X$
Where X is the de-meaned data matrix (individuals in rows, variables in columns).

2.6 Solving matrix equations

Solving equations with matrices works much like solving “normal” equations. There are four main principles: (1) add/subtract or substitute equations, (2) perform operations with the neutral element, (3) perform same operation on both sides (4) apply special matrix manipulations. Note, however, that there are a few additional rules that must be observed.

(1) Adding, subtracting and substituting:

- You can add up two equations by adding the left sides and the right sides. The matrices have to be of the same dimension. Example:
$$\begin{array}{rcl} A & = & B \\ C & = & D \\ \hline A + C & = & B + D \end{array}$$
- You can substitute for an expression. Example:
$$\begin{array}{ll} \text{(I)} & A = B + C \\ \text{(II)} & B + A = -C \end{array}$$
 Then we can substitute A in equation (II) with $B + C$ from equation (I) and we get: $B + (B + C) = -C$ and finally $B = -C$.

(2) Operations with the neutral element:

- Add zero: you may add zero to just one side, e.g. in the form $(+AB - AB)$. One application is integration by completing the square, which is useful in the context of multivariate Gaussian distributions.
- Multiply with Id , e.g. by multiplying with AA^{-1} . This can be used e.g. for simplifying the expression $[Id - \alpha(A + \alpha Id)^{-1}]$. Replace Id with $(A + \alpha Id) \cdot (A + \alpha Id)^{-1}$. This gives $[(A + \alpha Id) \cdot (A + \alpha Id)^{-1} - \alpha(A + \alpha Id)^{-1}]$. Now we can factor out the term $(A + \alpha Id)^{-1}$, leaving us with $[(A + \alpha Id) - \alpha Id](A + \alpha Id)^{-1} = A(A + \alpha Id)^{-1}$.
- Reshuffle terms and simplify: $AA^{-1} = Id$ and $A - A = 0$. *Note:* Of course, you have to obey the rules for re-shuffling, e.g. that $AB \neq BA$. Make sure to use all special properties e.g. of the trace and the determinant.

(3) Operations on both sides:

- Addition: you may add/subtract any matrix A of matching dimensions to the left and right side.
- Multiplication: you may left- or right-multiply both sides. *Note:* You can only multiply from the left and from the right side, not in the middle.

2 A recap of matrix algebra

- Divide: instead of dividing it is better to multiply with the inverse matrix. Just as with the multiplication, you can only right- or left-multiply with the inverse. Note: If you multiply with the inverse of a matrix, you must ensure that this inverse exists.
- As there is no inverse for vectors, the procedure is slightly more complicated. First, multiply both sides with the transpose of the vector to get $x'x$, which is a scalar. Second, multiply both sides with the inverse of this expression.

(4) Special matrix manipulations:

- Transpose or invert. Note that the first step is to transpose or invert the left and right sides *as a whole*. Then apply the specific rules for transposing and inverting, especially $(AB)' = B'A'$ and $(AB)^{-1} = B^{-1}A^{-1}$.
- Apply special rules for Tr and det (see previous section).
- Decompose a matrix (see matrix decompositions in section 8.2).

A few things you cannot do:

- Remove det or Tr . Note that $\det A = \det B$ does *not* imply $A = B$. The same is true for the trace.
- Divide from different sides or “from the center”. Example for a wrong solution: given $BAC = Id$ you could be tempted to multiply with A^{-1} from the center, giving $BC = A^{-1}$ and finally $A = C^{-1}B^{-1}$. This is *wrong*. Finding the correct solution is part of exercise 2.1.

2.7 Exercises

Exercise 2.1. Isolate A . Assume that all matrices are square and regular.

- | | | |
|-------------------------|-----------------------|------------------------------|
| a) $A + B = C$ | b) $\alpha A + B = C$ | c) $AB = C$ |
| d) $AB + AC = D$ | e) $BAC = D$ | f) $A'B^{-1} = C$ |
| g) $A^{-1} = \alpha BC$ | h) $BAC = Id$ | *i) $B = \alpha(A - C)^{-1}$ |

Exercise 2.2. Isolate x . Assume that x, y are $n \times 1$ column vectors, that all matrices are $n \times n$ and that all inverses exist.

- | | |
|--------------|---------------------|
| a) $y = Ax$ | b) $y = Ax + Bx$ |
| c) $A = xy'$ | d) $yx' + Ayx' = B$ |

Exercise 2.3. X be a general matrix. Show that (a) XX' and (b) $X + X'$ are symmetric matrices. For (b), we need an additional assumption.

2 A recap of matrix algebra

Exercise 2.4. X be a nonsquare matrix.

- (a) Prove that $X(X'X)^{-1}X'$ is a projection matrix by calculating the squared of this expression. Assume that $(X'X)^{-1}$ exists.
- (b) What are the conditions for the existence of $(X'X)^{-1}$? Include a statement on the dimension of X .
- (c) Now suppose that X is a regular, square matrix. Is it easier to show that the expression in (a) is an projection matrix?

Exercise 2.5. Verify that the identity matrix and the example for a projection matrix on page 12 are idempotent matrices by calculating the square and the tenth power of these matrices in MatLab.

Exercise 2.6. Identify if the following equalities are correct or not. Assume that all matrices are of matching dimensions (not necessarily square) and that the inverse exists, if stated in the identity.

- | | |
|----------------------------------|-----------------------------------|
| a) $Tr(AB) = Tr(A'B')$ | b) $Tr(AB) = Tr(A) \cdot Tr(B)$ |
| c) $Tr(A^{-1}BA) = Tr(C^{-1}BC)$ | d) $Tr(A) = \frac{1}{Tr(A^{-1})}$ |

Exercise 2.7 (Positive definiteness). Verify that the matrix $A = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$ is positive definite by calculating the quadratic form $\mathbf{x}'A\mathbf{x}$ using the general vector $\mathbf{x} = (x_1, x_2)'$.

Exercise 2.8 (*). (a) Prove that $Tr(AB) = Tr(BA)$. Start writing an expression for the i -th diagonal element of AB and of BA .

Exercise 2.9 (*). Show that $\det(Id + AB) = \det(Id + BA)$.

Hint: represent $(Id + AB)$ as a product of matrices and apply a suitable property of the determinant.

Exercise 2.10 (*). Simplify the following expressions:

- a) $x'A'BAx + x'A'B(C - D)^{-1}BAx$
- b) $A'BC(Id - CBC)^{-1}CBA$
- c) $Id + 2(A^{-1}B^{-1} - 2Id)^{-1}$

Exercise 2.11 (*). Verify in MatLab that Gaussian elimination is much faster than explicitly calculating the inverse and performing a multiplication. To do so, use `A=rand(1000)` to produce a square matrix A that consists of 1000x1000 random numbers. Use `x=rand(1000,1)` to produce a (column) vector x that consists of 1000 random numbers. Solve the system of equations $x = Ab$ in two ways: a) with explicit inversion of A and b) using Gaussian elimination. Compare the execution time.

Hint : to find out about gaussian elimination, enter `help mldivide`. To find out about timing, enter `help tic`.

3 An interactive introduction to MATLAB

References: Getting started with MATLAB, sections 1, 2 and 6-1 to 6-13; , Moler, Hanselman and Littlefield

3.1 Why MATLAB?

MATLAB features an excellent combination of flexibility, ease of use and speed. It is much easier to learn than conventional programming languages such as Java or C++, yet it brings most of their flexibility. With MATLAB's interactive mode one can solve simple problems even without writing a program. Yet it is much more versatile than statistics programs such as Stata or Eviews. The main asset of MATLAB, however, is its matrix-centric design. Almost everything that can be done with MATLAB can be done with matrices without any extra effort.

MATLAB is the standard for advanced quantitative modelling in the finance industry and in many engineering disciplines. It is also being used increasingly in economics research.

3.2 Getting started

TIP: If your screen does not look like Fig. 3.1, click on Desktop/Desktop Layout/Default.

(1') Current Directory Bar By default, everything will be loaded or saved from/to this directory. Change the current directory by clicking into the three dots.

(1) Current Directory Window The contents of the current directory are shown in this window. If you double-click a program, it will be opened in the program editor (4). To run a program, right-click on its name and select Run.

(2) Workspace Note that at the lower end of the current directory window, there is a tab where you can switch to the workspace (2). The workspace is a view into the “memory” of MATLAB. Here, you can see all current variables and their values. When going through

3 An interactive introduction to MATLAB

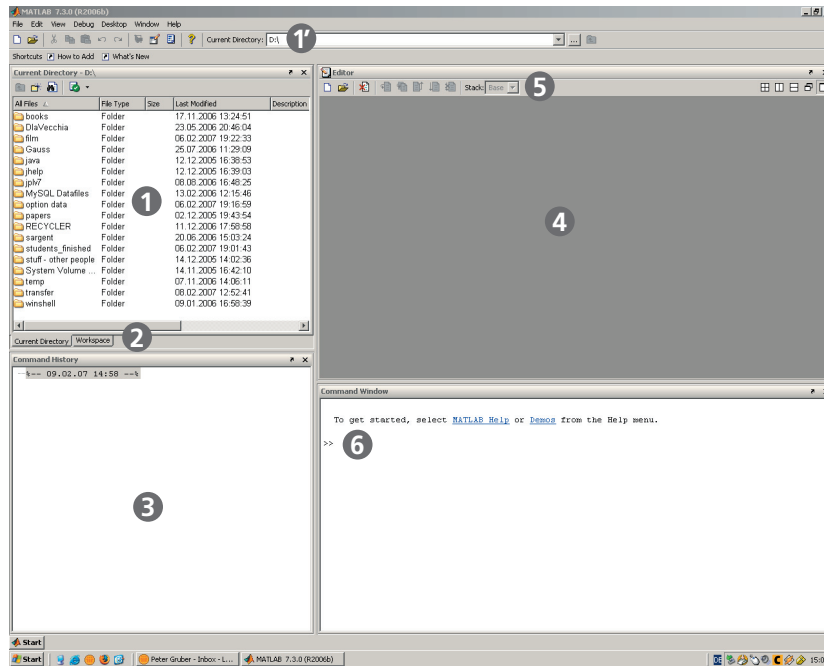


Figure 3.1: MATLAB startup screen. (1') Current Directory Bar, (1) Current Directory Window, (2) switch to Workspace, (3) Command History, (4) Program Editor, (5) symbols for editing programs, (6) Command Window.

the interactive introduction to MATLAB in section 3.4, keep an eye on the workspace and observe how the variables evolve.

(3) Command history. Here you see a list of all commands that have been entered, in reverse chronological order (you can scroll back to your first command). The command history can be used to create a program from trials in the interactive mode.

(4) Program editor. Here you can write, edit and start your programs.

(5) Toolbar of the program editor. The symbols are shortcuts for opening and saving program files as well as to test-run a program.

(6) Command window. Any command that is entered here is executed immediately. This can be used as a simple pocket calculator or to perform calculations step by step without writing a program. See the interactive introduction to MATLAB in section 3.4. If you run a program, the output is shown in the command window. Another use of the command window is to test new ideas. Once the results are satisfactory, the command history (3) can be used to assemble a program.

Getting started with Octave. You can think of the Octave user interface as having only a command window. For all other elements, Octave relies on substitutes. The workspace is replaced by the `who` command, the current directory window by the commands `ls`, `pwd` and `cd`, the command history by `history`. Octave has no program editor, you have to use the external editor of your choice. See Appendix ??.

TIP: If you don't want to install anything and play a little bit with MATLAB/Octave, try Online Octave at http://lavica.fesb.hr/octave/octave-on-line_en.php

3.3 The main elements of the MATLAB language

Variables. Variables are created *implicitly*, i.e. by assigning a value to them. The command `a=1` performs two things: (1) a variable called `a` is created and (2), the value 1 is assigned to it. Variables can be of different *types*: scalars, vectors, matrices (like their mathematical counterparts) and string (for text). Note that the names of variables are case sensitive (so `A` is a different variable than `a`). Variables should be named with self-explaining names.

Arithmetic operators. An operator is best explained with an example: `+`, `-`, `*`, `/` are all *arithmetic operators*. The hat `^` is the power operator, e.g. `5^2` is `5^2`. (Remember that $\sqrt[n]{x} = x^{1/n}$). The order in which operators are executed is called *operator precedence*, see Tab. ??. *Example:* `2+3*5`. The multiplication is executed before the sum.

Relational operators. So-called *relational operators* compare two variables and give the results `true` or `false`. Among these are: equal (`==`), not equal (`~=`), less (`<`), less or equal (`<=`), greater (`>`), greater or equal (`>=`). Relations can only be true or false and there are numerical values for the two: `true`=1 and `false`=0. *Example:* `2<3` is `true` (or 1), `4.4==4.5` is `false` (or 0).

Logical operators. Most noteworthy are “and” (`&`), “or” (`|`) and not (`~`). They are used to combine two logical conditions. Learn more about operators by entering `help .` in the command window. *Example:* `2<3 || 4.4==4.5` is `true`. For details, see section ??.

Functions. A MATLAB function works like standard mathematical functions: it takes one or more arguments and produces a result. Widely used built-in functions are `log()`, `exp()`, `abs()`, `sqrt()`. MATLAB extends the concept of a function to vectors. *Example:* given a vector $\mathbf{x} = (x_1, x_2, x_3)'$, MATLAB computes $\log(\mathbf{x}) = (\log(x_1), \log(x_2), \log(x_3))'$. See section ??.

3 An interactive introduction to MATLAB

Apart from these built-in functions, users have the flexibility to write self-defined functions. Note that these work only as well as they have been programmed. User-defined functions are only vector compatible, if the user has explicitly programmed for this. A user defined function must be saved in a separate m-file.

Flow control. Flow control makes it possible to execute some commands only, if a certain condition is met (`if - else`) or to execute code several times (`for`, `while`).

Comments. MATLAB ignores all text after a percent sign (%) until the end of the line. Start a line with % for long comments or add a short remark after a command in the same line. Comments are an important element of programming style, see Section ??.

Data input and output. Calculations would make no sense if there is no output of the results. By default, MATLAB outputs the result of *every* calculation immediately on the screen. This may not be desirable for intermediate results. To suppress the output of a specific line, use the semicolon (;) at the end.

It is not customary to let the user interactively input data. Small amounts of data are often written into the program (preferably at the top), while MATLAB has a range of built-in functions to input large amounts of data and to store it to a file for later use. MATLAB can read and write a few file formats, including MS Excel. See Appendix ??.

Graphics. are a special form of data output. MATLAB supports a wide range of plot types. The data is taken from one or more vectors or a matrix and passed to a plot function. There are options to control almost every aspect of a plot. Alternatively, the command `plottools` opens a graphical user interface to fine-tune plots. For more details, see Appendix ??.

Toolboxes. are libraries of useful functions for a certain field. There are more than 90 different toolboxes available, with about a dozen relevant for economics and finance. Programs that rely on a toolbox will only run on machines where this toolbox is installed; this is impractical if several people work on a project. Also notice that the more complex a toolbox becomes, the more theory is contained in it, which increases the danger of misusing it.

3.4 PC-Lab: An interactive introduction to MATLAB

MATLAB is best explained by using it. This interactive demo introduces many elements of the MATLAB language. Just enter the commands in the command window (see Fig. 3.1), press return and observe the output. A short explanation is given for every command; detailed explanations follow in the two subsequent chapters.

Before getting started

TIP: MATLAB's diary command makes it possible to save the results of this interactive introduction. First, verify if the current directory (see Fig. ??), is set suitably. Now we can start.

`diary('intro')` Set the diary file name and start recording.

MATLAB as a pocket calculator

`1+1`
`2+3*5` MATLAB knows the correct *operator precedence*.

Variables

`clear` It's a good idea to start with clearing the workspace ...
`clc` ... and the screen in the command window.
`a` Error message, because the variable `a` has not yet been defined.
`a=2` This does two things: (1) create a variable `a` and (2) assign the value 2 to this variable. Watch the workspace (see Fig. ??).
`a` Recall the value of the variable.
`b=3.5` Define a variable `b` with value 3.5 (its a dot and not a comma!)
`b=3,5` If we use the comma (,) it means next command.
`b=3, c=5` E.g. to write two commands in one line (rarely used).
`A` You get an error message; there is no variable `A` (only small `a`). Variable names (and commands and file names) are *case sensitive* in MATLAB.
TIP: to avoid confusion, adopt a naming convention for variables.

Arithmetic operators

`a+b` Arithmetic operators are `+` `-` `/` `*` and `^` for "power".
`a-b` Negative numbers are simply entered with a minus in front.
`d=a+b` We can assign the result of every operation to a new variable.
`a+b*c` Operator precedence ("Punktrechnung vor Strichrechnung")

3 An interactive introduction to MATLAB

<code>(a+b)*c</code>	Round brackets have the highest precedence. Note that every shape of brackets has its own meaning, see Appendix ??.
<code>((a+b)*c+a)/b</code>	Several levels of brackets are all done with the round bracket <code>()</code> .
<code>a^b</code>	Power.
<code>a*b^c</code>	Power has a higher operator precedence.
<code>a^b+c</code>	This possibly gives an undesired result ...
<code>a^(b+c)</code>	If you want to enter a^{b+c} , you have to use a bracket.
<code>a^2</code>	Squared
<code>b^3</code>	Cubed
<code>sqrt(a)</code>	Square root. But what is the third root of a? There is no special function for it. We need to use our mathematics knowledge:
<code>a^(1/3)</code>	$\sqrt[3]{x} = x^{1/3}$. TIP: we never write <code>a^0.33</code> , because this is imprecise.

Some floating point formatting

<code>format long</code>	More digits for real numbers. From now on, output will be displayed with 15 digits.
<code>[2xcursor up]</code>	Cursor up gives the last result. Cursor up twice should give you <code>a^(1/3)</code> .
<code>format short</code>	Back again to 5 digits.
<code>[2xcursor up]</code>	
<code>help format</code>	We want to find out how to display a result in short scientific notation: <code>1.2570e+00</code> . The solution is:
<code>format short e</code>	
<code>format rat</code>	Also interesting (but maybe not too useful)

Functions

<code>sqrt(a)</code>	This has been the only built-in function so far.
<code>log(a)</code>	Let's have a look at more built-in functions: <code>log</code> stands for the natural logarithm (some people write <code>ln()</code> for this) ...
<code>log10(a)</code>	... as opposed to the logarithm with basis 10.
<code>exp(a)</code>	Exponential.
<code>abs(a)</code>	Absolute value.
<code>sign(a)</code>	Is +1 for all positive numbers and -1 for all negative numbers. Useful for sorting out positive and negative numbers.
<code>rand</code>	A special function that has no argument is the random function. Try it several times. Use cursor-up.
<code>help rand</code>	Does <code>rand</code> really have no arguments? Use the help function to find out.

Errors and NaN (not a number)

<code>log(0)</code>	Just as with the pocket calculator this gives an error.
<code>0/0</code>	Another error.
<code>a = 0/0, 1+a</code>	We can actually perform calculations with NaNs. However, in most of the cases we get another NaN and this may not be very useful.

Vectors

<code>x=[1, 3, 7]</code>	This creates a row vector: commas separate the columns of a vector.
<code>y=[2 8 0]</code>	We are lazy and use spaces as shortcut.
<code>y2=[3; -1; 1]</code>	The semicolon means “new line” – needed for column vectors (Note: by default, we use column vectors.)
<code>y2=[3 -1 1]'</code>	The transposed of a row vector is a fast alternative for entering column vectors.
<code>x*y</code>	You get an error message, because two row vectors cannot be multiplied.
<code>why</code>	A (joking) explanation for why you got an error message.
<code>x*y'</code>	Joking aside, for the inner vector product, we need to transpose y .
<code>y*x'</code>	Verify that the vector product is commutative.
<code>x'*y</code>	The outer product of two vectors yields a matrix.
<code>x(1)</code>	Get individual elements of a vector.
<code>x(1) = 9</code>	Change individual elements of a vector.
<code>x(2) = 8</code>	Change another element of the vector.
<code>x(2) = []</code>	Delete one element of the vector. Compare the result to the line above.
<code>x(end)</code>	Get the last element of x .
<code>x=[x 7]</code>	Grow the vector x .

Matrices

<code>z=[1 2 3; 4 5 6; 7 8 9]</code>	creates a 3×3 matrix
<code>eye(3)</code>	Identity matrix.
<code>eye(3)*x'</code>	If we multiply any vector or matrix with the identity matrix,
<code>eye(3)*z</code>	it remains the same.
<code>z*x'</code>	Matrix times vector.
<code>x*z*x'</code>	A quadratic form ... guess the output before hitting return.
<code>z*x</code>	If the dimensions do not match you get an error message.

3 An interactive introduction to MATLAB

<code>z(1,2)</code>	Before hitting return, predict the result. Remember: row, column.
<code>z(3,2)=-1</code>	Change an element of a matrix.
<code>x2=x'</code>	Prepare another column vector (note: <code>x2</code> and <code>y2</code> are column vectors).
<code>r=[x2 y2 y2]</code>	Create a matrix by concatenating column vectors.
<code>s=[z x2 y2]</code>	We can even concatenate matrices and vectors.
<code>t=[x; y; y]</code>	To concatenate row vectors use the semicolon (<code>;</code>)
<code>u=[x y y]</code>	What happens if we forget the semicolon? Guess the result before hitting return.

*More on vectors and matrices

<code>clear x</code>	Delete an entire variable. Watch what happens in the workspace.
<code>x(3) = 8</code>	This creates a new 1×3 vector. All leading elements are zero.
<code>w(3,3) = 5</code>	Likewise, create a matrix.
<code>size(w)</code>	Get the dimensions of <code>w</code> .
<code>q=[1 3 5 7 11 13 17]</code>	
<code>q([1,2,3])</code>	This gives us the first three elements of <code>q</code> .
<code>q([2,4,5])</code>	This gives us the second, fourth and fifth element of <code>q</code> .
<code>q(y)</code>	We can even do this: get the elements of <code>q</code> with the numbers that are in <code>y</code> . To better understand this line, output <code>q</code> and <code>y</code> again.



Semicolon and colon

<code>a=1;</code>	Reset the value of <code>a</code> .
<code>a+b;</code>	A semicolon at the end of a command suppresses the output. This seems to make no sense here, but the following does ...
<code>d=a+b;</code>	Most of the time, we are not interested in intermediate results. Therefore most lines MATLAB programs end with a semicolon.
<code>disp(d);</code>	Use <code>disp()</code> to display a result on the screen.
<code>1:5</code>	Colon – produce a vector from the first to the last number in steps of 1.
<code>1.2:5.5</code>	Fractions in the interval are suppressed.
<code>q(1:3)</code>	This is the main application of the colon operator: to access a range of elements of a variable, e.g. here the first three elements of <code>q</code> .
<code>q(1:end-1)</code>	Get all but the last elements – an often used application.

True or false?

<code>1<2</code>	This is of course true. Remember: <code>true=1</code> and <code>false=0</code> .
<code>0==1</code>	False = 0.
<code>x<y</code>	Vectors are being compared element-by-element. One result per element.

Plots

<code>x=1:50</code>	Remember, the colon (<code>:</code>) means: “all integer numbers in between”
<code>y=log(x)</code>	Note that all functions in MATLAB are vector-compatible.
<code>plot (y)</code>	Our first plot.
<code>x=[1 10 100]</code>	If the x -values are not evenly spaced ...
<code>y=log(x)</code>	... as it is the case here ...
<code>plot (y)</code>	... <code>plot(y)</code> gives a misleading result.
<code>plot (x,y)</code>	This is better.
<code>x=randn(1000,1)</code>	Produce a vector of normally distributed random variables. Should we have used a semicolon?
<code>hist(x)</code>	Histogram.
<code>hist(x,20)</code>	<code>hist(x,n)</code> produces a histogram with n bins.
<code>s=cumsum(x)</code>	Cumulative sums ($s_1 = x_1$; $s_2 = x_1 + x_2$; $s_3 = x_1 + x_2 + x_3$; ...).
<code>plot(s)</code>	A Monte Carlo simulation in a box!
<code>plottools</code>	Use this to change the design of a plot. Alternatively, click on the <code>plottools</code> symbol () in the plot window. You can also create a plot by clicking on () in the workspace.
<code>diary off</code>	Now we turn off the diary. A file called <code>intro.m</code> can be found at the location of the current directory. Why not e-mail this file home?

TIP: Now it is a good moment to have a look at Appendix ?? to get an overview over MATLAB’s symbols.

3.5 Getting help

- Use `help ;command;`
- Use `lookfor ;concept;`
- Use the menu Help/MATLAB Help
- You can also get help by selecting a command (in the command window or in the program editor), right-clicking on it and choosing Help on Selection.

- Demo programs: use `demo` to get a list. Interesting demos include `travel`, a travelling salesman problem.
- Useful web resources:
 - `www.mathworks.com`
 - `wiki.octave.org/wiki.pl?CategoryCode`
Homepage of Octave, the free program that is compatible to MATLAB.
 - `http://lavica.fesb.hr/octave/octave-on-line_en.php`
An online version of Octave: try simple commands and short programs without installing anything on your computer.
 - `www.mathworks.com/matlabcentral`
Huge collection of programs.
 - `www.wilmott.com`
`www.nuclearphynance.com`
`www.mathfinance.com`
Online communities of quants.
 - `www.statsci.org/matlab.html`
A list of statistical toolboxes for MATLAB.
 - `www.spatial-econometrics.com`
A comprehensive and well-documented econometrics toolbox.
 - `home.tiscalinet.ch/paulsoderlind`
Homepage of Paul Söderlind (Uni St. Gallen), collection of scripts, tools and links.
- Some further reading:
 - `history.siam.org`
The history of numerical analysis and scientific computing.

3.6 Exercises

Exercise 3.1. Use `A=magic(5)`. Create a variable `B` that contains the second row vector and a variable `C` that contains the third column vector of `A`. Calculate the inner (dot) product of `B` and `C` and the matrix product of `B` and `C`.

Exercise 3.2. Create a column vector with 4 elements and a 4x4 matrix (choose any numbers). Multiply the vector, the transposed of the vector and the matrix in the correct order so that you get (a) a scalar and (b) a matrix. Do not use try and error, but think first and aim at making this work at the first try.

Exercise 3.3. * Many matrix operations have only limited precision. To see this, let

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 9 & 8 \end{bmatrix} \text{ and } B = \begin{bmatrix} 2 & 2 & 4 \\ 1 & 3 & 5 \\ 2 & 4 & 8 \end{bmatrix}.$$

- (a) Verify the identities for the inverse on page 15.
- (b) Verify the identities for the transposed on page 16.
- (c) Verify the identities for the trace on page 16.
- (d) Verify the identities for the determinant on page 19.

Use `format long` to see the tiny differences or check for equality using the comparison operator `==`.

Exercise 3.4. Start with `A=magic(5)`. This creates a 5×5 magic square.

- (a) Produce a matrix B, which is the top left 3×3 submatrix.
- (b) Produce a vector x, which is the 2nd row vector of A.
- (c) Produce a vector y, which is the 3rd column vector of A.
- (d) Produce a 5×2 matrix C, which consists of the 1st and 4th column vectors of A.

Exercise 3.5. Compute the rank of at least ten random matrices of different size (square and non-square). Can you find a rule? *Hint:* use `rank()` and `rand()`.

Exercise 3.6. *X* contains a data set with the three variables *EXPR*, *MALE* and *FEMALE*. *EXPR* stands for years of work experience, *MALE* is a dummy variable that is 1 if the individual is male and *FEMALE* is a dummy variable that is 1 if the individual is female.

$$\text{Let } x = \begin{bmatrix} 5 & 0 & 1 \\ 7 & 1 & 0 \\ 12 & 0 & 1 \\ 3 & 1 & 0 \end{bmatrix}.$$

- (a) calculate the rank of *X*.
- (b) create three new matrices *A*, *B* and *C* that contain pairwise two of the three variables. Calculate the rank of each of these matrices.

Exercise 3.7. There is no `vec()` function in MatLab. Write a short program that calculates `vec(A)` for a matrix *A* of any size. Test your program with the matrix `A=magic(3)`. *Hint:* this exercise can be implemented in at least three different ways. One obvious way is to use a `for`-loop (which has not yet been discussed), but there are alternatives. One is a smart way to use the colon operator (explained in `help colon`), another one is to use the command `reshape()` (see `help`).

4 Some multivariate probability theory and statistics

References (for probability): Gut (advanced), Shao, section 1

References (for statistics): Shao (advanced), Wooldridge, Appendices B and C, Härdle Simar (section 4), LeSage

4.1 *Partitions and the concept of a random variable

Disclaimer. The following section may look a little bit technical or difficult. In fact, it just contains some new words and some unusual notation. If you do not like this section, skip directly to page 36. It is enough to know the following: *Simply put, a random variable is a function that assigns a real number to every possible state of nature.*

Example 2. When we throw a coin, a lot of things may happen: we can catch the coin or actually miss it, if the coin falls on the floor, it may make “plink” or “plonk”, the sun may be shining outside Still, most people will only be interested in one question: heads or tails? (See Fig. 4.1).

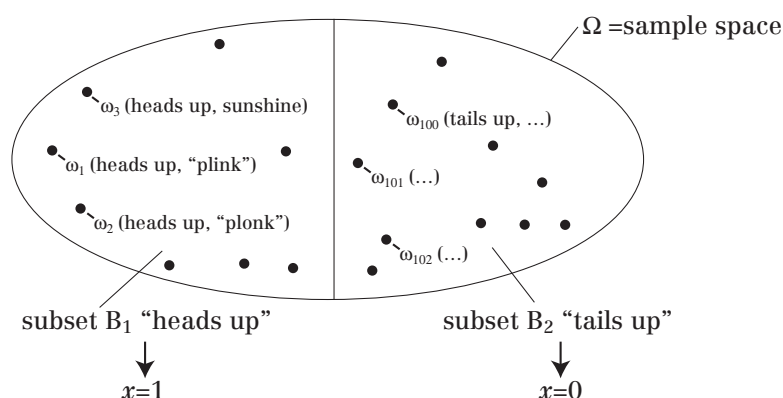


Figure 4.1: The sample space Ω for throwing a coin may contain many states of nature ω_i . We are only interested in the partition $\{B_1, B_2\}$, (“heads” and “tails”).

Sample space. The sample space is the set of all possible (relevant) events. We denote it with a big omega (Ω). In our example, this set contains all the above (and even more) possibilities.

States of nature. Each possible outcome is called a state of nature, denoted with a small omega (ω_i). The number of different omegas may be finite ($i \in \{1, 2, \dots, N\}$) or infinite ($i \in \mathbb{N}$). The number of different ω_i depends much on how “outcome” is defined. In our example, we have a very broad definition of outcome and therefore an infinite number of states of nature.

Partitions of Ω . Partitioning a set is like slicing a pizza. By partitioning Ω , we get a collection of subsets called $\mathcal{P} = \{B_1, \dots, B_n\}$. There are two rules for partitions:

1. No part of Ω must be forgotten (or $B_1 \cup B_2 \cup \dots \cup B_n = \mathcal{E}$)
2. No part of Ω must appear twice in a subset (or $B_i \cap B_j = \{\}$ $\forall i \neq j$).

Note that Ω can be partitioned in many different ways. In our example, $\mathcal{P} = \{B_1, B_2\}$ fulfills these rules, as can easily be verified in Fig. 4.1. If you slice a pizza, the two conditions are automatically fulfilled.

Sigma algebra of Ω . The set of subsets \mathcal{F} is a σ -algebra (or σ -field) of Ω if it fulfills the following conditions:

1. $\Omega \in \mathcal{F}$
2. If $B \in \mathcal{F}$, then its complement $B^c = \Omega \setminus B$ must also be an element of \mathcal{F} .
3. All unions of B_i are also elements of \mathcal{F} .

The partition is closely linked to the σ -algebra. One can generate a sigma algebra from a partition $\mathcal{P} = \{B_1, \dots, B_n\}$. This is expressed as $\mathcal{F} = \sigma(\mathcal{P})$

$$\sigma(\mathcal{P}) = \{\{\}, \Omega, B_1, \dots, B_n, \text{and all unions of } B_i\}$$

In the language of the pizza slices, a σ -algebra is the set of all possible dishes that one could make out of the pizza slices. One could eat nothing (empty set), everything (Ω) any one piece (B_i) or any combination of two or more pieces (“all unions of B_i ”).

In our example, $\sigma(\mathcal{P}) = \{\{\}, \Omega, B_1, B_2\}$. This expression contains no “and all unions of B_i ” terms, because $B_1 \cup B_2$ is already Ω .

Measurability. A function $f : \Omega \rightarrow \mathbb{R}$ is measurable with respect to $\sigma(\mathcal{P})$ if the value of f is the same for all states of nature (ω) in a given B_i .

4 Some multivariate probability theory and statistics

Note: We only require that all the ω within a B_i yield the same value of $f(\cdot)$. It is not necessary, that $f(\cdot)$ takes distinct values for every B_i . This is an injective relation. Thus, even the constant function $f(\omega) = 1$ would be measurable with respect to any σ -algebra.

We can use the sigma algebra to define this injective relationship. If $f : \Omega \rightarrow \mathbb{R}$ takes values in \mathcal{G} , then f is measurable if and only if the inverse function $f^{-1}(\mathcal{G}) \subset \mathcal{F}$.

Random variable. A measurable function from (Ω, \mathcal{F}) to \mathbb{R} is called a random variable.

Example 3. Let us throw a coin twice and define a random variable x that is the sum of “heads”. Obviously, x can take the values 0, 1 or 2. Let us denote T for tails and H for heads. Then a reasonable partition of Ω would be $\mathcal{P} = \{HH, HT, TH, TT\} = \{B_1, \dots, B_4\}$. The sigma algebra $\mathcal{F} = \sigma(\mathcal{P})$ of this partition would look like this:

$$\mathcal{F} = \{\{\}, \Omega, HH, HT, TH, TT, \{HH \cup HT\}, \{HT \cup TH\}, \dots, \{HH \cup HT \cup TH\}, \dots\}$$

where we suppressed most unions of the subsets. Is x measurable with respect to \mathcal{F} ? Let us look at the inverse function:

$$f^{-1} = \begin{cases} HH & \text{for } f(\omega) = 2 \\ \{HT \cup TH\} & \text{for } f(\omega) = 1 \\ TT & \text{for } f(\omega) = 0 \end{cases}$$

When $f(\omega) = 1$, we only know that either HT or TH happened, but not which one. Thus, x would not be measurable with respect to \mathcal{P} . Fortunately, this is not required. The set $\{HT \cup TH\}$ is an element of \mathcal{F} (see above), so x is indeed measurable with respect to \mathcal{F} .

Interpretation: The σ -algebra \mathcal{F} determines how detailed our knowledge of the real world can be, given we know the value of a random draw. The best we can do is to infer from x to a specific B_i . In some cases, as above, we can only infer to a set of B_i s, such as $\{HT \cup TH\}$. In any case, we will never get a more detailed information from a random variable than what is given by the partition \mathcal{P} .

4.2 Random vectors and matrices

Definition 3. A random vector (matrix) is a finite-dimensional vector (matrix) of random variables.

4 Some multivariate probability theory and statistics

Notation/MATLAB: We choose to define random vectors always as row vectors (\mathbf{x}'). This is in accordance with chapter 2, where we have defined that properties of an individual should be written as a row vector, whereas different realizations of one property as a column vector. It also fits well with MATLAB, where statistics (e.g. the mean or the variance) are calculated along columns by default.

Example 4. Consider two attributes of HSG students:

- x_1 average grade last year
- x_2 hours/week spent on studying

We can unite these two random variables in a random (row) vector \mathbf{x}' with two components:

$$\mathbf{x}' = (x_1, x_2).$$

Example 5. Consider the returns of two stocks at different points in time t : $(r_{1,t}, r_{2,t})$. Stock returns vary over time and nearly all stock returns are correlated. The return variance $(\sigma_{11}, \sigma_{22})$ is a measure for the risk, whereas the correlation $(\sigma_{12} = \sigma_{21})$ measures the diversification benefit. So it makes sense to study the variance-covariance matrix of the stock returns Σ :

$$\Sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix}$$

Simple models, such as the CAPM (capital asset pricing model) take Σ as constant, but this is an over-simplification. We know that there are “calm” and “rough” times at the stock exchanges, i.e. that the variability of stock returns changes over time. The same is true for correlations. Thus it makes more sense to model $\Sigma(t)$ as a time-dependent random matrix.

A semi-formal argument. A random variable is a measurable function from (Ω, \mathcal{F}) to \mathbb{R} . If a random vector contains several random variables, it would mean that it contains several mappings from Ω to \mathbb{R} . However, as there is only one world, it makes sense to speak of only one Ω . Instead of slicing Ω into different \mathcal{F}_i , we could make a new – finer – $\tilde{\mathcal{F}}$. Then we can simplify the argument by stating that a random vector is *one* mapping from $(\Omega, \tilde{\mathcal{F}})$ to \mathbb{R}^n .

4.3 Expectations and Covariances

4.3.1 Expected value

Definition 4. The expected value of a random vector \mathbf{x}' (a random matrix W) is the vector (the matrix) of the expected values of their elements. They are denoted as:

$$E(\mathbf{x}') = (E(x_1), E(x_2), \dots, E(x_n)) \quad E(W) = [E(w_{ij})]$$

Example 4 (continued). $E(x_1) = 5.2 = \mu_1; E(x_2) = 15 = \mu_2$

$$\Rightarrow E(\mathbf{x}') = (\mu_1, \mu_2) = (5.2, 15)$$

Properties. From the properties of the expectations operator E we derive:

P1 : $E(A) = A$; $E(b) = b$ with A, b being a constant matrix/vector

P2 : Linearity:

$$E(X_1 + X_2) = E(X_1) + E(X_2)$$

X_1 and X_2 vectors (or matrices) of equal dimension.

Application: $Tr(E(W)) = E(w_{11}) + E(w_{22}) + \dots = E(w_{11} + w_{22} + \dots) = E(Tr(W))$.

P3 : Multiplication with constant matrices:

- Let \mathbf{x} be an $n \times 1$ random vector, A a constant $m \times n$ matrix

$$E(A\mathbf{x}) = AE(\mathbf{x})$$

- Let W be a $m \times n$ random matrix, A a $p \times m$ matrix of constants and B a $n \times q$ matrix of constants

$$E(\underbrace{AWB}_{(p \times q)}) = AE(W)B$$

4.3.2 Variance-covariance matrix of a random vector

Example 4 (continued). Suppose x_1 and x_2 have second moments. In this case we have:

- 2 variances:

$$V(x_1) = E(x_1 - \mu_1)^2 = \sigma_{11} > 0$$

$$V(x_2) = E(x_2 - \mu_2)^2 = \sigma_{22} > 0$$
- 2 covariances:

$$Cov(x_1, x_2) = E(x_1 - \mu_1)(x_2 - \mu_2) = \sigma_{12}$$

$$Cov(x_2, x_1) = E(x_2 - \mu_2)(x_1 - \mu_1) = \sigma_{21}$$

By symmetry, $\sigma_{21} = \sigma_{12}$.

4 Some multivariate probability theory and statistics

We can rearrange this to get the (2×2) variance-covariance matrix $V(\mathbf{x})$:

$$\begin{aligned}
 V(\mathbf{x}) &= \begin{bmatrix} V(x_1) & Cov(x_1, x_2) \\ Cov(x_2, x_1) & V(x_2) \end{bmatrix} = \begin{bmatrix} E(x_1 - \mu_1)(x_1 - \mu_1) & E(x_1 - \mu_1)(x_2 - \mu_2) \\ E(x_2 - \mu_2)(x_1 - \mu_1) & E(x_2 - \mu_2)(x_2 - \mu_2) \end{bmatrix} = \\
 &= E \underbrace{\begin{bmatrix} (x_1 - \mu_1)(x_1 - \mu_1) & (x_1 - \mu_1)(x_2 - \mu_2) \\ (x_2 - \mu_2)(x_1 - \mu_1) & (x_2 - \mu_2)(x_2 - \mu_2) \end{bmatrix}}_{(2 \times 2)} = E \underbrace{\begin{bmatrix} (x_1 - \mu_1) \\ (x_2 - \mu_2) \end{bmatrix}}_{(2 \times 1)} \underbrace{[(x_1 - \mu_1) \quad (x_2 - \mu_2)]}_{(1 \times 2)} \\
 &V(\mathbf{x}) = E(\mathbf{x} - \mu)(\mathbf{x} - \mu)'
 \end{aligned}$$

Definition 5. Let \mathbf{x} be a random vector of n components with expected value $E(\mathbf{x}) = \mu$. The variance-covariance matrix of \mathbf{x} , denoted as $V(\mathbf{x})$, is by definition the following $(n \times n)$ matrix:

$$\underbrace{V(\mathbf{x})}_{n \times n} = E \underbrace{(\mathbf{x} - \mu)}_{(n \times 1)} \underbrace{(\mathbf{x} - \mu)'}_{(1 \times n)} \quad (4.1)$$

Properties of $V(\mathbf{x})$

- V is symmetric
- Its diagonal elements (V_{ii}) are the variances $V(x_i) = E(x_i - \mu_i)^2 \quad i = 1, 2, \dots, n$
- Its out-of diagonal elements are the covariances $Cov(x_i, x_j) = E(x_i - \mu_i)(x_j - \mu_j)$
- V is positive definite (or semi-positive definite if the x_i are linearly dependent).

Special cases

- if the components of \mathbf{x} are pairwise independent (or uncorrelated) $\Rightarrow V(\mathbf{x})$ is diagonal
- if additionally all components have the same variance $\Rightarrow V(\mathbf{x}) = \sigma^2 Id$
- if the expected value of \mathbf{x} is zero ($\mu = \mathbf{0}$), $\Rightarrow V(\mathbf{x}) = E(\mathbf{x}\mathbf{x}')$

Notation: We write $\mathbf{x} \sim (\mu, V)$ for a random vector \mathbf{x} that is distributed with:

- expected value $E(\mathbf{x}) = \mu$
- and variance-covariance matrix $V(\mathbf{x}) = V$

Matrices. There is no easy way to generalize the variance-covariance matrix of a vector for matrices. (One could define a three-dimensional tensor.) The alternative is to define a vector $\mathbf{x} = vec(A)$ and to study $V(\mathbf{x})$.

4.3.3 Linear transformations

Given a random vector $\mathbf{x} \sim (\mu, V)$ with n elements, consider the following transformation with the constant matrix A and the constant vector b :

$$\underset{(k \times 1)}{\mathbf{y}} = \underset{(k \times n)}{A} \underset{(n \times 1)}{\mathbf{x}} + \underset{(k \times 1)}{b}$$

In the case of:

- $b = 0$ this transformation is called *linear*.
- $b \neq 0$ this transformation is called *affine*.

Expected value and error term

$$E(\mathbf{y}) = E(A\mathbf{x} + b) = E(A\mathbf{x}) + E(b) = AE(\mathbf{x}) + b = A\mu + b \quad (4.2)$$

$$\mathbf{y} - E(\mathbf{y}) = A\mathbf{x} + b - (A\mu + b) = A\mathbf{x} - A\mu = A(\mathbf{x} - \mu) \quad (4.3)$$

Note that b does not enter the error term.

Variance-covariance matrix of a linear transformation. Let us start with the definition:

$$\begin{aligned} V(A\mathbf{x} + b) &= E(\underbrace{\mathbf{y} - E(\mathbf{y})}_{(k \times 1)})(\underbrace{\mathbf{y} - E(\mathbf{y})}_{(1 \times k)})' = E(A(\mathbf{x} - \mu))(A(\mathbf{x} - \mu))' \\ &= E[A(\mathbf{x} - \mu)(\mathbf{x} - \mu)'A'] = A \underbrace{E[(\mathbf{x} - \mu)(\mathbf{x} - \mu)']}_{=V(\mathbf{x})=V} A' = AVA' \end{aligned} \quad (4.4)$$

Positive definiteness of V

$$\mathbf{y} = a_1x_1 + a_2x_2 + \dots + a_nx_n = a'\mathbf{x} \quad (a' \text{ is a } 1 \times n \text{ vector; compare to above } A \text{ is } k \times n, \text{ set } k=1)$$

$$\Rightarrow V(\mathbf{y}) = a'Va$$

Let V be the variance of \mathbf{x} . V is by definition positive definite if

$$a'Va > 0. \quad \forall a \neq 0$$

But $a'Va$ is the variance of the scalar random variable $\mathbf{y} = \mathbf{x}'a$. We know that the variance of a scalar random variable is nonnegative and that it is zero if and only if \mathbf{y} is a constant (if \mathbf{y} is non-stochastic). Therefore, V is positive definite (or semi-positive definite if the elements of \mathbf{x} are linearly dependent).

4 Some multivariate probability theory and statistics

Standardization. The vector $\mathbf{z} \sim (0, I_n)$ is called standardized vector. It contains n independent random variables with zero mean and variance of 1.

Any given random vector $\mathbf{x} \sim (\mu, V)$ (V positive definite) can be standardized.

$$\nearrow E(\mathbf{y}) = 0$$

- Step 1: $\mathbf{y} = \mathbf{x} - \mu$

$$\searrow V(\mathbf{y}) = V$$

- Step 2: For V positive definite $\exists P$ nonsingular such that $PVP' = Id$.

With P we can define:

$$\nearrow E(\mathbf{z}) = E(P(\mathbf{x} - \mu)) = 0$$

$$\mathbf{z} = P\mathbf{y} = P(\mathbf{x} - \mu)$$

$$\searrow V(\mathbf{z}) = V(P(\mathbf{x} - \mu)) = PVP' = Id$$

$$\Rightarrow \mathbf{z} = P(\mathbf{x} - \mu) \sim (0, Id)$$

If we follow this derivation in the opposite direction, we can create draws from a distribution $\mathbf{x} \sim (\mu, V)$ starting from a normalized distribution. This is used e.g. to create correlated random numbers. The missing piece – how to calculate P – will be answered in section 8.2.

4.3.4 Expectation of a quadratic form

Theorem: Given $\mathbf{x} \sim (0, V)$, consider the quadratic form $\mathbf{x}'A\mathbf{x}$ with A constant and symmetric. From the properties of the trace we get:

$$E(\mathbf{x}'A\mathbf{x}) = trAV$$

Proof:

- (i) $\mathbf{x}A\mathbf{x}'$ is a scalar and therefore equal to its trace

$$\mathbf{x}'A\mathbf{x} = tr \mathbf{x}'A\mathbf{x}$$

- (ii) The trace is commutative

$$tr \mathbf{x}'A\mathbf{x} = tr A\mathbf{x}\mathbf{x}'$$

- (iii) The trace is a linear operator (see P2 of the expectation): $E(tr) = tr(E)$

$$E(tr A\mathbf{x}\mathbf{x}') = tr E(A\mathbf{x}\mathbf{x}') = tr AE(\mathbf{x}\mathbf{x}') = tr AV$$

Note: we would have had to subtract the means to get V , but \mathbf{x} is zero mean.

4.3.5 Variance of a quadratic form

See Graybill, 10.9.10(3).

4.3.6 Law of iterated expectations

To follow.

4.3.7 Covariance matrix of two random vectors

Let $\mathbf{x}_{(m \times 1)} \sim (\mu_X, V_X)$, and $\mathbf{y}_{(n \times 1)} \sim (\mu_Y, V_Y)$ be two random vectors.

Definition 6.

$$\begin{aligned} \text{Cov}(\mathbf{x}, \mathbf{y}) &= E \underbrace{(\mathbf{x} - \mu_{\mathbf{x}})}_{(m \times 1)} \underbrace{(\mathbf{y} - \mu_{\mathbf{y}})'}_{(1 \times n)} = \underbrace{V_{XY}}_{(m \times n)} \\ \text{Cov}(\mathbf{y}, \mathbf{x}) &= E \underbrace{(\mathbf{y} - \mu_{\mathbf{y}})}_{(n \times 1)} \underbrace{(\mathbf{x} - \mu_{\mathbf{x}})'}_{(1 \times m)} = \underbrace{V_{YX}}_{(n \times m)} \end{aligned}$$

therefore: $V_{YX} = V'_{XY}$

$$\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \quad E(\mathbf{z}) = \begin{bmatrix} \mu_X \\ \mu_Y \end{bmatrix} \quad V(\mathbf{z}) = \begin{bmatrix} V_X & V_{XY} \\ V_{YX} & V_Y \end{bmatrix}$$

See also Hrdle, p. 126.

| MATLAB note: To create \mathbf{z} , use the vertical concatenation: $\mathbf{z} = [\mathbf{x}; \mathbf{y}]$;

4.3.8 Multivariate normal distribution

Let x be a univariate (scalar) random variable $x \sim N(\mu, \sigma^2)$. The p.d.f. of x is:

$$\begin{aligned} f(x) &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right) \\ &= (2\pi)^{-\frac{1}{2}} \underbrace{(\sigma^2)}_{V(x)}^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x - \mu)(\sigma^2)^{-1}(x - \mu)\right) \end{aligned}$$

Definition 7. A vector $\mathbf{x} \sim (\mu, \Sigma)$ of n components with Σ positive definite is a normal vector denoted as $\mathbf{x} \sim N(\mu, \Sigma)$ if its p.d.f. is given as

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)' \Sigma^{-1} (\mathbf{x} - \mu)\right)$$

Remarks

- As $f(\cdot)$ is a density, its integral is one: $\int_{-\infty}^{\infty} f(\mathbf{x}) d\mathbf{x} = 1$. Note that this is an n -dimensional integral. This property is useful to solve infinite integrals of some exponential functions.
- Linear transformations of normal variables are again normal distributed.

$$\mathbf{x} \sim N(\mu, \Sigma) \Rightarrow \mathbf{y} = A\mathbf{x} + B \sim N(A\mu + B, A\Sigma A')$$

- If $\mathbf{y} \sim N(0, I_n)$ we speak of a standard normal random distribution. It consists of n independent variables which are all $N(0, 1)$.
- As previously shown, any normally distributed vector $\mathbf{x} \sim N(\mu, \Sigma)$ can be standardized using P such that $P\Sigma P' = I$

$$\mathbf{y} = P(\mathbf{x} - \mu) \sim N(0, Id)$$

- Finally, note that

$$\mathbf{y}'\mathbf{y} = \sum_{i=1}^n y_i^2 \sim \chi_n^2$$

4.4 Exercises

Exercise 4.1 (*). Write down every element of the sigma algebra in example 3.

Exercise 4.2. Multivariate variances and covariances.

- For the 1-dimensional case we know that $var(a+b) = var(a) + var(b) + 2cov(a, b)$. Derive the corresponding expression for the multivariate case, i.e. $var(\mathbf{x} + \mathbf{y})$, where \mathbf{x} and \mathbf{y} are both $n \times 1$ vectors. Start with equation 4.1 and replace $(\mathbf{x} - \mu)$ by $((\mathbf{x} - \mu_x) + (\mathbf{y} - \mu_y))$. Rearrange this expression according to the rules that we have discussed. Make use of definition 6. *Last hint:* The equation in the vector case looks similar to the scalar case. If you want to verify your result, assume $n = 1$. In this case \mathbf{x} and \mathbf{y} become scalars and your result must be equal to the scalar case. Clearly express the dimension of all scalars, matrices and vectors that you use.
- Prove that $cov(A\mathbf{x}, B\mathbf{y}) = Acov(\mathbf{x}, \mathbf{y})B'$ for matrices A, B of suitable dimensions. Clearly express the dimension of all scalars, matrices and vectors that you use.
- Based on (a) and (b), find an expression for $var(A\mathbf{x} + B\mathbf{y})$. Clearly express the dimension of all scalars, matrices and vectors that you use.

Exercise 4.3. Products of random variables.

4 Some multivariate probability theory and statistics

- (a) Consider the one-dimensional random variables $x \sim (0, 1)$ and $y \sim (0, 1)$. x and y are independent. Calculate $E[x \cdot y]$ and $V[x \cdot y]$.
- (b) Consider the two random variables $(x_1, x_2)' \sim (0, \Sigma)$. Calculate $E[x_1 \cdot x_2]$ and $V[x_1 \cdot x_2]$.

5 Some multivariate statistics in MATLAB

Notation: In this chapter only, we use big caps for random variables and small caps for realizations of a random variable. The dimension (vector, matrix) is given separately where necessary.

5.1 One-dimensional case

One-dimensional sample mean. To estimate the expected value for a scalar random variable X , we use the sample mean \bar{x} :

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

Representation of realizations as vectors. We can represent the n realizations of X (1×1) as a column vector x . Note that this is *not* the same thing as a random vector.

$$x = (x_1, x_2, \dots, x_n)'$$

Using a vector of ones, $S = (1, 1, \dots, 1)'$, we can now rewrite the mean of x as

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n 1 \cdot x_i = \frac{1}{n} S'x$$

MATLAB note: There are two ways to do this in MATLAB. The easiest is to use `mean(x)`. Alternatively, one can create the vector S using `S=ones(n,1)` (here S is still a column vector) and the performing the multiplication `m=1/length(x)*S'*x`. The latter is widely used by many MATLAB programmers.

One-dimensional sample variance. To estimate the variance, we use the sample variance:

$$s^2 = \sum_{i=1}^n \frac{(x_i - \bar{x})^2}{n - 1}$$

5 Some multivariate statistics in MATLAB

If we define a vector of error terms $\tilde{x} = x - S\bar{x}$ we can express the sample variance as

$$s^2 = \frac{1}{n-1} \tilde{x}' \tilde{x}$$

MATLAB note: In MATLAB, this is even simpler as we can write

```
xtild = x-m;           (vector minus scalar)
ssqua = 1/(n-1)*xtild'*xtild;
```

5.2 Multidimensional case

Remember the convention that one observation is one row. For a $1 \times k$ random vector X and n observations, we get a $n \times k$ data matrix x . We can generalize the above equations for the one-dimensional case:

Vector of sample means:

$$\underbrace{\bar{x}}_{1 \times k} = \frac{1}{n} \underbrace{S'}_{1 \times n} \underbrace{x}_{n \times k}$$

MATLAB note: The MATLAB command `mean()` is matrix-compatible. If \mathbf{x} is the data matrix as described above, `mean(x)` produces a $1 \times k$ vector of the means. Alternatively, we can perform the above equation explicitly:

```
S=ones(n,1);
m=1/n*S'*x;
```

For the sample variance-covariance matrix, we need the de-meanned terms first:

$$\underbrace{\tilde{x}}_{n \times k} = \underbrace{x}_{n \times k} - \underbrace{S}_{n \times 1} \underbrace{\bar{x}}_{1 \times k}$$

$$\underbrace{\hat{V}}_{k \times k} = \frac{1}{n-1} \underbrace{\tilde{x}'}_{k \times n} \underbrace{\tilde{x}}_{n \times k}$$

MATLAB note: The MATLAB command `cov()` is matrix-compatible. If \mathbf{x} is the data matrix as described above, `cov(x)` produces the $k \times k$ variance-covariance matrix. Alternatively, we can perform the above equation explicitly:

```
xtild = x-S*m;           (vector minus scalar)
ssqua = 1/(n-1)*xtild'*xtild;
```

An alternative expression for the first line is `xtild = x-repmat(m,n,1);`

5.3 PC-Lab: Data handling and descriptive statistics

General rules for handling data. Empirical work is at best as good as the data on which it is based. Special care when handling data should be the standard in today's economics work. Here are a few general rules for working with data:

- Document the data format. Without documentation, data is just a bunch of numbers. Any documentation must contain the exact meaning of each column, including the measurement unit (USD, CHF, ...). Document also where the data starts, i.e. if there are headers. If you get data from outside, request a complete documentation or don't use the data.
- Use only data with known (and quotable) data source.
- Use a standard, plain text file format. Do not save your files in the format **XYZstat**, because your collaborators may not have this program. Worse, in few years from now that program may not exist anymore and you will be unable to use your own data.

MATLAB note: The MATLAB command **save** produces a file that can only be read by MATLAB. It should only be used for short-term storage of intermediate result. For archiving or data exchange with collaborators, use **save -ascii**.
- Consider the use of a standard (SQL) database for large data sets. Setting up a database is not very difficult and many databases can be accessed directly from MATLAB. The benefit is a much faster operation and a cleaner program.

Loading data

Get the files `attend.des` and `attend.txt` from the StudyNet and save them in a place where you can find them. A good idea is to create a new directory called `pclab` directly on the `d:` drive of the PC.

In MATLAB, change the current directory (see Fig. ??, item 1') to the file that you have just created. You are in the right directory if you see the two files in the Current Directory window of MATLAB.

<code>load attend.txt</code>	Load data into a newly created variable <code>attend</code> . See workspace.
<code>[n,k]=size(attend)</code>	Number of rows, number of columns.
<code>attend(1:10,:)</code>	Get a sample of the data. <i>Hint:</i> make the command window wider.
Question:	What is in the data? We could read the description in <code>attend.des</code> or
<code>type attend.des</code>	Type the description file.

5 Some multivariate statistics in MATLAB

Built-in statistics

<code>min(attend)</code>	Minimal values, across columns.
<code>max(attend)</code>	Maximal values, across columns.
<code>mean(attend)</code>	Mean, across columns.
<code>median(attend)</code>	Median, across columns.
<code>mode(attend)</code>	Modus, across columns.
<code>var(attend)</code>	Variances, across columns.
<code>[min();max();...]</code>	A nicer way to output the results in the form of a table.
<code>cov(attend)</code>	Variance-covariance matrix.
<code>diag(cov(attend))</code>	The diagonal elements of the variance-covariance matrix are ...
<code>var(attend)</code>	... of course the variances.
<code>quantile(attend,[.025 .25 .50 .75 .975])</code>	Quantiles. (Statistics toolbox only).

Subsamples

<code>Y=attend(:,2)</code>	Explained variable is second column of <code>attend</code> .
<code>X=[attend(:,1) attend(:, 3) attend(:,7)]</code>	
<code>X=attend(:,[1 3 7])</code>	Interesting explanatory variables are in column no. 1,3 and 7. (This command and the previous one give the same results).
<code>small=attend(1:99,:)</code>	A small subsample with the first 99 observations.
<code>idx=find(X(:,1)>29)</code>	Step 1: <code>find</code> creates a list of entries (row numbers), for which the criterion (attendance > 29) is fulfilled.
<code>Z=X(idx,:);</code>	Step 2: Create a subsample using the list of relevant rows.
<code>Z=X(find(X(:,1)>29),:)</code>	This creates a subsample in which the first variable (attendance) is larger than 29.

Manually calculated statistics

<code>[n k]=size(X)</code>	Dimensions for use below.
<code>S=ones(n,1);</code>	S-vector for use below.
<code>m=1/n*S'*X</code>	Means.
<code>Xt=X-S*m;</code>	De-meaned \tilde{x} .
<code>V=1/(n-1)*Xt'*Xt</code>	Variance-covariance matrix.
<code>cov(X)</code>	Verify results
<code>Xs=sort(X)</code>	Sort X along columns.
<code>Xs(1:10,:)</code>	Verify sorting ...
<code>med=Xs(n/2,:)</code>	This gives an error message.
<code>med=Xs(round(n/2),:)</code>	This is not 100% precise, but close to the median.

Graphics

<code>hist(attend)</code>	This does not make a lot of a sense, because it uses a common scale.
<code>hist(attend(:,1))</code>	Histogram of the first column.
<code>subplot(2,1,1)</code>	Combine two (or more) plots in one. <code>subplot(rows,cols,number)</code>
<code>hist(attend(:,1))</code>	Part 1 of combined plot.
<code>subplot(2,1,2)</code>	Set up for second part.
<code>hist(attend(:,2))</code>	Part 2 of combined plot.
<code>scatter(attend(:,1), attend(:,2),1)</code>	Scatter plot. Is there a correlation between the variables?
<code>plotmatrix(attend)</code>	Matrix of scatterplots / histograms of all variables.
<code>plotmatrix(attend(:,1:6))</code>	Only the first six variables for a better overview.
<code>plotmatrix(attend(:,[1 2 3 7]))</code>	
<code>boxplot(attend)</code>	Boxplot (requires statistics toolbox).
<code>plottools</code>	Opens an interface to fine-tune plots.

Getting time series data

- Go to <http://finance.yahoo.com>
- Enter **AAPL** as a stock symbol and click on **Get Quotes**.
- Click on **Historical Prices**.
- Enter Jan 2, 2006 as starting date and yesterday as end date.
- Click on **Get Prices**.
- Scroll to the end of the page and click on **Download To Spreadsheet**.
- Save the file in the `pclab` directory. Rename the file to **AAPL.csv**

Working with time series data

<code>load AAPL.csv</code>	Does not work. Either delete headers in Excel or use <code>csvread(filename, 1, 1)</code> .
<code>x=AAPL(:,7)</code>	We are only interested in the adjusted close price.
<code>ret=diff(log(x))</code>	Log-returns are the differences of log prices.
<code>mean(ret)</code>	What is the average (daily) return?
<code>hist(ret)</code>	Are stock returns normally distributed?
Question:	Does the variance/correlations change over time?
<code>for i=30:length(x)</code>	To check, we loop over the whole data.
<code>vr(i)=var(x(i-29:i));</code>	Form every day, we look 30 days back and calculate the variance.
<code>end</code>	Every loop ends with an <code>end</code> .
<code>plot(vr(30:end))</code>	We plot starting from 30, because the first 29 entries are empty.

Finally: Create a program file from the Command History. (Shift-click then right-click)

TIP: On first sight, it may seem easier to import data using mouse commands. When doing research, you will usually run this command very often, so programming data import is a good idea.

5.4 Some sources of data

- www.econstats.com
Another huge collection of data files. Some data not for free.
- www.economagic.com
Web site claims to have over 200,000 data files. File download for subscribers only.
- www.crbtrader.com
Financial data (stocks, interest rates, futures, ...) for comparatively low prices (20–100 USD per series).
- finance.yahoo.com
A good source for stock and option prices, including download in spreadsheet format.
- research.stlouisfed.org/fred2/
Federal Reserve Economics Data Server – a lot of US macro data.
- sdw.ecb.int/
The statistical data warehouse of the ECB.
- www.bundesbank.de/statistik/statistik_zeitreihen.en.php
Data from the German Bundesbank.

5.5 Exercises

Exercise 5.1. (a) Create a matrix with 100 draws of a standard-normally distributed random vector with 3 elements.

(b) Calculate the means and the variance-covariance matrix using MATLAB functions.

(c) Re-do (b) step-by-step without using statistics functions from MATLAB.

(d) Interpret the results. What did you expect?

Exercise 5.2 (**). Create a small program (or function) that produces a boxplot *without* using the boxplot command.

Exercise 5.3. Repeat the PC-Lab of this section with a data set of your choice.

6 The linear model and OLS

References: Wooldridge, Appendices D.6 and E; Greene, Appendix A.8

6.1 Derivatives of vectors and matrices

Scalar functions of n variables. A function $y = f(\mathbf{x})$, $\mathbb{R}^n \rightarrow \mathbb{R}$ is a scalar function of n variables. It takes the $n \times 1$ vector $\mathbf{x} = (x_1, x_2, \dots, x_n)'$ as arguments and gives a scalar result y .

Gradient vector. As f depends on several variables, we can calculate several partial derivatives, for example $f_1 = \frac{\partial f}{\partial x_1}$; $f_2 = \frac{\partial f}{\partial x_2}$ and so on. It seems practical to group all these partial derivatives into a vector, the gradient vector \mathbf{g} :

$$\mathbf{g} = \frac{\partial f}{\partial \mathbf{x}} = \begin{pmatrix} \partial f / \partial x_1 \\ \vdots \\ \partial f / \partial x_n \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} \quad (6.1)$$

Interpretation: The gradient has a geometric interpretation. It is the direction of the steepest incline of the function.

MatLab note: This can be used to find a (local) maximum or minimum of a function. Just follow the direction of the steepest incline to find the maximum (viz. the opposite direction to find the minimum). See the MatLab commands `fminunc`, `fmincon`.

It is a convention that derivatives with respect to \mathbf{x} are always column vectors, while derivatives with respect to \mathbf{x}' are always row vectors. We can write:

$$\frac{\partial f}{\partial \mathbf{x}'} = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right) \quad \text{and} \quad \frac{\partial f}{\partial \mathbf{x}} = \left(\frac{\partial f}{\partial \mathbf{x}} \right)'$$

Hessian matrix. If the denominator itself is a vector, applying the gradient yields a matrix. One common application is the Hessian matrix, the multivariate equivalent of the second derivative. Remember that $\frac{\partial f}{\partial \mathbf{x}}$ is already a (column) vector. Take the first element (f_1) and calculate the derivative with respect to \mathbf{x}' :

$$\frac{\partial f_1}{\partial \mathbf{x}'} = \left(\frac{\partial f_1}{\partial x_1}, \dots, \frac{\partial f_1}{\partial x_n} \right) = \left(\frac{\partial^2 f}{\partial x_1^2}, \dots, \frac{\partial^2 f}{\partial x_1 \partial x_n} \right)$$

6 The linear model and OLS

Repeating this for every element of the first derivative gives the Hessian matrix:

$$\frac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{x}'} = \begin{pmatrix} \frac{\partial f_1}{\partial \mathbf{x}'} \\ \vdots \\ \frac{\partial f_n}{\partial \mathbf{x}'} \end{pmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \quad (6.2)$$

Interpretation: the Hessian matrix can be used to decide whether an extreme value is a maximum or minimum; if it is positive definite, it's a minimum, if it is negative definite, it's a maximum.

Derivative of a linear form

A linear form $a' \mathbf{x} = \mathbf{x}' a$ is a scalar

$$y = a' \mathbf{x} = x_1 a_1 + x_2 a_2 + \cdots + x_n a_n = \mathbf{x}' a$$

The partial derivatives are

$$\frac{\partial a' x}{\partial x_1} = a_1 \quad \frac{\partial a' x}{\partial x_i} = a_i$$

$$\boxed{\frac{\partial a' x}{\partial x} = a \quad \frac{\partial x' a}{\partial x} = a}$$

Derivative of a projection

Take a projection

$$\underbrace{\mathbf{y}}_{k \times 1} = \underbrace{A}_{k \times n} \underbrace{\mathbf{x}}_{n \times 1}$$

Interpret matrix A as a set of row vectors

$$A = \begin{bmatrix} a'_1 \\ \vdots \\ a'_k \end{bmatrix}$$

Then $y_1 = a'_1 \mathbf{x}$ and $\frac{\partial y_1}{\partial \mathbf{x}} = \frac{\partial a'_1 \mathbf{x}}{\partial \mathbf{x}} = a_1$. Repeating this for all elements of A gives

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_k}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \cdots & \frac{\partial y_k}{\partial x_n} \end{bmatrix} = [a_1 \dots a_k] = A'$$

$$\boxed{\frac{\partial A x}{\partial x} = A' \quad \frac{\partial A x}{\partial x'} = A}$$

Derivative of a quadratic form

The product rule is valid in the vector case, as well: $\frac{\partial}{\partial \mathbf{x}}(f \cdot g) = \frac{\partial f}{\partial \mathbf{x}}g + f\frac{\partial g}{\partial \mathbf{x}}$.

$$\frac{\partial(\mathbf{x}'A\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial(\mathbf{x}'A)}{\partial \mathbf{x}}\mathbf{x} + \underbrace{\mathbf{x}'A}_{a'}\frac{\partial \mathbf{x}}{\partial \mathbf{x}} = A\mathbf{x} + \frac{\partial a'\mathbf{x}}{\partial \mathbf{x}} = A\mathbf{x} + a = A\mathbf{x} + \underbrace{A'\mathbf{x}}_a$$

$$\boxed{\frac{\partial(\mathbf{x}'A\mathbf{x})}{\partial \mathbf{x}} = (A + A')\mathbf{x}} \quad \text{If } A \text{ is symmetric: } \boxed{\frac{\partial(\mathbf{x}'A\mathbf{x})}{\partial \mathbf{x}} = 2A\mathbf{x}}$$

6.2 The linear model in matrix notation

So far, we have considered a one-dimensional representation of the linear model:

$$y = \beta_0 + \beta_1x_1 + \beta_2x_2 + \cdots + \beta_kx_k + \varepsilon$$

Normally, we observe several individuals i . This can be written by adding an additional index i :

$$y_i = \beta_0 + \beta_1x_{1i} + \beta_2x_{2i} + \cdots + \beta_kx_{ki} + \varepsilon_i$$

Note that the β have no index, because they are assumed to be constant among individuals. In a first step, we can collect the β and x_k terms into (column) vectors¹:

$$\beta = (\beta_0, \beta_1, \dots, \beta_k)' \quad \mathbf{x}_i = (1, x_{1i}, \dots, x_{ki})'$$

This simplifies the above equation to

$$y_i = (\mathbf{x}_i)'\beta + \varepsilon_i$$

Next we can collect all individuals into

$$\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}_{n \times 1} \quad X = \begin{bmatrix} \mathbf{x}'_1 \\ \vdots \\ \mathbf{x}'_n \end{bmatrix}_{n \times (k+1)} \quad \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}_{n \times 1}$$

The $(\mathbf{x}_i)'$ have already been $1 \times (k+1)$ vectors, thus X is a $n \times (k+1)$ matrix that follows the convention that one individual is in one row and one variable is in one column.

¹Note that Wooldridge defines \mathbf{x} as a row vector, which changes the following equation to $y^i = \mathbf{x}^i\beta' + \varepsilon^i$. We prefer a consistent notation in which every vector is a column vector, unless transposed.

6 The linear model and OLS

This gives the linear model in matrix notation (later denoted as assumption MLR.1):

$$\boxed{\underbrace{\mathbf{y}}_{n \times 1} = \underbrace{X}_{n \times (k+1)} \underbrace{\beta}_{(k+1) \times 1} + \underbrace{\varepsilon}_{n \times 1}} \quad (6.3)$$

6.3 Least squares

Objective: find β that minimizes the squared error term ($\varepsilon'\varepsilon = \text{scalar}$) in (6.3). Calculate the squared error term and form the first derivative with respect to β

$$\begin{aligned} \varepsilon &= \mathbf{y} - X\beta \\ \varepsilon'\varepsilon &= (\mathbf{y} - X\beta)'(\mathbf{y} - X\beta) \\ &= \mathbf{y}'\mathbf{y} - \underbrace{\beta'X'\mathbf{y}}_{\beta'a} - \underbrace{\mathbf{y}'X\beta}_{a'\beta} + \underbrace{\beta'X'X\beta}_{\beta'X\beta} \\ \frac{\partial}{\partial \beta} \varepsilon'\varepsilon &= 0 - X'\mathbf{y} - X'\mathbf{y} + 2(X'X)\beta \stackrel{!}{=} 0 \\ (X'X)\beta &= X'\mathbf{y} \\ \hat{\beta} &= (X'X)^{-1}X'\mathbf{y} \end{aligned} \quad (6.4)$$

See exercise 2.4(a) for a discussion when expression (6.4) exists.

Is this really a minimum? Build the second derivative (Hessian)

$$\frac{\partial^2}{\partial \beta \partial \beta'} \varepsilon'\varepsilon = \frac{\partial}{\partial \beta'} \left[0 - X'\mathbf{y} - X'\mathbf{y} + 2(X'X)\beta \right] = 2(X'X)$$

As we have shown in the previous section, $(X'X)$ is positive definite, so indeed, it is a minimum.

Note that we made no assumptions on the properties of the error term. This is only least squares and (not yet) OLS. We can find an optimal β , however we cannot make any statement on the properties of the estimator for β .

So is this the solution? We made no assumptions on the error term, the sampling, on independence ... and still got some result. However, we will not be able to answer important questions about the *validity* or *quality* of our result. Answering these questions is vital for statistical inference.

STEP 1	$\mathbf{y} = X\beta + \varepsilon$ CHOICE
STEP 2	$\min \Sigma \varepsilon_i^2 = \varepsilon' \varepsilon$ CHOICE (check ass. equality!)
STEP 3	$\hat{\beta} = (X'X)^{-1}X\mathbf{y}$ CALCULATION – 3.1 X has full rank – 3.2 $n \geq (k + 1)$ – conditions for existence of solution
STEP 4	$\hat{\beta}$ minimizes $\Sigma \varepsilon_i^2$ CALCULATION (verification)

Important observation:

- $\hat{\beta}$ can be calculated from the data (feasibility)

Questions:

- Quality / reliability of $\hat{\beta}$ (significance?)
- Does $\hat{\beta} \rightarrow \beta$ (unbiasedness?)
- Is there a better way to calculate $\hat{\beta}$ (efficiency?)

6.4 Estimators

Starting point: Population distribution Y with unknown parameter θ .

Sample: A draw from the distribution: $(y_1, y_2, \dots, y_n)'$

Estimator W : A function h of the draw: $W = h(y_1, y_2, \dots, y_n)$

Estimator is a random variable,
because it depends on a random sample.

We can calculate moments of the estimator, e.g. $E(W)$.

Feasibility: estimator is a function of the observations

Relevant properties of estimators

Unbiasedness: $E(W) = \theta$ Otherwise $bias = \theta - E(W)$

Relative Efficiency of two estimators: $V(W_1) \leq V(W_2) \quad \forall \theta$

Consistency

Can we assess the properties of our $\hat{\beta}$?

If not, it is not an estimator.

6.5 Multiple linear regression model: Assumptions

Notation: The numbering follows the lecture “Empirische Wirtschaftsforschung”, which is different from Wooldridge, section 3 and appendix E.

MLR.1 Linearity in parameters

$$\mathbf{y} = X\beta + \varepsilon$$

MLR.2 Zero conditional mean

$$E(\varepsilon|X) = 0$$

MLR.3 Random sampling

This is implied in the fact that we have $\mathbf{y}_{n \times 1}$ and $X_{n \times (k+1)}$

MLR.4 No perfect collinearity

$$rk(X) = k + 1$$

6.5.1 Unbiasedness of the OLS estimator

Objective: show that $E(\hat{\beta}) = \beta$. We first rewrite (6.4) and then calculate the expected value. Note that $E[\beta] = \beta$ and $Cov(\beta, \hat{\beta}) = 0$, because β as the true parameter is not random.

$$\begin{aligned} \hat{\beta} &= (X'X)^{-1}X'\mathbf{y} \\ &= (X'X)^{-1}X'(X\beta + \varepsilon) \\ &= (X'X)^{-1}X'X\beta + (X'X)^{-1}X'\varepsilon \\ &= \beta + (X'X)^{-1}X'\varepsilon \\ E[\hat{\beta}] &= \beta + E[(X'X)^{-1}X'\varepsilon] \\ &= \beta + E[E[(X'X)^{-1}X'\varepsilon|X]] \\ &= \beta + E[(X'X)^{-1}X' \underbrace{E[\varepsilon|X]}_{=0 \text{ (MLR.2)}}] \end{aligned} \tag{6.5}$$

For the last but one step, we used the law of iterated expectations: $E[f] = E[E[f|X]]$.

6.5.2 Variance-covariance matrix of the OLS estimator

Objective: find the variance of $\hat{\beta}$ given X . In order to do so, we need one additional assumption:

MLR.5 Homoskedasticity and no serial correlation

$$\text{Var}(\varepsilon|X) = \sigma^2 Id$$

We start from (6.5) and calculate the conditional variance :

$$\text{Var}(\hat{\beta}|X) = E[(\hat{\beta} - E[\hat{\beta}])(\hat{\beta} - E[\hat{\beta}])'] \quad (6.6)$$

$$= E[(X'X)^{-1}X'\varepsilon|X](X'X)^{-1}X'\varepsilon|X] \quad (6.7)$$

$$= \text{Var}[(X'X)^{-1}X'\varepsilon|X]$$

$$= (X'X)^{-1}X' \text{Var}[\varepsilon|X] X(X'X)^{-1}$$

$$= (X'X)^{-1}X' \sigma^2 Id X(X'X)^{-1}$$

$$= \sigma^2 (X'X)^{-1}X'X(X'X)^{-1}$$

$$\text{Var}(\hat{\beta}|X) = \sigma^2 (X'X)^{-1} \quad (6.8)$$

$$sd(\hat{\beta}|X)_i = \sqrt{\sigma^2 (X'X)^{-1}_{ii}} = \sigma \sqrt{(X'X)^{-1}_{ii}} \quad (6.9)$$

Problem: we do not know σ , we need to estimate it, see section 6.5.4.

6.5.3 Gauss-Markov Theorem

Theorem 2 (Gauss-Markov Theorem). *Under the assumptions MLR.1 through MLR.5, the OLS estimator $\hat{\beta}$ is the best linear unbiased estimator (BLUE).*

Proof. We need to show that the OLS estimator is “best”, which means it has the lowest variance. We do this by assuming an alternative estimator and by showing that *any* alternative estimator is worse than the OLS estimator.

Step1 Any linear estimator of β can be written using $A_{n \times (k+1)}$

$$\tilde{\beta} = A'y$$

Step 2 The alternative estimator $\tilde{\beta}$ must be unbiased $E[\tilde{\beta}|X] = \beta$. Using $y = X\beta + \varepsilon$,

$$E[\tilde{\beta}|X] = E[A'(X\beta + \varepsilon)|X]$$

We can only proceed, if we can take A' out of the conditional expectation. This is possible if A is constant or a function of X (but not, for example, a function of y).

$$E[\tilde{\beta}|X] = A'X\beta + A'E[\varepsilon|X]$$

6 The linear model and OLS

Step 3 Using MLR.2 the second term vanishes. Remembering that unbiasedness means $E[\tilde{\beta}|X] = \beta$, we get the expression

$$A'X\beta = \beta$$

which implies that $A'X = Id$. This result will be used in step 5.

Step 4 Now we can calculate the conditional variance of $\tilde{\beta}$ using MLR.5:

$$Var(\tilde{\beta}|X) = A'Var(\varepsilon|X)A = \sigma^2 A'A$$

Step 5 Is this expression larger or smaller than $Var(\hat{\beta}|X) = \sigma^2(X'X)^{-1}$? To find out, we calculate the difference:

$$\begin{aligned} Var(\tilde{\beta}|X) - Var(\hat{\beta}|X) &= \sigma^2[A'A - (X'X)^{-1}] \\ &= \sigma^2[A'A - A'X(X'X)^{-1}X'A] && \text{with } A'X = Id \\ &= \sigma^2 A'[Id - X(X'X)^{-1}X']A \\ &= \sigma^2 A'MA \end{aligned} \tag{6.10}$$

with $M = Id - X(X'X)^{-1}X'$. This matrix is symmetric and idempotent and therefore positive semi-definite (see Wooldridge, Appendix D.5). This concludes the proof that $\hat{\beta}$ is BLUE.

6.5.4 Estimating the variance $\hat{\sigma}$

MLR.5 states that $Var(\varepsilon|X) = \sigma^2 Id$. How can we estimate σ^2 ?

Theorem 3. *The unbiased estimator of the error variance σ^2 is*

$$\hat{\sigma}^2 = \frac{\hat{\varepsilon}'\hat{\varepsilon}}{n - k - 1} \tag{6.11}$$

Proof. We need to show that $E[\hat{\sigma}^2|X] = \sigma^2$. We start with the expression for $\hat{\varepsilon}$:

$$\hat{\varepsilon} = \mathbf{y} - X\hat{\beta} = \mathbf{y} - X(X'X)^{-1}X'\mathbf{y} = M\mathbf{y} = M\varepsilon$$

For the last step, we used $\mathbf{y} = X\beta + \varepsilon$ and $MX = 0$. Next, we calculate $\hat{\varepsilon}'\hat{\varepsilon}$

$$\hat{\varepsilon}'\hat{\varepsilon} = \varepsilon'M'M\varepsilon = \varepsilon'M\varepsilon$$

where we made use of the fact that M is symmetric and idempotent. The expression $\varepsilon'M\varepsilon$

6 The linear model and OLS

is a scalar, so it equals its trace:

$$\begin{aligned}
 E[\varepsilon' M \varepsilon | X] &= E[\text{tr}(\varepsilon' M \varepsilon) | X] = E[\text{tr}(\varepsilon' \varepsilon M) | X] \\
 &= \text{tr}[E(M \varepsilon \varepsilon') | X] = \text{tr}[M E(\varepsilon \varepsilon') | X] \\
 &= \text{tr}(M \sigma^2 Id) = \sigma^2 \text{tr}(M) = \sigma^2(n - k - 1)
 \end{aligned} \tag{6.12}$$

where $\text{tr}(M) = \text{tr}(Id_n) - \text{tr}(X(X'X)^{-1}X') = n - \text{tr}(Id_{k+1}) = n - k - 1$. To conclude

$$E(\hat{\sigma}^2 | X) = E\left(\frac{\varepsilon' M \varepsilon | X}{n - k - 1}\right) = \sigma^2$$

Finally, define a standard error of $\hat{\beta} | X$

$$s.e.(\hat{\beta}_i | X) = \hat{\sigma} \sqrt{(X'X)^{-1}_{ii}}$$

6.6 Statistical inference and testing

In order to be able to construct a test statistic, we need to make one final assumption:

MLR.6 Normality of errors. Conditional on X , ε_i are i.i.d. (independent and identically distributed) as: $\varepsilon_{n \times 1} \sim N(0, \sigma^2 Id)$

6.6.1 Normality of $\hat{\beta} | X$

Theorem 4. Under the assumptions *MLR.1* through *MLR.6*,

$$\hat{\beta} | X \sim N(\beta, \sigma^2 (X'X)^{-1}) \tag{6.13}$$

Proof. The proof is similar to section 6.5.2, just that we use *MLR.6* instead of *MLR.5*, which gives us also information about the distribution of $\varepsilon | X$.

6.6.2 t-statistic

Theorem 5. Under the assumptions *MLR.1* through *MLR.6*,

$$\frac{\hat{\beta}_j - \beta_j}{se(\hat{\beta}_j)} \sim t_{n-k-1} \tag{6.14}$$

Proof.

Step 1 We start with the distribution of $\hat{\beta} | X$ (6.6) and standardize it using $sd(\hat{\beta}_j | X) =$

$$\sqrt{\text{var}(\hat{\beta}_j|X)} = \sigma \sqrt{((X'X)^{-1})_{jj}}$$

$$\frac{\hat{\beta}_j - \beta_j}{\text{sd}(\hat{\beta})} \Big| X \sim N(0, 1)$$

Step 2 Next we observe that

$$\frac{(n - k - 1)\hat{\sigma}^2}{\sigma^2} \Big| X = \left(\frac{\varepsilon}{\sigma}\right)' M \left(\frac{\varepsilon}{\sigma}\right) \Big| X \sim \chi_{n-k-1}^2$$

because $\frac{\varepsilon}{\sigma} \Big| X \sim N(0, Id)$ (normalized from MLR.6) and M is symmetric, idempotent and has rank $n - k - 1$.

Step 3 We show that $\hat{\beta}$ and $\hat{\sigma}$ are independent. We use a theorem from Wooldridge, Appendix D 5, page 817 that for $\mathbf{y} \sim N(0, \sigma^2 Id)$ the expressions $A\mathbf{y}$ and $\mathbf{y}'B\mathbf{y}$ are independent if and only if $AB=0$. With $\hat{\beta} = \beta + (X'X)^{-1}X'\varepsilon$ and $\hat{\sigma}^2 = \varepsilon' M \varepsilon / (n - k - 1)$, we need to show that $(X'X)^{-1}X' \cdot M$ is zero. This is the case.

Step 4 We re-write the initial expression as

$$\frac{\hat{\beta}_j - \beta_j}{\text{se}(\hat{\beta}_j)} = \frac{\frac{\hat{\beta}_j - \beta_j}{\text{sd}(\hat{\beta}_j)}}{\sqrt{\frac{\hat{\sigma}^2}{\sigma^2}}} = \frac{\sim N(0, Id)}{\sim \sqrt{\frac{\chi_{n-k-1}^2}{n-k-1}}} \sim t_{n-k-1} \quad (6.15)$$

The last step is simply applying the definition of the t-distribution. Note that $\text{se}(\hat{\beta}_j) = \frac{\hat{\sigma}}{\sigma} \text{sd}(\hat{\beta}_j)$. We were allowed to suppress the condition on X , because the t -statistic is independent of X , and if the right side of an equation is independent of X , the left side must be independent, as well.

This result can be used to test hypothesis about β . The standard test is to see if $\hat{\beta}_j$ is significantly different from zero. In this case, we use zero for β_j . See PC-Lab.

6.7 Scale dependence of the β coefficients

Assume that the X are expressed in a different unit of measurement, i.e. in percent $\in [0..100]$ instead of fractions $\in [0..1]$. We show that this will change the values of β . To simplify the calculation, assume all variables in X are multiplied with a fixed α . In the above example, $\alpha = 100$. We define a new set of explanatory variables \tilde{X} . \mathbf{y} should remain unchanged.

$$\begin{aligned}
\tilde{X} &= \alpha X \\
\hat{\tilde{\beta}} &= (\tilde{X}\tilde{X}')^{-1}\tilde{X}\mathbf{y} \\
&= \frac{1}{\alpha^2}(XX')^{-1}\alpha X\mathbf{y} \\
&= \frac{1}{\alpha}\hat{\beta}
\end{aligned} \tag{6.16}$$

6.8 PC-Lab: Linear equations and OLS

In this PC-Lab, we perform an OLS estimation including testing for significance “manually” step-by step. It is clear that this can be done much faster with the built-in functions of MatLab or with a dedicated statistics program.

We have two different goals here. First, we can watch an estimation step-by-step at work, which might help understanding the estimation process. Second, we learn how to program a known and simple estimator. This may be of use in later stages courses or in research, when having to program novel estimators, which are not yet implemented in a statistics program. We use once more the example `attend.txt` and `attend.des`.

MatLab note: Normally, all intermediate calculations should end with a semicolon (;) to suppress unnecessary output. Throughout this PC-lab, however, it is suggested to end all lines without semicolon, to see what is happening.

Preparation

<code>load attend.txt</code>	Load data.
<code>[n,k]=size(attend)</code>	Number of rows, number of columns.
<code>Y=attend(:,2);</code>	New explained and explanatory variables.
<code>X=[ones(n,1) attend(:,1) attend(:,3) attend(:,7)]</code>	
<code>[n,k]=size(X)</code>	We have changed the sample.
<code>type attend.des</code>	Check again the data description.

OLS estimate with MatLab

Note: `regress` requires the statistics toolbox.

`b=regress(Y,X)` Simple regression. (We write `b` for β to save space).

Question: Interpret the result.

`mean(X)` The means of each observable in `X`.

`mean(Y)` The mean grade.

`mean(X)*b` This is the same as `mean(Y)`. Why?

`[b, biv]=regress(Y,X)` 95% confidence intervals for β .

`[biv(:,1) b biv(:,2)]` Lower bound, point estimate, upper bound

Now the full version of `regress`.

`[b, biv, r, riv, stats]=regress(Y,X)`

`r` The residuals. Let us check one:

`r(1)` Residual of first observation ...

`Y(1)-X(1,:)*b` and “manual” calculation

`riv(1:5,:)` These are intervals to detect outliers. How are they calculated?

`Y(1)-X(1,:)*biv(:,2)` Lower bound for admissible errors.

`Y(1)-X(1,:)*biv(:,1)` Upper bound for admissible errors.

Note: If you don’t like `riv`, ignore it and go on.

`stats` These are: R^2 , F -stat, the p -value of the F -stat and $\hat{\sigma}^2$.

Question: Interpret the results.

Step-by-step OLS estimate

<code>b2=inv(X'*X)*X'*Y</code>	Compare this with the previous <code>b</code> .
<code>u=X*b-Y</code>	The residuals (we write <code>u</code> for ε .)
<code>mean(u)</code>	This is not a verification of MLR.2, this is by construction of the estimator.
<code>rank(X)</code>	Check MLR.4
<code>rank(X'*X)</code>	Same.
<code>SST=(Y-mean(Y))'*(Y-mean(Y))</code>	This is equivalent to $SST = \sum_{i=1}^n (y_i - \bar{y})^2$.
<code>SSR=u'*u</code>	This is equivalent to $SSR = \sum_{i=1}^n \varepsilon_i^2$.
<code>R2=1-SSR/SST</code>	$R^2 = 1 - SSR/SST$
<code>si2hat=u'*u/(n-k-1)</code>	Estimate $\hat{\sigma}^2$.
<code>sihat=sqrt(si2hat)</code>	And calculate $\hat{\sigma}$.
<code>Xinv=inv(X'*X)</code>	First step towards standard errors.
<code>dXi=diag(Xinv)</code>	New command <code>diag</code> , self-explaining.
<code>sdXi=sqrt(diag(Xinv))</code>	Vector of $\sqrt{((X'X)^{-1})_{jj}}$
<code>seb=sihat*sdXi</code>	$se(\hat{\beta}_j) = \sigma \sqrt{((X'X)^{-1})_{jj}}$
<code>tval=ttinv(.975,n-k-1)</code>	Value of the t-statistic for 95% confidence interval.
<code>bmin=b-tval*seb</code>	Lower bound.
<code>bmax=b+tval*seb</code>	Upper bound.
<code>[b seb]</code>	Output β and its standard errors.
<code>[bmin b bmax]</code>	Output the confidence interval and the point estimate of β .
<code>tstat=b./seb</code>	Values for the t-statistic. Note the dot (.) before the division operator. The dot means "element-by-element".
<code>pval=2*tcdf(-abs(tstat),n-k-1)</code>	
<code>[b seb tstat pval]</code>	The p-values and a typical output.

6.9 Exercises

Exercise 6.1. (a) Verify that M in (6.10) is symmetric and idempotent by calculating M' and M^2 . (b) Verify that $MX = 0$.

Exercise 6.2. (a) Why did we only take a few explanatory variables of `attend` when we declared `X=[ones(n,1) attend(:,1) attend(:,3) attend(:,7)]`? (b) Show with MatLab, that taking all explanatory variables in `attend` would not work. (c) Find the maximum number of explanatory variables of `attend`, for which we can still perform an OLS regression. (d) Give one example of such a set of variables. *Hint:* there is a very convenient way of finding variables that are plain multiples of another one in X . Try to use this method. If you cannot find it, you can use trial and error. (d) Repeat the PC-Lab with the new, larger data set. (e) If possible, compare the results. Did anything change?

Exercise 6.3. Start with `X=[ones(n,1) attend(:,1) attend(:,3) attend(:,7)]`; and `Y=attend(:,2)`. (a) create five variables called X_1 through X_5 , so that each X_i contains 100 observations: X_1 should contain the first hundred observations of X , X_2 the second hundred observations and so on. Create the corresponding variables Y_1 through Y_5 . (b) Perform an OLS regression of Y_1 on X_1 , Y_2 on X_2 and so on. Compare the results and make a comment. (c) Would it make sense to regress Y_1 on X_2 and so on?

Exercise 6.4. Take X and Y from the previous exercise. (a) perform a full regression of Y on X , including t-statistics. (b) Change the scale of 'homework' such that it ranges from 0.125 to 1 instead of [12.5, 100]. Repeat the regression. How do the β coefficients change? Does the significance levels change, as well? (c) What do we learn from this exercise about interpreting β -values?

Exercise 6.5. In chapter 7.4 we illustrate the bias of the OLS estimator once MLR.2 is violated. Now we calculate this bias for the case $k = 1$. Assume that X is made of a column of ones and of x_1 , which is uniformly distributed between 0 and 1. Furthermore, the error term be correlated with x_1 in the form $\varepsilon_1 = \alpha x_{1,i} + 0.5\zeta$, where $\zeta \sim N(0,1)$, independent of everything else. (In chapter 7.4, we set α equal to 0.2, 0.5 and 1). Here, we want to find a solution for a general α . Go back to equation 6.5 and calculate the bias $E[\hat{\beta}] - \beta$ using the new facts. Do not forget that the result is a 2×1 vector.

Hints: Do not focus on the expectation or the trick with the law of iterated expectations in the beginning, find the expression in the bracket first. The key to the solution is to find a clever way to write ε as a function of X .

Exercise 6.6. Try the estimator for $\hat{\sigma}$ for $n = k + 1; n = k + 2; n = k + 3; \dots$

7 Monte Carlo simulation

References: Greene, Appendix E.3; Brandimarte chapters 4 and 7; Judd, section 8

7.1 An introduction to simulation-based methods

The basis for every simulation is a model: a fair coin has a 50% probability of heads and tails, a dice has a $1/6$ probability for every number. Once we have a model for the behaviour of a coin, a dice, an economy or the stock-market, it is an obvious step to simulate it.

Example 6 (Rolling a dice in MATLAB.). MATLAB (and most other programming environments) can only produce uniformly distributed random variables between zero and one. We need to transform this random variable to suit our needs. In most cases, this is a two step process: first, multiplying with 6 gives a uniformly distributed random variable between zero and 6. Second, rounding up to the next integer gives the desired transformation. The mathematical notation is $x \rightarrow \lceil 6 \cdot x \rceil$. In MATLAB, rolling one dice looks like this:

```
% Rolling a dice in MATLAB
dice=ceil(6*rand);
```

7.1.1 Producing random numbers in MATLAB

MATLAB – just as any other computer program – cannot produce really random numbers. It rather produces a deterministic sequence of numbers that has almost no correlation, which are called *pseudo-random numbers*. When MATLAB is start up, the random number generator is initialized and starts producing the same deterministic series. One can manually initialize the random number generator. This can be useful if one wants to repeat exactly the same simulation.

7 Monte Carlo simulation

Distribution	probability fn. p.d.f.	MATLAB command	See ...
Bernoulli 0/1	$P(0) = P(1) = \frac{1}{2}$	<code>round(rand)</code>	
Symmetric Bernoulli	$P(-1) = P(1) = \frac{1}{2}$	<code>sign(randn)</code>	
Bernoulli	$P(0) = q; P(1) = p$	<code>rand<=p</code>	
Binomial (n,p)	$P(k) = \binom{n}{k} p^k q^{n-k}$	<code>sum(rand(n,1)<=p)</code>	
Uniform [0...1]		<code>rand</code>	
Uniform [a...b]		<code>a+(b-a)*rand</code>	
Standard normal		<code>randn</code>	
Correlated n-dim. norm.	$\mathbf{x} \sim N(\mu, \Sigma)$	<code>mu+randn(n,1)*chol(Sigma)</code> or <code>mvnrnd(mu,sigma)</code>	p. 81

Table 7.1: Random number generation in MATLAB.

Example: random number generation

<code>rand('twister',10);</code>	Set the state of the random number generator
<code>rand(1,5)</code>	Produce five random numbers
<code>rand(1,5)</code>	Produce five different ones
<code>rand('twister',10);</code>	Reset the random generator
<code>rand(1,5)</code>	Produce the same five random numbers

General procedure for discrete distributions

Imagine a discrete distribution that can take the values $1, 2, \dots, n$ with the probabilities $p = (p(1), p(2), \dots, p(n))'$. Obviously, $p_i > 0$ and $\sum p_i = 1$. To produce draws from this distribution, start with uniformly distributed random numbers and imagine that the interval $[0, 1]$ is divided into intervals with the lengths $p(1), p(2), \dots, p(n)$. Our discretely distributed random number is the number of the interval in which the uniform random number lies.

<code>p = [0.1 0.2 0.3 0.4];</code>	small 'p': probabilities $p(1) \dots p(4)$
<code>P = cumsum(p);</code>	large 'P': cumulative probability
<code>P = [0 P(1:end-1)];</code>	lower bracket values
<code>r = sum(rand>P);</code>	random value

General procedure for continuous distributions

The standard way to produce random numbers of any continuous distribution is to use the inverse function of the cdf: $G(y) = F^{-1}(x)$. If u is uniformly random distributed on $[0, 1]$, then $G(u)$ is distributed with distribution $F(\cdot)$.

7 Monte Carlo simulation

Example 7 (Pareto distribution). The pdf of the pareto distribution is $f(x) = \frac{\alpha}{(1+x)^{1+\alpha}}$ for $x > 0$. The cdf is accordingly $F(x) = 1 - (1+x)^{-\alpha}$. Its inverse is then $G(y) = (1-y)^{-\frac{1}{\alpha}} - 1$ ¹

```
| (1-rand)^(-1/alpha) - 1
```

7.2 A first Monte Carlo example

Example 8. The number of points on one side of a dice be x . What is the expected value $E[x]$ if we throw the dice once?

Theoretical calculation. $E[x] = \frac{1}{6}(1 + 2 + 3 + 4 + 5 + 6) = 3.5$

Experimental solution. We roll a real dice 20 times and take the average. The result is, for example, $\mu(x) = 3.4$.

MATLAB simulation. Let MATLAB roll the dice n times and let MATLAB calculate the mean.

```
| Example: A first simple simulation program dicer.m
n=100; % Good style: define n in the beginning
dice=ceil(6*rand(1,n)); % Roll 100 dice in one line
disp(dice(1:20)); % output the first few rolls for verification
disp(mean(dice));
```

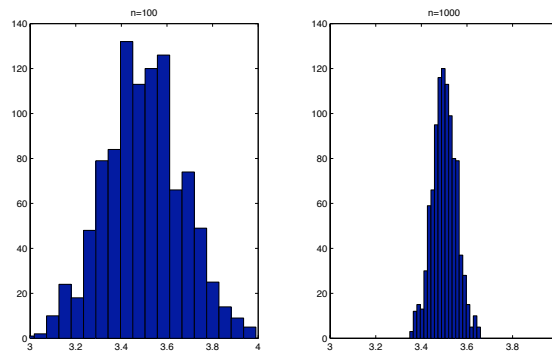


Figure 7.1: The central limit theorem in action: Increasing n yields a more precise simulation result. Each simulation was repeated 1000 times using `manydice.m`

Observations

¹To see why, solve for y :

$$\begin{aligned} y &= 1 - (1+x)^{-\alpha} \\ 1-y &= (1+x)^{-\alpha} \\ (1-y)^{-\frac{1}{\alpha}} &= 1+x \end{aligned}$$

7 Monte Carlo simulation

- The results are close to the theoretical value, but not exact.
- Every time we repeat the simulation, we get a slightly different value.
- If we repeat the simulation many times, we can plot a distribution, see Fig. 7.1.
- If we repeat the simulation with $n=1000$, the distribution becomes more narrow.

Conclusions

- The mean is a good estimator for the expected value.
- The result of a simulation is a random variable.
- By repeating the simulations, we can assess its properties.
- If we increase n , the variance of this random variable decreases.

Two questions immediately arise. (1) Can we always use the mean as an estimator for the expectation or were we just lucky? (2) Does the variance of the estimation always decrease if we increase n ? By how much? The next sections will answer them.

Why simulate? Example 8 is not breathtaking. We got a result (≈ 3.5) that we already knew, just with less precision. So is simulation any good? Not, if we are able to calculate the theoretical values. This is, however, not possible or not practical in many cases.

Simulation is the method of choice for stochastic problems that are either too difficult to solve analytically or for which it is known that analytical solutions do not exist. The disadvantage of delivering the result in the form of a random variable is outweighed by the fact that we can repeat the simulation as often as we want and therefore assess its precision.

7.3 The Monte Carlo Method

The Monte Carlo method is a numerical method of solving mathematical problems by random sampling. Simpler put, it is a method for calculating expectations. It was developed at the Los Alamos National Laboratory, the first US atomic bomb factory to calculate neutron diffusion in atomic bombs. The first paper about this method is from Metropolis and Ulam in 1949. The Monte Carlo method got its name after the city of Monte Carlo, which is famous for its casinos, see Fig. 7.3. Note that the spinning wheel in a casino is one of the simplest mechanical devices to produce random numbers.

7.3.1 The method to calculate x

To calculate an unknown quantity x , we have to first construct a *probabilistic model* (often quite obvious) and follow a few simple steps.

7 Monte Carlo simulation

1. Find a random variable ξ such that $E[\xi] = x$.
Very often, this is trivial, as the desired quantity is $E[\xi]$.
2. Produce n independent random draws $\xi_1, \xi_2, \dots, \xi_n$.
3. Use the mean as an estimator for the expected value (thanks to the LLN):

$$\hat{x} = \hat{\mu}_n = \frac{1}{n} \sum \xi_i.$$

4. Estimate $Var(\mu)$ (see below, MCSE) and verify if it is below the desired value. If not, increase n accordingly and repeat steps 2 and 3.

Reflection. At first sight, step (1) may seem difficult. How could one find a suitable ξ that fulfills the condition $E[\xi] = x$ in order to estimate x ? Fortunately, in many economics problems, the unknown quantity *is* the expectation $E[\xi]$. In this case, the problem is already defined in the sense of step 1.

Example 9 (Risk-neutral asset pricing). A risk-neutral agent values any asset as the discounted expected payoff.

$$p_0 = e^{-rt} E_t[\xi_t] \tag{7.1}$$

With p_0 the asset price today (at time 0), r the risk-free interest rate, t the time until the payoff is paid and ξ_t the payoff at time t . Notice that ξ_t is a random variable (we assume we cannot predict asset returns). This problem is already formulated as in step (1) above.



Figure 7.2: The casino of Monte Carlo, which gave the Monte Carlo method its name.

7.3.2 Law of Large Numbers – LLN (Kolmogorov)

Given a sequence of n independent, identically distributed (i.i.d.) random variables ξ_i with

$$E[\xi_i] = \mu \quad (7.2)$$

Define the running sum S_n and the running average X_n as

$$\begin{aligned} S_n &= \sum_{i=1}^n \xi_i \\ X_n &= \frac{1}{n} S_n = \frac{1}{n} \sum_{i=1}^n \xi_i \end{aligned} \quad (7.3)$$

Then

$$X_n \xrightarrow{\text{a.s.}} \mu \quad (7.4)$$

The law of large numbers is *the* basis for simulation-based methods like the Monte Carlo method. It ensures that our simulation actually converges to the desired value. If we took an infinite number of draws, we would get the precise value of the expectation.

7.3.3 Central limit theorem – CLT (Lindberg-Levy)

Given a sequence of n i.i.d. random variables ξ_i with $E[\xi_i] = \mu, Var[\xi_i] = \sigma^2$. We first define a new random variable Y_n

$$Y_n = \frac{S_n - n\mu}{\sigma\sqrt{n}}.$$

The CLT states that Y_n converges in distribution to a standard normal distribution:

$$Y_n \xrightarrow{\text{distr}} N(0, 1)$$

An alternative formulation is:

$$X_n \xrightarrow{\text{distr}} N\left(\mu, \frac{\sigma^2}{n}\right) \quad (7.5)$$

The CLT proves our observation that increasing n lowers the variance of the estimate. Furthermore, we can quantify this effect: if we increase n by a factor of 100, then the variance of the simulation will decrease by a factor of 100; the standard deviation will decrease by a factor of 10.

The advantage of simulation based methods such as the Monte Carlo method is that by increasing n , the variance can be made arbitrarily small. The drawback is the slow convergence speed and the computational intensity: to achieve 10 times the precision, we

7 Monte Carlo simulation

need 100 times the number of events.

The variance $\frac{\sigma^2}{n}$ is a measure for the precision of the Monte Carlo method. In theory, it could be made arbitrarily small.

Note: The σ is the variance of the whole process. It is not always possible to calculate σ . This σ is completely different from a possible σ -statistic in the process.

7.3.4 Monte Carlo Standard Error (MCSE)

The CLT gives the precision of the Monte Carlo method in stating that $X_n \xrightarrow{\text{distr}} N(\mu, \frac{\sigma^2}{n})$. By increasing n , the variance decreases. But what is the value of σ^2 ?

Example 8 continued. In some cases, it is possible to calculate σ^2 . In our example, ξ has a discrete distribution with equally probable values of 1 to 6. Then $V(\xi) = E[(\xi - E[\xi])^2] = \frac{1}{6}[(-2.5)^2 + (-1.5)^2 + (-0.5)^2 + 0.5^2 + 1.5^2 + 2.5^2] \approx 2.92$. For $n = 1000$, the variance of a Monte Carlo estimate of $E[\xi]$ should be $2.92/1000 = 0.0029$. We verify this by running the program `manydice.m` with $n = 1000$ and calculating `var(result)`. In a test run, we obtain a value of 0.0028.

In the more interesting cases, in which already $E[\xi]$ is not accessible, we cannot calculate $V[\xi] = \sigma^2$, so we have to estimate it. It would be desirable to find an estimator $\hat{\sigma}$ that does not require to repeat the whole simulation several times. This estimator is the Monte Carlo Standard Error:

$$\hat{\sigma}_n = \sqrt{\frac{1}{n} \sum_{i=1}^n (\xi_i - \hat{\mu}_n)^2} \quad (7.6)$$

with $\hat{\mu}_n = \frac{1}{n} \sum \xi_i$ being the Monte Carlo estimate. The index in $\hat{\sigma}_n$ indicates that the standard error has been estimated using a sample size of n .

The MCSE is called “standard error”, because it is standardized to a sample size of one – if we ran a Monte Carlo with $n = 1$, then the standard deviation of the result would be $\hat{\sigma}$. For larger samples, we apply (7.5). We get the MCSE $\hat{\sigma}_n$ with only one additional calculation after the Monte Carlo simulation. In most cases, producing the ξ_i is computationally quite intensive, so $\hat{\sigma}$ comes almost “for free”.

Application. The MCSE can be used to determine the sample size that is required for some target precision. A common approach is to perform a preliminary simulation with $n = 1000$ and to calculate $\hat{\sigma}_n$. If $\hat{\sigma}_n$ is already lower than the desired precision, we are done. If not, we can calculate the number of draws N^* to achieve a target precision σ_t as

$$N^* = \frac{\hat{\sigma}_n^2}{\sigma_t^2} \quad (7.7)$$

7.3.5 Extensions

General functions. The Monte Carlo method is not limited to estimating $E[\xi]$, it can in fact estimate $E[g(\xi)]$ for almost any well-behaved $g(\cdot)$. Furthermore, we can calculate any statistic of ξ , not only the expected value.

Variance reduction. A great deal of research has been devoted to making the Monte Carlo method faster and thus more efficient. All these techniques aim at reducing σ^2 , which in turn makes less steps necessary to obtain a certain precision. The most popular techniques are **importance sampling**, in which the sample is drawn only out of the relevant part of the distribution of ξ and **antithetic sampling**, in which pairwise negatively correlated events reduce the overall variance. See P. Jaeckl: Monte Carlo Methods in Finance, Wiley 2002, chapter 10.

Simulation and estimation. A large class of economic problems (e.g. asset pricing in finance) aim at explaining observed data with a probabilistic model. In such a model, the random variable ξ depends on a few parameters a, b, \dots . In this setting, we want the result of the Monte Carlo simulation to be as close as possible to the data. The procedure is the following: (1) choose some parameters a, b, \dots (2) calculate the result of the MC. (3) compare it to the data. (4) if the result is close enough to the data, we are done. If not, we try some other parameters and continue with (2). In such a setting, we usually perform hundreds or thousands of MC simulation to estimate the optimal parameter set.

7.3.6 Conclusion

- We perform a MC, if the random process is known, but there are no closed-form solutions for the expected value or other statistics.
- Sometimes we do a MC even if the closed-form solution is known, because it is simpler or faster.
- Step 1: Calculate N instances of the random variable ξ_i
- Step 2: Take $mean[\xi_i]$ as an estimator for the expected value.
- Advantages of the MC method:
 - Simple algorithm.
 - Can be easily parallelized (for multiprocessor environments / clusters).
 - Usable for all random processes.

For other problems, one may be able to construct a probabilistic model.

7.4 PC-Lab: Simulation and estimation

First steps

<code>ceil(6*rand(1,3))</code>	Throw three dices.
<code>ceil(6*rand(10,3))</code>	Throw three dices ten times.
<code>sum(ceil(6*rand(10,3)),2)</code>	The sums of points of three dices; ten times.
<code>a=sum(ceil(6*rand(1000,3)),2);</code>	The sums of 1000 throws of three dices.
<code>hist(a,15)</code>	Histogram.

OLS properties

Linear model $Y = \beta X + \varepsilon$. We run an OLS with simulated data:

1. We choose the dimension of the model. $K=3$, $N=200$.
2. We define $\beta = (1, 2, 5, 0.5)'$. (β has dimension $K + 1$).
3. We produce random X -values and a random error term.
4. We calculate Y from these.
5. Next, we estimate $\hat{\beta}$ from our results and compare it to the initial β .

<code>K=3; N=200</code>	Parameters.
<code>b=[1 2 5 0.5]'</code>	Chosen ("real") β . Slope parameter is 1.
<code>X=[ones(N,1) rand(N,K)];</code>	Include row of ones in X .
<code>e=randn(N,1);</code>	Produce the error term.
<code>Y=X*b+e;</code>	This is the linear model, Eq. 5.3.
<code>[b_hat, binv]=regress(Y,X)</code>	Now we estimate $\hat{\beta}$.
<code>[b b_hat]</code>	Compare the results.

```

K=3;N=10000;
for obs=1:1000
    X=[ones(N,1) rand(N,K)];
    e=randn(N,1);
    Y=X*b+e;
    [b_hat binv]=regress(Y,X);
    b_results(:,obs)=b_hat;
end
b_results
hist(b_results(1,:))
    
```

Violating OLS assumptions

Recall some of the OLS assumptions from the previous section:

7 Monte Carlo simulation

MLR.2 Zero conditional mean: $E(\varepsilon|X) = 0$

MLR.4 No perfect colinearity: $rk(X) = k + 1$

Let us first violate MLR.2.

```
e=0.5*randn(N,1)+0.5*X(:,2);
```

Produce a correlated error term.

```
Y=X*b+e;
```

Produce a new Y with the new error term.

```
[b_hat, binv]=regress(Y,X);
```

We now perform a new regression.

```
[b_hat b]
```

Compare $\hat{\beta}$ to the true β . The second component – the one which is correlated to the error term – is biased.

```
for obs=1:1000
```

```
    X=[ones(N,1) rand(N,K)];e=0.5*randn(N,1)+0.5*X(:,2);
```

```
    Y=X*b+e;
```

```
    [b_hat binv]=regress(Y,X);
```

```
    b_results(:,obs)=b_hat;
```

```
end
```

```
for k=1:4
```

```
    subplot(2,2,k)
```

```
    hist(b_results(k,:))
```

```
end
```

Now, let us violate MLR.4.

```
X=[ones(N,2) rand(N,K-1)];
```

 MLR.4 is violated.

```
rank(X)
```

 Verify that MLR.4 is violated.

```
X(1:10,:)
```

 Verify what we produced.

```
e=randn(N,1);
```

 Produce an unbiased error term.

```
Y=X*b+e;
```

 Produce a new Y with the “bad” X .

```
[b_hat, binv]=regress(Y,X);
```

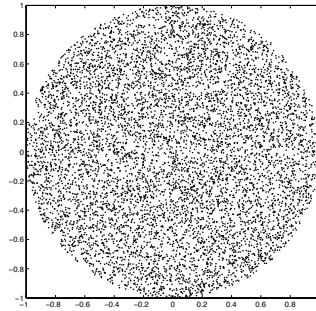
 Enjoy the error message!

Calculating π

We first need a probabilistic model of π . The idea is to draw random points out of the unit square. If $x^2 + y^2 \leq 1$, then they are part of the unit circle, else they are not. π is the fraction of points that are in the circle times 4.

The program `circle.mc.m` produces the following figure:

7 Monte Carlo simulation



7.5 Exercises

Exercise 7.1. Monte Carlo basics.

- (a) Produce a vector $\mathbf{r1}$ with $N=1000$ random numbers that are uniformly distributed between -0.5 and $+0.5$
- (b) Produce a vector $\mathbf{r2}$ with $N=1000$ random numbers that are normally distributed with mean zero and variance 0.5
- (c) Produce a vector $\mathbf{r3}$ that is an element-by-element sum of $\mathbf{r1}$ and $\mathbf{r2}$. Calculate analytically the mean and variance of the underlying distribution of $\mathbf{r3}$. Write your calculation and the results in a comment. *Hint:* $\mathbf{r1}$ and $\mathbf{r2}$ are uncorrelated.
- (d) Calculate the sample mean and sample variance of $\mathbf{r3}$.
- (e) Re-do steps (a), (b) and (d). Check if you get the same result. Write a short explanation in the comment.
- (f) Apply (7.6) to estimate $\hat{\sigma}$. Apply (7.7) to find N^* with $\sigma_t = 0.03$.
- (g) Run 100 Monte Carlo simulations of $\mathbf{r3}$ with $N = N^*$ from question (f). In how many runs is the variance of $\mathbf{r3}$ is within ± 0.03 of the estimated value. Does the result match your expectations?

Exercise 7.2. Find the probability for scoring exactly N points with M throws of a regular (six-sided) dice. (a) Start by producing all possible combinations and count the winning combinations. (b) Solve the problem using a Monte Carlo simulation.

Exercise 7.3. Find the probability for the case that a) exactly two persons b) at least two persons in a room have the same birthday (day+month). Assume that all years have 365 days and that birthdates are evenly distributed among the population. Calculate the above probabilities for every group size from 2 to N . Make one plot with both probabilities as a function of group size. *Hint:* do not think in terms of days and months; it is sufficient to use the fact that a year has 365 days, months are not needed.

8 Spectral theory

References Härdle and Simar (2003) sections 2.2 and 9

8.1 Eigenvalues and eigenvectors

Spectral theory is the theory of eigenvalues and eigenvectors¹. Spectral theory is the main tool to understand the structure of a matrix – and more general of linear operators. The main idea behind it is to split the matrix into simple pieces and to analyze each piece separately.

Definition 8. A scalar value λ is called an *eigenvalue* of the $n \times n$ matrix A if there exists a nonzero vector \mathbf{x} such that

$$A\mathbf{x} = \lambda\mathbf{x} \quad (8.1)$$

Definition 9. A vector \mathbf{x} that satisfies (8.1) is called *eigenvector*.

To find the eigenvalues, one has to solve (8.1) for *any* possible \mathbf{x} :

$$\begin{aligned} A\mathbf{x} &= \lambda\mathbf{x} \\ (A - \lambda Id)\mathbf{x} &= 0 \end{aligned} \quad (8.2)$$

Equation (8.2) is a system of equations which has a nontrivial solution only if the coefficient matrix $(A - \lambda Id)$ is singular. This is equivalent to the statement $\det(A - \lambda Id) = 0$. In the 2×2 case, $\det(A - \lambda Id)$ takes the following form:

$$\begin{aligned} \det \begin{bmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{bmatrix} &= (a_{11} - \lambda)(a_{22} - \lambda) - a_{12}a_{21} \\ &= \lambda^2 - \lambda(a_{11} + a_{22}) + a_{11}a_{22} - a_{12}a_{21} \end{aligned} \quad (8.3)$$

The polynomial in 8.3 is called *characteristic polynomial*. Finding the eigenvalues λ_i is equivalent to finding all roots of the characteristic polynomial. It is easy to see that for a $n \times n$ matrix, the characteristic polynomial is of degree n . This implies that such

¹The famous German mathematician David Hilbert coined the terms “Eigenwert” and “Eigenvector” in 1904. The English translation “proper value” was used for some time, but did not prevail.

8 Spectral theory

a matrix has n eigenvalues (although some could be identical or complex). If a value λ_i occurs more than once – k times – as an eigenvalue, it is said to have a *multiplicity* of k .

Definition 10. The set of all eigenvalues of a matrix A is called its *spectrum*. It is sometimes denoted as $\sigma(A)$.

Example 10. The characteristic polynomial of Id_2 is $(1 - \lambda)(1 - \lambda)$. It has two eigenvalues, both are 1. Its spectrum is $\sigma(Id_2) = \{1, 1\}$.

To find the eigenvector that corresponds to a certain eigenvalue, one has to solve the system of equations $(A - \lambda_i Id)\mathbf{x}_i = 0$. The eigenvector \mathbf{x}_i may have complex elements. Note that if \mathbf{x}_i is an eigenvector, then $\alpha\mathbf{x}_i$ is an eigenvector, as well. The convention is to normalize all eigenvectors to a length of 1.

Eigenvectors for distinct eigenvalues are always linearly independent (Proof: Simon-Blume, sect. 23.9). In the case of multiplicity, there still may exist linearly independent eigenvectors (e.g. for the identity matrix in example 10). See exercise 7.1.

Properties: $Tr A = \lambda_1 + \lambda_2 + \dots + \lambda_n = \sum_{i=1}^n 1\lambda_i$
 $\det A = \lambda_1 \cdot \lambda_2 \cdot \dots \cdot \lambda_n = \prod_{i=1}^n \lambda_i$
 A triangular: $\lambda_i = a_{ii}$ (also valid for diagonal matrices)
 $\lambda_i \neq \lambda_j \rightarrow \mathbf{x}_i \mathbf{x}_j = 0$
(Eigenvectors of different eigenvalues are linearly independent)
 A singular \rightarrow at least one eigenvalue = 0

MATLAB note: Finding eigenvalues and eigenvectors of large matrices is virtually impossible by hand. MATLAB offers the command `eig` for this task. `1=eig(A)` produces a vector `1` with all eigenvalues and `[X,D]=eig(A)` produces a matrix `X` whose columns are the eigenvectors and a diagonal matrix `D` whose elements are the eigenvalues.

8.2 Matrix decompositions

8.2.1 Diagonalization

It would be very practical to decompose a $n \times n$ matrix A in the following way:

$$A = PDP^{-1} \tag{8.4}$$

where D is a diagonal matrix and P is any general matrix. A^2 could then be calculated as $A^2 = (PDP^{-1})^2 = PDP^{-1}PDP^{-1} = PD^2P^{-1}$ and generally we could write

8 Spectral theory

$$A^n = PD^nP^{-1} \quad (8.5)$$

This is practical, because it is very easy to compute D^n , e.g. in the 3×3 case:

$$D^n = \begin{bmatrix} (d_{11})^n & 0 & 0 \\ 0 & (d_{22})^n & 0 \\ 0 & 0 & (d_{33})^n \end{bmatrix} \quad (8.6)$$

Proposition 1. Any square matrix A with distinct eigenvalues can be diagonalized in the form of (8.4) with P the matrix of eigenvectors $P = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$ and D a diagonal matrix of eigenvalues.

Proof.

$$\begin{aligned} AP &= A[\mathbf{x}_1 \cdots \mathbf{x}_n] \\ &= [A\mathbf{x}_1, \dots, A\mathbf{x}_n] \\ &= [\lambda_1\mathbf{x}_1 \cdots \lambda_n\mathbf{x}_n] \\ &= [\mathbf{x}_1 \cdots \mathbf{x}_n] \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_k \end{bmatrix} \\ &= PD \end{aligned} \quad (8.7)$$

We denoted the matrix of the eigenvalues as D , because it is – as desired – a diagonal matrix. Next we multiply (8.7) with P^{-1} from the right to get expression (8.4).

It remains to be shown that P^{-1} exists. This is the case, if P is regular, which means that all eigenvectors are linearly independent. This is exactly the case if all eigenvalues are distinct. See also exercise 7.1.

Symmetric matrices. Symmetric matrices have only real eigenvalues. Furthermore, even if some eigenvalues may not be distinct, P is always invertible. (Proof: Simon-Blume, sect 23.7). Thus, all symmetric matrices can be diagonalized. One can show, that P is orthogonal for symmetric matrices:

$$\begin{aligned} A &= A' \\ PDP^{-1} &= (PDP^{-1})' \\ PDP^{-1} &= (P^{-1})' D' P' \end{aligned} \quad (8.8)$$

Notice that $D = D'$. Comparing the coefficients, we find that P is orthogonal: $P' = P^{-1}$. Thus we can write (8.4) also as

$$A = PDP^{-1} = PDP' \quad (8.9)$$

8 Spectral theory

Notation. Some books write $D = PAP^{-1}$ instead of $A = PDP^{-1}$, which is equivalent. Matrix diagonalization is sometimes referred to as PAP-decomposition.

8.2.2 Square root of a matrix

For scalars, the square root is well defined: $\sqrt{a}\sqrt{a} = a$ and there are normally two solutions (one positive, one negative). We can use (8.5) to calculate $A^{1/2}$. Note that this approach will give complex solutions if there are negative eigenvalues.

8.2.3 Choleski decomposition

In many cases, we are not so much interested in $A^{1/2}$, but in a decomposition $A = RR'$, with R being an (upper) triangular matrix. For symmetric, positive-definite matrices (e.g. all covariance matrices), R always exists. The calculation is rather tedious, but MATLAB can do it for us with `chol(A)`.

Application. The Choleski decomposition can be used to produce correlated random numbers (most computer systems can only produce uncorrelated ones). Suppose $\Sigma = RR'$. Produce a random vector $\mathbf{y} = R\mathbf{z}$ with $\mathbf{z} \sim N(0, Id)$. Then $\mathbf{y} \sim N(0, \Sigma)$.

Proof: $V(\mathbf{y}) = E[(\mathbf{y} - \mu_y)(\mathbf{y} - \mu_y)'] = E[\mathbf{y}\mathbf{y}'] = E[R\mathbf{z}\mathbf{z}'R'] = RIdR' = RR' = \Sigma$

8.2.4 Singular value decomposition (SVD)

The Choleski decomposition is only available for symmetric, square and positive-definite matrices. Although all covariance matrices fall in this category, it would be desirable to have a matrix decomposition that allows more general input. This is the singular value decomposition (SVD).

Proposition 2. One can decompose any matrix $A_{n,m}$ with rank $p \leq \min(m, n)$ as

$$A_{n \times m} = U_{n \times p} \Sigma_{p \times p} V_{p \times m}' \quad (8.10)$$

with U, V are orthogonal matrices and contain the first p eigenvectors of AA' and $A'A$ respectively and Σ a diagonal matrix that contains the square roots of the first p eigenvalues of AA' .

Proof (sketch). Matrix diagonalization is only possible for square matrices. However, the matrices AA' and $A'A$ are square and furthermore symmetric such that the decomposition (8.9) is possible. We verify that this is the case:

$$\begin{aligned} AA' &= U\Sigma V'V\Sigma'U' = U\Sigma^2U' \\ A'A &= V\Sigma U'U\Sigma'V' = V\Sigma^2V' \end{aligned} \quad (8.11)$$

By comparing this result to (8.9), we find that U and V are indeed the eigenvectors of AA' and $A'A$. Furthermore, $\Sigma^2 = D$ or $\Sigma = \sqrt{D}$ from (8.9). This is indeed the diagonal matrix of the square roots of the eigenvalues. To calculate \sqrt{D} , see (8.6).

Equality of SVD and matrix diagonalization for symmetric matrices. To follow.

8.3 Condition number

In chapter 2, we have stated that the determinant of a singular matrix is zero. With real data, we will hardly ever run across perfect linear dependence, so one could argue that a determinant “close to” zero is a good criterion for singularity. Now, consider the matrix $10^{-3} \cdot Id_4$. Its determinant is 10^{-12} , which is definitely close to zero, however this matrix is perfectly invertible.

To find a better criterion, we notice a singular matrix has at least one eigenvalue which is zero. Furthermore, matrices that are close to being singular, have at least one eigenvalue which is very small compared to the other eigenvalues. This can be used to define a measure for linear dependence that is more precise than the determinant.

Definition 11. The condition number of a matrix A is defined as

$$k(A) = \frac{\sqrt{\lambda_{max}}}{\sqrt{\lambda_{min}}}$$

Where $\lambda_{max,min}$ are the largest/smallest eigenvalue of A . A rule of thumb states that a condition number of more than 15 indicates a possible problem and a condition number of more than 30 indicates a serious problem with colinearity. Note that our matrix $10^{-3} \cdot Id_4$ has a condition number of 1 – the smallest possible one.

8.4 PC-Lab

8.4.1 Correlated random numbers.

Imagine you want to simulate a portfolio of two stocks $i = 1, 2$. Each stock has a normally distributed random payoff with mean μ_i and variance σ_i^2 , but it is further known that the returns are correlated with a correlation coefficient ρ .

8 Spectral theory

MATLAB, like many other computing environments, can only produce uncorrelated random variables. We have to put in the correlations ourselves. In a first try, we take the independent random variables z_1 and z_2 and try to construct a set of correlated random variables x_1 and x_2 :

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} z_1 \\ \rho z_1 + (1 - \rho)z_2 \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ \rho & (1 - \rho) \end{bmatrix} \mathbf{z}$$

MATLAB code For simplicity, $\mu_i = 0$ and $\sigma_i^2 = 1$ <code>z1=randn(1000,1);</code> <code>z2=randn(1000,1);</code> <code>rho=0.5;</code> <code>x1=z1;</code> <code>x2=rho*z1+(1-rho)*z2;</code> <code>corr(x1,x2)</code>	$Z_1 \sim N(0, 1)$ $Z_2 \dots$ same Choose some number for ρ . Big surprise.
---	--

What went wrong? Let us calculate the variance-covariance matrix of \mathbf{x} using (4.1):

$$V(\mathbf{x}) = E(\mathbf{x} - \mu_x)(\mathbf{x} - \mu_x)'$$

Note that $\mu_z = 0$ and therefore $\mu_x = 0$, as well. We define $Q = \begin{bmatrix} 1 & 0 \\ \rho & (1 - \rho) \end{bmatrix}$. Then $\mathbf{x} = Q\mathbf{z}$ and we get:

$$V(\mathbf{x}) = E[\mathbf{x}\mathbf{x}'] = E[Q\mathbf{z}\mathbf{z}'Q'] = QE[\mathbf{z}\mathbf{z}']Q' = QQ'$$

Lesson: If we multiply a random vector with a matrix Q , its covariance matrix will be QQ' . Conversely, to produce correlated random numbers with a covariance matrix Σ , we have to multiply the vector of independent random numbers with the Choleski decomposition of Σ .

Verification: suppose $\Sigma = RR'$. Produce a new random vector $\mathbf{y} = R\mathbf{z}$ with $\mathbf{z} \sim N(0, Id)$. Then:

$$V(\mathbf{y}) = E[(\mathbf{y} - \mu_y)(\mathbf{y} - \mu_y)'] = E[\mathbf{y}\mathbf{y}'] = E[R\mathbf{z}\mathbf{z}'R'] = RIdR' = RR' = \Sigma$$

MATLAB code (still, $\mu_i = 0$ and $\sigma_i^2 = 1$) <code>z = randn(1000,2);</code> <code>Sigma=[1 0.5; 0.5 1];</code> <code>R = chol(Sigma);</code> <code>x = z*R;</code> <code>corr(x)</code>	Produce z_1 and z_2 in one step. Desired variance-covariance matrix. Perform a Choleski decomposition. Produce correctly correlated random numbers. Now we get what we expected.
---	--

8 Spectral theory

```
MATLAB code for three variables (can be extended to any number of variables)
cor=[1 0.2 0.8; 0.2 1 -0.2; 0.8 -0.2 1];
A=chol(cor);
X0 = randn(1000,3);
X = X0*A;
corr(X)
```

The result does not perfectly match the desired covariance matrix, because the random numbers produced by MATLAB are slightly correlated. We can improve the result by taking out the initial correlations first. To do so, we multiply the data with the choleski decomposition of the inverse of the correlation matrix.

```
MATLAB code taking out initial correlations improves the result
cor=[1 0.2 0.8; 0.2 1 -0.2; 0.8 -0.2 1];
A=chol(cor);
X0 = randn(1000,3);
B=chol(inv(corr(X0)));
X = X0*A*B;
corr(X0)
corr(X0*B)
disp('Without correction:');
disp(corr(X0*A));
disp('With correction:');
disp(corr(X));
```

8.4.2 SVD decomposition

```
MATLAB code
X= gallery(3);           Get some sample data.
[USV] = svd(X,'econ');   Perform the singular value decomposition.
```

A step-by-step implementation can be found in `svd_manual.m`

8.5 Exercises

Exercise 8.1. Calculate using MATLAB the eigenvalues and eigenvectors of the following matrices. Verify equation 8.4. Comment on the result.

$$(a) \mathbf{A} = \text{magic}(5) \qquad (b) \mathbf{B} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Exercise 8.2. Download `nash.jpg` and `svd_foto.m` from the StudyNet and run it. Change the value of `max` in line 22 to different values between 10 and 300.

Exercise 8.3. Download `svd_manual.m`, read the code and try to understand it.

9 Worked problem: PCA applied to the term structure of interest rates

References: Härdle and Simar (2003), chapter 9.

9.1 Motivation

The yield curve is a plot of interest rates as a function of their respective maturities. As such, it describes the *term structure* of interest rates in the economy. The complexity of this object lies in its multidimensional nature:

- **Cross-sectional dimension.** At any given date, we observe a large number of yields with different maturities.
- **Time-series dimension.** The yield curve moves over time, and the relations between its different segments change dynamically.

The two dimensions of the yield curve are illustrated in Figure 9.1, which plots a sample of US Treasury yields with maturities from 3 months to 10 years over the period 1952:01–2005:06.

Since the yield curve is the natural starting point for the pricing of fixed-income securities, it has been an object of extensive research in financial economics. Much of this research has been motivated by the observation that yields with different maturities move in lockstep and has thus focussed on defining the *common forces* behind this co-movements. The argument for studying the common yield factors is intuitive: Due to their high (even though not perfect) correlations, yields provide redundant information of one another. Therefore, there should be a (small) set of common independent economic factors that account for most of the variation in the cross-section of yields over time. The principal component analysis (PCA) provides a convenient tool for extracting these factors from the data.

9 Worked problem: PCA and the term structure of interest rates

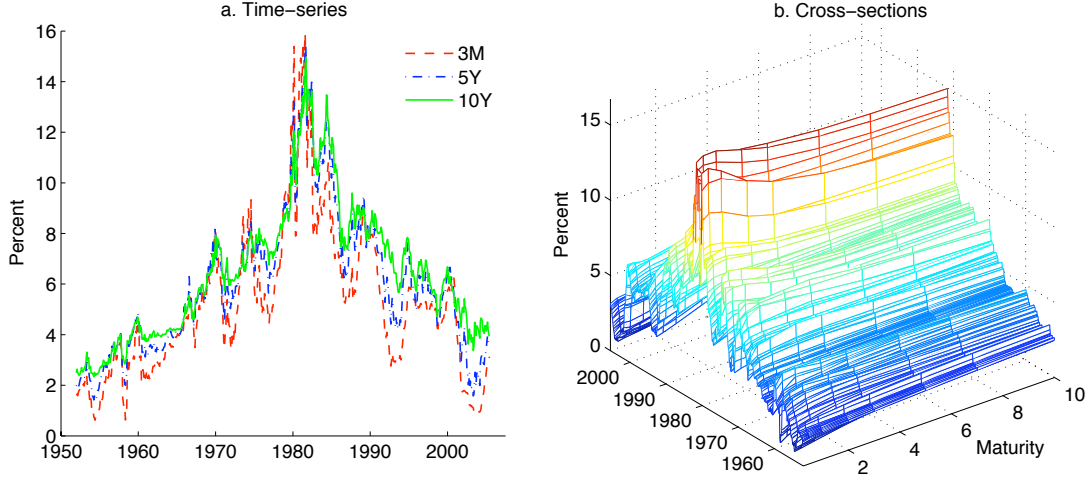


Figure 9.1: US Treasury yields in the time-series and cross-section (1952:01–2005:06).

9.2 The method

Technically, the PCA consists in transforming a set of *correlated* yields into a set of *orthogonal* variables—the **principal components**—that reproduce the original information present in the correlation structure. The technique amounts to diagonalizing the variance-covariance matrix of yields by means of the *singular value decomposition*.

Suppose you have T months of data on K yields with different maturities collected in matrix Y^{raw} of dimension $T \times K$. For convenience, we transform the raw yield data as follows:

$$Y_\tau = \frac{1}{\sqrt{T-1}}(Y_\tau^{raw} - \bar{Y}_\tau^{raw}), \text{ for } \tau = 1, \dots, K, \quad (9.1)$$

where \bar{Y}_τ^{raw} is the average τ -maturity yield, and Y_τ denotes a τ -th column of a transformed matrix of yields Y . (In this way, we get rid of the division by the number of observations and the cumbersome mean which otherwise needs to be subtracted from yields when computing the second moments.)

Let us express the variance-covariance matrix of yields as:

$$V = Y'Y, \quad (9.2)$$

Since V is by definition a symmetric, positive-definite matrix, we can apply the matrix diagonalization (8.9):

$$V = U\Lambda U', \quad (9.3)$$

Without loss of generality, we can assume that the eigenvalues are in decreasing order:

9 Worked problem: PCA and the term structure of interest rates

$\lambda_1 > \lambda_2 > \dots > \lambda_K$. Note: In practise, if we change the order of the eigenvalues, we must not forget to change the order of the eigenvectors accordingly.

From equation (9.3) it follows:

$$\begin{aligned}\Lambda &= U' V U \\ &= U' Y' Y U \\ &= pc' pc = \text{Var}(pc)\end{aligned}\tag{9.4}$$

where $pc = YU$ is the $T \times K$ matrix whose columns are the K *principal components*, often called the yield curve factors. Notice that the principal components are by construction independent, and Λ can be interpreted as their variance-covariance matrix. One way to see that the pc s are independent is the fact that Λ is a diagonal matrix. Thus, the variance of the i -th principal component is just equal to the i -th eigenvalue λ_i :

$$\text{Var}(pc_i) = \lambda_i,\tag{9.5}$$

and the total variation of yields in Y is equal to the total variation of the principal components:

$$\text{tr}(V) = \text{tr}(\Lambda) = \sum_{i=1}^K \lambda_i,\tag{9.6}$$

The fraction of the total variance of yields that is accounted for by the j -th factor, pc_j , is:

$$\frac{\lambda_j}{\text{tr}(\Lambda)}.\tag{9.7}$$

Clearly, the principal components associated with the largest eigenvalues have the largest explanatory power – this was the reason why we ordered them in descending order in the first place. To use only $k \leq K$ principal components, we can define the $K \times k$ matrix \tilde{U} which collects the first k columns of U :

$$\tilde{U}_{ij} = U_{ij} \text{ for } j \leq k\tag{9.8}$$

and compute k principal components of yields as

$$\tilde{pc} = Y\tilde{U}.\tag{9.9}$$

Notice that principal components are linear combinations of K yields. The eigenvectors – columns of U – correspond to the coefficients (weights, or loadings) of these linear combinations. We will show below that in the case of yields the shape of the loadings lends itself to an intuitive interpretation in terms of the level, slope and curvature of the yield curve.

9.2.1 Criteria and tests for the PCA

- mean eigenvalue criterion
- Hotellings T^2

9.3 Empirical results

To illustrate the information conveyed by the PCA, we use the monthly US treasury yields with maturities 3 and 6 months, 1, 2, 3, 5, 7 and 10 years over the time period 1952:01–2005:06 in order to discern the common factors driving them. A typical pattern documented in the term structure literature is that as few as three principal components can explain over 95% of the variance of yields. The contribution of the remaining factors is negligible. Our sample confirms this pattern. Table 9.1 reports the percentage of variation in yield levels explained by the first three principal components. For comparison, we also perform the PCA on yield changes.

Table 9.1: Percentage of yield variation explained by the j -th principal component

j	1	2	3	4	5
% var. expl. (yield levels)	97.29	2.51	0.18	0.02	0.00
% var. expl. (yield changes)	86.28	11.14	1.73	0.60	0.15

Note: The table displays the percentages of variation in levels and changes of yields explained by the j -th principal component. We use monthly US yields for eight maturities: 3, 6 months, 1, 2, 3, 5, 7, 10 years, and the sample period 1952:01–2005:06.

Figure 9.2 plots the evolution of the first three principal components of yield levels over the sample period (panel *a*), and the corresponding eigenvectors (loadings) as a function of maturity (panel *b*).

The first three principal components have an intuitive interpretation:

Level factor. Note that the weights on the first principal component pc_1 form approximately a straight line. This suggests that yields of different maturities react in a similar way to the changes in the first principal component. As such, one can interpret pc_1 as a parallel shift in the yield curve, or as the *level* factor.

Slope factor. The shape of loadings on the second principal component pc_2 suggest its impact on the *slope* of the yield curve. The shock to pc_2 increases short-term interest rates by much more than the long-term interest rates: the yield curve becomes less steep.

Curvature factor. Finally, the loadings on pc_3 tend to have a parabolic shape: they start high at the short maturity, decline for the intermediate range of maturities and rise

9 Worked problem: PCA and the term structure of interest rates

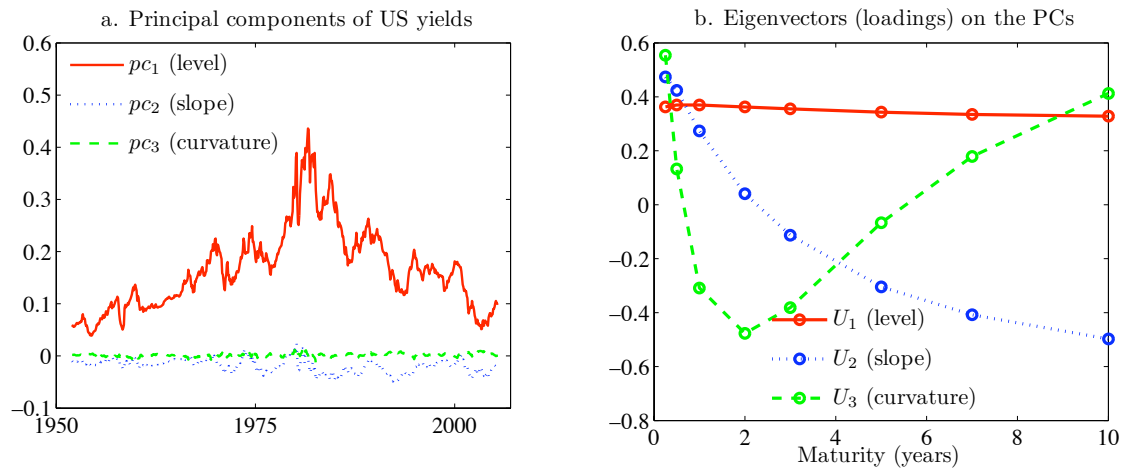


Figure 9.2: The first three principal components of US yields in sample period 1952:01–2005:05 (panel *a*) and the corresponding loadings (panel *b*).

again for longer maturities. Hence, the third principal component is reminiscent of a *curvature* factor.

9.4 PC Lab: PCA in MATLAB

```
load Ylddata;      Load data.
Y = Ylddata./100;  Convert to yields
V = cov(Y);        Compute the covariance matrix of yields.
[U, l] = eig(V);    Compute the eigenvalues and eigenvectors of the covariance matrix.

U(:,1)'+U(:,2)     You can check that the eigenvectors in U are orthogonal.
[X, idx] = sort(diag(l), 'descend');
l = diag(X);        Eigenvalues on the diagonal of Λ are not necessarily sorted. Sort them in a descending order ...

U = U(:,idx);       ...and reshuffle the corresponding eigenvectors accordingly.
pc = Y*U;           Obtain the principal components.
pc'*pc              This is the identity matrix, no surprise (orthogonal and normalized).

V_expl = 100*diag(l)./trace(l);
                    Compute the percentage of yield variance explained by each principal component, like in Tab. 9.1. (How is this linked to the R2?)

plot(pc(:,1:3))     Plot the first three principal components, like in Fig. 9.2a
plot(U(:,1:3))      Plot the first three eigenvectors = factor loadings, like in Fig. 9.2.b
```


9 Worked problem: PCA and the term structure of interest rates

A more detailed version of this PC-lab can be found in `PCA_yieldcurve.m` on the Stundynet.

MATLAB has also several built-in functions related to the PCA, e.g., `svd` (singular value decomposition), `princomp`, `pcacov`.

Bibliography

- Abadir, Karim M., and Jan R. Magnus (2005) *Matrix Algebra (Econometric Exercises)* (Cambridge University Press)
- Brandimarte, Paolo (2006) *Numerical Methods in Finance and Economics* (Wiley and Sons)
- Gaughhofer, M., and H. Müller (2003) *Mathematik für Ökonomen, Band 2* (Verlag Wilhelm Subir, St. Gallen)
- Gentle, James E. (2007) *Matrix Algebra* (Springer)
- Greene, W. H. (2002) *Econometric Analysis* (Prentice Hall; 5th edition)
- Gut, Allan (2005) *Probability: A Graduate Course* (Springer)
- Hanselman, D., and B. Littlefield (2005) *Mastering Matlab 7* (Pearson Education International)
- Härdle, W., and L. Simar (2003) *Applied Multivariate Statistics* (Springer)
- Judd, Kenneth L. (1998) *Numerical Methods in Economics* (MIT Press)
- Kerrighan, Brian W., and P. J. Plauger (1974) *The elements of programming style* (McGraw-Hill)
- Kuniciky, David C. (2004) *Matlab Programming* (Pearson Prentice Hall)
- LeSage, James P. (2006) *Econometrics Toolbox*. Online, www.spatial-econometrics.com
- Mathworks, The (2007) *Getting started with MatLab*. (Online at http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf)
- Metropolis, M., and S. Ulam (1949) 'The monte carlo method.' *Journal of the American Statistical Association* 49, 335–341
- Moler, Cleve (2004) *Numerical Computing with Matlab* (SIAM (also available at <http://www.mathworks.com/moler/>))
- Pedersen, M. S., and K. B. Petersen (2007) *The Matrix Cookbook* (<http://matrixcookbook.com/>)
- Shao, Jun (2003) *Mathematical Statistics* (Springer)
- Simon, C. P., and L. Blume (1994) *Mathematics for economists* (Norton)
- Verbeek, M. (2004) *Modern Econometrics* (Wiley)
- Wooldridge, J. (2005) *Introductory Econometrics: A Modern Approach* (South-Western College Pub; 3 edition)

Note. This course does not follow a particular book. The references at the beginning of each chapter and summed up here may or may not be useful to the student. None of the books in this list has the same focus as this lecture. Therefore, only parts of each book will be relevant. Moreover, every book uses its own notation, which may confuse the student. Before buying a book, it is strongly advised to assess it in the library.