



2-FUN

*Full-chain and **UN**certainty Approaches for Assessing Health Risks in
FUture ENvironmental Scenarios*

**FP6 Project-2005-Global-4
Integrated Project - Contract n°: 036976**

– DEFINITION AND TRANSFER OF SELECTED METHODS FOR DATA TREATMENT VIA WEB BASED ON OPEN-SOURCE PLATFORM (R- LANGUAGE) –

Due date of delivery: *31/07/2009*

Actual submission date: *23/07/2009*

Start date of the project: *01/02/2007*

Duration: *48 Months*

Lead contractor organisation name for this deliverable: *IPH*

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)	
Dissemination Level	
PP	Restricted to other programme participants (including the Commission Services)

PROPRIETARY RIGHTS STATEMENT

This document contains information, which is proprietary to the 2-FUN Consortium. Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to any third party, in whole or in parts, except with prior written consent of the 2-FUN consortium.



Document Information

Document Name Definition and transfer of selected methods for data treatment via web based on Open-Source platform (R-language)
ID D1.8.doc
Revision Version 2
Revision Date 23/07/2009
Author P. KOVANIC/IPH and L. PAVLISKA/ IPH

Approvals

	Name	Company	Date	Visa
Author	P. KOVANIC	IPH	23/07/2009	P. Kovanic
Co-Author	L. PAVLISKA	IPH	23/07/2009	L. Pavliska
WP Leader	A. MARCOMINI	UNIVE	30/07/2009	po E. Giubilato
Coordinator	F. BOIS	INERIS	10/09/2009	F. Bois

Documents history

Revision	Date	Modification	Author
Version 0	18/01/2008	Template made available to the WP leaders	F. BOIS
Version 1	23/07/2009	First version	P. KOVANIC
Version 2	10/09/2009	Final version	F. BOIS



Contents

INTRODUCTION.....	3
1. DEFINITION OF THE “GNOSTIC” METHODS	4
1.1. Structure of gnostic functions.....	5
1.2. Grouping of functions of gnostic package.....	8
2 TWO SUITABLE ENVIRONMENTS	12
2.1 S-PLUS	12
2.2. R-project	12
2.3 Differences between the systems	13
3 THE SYNTAX OF THE S-LANGUAGE	14
3.1 Classes of expressions	14
3.2 Basic description of the language.....	14
3.3 Formal Description of the S-Language	15
3.4 Data in S.....	17
4 INSTALLATION AND RUNNING THE R ENVIRONMENT.....	18
4.1 Downloads and installation	18
4.2 Working in the R-environment	19
4.3 Working in RExcel.....	20
4.4 Writing and debugging of user’s functions	22
4.5 A suitable editor for R.....	22
5 R LANGUAGE AND ENVIRONMENT.....	23
5.1 Setting the environment	23
5.2 Source files.....	24
5.3 System packages	24
5.4 R Console command line	26
6 GNOSTIC FUNCTIONS IN R.....	27
6.1 Transfer of gnostic functions.....	27
6.2 GUI for Gnostic functions.....	27
6.3 Example of marginal analysis	27
7 CONCLUSIONS.....	30

INTRODUCTION

The gnostic methodology is based on the gnostic theory of individual uncertain data and small data items, which represents a new (not yet generally accepted) paradigm. The report¹ exposed main theoretical aspects of this paradigm, demonstrated its suitability to a broad applications in health risk analysis as well as in many other

¹ Kovanic P., Ocelka T., Extended report on review and tests of methods for filling data gaps, Deliverable D1.9, 2-FUN project, 40pp., (2009).



fields of practice and showed its superiority over the methods of robust statistics in solving several difficult tasks. However, the main goal of making the gnostic methodology a subject of the 2-FUN project is to help to the project participants in using these methods in applications to health risk analysis. This can be realized only by means of a software running within a computing environment making available a rich assortment of mathematical and statistical procedures. The gnostic software has been developing for years in the environment S-PLUS² by using its S-language to write and test the new algorithms. However, this environment is a commercial product. Although it is broadly known and applied throughout the world, its general availability for all potential users cannot be assumed. Fortunately, the danger of commercialization of the software with its negative impacts on the development and using these accomplishments of people's spirit were soon recognized by progressive people and ways to face it were found in introducing the idea and practice of *copyleft* instead of the *copyright*. A broad movement was founded by the GNU-project thanks to Richard Stallman (1983) resulting in free operation systems. This made way to free application programs, too. Ross Ihaka and Robert Gentleman developed and made available the first version of the R-project environment in 1997 to provide functions similar to those of the S-PLUS. Their initiative found a broad response. Many top specialists took part in development of this system. This resulted in forming the international R Development Core Team and in launching updated version of the software many times. The latest version is 2.9.1/2009-06-26. It is already available at <http://www.r-project.org/>.

The capabilities of R are extended through user-submitted *packages*, which allow specialized statistical techniques, graphical devices, as well as [programming interfaces](#) and import/export capabilities to many external data formats. These packages are developed in R, [LaTeX](#), [Java](#), and often [C](#) and [Fortran](#). A core set of packages are included with the installation of R, with a total of 1862 (as of June 2009) available at the Comprehensive R Archive Network (CRAN). Notable packages by subject area are listed along with comments on the official R Task View pages.

The productivity of the R Development Team along with the large number of already available packages provide a measure of the mass support of the scientific and technical community to the idea of free availability of recent methods. This also motivates the efforts aiming to enable the advanced methods of mathematical gnostics to enter the consciousness of specialists of data treatment. This makes the transfer of gnostic methods from the S-PLUS into the environment of the R-project necessary.

1. DEFINITION OF THE “GNOSTIC” METHODS

The Greek word “gnostic” used for the non-statistical theory of individual uncertain data and small data samples³ means “knowledge”. It was applied (in no connection with the historical Gnostic movement inside the Christian society) to denote a deterministic counterpart to the “agnostic” statistical notion of “randomness”. The gnostic

² S-PLUS is a registered trademark of the Insightful Corp., Seattle, Wa, USA.

³ Kovanic P., Gnostical Theory of Individual Data, Problems of Control and Information Theory 13 (1984), 4, 259-274

Kovanic P., Gnostical Theory of Small Samples of Real Data, Problems of Control and Information Theory 13 (1984), 5, 303-319

Kovanic P., On Relations between Information and Physics, Problems of Control and Information Theory 13 (1984), 6, 383-399



theory of uncertainty has a realistic character based on Laws of Nature reflected especially in physics and geometry.

The notion of gnostics includes:

- The gnostic theory of uncertain data,
- the gnostic theory of small data samples,
- methods and algorithms using the results of gnostic theory,
- experience with applications of gnostic methods.

1.1. Structure of gnostic functions

Recent structure of gnostic algorithms written in S-language for the S-PLUS and in R-language for the R-project consists of functions described in Tab.1.

Analysis dimension	Function's name	Function's level	Called by function	Direct use	Input data	Output data
MD	AnCorMat	1.		X	"LM"	"NM"
1D	Angle	1.		X	"NM"	"NV"
MD	AvailsubM	M		X	"NM"	„NM“
1D	Certifydf	1		X	"NDF" "HDF"	"LDFM"
1D	checkddf	Ext			auto	auto
1D	ClasIntls	3.		X	"NDF" "HDF"	"LM"
1D	Complete	Int	GNDF		"NM"	"NM"
1D	clean.sample	Ext		X	"NM"	"LM"
1D	Complete1	Int	GNDF		"NM"	"NM"
1D	Complete2	Ext		X	"DF"	"NM"
1D	compress	Ext			auto	auto
1D	convert.az	Ext			auto	auto
1D	convert.fininf	Ext			auto	auto
1D	convert.inffin	Ext			auto	auto
1D	convert.mz	Ext			auto	auto
1D	convert.za	Ext			auto	auto
MD	Cortest	Ext			"NM", 2 char	"LNM"
Analysis dimension	Function's name	Function's level	Called by function	Direct use	Input data	Output data
1D	convert.zm	Ext			auto	auto
1D	criterion	Int	optimdf		auto	auto
1D	critLP	Int	Glocparm		auto	auto
1D	critSB	Int	Gsbounds		auto	auto
MD	crs.comp	Int	optimdf		auto	auto
1D	d2Egdf.k	Int	Gsbounds		auto	auto
1D	d3Egdf.k	Int	Gsbounds		auto	auto
1D	dEgdf	Ext			auto	auto
1D	dEldf	Ext			auto	auto
1D	DistribLDF	3.		X	"LDF"	"NM"



MD	DMtoList	M		X	„NM“	„LM“
1D	dQgdf	Ext			auto	auto
1D	dQldf	Ext			auto	auto
1D	Egdf	Ext			auto	auto
1D	Eldf	Ext			auto	auto
MD	ExComRs	M		X	„LM“	„LM“
MD	FindComRs	M		X	„LM“	„NM“
MD	Gcor	3.		X	“NM”	“LM”
MD	GcorP	2.		X	“LDF”	“LM”
MD	Gcor2	2.		X	“DF”, “DF”	“LM”
1D	Gdatawt	1.		X	„NDF“ HDF	„NV“
MD	Gdfbatch	1.		X	„NM“	„LDF“
1D	Gepilog	Ext			Auto	auto
1D	Gintervals	2.		X	“NDF“ “HDF“	„NM“
1D	Gkernel	2.		X	“NDF“ “HDF“	„NM“
1D	Glocparm	2.		X	“NDF“ “HDF“	„NV“
MD	Glogreg	1.		X	„NM“	„LAO“
1D	GNDF	1.		X	“NV” “NM”	“DF”
1D	Gprobab	2.		X	“NDF“ “HDF“	„NM“
MD	Gprobreg	1.		X	„LDF“	„LAO“
1D	Gquantile	2.		X	“NDF“ “HDF“	„NV“
MD	GraphMDres	2.		X	„MDM“	Graph
1D	GraphQPE	2.		X	“NDF“ “HDF“	Graph
MD	GWLSX	1.		X	„NM“	„LAO“
MD	GwlsW	Ext			auto	auto
MD	GwlsWX	Ext			auto	auto
1D	Gsbounds	2.		X	“NDF” „HDF“	„LAO“
1D	Gscale.dir	Ext		X	„NV“ “NM“	„LAO“
1D	Gscale.loc	Ext			auto	auto
1D	homogenizeE	2.		X	„NDF“	„HDF“
MD	homogenizeC	1.		X	“NM”& (“NV” 0)	„LM“
MD	homogenizeM	1.		X	„LM“	„LM“
1D	homogenizeW	2.		X	“NDF“ “HDF“	„DF“
Analysis dimension	Function's name	Function's level	Called by function	Direct use	Input data	Output data
1D	IdSimple	Ext	Recogn	X	a data object	Object's code
1D	IdList	Ext	Recogn	X	a data object	Object's code
MD	Info	1.			„LDF“ “LLDF“	“NM”
1D	is.hgndf	Ext	Recogn	X	a data object	TRUE FALSE
1D	is.ngndf	Ext	Recogn	X	a data object	TRUE FALSE
1D	InfoQ	Int	optimdf		Auto	auto
1D	init	Int	GNDF		Auto	auto
1D	locextr	Int	checkddf		Auto	auto
MD	MainClust	1.		X	„NM“	„LM“
1D	MarkClust	Int	GNDF		Auto	auto



1D	Marganal	1.		X	„NM“ „NV“	„LM“
1D	MarkClustF	Ext			“NDF” “HDF“	„NM“
MD	MDpred	1.		X	„LM“	„LAO“
1D	Merge	1.		X	„NM“	„NM“
MD	ModelMDCS	1.		X	„LM“	„LM“
MD	ModelMDser	1.		X	„NM“	„NM“
1D	optimdf	Int	GNDf		auto	auto
MD	p.inversion	Ext			auto	auto
1D	Parmsdf	2.		X	“NDF” „HDF“	„LAO“
1D	ParmsLDF	3.		X	“LDF”	„NM“
1D	plotdf	2.		X	“NDF” “HDF“	Graph & „NV“
MD	plotgr	1.			“NM”	Graph, „LAO“
MD	plotrlog	1.			“NM”	Graph, „LAO“
1D	PredRisk	1.		X	„NM“	„NM“
1D	prep.sample	M		X	„NM“ „NV“	„NM“
MD	PrepMDser	1.		X	„LM“	„NM“
1D	Qgdf	Ext		X	auto	auto
1D	Qldf	Ext		X	auto	auto
1D	ReadOut	3.		X	„LLDFM“	„CM“
1D	Recogn	1.		X	a data object	Object's code
1D	ResumeLDF	3.		X	„LDF“	„LLHDF“
1D	ReviewQs	3.		X	„LDF“ “LHDF“	„NM“
1D	RevParms	2.		X	“NDF” “HDF“	“NDF” “HDF“
1D	SelectX	M		X	„FR“	„NM“
MD	SignifCor	Ext			„LNM“	„NM“
MD	Sim	1.			„NM“	„LM“
MD	SimQuant	1.			„NM“	„LM“
1D	SortClust	Int	homogenize		auto	auto
MD	TakeOut	M		X	„NM“	„NM“
2D	TestHyp	1.		X	2x„NDF“ “HDF“	„LAO“
1D	Tollnt	1.		X	„NDF“ “HDF“	„LM“
1D	TreatDM	1.		X	„NM“	„LDFM“
1D	VarS	Ext			auto	auto
1D	wedf	Int			auto	auto
MD	psi.bisquare	Ext			auto	auto
Analysis dimension	Function's name	Function's level	Called by function	Direct use	Input data	Output data
MD	psi.gnostic	Ext			auto	auto
MD	psi.hampel	Ext			auto	auto
MD	psi.huber	Ext			auto	auto

Comments: Tab.1 represents a review of functions used in gnostic analysis of uncertain data. Column “Analysis dimension” refers to the dimension of the data treated. Functions for one-dimensional analysis are denoted “1D”, while “2D” and “MD” denote the two-and multi-dimensional analysis. The second column belongs to the names of functions. Levels of calling the functions are denoted in the third column in the following way:

Level “M”: Auxiliary functions for manipulations with data.

Level “Ext”: External gnostic functions called by functions of a higher level.



Level “Int”: Internal functions working only as sub-procedures within functions of higher levels.

Level “1.”: Functions applicable directly to data.

Level “2.”: Functions making use of already available gnostic distribution function.

Level “3.”: Functions operating on a list of available gnostic distribution functions.

Application of a function of the level “2.” can be illustrated by the function Gsbounds:

$$\text{Data} \Rightarrow \text{GNDF} \Rightarrow \text{Gsbounds}$$

An example of the function of the level „3”:

Data A	⇒	GNDF A	⇒	homogenize A	}	Creation of the list of homogeneous distribution functions A ... L; then ⇒ ResumeLDF
Data B	⇒	GNDF B	⇒	homogenize B		
....		
Data L	⇒	GNDF L	⇒	homogenize L		

The fourth column defines the function, which calls the internal function.

Sign X in the fifth column denotes functions, that can be applied directly by the user.

Other symbols defining types of input and output data of the functions:

“FR” ... a data frame,

“NV”... a numeric vector,

“NM”... a numeric matrix,

“DF” ... an arbitrary gnostic distribution function,

“NDF”... a non-homogeneous gnostic distribution function,

“HDF”... a homogeneous gnostic distribution function,

“LAO”... a list of atomic objects,

“LNM” ... a list of numeric matrices,

“LDF” ... a list of gnostic distribution functions,

“LDFM” ... a list of gnostic distribution functions and matrices,

“LLDF” ,, a list of lists of gnostic distribution functions.

“auto” ... the data format is automatically determined by the calling function and/or of the function receiving the output data.

1.2. Grouping of functions of gnostic package

Gnostic analysis is performed by the functions of the gnostic package, which call many standard functions of the S-PLUS and/or R-environment. Functions contained in gnostic package form following groups:

1.2.1 Auxiliary non-standard functions

- ◆ *p.inversion*: Performs pseudo-inversion of a matrix.
- ◆ Optimization functions:



- *nminb*: Nonlinear local minimization of a multivariate function subject to box constraints (in S-PLUS),
- *L-BFGS-B*: a local minimization algorithm allowing box constraints (in R-environment),
- *crs.comp*: a global optimization procedure based on controlled random search with competition of heuristics,
- ◆ *checkddf*: to check the form of the probability density,
- ◆ *peaks*: to find local maxima and minima of the density function.

1.2.2 Functions preparing or recognizing data structures

- *SelectX*: to select some or all rows and columns of an array of the class “data.frame” and to present them as a numeric matrix,
- *Angle*: to compute Euclidean angle between two vectors,
- *AvailsuM*: to extract usable (not containing non-available (NA) or infinite items) rows and columns from a matrix M and to apply them in the form of a numeric matrix,
- *clean.sample*: to identify items of a data sample and to present them separately as a list with non-available, positive, negative and zero-valued components,
- *DMtolist*: to create a list of matrices from a large matrix, the rows of which have the same row-names, but different identification code in a column,
- *FindComRs*: to find identically named rows existing in all matrices of a list created by the function *DMtolist*,
- *ExComRs*: to extract identically named rows from a list of matrices created by the function *Mtolist*, found by the function *FindComRs*, and to present results as a list of matrices,
- *IdList*: to describe a structure of a list,
- *IdSimple*: to identify a simple object by attaching to it one of 15 possible codes,
- *is.ngndf*: to check, if a structure is a non-homogeneous gnostic distribution function,
- *is.hgndfi*: to check, if a structure is a homogeneous gnostic distribution function,
- *PrepMDser*: to extract a multi-dimensional sequence or time series of data matrices from a list of matrices prepared by the function *ExComRS*,
- *prep.sample*: returns an one-column matrix suitable as the argument Sample of the function GNDF,
- *Recogn*: to recognize an object to be used as a suitable argument of a gnostic function,
- *SimQuant*: to quantify similarity of two vectors,
- *Sim*: to evaluate both internal and external similarity/consistency of two measuring methods,
- *TakeOut*: to extract rows defined by row-names from a matrix.

1.2.3 Internal functions in the main function GNDF

- *init*: to initiate the optimization procedure *optimdf*,
- *optimdf*: to optimize parameters to be estimated (*Sopt*, *LB*, *B*),
- *Complete1*: to preliminarily estimate the values of the censored data,
- *criterion*: to evaluate the current data fit error of the d.f.,
- *crs.comp*: to perform global optimization of the distribution’s goodness-of-fit,
- *InfoQ*: to evaluate the information measure of the quality of the distribution function,
- *checkddf*: to check the form of the probability density with respect to local extremes,
- *locextr*: to estimate the quantiles of the probability density’s extremes,
- *MarkClust*: to assign each data item the mark 0, -1, or 1 according to its location in the main cluster or in the lower or upper cluster of outliers as manifested by the density distribution over the infinite data support.

1.2.4 External Functions Used by the main function GNDF

- ◆ *Complete2* to estimate the values of the censored data within the *Rounds = 2* of *GNDF*,
- ◆ *compress* to make the data matrix shorter by summarizing the weights of repeated rows,
- ◆ *convert.az* to transform additive data into the normalized finite interval of multiplicative data,
- ◆ *convert.finiinf* to transform the data from normalized finite interval of multiplicative data onto the infinite interval of positive numbers,



- ◆ *convert.inffin* to transform the data from the infinite interval of positive numbers into the normalized finite interval of multiplicative data,
- ◆ *convert.mz* to transform multiplicative data into the normalized finite interval of multiplicative data,
- ◆ *convert.za* to transform the data from normalized finite interval of multiplicative data onto their original additive interval,
- ◆ *convert.zm* to transform the data from normalized finite interval of multiplicative data onto their original multiplicative interval,
- ◆ *dEgdf* to estimate the density of the estimating global distribution function,
- ◆ *dEldf* to estimate the density of the estimating local distribution function,
- ◆ *dQgdf* to estimate the density of the quantifying global distribution function,
- ◆ *dQldf* to estimate the density of the quantifying local distribution function,
- ◆ *Egdf* to estimate the estimating global distribution function,
- ◆ *Eldf* to estimate the estimating local distribution function,
- ◆ *Gdatawt* to estimate the a posteriori data weights,
- ◆ *Gepilo g* to transform variables into their original scales,
- ◆ *Glocparm* to estimate the location parameter (density's mode),
- ◆ *Gscale.dir* to directly estimate the global scale parameter of a data sample,
- ◆ *Gscale.loc* to estimate a local scale parameter of a data sample in a given point,
- ◆ *Gquantile* to estimate the quantile to a given probability,
- ◆ *plotdf* to draw four graphs of probability and density distributions over the infinite and finite data supports,
- ◆ *Qgdf* to estimate the quantifying global distribution function,
- ◆ *Qldf* to estimate the quantifying local distribution function,
- ◆ *VarS* to estimate local scale parameters of a heteroscedastic data sample,
- ◆ *Wedf* to estimate the weighted empirical distribution function.

1.2.5 Functions using single distributions *GNDf*s

- ◆ *Certifydf(defDX)* certifies the distribution function by analysis of its errors of two kinds: data errors and errors of data probability,
- ◆ *ClasIntls(defDX)* estimates intervals' bounds (in the sense of the gnostic interval analysis) of a homogeneous data sample, to which the *defDX* belongs and classifies all data of the sample with respect to its membership in individual intervals,
- ◆ *Gintervals(defDX)* performs the interval analysis of the data sample,
- ◆ *Gkernel(defDX, which = I)* to estimate the relative contribution of the data item called *which* to the probability and density distributions.
- ◆ *Glocparm(defDX)* evaluates and prints on the screen global location parameter, the location of the density's maximum,
- ◆ *Gprobab(Q, defDX)* returns the values of the distribution function and its density in the point (quantile) *Q* (or in all quantiles given as a vector *Q*),
- ◆ *Gquantile(P, defDX)* returns the estimate of quantiles for a vector *P* of probability values (which has an arbitrary length). This function is also usable for robust estimation of the median (*Gquantile(0.5, defEG.DX)*), or of the quartile (*Gquantile(0.25, defEG.DX)*), etc.
- ◆ *Gsbounds(defDX)* estimates the bounds of the membership interval, ie conditions for the proper data of a (homogeneous) sample,
- ◆ *homogenizeE(defDX)* to extract the homogeneous data sub-sample from a sample of a general type by elimination of data not belonging to the Sample's main cluster,
- ◆ *homogenizeW(defDX)* to homogenize a data sample by means of repeatedly iterated data weights,
- ◆ *Gdatawt* estimates the a posteriori data weights of a data sample, to which a distribution function was computed,
- ◆ *Marganal* to split a multi-modal data sample into three clusters/subsamples (the lower, main and upper) by means of the local distribution function,
- ◆ *MarkClustF* marks data items of a sample with respect to their location in sub-clusters determined by the density function over the infinite data support and by the form of the probability distribution,
- ◆ *Parmsdf(defDX)* to summarize parameters of the *defDX* in the form of a numerical matrix,



- ◆ *PredRisk(...)* to successively estimate probabilities of reaching given quantiles and quantiles to given probabilities by operating on a multi-dimensional time series,
- ◆ *plotdf(defDX)* returns four graphs of probability and density functions over the infinite and finite data support,
- ◆ *RevParms(defDX)* to adapt the scale parameter S , the a priori weights and the bounds of the data support so as to make the distribution function homogeneous, while using the original data,
- ◆ *SortClust(defDX)* to separate the sub-samples of the *defDX\$Data[, 2]*: the main (homogeneous) cluster, the lower cluster and the upper cluster.
- ◆ *TolInt* to estimate bounds of the confidence (or 'toleration') interval of the statistical test for given significance (error alpha) completed by the bounds of data support.
- ◆ *TestHyp* to perform an extended two-samples test of hypotheses (including estimation of the errors of the first kind (alpha) and second kind (beta) and other parameters supporting the decision.

1.2.6 Functions related to a list of GNDFs

- ◆ *Gdfbatch*: to compute a list of distribution functions of selected columns of a numeric matrix.
- ◆ *DistribLDF(LdefDX)*: to calculate probability and density values in a given set of quantiles by using all the distribution functions of the list *LdefDX*.
- ◆ *ParmsLDF*: to prepare matrix of characteristics of all distribution functions of a list: functions' parameters, quality of the fit of data and percentiles to a given set of probabilities.
- ◆ *Gcor*: to robustly estimate correlation coefficients and matrices to columns of a matrix and to test the significance.
- ◆ *GcorP(LdefDX)*: to estimate the gnostic correlation matrix of *length(LdefDX)* data vectors *LdefDX[[i]]\$Data[, 2]* ($i = 1, \dots, \text{length}(LdefDX)$).
- ◆ *Info* to estimate the amount of information of each of data samples, to which a list of gnostic distribution functions has been computed.
- ◆ *ReadOut(LdefDX)*: to read (and to store as rows of the output matrix) five elements ('Rowname', 'Colname', 'Value', 'Censoring', '-1/1(L/U)') of each row in *LdefDX[[_]][[2]]* and *LdefDX[[_]][[3]]*, i.e. of all rows denoted as lower or upper outliers by the operation *homogenizeE* applied to several non-homogeneous data samples.
- ◆ *ReviewQs(LdefDX, Prob = 10)*: to compute a matrix containing quantiles for Prob uniformly distributed probabilities using the GNDFs listed by *LdefLDF*.
- ◆ *Gprobreg(LdefDX, ...)* to create an MD-regression models in probabilities.

1.2.7 Functions preparing and building the multi-dimensional models

- ◆ *Cortest* to estimate a correlation matrix using the Classical (Pierson) or robust (Kendall or Sperman) method and test the significance of correlations,
- ◆ *SignifCor* an auxiliary function enabling to list the items of a correlation matrix prepared by the functions *Gcor*, *GcorP* or *Cortest*, which satisfy a given level of the significance.
- ◆ *AnCorMat* to order and review the most important features of a correlation matrix made by the functions *Gcor* or *GcorP*.
- ◆ *p.inversion* to perform the Singular Value Decomposition of matrices,
- ◆ *GwlsW* to provide solution of the Weighted Least Squares system of equations,
- ◆ *GwlsWX* to provide solution of the Weighted Least Squares system of equations improved by the Minimum Penalty Estimate,
- ◆ *GWLSX* to compute and evaluate a robust regression model by the Iterated Weighted Least Squares Method,
- ◆ *plotgr* to draw a graph of a regression model,
- ◆ *Gprobreg* to estimate a robust regression model of interactions of data probabilities,
- ◆ *Glogreg* to estimate a robust regression model of interactions between logarithmized variables,
- ◆ *psi.gnostic* to evaluate gnostic weights of residues,
- ◆ *GraphMDres* to draw a graph of residues of a regression model,
- ◆ *plotgrlog* to draw a graph of a regression model in logarithms.



1.2.8 Functions applying the multi-dimensional regression models

- ◆ *MainClust* to extract the main cluster (a subset of rows) from the MD-equation system subjected to operations of robust regression,
- ◆ *MDpred* to predict a dependent value of a multi-dimensional series of regression data,
- ◆ *homogenizeM* to apply function *MDpred* repeatedly to a whole data series and to gradually eliminate the objects with the smallest probability of true predictions,
- ◆ *homogenizeC* to perform the robust multi-dimensional cluster analysis,
- ◆ *ModelMDCS* to quantify a multi-dimensional process described by a list of matrices of observations by a sequence of models and vectors of estimated data, their weights, modeling errors and data scores.
- ◆ *ModelMDser* to monitor dynamics of an object's MD-series by sequential (moving window) robust identification of object's models.

2 TWO SUITABLE ENVIRONMENTS

2.1 S-PLUS

The Gnostic functions have been originally developing by using the S-language in the commercial environment **S-PLUS®** 6.2 Insightful Corporation. S-PLUS is a tool for exploratory data analysis, data visualization and exploration, statistical modeling, and programming the data treatment. The great disadvantage of this environment is its dependence on the contractor and his trading conditions. All of his products are licensed. After updating a software version, purchasing of the new version is assumed. But two factors are to be taken into account:

- 1) A large number of universities and other institutions already possess the S-PLUS licenses, are used to it and store within it large databases.
- 2) Gnostic methods were created by using S_PLUS environment and applicable versions of gnostic functions suitable at least for the S-PLUS version 6.2 are available.

It would be therefore pity to not offer the chance of using the new methods to the users of S-PLUS. However, authors of this report do not have an access to the new generation of the S-PLUS (8.0), so the possible corrections of the present state of the gnostic software have to leave to the users.

S-PLUS is a system combining the usage of standard routines professionally written in Fortran, C+, Java and other languages with the functions of an interpreter of the S-language written by the user. The main advantage of the interpreted language is that it allows the incremental development of functions. A function is written, run, another function is written and then a third function that call the previous two is written. The incremental development is what makes S-PLUS a prototyping tool. The compiled standard functions called by the interpreted user's function execute the operations quickly, making thus the combined software acceptably efficient.

2.2. R-project

It became obvious from the development of the programming field, that the commercial way cannot ensure such popularity to a new ideas as the open-source technology. The R-project appeared to be the most attractive for gnostics:

- 1) Its language (R) is to a large degree similar to the S-language.
- 2) R-project also combines compiled standard functions with the interpreted user's language.
- 3) Versions of the R-environment are available for many platforms (e.g. UNIX, LINUX, FreeBSD, Windows, MacOS)



- 4) R-project is supported by a broad scientific and technologic community, thanks to which it develops and improves in a fast and reliable manner.
- 5) Thanks to its GNU character, R-project is becoming a preferred tool for data processing in many world universities and laboratories.

A brief characteristics of the R-project is in order here: R is a language and environment for statistical computing and graphics. Its programming language (R) is similar to the S-language and environment, which was originally developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a slightly different implementation of S. There are some important differences, but many codes written for S run unaltered under R. R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity. One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains a full control. R is available as Free Software under the terms of the [Free Software Foundation's GNU General Public License](#) in source code form.

2.3 Differences between the systems

Although similar, the R-environment is far from being identical with the S-PLUS:

- 1) Some mathematical and statistical functions differ. This relates to many functions of robust statistics, but also – among others – to such frequently used functions as that for optimization.
- 2) The assignment operator “_” acceptable in S is not used in R, it must be substituted by the “<-“.
- 3) A part of graphical commands is entirely different.
- 4) GUI (The Graphical User’s Interface) is different:
 - ◆ An object browser provided with recognition and filtering the objects available already in S-PLUS 6.2 is missing in R-environment.
 - ◆ A graphical facility to import and export data from and to different sources available in S-PLUS is missing in R-environment.
 - ◆ An interactive graphical tool available in S-PLUS is missing in R.
 - ◆ An automated generator of windows for completing the parameters and running functions available in S-PLUS is missing in R.
 - ◆ A link for a perfect communication and data transfer from R to Excel and backwards is available in R but not in S-PLUS.

All this leads to necessity to solve many technical problems to ensure a user comfort for running the gnostic functions in the R-project environment⁴.

The GUI (Graphical User Interface) in S-PLUS is a component of its own technology using the libraries of basic package. S-PLUS is strongly oriented on graphical MS Windows environment. It can therefore have a lot of features supported by that environment. The R-project is broadly oriented to different platforms including such independent environments as Unix/Linux. This has an impact on GUI, because the GUI in Unix systems is build on completely different fundamentals. This is, why in R is its own graphical user interface to design graphical features and GUI missing. Instead, external packages to extend the graphical technique and properties are to be used. There already exist many packages in various program languages (e.g.Tcl/TK, GTK and many others), which provide the interface structure for such services. However, it is necessary to built up user’s own functions for GUI of specific programs. Creation of its own GUI is thus also necessary for the gnostic SW transferred to the R. The approach to this problem is exposed below.

⁴ Authors cannot warrant neither completeness nor accuracy of this description of the differences. Searching through all the 1862 (as of June 2009) available packages of R-project is not an easy task. Moreover, the project is developing fast, therefore new instruments can emerge every day.



3 THE SYNTAX OF THE S-LANGUAGE

This section briefly describes the structure of the S-language including the precedence of operators. When the system's Commands window is used, the keyboard standard input is directed immediately to the parser, which converts the typed characters into expressions evaluated by the system's evaluator.

3.1 Classes of expressions

There are seven classes of complete S-expressions:

Class	Expression
Literals	Literals are the simplest objects known to S. Any individual number, character, string or name is a literal.
Calls	Perhaps the most common S expression, a call is any actual use of a function or operator.
Assignments	Assignments associate names and values.
Conditionals	Conditionals allow branching depending upon the logical value of a user-defined condition.
Loops	Loops allow iterative calculations.
Flow-of-control statements	Direct evaluation out of a loop or the current loop iteration.
Grouping statements	Grouping allows evaluation to be controlled by overriding the default precedence of operations or by modifying the expected end-of-expression signal.

3.2 Basic description of the language

To improve readability, the S-language elements are typed in blue.

Assigning a value to a name is achieved with an "arrow" "<", ">" (or "_" in S-PLUS) or by using the assign function. Arithmetic binary operators are placed between the (names of) two vectors. Parentheses specify the order of evaluation, see the precedence table below. The form of a function call is: the name of the function followed by an opening parenthesis, then the list of arguments (if any) followed by a closing parenthesis. Items in the argument list are separated by commas; arguments listed out of order must be in the `name=value` format. In general, argument names need not be typed in full; only enough to uniquely identify the argument is required. The exception is, when an argument follows the 'dot-dot-dot' construction '...' in the definition - in this case the argument must be given in the `name=value` form with the name specified exactly.

To subscript from vectors and arrays use `vname[expr]`⁵. When extracting portions of a list, use `lname[expr]`, `lname[[expr]]` or `lname$name`. Use of the single square brackets returns a sublist of `lname`, while the double square brackets and the \$ operator return one component of `lname`. The advantage of the double bracket form is, that components can be extracted by their order in the list as well as by name. Not all of a component name need to be specified—the requirement is, that the sufficient letters are written to uniquely identify the component.

Curly braces ('{' and '}') surrounding a number of expressions cause them to be treated as a single expression. This is useful in writing the functions and in using the qualifiers `if`, `for`, `while`, and `repeat`. A typed expression, which does not fit on a line, may be continued by ending the line at a point, where it is obviously incomplete,

⁵ The string `vname` is the name assigned to a vector, `lname` is a name given to a list and `expr` is an S-expression.



with a trailing comma, operator, or with more left parentheses than right parentheses (implying, that additional right parentheses will follow).

The default prompt character is “>”; when continuation is expected it changes to “+.” Each line can have at most 128 characters, so if there are complicated expression, they will need to be continued on one or more lines. On the other hand, two or more expressions can be placed on a single line if they are separated by a semi-colon.

The flow of control in **for**, **while**, and **repeat** loops can be controlled with **next** and **break**. If **next** is encountered, the next iteration is immediately begun. If a **break** is encountered the loop is exited immediately.

A function can be exited with **return(expr)**.

String literals are contained between matching apostrophes or matching double quotes. Characters inside can be escaped by preceding them by the back-slash character: **\n** (newline), **\t** (tab), **** (back-slash), **\r** (carriage return), **\b** (backspace). In addition, a back-slash followed by 1 to 3 octal digits represents the character with the corresponding octal representation in ASCII. A back-slash preceding other characters is ignored. This follows C language conventions.

Any sequence of characters between matching “%” characters, not including a new line, is recognized as an infix operator. An expression, whose first character is “!” is executed as a DOS command. In such expressions double backslashes must be used, where a single backslash is intended. Use curly braces to avoid having “!” as the first character of an expression, when it is to mean **not**.

A function is defined by the word **function** followed by matching parentheses that contain the names of the arguments separated by commas. Default values can be given in the **name=default.value** form. Use **...** to pass an arbitrary number of arguments.

3.3 Formal Description of the S-Language

The following infix operators are recognized by the parser. They are listed in decreasing precedence. In the event of ties, evaluation is from left to right.

Precedence

\$
 [] [[]]
 {}^{ }
 -
 :
 %anything%
 * /
 + -
 !
 <> <= >= == !=
 & | && ||
 <<- <- ->

Precedence

HIGH

component selection
 subscripts, elements
 exponentiation
 unary minus
 sequence operator
 special operator
 multiply, divide
 add, subtract
 not
 comparison
 and, or (logical formulas)
 assignment

LOW

Expressions in S are typed by the user, parsed, and evaluated. The following rules define the expressions considered legal by the parser, and the mode of the corresponding object.

Literals

numeric
 character
 name
 comment

MODE

number
 string
 name
 comment



Function definition

function (formals) expr function

Calls

expr infix expr call
expr %anything% expr
unary expr
expr (arglist)
expr [arglist]
expr [[arglist]]
expr fname

Assignments

expr <- expr <-
expr { } <- expr
expr -> expr
expr <<- expr <<-

Conditionals

if (expr) expr if
if (expr) expr else expr

Iterations

for (name in expr) expr for
repeat expr repeat
while (expr) expr while

Flow

break break
next next
return (expr) return
(expr)
{ exprlist }

The additional forms of syntax introduced in the above rules are defined as follows:

exprlist: expr
exprlist ; expr
arglist: arg
arglist, arg
formals: empty
formal
formals, formal
arg: empty
expr
fname =
fname = expr
formal: name
...
name = expr
fname: name
string

Notice, that the above rules are rules of syntax, but there may be additional semantic rules, which then will determine, what expressions can be evaluated. In particular, the left-hand-side of assignment expressions is syntactically an expression, but only certain of them, involving subscripts, attributes, and names, are allowable at execution time.



Numeric literals (numbers) are defined by the following rules:

numeric: **integer**
float
complex: **numeric i**
numeric [\pm] **numeric i**
name: (.|letter)
(.|letter|digit)
integer: digit
exponent: **e** [\pm] integer
float: integer exponent
integer digit_exponent
integer exponent

Names are defined by:

name: (.|letter) (.|letter|digit)

3.4 Data in S

To properly understand functions written in the S-language, one must distinguish between several data structures used by the language. These form a pyramid-like system, which can be presented in an order from the most general to most primitive data structures:

list	MOST GENERAL
data frame	
matrix	
vector	
literal	MOST PRIMITIVE

There are six classes of literals:

- Numbers in S are real or complex and are expressed as decimals or in exponential (scientific) form. They can be generated by such S expressions eg as **pi** or **exp(2.5)** or **3/5**. A number can have the IEEE special value **Inf**. This value may be either assigned to objects or returned from computations (e.g. if division by zero is attempted).
- Strings consist of zero or more characters typed between two apostrophes (' ') or double quotes (" ") as described above.
- Names are unquoted strings not starting with a number and consisting of alpha-numeric characters and periods (.
- Comments are anything typed between a "#" character and the end of a line.
- Functions consist of the word **function**, a parenthesized set of formal arguments (which may be empty), and a complete expression. Function literals, in general, are on the right side of the assignment arrow and appear only during function assignment.
- Symbolic constants (reserved words in S) are:
TRUE T FALSE F NULL NA Inf NaN.

There also are reserved names **if, else, for, while, repeat, next, break, in, function, return.**

A set of related literals can be stored and manipulated in S as a **vector** set out as a collection of literals of the same class listed in a specific order. The subscript identifies the element's position in a vector. A vector can be created by scanning a file (using the function **scan(fname)**) or composed by elements using the function **c(element1,element2,.. . ,elementL)**, where L is the **length** of the vector. Alternatively, to create a vector of a length L commands **vname <- rep(element,L)** (repeat element L-times) or **vname <- 1:L** (the vector's elements will be 1, 2, . . . ,L) can be used.

A **matrix** is a set of vectors of the same type and of the same length. An element of a matrix is defined by two subscripts (indexes); the first specifies the row, the second the column. A matrix can be created by vector-rows using the function **rbind(row1,row2, . . . , rowN)** or by vector-columns (**cbind(col1,col2,.. . ,colM)**), where N and M are the dimensions of the obtained matrix. For any matrix K, using the function **dim(K)** respectively



returns the number of rows and columns (N and M). Both rows and columns of a matrix can be given names by using the `rownames` and `colnames` functions. Alternatively, the function `dimnames(K)` can be used to assign these names in the form of `list(rownames,colnames)` to the matrix K. A $N \times M$ matrix with all elements equal to X can be created by the command `mname <- matrix(X,N,M)`.

A **data frame** is a tabular structure, in which the columns represent variables, that may be of any type and the rows represent cases (sets of values for those variables). As in matrices, elements of data frames are indexed by the row and column number. A typical application for a data frame is to import data from a spreadsheet or a data base. A tabular structure of unknown data types can be read into S; it will be accepted as a data frame.

A **list** is the most flexible of the basic S data types. It shares with data frames the ability to combine data of different types, but it is not subjected to the data frame's restriction of having the same length of all its variables. Lists are important for retrieving the returned values of S functions. Another important application for a list is connected with the function `dimnames(M)`, the value of which is a list composed of two vectors: the first (`dimnames(M)[[1]]`) returns the names of the rows, while the second (`dimnames(M)[[2]]`) provides the names of columns of the matrix M. This output is similar to that obtained by using the `rownames` and `colnames` functions noted above.

All attributes of objects are registered by S without explicitly distinguishing their nature. An object appearing in programs as 'X' may be a literal as well as a vector, a matrix, a data frame or a list; it depends on the way, in which it was introduced. It can be characterized by some of many defining functions or generated as a value of a function. When in doubt, all pertinent information on the object 'X' may be obtained by using S functions such as `mode(X)`, `length(X)`, `class(X)`, `dim(X)`, `dimnames(X)`, `attributes(X)` etc. Such a proliferation of notation can sometimes cause difficulties in reading programs, but on the other hand, it increases the programmer's flexibility in writing functions.

To ensure the required interpretation of an object, functions `as.(classname)` and `as.(modename)` are to be used.

4 INSTALLATION AND RUNNING THE R ENVIRONMENT

4.1 Downloads and installation

Working with the R-environment is started by installation of the necessary programs.

The basic link of R-project is <http://www.r-project.org/index.html> . The main informations about R system documentations, installation of the current version, on system contributors, links to downloads and many others can be found here. Download of binaries, packages and other sources are available on an arbitrary site CRAN (The Comprehensive R Archive Network) with many mirrors for download files. For example, <http://cran.at-r-project.org/> offers the Windows installation file with the complete installer shown in Fig.1.



CRAN

R-2.9.1 for Windows

[Download R 2.9.1 for Windows](#) (36 megabytes)

[Installation and other instructions](#)

New features in this version: [Windows specific](#), [all platforms](#).

Fig. 1 Example of the Windows installer file for the current version of R.



Process of installation is a standard procedure like many others of Windows software. There is only a specific structure of directories (Fig.2). This is a constant structure and presents definite ordering of files. These are running files, document files, all loaded libraries and sometimes example files.

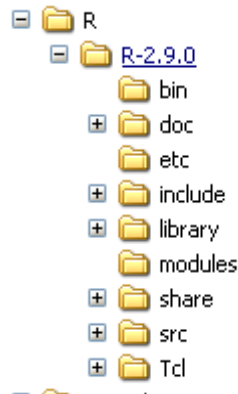


Fig. 2 Directories structure of R-project

This installer prepares everything necessary for working in

C:\Documents and Settings\All Users\START\Programs\R:

- ◆ R 2.9.1
- ◆ R 2.9.1 Help
- ◆ Uninstall 2.9.1

A very useful complement is the Rexcel, which can be downloaded from the <http://rcom.univie.ac.at/>

After the Rexcel is downloaded, following windows open on the start:

- ◆ R Console
- ◆ R Commander
- ◆ Excel Sheet1

[4.2 Working in the R-environment](#)

Like S-PLUS, the R automatically files all functions passed by the parser in the working directory (in Rdata, as the Workspace) along with the commands applied in the current session (in History). The user can arrange all starting operations in the form of the function .First to properly select the working directory and to load all packages ordinarily needed. The alternative is using the R-GUI of the R-console, which offers following choices:

- ◆ **File:** to choose and put a script to the parsing, to open or display scripts, to load or save the workspace and/or the History, to change the working directory, to print and save to a file and to exit.
- ◆ **Edit:** to copy, past, clear the console, to select the editor and to open Rgui Configuration Editor.
- ◆ **Misc:** to stop computations, to buffer output, to complete words/filenames, to list objects and search paths.
- ◆ **Packages:** to set CRAN mirror (one of 69 worldwide available) or repositories, to load, install/update packages.
- ◆ **Help:** to get information from many sources and documentation.

The most direct writing command is done immediately in the row of the R console. To write and debug a sequence of commands and/or functions, the script window is to be opened by `fix(name)` or `name <- edit()`. (The latter way enables to continue in making changes in the script). An advantageous alternative is using the R Commander, because it allows an incremental execution of lines of a script selected for submitting by the mouse.



Scripts are written in R-language submitted to the parser in the text form (as files *.txt or *.R). More than one script can be treated. (The gnostic functions form two packages of 1-dimensional and MD-files. They are read in to parsing as a whole).

Working with R by means of its standard GUI environment (Fig.3) is an alternative, too. It can support the basic application functions to develop user's functions as well as running the programs. R console environment serves to the command line operation and to loading and editing functions and data files.

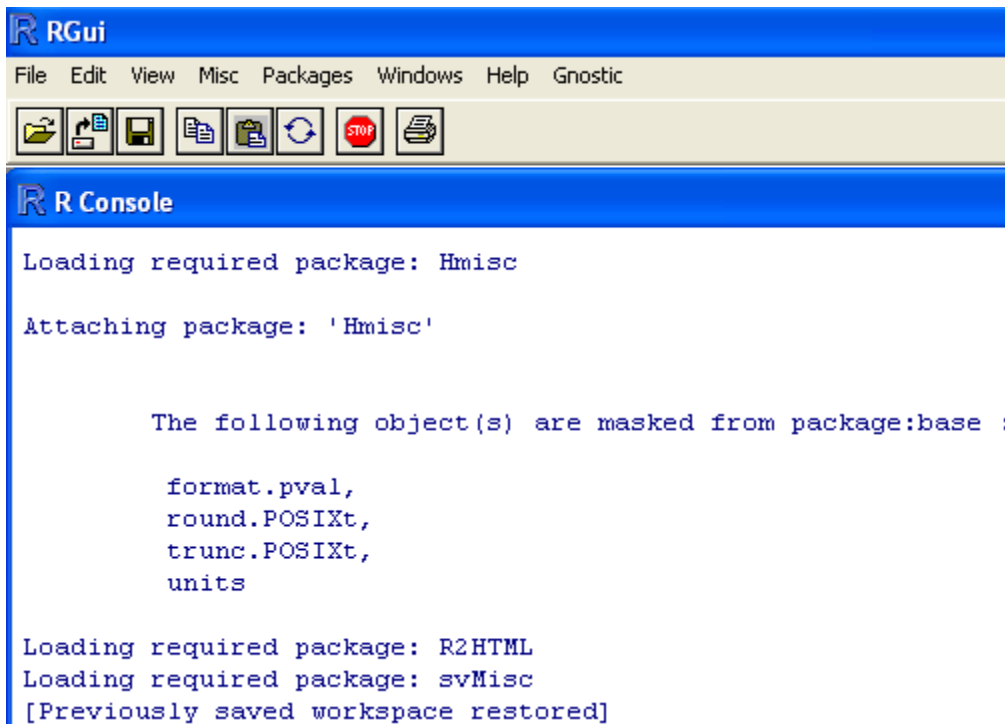


Fig.3. Windows GUI R-project environment

[4.3 Working in RExcel](#)

The most easy way of entering data files into R and taking over the computed results from R is enabled by the RExcel program. An Excel sheet is linked to the working directory of the R system and to the R Commander. This enables command written in Excel to be executed directly in R or via the R Commander, matrices and data frames to R or backwards to be sent and commands, outputs and messages from the R commander to be filed in Excel. The Microsoft Excel can thus serve as a gate of R to communicate with other computing systems and databases.

Using the Excel sheet within the RExcel system is demonstrated in Fig.4.

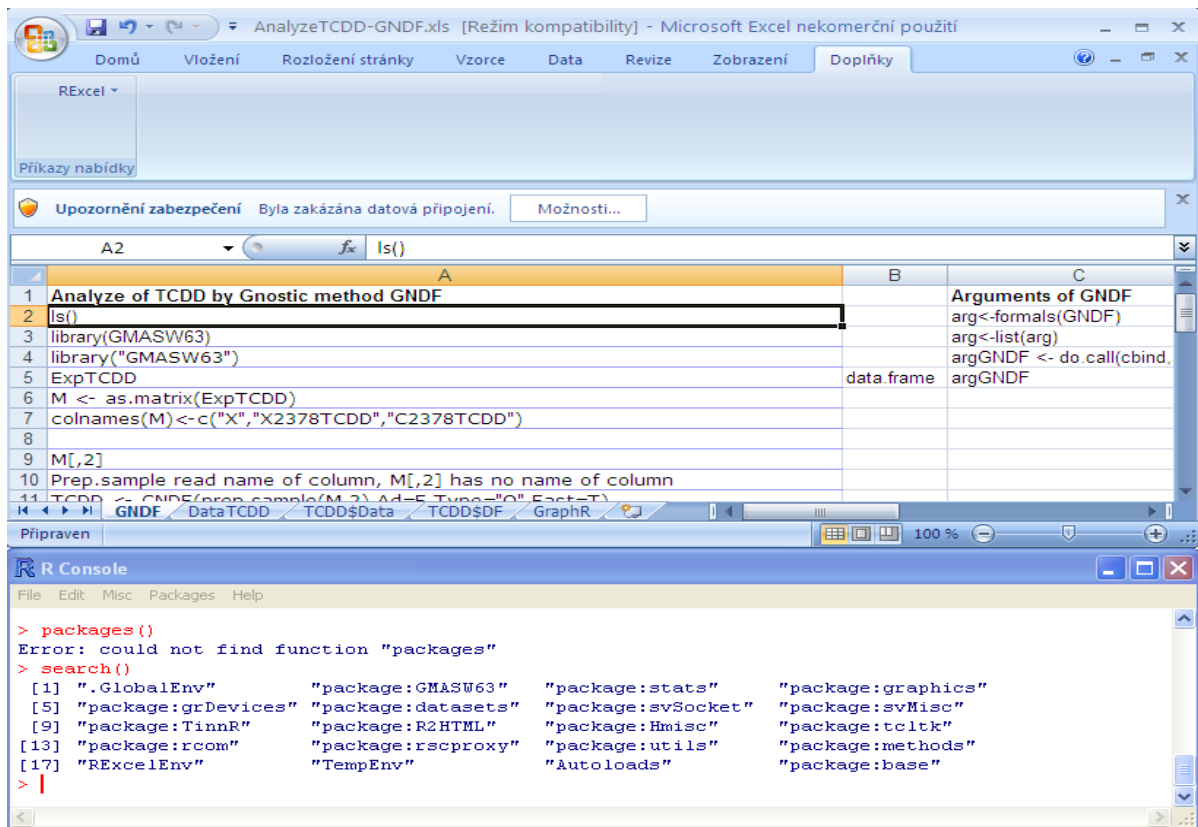


Fig.4 RExcel environment

A right click to a cell in the Excel sheet of the RExcel opens menu demonstrated in Fig.5.

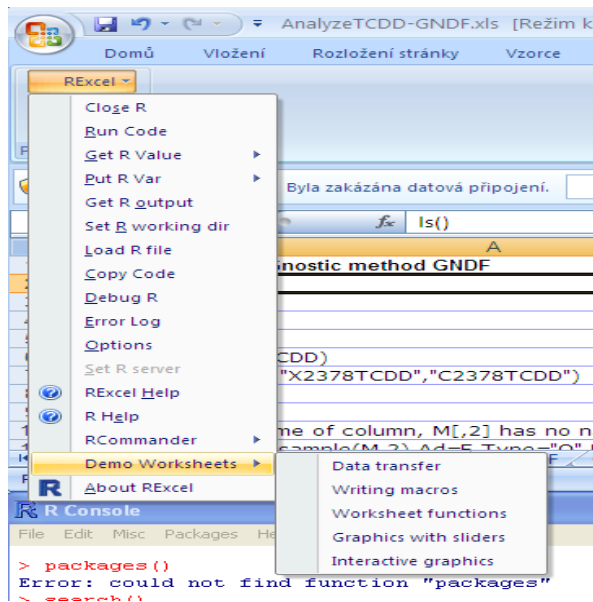


Fig 5 Menu for operations in RExcel



5 R LANGUAGE AND ENVIRONMENT

It follows from the exposed facts, that the R environment is suitable for development and applications of gnostic methods of the advanced analysis and for their wide distribution. This environment is being extensively improved and supported by permanently increasing number of contributors and users. This results in its extended functionality. To make use of all the capacity of this tool, it is necessary to be informed on its proper setting.

5.1 Setting the environment

The basic user interface is the already mentioned R Console GUI, which provides services to use many built-in functions and to support design of user's functions and tasks along with the access to the great number of available packages.

The basic setting of environment is stored in file `R_HOME\etc\Rprofile.site`. The default configuration is set by the installation procedure. Changes of this setting are necessary only if there is a special need. However, an important step is the setting of the working directory. It is easy done by means of the R console menu (Fig.7).

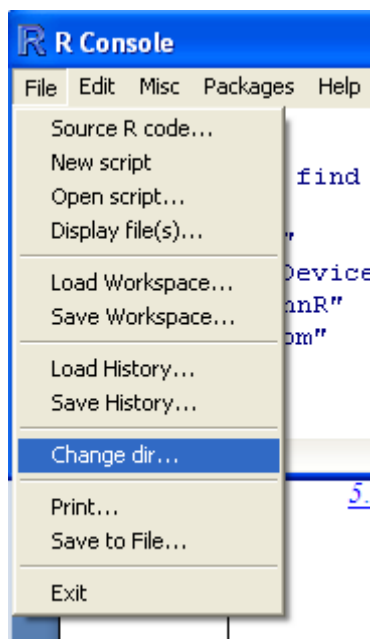


Fig 7. Setting the working directory

File `.Rprofile` placed in the working directory can be also used to store the setting of the environment.

The next important action is the setting of the workspace. This is the part of memory where the programmes store all the current values of variables, data structures, and functions. The workspace is automatically saved in working directory during operation and saved to a required place at the end of a session to be reloaded at the start of a new session. All this can be done by the line commands as well as by using the R console menu. The default file `RData` is used to save the workspace image.. There can be more image files in working directories kept enabling to work on various tasks.

Subsets of a workspace are local environments. Environments consist of a **frame**, or collection of named objects, and a pointer to an **enclosing environment**. The most common example is the frame of variables local to a function call; its enclosure is the environment where the function was defined. The enclosing environment is distinguished from the **parent frame**: the latter (returned by `parent.frame`) refers to the environment of the caller



of a function. The global environment `.GlobalEnv`, more often known as the user's workspace, is the first item on the search path. It can also be accessed by `globalenv()`. On the search path, each item's enclosure is the next item. The sequence of command line operations can be saved to the `.Rhistory` and reloaded again.

5.2 Source files

The source code of programs is in the form of a script file. It can be written in an arbitrary editor and stored in the ASCII text file with the suffix `txt` or `R` (default). To make a new function useable in the working directory, its script is to be put to the parser by using one of the ways:

- 1) By using the R console menu (File/Source R code...) (Fig.6). The directory window is opened to set the path to the prepared script.
- 2) File/New script: a new script window is opened for editing or copying the script.
- 3) File/Open script: to reopen a script.
- 4) By commands `fix(name)` or `name <- edit()` opening the built-in script window with an old text.
- 5) By submitting a text (or it's part) from the R Commander.
- 6) By using the RExcel spreadsheet.

The parser is activated by closing the script window. In the case of a programming error, the message is issued, otherwise the parsed command is executed or function's body made available for calling.

5.3 System packages

The other ways to add a new functionality to R is loading and installing a package. Package is a special group of programs used to distribute a new functionality among the R systems. Some standard packages with basic functionality are installed already with the R system. A package presents a source code packed up and stored in the system as a library. R has a particular directory `R_HOME\library` for all packages. The process of importing a new package consists of three parts. The first step is the downloading of a package from a chosen *CRAN mirror*, Then operations *Install packages* and *Load package* follow as shown in Fig. 7 and Fig. 8.

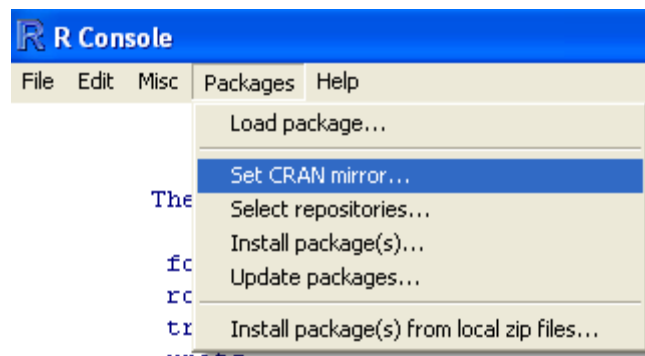


Fig. 7 Install new packages

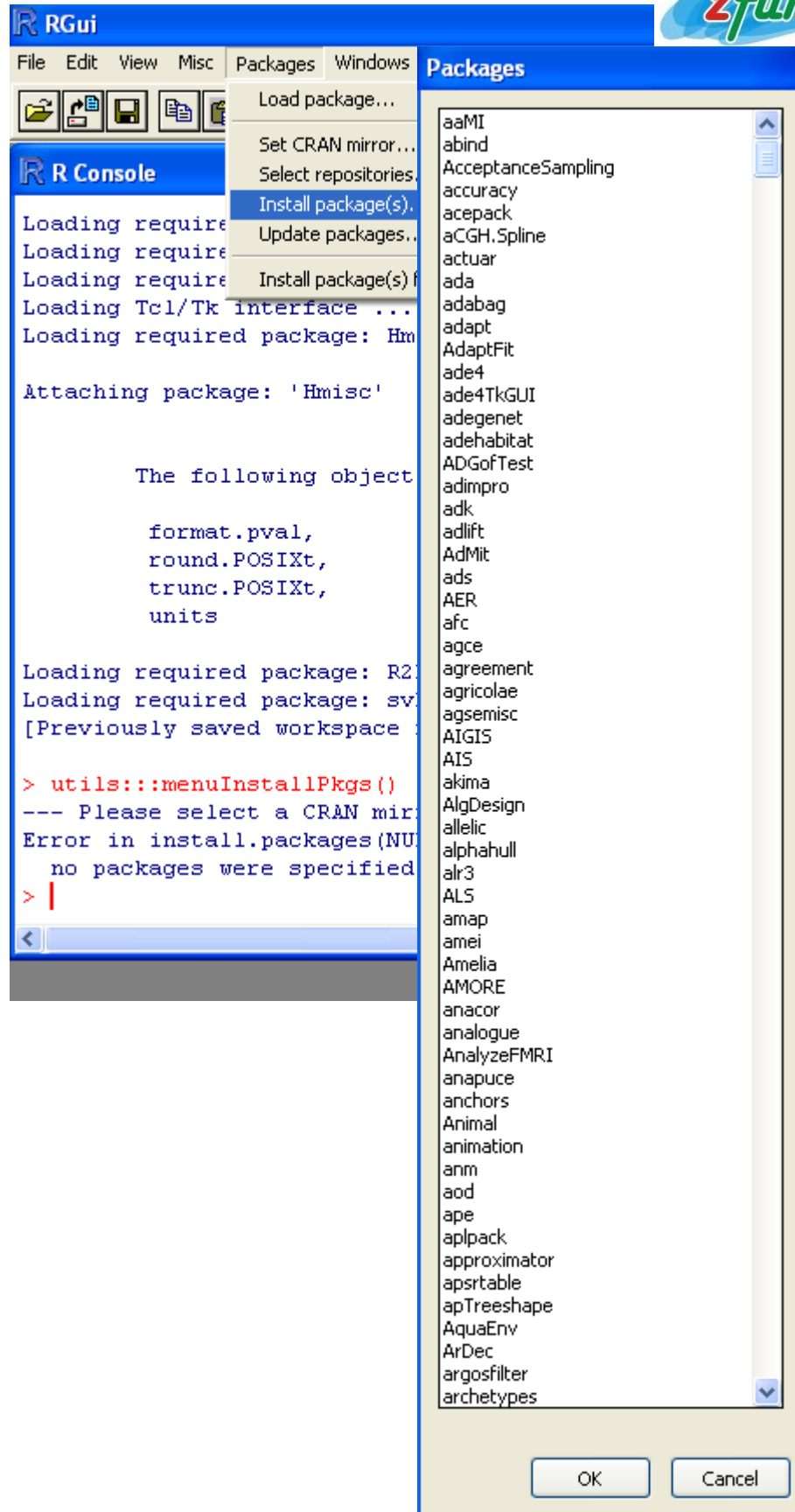


Fig.8 Selection of packages



There are many hundreds of packages currently available to choose. Their categories are: statistical, environment and programming tools and graphics interface. All actions can be done by means of the R console menu without necessity of using the command line.

5.4 R Console command line

Although the menu and graphical interface can be used for all of important actions in R environment, application of the command line cannot be completely avoided. The command line operations are performed in R console. What follows is a list of basic operations, a part of which was already described:

Setting environment:

List of objects	> ls()
Packages currently in the system	> search() > (.packages())
Work directory	> getwd()
Setting working directory	> setwd('C:/Home/Tools/R-working/R-training/')
Save workspace image	> save.image("D:\\R-PROJECT\\GUI-GNOSTIC\\test.RData")
Load work directory	> load("D:\\R-PROJECT\\GUI-GNOSTIC\\test.RData")
Help system	> help.start()
Help functions	> ?ls

Source code and edit scripts:

Read source code from text file	> source("C:\\Home\\Tools\\R-working\\TestGNDF\\GNDF.R")
or	> source("GNDF.R")
Edit function with save to the file	> edit(ls)
Edit function with save to the system	> fix(ls)

Packages:

Installed packages	> installed.packages()
Install package	> install.packages("GMASW63")
Load library	> library(GMASW63)



6 GNOSTIC FUNCTIONS IN R

As already mentioned, Gnostic functions were developed in S-PLUS. Their source texts were exported, checked with respect to syntax of R and completed by functions missing in R. Files prepared to be imported into the R system were thus created.

[6.1 Transfer of gnostic functions](#)

Gnostic source functions are arranged in two packages composed of sequences of complete procedures:

- 1) GMASW (the software of Gnostic Marginal Analysis),
- 2) GMDA (the Gnostic Multi-dimensional Analysis).

Each function begins with the function's name and the assigned operator *function()* followed by the function's body. It is closed by the command *return()*. The way of editing and parsing was already described. A special care is to be taken to debugging of functions differing in R from S-PLUS. If properly prepared, all the gnostic functions of a package can be parsed at once. An individual care is required for the tools of gnostic GUI and for graphical functions. Specific features of the hardware must be taken into account when setting the graphic details like colors, types and widths of lines, symbols etc.

It is useful to run functions on both R and S-PLUS systems and to compare the results. (When there is no access to S-PLUS, comparison of the results with examples in Guide can be recommended.)

[6.2 GUI for Gnostic functions](#)

Every Windows user welcome a comfort to control programs. This also applies to statistical programs and gnostic methods should have a comfortable environment, too. The mostly used command line operations were presented above. There exist graphical interface libraries in the R environment. Unfortunately, some of them are very complicated to use while others are not very useful. The issue is that to run functions, parameters are to be passed. Many of gnostic functions have a lot of arguments with default values or lists of values and it is impossible to remember the proper values. All of the graphical systems in R packages are oriented on other programming languages, because R language has not its own graphical functions. For application to gnostic software, the libraries *tcltk* and *tcltk2* were chosen, which are built in Tcl/Tk environment. Tcl is a free interpreter string-based command language developed for Linux platforms and Tk is its graphical extension. Using Tcl/Tk library, an interface for management and running gnostic functions was created. Tcl/Tk is a combination of a scripting language and of a toolkit for graphical user interfaces. The first version of this interface is shown in the next chapter in application to an example of the function GNDF (Gnostic Distribution Function).

[6.3 Example of marginal analysis](#)

GUI environment consists of a menu and a window for entering the parameters. An example of the menu is presented in Fig. 9.



Graphical outputs of distribution function GNDF are demonstrated in their standard form in Fig. 11.

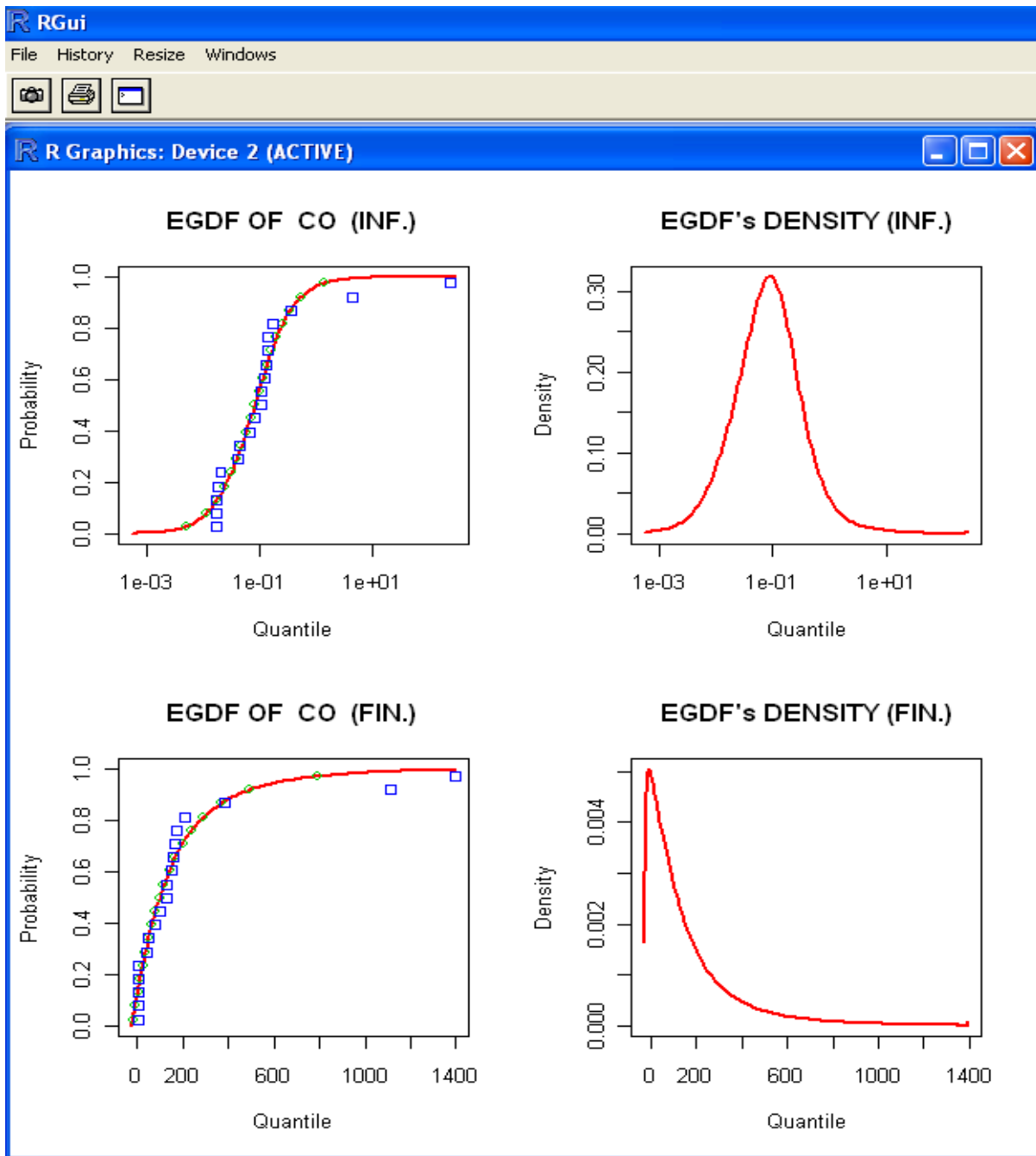


Fig. 11 Graphical output

The upper pair of graphs depicts the probability distribution function of the CO and its density over the (theoretical) infinite domain, while the lower ones show the distribution and density functions over the finite domain using the actual scale of input data. The red lines demonstrate the continuous fit of the discrete data values. The blue squares are data values of the Empirical Distribution Function set up directly by data. Green circles denote the projections of the data on the continuous distribution function ("the cross-section filtered data values"). Using both infinite and finite domains is advantageous in a reliable recognition of possible non-homogeneity of some data manifested by a second local maximum of the density function. The form of the



density function over the finite domain can be influenced by the boundary conditions (domain's bounds close to data values).

7 CONCLUSIONS

Gnostic data treatment methods were defined as application of the gnostic theory of individual uncertain data and small data samples. The structure of this software was described along with the details of applications and purpose of gnostic functions. Features of the open-source computing environment of the R-project were compared with the commercial system of S-PLUS of the Insightful Corp. Differences between both environments were studied to find necessary problems to be solved enabling the transfer of the gnostic software into the R environment to be completed. Achieved results and progress of the transfer process demonstrate, that the goal of making gnostic methodology broadly available via the open-source R environment and the web is realisable in the assumed time horizon.