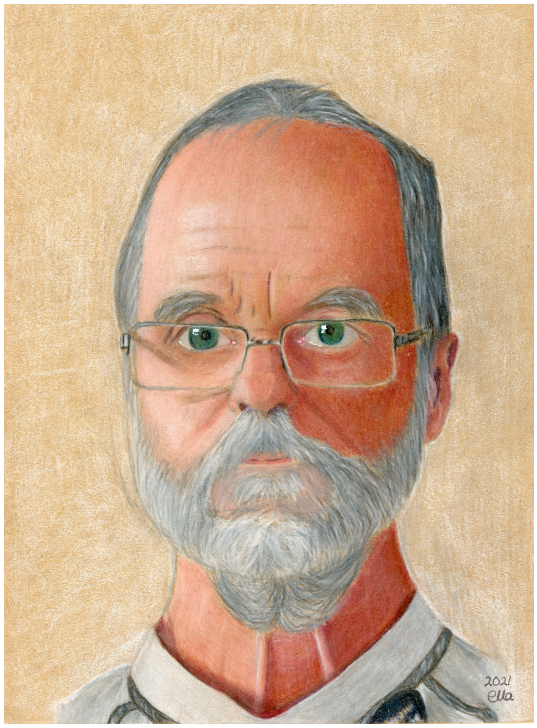
Jlp22
User's Guide

Version August 4, 2023

Juha Lappi
Reetta Lempinen



Contents

1 Preface

The **Jlp22** software is the newest version of **J** software. Don Shepard told via Fortran Discourse that that there is already **J** programming language. In order to avoid confusion, the name of **J** is now changed into **Jlp22**, the name which shows the inheritance from JLP. The name **J** was used in everything which happened to the software before this point. In the current version **Jlp22**, the beef if still the linear programming in forest management planning. But **Jlp22** has so many tools for data management and numeric computations, that also vegetarians can use it. Perhaps these tools will be first used for preprocessing data and postprocessing the results in conjuction of linear programming. The random number generators provide possibilities to do risk computations related to linear programming within **Jlp22**.

History of J before Luke

JLP is the linear programming (LP) software I made for the forest planning system Mela which was created by Markku Siitonen in Finnish Forest Researc Institute (Metla). Section 'JLP linear programming algorithm' describes the development and key points in the algorithm. Here only the development is described from the organizational point of view. JLP was published 1992 (see jlp92.pdf) and Finnish Forest Research Institute (Metla) started to distribute it. I wrote most part of it when payed by Academy of Finland. NMBU started to use JLP from the beginning in its GAYA software which was then called GAYA-JLP.

I wrote **J** as a successor for JLP. First version of **J**, version 0.9.3 was published in 2004. The beef in **J** was the same linear programming algorithm as in JLP. **J** provided many new possibilities for preprocessing of data and postprocessing of results, and these computations can be used also without LP problems. Even Bergseng from NMBU started to use **J** in a premature face and thus he had to tolerate quite much mix-up. The GAYA started to use **J** in the planning system called thereafter GAYA-**J**. Mela has not included **J** as as the optimizer, but it is possible to pull out simulated schedules from Mela and then define and solve linear programming problems using **J**.

Simosol started to use **J** as an optimizer from the beginning. Simosol has privately admitted that they are using much **J**. For some reason Simosol wants to keep this hidden from the users of their sofware and from the public. It would be important for an open source software that the users would rise a flag when they use the sofware. This would give feedback to the developers and would allow communication between users.

J version 2.0 published in 2013 made it possible optimize simultatenously forestry and transportations to factories and factory production. Reetta Lempinen started to work with me in the factory optimization. Reetta has professional training in programming, while I'm a self-learning amateur. Reetta had a clear head to keep complicated data structures I had created in order.

She also maintained the spirit during the black moments. The theory for **J** with factories was published Lappi and Lempinen 2014.

J in Luke

When working with Reetta, I noticed how terribly dirty the code was. I started to clean the code. In 2015 Metla and two other institutes were merged into Institute of Natural Resources Finland (Luke). When my retirement was approaching, I suggested Luke that **J** would be published as an open source software so that I could continue its development after my retirement. The leaders of Luke told that it is a good idea to open the software, and this is so important that Luke cannot let me take care of the publication but the leaders wanted to publish **J** themselves. This started a frustrating and humillating hostile overtake process which lasted six and half years. Eleven leaders of Luke were involved, and none of them was eventually interested to get the software publisehd.

I retired in February 2017. Research director Johanna Buchert signed the opening decision 30. 8. 2017. I was not informed of the decision. Later Luke rejected explicitly any cooperation in the publication or in the development of the software or in testing the behavior of Mela-software.

When Lauri Mehtätalo became a professor in Luke, he was able to mediate negotiations between me and Luke. The co-operation agreement was signed 7.10. 2021, in the 100 years celebration of NFI in Kings Hall in the medieval Olavinlinna castle in Savonlinna.

I got a permission to publish the software, and Luke agreed to provide treatment schedules simulated with Mela using NFI data from whole Finland and data from pulp mills and saw mills. Luke should provide me access to CSC supercomputers. Luke should distribute **Jlp22** via Github. This point was not necessary as Reetta adviced me how to use Github.

I was told in Olavinlinna that there has actually been no disagreement between me and Luke in this process. As I could not accept the interpretation that I had fighted six and half years against wind mills, I described in previous versions some details of the process. This kept my bitterness alive which was unfortunate for my peace of mind and also because I knew well how pathetic old bitter men are. After some recent rounds of opposite argumentation, the research director called me in January 12 2023 and admitted that Luke did not handle the opening and publication properly and that Luke will improve its procedures. I want now look forward, and I make only two comments.

First, I think that if a government research institute decides that a sowtware can be opened and published, the principal author should have the right to participate in the preparation of the formal opening decision and to take care of the publication.

Second. The software contained so much dirty code that it could not be developed without my involvement, and barely even maintained. The leaders of Luke did not know this because they

were not interested to discuss with me anything related to the state, publication or development of the software. I think that an expert organization should always consult best experts.

Rewriting the software

Open **Jlp22** was published 22.4.2022 in a web-seminar, and it has been distributed in Github since then. It contained many errors, but I wanted to get the open **J** started. After my retirement I have rewritten completely the software, the most part after the agreement in 7.10.2021. Only the sceleton and innermost loops in the optimization come from the time I worked for Metla and Luke. One important change was to put the software into 64-bit which allows the analysis of big data sets. The following manual tells what **Jlp22** can do now. The factory optimization is under development, and I will be published it later. Reetta has been actively testing the software, in addition to teaching the use of Github and giving encouragement all the time.

Book: Linear and nonlinear programming methods

I signed in Dec 15 2022 with Chapman & Hall the author agreement that I will write a book with title: 'Linear and nonlinear programming methods for joint analysis of forestry and factories: **Jlp22** approach'.

The book will be published with Open Access if I pay 16500 Euro when the book is ready. I want it be OA as it is based on an open software. Alongside with the book, there will be a script file so that the readers can repeat computations just writing the name of the example, similarly as the users of **Jlp22** can compute the examples in this manual. So I need data which I can utilize for such OA computations. My current agreement with Luke does not allow such utilizitation of Luke's data.

Acknowledgements

I acknowledge the contribution of Even Bergseng and Victor Strimbu from NMBU for co-operation in the development of **J**, Kyle Eyvindson from NMBU/Luke for providing me access to CSC super-computer, Hannu Salminen for co-operation. Hannu Hirvelä provided me Mela data, and Pekka Hyvönen factory data. Lauri Mehtätalo was building with me the links between **J** and R. Ron Shepard adviced in Fortran Discourse to make expansion of allocated vectors faster, in addition to advicing to chance the name of **J** software. I thank Reetta Lempinen for being a loyal colleague for a long time.

I thank my grand daughter Ella for the permission to use the portrait she drawed for my 70 yr birthday in 2021.

2 Recent changes

After renaming the software to **Jlp22**, this chapter starts to describe changes in new versions. The corrections of errors are not described in detail. The first version of **Jlp22** contains so many new properties that also old users should look at the manual. All the examples of the manual should now go through using ALL in running jexamples.inc.

Version 4.8. 2023 Some bugs corrected, and user errors are detected so that system crash is prevented.

Version 12.7. 2023. Numeric values can obtained from variable names using namenum() and namenum%() functions. This is useful with data with case or making lists using the new makelist-> option in data(). continuation character can be removed with '«'. This is useful when using ;do-loops to generate an input line. minloc() and maxloc() functions can be used to find the location of minimum and maximum in a vector or variable in data (thus returning the number of observation). New options step-> and points-> in drawline().

Version 29.6.2023. Function find_d() added. This function can be used to get ILIST of all observatiosn satisfying a condition. The data() and matrix() functions can get values from variable or using computation rules. Sparse data can be obtained using sparse-> option. Data with named cases is now obtained using case->option in data(), and datawcase() is removed. When printing is indicated with semicolon or double semicolon, and the object can contain piecewise linear terms. Nonlinear terms will be easy to include in a similar way.

Version 14.6.2023. New input programming function ;now() added. This is useful within trans() or within problem() when ;do() is present. unit-> option in jlp() can now define several variables, and the unit changes if atl least one variable changes. When printing using semicolon or double semicolon at the end of line, if Printpause has value 1 or 3 and Printresult indicates that lines with single semicolon are printed, then pause() is generated after printing. Similarly values 2 or 3 indicate pause(), if double semicolon lines are printed. area-> option may now work in jlp().

Version 5.5. 2023. Bug for treating extremly large matrices containg more elements than is the largest integer*4. Bugs in rfcode-> and rchead-> in data() function were corrected.

Version 30.4. 2023. Errors corrected. More examples for graphics. In rewriting of ther software, options are set for individual function occurences while in previous versions options were set for each code line. Earlier it was necessary to clear options within each function. Now they can be cleared just in one place before entering a new function. Now full advantage is taken of

the new option structure. It may be that some tricky complications may arise.

Version 26.4. 2023. draw() can now make circles etc. stat() can now compute statistics for a function without need to put the function values in the data. classify() merges classes with too few observations in a better way. Some errors corrected. New yet undocumented functions for stem curve analyses, e.g. for bucking purposes. Use of hierarchical data in jlp() was made compatible with the use of hierarchical data in other functions. Some not-yet-corrected bugs were possibly generated.

Version 19.3.2023 Function jlp() corrected, in the previous version it was completely out of order.

Version 18.3 2023 Function partdata() can pick part of data by dropping observations and/or variables. function info() prints current objects and options. The checking during the parsing phase whether a function has too many arguments is reactivated. It may cause unwarranted error conditions. It is possible to make lists of lists and pick the name of the list element which is also a list into the input paragraphs. This is useful when making the same analysis for many data objects. The stemcurve() function has new properties which can be documented if anyone is interested. Now the function can be used to analyse stem vectors generated with harvesters.

Version 14.3. 2023 Function exceldata() corrected. A bug corrected in jlpz(). More user errors noticed before a memory overflow. A bug corrected when space for numeric constants had to be increased.

Version 8.3.2023 Some bugs corrected, e.g. in data() function and one bug in the precompiler. Some user errors are reported better. The manual is updated. There is now function stemcurve() which defines a stem curve used with linear interpolation. This is reported if anyone is interested.

Version 27.2. 2023 Some bugs corrected, i.e. in jlpz(), and some user errors are detected to avoid memory overflow. If values of matrix elements were given in the input paragraph after matrix(), it was not possible to separate numbers with spaces (even if this was not told in the manual). Now spaces are allowed as separators but input programming is not. If input programming tools are needed, it is easly to add option which makes it possible. ;if();then structures in the input programming were not described in the manual, now they are.

Version 23.2.2023. The most important change is that the data() function is rewritten. Now it is simpler and more easy to maintain. It is not possible to make two DATA objects with a single call to data(), but it is not really reasonable to put unit data and schedules data into the same file. There are some other changes which are note yet fully reported. Examples go through but there are some not properly tested features.

Version 25.1.2023. Factory optimization started. It does not work yet but the example shows how it can be used in the future.

Version 31.12.2022. jlp() solves (again) problems without any constraints. The showdomain-> works also for problems without contraints. The new subroutines needed for this will smooth the road to factory optimization.

Version 30.12.2022 jlp() has now showdomain-> option which defines additional domains for which the sums of x-variables are also computed. This option was already in JLP. All sums of x-variables are now stored into variables with names starting with Output%. Sums in domains are stored in variables with postfix [domaindefinition], e.g. [site.lt.3].

In version 28.12. 2022 joindata() which builds schedules data from tree format data has two new options which allow combining several data sets faster and it is possible to save memory so that larger schedules data fits the memory. Some fine tuning of the preface of the manual.

In version 27.12. 2022 the main change is the possiblity to handle so large matrices that their elements cannot be referred with 4-byte integers. Such matrices are needed for schedules from the whole country. This change did cause a large amount of boring editing. There are places which are not put into 8-byte integer mode, and I will update these places whenever needed. In the data structures, I have the hidden property to work with single precision matrices and with tree-structure data which is not made into the tabular form using joindata() function. I will start to utilize these implicit properties when there is lack of memory even in the 64-bit environment. In this Github publication, the release exe-file is not updated.

Version Dec 20 2022 provides simple data formats for storing schedules so that in the simulation tree only thos nodes after branching are stored. See keepopen-> and continue-> options in data(). Function joindata() is used to build the schedules from their components. Function splitdata() can be used to split data caontaining whole schedules into these new compponent datas. splitdata() can be used to convert data generated with previous simulators into this new format. It is recommeneded that a simulator generating data for **Jlp22** starts to write directly into this new format.

A bug in Fletchers code was found. When after some pivot steps only residual variables are in the basis, the code computed wrong shadow prices, and the objective was getting worse. This situation will not happen in ordinary linear programming problems. When hunting the bug, I developed new debugging tools, so I hope that all users would now send problematic runs to me.

Now the variable telling the number of observations in DATA datab is always datab%obs. Similarly the variable telling how many children that observation has in the lower level is now always datab%nobsw and it must be the first varaible in the list of variables. If the first variable is %nobsw, then its name is automatically changed into the proper form. This new legislation was necessary to keep the new multilevel datas in order.

Version Dec 20 2022 gives possibility to change fonts in figures.

3 Introduction

Jlp22 can be used as is, i.e. using exe files. The User's guide concentrates on using binary files, but some reference is also made to additional possibilities offered by the open **Jlp22** code.

3.1 Using Jlp22 exe files

Jlp22 is a general program for many different tasks. In one end, **Jlp22** is a programming language which can be used to program several kind of applications and tasks, starting from computing 1+1, either at sit> promt or inside a trasformation object. In the other end, it contains many functions which can do several tasks in statistics, plotting figures, deterministic and stochastic simulation and linear optimization. Forest planning with factories using jlp() function can take several hours. The general **Jlp22** functions and **Jlp22** trasformations can be combined in many ways.

There are several alternatives for doing general mathematical and statistical computations available in **Jlp22**, most prominent being R. For users of R, the most interesting functions in **Jlp22** are evidently the linear programming functions which utilize the the structure of typical forest managment planning problems. The forest management planning can now in principle be combined with factories. Just now a completely new version of factory optimization is under development, and thus it is not usable. R users can do general data management in R and use **Jlp22** only for linear programming. But after learning basics of **Jlp22**, it may be more straightforward to do also data management in **Jlp22**.

Jlp22 can now be used for general matrix computations. I have included in **Jlp22** all matrix functions of Matlab which I found useful in a consulting project. **Jlp22** uses gnuplot for making figures. It is straigtforward to extend these graphics functions. **Jlp22** can be used an interface to gnuplot graphics. **Jlp22** contains many tools to deal with classified data.

The current binary versions of **Jlp22** are developed using Gfortran Fortran 90 compiler in MSYS2 MINGW 64-bit environment under Windows 10. Binary versions are ordinary console applications. It is recommended that **Jlp22** is used in command prompt window, so that if execution of **Jlp22** terminates unexpectedly, the error debugging information remains visible. With the debug version the problematic line is indicated. See chapter 38 for more information of error handling.

3.2 System requirement

!

Figures are made with Gnoplot. gnuplot is freely available at

https://sourceforge.net/projects/gnuplot/files/gnuplot/5.4.2/

Download download gp542-win64-mingw.exe. This will install gnuplot on your windows system under C::\Program Files
textcolortealgnuplot˙Jlp22 will strat gnuplot automatically when plotting figures if gnuplot is on the PATH (see section Installing gnuplot and **Jlp22**). Dcomentation is found also in **??**

3.3 Developing own Jlp22 functions in Fortran

The user's of **Jlp22** can utilize the open source of **Jlp22** in principally two different ways. Either the user can develop new versions of existing **Jlp22** functions or the user can can make new functions. It both cases the user should make new developments using so called own functions, which can be independently of the main **Jlp22** functions. When modifying an existing **Jlp22** function, the user should make a copy of the function under a different name. Then the old and new versions can exist simultaneously in the function space of **Jlp22**. It is very easy to add new functions in **Jlp22** and even more easy to add new options.

When deloping a new methods in **Jlp22**, it is possible to first use the **Jlp22** script language to make developments. Then the user can make an own function where the method is written in Fortran to make the method faster. When writing new methods in Fortran, the user can concentrate on essential parts of the method, and utilize the standard data management services provided by **Jlp22**.

See **Jlp22**Development.docx to start develop the software or to add own functions. This guide is not up to data, so all who are willing to start development shoud caotact first juha.lappi.sjk@gma
The development package contains, in addition to source code for the standard **Jlp22** software, program Jmanual which can be used to generate Latex code for the manual and the include file for examples. The precompiler Jpre writes necessary Fortran statements to access all global **Jlp22** data structures, makes indentations and checks if-then and do strutures and gives better error messages for them than Gfortran.

4 Loading Jlp22 from Github

The Github ditribution was made under the guidance of Reetta Lempinen. This section describes the folders of the Github distribution.

4.1 Git package

The **Jlp22** package can be loaded by pressing load zip button under the green code button in the right side of the page Github.com/juhalappi/**Jlp22**. The package contains the following files.

- LICENSE the license file
- README.m readme file

The package contains following folders:

- **JR** using **Jlp22** from R, courtesy of Lauri Mehtätalo.
  - j.par default include file for starting **Jlp22**. It contains also a LP problem and its solution.
  - JR_0.0.tar.gz File needed to use Fortran subroutines in R
  - cdat.txt
  - xdat.txt
- **Jbin** binary .exe files and dll files
  - jlp22.exe Debug version of **Jlp22**.
  - jlp22r.exe Release version of **Jlp22**. If the release version crashes, use the debug version to get more information of the cause.
  - jmanual.exe makes the latex code file jmanual.tex and the example file jexamples.inc.
  - jpre.exe the precompiler which generates the code for accessing variables in modules, makes indentations and gives better error messages for mixed do-loops and if-then structures than Gfortran.
  - dll: libgcc_s_seh-1.dll, libgfortran-5.dll, libquadmath-0.dll and libwinpthread-1.dll which must be available in the path. e.g., in the same folder as the exe
- **Jdocdemo** documents and include file for running examples from User's guide
  - **Jlp22**.pdf This user's guide made with Latex using Overleaf `http://www.overleaf.com`.
  - jexamples.inc include file which can be used to run all examples in the manual and which is generated with jmanual.exe
  - cdat.txt example unit data file for small jlp() example in jexample.inc.
  - xdat.txt example schedule file for small jlp() example
  - jlp92.pdf Manual of old JLP which explains the theory behind the jlp algorithm
  - lappilempinen.pdf Paper explaining the theory behing factory optimization.
  - hyvonenetal2019.pdf A paper utilizing the factory optimization.

    - Jlp22development.docx not up-to-date manual for developers
    - fletcher.pdf The theory behind the subroutines of Fletcher
- **Jmanual** Source files for making Latex code for the manual and the include file for running examples
    - jmanual.f90 source for making the Latex code and jexamples.inc
    - jmanual.tex Latex code generated with jmanual.exe
    - jsections.txt describes manual sections not in source files
    - jsections2.txt tells in what order sections found in jsections.txt and source files are put into the manual and what is the level of the sections
    - main.tex Preample code containing Latex definitions
    - Makefile_debug Makefile for making jmanual.exe
- **Source** source code before precompilation
    - fletcherd.for Fletchers subroutines turned into double precision
    - j.f90 code for **Jlp22** functions
    - jlp.f90 code for linear programming
    - j.main main prorgam for calling **Jlp22** when used as is, if **Jlp22** is used as a subroutine then this must be made a subroutine
    - jmodules.f90 data structure definitions
    - jutilities.f90 subroutines for handling objects etc.
    - jsysdep_gfortran.f90 system dependent routines
    - matsub.f subroutines obtained from other sources, e.g. from Netlib
    - other subroutines for setting up users own functions
- **Source2** source code files after precompilation in addition to files in Source (such files which are not precompiled are put in both folders)
    - makefile_debug makefile for making debug version **Jlp22**.exe
    - makefile_release makefile for making release version **Jlp22**r.exe

4.2 Loading the package

The installation is here described from the viewpoint of an user who just want to use **Jlp22**, not develop it (yet).

The load zip button loads file **Jlp22**-master.zip. Copy this file into a proper folder. Let it be Jgit. Clicking the **Jlp22**-master.zip icon shows folder **Jlp22**-master in 7Zip (if this is installed). Clicking unpack 7zip unpacks it to the folder Jgit. Too many folder levels are avoided, if everyhing in folder **Jlp22**-master is copied directly in folder Jgit.

The jlp22.exe is the debug version of **Jlp22** and **Jlp22**r is the release version. The debug version should be used still used to set up a project. When everythings seems to work, the user can try the release version in production runs where the time is of some concern. Let us assume that the working folder is jtest.

5 Install

5.1 Install Jlp22

After loading **Jlp22** from the github.com/juhalappi/**Jlp22**, the folders of the exe files must be put into the environmental variable PATH. In my computer the PATH contains line C:\jlp22 An ordinary user may use **Jlp22** and **Jlp22**r from the folder Jgit**Jlp22**bin.

5.2 Install gnuplot

Let us then install gnuplot. Go to page `http://\textcolor{teal}{gnuplot}.info`, and select there download, and in the download page select green 'download latest version', which loads 'gp543-min64-mingw.exe' (or similar), and if you let it install with usual yes-next procedures gnuplot is installed in folder c:\program files\gnuplot which should be placed into the environmental variable Path.

5.3 Install Rscript

If you plan to run R scripts from **Jlp22**, Rscript program mus be installed and the folder must be put into the PATH. In my computer the folder is C:\Program Files\R\R-4.1.2\bin

5.4 After install restart

After editing the environmental varaible Path, the computer mus be restarted.

6 Typographical conventions

6.1 sin() is a function

6.2 xrange-> is an option

6.3 ;incl is and input programming function

6.4 matrixa is an object

Names matrixa, matrixb, proba,probb, jlpa,jlpb etc are common names used for matrices etc. Coloring of object names is not yet complete.

6.5 MATRIX is shorthand notaion for a matrix object

if(type(matrixa).eq.MATRIX)then ..

7 Subroutines from other sources

The following subroutines are obtained from other sources.

- subroutine tautsp used in j_function tautspline from Carl de Boor (1978) A practical guide to splines. Springer, New York, p.310-314 No licence restrictions known. Distribution: `https://www.researchgate.net/publication/200744645_A_Practical_Guide_to_Spline`, see also `http://pages.cs.wisc.edu/~deboor`
- Several subroutines from `www.netlib.org/lapack` with licence : `http://www.netlib.org/lapack/LICENSE.txt`

8 Running examples

If you plan to edit the example file, copy jexamples.inc and jlp22.pdf from jdocdemo folder into the Jtest folder. It is not wise to start working in the Jdocdemo folder, because if you load a new version of **Jlp22** in a similar way into Jgit folder and allow the computer replace existing folders with the same name, you would loose the work done in Jgit folder.

Open the command prompt and move to the directory 'jtest' by 'cd' commands. It is possible to use **Jlp22** also directly, but its necessary to use **Jlp22** through the command prompt at this testing phase, because then, if **Jlp22** crashes, the error messages do not disappear. In first time, you may want to change the colors etc of the commant prompt (color may take effect only after

closing command prompt and reopening it). It may be reasonable to tick all editing properties under the properties button of the command prompt.

If you would like that when starting **Jlp22** in this folder, **Jlp22** immediately starts with examples, make file j.par to folder jtest and write to it

;incl(jexamples.inc)

It is possible to write to the first line of j.par

*3000

which would mean that **Jlp22** would generate intially 3000 named objects.

Then start **Jlp22** by giving command **Jlp22** at the command prompt. If you have not done j.par file, write at the sit> prompt: ;incl(jexamples.inc). Alternatively you can diretly start in the jexamples.inc by launching **Jlp22** by
Jlp22 jexamples.inc

Jlp22 will then print all shortcuts available. You can run examples one by one by giving example shortcuts individually, or you can run all examples by shortcut ALL. If you give shortcut ALL, **Jlp22** asks whether a ;pause is generated after each example ('pause after each example(1/0)>'). Even if pause() is not generated after each example, it is generated after each plot. This can be prevented by pressing <return> when **Jlp22** asks value for fpause. Examples in the ALL section can anyhow be started at any point by putting label ';current:' to any point after label ;ALL: and giving shortcut 'current'. This is also handy when examples have errors which break the execution. If the errors are made intentionally to demonstrate error situations, the interruption of excution after these intentional errors are prevented so that the jexamples.inc have command Continue=1 before the intentional error. Theafter the normal error handling is put on by command Continue=0.

It is useful to keep the manual open and follow simultaneously the manual and the execution of examples.

9 Command input and output

Jlp22 has two programming levels. First level, called input programming, generates text lines which are then transmitted to the parser which generates code which is the put into transformations sets or excuted directly. Input programming loops make it possible to generate large number of command lines in a compact and short form. This chapter describes input programming concepts and commands.

9.1 Input record and input line

Jlp22 reads input records from the current input channel which may be the console, file or a text object. When **Jlp22** interprets input lines, spaces between limiters and function or object names are not significant. In input programming, functions start with ';' which is part of the function name (and there can thus be no space immediately after ';'). If a line (record) ends with ',' ,'+', '* ´ , '-', '(', '=' or with '>', then the next record is interpreted as a continuation record and the continuation character is kept as a part of the input line. Tf a record ends with ',' and the next record starts also with ',', only one ',' is obtained. If a line ends with '>>', then the nex line is also continuation line, and '>>' is ignored. All continuation records together form one input line. In previous version input programming functions operated on input lines but now they operate on recors. One input record can contain 4096 characters, and an input line can contain also 4096 characters (this can be increased if needed). There can be comment lines within a command line.

When entering input lines from the keyboard, the previous lines given from the keyboard can no more be accessed and edited using the arrow keys owing to MSYS2 MSYS environment used to build the exe-file. To copy text from the **Jlp22** window into the clipboard right-click the upper left icon, select Edit, and then select Mark. Next click and drag the cursor to select the text you want to copy and finally press Enter (or right-click the title bar, select Edit, and in the context menu click Copy). To paste text from the clipboard into the **Jlp22** command line right-click the title bar, select Edit, and in the context menu click Paste. Console applications of Intel Fortran do not provide copy and paste using <cntrl>c and <cntrl>v. An annoying feature of the current command window is that it is possible All input lines starting with '*' will be comments, and in each line text starting with '!' will also be interpreted as comment (!debug will put a debugging mode on for interpretation of the line, but this debug information can be understood only by the author). If a comment line starts with '*!', it will be printed.

9.2 Input Paragraph

Many **Jlp22** functions parsed and executed (interpreted) at the command level need or can use a group of text lines as input. In these cases the additional input lines are immediately after the function. This group of lines is called input paragraph. The input paragraph ends with '/', except the input paragraph of text function text() and txt() end with '//' as a text object can contain ordinary input paragraphs. It may be default for the function that there is input paragraph following. When it is not a default, then the existence of the input paragraph is indicated with option in-> without any value. For many functions, the input paragraph can contain input programming commands. The resulting text lines are transmitted to the function which interprets the input paragraph either using the parser (e.g. trans() function) or by other means (e.g problem()).

Example 9.1 (inpuparag). `Example of inputparagraph`

```
transa=trans()
```

```
a=log(b)
write($,'(~sinlog is=~,f4.0)',sin(a))
/
b=matrix(2,3,in->)
1,2,3
5,6,7
/
```

9.3 Shortcuts

Command shortcuts are defined by defining character variables. When entering the name of a character variable at sit> prompt or from an include file, **Jlp22** excutes the command. The command can be either input programming command or one-line command. The file jexamples.inc shows an useful way to organize shortcuts and include files.

Example 9.2 (shortex). Example of using shortcuts and include files

```
 short1='sin(Pi)+cos(Pi);'
short1
te=text()
this=thisfile()
ju1=';incl(this,from->a1)'
ju2=';incl(this,from->a2)'
;return
;a1:
!!  greetings from a1
;return
;a2:
** here, jump to a1
ju1
*!  back here, return to sit> or next example in ALL
;return
//
write('shortex.txt',$,te)
close('shortex.txt')
;incl(shortex.txt)
ju1
ju2
delete_f('shortex.txt')
te=0 !delete also text object te
```

9.4 system() executes Windows commands

Windows system commands can be given by

system('command')

For instance

system('cls') clears screen system('dir') prints the directory

9.5 cls clears screen

command cls clears screen, thus cls is equivalent to system('cls')

9.6 Input programming

The purpose of the input programming is to read or generate **Jlp22** commands or input lines needed by **Jlp22** functions. The names of input programming commands start with semicolon ';'. There can be no space between ';' and the following input programming function. The syntax of input programming commands is the same as in **Jlp22** functions, but the input programming functions cannot have an output. There are also control structures in the input programming. An input paragraph can also contain input programming structures.

9.7 Labels in input programming

The included text files can contain labels. Labels define possible starting points for the inclusion or jump labels within an include file. A label starts with semicolon (;) and ends with colon (:). There can be text after the label and the text is printed but otherwise ignored.

```
;ad1: At this point we are doing thit and that
```

Note 9.7.1. The definition of a transformations object can also contain labels. These labels start with a letter and end also with colon (:). When defining a transformation object with trans() function, the input paragraph can contain input programming labels and code labels. It is up to input programming what code alabels become part of the transformation object.

9.8 Changing "i" sequences

If an original input line contains text within quotation marks, then the sequence will be replaced as follows. If a character variable is enclosed, then the value of the character variable is substituted: E.g. directory='D:/ɹ/ńame='area1' extension='svs' then in->'"directory""name"."extension"' is equivalent to in->'D:/ɹ/1.svs' If the "-expression is not a character variable then **Jlp22** interprets the sequence as an arithmetic expression and computes its value. Then the value is converted to character string and substituted into the place. E.g. if nper is variable having value 10, then lines

```
x#"nper+1"#"nper" = 56
chv = 'code"nper"'
```

are translated into

```
x#11#10 = 56
chv = 'code10'
```

With " " substitution one can define general macros which will get specific interpretation by giving values for character and numeric parameters, and numeric parameters can be utilized in variable names or other character strings. In transformation sets one can shorten computation time by calculating values of expressions in the interpretation time instead of doing computations repeatedly. E.g. if there is in a data set transformation x3 = "sin(Pi/4)"*x5 Then evaluation of sin(Pi/4) is done immediately, and the value is transmitted to the transformation set as a real constant. If value of the expression within a "" sequence is an integer then the value is dropped in the place without the decimal point and without any spaces, otherwise its value is presented in form which is dependent on magnitude of the value. After J3.0 the format can be explicitly specified within [] before the numeric value. Eg. text can be put into a figure as fig = drawline(5,5,mark->'y="[f5.2]coef(reg,x1)"*x1+"[f5.2]coef(reg,1)"") See file jex.txt and Chapter 8 for an ex

9.9 ;incl() lines from a file

Includes lines from a file or from a text object. Using the from-> option the include file can contain sections which start with adresses like ;ad:
and end with
;return

Args	0|1	Ch|Tx
	file name. Default: the same file is used as in the previous ;incl().	
from	N|1	Ch
	gives the starting label for the inclusion, label is given without starting ';' and ending ':'.	
wait	N|0	
	Jlp22 waits until the include file can be opened. Useful in client server applications. See chapter **Jlp22** as a server.	

Note 9.9.1. Include files can be nested up to 4 levels.

Note 9.9.2. In the current version of **Jlp22**, the file name and the adress can be without apostrophes ' ', but the previous names with apostrophes are allowed.

Note 9.9.3. It is possible to start reading the script from the same file. In that case ;return returns the reading of the script after the ;incl- line.

Note 9.9.4. ;goto(adr) and ;incl(from->adr) go to the same line in the include file but after ;goto the ;return-command closes the include file but after ;incl() the ;return-command returns the control to the calling point.

Note 9.9.5. See Chapter Defining a text object with text function and using it in ;incl how to include commands from a text object.

Note 9.9.6. When editing the include file with Notepad ++, it is reasonable to set the language as Fortran (free form).

Example 9.3 (inpuincl). Example of ;incl()

```
 file=text()
** File start
i=1;
;goto(ad1)
** Never here
i=2;
;ad1:i=66;
**After ad1
;goto(ad2,ad3,2) !select label from a label list
;ad2:
** After ad2
i=3;
;ad3:
** After ad3
i=4;
;ad4:
** After ad4
i=5;
;ad5:
** After ad5
i=6;
//
file;
;if(exist_f('file.txt'))delete_f('file.txt')
write('file.txt',$,file)
close('file.txt')
;incl(file.txt)
;incl(file.txt,from->ad2)
```

Note 9.9.7. The adress line can contain comment starting with '!'.

9.10 ;goto()

Go to different adress in ;incl() file.

Args	1	CHAR
	The label from which the reading continues. With ;goto(adr1) the adress line starts ;adr1:	

Example 9.4 (inpugotoex). Example of ;goto() and ;incl()

```
 gototxt=text()
!!  Start jumping
;goto(ad2)
;ad1:
!!Greetings from ad1
;return
;ad2:
!!Greetings from ad2
;goto(ad1)
//
print(gototxt)
if(exist_f('goto.txt'))delete_f('goto.txt')
write('goto.txt',gototxt)
close('goto.txt')
print('goto.txt')
;incl(goto.txt)
;incl(goto.txt,from->ad1)
delete_f('goto.txt')
```

Note 9.10.1. In the previous versions the adress had to be withing apostrophes ' ', but now tis is not necessatry even if it is possible.

9.11 ;do() input records in a loop.

Args	3|4	Var,Num..
	Arguments are: iteration index, starting limit, final limit and step. First argument must be a variable name and others can be REAL variables or numeric constants.	

Example 9.5 (inpudoex). Examples of ;do()

```
 ;do(i,1,2)
x"i"="i"*10
print('Greetings from iteration "i"')
;enddo
```

```
print(x1,x2)
varlist=list(x0,y0,
;do(i,1,3)
x"i",y"i",
;enddo
x4,y4);
```

```
<print('Greetings from iteration 1')
'Greetings from iteration 1'
<print('Greetings from iteration 2')
'Greetings from iteration 2'
sit< print(x1,x2)
<print(x1,x2)
x1=    10.000000000000000
x2=    20.000000000000000
```

9.12 ;if()...

If the condition within the parenthesis if TRUE (i.e. has a nonzero value)the following text if the next input line generated.

Note 9.12.1. In the previous **J** version it was possible to write these one line ifs outsoed a TRANS object the semicolon, i.e. in form if().... Now this is not allowed, even if it would be simple to implement in order to make a clear distinction between input programming and transformations

9.13 ;if();then

If the conditon within the parenthesis is tRUE (i.e. ahas a nonzero value, the text up to ;elsif());then or ;else becosme part of the input. Thus the whole thing can look like ;if(...);then
...
;elseif(...);then
...
;elseif(...);then
...
;else
...
;endif

9.14 ;sum() sums into input

Jlp22 an generate text of form part1+part2+...partn into input line using input programming function ;isum(). The syntax of the function is as follows:
;sum(i,low,up,step)(text)
or
;sum(i,low,up)
Arguments low, up and step must be integers (actually from nonintger values, the integer part is used) or REAL variables. Thus te valuse cannot be obtained from arithmetic operations. Sum is useful at least in problem() function.

Example 9.6 (inpusumex). Example of ;sum()

```
 prob=problem()
;sum(i,1,5)(a"i"*x"i")==max
;sum(i,1,3)(a"i"*x"i")<8
/
```

Note 9.14.1. ;dif() works similarly for minus

9.15 ;dif() differences into input

Jlp22 can generate text of form part1-part2-...partn into input line using input programming function ;dif(). The syntax of the function is as follows:
;dif(i,low,up,step)(text)
or
;dif(i,low,up)
Arguments low, up and step must be integers (actually from nonintger values, the integer part is used) or REAL variables. Thus te valuse cannot be obtained from arithmetic operations. ;dif() is useful at least in problem() function.

Note 9.15.1. ;sum()() works similarly for plus. See ;sum() for examples.

9.16 ;pause in script processing

Including input from an include file can be interrupted using an input programming command ;pause promt or the **Jlp22** function pause('<prompt>'). In both cases the user can give **Jlp22** commands, e.g., print objects, change the value of Printdebug etc. The difference is that pause('<prompt goes first through the interpreted and the interptreted code is transmitted to the **Jlp22** function driver. In the ;pause- pause it is possible to use input programming commands while in pause()- pause it is not possible. In both cases, when an error occurs, the control remains at the pause prompt. If the user is pressing <return> **Jlp22** continues in the include file. If pause() is part of a transformation object, pressing <return>, the function driver continues in the transformation

object. If the user gives command 'e' or 'end', then **Jlp22** procees similarly as if an error had occured, i.e. print error messages and returns control to sit> -promt.

9.17 ;return from ;incl

;return in an input file means that the control returns to the point where a jumpt to an label was found. Two different cases need to be separated:

- The conrol came to the starting label or to the beginning of the include file from outside the current include file using a ;incl command. Then ;return returns the control to upper level include file or to the sit> prompt.
- The control came to the starting label from within the same include file using an explicit ;incl or ;incl is generated with command shortcut.

9.18 ;where the current line in ;incl -files

9.19 Objects with common part of name

There are two ways to acces objects having common part in name. These methods are especially useful when seeing sub objects or side object produced with funtion whose names start with output%.

9.19.1 Printing names of subobjects

The names of subobjects generated with a function can be seen according to the following example.

Example 9.7 (subobjex). Seeing subobjects

```
 dataa=data(in->,read->(x,y))
1,2
3,4
/
li=;list(dataa%?);!prints subobjects
```

9.19.2 ;list(part%?) LIST of objects having common part in name

;list() Makes a list of objects. The example shows how the list can be printed and how to print all objects in the list.

Example 9.8 (subobjex2). ex2

```
 dataa=data(in->,read->(x,y))
```

```
1,2
3,4
/
lista=;list(dataa%?);
**Then all objects in the list are printed.
@lista;
**Seeing and printing relatives
x1a...x3a=3...5


  rel=;list(x?a);
@rel;
```

9.20 @List expands a LIST

If a LIST is generated explicitly with list() function or as a byproduct of an other function, then all elements of the list can put into the code using @-sign in the front of the name of the list. The code then works similarly as if all the object names had been written consecutively and separated with commas.

Note 9.20.1. A function or an option can have several parts in the arguments generated with @

Note 9.20.2. In the earlier versions of the software,expanding lists with @ was implemented during the generation of the input line similarly as in the input programming proper. This was stupid. It is more simple and more efficient to to implement the expanding of lists during the code parsing stage. This is presented here because, @-sign works as if it were part of input programming.

Example 9.9 (expandex). Example of expanding lists with @

```
 list0=list(site,altitude)
list1=list(
;do(i,1,3)
vol"i",ba"i",
;enddo
vol4,ba4);
dat=data(in->,read->(@list0,@list1))
1,2,3,4,5,6,7,8,9,10
2,3,4,5,6,7,8,9,10,11
/
stat()
```

9.21 @List(index) Gets a name from a LIST

The name of an object in a LIST can be obtained into the input using the following example.

Example 9.10 (inpulistelem). Getting name from LIST

```
 lis=list(c2...c5);
@lis=2...5;
@lis(2)=6;
@lis(4)=@lis(1);
@lis(2)%@lis(4)=66;
** Utilizing LISTs of LISTs
lists=list(lis1...lis3);
lis1=list(x1...x4);
@lis1=1...4
lis2=list(z1...z3);
@lis2=10,20,30
lis3=list(y1,y2);
@lis3=100,200
** @lists(2) drops lis2 to the input line giving @lis2; which cause
** that elements of lis2 are printed
@@lists(2);
```

9.22 ... sequences

It is often natural to index object names, and often we need to refer object names having consecutive index numbers or index letters. In **Jlp22** versions before version 3.0 it was possible to generate object lists using ... -construct which replaced part of the input line with the names of objects being between the object name before ... and after ... . Now the dots construct is no more done as part of the input programming but in the interpret subroutine which interprets the input line and generates the integer vector for function and argument indices. But as dots work as if it would be part of the input programming, it is presented in this section. Currently also sequences of integer constants can be generated with dots and sequences can be from larger to smaller.

Example 9.11 (dotsex). Example of dots construct

```
 dat=data(read->(x4...x7),in->)
1,2,3,4
11,12,13,14
/
stat(min->,max->,mean->)
x3%mean...x7%mean;
A...D=4...1;
Continue=1 !demo of error in data()
dat=data(read->(x3...x7),in->)
1,2,3,4
11,12,13,14
/
Continue=0
```

10 Jlp22 objects

Jlp22 objects have a simple yet efficient structure. Each object is associatd with two integer vectors, one single precision vector, one double precision vector, one vector of characters and one vector of text lines. All vectors are allocated dynamically. There are several object types which store data differently in these vectors. Object can be either simple or compound objects. Compound objects are linked to other objects which can be used also directly utilizing the standard naming conventions. All objects are global, i.e. also users can acces all objects. Some predefined objects are locked so that users cannot change them.

There are three types of objects

- Named objects. The number of named objects is specified at the initilaization, and it cannot be changed later. Deleting an object means deallocating all the allocated vectors associated with the objects.
- Temporary objects used to store intermediate results.
- Temporary objects used to store the partial derivatives when computing the derivatives using the derivations rules.

Alongside the objects there is a vector of double precision values, call j_v-vector. When an object is called a REAL object or variable, this vector is referenced. After the parts corresponding to named objects and those two sets of temporary objects, there is an area used to store numeric constants. If a REAL variable a has object number of 100 and constan 7 is in the position 7000 in the v-vector, then a+7 can be presented as j_v(100)+j_v(7000). Thus after parsing, all arithmetic computations can be done wihtou reference to whether a variable is in the named part or constant part of the j_v-vector. Because the numeric contants are at the end section of the j_v-vector, new space can be added for the numeric constants without mixing up parsed transformations.

10.1 Object names

Object names start with letter or with $#. Object names can contain any of symbols #%§/_ **Jlp22** is using '%' to name objects related to some other objects. E.g. function stat(x1,x2,mean->) will store means of variables x1 and x2 into variables x1%mean and x2%mean. Objects with name starting with '$' are not stored in the automatically created lists of input and output variables when defining transformation objects. The variable Result which is the output variable, if no output is given, is not put into these lists. Object names can contain special characters (e.g. +-*=()) if these are closed within '[' and ']', e.g. a[2+3]. This possibility to include additional information is borrowed from Markku Siitonen, the developer of Mela software. If an transformation

object is created with trans() function, and the intended global arguments are given in the list of arguments, then a local object ob created e.g. with transformation object tr have prefix tr/} yelding tr/. Actually also these objects are global, but their prefix protects them so that they do not intervene with objects having the same name in the calling transformation object. There are many objects intialized automatically. Some of these are locked so that the users cannot change them. In transformation objects there can be objects which are intended to be used only locally. These are protected by putting an unvisible prefix to the object names, but these can be anyhow accessed by writing the prefix. Names of objects having a predefined interpretation start with capital letter. The user can freely use lower or upper case letters. **Jlp22** is case sensitive. All objects known at a given point of a **Jlp22** session can be listed by command: print(Names)

10.2 Object types

The following description describes shortly different object types available in **J**. More detailed descriptions are given in connection of **Jlp22** functions which create the objects and in Developers' guide.

10.2.1 Real variables and constants

A REAL variable is a named object associated with a single double precision value. Before version J3.0 the values were in single precision, and thus this objecttype is still called REAL. The value can be directly defined at the command level, or the variable can get the value from data structures. E.g. stat(D,H,min->,max->) ! Here arguments must be variable names a = sin(2.4) ! argument is in radians sind() is for degrees h = data(read->(x1...x4)) ! x1, x2 ,x3, x4 are variables in the data set, and get their values when doing operations for the data.

Note 10.2.1. All objects have also an associated REAL value. In order to make arithmetic operations fast, the argument types in simple arithmetic functions are not checked. If a general object is used as an argument in an arithmetic operation, then the REAL value associated with the object is used. This will usually prevent the program to stop due to Fortran errors, but will produce unintended results.

Note 10.2.2. In this manual 'variable' refers to a **Jlp22** object whose type is REAL.

10.2.2 Character constants and variables

Character constants are generated by closing text within apostrophe signs ('). Apostrophe character (') within a character constant is indicated with (~) (if the character ~ is not present in the keyboard, it can be produced by <Alt >126, where numbers are entered from the numeric

keyboard) Character constants are used e.g. in I/O functions for file names, formats and to define text to be written. , e.g a = data(in->'file1.dat', read->(x1,x2))

write('output.txt', '(~kukkuu=~,4f7.0)', sqrt(a)) Character variables are pointers to character constants. An example of a character variable definition:

cvar='file1.dat' After defining a character variable, it can be used exactly as the character constants.

Note 10.2.3. The quotation mark (") has special meaning in the input programming. See Input programming how to use character constants within character constants.

10.2.3 Logical values

There is no special object type for logical variables. Results of logical operations are stored into temporary or named real variables so that 0 means False and 1 means True. In logical tests all non-zero values will mean True. Thus e.g. if(6)b=7 is legal statement, and variable b will get value 7. E.g. sit>h=a.lt.b.and.b.le.8 sit>print(h) h= 1.00000

10.3 Predefined objects

The following objects are generated during the initilaization.

Names	Text	Text object containg the names of named objects
Pi	REAL	The value of Pi (=3.1415926535897931)
$Cursor$	TRANS	The transformation object used to run sit> prompt
$Cursor2$	TRANS	Another transformation object used to run sit> prompt
Val	TRANS	Transformation object used to extract values of mathematical statements, used,
Round	REAL	jlp(): The current round through treatment units in jlp() function.
Change	REAL	jlp(): The change of objective in jlp() in one round before finding feasible and thereafter
Imp	REAL	jlp(): The number of improvements obtained from schedules outside the current active
$Data$	List	Default data set name for a new data set created by data()-function.
Obs	REAL	The default name of variable obtaining the the number of
Maxnamed	REAL	The maximum number of named objects. Determined via j.par in

Record	REAL	The name of variable obtaining the the number of
Subecord	REAL	The name of variable obtaining the the number of
Duplicate tions	REAL	A special variable used in data() function when duplicating observa-
LastaData set.	List	A list object referring to the last data set made, used as default data
$Buffer	Char	A special character object used by the write() function.
$Input$	Text	Text object used for original input line.
1$Input1$	Text	Text object for input line after removing blanks and comments.
Data	List	List object used to indicate current data setsDat
$ ing	REAL	Object name used to indicate console and '*' format in reading and writ-
x#	REAL	Variable used when drawing functions.
Selected	REAL	Variable used to indicate the simulator selected in simulations
Printinput	REAL	Variable used to specify how input lines are printed. Not properly used.
Prinoutpu used.	REAL	Variable used to indicate how much output is printed. Not properly
$Debug	REAL	Variable used to put debugging mode on.
Accepted	REAL	The number of accepted observations in functions using data sets.
Arg function.	REAL	The default argument name when using transformation object as a
Continue	REAL	If Continue has nonzero value then the control does not return to the
Err	REAL	If Continue prevents the control from returning to sit> prompt
Result	?	The default name of output object.

11 Jlp22 functions

The structure of **Jlp22** functions is easiest to explain using an example. For instance a data object can be created with
data2=newdata(matrx,matrz,read->(x1...x3,z1...z4),maketrans->mt)
Here data2 is the output, matrx and matrz are matrices having equal number of rows. Matrix matrx has 3 columns, matrz 4. read-> is an option which tells what are the variable names in the

data. x1...x3 is equivalent to x1,x2,x3. Option maketrans->mt tells that for each observation the trasformations defined in the trasformation object mt are computed from variables x1...x3 and z1...z4. The output variables whose names do not start with $ are included in the DATA object.

A function call obtained from the input programming can have the following components separed with commas.

11.1 Output of a function

There are the following cases with respect to the output of the function

- The function appears in a code line looking like
  Output=func()
  Then the function produces an object Output. The type of the Output depends on the function. Several functions produce additional objects whose name start with]Output%[. The object Output may or may not store links to these object. If such links are stored then the output is a compound object. An objects to which there is a link, is called a subobject. Objects without such links are called side objects. For instance linear programming function jlp() does not produce any object with name Output just several side objects.
- The expression
  func()
  is equivalent to
  Result=func()
- If code line is Output=func(); or func(); the Output or Result are printed if variable Printvalue has value 1 or 3. If code line is Output=func();; or func();; if variable Printvalue has value 2 or 3.
- Output is a temporal object. If a function is part of arithmetic computations, then the intermediate results are stored into temporary objects according to the parse tree. Thus in the expression y=sin(x)+cos(x), sin(x) produses first a temporary object and cos(x) another temporary object.
- Output is a submatrix of a MATRIX object. If A is a 4x4 matrix, and B is a3x3 MATRIX,then expression
  A(2,-4,1,-3)=B
  puts MATRIX B into A.

11.2 Function names

Function name can be be any of the four cases.

- A standard **Jlp22** function,
- An arithmetic or logical function obtained when translating the code into the polish notation.nslated first into Thus
  if(region.eq.savo)c=2*3+4;
  is translated into
  if(EQ(region,savo))c=PLUS(MULT(2,3),4);
  The user can try directly the above translated form
- An own-function of the user included in the function space.
- Implicit function generated with the parser. For instance, options are implemented using setoption() function.
- **Jlp22** object which can be used as if it were a function. Currently there are four such cases

11.2.1 MATRIX as a function

If matrixa is a MATRIX then expression matrixa() can be both in the input side or output side to indicate a submatrix or element of the matrix. See matrix chapter.

11.2.2 REAL as a MATRIX

Often in matrix computations, a REAL is a limiting value of a sequence of matrices. To facilitate such computations, matrixa(1) and matrixa(1,1) are legal ways to refer to matrixa even if it is REAL. If matrixa is REAL, then expression matrixa(3) is causing error:
matrixa is REAL, ranges 3 are illegal, only (1) or (1,1) are allowed

11.2.3 DATA as a function

If Data is DATA then Data(var1,..,varn) indicates a matrix obtained by picking the columns of the data matrix corresponding the argument variables.

11.2.4 Using a transformation object as a function

It is now possible to use a transformation object as a function which computes new objects when generating arguments for functions or options, or values of code options, or in any place within a transformation object. If tr is a transformation and the transformation computes an object A then tr(A) is first calling transformation tr and provides then object A into this place. As the transformation computes also other objects which are computed within it, also thes objects are available. At this point it is important to note that arguments of a transformation line are computed from right to left, because options must be computed before entering into a function.

Example 11.1 (transfunc). Transformation as a function

```
 delete_o(a,c)
transa=trans()
a=8;
c=2;
/
transb=trans()
a=5;
c=1;
/
c=2
a=c+transb(a)+c+transa(a);
```

11.2.5 REGR as a function

If Regr is a regression object produced with regr, then Regr() produces the value of the regression function as explained for regr(). The values of arguments can be given in the calling code or the existing values can be used.

11.2.6 Cannot produce -error

If the code looks like a function, the control goes to j_getelem function which computes the results for the above mentioned object types. But if the object cannot produce anything, there will be an error message:
produce but it is not a function or object which can provide something then there will be an error message:
* (object name) cannot produce anything

11.2.7 is REAL ranges .. illegal -error

If the object is REAL then the error message can be
matrixa is REAL, ranges 5 are illegal, only (1) or (1,1) are allowed
A REAL can be treated as a 1x1 MATRIX, so for REAL var var(2,2) means illegal dimensions.

11.3 Arguments of a function

Arguments separated with commas. Arguments can be a combination of the following types

- object name
- numeric constant
- A code which produces a temporary object generated with any **Jlp22** functions

- a sequence of object names generated with ... as x1...x3 above.
- @List where List is a LIST. If List is obtainedwit List=list(x1...x3), then @List is equivalent to x1...x3 which is equivalnet to x1,x2,x3.

a = sin(cos(c)+b) ! Usual arithmetic functions have numeric values as arguments. here the value of the argument of cos is obtained by 'computing' the value of real variable c. stat(D,H,min->,max->) ! Here arguments must be variable names plotyx(H,D,xrange->(int(D%min,5), ceiling(D%max,5)) !arguments of the function are variables, arguments of option xrange-> are numeric values c = inverse(h+t(g)) ! The argument can be intermediate result from matrix computations. If it is evident if a function or option should have object names or values as their arguments, it is not indicated with a special notation. If the difference is emphasized, then the values are indicated by val1,...,valn, and objects by obj1,...,objn, or the names of real variables are indicated by var1,...,varn. There are some special options which do not refer to object names or values. Some options define a small one-statement transformation to be used to compute something repeatedly.

11.4 Options of a function

Options give additional arguments to the function.

11.4.1 Option structure

An option starts with the option name followed with '->' after which there can be

- Nothing. In this case the option indicated that the option is put 'on'. E.g. continue-> in a graphics function indicates that no pause() is generated after plotting the figure.
- Numeric value, e.g. continue->fcont in a graphics function indicates that no pause() is generated if a REAL object fcont has a nonzero value.
- Object name.
- Function name followed by the arguments within parenthesis, e.g., xrange->(0,ask('xmax')) where ask() is the function which asks a numeric value from the user.
- Arguments of the option expressed in the same way as the arguments of the function, i.e., using acode which is producing a temporary object.

11.4.2 Codeoptions

Options are grouped into two groups: code options and regular options. Code options define a small one-statement transformation to be used to compute something repeatedly. As these one-statement can use transformation objects as functions, the code option can actually execute long computations. For instance
stat(D,H,filter->(region.eq.savo))
only those observations are accepted which pass the filter.
draw(func->(sin($x)+1),x->$x,xrange->(0,10,1),continue->fcont)
func-> option transmits the function to be drawn not a single value.

11.4.3 Common options

There are some options which are used in many **Jlp22** functions. Such options are e.g.

---

in 0|1- Char

Indicates from where the data are read in. If there are no arguments, then the data are in the following input paragraph. If the values are character constants or a character variables, then data are read in from files having those names.

---

data N|1- DATA

DATA object. Default last DATA defined.

---

from N|1 REAL

First observation used.

---

until N|1 REAL

Last observation used.

---

trans N|1 TRANS

TRANS computed fro each observation

---

epilog N|1 TRANS

TRANS computed after going through data. All data sets will be treated logically as a single data set. If the function is using data sets, the daenta sets are given in data-> option. All data sets will be treated logically as a single data set. If a **Jlp22** function needs to access data, and the data-> option is not given then **Jlp22** uses default data which is determined as follows. If the user has defined

	an object list Data consisting of one or more data sets, then these will be used as the default data set. E.g. Data=list(dataa,datab) When a data set is created, it will automatically become the only element in LastData list. If the Data list has not been defined and there is no data-> option, then the LastData dataset will be used.	
trans	-1 | 1	TRANS
	transformation set which is executed for each observation. If there is a transformation set associated with the data set, those transformations are computed first. In all functions which are using data sets, trans-> option defines a transformation set which is used in this function.	
filter	-1 | 1	Code
	logical or arithmetic statement (nonzero value indicating True) describing which observations will be accepted. trans-> transformations are computed before using filter.	

Example 11.2 (comoptex). data1

```
 dat=data(read->(x,y),in->)
1,2
3,4
5,6
/
transa=trans()
xy=x*y
x,y,xy;
/
stat(trans->transa)
```

12 Arithmetic and logical functions and operations

Jlp22 has all the standard arithmetic and logical operations and functions. The arithmetic and logical functions return single REAL value or a MATRIX.

12.1 Functions sqrt(), sqrt2(), exp(), log(), log10(), abs()

!

- sqrt(x) square root, sqrt(0) is defined to be 0, negative argument produce error. If x is matrix, then an error occurs if any elemet is negative.

- sqrt2(x) If x or an element of x is negative then $sqrt2()=-sqrt()$. Actually sqrt2() might be a useful sigmoidal function in modeling context.
- exp(x) e to power x. If x>88, then **Jlp22** produces error in order to avoid system crash.
- log(x) natural logarithm
- log10(x) base 10 logarithm
- abs(x) absolute value

12.2 Trig sin(),sind(),cos(),cosd(),tan(),tand(),cotan(),cotand()

In sin(), etc, argument is in radians, in sind(), etc in degrees

- sin()
- sind()
- cos()
- cosd()
- tan()
- tand()
- cotan()
- cotand()

12.3 arc-trig asin() asind() acos() acosd() atan() atand() acotan() acotand()

Inverse trigonometric functions. In asin(), etc, argument is in radians, in asind(), etc in degrees

- asin()
- asind()
- acos()
- acosd()
- atan()
- atand()
- acotan()
- acotand()

12.4 Hyperbolic sinh() cosh() tanh()

- sinh()
- cosh()
- tanh()

12.5 Functions mod() fraction()

- mod()
- fraction()

12.6 Conversion to integer

- nint(x) nearest integer value
- nint(x,modulo) returns modulo*nint(x/modulo) ,e.g.nint(48,5)=50; nint(47,5)=45;
- int(x) integer value obtained by truncation
- int(x,modulo) returns modulo*int(x/modulo), e.g. int(48,5)=45
- ceiling(x) smallest integer greater than or equal to x.
- ceiling(x,modulo) returns modulo*ceiling(x/modulo), e.g. ceiling(47,5)=50.
- floor(x) greatest integer smaller than or equal to x.
- floor(x,modulo) returns modulo*floor(x/modulo), e.g. floor(47,5)=45.

12.7 min() and max()

Functions min() and max () behave in a special way, max() behaves similarly as min() here:

- min(x1,x2):: minimum of two REAL
- min(MATRIX,REAL):: each element is min(elem,REAL)
- min(MATRIX):: row vector having minimums of all columns
- min(MATRIX,any->):: minimum over the whole amtrix

12.8 sign(A,B)

sign(A,B) returns the value of A with the sign of B.

12.9 Numeric operations -, +, *, **, ***, /, *., /.

An arithmetic expression consisting of ordinary arith metic operations is formed in the standard way. The operations are in the order of their precedence.

- - unary minus
- *** integer power
- ** or r̂eal power
- * multiplication
- / division
- + addition
- - subtraction

The reason for having a different integer power is that it is faster to compute and a negative value can have an integer power but not a real power.

In matrix computations there are two additional operations.

- *. elementwise product (Hadamard product)
- /. elementwise division

The matrix operations are explained in section Arithmetic and logical operations. Their operation rules extent the standard rules.

12.10 Logic and relation .eq. .ne. .gt. .ge. .lt. .le. .not. .and. .or. .eqv. .neqv.

There are following relational and logical operations. The first alternatives follow Fortan style:

Note 12.10.1. Testing equivalence can be done also using 'equal to' and 'not equal', as the same truth value is expressed with the same numeric value.

Note 12.10.2. when the truth value of an expression is tested with if(), then all nonzero real values means that the expression is true.

13 Operation of Jlp22

In this section the main structure of the operation of **Jlp22** and tools in code development in a project are presented. The two operation modes are interactive operation and batch operation.

13.1 Looping between j_getinput, j_parser and dotrans

The are three key subroutines in **Jlp22** operation.

- j_getinput gets a new command line from the console, an include file. j_getinput can read several records to make one command line, and it can edit or duplicate the input with special **input programming** tools. Jlp prints sit> prompt when reading form the console. Input programming lines are indicated with initila %.
- If the obtained input line is not an input programming command or if it is not captured by some function which is utilizing j_getinput, the line goes to j_parser which generates an integer vector.
- The integer vector produced by the j_parser is interpred ('executed' in this manual) with dotrans subroutine.
- trans() function packs several lines generated in j_getinput - j _parser loops into a tranformation (TRANS) object, which can be called from different functions.
- dotrans subroutine executes a single line command which is not part of an TRANS object. **Jlp22** writes '#' in te front of the line.

13.2 Printing of Jlp22

Jlp22 prints information of the proceeding of the control. There are different ways to control the output. Finetuning is needed in the output. There are three logical stages in the flow of the control. The output indicates these as follows.

13.2.1 % -lines are printed from input programming

The input programming lines are printed starting with %. The amount of printing is controlled with Printinput variable. the dafault is Printinput=1. A problem in printing the input is how much should be printed when many input lines are produced with ;do()

13.2.2 # -lines are printed in one-line commands.

The input lines which go directly to execution are printed starting with #.

13.2.3 Comments starting with !!, !*,** or *! are printed

The first non blank character in a command line is '!' or '*'. If there are two consecutive comment characters, the line is printed, dropping the first character.

Note 13.2.1. There can be comment lines between continuation lines.

13.2.4 Printoutput guides printing of functions.

The amount of printing in functions is guided with with variable [Printoutput[. The default is

In the interactive operation, the processing of sripts can be interrupted with ;pause, and execution of TRANS can be intterrupted with pause(). Even if they are described in 'Command input and output' and '**Jlp22** transformations' chapters, descriptions are repeated here as they are useful in the interactive operation of **Jlp22**.

13.2.5 ;pause in script processing

Including input from an include file can be interrupted using an input programming command ;pause promt or the **Jlp22** function pause('<prompt>'). In both cases the user can give **Jlp22** commands, e.g., print objects, change the value of Printdebug etc. The difference is that pause('<prompt goes first through the interpreted and the interptreted code is transmitted to the **Jlp22** function driver. In the ;pause- pause it is possible to use input programming commands while in pause()-pause it is not possible. In both cases, when an error occurs, the control remains at the pause prompt. If the user is pressing <return> **Jlp22** continues in the include file. If pause() is part of a transformation object, pressing <return>, the function driver continues in the transformation object. If the user gives command 'e' or 'end', then **Jlp22** procees similarly as if an error had occured, i.e. print error messages and returns control to sit> -promt.

13.2.6 pause() in a TRANS or in command input

Function pause() stops the execution of **Jlp22** commands which are either in an include file or in TRANS object. The user can give any commands during the pause() except input programming commands. If the user presses <return> the excution continues. If the user gives 'e', then an error condition is generated, and **Jlp22** comes to the sit> promt, except if Continue has values 1, in which case the control returns one level above the sit> promt. If pause() has a CHAR argument, then that character constant is used as the prompt.

Note 13.2.2. When reading commands from an include file, the a pause can be generated also with ;pause, which works similarly as pause(), but during ;pause also input programming commands can be given.

13.3 Batch mode

If the default main program is replaced with a program which tells that Jlp is operated in the batch mode, or if the initial ;incl -file containscommand batch(), the **Jlp22** is operated in the batch mode. In the batch mode, the control never starts to read commands from the console at sit> prompt. Using **Jlp22** from R, developed with Lauri Mehtätalo is using **Jlp22** in the batch mode.

13.4 Using shortcuts and sections in ;incl files

It is useful to organize the project sript into one script file, which conatians swections starting with a lable and ending with ;return. I think that it is more difficult to have several script files. The example file jexamples.inc is a good example of a script file. different versions of the same script file are stored in different names, it is useful to have as first line something like this=thisfile()
Then it is not necessary to change anything if the file is stored in a different name. Thereafter comes the shortcut definitions for different sections. the same For instance, if the section label is
;thistask:
then the shortcut definition could be thistaskh=';incl(this,from->thistask)'

The section should end with
;return
After defining shortcuts for all sections, it is useful to have a shortcut as:
again=;incl(this)
If new sections are added, then one needs to give just shortcut
again
and then the new shortcuts will be defined. It does not matter if the earlier shortcuts are redefined.

The last shortcut could be
current=';incl(this,from->current) The label 'current' can be a floating label which is put into the section which is under developed in place the problems started.

If a **Jlp22** code line, either in the input paragraph defining a transformation object or outside it, ends with ';' or ';;', the output object of the code line may be printed. The output of ';'-line is printed if the variable Printoutput has value 1 or 3 at the time when the code line is computed. The output of ';;'-line is printed if the variable Printoutput has value 2 or 3 at the time when the code line is computed.

If a code line within a tranformation has function pause('text'), then a pause is generated

during which the user can give any commands except input programming commands. If the user will press <return> then the exceution continues. If the user presses 'e' and <return>, the control comes to the sit> promt similarly as during an error.

If the line outside the transformation definition paragraph is ';pause', then a similar pause is generated except also input programming commands can be give.

If the variable Debugtrans has value 1, them a pause() is genertated before each line within a tranformation object is executed. If variable Debugconsole has value 1, a ';pause' is generated before the line is excuted. In bot cases the user can give new values for Debugtrans and Debugconsole.

What happens when an error is encountered is dependent on the value of variable Continue. If Continue has value 0 (the default case), the control comes into sit> prompt when an real error occurs or if an atrficial error condition is generated with errexit(). If Continue has value 1 then the computation continues in the same script file where the error occured. This property is used in file jexamples.inc to demonstrate possible error conditions so that the computation continues as if no error had occured.

13.5 end ends Jlp22

To exit **Jlp22** program and close console window, just give end command:

sit>end

14 Functions for handling objects

The following functions can handle objects.

14.1 Copying object: a=b

A copy of object can be made by the assignment statement a=b.

14.2 type() Type of an object or all available types

The type of any object can be access by type(object). If the argument is a character variable or character constant referring to a character constant, and there is content-> option, and the character is the name of an object , the type() returns the type of the object having the name given in the argument. If there is no object having that name, then type() returns -1 and no error is generated.

Example 14.1 (typeex). Example of type

```
ttt=8; !REAL
```

```
type(ttt);
type('ttt');!type is CHAR
type('ttt',content->);
cttt='ttt'
type(cttt);
type(cttt,content->);
```

14.3 delete_o() Deletes objects

The function delete_o() deletes all the argument objects, which means that the associated allocated vectors are deallocated, and the object will be REAL.

Note 14.3.1. Note that delete_o() is actually needed only for matrices, because objects can be deleted with Object=0. When Object is a matrix, then Object=0 puts all elements into zero.

Example 14.2 (deleteoex). `Delete object`

```
 a=matrix(2,3,do->);
b=t(a);
delete_o(a,b)
a,b;
```

14.4 exist_o(): does an object exist

looks whether an object with the name given in the character constant argument exists.

Note 14.4.1. In the previous versions of **Jlp22** same function was used for files and objects.

14.5 name(): writes the name of an object

The argument gives the index of the object. This function may useful if **J** prints in problem cases the object indices.

15 Jlp22 transformations

Most operation commands affecting **Jlp22** objects can be entered directly at the command level or packed into transformation object. In both cases the syntax and working is the same. A command line can define arithmetic operations for real variables or matrices, or they can include functions which operate on other **Jlp22** objects. General **Jlp22** functions can have arithmetic statements in their arguments or in the option values. In some cases the arguments must be object names. In principle it is possible to combine several general **Jlp22** functions in the same

operation command line, but there may not be any useful applications yet, and possibly some error conditions would be generated. Definition: A numeric function is a **Jlp22** function which returns a single real value. These functions can be used within other transformations similarly as ordinary arithmetic functions. E.g. weights() is a numeric function returning the number of schedules having nonzero weight in a JLP-solution. Then print(sqrt(weights())+Pi) is a legal transformation.

15.1 Transformation object

A transformation object groups several operation commands together so that they can be used for different purposes by **Jlp22** functions and **Jlp22** objects. A transformation object contains the interpreted transformations. For more details see **Jlp22** function for defining transformation objects: trans(). Transformation objects can be called using call() function, so that all transformations defined in the object are done once. Function result() also calls transformations but is also returning a value. When transformation objects are linked to data objects, then the transformations defined in trans-formation object are done separately for each observation. There is an implicit transformation object $Cursor$ which is used to run the command level. The name $Cursor$ may appear in error messages when doing commands at command level. An-other transformation object Val which is used to take care of the substitutions of "-sequences in the input programming. Some **Jlp22** functions use also implicitly transformations object $Cursor2$.

15.2 trans() Creates a TRANS (transformation) object

trans() function interprets lines from input paragraph following the trans() command and puts the interpreted code into an integer vector, which can be excuted in several places. If there are no arguments in the function, the all objected used within the transforamations are global. This may cause conflicts if there are several recursive functions operating at the same time with same objects. **Jlp22** checks some of these conflict situations, but not all. These conflicts can be avoided by giving intended global arguments in the list of arguments. Then an object 'ob' created e.g. with transformation object tr have prefix]tr/

yelding]*tr*/[*.Actuallyalsotheseobjectsareglobal,buttheirprefixprotectsthemsothattheydonotintervenewith*

Each line in the input paragraph is read and interpreted and packed into a transformation object, and associated tr%input and tr%output lists are created for input and output variables. Objects can be in both lists. Objects having names starting with '$' are not put into the input or output lists. The source code is saved in a text object tr%source. List tr%arg contains all arguments. ! If a semicolon ';' is at the end of an input line, then the output is printed if REAL

variable Prindebug has value 1 or value>2 at the execution time. If the double semicolon ';;' is at the end then the output is printed if printresult>1. If there is no output, but just list of objects, then these objects will be printed with semicolns.

!

Output	1	Data
	The TRANS object generated.	
Args	N|1-	
	Global objects.	

Note 15.2.1. Options input->, local->, matrix->, arg->, result->, source-> of previous versions are obsolte.

Note 15.2.2. The user can intervene the execution from console if the code calls read($,), ask(), askc() or pause() functions. During the pause one can give any command excepts such input programming command as ;incl.

Note 15.2.3. The value of printresult can be changed in other parts of the transformation, or in other transforamations called or during execution of pause().

Note 15.2.4. Output variables in maketrans-> transformations whose name start with $ are not put into the new data object.

Example 15.1 (transex). Demonstrates also error handling

```
 transa=trans()
$x3=x1+3
x2=2/$x3;
/
transa%input,transa%output,transa%source;
x1=8
call(transa)
transb=trans(x1,x2)
$x3=x1+3
x2=2/$x3;
x3=x1+x2+$x3;
/
transb%input,transb%output,transb%source;
call(transb)
transb|x3; !x3 is now local
transc=trans()
x1=-3
call(transb)!this is causing division by zero
/
Continue=1 !  continue after error
```

```
call(transc)
```

```
sit>transex
<;incl(exfile,from->transex)


<transa=trans()
<$x3=x1+3
<x2=2|$x3;
</
<transa%input,transa%output,transa%source;
transa%input is list with            2  elements:
x1 $x3
transa%output is list with            1  elements:
x2
transa%source is text object:
1 $x3=x1+3
2 x2=2/$x3;
3 /
///end of text object

<x1=8


<call(transa)
x2=0.18181818

<transb=trans(x1,x2)
<$x3=x1+3
<x2=2/$x3;
<x3=x1+x2+$x3;
</
<transb%?;


<call(transb)
```

```
<transc=trans()
<x1=-3
<call(transb)
</
<Continue=1
<call(transc)
*division by zero
*****error on row            2  in tr%source
x2=2/$x3;
recursion level set to    3.0000000000000000

*****error on row            2  in transc%source
call(transa)
recursion level set to    2.0000000000000000

*err* transformation set=$Cursor$
recursion level set to    1.0000000000000000
****cleaned input
call(transc)
*Continue even if error has occured
<;return
```

15.3 call() executes TRANS object

Parsed transformations in a TRANSobject can be automatically executed by other **Jlp22** functions or they can be executed explicitly using call() function.

Arg	1	TRANS
	The transformation object executed.	

Note 15.3.1. A transformation objects can be used recursively, i.e. a transformation can be called from itself. The depth of recursion is not controlled by **J**, so going too deep in recursion will eventually lead to a system error.

Note 15.3.2. Professional programmers would probably say that the integer vector produced with the parser is interpreted and not executed. An amateur programmer can be more flexible with terms.

Example 15.2 (recursion). `Recursion produces system crash.`

```
  transa=trans()!level will be initialized as zero
level;
level=level+1
call(transa)
/
Continue=1 !error is produced
call(transa)
Continue=0
```

15.4 pause() in a TRANS or in command input

Function pause() stops the execution of **Jlp22** commands which are either in an include file or in TRANS object. The user can give any commands during the pause() except input programming commands. If the user presses <return> the excution continues. If the user gives 'e', then an error condition is generated, and **Jlp22** comes to the sit> promt, except if Continue has values 1, in which case the control returns one level above the sit> promt. If pause() has a CHAR argument, then that character constant is used as the prompt.

Note 15.4.1. When reading commands from an include file, the a pause can be generated also with ;pause, which works similarly as pause(), but during ;pause also input programming commands can be given.

16 Special implicit functions

The special functions are such that the parser uses these functions for special operations. Only list2() is function which also the user can use, but the parser is is using it implicitly.

16.1 setoption(): set option on

When a function has an option then the parser generates first code setoption(...) where the arguments of the option are interpred in the similar way as arguments of all functions. Then the parser genarates the code for setoption() function in a special way.

16.2 Get or set a matrix element or submatrices

Matrix elements or submatrices can be accessed using the same syntax as accesing **Jlp22** functions.

One can get or set matrix elements and submatrices as follows. If the expression is on the right side of '=' then **Jlp22** gets a REAL value or submatrix, if the expression is on the left side of

'=', the **Jlp22** sets new values for a matrix element or a submatrix. In the following formulas C is a column vector, R a row vector, and M is a general matrix with m rows and n columns. If C is actually REAL it can be used as if it would 1 x 1 MATRIX. This can be useful when working with matrices whose dimensions can vary starting from 1 x 1. Symbol r refers to row index, r1 to first row in a row range, r2 to the last row. The rows and columns can be specifiel using ILIST objects il1 and il2 to specify noncontiguous ranges. It is currently not possible to mix ILIST range and contiguous range, so if ILIST is needed for rows (columns), it must be used also for columns (rows). ILIST can be specified using ilist() function or using construction. Similarly columns are indicated with c. It is always legal to refer to vectors by using the M formulation and giving c with value 1 for column vectors and and r with value 1 for row vectors.

Args	0-4	REAL | ILIST row and column range as explained below.
diag	N| Get or set diagonal elements	
sum	N|0|1 When setting elements, the right side is added to the current elements. If the sum-> option has argument, the right side is multiplied with the argument when adding to the curretn elements.	

Row and column ranges can be spcidie as follows.

- M(r,c) Get or set single element.
- C(r) Get or set single element in column vector.
- R(c) Get or set single element in column vector.
- M(RANGES) Get or set a submatrix, where RANGES can be. For a column vector, the column range need not to specified.
  - r1,-r2,c1,-c2
  - r,c1,-c2 part of row r
  - r1,-r2,c part of column c
  - r1,-r2,All All columns of the row range
  - All,c1,-c2 All rows of the column range
  - r1,....rm,c1,...,cn Given rows and columns

- r1,....rm Given rows for column vector
- il1,il2 for matrix with several columns
- il1 for column vector

- When r2= m, then -r2 can be replaced with Tolast.
- When c2= n, then -c2 can be replaced with Tolast.

If option diag-> is present then

- M(diag->) Get or set the diagonal. If M is not square matrix, and error occurs.
- M(r1,-r2,diag->) (Again -r2 can be Tolast.

Note 16.2.1. Note When setting values to a submatrix the the the values given in input matrix are put into the outputmatrix in row order, and the shape of the input and output matrices need not be the same. An error occurs only if input and output contain different number of elements.

Example 16.1 (getset). Get or set submatrices

```
 A=matrix(3,4,do->);
B=A(1,-2,3,-4);
A(1,-2,3,-4)=B+3;
A(1,-2,3,-4,sum->)=-5;
A(1,-2,3,-4,sum->2)=A(2,-3,1,-2);
C=A(1,3,4...2);
H=matrix(4,4,diag->,do->3);
H(3,-4,diag->)=matrix(2,values->(4,7));
```

Note 16.2.2. When giving range, the lower and upper limit can be equal.

16.3 getelem(): extracting information from an object

The origin of this function is the function which was used in previous versions to take an matrix element, which explains the name. Now it is used to extract also submatrices (e.g. a(1,-3,All)) , or to get value of an regression function or to compute a transformation and then take argument object as the result. E.g. if tr is a transformation then the result of tr(a) is object a after calling tr.

Note 16.3.1. If someone starts to use the own function property of the open source **J**, she/he probably would like to get the possibility to extract information from her/his object types also. To implement this property requires some co-operation from my side.

16.4 setelem(): Putting something into an object.

The origin of this function is the function which was used in previous versions to set an matrix element, which explains the name. Now it is used to replace values of submatrices when submatrix expression is on the output side (e.g. a(1,-3,All)=..).

Note 16.4.1. If someone starts to use the own function property of the open source **J**, she/he probably would like to get the possibility to put information into from her/his object types also. To implement this property requires some co-operation from my side.

Note 16.4.2. In effect the getelem() and setelem() functions are excuted in the same getelem() subroutine, because bot functions can utilize the same code.

16.5 list2()

The interpreted utilizes this to separate when spearating output and input objects. See Section **??** hw user can use this function

16.6 setcodeopt(): Initialization of a code option

This function initializes an code option for a function which has the option.

16.7 o1_funcs(), o2_funcs() and o3_funcs() calls own functions

The users of open **Jlp22** can define their own functions using three available own-function sets. In addition to own functions open **Jlp22** is ready to recognize also own object types and options which are defined in the source files controlled by the users. The main **Jlp22** does not know what to do with these own object types and options, they are just transmitted to the own-functions. In the main **Jlp22** the control is transmitted to the own functions using implicit **Jlp22** functions o1_funcs(), o2_funcs() and o3_funcs().

17 Transformation objects

The code lines generated by the input programming can be either executed directly after interpretation, or the interpreted code lines are packed into a transformation object, which can be excuted with call() which is either in the code generated with the input programming or inside the same or other transformation object. Recursive calling a transformation is thus also possible. Different functions related to transformation object are described in this section.

18 Loops and control strucures

This section describes nonstadard functions.

18.1 do() loops

The loop construction in **Jlp22** looks as follows: do(i,start,end[,step]) enddo

Note 18.1.1. cycle and exit are implemented in the current **Jlp22** version with goto() Within a do–loop there can be cycleand exitdostatements

Note 18.1.2. There can be 8 nested loops. do-loop is not allowed at command level.

Example 18.1 (doex). do-loop

```
 !!begin
transa=trans()
do(i,1,5)
i;
ad1:if(i.eq.3)goto(cycle)
i;
if(i.eq.4)goto(jump)
cycle:enddo
jump:i;
/
call(transa)
```

18.2 if()

if()j_statement...
The one line if-statement.

18.3 if() elseif() else endif

There can be 4 nested if()then structures. If-then-structures are not allowed at command level.

if()then .... elseif()then ... else .... endif

18.4 output=input

There are two assignment functions generated by '=', when the line is of form output=func[]input[), then the output is directly put to the output position of the function without explicitly generating assignment. When the codeline is in form output=input then the following cases can occur

- output is MATRIX and input is scalar, then each element of MATRIX is replaced with the input in assone() function.
- output is submatrix expression, then the elements of the submatrix are asigned in setelem() funtion whether input is MATRIX or submatrix expression, scalar or LIST.
- output is MATRIX and on input side is a random number generation function, the random numbers are put to all elements of the matrix.
- If on output side are many object names, and input side is one REAL value, this is put to all variables.
- If on output side are many object names, and input there are several variables then both sides should have equal numbers of object names, then then copies of the input objects are put into output objects.

Example 18.2 (assignex). Examples of assignments

```
 a=matrix(2,3);
a=4;
a=rann();
v1...v5=2...6;
v1...v5=77;
Continue=1 !  ERROR
v1..v3=1,5;
v1..3=1...3 !  v is missing from the front of 3
Continue=0
```

18.5 which() Value based on conditions

Usage:

output=which(condition1,value1,...,conditionn,valuen)
or
output=which(condition1,value1,...,conditionn,valuen,valuedefault) Where conditionx is a REAL value, nonzero value indicating TRUE. Output will get first value for which the condition is TRUE. When the number of arguments is not even, the the last value is the default value.

Example 18.3 (whichex). Example of which()

```
 c=9
which(a.eq.3.or.c.gt.8,5,a.eq.7,55);
a=7
which(a.eq.3.or.c.gt.8,5,a.eq.7,55);
a=5
which(a.eq.3.or.c.gt.8,5,a.eq.7,55);
```

```
which(a.eq.3.or.c.gt.8,5,a.eq.7,55,108);
```

18.6 errexit() returns to sit>

Function errexit() returns the control to sit> prompt with a message similarly as when an error occurs.

Example 18.4 (errexitex). errexit() returns as if error had occured

```
 transa=trans()
if(a.eq.0)errexit('illegal value',a)
s=3/a;!  division with zero is tested automatically
/
a=3.7
call(transa)
transa(s);!tr can also be used as a function
a=0
Continue=1 !Do not stop in this selfmade error
call(transa)
Continue=0
```

18.7 goto() goto a label in TRANS

Control can be transfered to a label in a transformation set with goto(). There are two types of goto's, unconditional goto to a given label and goto to an label from a group af labels based on a condition (Computed goto in Fortran). These are decribed in separate subsections. Notes common to both are presented here.

Note 18.7.1. It is not recommended to use goto() according to modern computation practices. However, it was easier to implement cycle and exitdo with goto().

Note 18.7.2. It is not allowed to jump in to a loop or into if -then structure. This is checked already in in the parser. DOES NOT WORK NOW, WILL BE CORRECTED

Note 18.7.3. Even if the labels are logically character arguments, they are not treated using CHARs. The parser handles them otherwise.

Note 18.7.4. The label lines can contain code but the labels can stand on the line also alone.

18.8 goto(label) Unconditional goto

An unconditional goto() has only one label argument.

Example 18.5 (gotoex). Example of unconditional goto

```
 transa=trans()
i=0
if(i.eq.0)goto(koe)
out=99;
koe:out=88;
/
call(transa)
out;
```

18.9 goto(index,lable1...labeln) Conditional goto

An conditional goto() selects the label from a group of labels.

Example 18.6 (congotoex). transa=trans()

```
out=999
goto(ad1)
77;
ad1:
1;
goto(2,ad1,ad3)
88;
ad2:
2;
goto(go,ad1,ad3)
out=0;
return
ad3:
3;
goto(3,ad3,ad1,ad2)
/
go=0 !  This determines the last goto
call(transa)
out;
go=4
** Now error occurs
Continue=1
call(transa)
Continue=0
;return
```

Note 18.9.1. A simulator for generating treatment schedules for forest stands can be nicely defined using the conditional goto, as will shortly be described.

19 Arithmetic and logical operations

The logical operations follow the same ruels as addition +. The following rules, extending the standard matrix computaion rules apply. The same rules aplly if the order of arguments is changed,

- MATRIX + REAL : REAL is added to each element
- MATRIX1+MATRIX2 :: elementwise addition, if matrices hav comptaible dimensions
- MATRIX+ column vector: column vector is added to each column of MATRIX if the numbers of rows agrees.
- MATRIX+ row vector: row vector is added to each row of MATRIX if the numbers of columns agree.

The same rules apply for the lelmentwise multiplication *. and elementwise division /. as for addition +.

19.1 min() and max()

Functions min() and max () behave in a special way, max() behaves similarly as min() here:

- min(x1,x2):: minimum of two REAL
- min(MATRIX,REAL):: each element is min(elem,REAL)
- min(MATRIX):: row vector having minimums of all columns
- min(MATRIX,any->):: minimum over the whole amtrix

20 Statistical functions for matrices

Functions mean(), sd(), var(), sum(), min() and max() can be used used to compute stastics from a matrix. Let mean() here present any of thes functions. The following rules apply:

mean(VECTOR) computes the mean of the vector, output is REAL mean(MATRIX) computes the mean of the each column. Result is row vector. mean(VECTOR,weight->wvector) computes the weigted mean of the vector,weights being in vector wvector. mean(MATRIX,weight->wvector) computes the weighted mean of each column, weights being in vector wvector, result is row vector.

20.1 mean() Means or weighted means

See section matrixstat for details

20.2 sd() Sd's or weighted sd's

See section matrixstat for details

20.3 var() Sample variances or weighted variances

See section matrixstat for details

20.4 sum() Sums or weighted sums

See section matrixstat for details

21 Derivatives der()

Derivates of a function with respect to any of its arguments can be computed using the derivation rules by using der() function in the previous line. The funcion must be expressed with one-line statement. The function can call other functions using the standard way to obtain objects from transformations, but these functions cannot contain variables for which derivatives are obtained. Nonlinear regression needs the derivatives with respect to the parameters.
der() function does not have an explicit output, but der() accompanied with the function produces REAL variable for each of the argument variables of der(). If the output of the next line is f and der() has argument a, then f\a will get the value of the derivative of f with respect to a.

Note 21.0.1. Derivative are now utlized in nonliear regression, but they will also be utilized in nonlinear programming.

Example 21.1 (derex). Derivatives with der()

```
 transa=trans()
der(x)
f=(1+x)*cos(x)
/
fi=draw(func->transa(f\x),x->x,xrange->(0,10),color->Blue,continue->)
fi=draw(func->transa(f),x->x,xrange->(0,10),color->Cyan,append->,continue->fc
```

Example 21.2 (derex2). 2

```
 X=matrix(do->(0,1000,10))
e=matrix(nrows(X))
```

```
e=rann(0,2);
A,Pmax,Res=0.1,20,2
A*Pmax*1000/(A*1000+Pmax);
Y=A*Pmax*X/.(A*X+Pmax)-Res+e !rectangular hyperbola used often for photosynthe


  transa=trans()
der(A,Pmax,Res)
f=A*Pmax*I/(A*I+Pmax)-Res
/


  fi=draw(func->(transa(f)),x->I,xrange->(0,1000),color->Orange,width->2,conti
da=newdata(X,Y,e,extra->(Regf,Resid),read->(I,P,er))
stat()
fi=plotyx(P,I,append->,show->0,continue->fcont)


  A,Pmax,Res=0.07,17,3 !initial values


  fi=draw(func->(transa(f)),x->I,xrange->(0,1000),color->Green,width->2,append
reg=nonlin(P,f,par->(A,Pmax,Res),var->,corr->,data->da,trans->transa)
reg%var;
reg%corr;
corrmatrix(reg%var);
fi=draw(func->(transa(f)),x->I,xrange->(0,1000),color->Violet,append->,contin
```

---

22 Special arithmetic functions

Jlp22 has the following arithmetic functions producing REAL values.These functions cannot yet have matrix arguments.

22.1 gamma() Gamma function

Function gamma() produses the value of gamma funtion for a positive argument. The function utilises gamma subroutine from library dcdflib in Netlib. For computing gamma function for a product, loggamma() is often needed. loggamma() Log of gamma function Function gamma() produces the value of loggamma funtion for a positive argument. ! The function utilises gamma subroutine from library dcdflib in Netlib. Loggamma is used in statistics in many cases where gamma function gets a too large value to be presented in double precision.

22.2 logistic() Logistic function

Returns the value of the logistic function 1/(1+exp(-x)). This can in principle computed by the transformation, but the transformation will produce an error condition when the argument -x of

the exp-function is large. Because the logistic function is symmetric, these cases are computed as exp(x)/(1+exp(x)). Because the logistic function can be needed in the nonlinear regression, also the derivatives are implemented. Note, to utilize derivatives the function needs to be in a TRANS object. Eg when f=logistic(a*(x-x0)), then the derivatives can be obtained with respect to the parameters a and x0 by

Example 22.1 (logisticex). Example of logistic function

```
 transa=trans()
der(a,x0)
f=logistic(a*(x-x0));
/


  x,x0,a=10,5,0.1
call(transa)
```

Note 22.2.1. In the previous example tr(d[x0] ahs the effect that TRANS tr is first called, which makes that also d[a] and d[x] have been computed. Remember that the parse tree is computed from right to left.

22.3 npv() Net present value

npv(]interest,income1,...,incomen,time1,...,timen[)// Returns net present value for income sequence income1,...,incomen, occurring at times time1,...,timen when the interest percentage is interest.

23 Probality distributions

There are currently the following functions relate to probability distributions.

Note 23.0.1. function density() can be used define density or probability function for any continuous or discrete distribution which can then be used to generate random numbers with random() function.

23.1 pdf() Normal density

Output	1	REAL
	the value of the density.	
Args	0-2	REAL

	Arg1 is the mean (default 0), Arg2 is the standard deviation (default 1). If sd is given, the mean must be given explicitly as teh first argument.

Note 23.1.1. See example drawclassex for an utilization of pdf()

23.2 cdf() Cumulative distribution for normal and chi2

Output	1	REAL
	The value of the cdf.	
Args	1-3	REAL
	Arg1 the upper limit of the integral. When chi2-> is not present, then Arg2, if present is the mean of the normal distribution (defaul 0), and Arg3, if present, is the sd of the ditribution. If chi2-> is present, then oblicatory Arg2 is ifs the number of degrees of freedom for chi2-distribution.	
chi2	N |0	
	chi2	N |0

23.3 bin() Binomial probability

bin(k,n,p)= The binomial probability that there will be k successes in n independent trials when in a single trial the probability of success is p.

23.4 negbin() Negative binomial

negbin(k,myy,theta)= The probability that a negative binomial random variable has value k when the variable has mean myy and variance myy+theta*myy**2.

Note 23.4.1. negbin(k,n*p,0)= bin(k,n*p).

Note 23.4.2. Sorry for the parameter inconsistency with rannegbin().

23.5 density() for any discrete or continues distribution

Make density for for random numbers either with a function or histogram generated with classify.

Args	0-1	MATRIX

	MATRIX generated with classify()
func	N|1 codeoption defining the density. The x-varaible is $.
xrange	0|2 REAL Range of x-values
discrete	-1|0 Presence implies the the distribution is discrete

Note 23.5.1. Actually the function generates a matrix having towo rows which has values for the cumulative distribution function.

Note 23.5.2. When defining the density function, the user need not care about the scaling constant which makes the integral to integrate up to 1.

Example 23.1 (densityex). Example of distributions

```
 ber=density(func->(1-p+(2*p-1)*$),xrange->(0,1),discrete->);Bernoully
bim=matrix(100)
bim=random(ber)
mean(bim);
p*(1-p); !theoretical variance
var(bim);
pd=density(func->exp(-0.5*$*$),xrange->(-3,3)) !Normal distribution


  ra=random(pd);
f=matrix(1000)
f=random(pd)
da=newdata(f,read->x)
stat(min->,max->)
cl=classify(x->x,xrange->);
fi=drawclass(cl,continue->fcont)
fi=drawclass(cl,area->,continue->fcont)


  fi=draw(func->pdf(x),x->x,xrange->,append->,continue->fcont)
f=matrix(1000)
f=rann()
da=newdata(f,read->x)
stat(min->,max->)
cl=classify(x->x,xrange->,classes->20)
fi=drawclass(cl,histogram->,continue->fcont)
den=density(cl);
fi=drawline(den,continue->fcont)
```

24 Random number generators

Random number generators are taken from Ranlib library of Netlib. They can produce single REAL variables or random MATRIX objects. Random matrices are produced by defining first a matrix with matrix() funtion and putting that as the output.

24.1 ran() Uniform

Uniform random numbers between 0 and 1 are generating usig Netlib function ranf.

Output	1	REAL |MATRIX
		The generated REAL value or MATRIX. Random matrix cab generated by defining first the matrix with matrix().

Example 24.1 (ranex). `ran();`

```
ran();
cpu0=cpu()
A=matrix(10000,5)
A=ran()
mean(A);
mean(A,any->)!mean over all elements
mean(A(All,2));
sd(A);
sd(A,any->);
min(A);
min(A,any->);
max(A);
cpu()-cpu0;
```

24.2 rann() Normal

Computes normally distributed pseudo random numbers into a REAL variable or into MATRIX.

Output	1	REAL|MATRIX
		The matrix to be generated must be defined earlier with matrix().
Args	0-2	num
		rannn() produces N(0,1) variables, rann(mean) will produce N(mean,1) variables

and rann(mean,sd) procuses N(mean,sd) variables.

Example 24.2 (rannex). Random normal variates, illustrating also find

```
 rx=rann(); !Output is REAL
rm=matrix(1000)
rm=rann()
rm(1,-5);
print(mean(rm),sd(rm),min(rm),max(rm))
Continue=1 !an error
large=find(rm,filter->($.ge.2),any)
Continue=0
large=find(rm,filter->($.ge.2),any->)
100*nrows(large)/nrows(rm);
cpu0=cpu()
rm2=matrix(1000000)
rm2=rann(10,2) !there cannot be arithmetix opreations in the right side
cpu()-cpu0;
mean(rm2),sd(rm2),min(rm2),max(rm2);
large=find(rm2,filter->($.ge.14),any->)
100*nrows(large)/nrows(rm2);
!
```

Note 24.2.1. When generating a matrix with random numbers, there cannot be arithmetic operations on the right side.That means that code:
rm=matrix(100)
rm=2*rann()
would produce a REAL value rm.

24.3 ranpoi() Poisson

ranpoi(myy
returns a random Poisson variable with expected value and variance myy

24.4 ranbin() Binomial

Binomial random numbers between 0 and n are generating usig Netlib ignbin(n,p). Random matrix can generated by defining first the matrix with matrix().

Output	1	REAL |MATRIX

The generated REAL value or MATRIX with number of successes. (**Jlp22** does not have explicit integer type object).

Args	2	REAL
	Arg1 is the number of trials (n) and Arg2 is the probability of succes in one trial. !	

Example 24.3 (ranbinex). Random binomial

```
 ranbin(10,0.1);
ranbin(10,0.1);
A=matrix(1000,2)
Continue=1
A(All,1)=ranbin(20,0.2)
Continue=0
A=matrix(1000)
A=ranbin(20,0.2)
B=matrix(1000)
B=ranbin(20,0.2)
da=newdata(A,B,read->(s1,s2))
stat(min->,max->)
cl=classify(1,x->s1,xrange->)
fi=drawclass(cl,histogram->,color->Blue,continue->fcont)
cl=classify(1,x->s2,xrange->)
fi=drawclass(cl,histogram->,color->Red,append->,continue->fcont)
```

24.5 rannegbin() Negative binomial

The function returns random number distributed according to the negative binomial distribution.

Output	1	REAL | MATRIX
	the number of successes in independent Bernoul trials before r'th failure when p is the probability of success. ranbin(r,1)returns 1.7e37 and ranbin(r,0)returns 0.	

Note 24.5.1. there are different ways to define the negative binomial distribution. In this definition a Poisson random variable with mean $$ is obtained by letting r go to infinity and defining p= $$/($$+r) The mean E(x) of this definition is p*r/(1-p) and the variance is V=p*r/(1-p)2. Thus given E(x) and V, r and p can be obtained as follows: p=1- E(x) /V and r= E(x)**2/(V- E(x)) . This is useful when simulating 'overdispersed Poisson' variables. Sorry for the (temporary) inconsistency of parameters with function negbin().

Note 24.5.2. can also have a noninteger values. This is not in accordance with the above interpretation of the distribution, but it is compatible with interpreting negative binomial distribution

as a compound gamma-Poisson distribution and it is useful when simulating overdispersed Poisson distributions.

24.6 select() Random selection

Output	1	MATRIX
	column vector with n elements indicating random selection of k elements out of n elements. The selection is without replacement, thus elements of the output are 1 or 0..	
Args	2	REAL
	k=Arg1 and n=Arg2.	

Example 24.4 (selectex). Random selection

```
 ** select 500 numbers without replacement from 10000
** output is vector of 10000 rows containing 0 or 1
S=select(500,10000)
nrows(S),mean(S),sum(S),500/10000;
```

24.7 random() Any distribution

usage random(dist) where dist is the density defined in density(). See density() for examples.

25 Interpolation

The following functions can be used for interpolation

25.1 interpolate() Linear interpolation

Usage:// interpolate(x0,x1[,x2],y0,y1[,y2],x]// If arguments x2 and y2 are given then computes the value of the quadratic function at value x going through the three points, otherwise computes the value of the linear function at value x going through the two points.

Note 25.1.1. The argument x need not be within the interval of given x values (thus the function also extrapolates).

25.2 plane() Interpolates an a plane

Usage:
plane(x1,x2,x3,y1,y2,y3,z1,z2,z3,x,y]
The function computes the equation of plane going through the three points (x1,y1,z1), etc and computes the value of the z-coordinate in point (x,y). The three points defining the plane cannot be on single line.

25.3 bilin() Bilinear interpolation

Usage:
bilin(x1,x2,y1,y2,z1,z2,z3,z4,x,y]
z1 is the value of function at point (x1,y1), z2 is the value at point (x1,y2), z3 is the value at (x2,y1) and z4 is the value at (x2,y2): the function is using bilinear interpolation to compute the value of the z-coordinate in point (x,y). The point (x,y) needs not be within the square defined by the corner points, but it is good if it is. See Press et al. ? (or Google) for the principle of bilinear interpolation

26 List functions

The following list functions are available

26.1 Object lists

An object list is a list of named **Jlp22** object. See Shortcuts for implicit object lists and List functions for more details. Object lists can be used also as pointers to objects, see e.g. the selector option of the simulate() function.

26.2 list() Creates LIST

Output	1	LIST
	The generated LIST object.	
Args	0-	
	named objects.	
merge	N	
	Duplicates are dropped.	

do	-1-	REAL
	Consecutive values are given to variables in the LIST. If do-> has no arguments, values 1,2,.. are used. If do-> has one argument, it gives the starting value. If do-> has two arguments, the first argument gives the starting value and the second argument gives the step.	
mask	N|1-	REAl
	Which object are picked from the list of arguments. value 0 indicates that he object is dropped, positive value indicates how many variables are taken, negative value how many objects are dropped (thus 0 is equivalent to -1). mask-option is useful for creating sublists of long lists.	

Note 26.2.1. The same object may appear several times in the list. (see merge())

Note 26.2.2. There may be zero arguments, which result in an empty list which can be updated later.

Note 26.2.3. Also matrix() has option do-> for similar purpose as list(), but it works slightly differently as in matrix(), do-> can determine the dimensions of generated MATRIX and it is possible to fill only part of the MATRIX.

Note 26.2.4. The index of object in a LIST can be obtained using index().

Note 26.2.5. Versions before March 2023 allowed list arguments which were then expanded. This property is dropped because lists can be included better with @-cosnstruct.

Example 26.1 (listex). `li=list(x1...x5);`

```
li=list(x1...x5,merge->);
li=list(x1...x5,x2...x4,merge->);
index(x2,li);
***returns zero
index(x6,li);
***
all=list(); !  empty list
sub=list();
nper=3
;do(i,1,nper)
period#"i"=list(ba#"i",vol#"i",age#"i",harv#"i")
sub#"i"=list(@period#"i",mask->(-2,1,-1))
all=list(@all,@period#"i") !note that all is on both sides
sub=list(@sub,@sub#"i")
;end do
li=list(x1...x13,do->);
```

```
li=list(x1...x13,do->4)
li=list(x1...x13,do->(3,2));
```

26.3 Symbolic constants

REAL variables having some fixed value are can be considered as constants. Only some constants created at the initialization are constants in the sense that their value cannot be changed later. For some time I made a possiblity to lock the values of other variables also, but this was not very useful. But in a DATA having symbolic names for cases, the names for cases can be considered as symbolic contants. In DATA with cases data%case is a list having the case names as elements. These case names are actually ordinary REAL variables but with the special property that the value of ith element is equal to i. This way the case name tells what is the observation number, which makes it possible to acces a row in data matrix using the case name. Other lists of symbolic constants can be made with list2sym() function, which can pick from the names of varaibles in the list a new LIST having this property. The purpose of symbolic constants is give a tag which ca be used to tell that subjects occuring in different parts of the code refer to the same subject. When the symbolic constant is in the case-list of a DATA, the value of the constant tells in what observation of the DATA further properties are defined. The namenum() and namenum%() functions allow to maintain also numeric variables in the name of the constant. This way it is not necessary to look these quantitative properties from the DATA because this infomation is carried in ther name. This is a handy way to maintain indexes without the need to establish permanent indexes. I will use this property in a forthcoming application where **Jlp22** is used for optimization of blade settings in saw mills.

26.3.1 list2sym() makes LIST of symbolic constants from another LIST.

List of symbolic constants is a LIST of REAL variables where the value of each variable equals the position of the variable in the list. Applications of this will be presented in the worthcoming project of optimization of sawing blade settings.

Output	1	LIST
	LIST of symbolic constants generated	

Args	2	List and REAL

The first argument tells from waht list the output is generated. It is probably the list of cases in a DATA with cases. The second argument tells how many '%' signs precede that part of variable names from which the list is made. The

variable names are taken to the next '%' or to the end of name.

Note 26.3.1. See functions namenum() and namenum%() how to pick numbers from variable names.

Example 26.2 (list2symex). Example of list2sym()

```
 *** Log boxes in a sawmill where boxes are defined with top diameter, length
and species.
logdata=data(read->(T_log,N_log),in->,case->)
L15%40%Spruce,200
L15%43%Spruce,300
L18%40%Pine,100
L18%43%Spruce,150
/
li=;list(logdata%?);
@li;
logdata%case;
** Pick after two %
ili=list2sym(logdata%case,2);
@ili;
```

26.3.2 namenum() and namenum%() take numbers from object name

Object names often contain some numbers. These can be picked using namenum() and namenum%() functions. Probably the function will be used so that the object is in an object LIST. The are two ways to pick object from LIST, either in input programming or either at sit> prompt or in TRANS object. Usually the numbers are picked from the list of cases in a data with cases. Function namenum() picks the number which is presented with consecutive numbers, while namenum%() picks the numbers from a REAL variable.

Output	1	REAL
	The number picked from the name	

Args	1-3	REAL|LIST

There are two cases:
The first argument is the object from whose name the number is picked. Then the second argument can be The second argument tells what is the order of the number among numbers (in namenum()) or REAL variables whose name starts after '%' (in Namenum%()) of the name. Default is one, i.e. the first number is picked.

The fist argument is LIST and sewcond argument is the element number and third argument can be the order of number.

Example 26.3 (namenumex). Example of namenum() and namenum%() in the data decribing the log storage in a sawmill

```
 logdata=data(read->(T_log,N_log),in->,case->)
L15%40%Spruce,200
L15%43%Spruce,300
L18%40%Pine,100
L18%43%Spruce,150
/
D=namenum(logdata%case,3,2);
D=namenum(@logdata%case(3),2);
logdata%case;
ili=list2sym(logdata%case,2);
@ili;


  species=namenum%(logdata%case,3,2);
species=namenum%(@logdata%case(3),2);
```

26.4 ;list(part%?) LIST of objects having common part in name

;list() Makes a list of objects. The example shows how the list can be printed and how to print all objects in the list. This function is described also in section 9.19.2

Example 26.4 (subobjex2). ex2

```
 dataa=data(in->,read->(x,y))
1,2
3,4
/
lista=;list(dataa%?);
**Then all objects in the list are printed.
@lista;
```

26.5 merge() Merges LISTs and ILISTs by dropping duplicates.

merge() will produce of a LIST consisting of separate objects or values in the arguments. !

Example 26.5 (mergex). Merging list

```
 lista=list(x1...x5,x3);
** LISTs must be expanded with @
listb=merge(@lista)
ilista=ilist(1...5,3);
```

```
ilistb=merge(ilista,ilist->);
*** without ilist the arguments are interpreted as object indices
ilistc=merge(ilista);
```

26.6 difference() Difference of LISTs or removes an object from LIST

difference() removes elements from a LIST

Output	1	LIST
	the generated LIST.	
Args	2	LIST|OBJ
	The first argument is the LIST from which the elements of of the are removed If second argument is LIST then all of its eleemts are remove, other wise it is assumed that the second argument is an object which is removed from the lisrt.	

Example 26.6 (diffex). `fex`

```
 lis=list(x1...x3,z3..z5);
lis2=list(x1,z5);
liso=difference(lis,lis2);
liso2=difference(liso,z3);
Continue=1
lisoer=difference(lis,z6);!  error occurs
liser=difference(Lis,x3); !error occurs
Continue=0
```

26.7 index() Index in a LIST or MATRIX

index(obj,list) return the index of object obj in LIST list. Fuction returns zero if list is not in LIST list.
index(valuea,matrixa) returns the location of valuea in MATRIX matrixa when going the matrix through in row order. If valuea is not in matrixa index() returns zero.
index(valuea,matrixa,any->) returns the location of the first element which is greater or equal to valuea.

26.8 index_o() Index of the first argument in the LIST of second argument

26.9 len() Length of LIST, ILIST or MATRIX

len(arg) Lenghts for the following argument types

- arg is MATRIX => len=the size of the matrix, i.e. nrows(arg)*ncols(arg)
- arg is TEXT => len=the number of chracter in TEXT object
- arg is LIST => len=the number of elements in LIST
- arg is ILIST => len=the number of elements in ILIST

If arg does not have a legal type for len(), then len(arg)=-1 if len() has option any->, otherwise an error is produced.

26.10 ilist() ILIST of integers

Generates a list of integers which can be used as indexes.

Output	1	ILIST
	The generated ILIST.	
Args	0-	REAL
	Values to be put into ILIST, or the dimesion of the ILIST when values are given in values->, or variables whose indeces in in the data are put into the ILIST.	
data	N|1	DATA
	The DATA from whose variable indeces are obtained.	
extra	1	REAL
	Extra space reserved for later updates of the ILIST.	
values	N|1-	REAL
	Values to be put into ILIST when dimesnion is determined as the only argument	

Note 26.10.1. ILIST is a new object whose all utilization possiblities are not yet explored. It will be used e.g. when developing factory optimization.

Note 26.10.2. eote Using ilist() by giving the dimension as argument and values with values-> option imitates the definition of a matrix (column vector). The structure of ILIST object is the same as LIST object which can be used in matrix computations.

Example 26.7 (ilistex). ILIST examples

```
 1,4,5;
4...1;
A=matrix(4,4)
A(1,4,3)=7;
```

26.11 putlist() puts into LIST an object

Usage:
putlist(LIST,OBJ)
put OBJ into LIST

26.12 table() Crossses two LISTs

Usage:
Output=table(rowlist,collist)
This can be used in factory optimizations

call j_getname(ivout) call j_getname(j_o(ivout)%i(6)) write(6,*)'jdjdj ',j_oname(1:j_loname)

26.13 lista(2,-4) makes LIST from part of LIST lista

A list can be made taking a part of LIST.

Example 26.8 (sublistex). taking sublist

```
 lista=list(x1...x5);
lista2%5=lista(2,-5);
```

---

27 TEXT and TXT text objects

The are now two object types for text.

27.1 text() Creates TEXT

Text objects are created as a side product by many **Jlp22** functions. Text objects can be created directly by the text() function which works in a nonstandard way. The syntax is:
output=text()
...
//

The input paragraph ends exceptionally with '//' and not with '/'. The lines within the input paragraph of text are put literally into the text object (i.e. even if there would be input programming functions or structures included)

27.2 txt() Creates TXT

Works as text(). The new TXT object is used to implement ;incl and gnuplot -figures.

28 File handling

The following functions can handle files.

28.1 exist_f(): does a file exist

looks whether a file with the name given in the character constant argument exists.

Note 28.1.1. In the previous versions of **Jlp22** same function was used for files and objects.

28.2 delete_f() Deletes files

The function delete_f() deletes all files having the names given in the arguments. The arguments can be character constants or character variables associated with character constants. After deleting a file whose name is given in character variable, the variable still refers to the same chracter constant.

Example 28.1 (deletfex). `Deletef file`

```
 write('delete_fex.txt',$,'a=matrix(2,4,do->);')
write('delete_fex.txt',$,'delete_o(a)')
write('delete_fex.txt',$,'a;')
close('delete_fex.txt')
;incl(delete_fex.txt)
delete_f('delete_fex.txt')
```

28.3 print_f() prints files

Prints the files associated with the CHAR arguments
maxlines-> tells the maximum number of lines to be printed, default the value of Maxlines

28.4 close() Closes a file

close(file) closes an open file where file is either a character constant or character variable associated with a file.

Note 28.4.1. No open()function is needed. An file is opened when it is first time in write() or print(file->).

28.5 showdir() shows the current directory

Note 28.5.1. showdir is defined in the system dependent file jsysdep_gfortran.f90. Using other compliers it may be neccessary to change the definition

28.6 setdir() sets the current directory

Note 28.6.1. setdir is defined in the system dependent file jsysdep_gfortran.f90. Using other complilers it may be neccessary to change the definition

28.7 thisfile() Name of the current ;incl -file

The name of the current include file is returned as a character variable by: out=thisfile() This is useful when defining shortcuts for commands that include sections from an include file. Using this function the shortcuts work even if the name of the include file is changed. See file jexamples.inc for an application

28.8 filestat() Information of a file

Function filestat(filename) prints the size of the file in bytes (if available) and the time the file was last accessed

29 Io-functions

Theree are following io functions

29.1 read() Reads from file

read(file,format[,obj1,...,objn][,eof->var] [,wait->])// Reads real variables or matrices from a file. If there are no objects to be read, then a record is by-passed.// Arguments: file the file name as a character variable or a character constant// format// b' unformatted (binary) data // 'bn' unformatted, but for each record there is integer for the size of the record. Does not work when reading matrices. 'bis' binary data consisting of bytes, each value is converted to real value (the only numeric data type in J). This works only when reading matrices.// '(....)' a Fortran format. Does not work when reading matrices. $ the * format of Fortran// obj1,...,objn **Jlp22** objects// Options:// eof Defines the variable which indicates the end of file condition of the file. If the end of the file is not reached the variable gets the value 0, and when the end of file is reached then the variable gets value 1 and the file is closed without extra notice.

When eof-> option is not present and the file ends then an error condition occurs and the file is closed.// wait **Jlp22** is waiting until the file can be opened. Useful in client-server applications. See chapter **Jlp22** as a server.

Note 29.1.1. Use ask() or askc() to read values from the terminal when reading lines from an include file.

Note 29.1.2. When reading matrices, their shapes need to be defined earlier with matrix() hfunction.

29.2 write() Writes to console or to file

write(file,format,arg1,...,argn)
Writes real values and text to a file or to the console.

Arguments:
file variable $ (indicating the console), or the name of the file as a character variable or a character constant.
format

- $ indicates the '*' format of Fortran for numeric values. The numeric values are converted to single precision. This format can be used also when there are CHAR and REAL arguments.
- $$ indicates the '*' format of Fortran, works only for numeric values. The numeric values are printed in double precision.
- 'b' The values are converted to single precision and written to a binary file

- 'B' The values are written to a binary file in double precision
- A Fortran format statement, e.g. (~the values were ~,4f6.0) when there are numeric values to be written
- If there are no arguments and the format is a CHAR, then this is written.

For other arguments the following rules apply:

- If the format is $ and the arguments are REAL or CHAR in any order, arguments are written
- If there is one MATRIX argument then this matrix is written row after row using the format indicated.
- If there are several arguments which are REAL or MATRIX, then the values are put into a vector which is then written. If format is 'b' or $ then the values are converted to single precision. For matrices, all values written without paying attention to to number of rows or columns. If matrices needs to be written row by row, the matrices can be put into a single matrix with matrix operations and then this matrix can be used as a single argument (after file and format).

tab	-1-1 if format is a Fortran format then, tab-> option indicates that sequences of spaces are replaced by tab character so that written text can be easily converted to Ms Word tables. If there are no decimals after the decimal point also the decimal point is dropped.

Note 29.2.1. There have been options to describe the format with tab and width parameters but they may not work now. If such formats are needed, I can reinstall them.

Note 29.2.2. Also text objects can be written, but I must check how this works now.

29.3 print() Prints objects to file or console

print(arg1,...,argn[,maxlines->][,data->][,row->] [,file->][,func->][,debug->])// Print values of variables or information about objects.// Arguments: arg1,...,argn Options: maxlines-1|1REAL the maximum number of lines printed for matrices, default 100. any -1|0 if the argument is the name of a text file, then any-> indicates that the file is read to the end and the number of lines is put into variable Accepted. data-1|1 DATA data sets. If data-> option is given then arguments must be ordinary real variables obtained from data. row if a text object is printed, then the first value given in the row-> option gives the first line to be printed. If a range of lines is printed, then the

second argument must be the negative of the last line to be printed (e.q. row->(10,-15)).Note that nrows() function can be used to get the number of rows. file the file name as a character variable or a character constant. Redirects the output of the print() function to given file. After printing to the file, the file remains open and must be explicitly closed (close('file')) if it should be opened in a different application. form when a matrix is printed, the format for a row can be given as a Fortran format, e.g. form '(15f6.2)' may be useful when printing a correlation matrix. debug the associated real variable part is first printed, and thereafter the tow associated two integer vectors, the real vector and the double precision vector func all functions available are printed

Note 29.3.1. For simple objects, all the object content is printed, for complicates objects only summary information is printed. print(Names) will list the names, types and sizes of all named **Jlp22** objects. The printing format is dependent on the object type.

Note 29.3.2. print() function can be executed for the output of a **Jlp22** command by writing ';' or ';;' at the end of the line. The excution of implied print() is dependent on the value of printresult. If printresult =0, then the output is not printed, If printresult =1, then ';' is causing printing, if printresult =2 then only ';;'-outputs are printed, and if printresult =3, then both ';' and ';;' outputs are printed.

Note 29.3.3. it was earlier to print a file using this function, but now function Print_f must be used fro taht purpose.

29.4 print_f() prints files

Prints the files associated with the CHAR arguments
maxlines-> tells the maximum number of lines to be printed, default the value of Maxlines

29.5 ask() Asks REAL

ask([var][,default->][,q->][,exit->])// Ask values for a variable while reading commands from an include file.// Argument:// var 0 or one real variable (need not exist before) Options: default default values for the asked variables q text used in asking exit if the value given in this option is read, then the control returns to command level similarly as if an error would occur. If there is no value given in this option, then the exit takes place if the text given as answer is not a number.

Note 29.5.1. If there are no arguments, then the value is asked for the output variable, otherwise for the argument. The value is interpreted, so it can be defined using transformations. Response with carriage return indicates that the variables get the default values. If there is no default-> option, then the previous value of the variable is maintained (which is also printed as the default-

> value in asking)

Example 29.1 (askex). Examples for ask()

```
 a=ask(default->8)
ask(a,default->8)
print(ask()+ask())!  ask without argument is a numeric function
ask(v,q->'Give v>')
```

29.6 askc() Asks CHAR

Usage :// askc(chvar1[,default->][,q->][,exit->]) Asks values for character variables when reading commands from an include file.

Args	0-4	REAL | ILIST
	row and column range as explained below.	
Args	0|1	CHAR
	character variable (need not exist before)	
default	0|1	CHAR
	default character stings	
q	0|1	CHAR
	text used in asking	
exit	-1|0	
	if the character constant or variable given in this option is read, then the control return to command level similarly as if an error would occur.	

Note 29.6.1. Note Response with carriage return indicates that the variable gets the default value. If there is no default-> option, then the variable will be unchanged (i.e. it may remain also as another object type than character variable).

Note 29.6.2. If there are no arguments, then the value is asked for the output variable, otherwise for the arguments.

29.7 Printresult controls printing of the result

If there is semicolon ; at the end of a line at command level or within a TRANS object line, the the output is printed with implicit printresult() function if the value of Printresult has value 1. If the line ends with double semicolon ;; then the output is printed if variable Printresult has value 2. If Printresult=3, then the output of both ; and ;; -lines are printed. Using ;pause in command level or pause() in TRANS object makes it possible to debug code initially by printing many variables and objects, and the the printing can be stopped during a pause by changing the value by putting during the pause Printresult into zero. If variable Printpause has the same value as Printresult, then a pause is generated after printing. In the version 28.6.2023 'printing' a TRANS object means taht the trasnforamtion is computed. This gives still further possibilities for debugging.

Note 29.7.1. Controlling the printing with ending semicolon is a property stolen from Matlab. In order to avoid possible lawsuits, it works differently: in Matlab ending semicolon prevents printing. In **Jlp22** this printing is more powerful as there are two different printing levels, and the the printing can be changed without changing the script.

30 Matrix functions

Jlp22 contains now the following matrix functions.

30.1 Matrices and vectors

Matrices and vectors are generated with the matrix() function or they are produced by matrix operations, matrix functions or by other **Jlp22** functions. E.g. the data() function is producing a data matrix as a part of the compound data object. Matrix elements can be used in arithmetic operations as input or output in similar way as real variables. See Matrix computations.

30.2 matrix() Creates MATRIX

Function matrix() creates a matrix and puts REAL values to the elements. Element values can be read from the input paragraph, file, or the values can be generated using values-> option, or sequential values can be generated using do-> option. Function matrix() can generate a diagonal and block diagonal matrix. A matrix can be generated from submatrices by using matrices as arguments of the values-> option. It should be noted that matrices are stored in row order.

Output	1	MATRIX | REAL

If a 1x1 matrix is defined, the output will be REAL. The output can be a temporary matrix without name, if matrix() is an argument of an arithmetic function or matrix function. If no element values are given in values-> or obtained from in-> input, all elemets get value zero.

Args 0-2 REAL

The dimension of the matrix. The first argument is the number of rows, the second argument, if present, the number of columns. If the matrix is generated from submatrices given in values->, then the dimensions refer to the submatrix rows and submatrix columns. If there are no arguments, then the it should be possible to infer the dimensions from values-> option. If the first argument is Inf, the the number of rows is determined by the number number of lines in source determined by in->.

---

in N|0|1 CHAR

The input for values. in-> means that values are read in from the following input paragraps, in->file means that the values are read from file. in both cases a record must contain one row for the matrix. If there is reading error and values are read from the terminal, **Jlp22** gives possibility to continue with better luck, otherwise an error occurs.

---

values N|1- REAL

values or MATRIX objects put to the matrix. The argumenst of values-> option go in the regular way through the interpreter, so the values can be obtained by computations. If only one REAL value is given then all diagonal elements are put equal to the value (ohers will be zero), if diag-> option is present, otherwise all elements are put equal to this value. If matrix dimensions are given, and there are fewer values than is the size the matrix, matrix is filled row by row using all values given in values->. If there are more values as is the size, an error occurs unless there is any-> option present. Thus matrix(N,N,values->1,diag->) generates the identity matrix. If value-> refers to one MATRIX,and diag-> is present then a block diagonal matrix is generated. Without diag->, a partitioned matrix is generated having all submatrices equal. In the new version also values given in the input paragraph or in the in-> file can have names of variables whose values are put into the matrix. The value can be also computed using transformations if the element is put within parenthesis. If values are read from a file, then code-> option must be present telling **Jlp22** that values can not be read directly. The same method can be used in data() function.

code	N|0	REAL
	The values given in in-> file are interpreted and not read directly.	
do	N|0-3	REAL
	A matrix of number sequences is generated, as followsws: do-> Values 1,2,...,arg1 x arg2 are put into the matrix in the row order. do->5 Values 5,6,...,arg1 x arg2+4 are put into the matrix do->	

Note 30.2.1. In the current version the values in the input paragraph or in-> file can be separated with commas or spaces, or tabulators, and there can be several separators between two values.

Note 30.2.2. Values given for vectors or for the diagonal in a diagonal matrix can be in several lines.

Note 30.2.3. If there are fewer values in the input paragraph or in the file than there are matrix elements, a warning but no error is generated. Extra values are ignored without a notice.

Example 30.1 (matrixex). Example of generatig matrices

```
 A=matrix(3,values->(sin(2),sqrt(3),6));
** The same in the new version
h=9
A=matrix(3,in->);
(sin(2)),(sqrt(3)),6,h
/
ma=matrix(5,5,diag->,in->);
15,(sin(4)),43,h
5,7
8,9
/
```

30.3 nrows() Number of rows in MATRIX, TEXT or BITMATRIX

can be used as:

- nrows(MATRIX)
- nrows(TEXT)
- nrows(BITMATRIX)

Note 30.3.1. If the argument has another object type, and error occurs

30.4 ncols() Number of columns in MATRIX or BITMATRIX

can be used as:

- nrows(MATRIX)
- nrows(BITMATRIX)

Note 30.4.1. If the argument has another object type, and error occurs

30.5 t() Transpose of a MATRIX or a LIST

t(MATRIX) is the transpose of a MATRIX. As LIST objects can now be used in matrix computations, t(LIST) is also available.

Note 30.5.1. Multiplying a matrix by the transpose of a matrix can be made by making new operation '*.

Note 30.5.2. The argument matrix can also be a submatrix expression.

30.6 inverse() Inverse and condition of MATRIX

inverse(matrixa) computes the inverse of a square MATRIX matrixa. The function utilized dgesv funtion of netlib. If the argument has type REAL, then the reciprocal is computed, and the output will also have type REAL. An error occurs, if matrixa is not a square matrix or REAL, or matrixa is singular according to dgesv. If the output is a named object (i.e. not a temporary object), the condition number is stored in REAL with name Output%condition.

Note 30.6.1. The condition number is **not** put into input%conditon which could be more logical.

Example 30.2 (inverseex). inverse() and condition number

```
 matrixa=matrix(4,4)
matrixa=1
*** well conditoned matrix
matrixa(diag->)=10
matrixa;
matrixb=inverse(matrixa);
matrixb%condition;
** almost singular matrix
matrixa(diag->)=1.05
matrixb=inverse(matrixa);
matrixb%condition;
** figure of condition number
transa=trans()
matrixa(diag->)=diag
```

```
matrixb=inverse(matrixa)
/
** Note that the lower bound is equal to the dimension
figa=draw(x->diag,xrange->(1.05,50),func->transa(matrixb%condition),
color->Blue,continue->fcont)
```

Note 30.6.2. instead of writing c=inverse(a)*b, it is faster and more accurate to write c=solve(a,b)

30.7 solve() Solves a linear equation A*x=b

A linear matrix equation A*x=b can be solved for x with code
x=solve(A,b)

Note 30.7.1. x=solve(A,b) is faster and more accurate than x=inverse(A)*b

Note 30.7.2. solve works also if A and b are scalars. This is useful when working with linear systems which start to grow from scalars.

30.8 qr() QR decomposition of MATRIX

Makes QR decomposition of a MATRIX This can be used to study if columns of a are linearly dependent. **Jlp22** prints a matrix which indicates the structure of the upper diagonal matrix R in the qr decomposition. If column k is linearly dependent on previous columns the k'th diagonal element is zero. If output is given, then it will be the r matrix. Due to rounding errors diagonal elements which are interpreted to be zero are not exactly zero. Explicit r matrix is useful if user thinks that **Jlp22** has not properly interpreted which diagonal elements are zero. In **Jlp22** qr() may be useful when it is studied why a matrix which shoudl be nonsingular turns out to be singular in inverse() or solve(). qr() is using the subroutine dgeqrf from Netlib. An error occurs if the argument is not MATRIX or if dgeqrf produces error code, which is just printed. Now the function just shows the linear dependencies, as sho in the examples.

Args	1	MATRIX
	A m-by-n MATRIX.	

30.9 eigen() Eigenvector and eigenmatrix from MATRIX

Computes eigenvectors and eigenvalues of a square matrix. The eigenvectors are stored as columns in matrix output%matrix and the eigenvalues are stored as a row vector output%values. The eigenvalues and eigenvectors are sorted from smallest to larges eigenvalue. Netlib subroutines DLASCL, DORGTR, DSCAL, DSTEQR, DSTERF, DSYTRD, XERBLA, DLANSY and DLASCL are

used.

Args	1	MAT
	A square MATRIX.	

30.10 sort() Sorts MATRIX

Usage:
sort(a,key->(key1[,key2]))
Makes a new matrix obtained by sorting all matrix columns of MATRIX a according to one or two columns. Absolute value of key1 and the value of key2 must be legal column numbers. If key1 is positive then the columns are sorted in ascending order, if key1 is negative then the columns are sorted in descending order. If two keys are given, then first key dominates.

Note 30.10.1. It is currently assumed that if there are two keys then the values in first key column have integer values.

Note 30.10.2. If key2 is not given and key1 is positive, then the syntax is: sort(a,key->key1).

Note 30.10.3. If there is no output, then the argument matrix is sorted in place.

Note 30.10.4. The argument can be the data matrix of a data object. The data object will remain a valid data object.

30.11 envelope() Convex hull of point

Output	1	MATRIX
	(nvertex+1, 2) matrix of the coordinates of the convex hull, where nvertex is the number of verteces. The last point is the same as the first point	
arg	1	MATRIX
	(n,2) matrix of point coordinates	
nobs	-1|1	REAL
	The number of points if not all points of the input matrix are used	

Note 30.11.1. The transpose of the output can be directly used in frawline() function to draw the envelope

Note 30.11.2. The function is using a subroutine made by Alan Miller and found in Netlib

30.12 find() Finds from a MATRIX

Function find() can be used to find the first matrix element satisfying a given condition, or all matrix elements satifying the conditon, and in that case the found elements can be put to a vector containg element numbers or to a vector which has equal size as the input matrix and where 1 indicates that the element satifies the condition.. Remember that matrices are stored in row order. If a given column or row of matrix A should be seaeched, use A(All,column) or A(row,ALL) to extract that row or column.

Output	1	REAL|MATRIX
	Without any-> or expand-> the first element found in row order. With any->, the vector of element numbers satisfying the conditon. If nothing found the output will be REAL with value zero. With expand->, the matrix of the same dimensions as the input matrix where hits are marked with 1.	
Args	1	Matrix
	The matrix searched.	
filter	1	Code
	The condition which the matrix element should be satisfied. The values of the matrix elements are put to the variable $.	
any	-1|0	
	The filtered element numbers are put to the output vector.	
expand	-1|0	
	The filtered elements are put the output matrix	

Example 30.3 (findex). Finding something from matrix

```
 ** Repeating the example, different results will be obtained
rm=matrix(500)
m,s=2,3
rm=rann(m,s)
mean(rm),sd(rm),min(rm),max(rm);
m+1.96*s;
** index of first row satisfying the condition:
first=find(rm,filter->($.ge.m+1.96*s));
** indeces of all rows satisfying the condition
large=find(rm,filter->($.ge.m+1.96*s),any->);
nrows(large),nrows(large)/nrows(rm),mean(large),sd(large),min(large),max(larg
```

```
** vector of equal size as rm containing 1 or 0
large2=find(rm,filter->($.ge.m+1.96*s),expand->)
mean(large2),min(large2),max(large2);
```

30.13 mean() Means or weighted means

See section matrixstat for details

30.14 sum() Sums or weighted sums

See section matrixstat for details

30.15 var() Sample variances or weighted variances

See section matrixstat for details

30.16 sd() Sd's or weighted sd's

See section matrixstat for details

30.17 minloc() Locations of the minimum values

minloc(MATRIX) generates a row vector containing the locations of the minimum values in each column. minloc(VECTOR) is the REAL scalar telling the location of the minimum value. Thus the VECTOR can also be a row vector.

Note 30.17.1. The observation for which a DATA variable gets minimum value can be obtained using minobs-> option in stat().

30.18 maxloc() Locations of the minimum values

maxloc(MATRIX) generates a row vector containing the locations of the minimum values in each column. maxloc(VECTOR) is the REAL scalar telling the location of the maxim value whether VECTOR is a row vector or column vector.

Note 30.18.1. The observation for which a DATA variable gets the maximum value can be obtained using maxobs-> option in stat()

30.19 cumsum() Cumulative sums

cumsum(MATRIX) generates a MATRIX with the same dimesnions as the argument, and puts the cumulative sums of the columsn into the output matrix.

Note 30.19.1. If the argument is vector, the cumsum makes a vector having the same form as the argument.

30.20 corrmatrix() Correlation matrix from variance-covariance matrix

This simple function is sometimes needed. The function does not test wether the input matrix is symmetric. Negative diagonal eleemnt produces error, value zero correaltion 9,99.

Output	1	MATRIX
	matrix having nondiagonal values Out(i,j)= arg(i,j)= arg(i,j)/sqrt(arg(i,i)*arg(j,j)).	
Args	1	MATRIX
	symmetric matrix	
sd	N|0	
	If sd-> is given, then diagonal elements will be equal to sqrt(arg(i,i)	

31 DATA functions

Data can be analyzed and processed either using matrix computations or using DATAs. A DATA is compound object linked to a data MATRIX and LIST containing variable (column) names, some other information. When data ere used via DATA in statistical or linear programming functions, the data are processed observarion by observation. It is possible to work using DATA or using directly the data matrix, wharever is more convenient. It is posible make new data objects or new matrices by to extracting columns of data matrix, computing matrices with matrix computations. It is possible to use data in hierarcchila way, This property is inherited fro JLP. There are two **Jlp22** functions which create DATAs from files, data() and exceldta(). data() can create hierarchical data objects. Function newdata() creates a DATA from matrices, which temselfs can be picked from data objects. Function linkdata() can link two data sets to make a hierarchical data.

Note 31.0.1. If a data file contains columns which are referred with variable names and some vectors, the it is practical to read data first into a matrix using matrix() function and then use ma-

trix operations and newdata() to make DATA with variable names and matrices. See Simulator section for an example.

Note 31.0.2. transdata() function goes through a DATA similarly as statitical functions, but does not serve a specific purpose, just transformations defined in the TRANS object refreed with trans-> option are computed. See again the simulator section.

Note 31.0.3. In earlier versions it was possible to give several data sets as arguments for data-> option. This festure is now deleted as it is possible to stack several data matrices and then use newdata() function to create a single data set.

31.1 data() Making a DATA

Data objects can be created with the data() function. A data object can created by a data() function when data are read from a file or from the following input paragraph. New data objects can be created with newdata() function from previous data objects and/or matrices. A data object can created using a TRANS object by creating first data matrix with matrix() and then using newdata() to create data object. In the new version data values can be obtained from a variable whose name is given or using computation rule within parenthesis. If the data is given in in-> file, there must be code-> option to indicate that interpretation is needed for getting values. If there is sparse-> or case-> then code-> is not needed for a file input either.

Output 1 DATA

Output 1 DATA Data object to be created.

---

read 0|1- REAL|List

Variables read from the input files. If the first variable is %nobsw its name is changed into Output%nobsw which is assumed in hierarchical data. If no arguments are given then the variables to be read in are stored in the first line of the data file separated with commas. Also the ... -shortcut can be used to define the variable list.

---

in 0- Char

input file or list of input files. If no files are given, data is read from the following input paragraph.

---

form -1|1 Char

Format of the data as follows
$ Fortran format '*', the default

b Single precison binary
B Double precison binary.
Char giving a Fortran format, e.g. '(4f4.1,1x,f4.3)'

---

case N|0|1 REAL

The corresponding column in the data gives names of variables used for the cases. The values for the variables will be the case number. If the case variable is Case, the LIST output%Case will contain all the case variables. The Case variable will be in the data, and will have values 1,2,..etc.

---

maketrans -1|1 TRANS

Transformations computed for each observation when reading the data

---

keep -1|1- REAL

variables kept in the data object, default: all read-> variables plus the output variables of maketrans-> transformations. If tr is the maketrans-> object, then keep->(x,x4,@tr%output) can be used to pick part of read-> variables and all output variables of tr. keep-> can be used to put into data matrix columns which are utilized later. If there the is read->@readv, then keep->(@readv,Regf,Resid) can be used to put Regf and Resid to the data, and these are the utilisez in regressions.

---

filter -1|1 Code

logical or arithmetic statement (nonzero value indicating True) describing which observations will be accepted to the data object. maketrans->-transformations are computed before using filter. Option filter-> can utilize automatically created variable Record which tells which input record has been just read. If observations are rejected, then the Obs-variable has as its value number of already accepted observations+1.

---

sparse N|0

In the input there can be for each observation after variables given in read-> any number of variable,value -pairs. The variables are collected from these line endings and they need not to be specified otherwise. The value part can also be a variable name or it can be obtained with transformations. Also the variables in the read-> part can be given with variables names or computaion rules withi parenthesis.

reject -1|1 Code

Logical or arithmetic statement (nonzero value indicating True) describing which observations will be rejected from the data object. If filter-> option is given then reject statement is checked for observations which have passed the filter. Option reject-> can utilize automatically created variable Record which tells which input record has been just read. If observations are rejected, then the Obsvariable has as its value number of already accepted observations+1.

keepopen -1:1 REAL

This option tells that if nobs-> option gives the number of observations then after the current DATA is read in, the file remains open so that other DATAs can be read from the same file. This option is useful when storing schedules having the tree structure and unpacking the structure with joindata(). See also splitdata().

continue -1:1 REAL

This tells that the reading continues from the file which was left open with the previous keepopen-> option.

up -1|1 DATA

Gives the upper level data to which the data is linked. The first keep variable must be updata%nobsw, which tells the number of children observations for each up data observation. If there is up data then the variable Data%obsw will give the number of observation within the upper level observation. The up data is automatically linked to the DATA created.

duplicate -1|2 TRANS

duplicate -1|2 TRANS The two TRANS object arguments describe how observations will be duplicated. This can be used only with up->. Without up_ the duplcation can be treated with standard matrix operations.The first transformation object should have Duplicates as an output variable so that the value of Duplicates tells how many duplicates ar made (0= no duplication). The second transformation object defines how the values of subdata variables are determined for each duplicate. The number of duplicate is transmitted to the variable Duplicate. These transformations are called also when Duplicate=0. This means that when there is the duplicate-> option, then all transfor-

mations for the subdata can be defined in the duplicate transformation object, and submaketrans-> is not necessary. If there are duplications the %nobsw variable of the upper level data is updated.

---

oldobs -1|1 REAL

If there are duplications of observations, then this option gives the variable into which the original observation number is put. This can be stored in the data by putting it into keep-> list, or, if keep-> option is not given then this variable is automatically put into the keep-> list of the data.

---

oldobsw -1|1 REAL

This works similarly with respect to the initial obsw variable as oldobs-> works for initial obs variable.

---

nobs -1|1 Real

Gives the number of records in the data file. If there are fewer records in file as given in nobs-> option, an error occurs. There are two reasons for using nobs-> option. First, one can read a small sample from a large file for testing purposes. Second, the reading is slightly faster as the data can be read directly into proper memory area without using linked buffers. Third, if the data file is so large that a memory overflow occurs, then it may be possible to read data in as linked buffers are not needed. If there is up dta, nobs is not needed as the numbers of observations can be computed from the %nobsw variable in the upper level data.

---

buffersize -1|1 Real

buffersize -1|1 Real The number of observations put into one temporary working buffer. The default is 10000. Experimentation with different values of buffersize-> in huge data objects may result in more efficient buffersize-> than is the default (or perhaps not). Note that the buffers are not needed if number of observations is given in nobs->.

---

par -1|1- Real

additional parameters for reading. If subform-> option is 'bgaya' then par option can be given in form par->(ngvar,npvar) where ngvar is the number of nonperiodic x-variables and npvar is the number of period specific x-variables for each period. Default values are par->(8,93).

---

rfhead	-1-1 When reading data from a text file, the first lines can contain a header which is printed but othewise ignored. If the number of header lines is geater than one, the argument gives the number of header lines. The default number of header lines, when rfhead-> is present, is one.
rfcode	-1-1 The data file can contain also **Jlp22**-code lines which are first executed. Note the code can be like var1,var,x1...x5=1,2,3,4,5,6,7, which give the possibility to define variables which describe the in-> file. The argument of rfcode-> gives the number of code lines. If there is no argument, one code line is read in.
time	-1|0 If time-> is present, the cpu-time and total time in function are printed

Note 31.1.1. there are both header and code lines, they are read in the same order as the options are.

Note 31.1.2. It was earlier possible to create both the upper level data and subdata with the same data(). This made the function very complicated. The function is also otherwise greatly simplified.

Note 31.1.3. Direct access formats and format for reading Pascal binary files are easy to include.

Note 31.1.4. data() function will create a data object object, which is a compound object consisting of links to data matrix, etc. see Data object object. If Data is the output of the function, the function creates the list Data%keep telling the variables in the data and Data%matrix containg the data as a double precision matrix. The number of observations can be obtained by nobs(Data) or by nrows(Data%matrix) or nobs(Data).

Note 31.1.5. See common options section for how data objects used in other **Jlp22** functions will be defined.

Note 31.1.6. In ealier versions, the user could select the name for the variable telling the number of the observation. After version Dec 20 2022, the name for DATA datab is always datab%obs.

Note 31.1.7. The values can be obtained from variables or form code also in matrix(), but case-> or sparse-> are not possible.

Note 31.1.8. In the previous versions DATA with case names was generated with datawcase() function. This function is not available any more as case-> option does the same thing. The

difference is that case-> varible need not be in the first column and the case column is now part of the DATA as earlier only the list containg case names was made.

Note 31.1.9. when case-> or sparse-> option is present keep-> will probably mix up **Jlp22**.

Note 31.1.10. Options nobs->100, reject->(Record.gt.100) and filter-> (Record.le.100) result in the same data object, but when reading a large file, the nobs-> option is faster as the whole input file is not read.

Note 31.1.11. If no observations are rejected, obs variable and Record variable get the same values.

Note 31.1.12. If virtual memory overflow occurs, see nobs-> optio. This should not happen easily with the currrent 64-bit application.

Example 31.1 (dataex). data() generates a new data object by reading data.

```
  **This example now contains several combinations of options
** and this example is not yet cleaned
** it may not contain all possible option comination s
da=data(read->(x1,x2),in->)
1,2
3,4
/
da%keep;
da%matrix;


  da2=data(read->(x1...x4),in->)
1,2,3,4
5,6,7,8
/
da2%keep;
da2%matrix;
write('da2.txt',$,da2%matrix)
close('da2.txt')
tr=trans()
x13=x1+x3
/
da2=data(in->'da2.txt',read->(x1...x4),keep->(x2,x4,x13),maketrans->tr)
da2%keep;
da2%matrix;
delete_f('da2.txt')


  write('da3.txt',$,'x1...x3')
write('da3.txt',$,da2%matrix)
close('da3.txt')
da3=data(in->'da3.txt',read->)
da3%keep;
da3%matrix;
```

```
delete_f('da3.txt')

  da=data(read->(x1,x2),keep->(x1,x2,Regf,Resid),in->)
1,2
3,4
/
da%keep;
da%matrix;

  tr=trans()
x1#x2=x1*x2
x1%x2=x1+x2
/
dab=data(read->(x1,x2),maketrans->tr,in->)
1,2
3,4
/
dab%keep;
dab%matrix;

  trup=trans()
x1#y1=x1*y1
x2%y2=x2+y2
/
**first variable must be %nobsw
da%keep(1)=da%nobsw

  dasub=data(read->(y1,y2),in->,up->da,maketrans->trup)
3,4
6,7
8,9
10,11
/
dasub%keep;
dasub%matrix;
stat(data->dasub)
**utilize upper level data
stat(data->dasub,up->)

  ndupl=trans()
Duplicates=0
if(da%obs.eq.2.and.Obsw.eq.2)Duplicates=2
/
dupl=trans()
x1#y1=x1*y1
x2%y2=x2+y2
if(Duplicate.gt.0)then
x1#y1=x1#y1+Duplicate
x1%y1=x1#y1+Duplicate
```

```
endif

  /


  da%keep;
da%matrix;
dasub3=data(read->(y1,y2),in->,up->da,maketrans->trup)!,duplicate->(ndupl,dup
3,4
6,7
8,9
10,11
/
dasub3%keep;
dasub3%matrix;


  da%keep;
da%matrix;
dasub2=data(read->(y1,y2),in->,up->da,duplicate->(ndupl,dupl))
3,4
6,7
8,9
10,11
/
da%matrix;
dasub2%keep;
dasub2%matrix;


  stat(data->dasub)
**utilize upper level data
stat(data->dasub2,up->)
;return
write('dasub.txt',$,dasub%matrix)
close('dasub.txt')
**this must be redefined because duplication changed nobsw
da=data(read->(da%nobsw,x2),keep->(da%nobsw,x2,Regf,Resid),in->)
1,2
3,4
/
da%keep;
da%matrix;
dasub=data(read->(y1,y2),in->'dasub.txt',up->da,maketrans->trup,
keep->(y1,y2,@trup%output,Regf,Resid))
dasub%keep;
dasub%matrix;
** x2 is now alwasy zero
regr(y1,x2,data->dasub)
** Regf and Resid are put into the data matrix
dasub%matrix;
```

```
  ** now x2 is picked from the up-data
regr(y1,x2,data->dasub,up->)
dasub%matrix;
delete_f('dasub.txt')


  write('data.txt',$,da%matrix)
close('data.txt')
dc=data(read->(x1,x2),in->'data.txt')
dc%matrix;
dc=data(read->(x1,x2),in->'data.txt',maketrans->tr)
dc%keep;
dc%matrix;
** another way to make data
damat=matrix(nrows(da%matrix),ncols(da%matrix),in->'data.txt')
da2=newdata(damat,read->(x1...x4))
da2%matrix;


  delete_f('data.txt')


  write('data.txt','(4f4.0)',da%matrix)
close('data.txt')


  dc=data(read->(x1,x2),form->'(2f4.0)',in->'data.txt')
delete_f('data.txt')
dc%matrix;


  write('data.bin','b',da%matrix)
close('data.bin')
dc=data(read->(x1,x2),form->'b',in->'data.bin')
dc%matrix;
dc=data(read->(x1,x2),form->'b',in->'data.bin',maketrans->tr)
dc%keep;
dc%matrix;
delete_f('data.bin')


  write('data.bin','B',da%matrix)
close('data.bin')j_mcase
dc=data(read->(x1,x2),form->'B',in->'data.bin')
dc%keep;
dc%matrix;


  dc=data(read->(x1,x2),form->'B',in->'data.bin',maketrans->tr)
dc%keep;
dc%matrix;
delete_f('data.bin')


  setdata=data(read->(set,set_D),in->,sparse->,case->)
```

```
set1,15,d50x100,2,d22x100,2
set2,18,d50x75,4,d20x100,2
/
stat()
setdata%matrix;
setdata%set;


  ;return
*** sparse data with case-> and makelist->
logdata=data(read->(D_log,L_log,T_log,N_log),makelist->T_log,in->)
15,40,Spruce,200
15,43,Spruce,300
18,40,Pine,100
18,43,Spruce,150
/
```

31.2 newdata() Making a DATA from MATRIXs and/or DATAs

Function newdata() generates a new data object from existing data objects and/or matrices possibly using transformations to generate new variables.

Output	1	Data
	The data object generated.	
Args	1-	Data|Matrix
	Input matrices and data objects.	
read	N|1-	REAL
	Variable names for columns of matrices in the order of matrices.	
maketrans	N|1	TRANS
	A predefined ransformation object computed for each observation.	
time	-1|0	
	If time-> is present, the cpu-time and total time in function are printed	
delete	-1|0|1	REAL
	If present, then the new data matrix is made sequentially so that used data ma-	

trices are deleted. This takes more time, but may be needed if there is shortage of memory.

Note 31.2.1. If a DATA has a link to an upper level DATA obtained with linkdata() without output, the the upper level DATA is not included. You can make a link to an upper level data using linkdata() for the DATA produced with newdata()

Note 31.2.2. It is not yet possible to drop variables.

Note 31.2.3. An error occurs if the same variable is several times in the variable list obtained by combining variables in data sets and read-> variables.

Note 31.2.4. An error occurs if the numbers of rows of matrices and observations in data sets are not compatible.

Note 31.2.5. Output variables in maketrans-> transformations whose name start with $ are not put into the new data object.

Example 31.2 (newdataex). newdata() generates a new data object.

```
  data1=data(read->(x1...x3),in->)
1,2,3
4,5,6
7,8,9
/
matrix1=matrix(3,2,in->)
10,20
30,40
50,60
/
transa=trans()
;do(i,1,3)
;do(j,1,2)
x"i"#z"j"=x"i"*z"j"
;enddo
;enddo
/
new=newdata(data1,matrix1,read->(z1,z2),maketrans->transa)
print(new)
```

31.3 Deleted function for data with cases

datawcase is a deleted function. Use data() with case->option to get names for cases.

31.4 exceldata() DATA from an excel file

Generates data object from csv data generated with excel. It is assumed that ';' is used as column separator, and first is the header line generated with excel and containing column names. The second line contains information for **Jlp22** how to read the data. First the first line is copied and pasted as the second line. To the beginning of the second line is put '@#'. Then each entry separated by ';' is edited as follows. If the column is just ignored, then put '!' to the beginning of the entry. If all characters in the column are read in as a numeric variable, change the name to accpetable variable name in **J**. If the column is read in but it is just used as an input variable for maketrans-> trasformations, then start the name with '$' so the variable is not put to the list of keep-> variables. If a contains only character values then it must be ignored using '!'. If the contains numeric values surrounded by characters, the the numeric value can be picked as follows. Put '?' to the end of entry. Put the variable name to the beginning of the entry. then put the the number of characters to be ignored by two digits, inserting aleading zero if needed. The given the length of the numeric field to be read in as a numeric value. For instance, if the header line in the excel file is

```
Block;Contract;Starting time;Name of municipality;Number of stem;Specie
```

and the first data line could be

```
MG_H100097362501;20111001;7.5.2021 9:37;Akaa;20;103;1;FI2_Spruce
```

then the second line before the first data line could be

```
##block0808?;!Contract;!Starting time;!Name of municipality;stem;specie
```

therafter the first observation would get values block=97362501,stem=1, and species=2.

If there are several input files, the header line of later input lines is ignored, and also if the second line of later files starts with '##', then it is ignored. if any later lines in any input files start with 'jcode:', then the code is computed. This way variables decribing the whole input file can be transmitted to the data. Currently jcode-output variables can be transmitted to data matrix only by using the as pseudo outputvariables in maketrans-transformations, e.g., filevar1=filevar1, if filevar1 is generated in jcode transformation. If there are several input files the file number is put into variable In before computing maketrans transformations and this variable is automatically stored in the data matrix.

Output	1	Data
	Data object generated	

in	1-	Char
	Files to read in.	
maketrans	N|1	trans
	Transformations used to compute new variables to be stored in the data.	

31.5 linkdata() Links or combines hierarchical DATAs

linkdata(updata,subdata,...) links hierarchical data sets. Currently linkdata can create also one flat file which can be used in jlp()

Args	2-999	DATA
	DATA objects in up to down order.	

Note 31.5.1. In versions before 20.12.2022 nobsw-> obtion was needed to tell what is the number of subdata -observations under the current upper level observation. When I made the data format for storing schedules having the tree structure efficiently, everything became too complicated when allowing any freedom with respect to the nobsw-variable. Thus the nobsw variable in DATA subdata has name subdat%nosw and it must be the first variable among the keep-variables of the DATA.

Note 31.5.2. In most cases links between data sets can be made be making a DATA with up-> option. If there is need to duplicate lower level observations, then this can be currently made only in data() function.

Note 31.5.3. When using linked data in functions, the values of the upper level variables are automatically obtained when accessing lower level observations, if option up-> is present.

Note 31.5.4. In the current version of **Jlp22** it is no more necassary to use linked data sets in jlp() function, as the treatment unit index in data containing both stand and schedule data can be given in unit-> option

Example 31.3 (linkdataex). Example for linkdata.

```
  ** make upper level DATA
dataa=data(read->(dataa%nobsw,site),in->)
2,4
3,5
/
dataa%matrix;
```

```
  ** make subdata as an ordinary DATA
datab=data(read->(x1,y),in->)
1,2
3,4
5,6
7,8
6,9
/
datab%matrix;
datab%keep;
**link now DATAs
** First varaible in the upper level dataa must be dataa%nobsw
linkdata(dataa,datab)
listb=;list(datab%?);
@listb;
** when working with subdata the upper level data is feeded in for all observa
** if up-> is present
** even if they are not part of the data matrix as seen from datab%keep.
stat()
stat(up->)

  **
** Note stat() and all functions assume that the last DATA created is used
** Thus when there are several DATAs around it is safer all use data-> option
** i.e.  the above could/should be stat(data->datab)
**

  **
** The flat file can be created also as follows:
** when dealing with the subdata the upper level data is automatically used
transa=trans()
Unit=Unit!adds Unit and site variables from up-data to sub-data
site=site
Unit%obsw=Unit%obsw
/
** In TRANS transa, the input variables come from the upper level data, and
** outputvariables go to the new data based on DATA datab.
datac=newdata(datab,maketrans->transa)
stat()
datac%matrix;
```

31.6 splitdata() splits a schedules DATA into components

Each component contains variables simulated for a given period. After the function the stand data is linked to DATA of schedules variables during the first period.

Output	1	?

	The variable name determining the names the generated DATAs. If the output is xdatab then the splitdata() generates DATAs xdatab#1...,xdatab#p for the number of periods p.	
Args	1	DATA
	Schedules data linked to an upper level stand data with linkdata().	
periodvars	2-	LIST
	LIST of LISTs, each list telling variables which are simulated for the corresponding period.	

Note 31.6.1. When deciding to which period a variable belongs, the only thing which matters is at what level the variable is put into the tree. Thus variable NPV0 which is the variable telling the NPV at the beginning of the plannin horizon must put to the last list in periodvars, because it can be computed only after the whole planning horizon is simulated.

Note 31.6.2. If the stand data has variable nobswold then the initial value of the nobsw variable is put into this variable. The easiest way to add this variable is to use extra->nobswold when making the stand data

Note 31.6.3. All generated matrices files can be written into one (e.g. binary) file starting from the first to the last period. These DATAS can be the read in using keepopen-> and continue-> options of data(). See below in the joindata() section

Note 31.6.4. The first variable in DATA dataa is dataa%nobsw

31.7 joindata() Joins hierarchical DATAs

There are two different cases of joindata()
The call has only two arguments and is like
xdata2=joindata(cdata,xdata)
where cdata and xdata are linked and the first variable in cdata is cdata%nobsw. Then xdata2 will contain all variables in cdata, except variable cdata%nosw is replaced by variable cdata%obs which tells into which cdata observation the xdata observation belongs, in practice, into which stand the schedule belongs. The output data will not be linked to any upper level data.
In the second case the call is like
xdata2=joindata(cdata,xdata#1...xdata#p)
DATAs xdata#1...xdata#p contain different periods (levels) in hierarchical data and the will contain whole schedules. This new data is automatically linked to to the first argument DATA (cdata).

Note 31.7.1. In the second case: As the output will be connected to the first argument, the link from the second argument to the first argument will be lost.

Note 31.7.2. In the second case: The variable cdata%nobsw in cdata is modified to correpond to the new number of children, i.e. the number of schedules in the stands.

Note 31.7.3. there are examples of the new data format after splitdata() function.

Example 31.4 (joindataex). splitting and joining period DATA

```
 cdata=data(in->'cdat.txt',read->(cdata%nobsw,site))
stat()
xdata=data(in->'xdat.txt',read->(npv#0,npv#5,income1...income5),time->)
stat()


  linkdata(cdata,xdata)
stat(up->)


  ;do(i,1,4)
pv"i"=list(income"i")
;enddo
pv5=list(income5,npv#0,npv#5)


  xdatap=splitdata(xdata,periodvars->(pv1...pv5))
stat(data->cdata)


  ;do(i,1,5)
stat(data->xdatap#"i")
write('xdatap.txt',$,xdatap#"i"%matrix)
;enddo


  close('xdatap.txt')
contd=0
keepo=1
stat(data->cdata,sum->)


  nobscur=cdata%nobsw%sum;


  ;do(i,1,5)
Pv"i"=list(xdataP#"i"%nobsw,@pv"i")
;enddo


  ;do(i,1,5)
;if(i.eq.5)keepo=0
xdataP#"i"=data(in->'xdatap.txt',read->(@Pv"i"),keepopen->keepo,continue->con
stat(data->xdataP#"i",sum->)
nobscur=xdataP#"i"%nobsw%sum;
```

```
contd=1
stat(data->xdataP#"i")
;enddo


  stat(data->cdata)
xdatanew=joindata(cdata,xdataP#1...xdataP#5)
stat(data->xdatanew)


  ** Did the procedure loose any data
stat(data->xdata)


  delete_f('xdatap.txt')
```

_____ ** To summarize

- First there was cdata and xdata linked to it.
- Then splitdata() was used to pull out different levels of the simulation tree into xdatap#1...xdatap#5 DATAs.
- Eeach data has %nobsw variable, in addition to the original period variables, telling how many children each observation has.
- Also the variable cdata%nobsw is updated.
- Then the %matrix of each DATA was written to 'xdatap.txt' from the first to last period.
- In practice it is beneficial to use binary files. Recall that format 'b' reads and writes in single precision. The matrices are in **Jlp22** in double precision but practically never double precision is needed. If insisting on double precision data storage, use format 'B'.
- Data is read in from the file period by period. Initially the sum of variable cdata%nobsw is computed using stat(data->cdata,sum->). This sum tells the number of period 1 observations.
- After making a period DATA, the number of observations in next period is computed similarly. It simplifies coding when the data for the last period has also %nobsw variable which is zero.
- keepopen-> keeps the file open making it possible to continue reading from the same file, which is indicated with the continue-> option. It would be possible to use keepopen also for the last period and close the file after reading all periods.
- The above procedure can be used to convert tabular form schedules data into this disk saving and fast reading format. But in long run a simulator could generate directly data in this format which is simple to implement: just write different periods to different files or collect data into different period matrices and write the directly all data into one file.

31.8 getobs() Obsevarion from DATA

Getting an observation from a data set:
getobs(dataset,obs[,trans->])
Get the values of all variables associated with observation obs in data object dataset. First all the variables stored in row obs in the data matrix are put into the corresponding real variables. If a transformation set is permanently associated with the data object, these transformations are executed.

Args	2	DATA,REAL
	DATA and the number of observation	
trans	-1|1	TRANS
	these transformations are also executed.	

31.9 nobs() number of observations in DATA or REGR

nobs(DATA) returns the number of rows in the data matrix of DATA// nobs(REGR) returns the number of observations used to compute the regression with regr().

31.10 classvector() Vectors from grouped DATA

Function classvector computes vectors from data which extract information from grouped data. These vectors can be used to generate new data object using newdata() function or new matrices from submatrices using matrix() function with matrix-> option or they can be used in transformation objects to compute class related things. There is no explicit output for the function, but several output vectors can be generated depending on the arguments and first->, last-> and expand-> options. The function creates automatically REAL variable Class%nobs whose value will be the number of classwes. The function prints the names of the output vectors generated.

Args	0-	REAL
	The variables whose class information is computed. Arguments are not necessary if first-> and/or last-> are present. Let Arg be the generic name for arguments.	
class	1	REAL
	The variable indicating the class. The class variable which must be present in	

the data object or which is an output variable of the trans-> transformations. When the class-> variable, denoted as as Class changes, the class changes.

---

data 0|1 Data

Data object used. Only one data object can be used; extra data-> objects just ignored. The default is the last data object generated.

---

expand -1|0

If expand-> is present then the lengths output vectors are equal to the number of observations in the data object and the values of the class variables are repeated as many times as there are observations in each class. If expand-> is not present, the lengths of the output vectors are. equal to the number of classes.

---

first 0

The the number of first observation in class is stored in vector Class%%first if expand-> is present and Class%first if expand-> is not present.

---

last 0

The the number of last observation in class is stored in vector Class%%last if expand-> is present and Class%lastobject if expand-> is not present.

---

nobsw 0

The number of observations in class is stored in vector nobsw%Class if expand is not present and in nobsw%%Class if expand-> is present

---

obsw 0

If there axpnad-> option then vector Class%%obsw

---

ext -1|1 Char

The extension to the names of vectors generated for arguments. Let Ext be denote the extension.

---

mean -1|0

The class means are stored in the vectors
Arg#Class%%mean with expand-> and without ext->
Arg#Class%%meanExt with expand-> and with ext-> are

	Arg#Class%mean without expand-> and without ext-> Arg#Class%meanExt without expand-> and with ext->
sd	-1|0 Class standard deviations are computed to sd vectors
var	-1|0 Class variances are computed to var vectors
min	-1|0 Class minimums are computed to min vectors
max	-1|0 Class maximums are computed to max vectors.

Note 31.10.1. Numbers of observations in each class can also be obtained by Class%nobs=Class%%last-Class%%first+1 when expand-> is present, and Class%nobs=Class%%last-Class%%first+1

Example 31.5 (classdata). Hierarchical data

```
 nstand=10
xm=matrix(nstand)
xm=rann(3)
ym=0.7*xm+0.1*xm
xm;
ym;
standdata=newdata(xm,ym,read->(X,Y))
stat()
ntree=6
xt=matrix(ntree*nstand)
yt=matrix(ntree*nstand)
standv=matrix(ntree*nstand)
ex=matrix(ntree*nstand)
ey=matrix(ntree*nstand)
transa=trans()
jj=0
do(i,1,nstand)
do(j,1,ntree)

  jj=jj+1
standv(jj)=i
ex(jj)=rann()
ey(jj)=0.3*ex(jj)+0.3*rann()
xt(jj)=xm(i)+ex(jj)
```

```
yt(jj)=ym(i)+0.3*ex(jj)+0.3*rann()
enddo
enddo
/
call(transa)
treedata=newdata(standv,xt,yt,read->(stand,x,y))
stat()

  !!  Making class level data object from treedata
classvector(x,y,class->stand,data->treedata,mean->,min->)
standdata2=newdata(x[stand]%mean,y[stand]%mean,x[stand]%min,y[stand]%min,
read->(x,y,xmin,ymin))
stat()
classvector(x,y,class->stand,data->treedata,mean->,expand->)
ex2=treedata(x)-x
```

31.11 values() Different values of variables in DATA

Extracting values of class variables: values( ).

Output	1	VECTOR
	the vector getting differen values	
arg	1	REALV
	variables whose values obtained	
data	1	DATA
	The data set.	

Note 31.11.1. The values found will be sorted in an increasing order.

Note 31.11.2. After getting the values into a vector, the number of different values can be obtained using nrows() function.

Note 31.11.3. values() function can be utilized e.g. in generating domains for all different owners or regions found in data.

31.12 transdata() Own computations for DATA

transdata() is useful when all necassy computions are put into a TRANS , and a DATA is gone through obsevation by observation. This is useful e.g. when simulating harvesting schdules using a simulator which is defined as an ordinary TRANS. The whole function is written below

to indicate how users' own functions dealing with DATA could be developped.

data	N|1-	DATA
	DATA object. Default last DATA defined.	
from	N|1	REAL
	First observation used.	
until	N|1	REAL
	Last observation used.	
trans	N|1	TRANS
	TRANS computed fro each observation	
epilog	N|1	TRANS
	TRANS computed after going through data.	

```
subroutine transdata(iob,io)
call j_getdataobject(iob,io)
if(j_err)return
!!call j_clearoption(iob,io)  ! subroutine

do j_iobs=j_dfrom,j_duntil
call j_getobs(j_iobs)
if(j_err)return
end do !do j_iobs=j_dfrom,j_duntil

if(j_depilog.gt.0)call dotrans(j_depilog,1)

return
```

31.13 find_d() makes a ILIST of observations satisfying a condition.

Output	1	ILIST
	The generated ILIST.	
data	N|1-	DATA

	DATA object. Default last DATA defined.	
from	N|1	REAL
	First observation used.	
until	N|1	REAL
	Last observation used.	
filter	N|1	CODE
	Code telling what observations are accepted	
reject	N|1	CODE
	Code telling what observations are rejected.	

Note 31.13.1. Function find() can be used to search all or first elements of a vector satifying a condition.

31.14 Taking data from data matrix

DATA is essentially a MATRIX data%matrix with names for columns. The matrix data%matrix can be used as any other matrix. But it is also possible to take data from data%matrix utlizing column names. The following examples show how this can be done.

Example 31.6 (fromdataex).
```
datab=data(read->(x1...x5),in->)
1,2,3,4,5
10,20,30,40,50
100,200,300,400,500
/
**taking a single value
h=datab(x3,row->2);
** taking whole columns
a=datab(x2,x5);
** taking whole row
a=datab%matrix(3,All);
**picking values from a row
a=datab(x1,x5,row->2);
**picking several rows
a=datab(x2,x5,from->1,until->2);
```

32 Statistical functions

There are several statistical functions which can be used to compute basic statistics linear and and nonlinear regression, class means, standard deviations and standard errors in one or two dimensional tables using data sets. There are also functions which can be used to compute statistics from matrices, but these are described in Section 30.2

32.1 stat() Basic statistics in DATA

Computes and prints basic statistics from data objects.

Output	0-1	REAL
	kokopo	
Args	0-99	REAL
	variables for which the statistics are computed, the default is all variables in the data (all variables in the data matrix plus the output variables of the associated transformation object) and all output	
data	N|1-	DATA
	DATA object. Default last DATA defined.	
from	N|1	REAL
	First observation used.	
until	N|1	REAL
	Last observation used.	
trans	N|1	TRANS
	TRANS computed fro each observation	
epilog	N|1	TRANS
	TRANS computed after going through data.	
data	-1,99	Data
	data objects , see section Common options for default! weight gives the weight of each observations if weighted means and variances ar transformation or it	

	can be a variable in the data object See 11.4.3 on page 40 for more details.
min	-1,99 REAL defines to which variables the minima are stored. If the value is character constant or character variable, then the name is formed by concatenating the character with the name of the argument variable. E.g. stat(x1,x2,min->'%pien') stores minimums into variables x1%pien and x2%pien. The default value for min is '%min'. If the values of the min-> option are variables, then the minima are stored into these variables.
minobs	N|0 Observation where minimum is obtained is stored in var%minobs
max	-1,99 REAL maxima are stored, works as min->
maxobs	N|0 Observation where maximum is obtained is stored in var%maxobs
mean	-1,99 REAL means are stored
var	-1,99 REAL variances are stored
sd	-1,99 REAL standard deviations are stored
sum	-1,99 REAL sums are stored, (note that sums are not printed automatically)
nobs	-1 | 1 REAL gives variable which will get the number of accepted observations, default is variable 'Nnobs'. If all observations are rejected due to fi
trans	-1 | 1 TRANS transformation object which is executed for each observation. If there is a trans-

	formation object associated with the data object, those	
filter	-1 | 1	Code
	logical or arithmetic statement (nonzero value indicating True) describing which observations will be accepted. trans-> transformations	
reject	-1 | 1	Code
	reject -1 | 1 Code	
transafter	-1 | 1	TRANS
	transformation object which is executed for each observation which has passed the filter and is not rejected by the reject->-optio	
func	-1|1	CODE
	Defines function for whose values the statistics are computed. Statistics are not computed for other variables in the data.	

Note 32.1.1. stat() function prints min, max, means, sd and sd of the mean computed as sd/sqrt(number of observations)

Note 32.1.2. If the value of a variable is greater than or equal to 1.7e19, then that observation is rejected when computing statistics for that variable.

Note 32.1.3. When making several analyses with several transformations, it is often convenient to define a basic TRANS object trans0, and define for each special analyses a TRANS object transcur, where the first line is call(trans0) and which is used as the argument in trans->. It is easier to handle short TRANS objects.

Note 32.1.4. If a **Jlp22** function used in tranformation generates MATRIX objects, then these vectors can be made ordinary data variables using lists. Defing that veclist=list(e1...e6), then a vector Evec defined inside a **Jlp22** function as a row or column vector can be accessed as data variables using veclist=Evec. It is possible to acces also matrices by remembering that matrices are stored in row order.

Example 32.1 (statex). `stat() computes minimums, maximums, means and std deviatons`

```
 data1=data(in->,read->(x1,x2,x3))
1,2,3
4,6,8
3,8,20
6,8,9
/
```

```
stat()
stat(data->data1,sum->x2,mean->,filter->(x3.le.18.5))
li=;list(x2%?);
@li;
stat(x1,data->data1,weight->x2)
stat(x1,weight->(x2**1.2))
```

32.2 cov() Covariance MATRIX

cov() computes the covariance matrix of variables in DATA.

output	1	MATRIX
	symmetric aoutput matrix.	
arg	1-N	LIST or REALV
	variables for which covarianes are computed, listing individually or given as a LIST.	
data	N|1-	DATA
	DATA object. Default last DATA defined.	
from	N|1	REAL
	First observation used.	
until	N|1	REAL
	Last observation used.	
trans	N|1	TRANS
	TRANS computed fro each observation	
epilog	N|1	TRANS
	TRANS computed after going through data.	
weight	-1|1	CODE
	Codeoption for weight of each observation.	

Note 32.2.1. the output is not automaticall printed, but it can be printed using ';' at the end of

line.

Note 32.2.2. The covariance matrix can changed into correaltion matrix with corrmatrix() function.

Note 32.2.3. If variable w in the data is used as the weigth, this can be expressed as weight->w

Example 32.2 (covex). Example of covariance

```
 X1=matrix(200)
X1=rann()
;do(i,2,6)
ad=matrix(200)
ad=rann()
X"i"=X"i-1"+0.6*ad
;enddo
Continue=1 !error
dat=newdata(X1...X6,read->(x1...x5))
Continue=0
dat=newdata(X1...X6,read->(x1...x6))
co=cov(x1...x5);
co=cov(dat%keep);
```

32.3 corr() Correlation MATRIX

corr(1) works similarly as cov()

32.4 regr() Linear regression

Ordinary or stepwise linear regrwession can be computed using regr().

output	1	REGR
	sRegression object..	
arg	1-N	LIST or REALV
	y-variable and x-variables variables listing them individually or given as a LIST.	
data	N|1-	DATA
	DATA object. Default last DATA defined.	
from	N|1	REAL

	First observation used.	
until	N|1	REAL
	Last observation used.	
trans	N|1	TRANS
	TRANS computed fro each observation	
epilog	N|1	TRANS
	TRANS computed after going through data.	
noint	-1|0	
	noint-> implies that the model does not include intercept	
step	-1|1	REAL
	t-value limit for stepwise regression. Regression variables are droped one-by-one until the absolute value of t-value is at least as large as the limit given. intercept is not considered.	
var	-1|0	
	if var-> is present regr() generated matrix]output%var[for the variance-covariance matrix of the coeffcient estimates.	
corr	-1|0	
	if vcorr-> is present regr() generated matrix]output%corr[for the correlation matrix of the coeffcient estimates. Standard deviations are put to the diagonal.	
variance	-1|1	CODE
	The variance of the residual error is proportional to the function given in this codeoption.	

Note 32.4.1. If the DATA contains variables Regr and Resid, then the values of the regression function and residuals are put into these columns. Space for these e coluns cab reserved with extra-> option in data() or in newdata()

Note 32.4.2. If re is the output of the regr() then function re() can be used to compute the value of the regression function. re() can contain from zero arguments up to the total number of arguments as arguments. The rest of arguments get the value they happen to have at the moment

when the function is called.

Information from the REGR object can be obtained with the following functions. let re be the name of the REGR object.

- coef(re,xvar) = coefficient of variable xvar
- coef(re,xvar,any->) = returns zero if the variable is dropped from the equation in the step-wise procedure of due to linear dependencies.
- coef(re,1) or coef(]re(,$1) returns the intercept
- se(re,xvar) standard error of a coeffcient
- mse(re) MSE of the regression
- rmse(re) RMSE of the regression
- r2(re) adjusted R2. If the intercept is not present this can be negative.
- nobs(re) number of observations used
- len(re) number of independent variables (including intercept) used

32.5 nonlin() Nonlinear regression

To be raported later, see old manual

32.6 varcomp() Variance and covariance components

TO BE RAPORTED LATER, see old manual

32.7 classify() Group means, variances and standard deviations

Classifies data with respect to one or two variables, get class frequencies, means and standard deviations of argument variables.

Output	1	Matrix		
	Output 1 Matrix A matrix containing class information (details given below)			
Args	-1-	REAL		

	Args -1- REAL Variables for which class means are computed. If there are no arguments, code option func-> must be present	
func	-1|1 CODE code telling how the variable to be classified is computed.	
xfunc	N | 1 CODE Code option telling how the x variable is computed from the data variables.	
data	N|1- DATA DATA object. Default last DATA defined.	
from	N|1 REAL First observation used.	
until	N|1 REAL Last observation used.	
trans	N|1 TRANS TRANS computed fro each observation	
epilog	N|1 TRANS TRANS computed after going through data.	
x	1 REAL The first variable defining classes.	
minobs	-1|1 REAL minimum number of observation in a class, obtained by merging classes. Does not work if z-> is given	
xrange	-1|0|2 Real Defines the range of x variable. If xrange-> is given without arguments and **Jlp22** variables x%min and x%max exist, they are used, and if they do not exist an error occurs. Note that these variables can be generate with stat(min->,max->). Either xtrange-> or any-> must be presente.	

any -1|0

Indicates that each value of the x-variables foms a separate class. either xrange-> or nay-> must be present.

---

tailtofirst -1|0

If the x-variable is less than the lower xrange, the observation is put to the first class

---

tailtolast -1|0

If the x-variable is greater than the upper xrange, the observation is put to the first class

---

classes -1|1 Real

Number of classes, If dx-> is not given, the default is that range is divided into 7 classes. minobs-> minimum number of observations in one class. Classes are merged so that this can be obtained. Does not work if z-> is present. !

---

z -1|1 REAL

The second variable (z variable) defining classes in two dimensional classification.

---

zrange -1|0|2 Real

Defines the range and class width for a continuous z variable. If **Jlp22** variables x%min and x%max exist, provided by stat(min->,max->), they are used.

---

dz -1|1 Real

Defines the class width for a continuous z variable. mean if z variable is given, class means are stored in a matrix given in the mean-> option classes number of classes, has effect if dx is not defined in xrangedx->. The default is classes->7. If z is given then, there can be a second argument, which gives the number of classes for z, the default being 7.

---

trans -1 | 1 TRANS

transformation set which is executed for each observation. If there is a transformation set associated with the data set, those transformations are computed first.

---

filter	-1 | 1	Code
	logical or arithmetic statement (nonzero value indicating True) describing which observations will be accepted. trans-> transformations are computed before using filter.	
reject	-1 | 1	Code
	logical or arithmetic statement (nonzero value indicating True) describing which observations will be rejected. trans-> transformations are computed before using reject->.	
print	-1|1	Real
	By setting print->0, the classification matrix is not printed. The matrix can be utilized directly in drawclass() function.	

Note 32.7.1. If z variable is not given then first column in printed output and the first row in the output matrix (if given) contains class means of the x variable. In the output matrix the last element is zero. Second column an TARKASTA VOISIKO VAIHTAArow shows number of observations in class, and the last element is the total number of observations. Third row shows the class means of the argument variable. The fourth row in the output matrix shows the class standard deviations, and the last element is the overall standard deviation

Note 32.7.2. Variable Accepted gets the number of accepted obsevations.

Note 32.7.3. Note the difference of xrange-> in classify() and draw-> when xfunc-> is present. In draw->, x-> and xrange-> refer to the background variable which is stepped in xrange-> and the x-variable in figure is then computed using xfunc->. In classify() xrange-> and xfunc refer to the same variable.

32.8 class() Class of a given value

Function class() computes the class of given value when classifying values similarly as done in classify().

Output	1	REAL
	The class number.	
Args	1	Real
	The value whose class is determined.	

xrange	2	Real
	The range of values.	
dx	N|1	Real
	The class width.	
classes	N|1	Real
	The number of classes.	

Note 32.8.1. Either dx-> or classes-> must be given. If both are given, dx-> dominates.

Note 32.8.2. If stat() is used earlier for variables including Var1 and options min-> and max-> are present, then xrange->(Var1%min,Var1%max) is assumed.

33 Linear programming

This chapter decribes the key points of the Jlp-algoritm and the available **Jlp22** functions.

33.1 JLP linear programming algorithm

This section will later describe more closely the theoretical aspects of the linear programming functions. Now only some key concepts are listed. Now the reader is referred to the old JLP manual jlp92.pdf, the description of the factory optimization in lappilempinen.pdf, and an application of the factory optimization in hyvonenetal2019.pdf

The development of JLP algorithm !

- JLP
  - The JLP algorithm was designed for forest management planning problems having the following structure. A simulator generates a large number of treatment schedules for each stand in a group of stands.
  - In an optimization problem a linear combination of sums of variables (e.g. amount of harvested sawlogs during a period) produced in treatments is maximized or minimized subject to constraints which are also linear combinations of variables.
  - The key point is the generalized upper bound (GUB) technique of Dantzik and Van Slyke (see gub.pdf) which I reinvented. The key idea is to remove with algebraic tricks the area constraints.

- In a optimization problem with schedules generated for each stand, there must be a area constraint for each stand telling that shares of schedules add up to one in each stand.
- In the GUB technique, the area contraints can be dropped with slight overhead cost. In a typical problem,the data can consist of 10000 stands, and and optimization problem can consist of 10 contraints for the sums ofe variables. Using an ordinary LP software in one iterative optimization step, a matrix consisting of 100200100 elements is updated. In JLP the equivalent step is computed by updating a matrix consisting 100 elements.
- The heuristic algorithm of Hoganson and Rose (1984) and Hoganson and Kapple (1991) leads to similar computations where a schedule in a stand is selected using shadow prices. Their shadow prices are updated heuritically but JLP algorith updates the prices using the linear programming theory.
- The key concept in GUB is the key variable, in our case the key schedule, which is any of the schedules which has nonzero weight.
- JLP algorithm is also using the ordinary upper bound technique. If there is both lower and upper bound in a contraint, the standard theory assumes that the lower bound and upper bound constraints are presented as different constraints. In the ordinary upper bound technique there is only one constraint, and the algorithm keeps either of the bounds as the active bound.
- JLP used own matrix subroutines which computed all the time the explicit inverse of the basis matrix.
- When a constraint is not binding, in standard algorithms, a residual variable tells the difference between the contraint row and the active bound. JLP reduced the dimension of the basis matrix, which was a feasible solution because the explicit inverse of the basis was used.
- JLP introduced domains, i.e.. subsets of stands into the problem defition. They decrease the memory needs and help to formulate more reasonable problems. Their utility to define spatial constraints is not fully utilized.
- JLP allowed to compute new variables from the simulated variables. I have suggested (in vain) since 1992 that Mela would compute only forest variables and economic variables would be feeded in JLP. This way it would not be necessary to resimulate when trying different prices.
- JLP offered the possibility to duplicate a schdedule. GAYA group has utilized this by duplicating a schdeule so tha that in one copy of a no-harvest schedule the final stae has postive net presetn value and anothe copy does not.

- JLP was written in Fortran77, which did not have dynamic allocation, pointers or own data types.
- In writing the JLP algorithm, oaly very small part of the working time went to get the algorithm work correctly mathematically. The most difficult thing was the fight against rounding errors, i.e. trying to infer whther a small number is zero or not, and whether a number is really larger than another number even if the computer values indicated that.
- JLP had a precompiler which could be used to handle compiler options and the sizes of vectors. Fortran77 did not have dynamic alloacation of vectors.

- **J**, the successor for JLP
  - Version 0.9.3 was published in 2004
  - Initially the same JLP algorithm but **J** was using matrix routines of Fletcher which were changed into double precision. Flethcer's code was written in Fortran77.
  - **J** was written in Fortran90 which has dynamic allocation of vectors, pointers etc.
  - New tools for preprocessing of data and postprocessing of results, and these computations can be used also without LP problems.
  - Mela has not included **J** as as the optimizer, but it is possible to pull out simulated schedules from Mela and then define and solve linear programming problems using **J**. item **J** version 2.0. published in 2013 made it possible optimize simultatenously forestry and transportations to factories and factory production. Theory is published in Lappi and Lempinen 2014 Paper explaining the theory behing factory optimization.
  - The generalized upper bound tehcnique is extended for constraints which tell that all harvested timber volume in a stand is transported somewhere.
  - The key idea is to have in each stand and for all harvested log types and for all periods a key factory to which some timber is tranported provided that the cuttings produce that log type for that period. enditemize
  - DTRAN algoritm of Hoganson is an competitor of **Jlp22** because it can deal with factories also. !

33.2 Problem definition object

Problem definition object is a compound object produced by the problem() function, and it is described in Linear programming.

33.3 problem() PROB for jlp() and jlpz()

An LP-problem is defined in similar way as a TEXT object. The following rules apply for problem rows:

- On the left there is any number of terms separated with + or -.
- Each term is either a variable name or coefficient*variable.
- A coefficient can be
  * a number
  * Computation code inside parenthesis. These coefficients are computed within problem().
  * computation code within apostrophes. These coefficients are computed in jlp(), jlpz() or jlpcoef() functions.
  * A legal name for an object.
- The variable must be a legal object name. The optimization variable can either be a z-variable or x-variable. A x-variable is an variable in schedules data set. In jlpz() all variables are z variables. Function jlpz() unpacks lists to z-variables.
- The right side of the first row ends either by ==min or ==max.
- On the right side for other rows there is a number or code for computing a numeric value within parnthesis or within apostrophes or a variable name. Numbers within parenthesis are computed within problem() but numbers within apostrophes are computed in jlp(), jlpz() or jlpcoef() functions. Between the left side and the right side there is

  * >Low <Up
  * >Low
  * <Up
  * = Value

Low, Up and Value can be

- Numeric constant
- Text within apostrophes. This value is interpreted later in jlp() or jlpz()
- A REAL variable. The value is looked later in jlp() or jlpz()
- A REAL variable preceded by '-' or '+'. The value is obtained in jlp() or jlpz()
- Text with or without surrounding parenthesis. This value is computed now.

Sign < means less or equal, and > means greater or equal. Pure less or greater would be meaningless in this context.
If there two different identical rows the other having '<' and the other '>', an error occurs, because the solution is obtained faster that way. If all rows have both lower limit and upper limit, the solution is obtained in half time when merging the lines.

In problems with x-data, there can be domain rows, which tell for what subset of the treament units the following constraints apply. Domains are defined using c-variables, i.e. variables in the unit data, or in nonhierarchical, flat data set, the value of the c-variable is obtained from the first observation where the variable given in unit-> gets a different value than in the previous observation. The variables in the flat data file having the same value for all observations in the same unit are called also c-variables. Later there will be variables related to factory problems. A domain definition ends with ':'. In a domain row there can be any number of domain definitions. There are three diffent kinds of domain definitions
All indicates all units. This domain is assumed to all rows before the first domain row.
c-variable, a nonzero value tells that the unit belongs to the domain.
A piece of code which tells how the indicator is computed from the c-variables. A nonzero value indicates that the unit belongs to the domain. Recall tha logical operations produce 1 for True and 0 for false. The code is parsed at this point, so syntax errors are detected at this point, but other errors (e.g. division by zero) are detected in jlp().

If a coefficient is a column vector and the variable is a x-variable in the objective row, this indicate a piecewise linear objective. The if the value of the x-varaible is smaller than the first value in the vector, the second value gives the coeffient. If the x-variable is between the first and third value, then the fourth value gives the coeffient which applies after the third value. The value is obtained using the value at the first knot plus the value coming from the second section. A nonlinear objective can be devloped quite easily using the same data structures and the derivative computations already present in **Jlp22**. An example is presented for optimization of sawmill sets, for which also the new properties of data() function are developed.

Output	1	PROB
	the PROB object created	
print	0|1	REAL
	If print-> gives a value, then values >2 tell that the problem is printed (default)	

Note 33.3.1. Examples are give in connection of jlpz() and jlp().

Note 33.3.2. the cofficients in a PROB can be interpreted also using jlpcoef() function, which is used also by jlp() and jlpz() functions.

Note 33.3.3. Note Problems without x-variables can be solved also without problem() function by feeding in the necessary matrices.

33.4 jlp() for schedules DATA

jlp() solves linear programming problems. The function now assumes that there is schedules data. Without schedules jlpz() must be used.

Output 1

Output tells how objects created by jlp() are named. There is no JLP object type, but the output indicates that e.g. the following objects are created. Many other objects are created but they are currently used for debuggging purposes and they will be described later. They can be used also to teach how the algorithm proceeds.

- Output%weights The weights of the schedules, see teh example below.
- Output%objective= value of the objective function
- Output%rows= the vector the valuef of the constraint rows.
- Output%shprice = vector of shadow prices of the rows
- Output%xvar= LIST of xvariables in the schedules data Output%xvarproblem= LIST of x-variables in the PROB
- Output%xsum= Vector of sums of variables in Output%xvarproblem
- Output%xprice Shadow prices of the variables in Output%xvarproblem
- Output%xsumint The sums of x-variables in the integer approximation, generated if integer-> is present

---

problem 1 PROB

Problem object produced with problem()

---

data 1 DATA

Schedules data (Sic!) linked to it with unitdata if unit-> is not given

---

unit -1-99 REAL

gives the unit variables from which at least one changes when unit changes.

area	-1|1	REAL
	The values in variables in xdata are per area variables, except variables given in notareavars-> option	
z	-1|0	
	This option must be present when there are z-variables in the problem, but the z-variables need not to be listed. The reason for this option is that often the purpose is to define the problem using only x-variables, but due to typing errors all variables are not among x-variables.	
showdomain	-1-99	CHAR
	the sums of the x-variables are computed also for these domains.	
print	-1:1	REAL
	print-> set printing level to 2, print->value set the printling level to value, where zero indicates no printing. Default level is 1.	
debug	-1|0|1	REAl
	debug-> sets debugging on at start debug->value sets debugging on when pivot=value, After the debugging pivot, **Jlp22** generates pause() and during the pause the user can do any computations. Before the pause some additional matrices are generated in addition to matrices which are used the computations.	
stop	-1|1	CODE
	codeoption telling when iterations over units stop. The variables Round (current number of rounds over units), Change% (change of objective during the last 10 rounds), Imp (number of entering schedules outside the active set when updating the active set), Active% (%-size of the active set) and all global variables in JLP22. default is stop->(Change%.lt.0.01.and.Round.ge.	
fast%	-1|1	CODE
	codeoption computing Fast%. All schedules whose price is larger than Fast% of the current key schedule. Same variables can be used as for stop-> and also current Fast%. A possible rule is fast%->(min(Fast%+5-(Imp.gt.0)*10,98))). The default is Fast%=85.	

maxrounds%	-1|1	REAL
	maximum number of rounds over all units. Default maxrounds->3000	
report	-1|1	CHAR
	the results are written to the file spesified.	
echo|-1|0		
	When results are printed to a file, echo-> implies that they are written alo to the terminal.	
refac	-1|1|	REAL
	refac->value tells that the factors of the basis matrix are recomputed after value pivot operations. The default is refac->1000.	
tole	-1|1	REAL
	tole->value tells that the default tolerances are multiplied with the value.	

Note 33.4.1. The data-> must now always refer to schdedules data (for several reasons)

Note 33.4.2. In small problems dCPU, i.e. increase of used CPU time is not very accurate.

Note 33.4.3. jlp() stores the sums of x-variables into output% -variables. If there are domains or showdomains, the variable names get postfix [domaindefinition], show examples below

Example 33.1 (jlpex). jlp() solves linear programming problem

```
 cdata=data(in->'cdat.txt',read->(cdata%nobsw,site))
stat()
xdata=data(in->'xdat.txt',read->(npv#0,npv#5,income1...income5),up->cdata
stat()

proba=problem();
** In this problem the 4% net present value at the beginning is maximixe
** subject to the constraints telling that net incomes are nondecreasing
npv#0==max
;do(i,2,5)
income"i"-income"i-1"=0
;enddo
npv#5-npv#0>0
/
plist=;list(proba%?);
@plist;
```

```
jlpa=jlp(problem->proba,data->xdata,showdomain->('site.le.3','site.gt.3')
** sums of x-variables are stored into the same variables with jlpa%
prefix
jlpa%income5;
jlpa%income5[site.le.3];
** Note income variables are in theory equal but as their
** values are computed numerically, they differ.
jlist=;list(jlpa%?);
** these could be printed with @jlist;


** jlpa%weights gets the weights of schdedules
** combain the weights with the data
xdataw=newdata(xdata,jlpa%weights,read->w)
stat(sum->)
**sum of weights is equal to the number of stands
w%sum;
** weighted statistics
** thesw agree with the jlp solution
stat(weight->w,sum->)
;do(i,1,len(xdataw%keep))
@xdataw%keep(i)%sum;
;enddo
***Problem with domains
probb=problem();
npv#0==max
**Domain definitions:
**there can be several domain definitions on a row
** one domain definition is:
** a logical statement in terms of stand variable
** a stand variable whose nonzero value implies that the domain applies
** All indicates all stands.
** before first domain definition row the default Domain is All
site.le.3:site.gt.3:
;do(i,2,5)
income"i"-income"i-1"=0
;enddo
npv#5-npv#0>0
/
plistb=;list(probb%?);
*** these could be printed with @plistb;
jlpb=jlp(problem->probb,data->xdata,showdomain->'3.lt.site')
jlpb%income5;
jlpb%income5[site.gt.3];
jlpb%income5[3.lt.site];
Continue=1
** Now problem without constraints
probc=problem();
npv#0==max
/
jlpc=jlp(problem->probc,data->xdata,showdomain->'3.lt.site')
Continue=0
```

33.5 jlpz() for an ordinary Lp-problem.

The problem defined in the problem() function can be given in problem-> or by giving values for max-> or min->, and zmatriz->, rhs-> and rhs2-> options.

Output	1	
	Theres is no jlpz -object, but the output is used to name several objects created with the function. The list of created objects can be seen with outlist=;list(Output%?); The objects can then be be seen with @outlist; The objects created can be used in debugging the algorithm and also in teaching how the alogorithm proceeds. This will demonstrated later.	
problem	-1|1	PROB
	Problem defined in problem(). If problem-> is not present the following 3 options must be present and either min-> or max->.	
zmatrix	-1|1	MATRIX
	Constraint matrix.	
rhs	-1|1	MATRIX
	Lower bounds as row or column vector having as many elements as there are rows in the matrix given in zmatrix->. Use value -1.e20 to indicate that there is no lower bound.	
rhs2	-1|1	MATRIX
	Upper bounds as row or column vector having as many elements as there are rows in the matrix given in zmatrix->. Use value 1.e20 to indicate that there is no upper bound.	
max	-1|1	MATRIX
	The objective vector for a maximization problem. It must have as many elements as the constraint matrix has columns.	
min	-1|1	MATRIX

	The objective vector for a minimization problem.	
dpivot	-1|1	REAL
	The objective function etc are printed after dpivot pivots.	
debug	-1|0|1	REAL
	NOT UP TO DATE Gives the value of Pivot at which a pause is generated. During the pause all essential matrices can be studied. Pure debug-> is the same as debug->0, which implies that pause is generated before pivoting. If variable Debug is given a new value, the the next pause is generated when Pivot.eq.Debug. The default is that the next pause is generated after the next pivot. The pause() function can now use also lis=;list(Output%?); even if ;list is input programming function which are not otherwise allowed during pause(). !	

Note 33.5.1. The ordinary Lp-algorithm can be taught using matrices generated in pause to show how the algorithm proceeds.

Example 33.2 (jlpzex). Problem with only z-variables.

```
  probza=problem()
2*x1+x2+3*x3-2*x4+10*x5==min
x1+x3-x4+2*x5=5
x2+2*x3+2*x4+x5=9
x1<7
x2<10
x3<1
x4<5
x5<3
/
probzalist=;list(probza%?);!subobjects created
@probzalist;!printing the subobjects

jlpza=jlpz(problem->probza,dpivot->1)
jlpzalist=;list(jlpza%?);
@jlpzalist;
** The same problem is defined using different tools available.
!
probzb=problem()
2*x1+x2+x34c*x34+10*x5==min
x1+x3-x4+(2+0)*x5=5
x2+2*x3+2*x4+x5=9
x1<7
x2<i10
x3<'1+zero'
```

```
x4<5
x5<3
/
x34=list(x3,x4)
x34c=matrix(1,2,values->(3,-2))
i10=10
zero=0
jlpzb=jlpz(problem->probzb,dpivot->1)
**Now different problem is obtained
x34c=matrix(1,2,values->(3,-3))
zero=1
jlpzb=jlpz(problem->probzb,dpivot->1)
**
**The matrices needed to use jlpz without problem-> can be obtained
from a problem as follows
jlpcoefa=jlpcoef(probza)
jlpcoefalist=;list(jlpcoefa%?);
jlpza=jlpz(zmatrix->jlpcoefa%matrix,rhs->jlpcoefa%rhs,rhs2->jlpcoefa%rhs2
```

33.6 jlpcoef() PROB into numeric form

This function is used at the beginning of jlp() and jlpz().

Note 33.6.1. see jlpz() for how jlpcoef() can be used to demonstate problem defintion using matrices only.

34 Plotting figures

The graphiscs of the current version of **Jlp22** is produced with gnuplot, see `http://www.overleaf.com` **Jlp22** offers an alternative interface to gnuplot, and it is quite easy to add more ploting routines later.

34.1 show(), show-> and continue->

By default, plotting functions plots (shows) FIGs immediately. When the final figure consists of several subfigures, it is convenient to prevent plotting using show->0. In developing scripts, it is convenient to switch with show->. This can be done by giving option in form show->showfig, then the ploting can be controlled with REAL showfig.

Withing show() it is possible to define or redefine xlabel->, ylabel->,title->, xrange-> and yrange->

By default **Jlp22** generates pause() after showing a FIG. During pause(), the user can give any commands. If the user types 'e', then **Jlp22** generates a error condition causing return

to to sit>.

It is possible to collect a FIG first (e.g. using loops), and show then the FIG using show() function. The argument of show() can be either a FIG or the name of the .jfig file withing apostrohes. Thus it is possible to edit the file using gnuplot capabilities.

Example 34.1 (showex). Example of show()

```
 fi=draw(func->sqrt2(x),x->x,xrange->(-50,50),continue->fcont)
show(fi,xrange->(-60,60),xlabel->'NEWX]Times Roman,16[',ylabel->'NEWY]Cour
show(fi,axes->10,continue->fcont)
show(fi,axes->01,continue->fcont)
show(fi,axes->00,continue->fcont)
Window='400,800'
show(fi,continue->fcont)
Window='700,700'
fi=drawline(1,10,3,1,color->Red,continue->fcont)
*** The line is not visible
show(fi,xrange->(1.1,11),continue->fcont)
dat=data(read->(x,y),in->)
1,4
2,6
3,2
5,1
/
stat()
*** gnuplot hides points at border
fi=plotyx(y,x,continue->fcont)
** The ranges needs to be adjusted manually
show(fi,xrange->(0,6),yrange->(0,7),continue->fcont)
```

34.2 Font in title->, xlabel->, ylabel->, label->

It is possible to change the the font by appending the the font defintion between][at the end of the label, e.g.,
xlabel->'My-xlabel]Times-Roman,15['
where 15 is the size of the font. Available fonts vary from system to system. Web provides further inforamtion, see e.g https://\textcolor{teal}{gnuplot}.sourceforge.net/docs_4.2/node356.html
According to this page, the following fonts may be available Helvetica Helvetica Bold Helvetica Oblique Helvetica Bold Oblique
Times Roman Times Bold Times Italic Times Bold Italic
Courier Courier Bold Courier Oblique Courier Bold Oblique
Symbol
Hershey/Cartographic_Roman Hershey/Cartographic_Greek Hershey/Simplex_Roman Her-

shey/Simplex_Greek Hershey/Simplex_Script
Hershey/Complex_Roman Hershey/Complex_Greek Hershey/Complex_Script Hershey/Complex_Italic Hershey/Complex_Cyrillic Hershey/Duplex_Roman Hershey/Triplex_Roman Hershey/Triplex_Italic
Hershey/Gothic_German Hershey/Gothic_English Hershey/Gothic_Italian Hershey/Symbol_Set_1 Hershey/Symbol_Set_2 Hershey/Symbol_Math
ZapfDingbats. space within font name can be replaced with '-', e.g. Times-Roman.

34.3 Scandic and other special characters

Scandic charcters can be put into figures as follows. Make first the figure without scandic characters e.g using characters without the dots. Let the name of the generated FIG object be fig. Open the file fig.jfig. Change the coding of the file into UTF-8 (under Encoding in Notepad++). Add the scandic characters to xlabel, ylabel, title, or label -parts in the header. Save the file. Give at sit> prompt the command show('fig.jfig').
Subscript is made with the underscore before the subscript.
Underscore is put into the labels with three backslash followed with the the underscore.
Superscript is made with ^before the character.
Character^can be put into the text with three backslash followed with ^
Longer subscripts or superscripts can be made using around the scripts.
For other things see gnuplot documentation.

34.4 color->Black, Red, Green, Blue, Cyan , Violet, Yellow, Orange

gnuplot allows arbitrary color definitions. Currently the above colors can be used. General color definitions are implemented when requested.

34.5 Terminal

It is possible to change the terminal type used by gnuplot by giving the name of the terminal to the predefine CHAR variable Terminal. The default is
Terminal='qt'.

34.6 Window size and shape

It is possible to change the window size and shape of gnuplot by using the predefined CHAR variable Window. The default is
Window='700,700'

34.7 Legends

- The default legends by gnuplot do not look nice, and they are not implemented. The user can write own legends using label-> option in drawline().

34.8 plotyx() Scatterplot

plotyx() makes scatterplot.

Output	1	FIGURE
		The FIGURE object created or updated.
Args	1 | 2	REAL
		y and x-variable, if func-> is not present. In case y-variable is given with func->only, x-variable is given as argument.
data	N | 1	DATA
		Data object used, default the last data object created or the dta given with data=list().
append	N | 0	
		The graph is appended to an existing FIGURE. If the output is does not exist beforehand or it is not a FIGURE, then the figure is (and not error) is generated.
continue	N|0	
		Often sequentila figures are made using include files. It would be diffuct to keep track of the figures if **Jlp22** continues the **Jlp22** script. Thus the default is that after plotting the figure, **Jlp22** excutes pause() function. The user can give any commands during the pause. Typing <return>, the exceution continues. Typing 'e' or 'end', an error is generated, and the control returns to sit> promt. If the function contains continue-> option, pause() is not generated.
xlabel	N | 1	CHAR
		Label for x-axes. Default is the name of the x-variable.
ylabel	N | 1	CHAR

Label for y-axes. Default is the name of the x-variable.

---

xrange N |1-2 REAL

the range for x-axes. gnuplot generates it automatically, but sometimes the range used by gnuplot needs to be changed.

---

yrange N |1-2 REAL

the range for y-axes. gnuplot generates it automatically, but sometimes the range used by gnuplot needs to be changed.

---

color N | 1 REAL

The color used. The color indices are put to following predefined REAL variables: Black, Red, Green, Blue, Cyan , Violet, Yellow, Orange.

---

show N | 1 REAL

show->0 indicates the that the figure is not yet plotted. As the value 0 can be given also as a variable or it can be computed, the same code can produce different showing combinations.

---

axes N | 1 REAL

Are axes drawn.

- axes->11 Both axes are drawn. (Default)
- axes->10 x- axes is drawn, y-axes not.
- axes->01 x- axes is drawn, y-axes not. ∼ axes->1.
- axes->00 Neiher axes is drawn. ∼ axes->0.

---

mark N | 1 REAL | CHAR

The mark used in the plot. Numeric values refer to mark types of gnuplot. The mark can be given also as CHAR varible or constant.

---

func N | 1 CODE

Code option telling how the y-variable is computed.

---

xfunc N | 1 CODE

Code option telling how the x-variable is computed.

Note 34.8.1. By default plotyx() uses the names of the x-variable and y-variable in the xlabel and ylabel. If the names contain e.g. underscore _, this would make the next character as under script. To plot the underscore character, use xlabel-> and ylabel->, and replace _ with

_.

Note 34.8.2. There can be both func-> and xfunc->

Note 34.8.3. If there is func-> and not xfunc, the x-variable must be before func-> as arguments are always before options.

Note 34.8.4. The default axes of gnuplot may hide points. This must be solved somehow later, but currently the user should define axes ranges explicitly in such cases, especially with plots with few points.

Example 34.2 (plotyxex). plotyx()

```
 ** plotyx() (line 16019 file c:/jlp22/j.f90)
xmat=matrix(do->(0,10,0.01))
transa=trans()
y=2+3*x+0.4*x*x+4*rann()
/
datyx=newdata(xmat,read->x,maketrans->transa,extra->(Regf,Resid))
fi=plotyx(y,x,continue->fcont)
** It is not necessary to put the function into the data
fi=plotyx(x,func->transa(y),mark->3,color->Orange,continue->fcont)
**look other way, rotate axes
fi=plotyx(x,xfunc->transa(y),mark->3,color->Orange,continue->fcont)
reg=regr(y,x)
** ranges are stored in order to use them in classify()
stat(min->,max->)
figyx=plotyx(y,x,show->0)
**It would be possible to draw the regression function using draw()
figyx=plotyx(Regf,x,append->,continue->fcont,color->Blue)
fir=plotyx(Resid,x,continue->fcont)
** When there are many observations, it is useful to compute class means
of residuals
cl=classify(Resid,x->x,xrange->,dx->1)
** Red error bars for standard deviations of residuals in calsees
fi22=drawclass(cl,sd->,mean->,continue->fcont,color->Red,show->0)
** Black error bars for standard errors for class means.  These can be used
to
** study how significantly residuals are nonzero.
fi22=drawclass(cl,se->,mean->,color->Black,width->2,append->,continue->fcont,
xlabel->'Xvariable,m]Arial,11[',
ylabel->'Yvariable,%]Arial,11[')
;if(wait);pause
```

Note 34.8.5. With data with integer values, the default ranges of gnuplot may be hide point at borderlines.

Note 34.8.6. fi=plotyx() produces or updates file fi.jfig] which contains gnuplot commands and file fi.jfi0 containg data.

34.9 draw() Draws a function

draw() draws a function.

Output	1	FIGURE
		The FIGURE object created or updated.
func	N | 1	CODE
		Code option telling how the y-variable is computed.
xfunc	N | 1	CDE
		Code option telling how the x-axes variable is computed from the x-variable.
xfuncrange	N|2	REAL
		Ranges for the x-axes when xfunc-> is present and xrange is used for the stepping of the x-> variable.
points	N|1	REAL
		Number of equal distance points used in xrange->
x	1	REAL
		Variable which is stepped before computing the value of func->.
xrange	0-4	REAL
		Range for the x-variable. If there are now arguments **Jlp22** tries to find variables var%min and var%max which where computed with stat() with options min-> and max->. If there is need to define a smaller range for computing the values as wished for the x-axes, then
mark	N | 1	REAL | CHAR
		The mark used in the plot. Numeric values refer to. mark types of gnuplot. The mark can be given also as CHAR varible or constant.
width	0 | 1	REAL

the width of the line

Example 34.3 (drawex). Example of draw()

```
 fi=draw(func->sin(x),x->x,xrange->(0,2*Pi),color->Blue,continue->fcont)
fi=draw(func->cos(x),x->x,xrange->(0,2*Pi),continue->fcont,color->Red,append-
** Circle
fi=draw(func->sin(u),xfunc->cos(u),x->u,xrange->(0,2*Pi),color->Red,continue-
**Spiral with too few points
fi=draw(func->(0.1*u*sin(u)),xfunc->(0.1*u*cos(u)),x->u,xrange->(0,100),color
**Spiral with more points
fi=draw(func->(0.1*u*sin(u)),xfunc->(0.1*u*cos(u)),x->u,xrange->(0,100),point
continue->fcont,color->Orange,width->2)
;if(type(figyx).ne.FIGURE)plotyxex
show(figyx,continue->fcont)
reg0=regr(y,x)
stat(data->datyx,min->,max->)
figyx=draw(func->reg0(),x->x,xrange->,color->Violet,append->,continue->fcont)
transa=trans()
x2=x*x
/
reg2=regr(y,x,x2,data->datyx,trans->transa)
transa=trans()
x2=x*x
fu=reg2()
/
Continue=1 !Error
figyx=draw(func->transa(fu),xrange->,color->Orange,append->,continue->fcont)
continue=0
figyx=draw(func->transa(fu),x-x,xrange->,color->Orange,append->,continue->fco
fi=draw(func->sin(u),xfunc->cos(u),x->u,xrange->(0,0.5*Pi),continue->fcont)
**there is some problem with xrange
Continue=1 !Errors
fi=draw(func->sin(x),x->x)
fi=draw(xrange->(1,100),func->Sin(x),x->x)
Continue=0
```

Note 34.9.1. When xfunc-> is present the the variable given in x-> is stepped in range given in xrange-> and x-axes variable is computed using xfunc->.

Note 34.9.2. fi=draw() produces or updates file fi.jfig] which contains gnuplot commands and file fi.jfi0 containg data. If scandic characters are needed for labels, the edit these characters into fi.jfig and use show('fi.jfig')

Note 34.9.3. It is possible to have different range for the x-axes range and range for stepping the x-variable. This is useful e.g. when the function is not defined for zero, which would be nice lower in limit for the axes. I need to check how this works.

Note 34.9.4. Note the difference of xrange-> in classify() and draw-> when xfunc-> is present.

In draw->, x-> and xrange-> refer to the background variable which is stepped in xrange-> and the x-variable in figure is then computed using xfunc->. In classify() xrange-> and xfunc refer to the same variable.

34.10 drawclass() Draws results of classify()

drawclass() can plot class means and/or lines connecting class means, with or without standard errors of class means, within class standard deviations, within class variances, frequency histograms, which can be scaled so that density funtions can be drawn in the same figure.

Output	1	FIGURE FIGURE object updated or generated.
Arg	1	MATRIX A MATRIX generated with classify().
se	N | 0	Presence of option tells to include that error bars showing standard errors of class means computed as sqrt(sample_within-class_variance)/number_of_obs
sd	N | 0	Within-class standard deviations are drawn, if there is no mean-> option. If there is also mean-> option then sd-error line are drawn around mean similarly as with se-> standard error lines are drawn. Both se error bars and sd errors bars can be obtained by first making figure with sd error bars with some color and then calling drawclass() with different color, possibly with larger width->, and with append->, see the example below.
mean	N | 0	Class measn are drawn. Has effect only together with sd->.
var	N | 0	Within-class sample variances are drawn.
histogram	N | 0	Within-class sample variances are drawn.
freq	N | 0	

		Absolute frequences are drawn in histogram. Default percentage if area-> is not present
area	N |0	
		the histogram is scaled so that that it can be overlayed to density function
cumulative	N|0	
		cumulativer histogram is drawn. If freq-> is presented then absolute cumulative frequences are drawn, otherwise cumulative percentages are draw, except if also area-> is presenet then then cumulative realtive freaquences are drawn.

Example 34.4 (drawclassex). Examples of drawclass()

```
 X=matrix(do->(1,100,0.1))
e=matrix(nrows(X))
e=rann()
X2=0.01*X*.X!elementwise product
Y=2*X+0.01*X2+(1+0.3*X)*.e !nonequal error variance,quadratic function
dat=newdata(X,Y,X2,read->(x,y,x2),extra->(Regf,Resid))
stat(min->,max->)
reg=regr(y,x)!  Regf and resid are put into the data
fi=plotyx(y,x,continue->fcont)
fi=drawline(x%min,x%max,reg(x%min),reg(x%max),width->3,color->Cyan,append->,c
cl=classify(Resid,x->x,xrange->,classes->5)
fi=drawclass(cl,color->Blue,continue->fcont)
fi=drawclass(cl,mean->,sd->,continue->fcont,color->Red)
fi2=drawclass(cl,se->,continue->fcont)
fi=drawclass(cl,se->,continue->fcont,color->Black,width->2,append->)
fi=drawclass(cl,sd->,continue->fcont)
fi=drawclass(cl,var->,continue->fcont)
** x-values were equally distributed due to data construction
fi=drawclass(cl,histogram->,area->,continue->fcont)
fi=draw(func->pdf(0,rmse(reg)),x->x,xrange->,append->,continue->fcont)!
xrange comes from stat()
```

Note 34.10.1. In previous versions of **Jlp22** if se-> and sd-> were both present, the error both bars were plotted. This possibility will be included later.

34.11 drawline() Draws a polygon through points.

Output	1	FIGURE
		The FIGURE object created or updated.

Args	1-	REAL | MATRIX

The points which are connected:

- x1,...,xn,y1,...,yn The x-coordinates and y-coordinates, $n \geq 1$*Ifthereisonlyonea*

- If there are two matrix (vector) arguments, then the first matrix gives the x-values and the second matrix gives the y-values. It does not matter if arguments are row or column vectors.

append	N | 0	

The graph is appended to an existing FIGURE. If the output is does not exist beforehand or it is not a FIGURE, then the figure is (and not error) is generated.

continue	N|0	

Often sequentila figures are made using include files. It would be diffuct to keep track of the figures if **Jlp22** continues the **Jlp22** script. Thus the default is that after plotting the figure, **Jlp22** excutes pause() function. The user can give any commands during the pause. Typing <return>, the exceution continues. Typing 'e' or 'end', an error is generated, and the control returns to sit> promt. If the function contains continue-> option, pause() is not generated.

xlabel	N | 1	CHAR

Label for x-axes. Default is the name of the x-variable.

ylabel	N | 1	CHAR

Label for y-axes. Default is the name of the x-variable.

xrange	N |1-2	REAL

the range for x-axes. gnuplot generates it automatically, but sometimes the range used by gnuplot needs to be changed.

yrange	N |1-2	REAL

the range for y-axes. gnuplot generates it automatically, but sometimes the range used by gnuplot needs to be changed.

color	N | 1	REAL

The color used. The color indices are put to following predefined REAL variables: Black, Red, Green, Blue, Cyan , Violet, Yellow, Orange.

---

show N | 1 REAL

show->0 indicates the that the figure is not yet plotted. As the value 0 can be given also as a variable or it can be computed, the same code can produce different showing combinations.

---

axes N | 1 REAL

Are axes drawn.

- axes->11 Both axes are drawn. (Default)
- axes->10 x- axes is drawn, y-axes not.
- axes->01 x- axes is drawn, y-axes not. ~ axes->1.
- axes->00 Neiher axes is drawn. ~ axes->0.

---

label N | 1 CHAR

Label written to the end of line. If arguments define only one point, then with label-> option one can write text to any point.

---

step N|1 REAL

Points taken from matrices are taken at steps given in step->

---

mark N | 1 REAL | CHAR

The mark used in the plot.

---

pointsonly N|0

Only points are drawn.

---

break N | 0

The line is broken when a x-value is smaller than the previous one.

---

set N|1 REAL<6

Set to which lines are put. If the option is not present, then a separate gnuplot plot command with possible color and width information is generated for each drawline() and data points are stored in file fi.jfi0, i.e. the same file used by plotyx(). If set is given e.g as set->3, then it is possible

		to plot a large number of lines with the same width and color. The data points are stored into file fi.jfi3. This is useful e.g. when drawing figures showing transportation of timber to factories for huge number of sample plots. Numeric values refer to gnuplot mark types. The mark can be given also as CHAR varible or constant.
width	0 | 1	REAL the width of the line. Default: width->1
label	N |1	CHAR Text plotted to the end of line.

Note 34.11.1. Both fi=drawline(x,y,mark->'*',append->) and fi=drawline(x,y,label->'*',append->) can be used to put the mark into a figure, but mark is centered at the point but for a label x and y give the upper-left coordiantes.

Example 34.5 (drawlineex). Example of drawline()

```
 fi=draw(func->sin(x),x->x,xrange->(0,2*Pi),color->Blue,continue->fcont)
fi=drawline(Pi,sin(Pi)+0.1,label->'sin()',append->,continue->fcont)
xval=matrix(do->(1,10));
mat=matrix(values->(xval,xval+1,xval,xval+2,xval,xval+3))
fi=drawline(mat,color->Red,continue->fcont)
fi=drawline(mat,color->Orange,break->,continue->fcont)
xm=matrix(do->(0,100,1))
e=matrix(101)
e=rann(0,3)
ym=2*x+0.3*xm*.xm+0.4+e
dat=newdata(xm,ym,read->(x,y),extra->(Regf,Resid))
reg=regr(y,x)
figyx=plotyx(y,x,continue->fcont)
figr=plotyx(Resid,x,continue->fcont)
reg0=regr(y,x)
stat(min->,max->)
figyx=draw(func->reg0(),x->x,xrange->,color->Violet,append->,continue->fcont)

transa=trans()
x2=x*x
if(type(reg2).eq.REGR)fu=reg2()
/
reg2=regr(y,x,x2,trans->transa)
figyx=draw(func->transa(fu),x->x,xrange->,color->Orange,append->,continue->fc
Continue=1 !Errors
fi=draw(func->sin(x),x->x)
fi=draw(xrange->(1,100),func->Sin(x),x->x)
Continue=0
;if(wait);pause
```

```
;return
```

Note 34.11.2. if a line is not visible, this may be caused by the fact that the starting or ending point is outside the range specified by xrange-> or yrange->.

34.12 plot3d() 3d-figure.

Plot 3d-figure with indicater contours with colours.

Output	1	fi=plot3d() generates gnuplot file fi.jfig. No figure object is produced.
Args	1 MATRIX	The argument is a matrix having 3 columns for x,y and z.
sorted	N | 1	plot3d() uses the gnuplot function splot, which requires that the data is sorted withe respect to the x-variable. sorted-> indicates that the argument matrix is sorted either natrurally or with sort() function. If sort-> is not presented, plot3 sorts the data.

Example 34.6 (plot3dex). plot3d() example see p. 328 in Mehtatalo Lappi 2020

```
 mat=matrix(1000000,3)
mat2=matrix(1000000,3)
transa=trans()!second order response surface
x=0
x2=0
xy=0
irow=1
do(ix,1,1000)
y=0
y2=0
xy=0
do(iy,1,1000)
mat(irow,1)=x
mat(irow,2)=y
mat(irow,3)=12+8*x-7*x2+124*y+8*xy-13*y2
mat2(irow,1)=x
mat2(irow,2)=y
mat2(irow,3)=50+160*x-5*x2-40*y-20*xy+10*y2
irow=irow+1
y=y+0.01
y2=y*y
```

```
xy=x*y
enddo
x=x+0.01
x2=x*x
enddo
/
*** This example takes some time
call(transa)
fi=plot3d(mat,sorted->,continue->fcont)
** This is commneted because it takes some time
** fi=plot3d(mat2,sorted->,continue->fcont)
```

---

35 Splines, stem splines, and volume functions

There are several spline functions.

35.1 tautspline() Creates a more regular TAUTSPLINE

tautspline(x1,...,xn,y1,...,yn[,par->][,sort->][,print->])// Output:// An interpolating cubic spline, which is more robust than an ordinary cubic spline. To prevent oscillation (which can happen with splines) the function adds automatically additional knots where needed.// Arguments:// x1,...,xn the x values// d1,...,dn the y values.// There must be at least 3 knot point, i.e. 6 arguments.// Options:// par Parameter determining the smoothness of the curve. The default is zero, which produces ordinary cubic spline. A typical value may 2.5. Larger values mean that the spline is more closely linear between knot points.// sort the default is that the x's are increasing, if not then sort-> option must be given// print if print-> option is given, the knot points are printed (after possible sorting). The resulting spline can be utilized using value() function. The taut spline algorithm is published by de Boor (1978) on pages 310-314. The source code was loaded from Netlib.

35.2 stemspline() Creates STEMSPLINE

To be reported later, soon if needed.

35.3 stempolar() Puts a stem into polar coordinates

To be reported later

35.4 stemcurve() stem curve used with linear interpolation

stemcurve() defines a stem curve by giving points from the curve. Linear interpolation is then used to compute diameters at given heights, heights of given diameter, volumes in certain height section, cylinder and cylinders which get maximum volume. !

Output	1	Matrix			
	Output information	1	Matrix	A matrix containing stem curve points	
Args	4-999	REAL			
	Args	4-999	REAL	First half of arguments hive heights of points in decimeters, second half gives the diamters in centimeters.	

35.5 laasvol() Volume equations of Laasasenaho

To be reported later.

35.6 laaspoly() Polynomial stem curves of Laasasenaho

To be reported later.

36 Bit functions

bit functions help to store large amount of binary variables in small space. These functions are used in domain calculations

36.1 Bitmatrix

A BITMATRIX is an object which can store in small memory space large matrices used to indicate logical values. A BITMATRIX object is produced by bitmatrix() function or by closures() function from an existing bitmatrix. Bitmatrix values can be read from the input stream or file or set by setvalue() function. The values of bitmatrix elements can be accessed with value() function.

Note 36.1.1. Also ordinary real variable can be used to store bits. See bit functions.

36.2 setbits() Sets bits

To be reported alter

36.3 clearbits() Clears bits

To be reported later

36.4 getbit() : Gets bit

To be reported later, see old manual

36.5 bitmatrix() Creates BITMATRIX

To be reported later, see old manual

36.6 setvalue() Set value for a BITMATRIX

To be reported later, see old manual

36.7 closures() Convex closure

To be desrribed later, see old manual

37 Misc. functions

There are some functions which do not belong to previous classes.

37.1 properties() Properties of subjects

This function has been used to define properties of factories. It will replaced with othe means in later versions.

37.2 cpu() Cpu time

Example 37.1 (cpuex). Example of cpu-timing

```
 cpu0=cpu()
a=matrix(100000)
a=ran()!uniform
mean(a),sd(a),min(a),max(a);
cpu1=cpu()
elapsed=cpu1-cpu0;
```

37.3 secnds() Clock time

Example 37.2 (secondsex). Example of elapsed time

```
 cpu0=cpu()
sec0=secnds()
a=matrix(100000)
a=ran()!uniform
mean(a),sd(a),min(a),max(a);
cpu1=cpu()
sec1=secnds()
elapsed=cpu1-cpu0;
selapsed=sec1-sec0;
```

37.4 info() print some information of current parameters.

Note 37.4.1. Late addtiona information sis printed.

38 Error handling

There can happen basically three kinds of errors:

38.1 Error types

- The user makes an error an **Jlp22** detects the error, writes the proper error message and returns the control to the sit> prompt in the console, if variable Continue has value zero. If Continue has a nonzero value **Jlp22** continues one level above the console.

  The user makes an error, but **Jlp22** does not recognize this. As **Jlp22** should recognize all errors the user can make, this case is thus also a programming deficiency. In this case **Jlp22** behaves similarly as in the following case.

- The error is a programming error in **Jlp22**. Then two cases can happen:
  * **Jlp22** does not recognize that something is in error, and the system crashes. The debug version writes then more information than the release version. It is important that **Jlp22** is used in the commend prompt, so that it does not just disappear.
  * **Jlp22** recognizes in lower level subroutines that something is wrong, but the lower level subroutine does not know waht is the reason. Thwe **Jlp22** prints an error message starting with *j*. Even if the user recognizes that she was guilty, thes cases should be reported so that detection of user errors can be improved.

38.2 Handling of errors

There are two control layers, input programming and execution of one-line commands. Both can utilize the code obtained with parser, but input programming is utilizing the parser only a little. **Jlp22** function can use TRANS objects containing packed parsed code in many different ways. There can be nesting both in input programming and executing TRANS objects. The ;incl files can be nested up to 5 levels. Functions can be recursive in many different ways. A function can call itself through many routes.

In case of error should be able to tell for all nested ;incl files what line is processed, and errors within TRANS objects should be able to tell through the calling sequence which line is under execution. Let us test this in the following example.

Example 38.1 (errorex). `rex`

```
 round=5
write('inca.txt',$,'**we are in inca.txt')
write('inca.txt',$,'round;')
write('inca.txt',$,';incl(incb.txt)')
close('inca.txt')
print_f('inca.txt')
write('incb.txt',$,'**we are in incb.txt')
write('incb.txt',$,'round;')
write('incb.txt',$,'round=round-1;')
write('incb.txt',$,'call(transa)')
write('incb.txt',$,';incl(inca.txt)')
close('incb.txt')
print_f('incb.txt')
transa=trans()
c=4/round;
/
** let us see what happens
Continue=1
;incl(inca.txt)
** maximum ;incl nesting was obtained

round=2
** what happens now?
;incl(inca.txt)
Continue=0
delete_f('inca.txt','incb.txt')
```

---

39 Co-operation between Jlp22 and R

Is is possible to run R scripst from **Jlp22** and **Jlp22** scripts from R.

39.1 R() Executes an R-script

An R script can be executed with R(script) where script is CHAR object defining the script text file. The function is calling // call execute_command_line('Rscript.lnk '//j_filename(1:le), wait=.false.)// Thus a shortcut for the Rscript program needs to be available.

Example 39.1 (Rex). Example of Rscript

```
 rscript=text()
#A simple R-script that generates a small data to file mydat.txt
wd<-"C:/jlp22/jmanual"
x<-runif(10,0,10)
y<-cbind(1,x)%*%c(1,2)+rnorm(10)
mydat<-data.frame(y,x)
write.table(mydat,file=paste(wd,"/mydat.txt",sep=""))
//
write('miniscript.r',rscript)
close('miniscript.r')
R('miniscript.r')
print('mydat.txt')
delete_f('mydat.txt','miniscript.r')
```

39.2 Calling Jlp22-scripts from R

File JR_0.0.tar.gz in the folder J_R contains R tarball for taking **Jlp22** subroutines into R. With R command JR("testr.inc") the example jp problem can be solved from R. Later lauri Mehtatälo will develop this co-operation further so that R can directly access also matrices in the **Jlp22** memory. For further information contact lauri.mehtatalo@luke.fi THIS DOES NOT WORK NOW We will reconsider it soon with Lauri

40 Future development

The previous version contained possibilities to include factories into the optimization. Factory optimization is not available in the current version, because I'm now building a completely new version. I have already cleaned the data structures of linear programming and put the algorithm into reasonable subroutines. This makes it possible to follow and optimized the the flow of control. This made it already possible to put ordinary linear programming into a separate jlpz() function.

I think that the current version of **Jlp22** provides many possibilities for future developments. For instance:

current version does not have any special functions for making simulators. The new goto() commands, possibility to work with submatrices, and the new transdata()

function provide much more efficient ways to develop simulators. Examples will be provided shortly.Using the possibility to compute derivatives using the analytic derivates makes it quite straightforward to make it possible to have a nonlinear objective function and nonlinear constraints. These things are under design. It would be quite easy to include tools for piecewise linear constraints and objectives. It would be quite easy to develop **Jlp22** so that integer solution is produced with respect to the schedule weight. It would be interesting to see how **Jlp22** can put to work with Heureka. The possibility to run R scripts from **Jlp22** and **Jlp22** scripts from R provide new possibilities. **Jlp22** can now be used as an interface to gnuplot. Google search show how many possibilities gnuplot provides. It is quite straightforward to implement these graphs if it is not currently possible. The possibility to generate random numbers from any discrete or continuous distribution provide new possibilities to study the effects of random errors in the optimization. The new tools for analyzing grouped data are useful when studying the grouped data. it would be straightforward to implement mixed model methods based on expected means squares.

References

[1] Dantzig, G.B. and VanSlyke, R.M. (1967) *Generalized upper bounding techniques* **J** Compt Sys Sci 1(10),213-226

[2] Fletcher,R. 1996. Dense factors of Sparse matrices. Dundee Numerical Analysis Report NA/170.

[3] Hoganson, H.M. and Rose, D.W. (1984), *A simulation approach for optimal timber management scheduling* Forest Science, 30:220-238

[4] Hoganson, H.M. and Kapple, D.C. (1991), *DTRAN version 1.0. A multi-market timber supply model. Users' guide* Minneapolis: University of Minnesota Department of Forest Resources Staff Series Paper 82,

[5] Hyvönen, Pekka, Lempinen, Reetta, Lappi, Juha, Laitila, Juha and Packalen, Tuula (2019) *Joining up optimisation of wood supply chains with forest*, Forestry an international journal of forestry, 93(1):163–177, DOI = https://doi.org/10.1093/forestry/cpz058

[6] Lappi, Juha (1992) *JLP – a linear programming package for management planning* Finnish Forest Research Institute Research papers; 414, 134 p.

[7] Lappi, Juha and Lempinen, Reetta (2014) *A linear programming algorithm and software for forest-level planning problems including factories* Scandinavian Journal of Forest Research,29 Supplement 178–184", DOI = http://dx.doi.org/10.1080/02827581.2014.886714

[8] Thomas Williams, Thomas and Kelley, Colin (2022), *gnuplot 5.5*, url = http://gnuplot.info/docs_5.5/