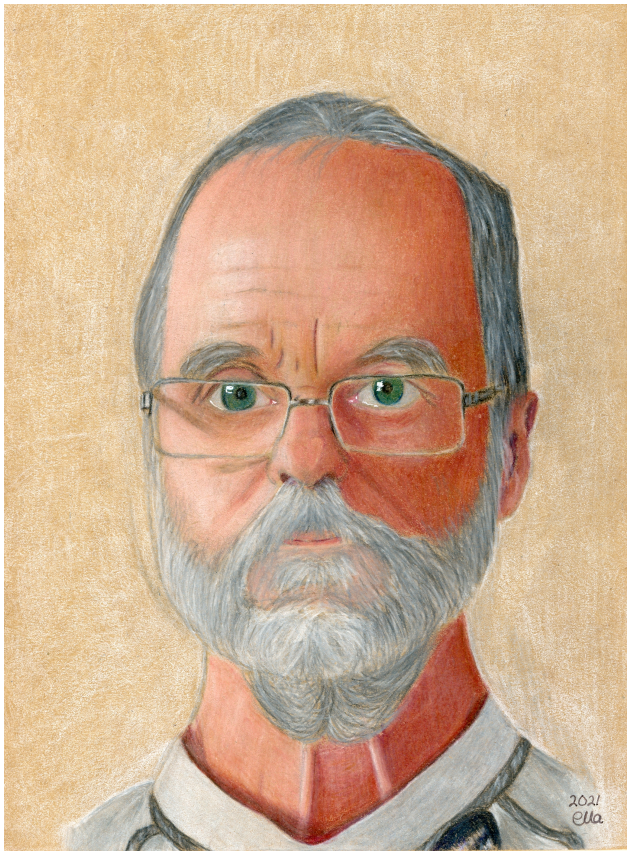
J
Users' Guide

Version 3.0 Friday 22nd April, 2022

Juha Lappi

juha.lappi.sjk@gmail.com

Reetta Lempinen



Contents

1 Preface

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

I will here describe the history of **J** in this preface in unusual length and detail. Readers interested ony in the use of the current software can perhasp look briefly the subsection Freedom. The reason why I'm going through in detail the difficulties with Luke is that I think that the policy insisted by the leaders of Luke that the researchers of Luke do not have any moral or legal right to publish the results of their research should be discussed. I think also that the leaders of Luke were willing to waste large amount of taxpayers' money when the prevented the development and timely publication of **J** software. I hope that this part of the preface gets some attention also outside the potential users of the software e.g. in the ministry. As an amateur actor, I also hope that some play writer would notice this and would make tragicomedy on the stage.

1.1 JLP and first version of J

I started to develop **J** software as a successor of the linear programming software **J**LP [**JLP**] around 2004. I had also done several programs for statistical computing (AKTA and Jakta) since late seventies when there was not easy to use software even for ordinary linear regression. Later I had to write own software for mixed linear models. AKTA, Jakta, and JLP contained means to use script files and make new variables described in arithmetic expressions. The starting idea in **J** was to put all my previous software developments into one program. Version 0.9.3 of **J** was published in August 2004.

In JLP the crucial property was the utilization of the generalized upper bound (GUB) technique which I invented and, which I could later find also in the literature ([**dant**]), as I expected. GUB is just the right technique which is needed in foreast management planning. JLP made it possible to solve significantly larger LP problems significantly faster than using standard commercial LP software. In JLP I programmed all the necessary matrix operations needed to change columns of the basis matrix in the revised simplex method. I updated the inverse of the basis matrix explicitly, instead of using factorization procedures as is the standard way in LP. I treated nonbinding constraints in a nonstandard way by reducing the dimension of the basis matrix instead of using residual basic variables, as is normally done.

When I started to develop J, I thought that it might be good idea to use

matrix routines based on factorizations and developed by a professional in this field. I found the LP software bqbd by Fletcher based on [**flet**]. The bqbd software was meant to use 'as is' software for linear and quadratic programming. When I told Fletcher that I plan to separate the matrix routines needed in the GUB algorithm, he told he is certain that I will not succeed, as his Fortran code did not have any comments. He wrote that he will not have any time to help me. However, it was not that difficult becasue his code was divided to proper subroutines. His code did computations in single precision, and it was necessary to change computations into double precision. Because I do not have a proper training in the rounding error business, I had hoped his advice in changing the several tolerance parameters to correspont double precision computations. In the development of the original JLP algorith, I spent two-three times more time to fight rounding errors than to get the code mathematically correct. Now I understood the rounding errors better, but anyhow they are the permanent nuisance. The fundamental question with respect to rounding errors is: is a small-looking number zero or non-zero.

When I later told him how large problems and how fast my algorithm could solve, his attitude towards my efforts clearly changed.

Even Bergseng from NMBU started to use **J** in forest management problems in a quite premature stage, and we had quite tough time to keep his project in time. It took some time to clear a bug in Fletcher code which every now and then took the optimization into a wrong path. I had a quite slow computer, and the data transwer between Norway and Finland was slow. The bug was such that it had never caused problems in standard applications of bqbd software. In forest planning problems it can happen that after many pivot operations only residual variables corresponding nonbinding constrains are in the basis. And when **J** refactorized of the basis after fixed number of pivot operations, as is the common procedure, the basis became slightly corrupted. In such a case, the refactorization is not needed. NMBU started to use **J** together with their GAYA simulator, and called the whole system GAYA-J, as the previous system was called GAYA-JLP. I appreciate that they have acknowledged our co-operation this way. **J** version 0.9.3 was published in August 2004.

J contained also a simulator language by which one describe a simulator which can be used to geneate treatment schedules which are later optimized with the LP funtions of J. When Simosol started to develop sofware which would compete with the Mela, which had had a monopoly status in planning software in Finland, and had all the problems which are inevitable in all monopoly projects. Simosol made first simulator developments using J, but moved to other solutions before first published version of Simo soft-

ware. Simosol however used the linear programming routines of J. Jussi Rasinmaki presented the software of Simosol in the seminar 21.9.2018 celebrating the work of Pekka Kilkki. When he did not mention J, which continues the work of Pekka Kilkki, I asked whether they have stopped to use J. RasinmÃ¤ki told that they are still using J. I do not know waht is the situation today.

1.2 J with factories: J2.0

For me the most disturbing feature of **J** and JLP was that they were not able to treat factories. I thought that the clever heuristic DTRAN algorithm of Howard Hoganson ([**dtran**] and [**howard**] showed that it should be possible to extend basic ideas of the **J**LP algorithm to treat with factory problems. I was able to derive the needed equations (see [**lappilem**]). Reetta Lempinen started to work with me to implement the formulas. Reetta has professional training in programming, while I'm a self-learning amateur. The necessary data structures were quite complicated. I had made some wrong decisions which made the data structes even more complicated than was necessary. Reetta had a clear head to keep data structures in order. It was a nice new experience to work together through video connection on the code which both could see simultaneously. I just gave commands, but it was not necessary to give detailed instructions because Reetta understood from a half word what to do. An important contribution of her was that when there were several dead ends, and I started to doubt whether we can really do what we were aiming at, she told, 'listen Juuso, we can solve this problem as we have solved all previous problems'. Interestingly, the problems were usually not in any programming errors, but the path could be found by raising to a little upper level and going to the main ideas. Reetta's role was essential in the factory optimization. Also later she has been helpful in minor technical matters, in writing the manual, testing the new developments etc. Recently she adviced me how to use Git and Github. Reetta has also been solidary during the painful Luke years, when she was not allowed to work with **J** as she had liked to do. I appreciate her contibution very much. She is the second author of this manual and the paper [**lappilem**] for good reasons. However, I use word 'I' for such situtions where Reetta has not been involved in the decisions how to proceed. The factory optimization was implemeted in **J** version 2.0. published in 2013.

1.3 Battle over open-source J3.0 starts in Luke

Initially I made the software so that only I could understand the structure and the code. When Reetta started to work with me in factory problems, I learned what are the weakest points in the software if it should be maintained by a group of people. But as I was always able to tell Reetta how the software worked, only very slight improvements in the documentation and comments of the code etc. were made. When I realized that my retirement age is approaching, I started to worry how the software could be maintained and developed after my retirement.

I had worked for Finnish Forest Research Institute (Metla) with small Academy and University of Joensuu brakes since 1978. In 2015 Metla and two other institutes were merged into Institute of Natural Resources Finland (Luke). Based on my long service in a government organization, I expected that the new organization would mean new leaders, new lawyers, new bureaucrats, and much more bureaucracy, and less time for researchers to do research, as always happens when government organizations simplify their administration and make it more efficient. My expectations were in the right direction, but even in my worst nightmares I could not have imagined what Luke would mean to me personally.

Lauri MehtÃ¤talo and I had agreed with an international publisher to make a book about forest biometry. In Metla this kind of agreement had been considered as a merit to Metla. But Luke forbad me to put any working time to this book project. After retirement I was free to prepare this book with Lauri. On the back-cover of the book I presented myself as a researcher of Metla in order to give the affilation credit to Metla. The cooperation among statisticians of Luke, led by Juha Heikkinen, the continued co-operation with Jaana Luoranen, and the support of the whole personell of the Research Station of Suonenjoki when the leaders of Luke started to humillate me were my only positive experiences at Luke.

I started to clean and reorganize the **J** code so that it would be easier to maintain it. My superior Olli Salminen forbad me to continue that effort because he was certain that I would never get that work ready. I didn't obey, of course not. The eternity of this first cleaning round lasted two and half weeks. Reorganization of files took 2-3 days, but programming of a precompiler took two weeks. In JLP I already had learned that a precompiler could help to manage data areas of JLP. In J, the precompiler has greatly helped to manage global variables. The precompiler allows the users access global module variables without knowing where they are. To separate the global variables from local variables, the global variables have j_ or jlp_ prefix. By changing the data structures, **J** also became significantly faster (main reason was that allocatable pointers were changed into allocatable

arrays).

In 2014 when we were looking at how **J** could be utilized in Mela software, I encountered old JLP concept âown functionâ, which was the method for allowing users of JLP to add own (arithmetic) functions if they had access to the source code. I thought that this idea would be useful also with respect to J. So, I organized **J** so that different users could add their own **J** functions, object types and options.

I was then convinced that the best way to utilize these improvements would be to make the soft-ware open source. This would also allow me to continue my work freely with **J** after my retirement.

The leaders of Luke agreed with me that the open-source **J** is a good idea. Unfortunately they also thought that open **J** is so important for Luke that the leaders could not allow me to take care of the publication and prepare and present the necessary decisions for their approval, or even participate as member in the group which started to hustle with the publication. The leaders wanted themselves take care of the publication. Unfortunately, none of the leaders involved considered that the publication of **J** would be even so important that he/she had taken care that **J** would be eventually published. They just wanted keep important in the process. This absurd deadlock lasted almost seven years.

The previous version utilized some subroutines of IMSL package and from the âNumerical Recipes in Fortranâ book. I replaced these subroutines mainly by open access routines from Lapack and Ranlib packages from the Netlib library. Roger Fletcher allowed to distribute his linear programming sub-routines in the open **J** package. Fletcher gave a very liberal permission to use his subroutines in open source distribution (the price of his bqbd sofware was perhaps 100-200 Euros). But the lawyers of Luke studied his permission so long that he died before giving permission with better formulation. A permission with better formualations was obtained from his department in Dundee University. But the leading laywer of Luke, Emilia Katajajuuri was not satisfied, she wanted make sure that the legal position of the person who gave the new permission was such that he could give the permission.

The following persons were involved in process: PÃ¤ivi Eskelinen, Olli Salminen, Kari T. Korhonen, Kimmo Kukkavuori, Tuula Packalen, Sirpa Thessler, Sari Forsman-Hugg, Emilia Katajajuuri and Johanna Buchert. Iikka Sainio was an outside legal consult.

The leading lawyer of Emilia Katajuuri made my position clear in a video meeting to which I also was accidentally invited to listen their new instructions. When I accidentally said 'could we decide that..', she interrupted me and said: 'Listen Juha, you do not decide anything in this matter'. Their

message was that I have done my part, and now I should keep out and not disturb their play with their new conquest.

When it was evident that the leaders of Luke wanted to get rid of me, I threatened that I will not retire before the opening decision is done. Sari Forsman-Hugg told she will make the decision. But as my threat seemed to make no influence, I retried at the beginning of 2017. When I despite of all the humiliation and frustration still hoped that some kind a co-operation with Luke would be possible, I applied the position outside researcher, which had provided me access to Luke's e-mail and library services. Sari Forsman-Hugg was not able to make a decision also with recpect to my application. I did withdraw my application when my e-mail connections were interrupted two times.

Later I heard from Reetta that Luke had made the opening decision but she did not know what kind decsion and I was not informed of the decision either, in disargeeent with the distribution list of the decision, as I noticed in when I finally could see the decision after three years.

1.4 Retired, frozen conflict

After my retirement I was still hoping that some kind of co-operation without outside researcher status with Luke would be possible, and I participated in the application of factory optimization which was published in **??**. Prof. Packalen told late 2017 that the publication of **J** is approaching. When I asked about the opening decision, she 'did not remember' what it contained but promised to check it out, which she did forget to do or atl least forget to tell me what the decision contained. Prof. Packalen was planning a very high level launch of **J** with prominent international guests. I wrote that such seminar would not probably be worth of for trans-Atlantic flights. I was so stupid that I believed her promise that the publication of **J** was approaching and I submitted a correction to a bug which produced some amount of negative timber quantities.

During 2018-2021 I made some small parameter adjustments for Simosol and Norway and made some additions for my own research. Prof. Packalen had lost all interest in the publication, and she did not allow Reetta to develop the publication documents. When I heard 2020 that prof Packalen had left Luke and could not prevent publication any more, I asked Reetta to ask her superiors whether she could continue the preparation of manuals. She got the permission to use some time for that. This work with all style details was not very nice for her as she had already done big part of the work for the previous format of Lukes's report series. The report series had changed the formats, and Reetta had to do such tedious work she al-

ready had done once before. Reetta's superiors did never discuss with me anything related to the manuals or to the code. This way they had been free to pretent that I have not contributed to the work if **J** had been some time published.

In spring 2021 Hannu Salminen from Luke contacted me. He had memory problems with a very large data set. I made a rapid fix for his acute problem and did develop more permanent solutions for large data sets using direct access files. Actually that work was unncessary as Viktor Strimbu from NMBU adviced me later how to make **J** a 64-bit application using MSYS2 environment. The Gfortran environment I had used before did not support 64-bit applications. The 64-bit program allows to put the whole country into one run aslo in factory optimization. Hannu organized a video meeting where we discussed the development and utilization of the **J** software. He was the first person in Lukes six years history who was willing to discuss with me such things.

I started to work with **J** again. Reetta had told that **J** had could collapse after some hours of computations in large data sets in factory optimization. I asked Luke to get the problematic data sets so that I could try to fix the problem. I got the answer that Luke is not allowed to send the data sets, and the answer did not indicate that Luke would do anything to get such permission. I was quite upset from such answer, and I started to think how to open the deadlock situation. Later I got the data, and I was able to fix the problem, which was not exactly a clear bug but rather a thin-ice place in the algorithm sot that **J** did drop throug the ice when going over the place repeatedly.

1.5 Operation Charles XII

When I strarted to think how I could find a way out from the deadlock, I remembered Charles XII, the king of Sweden (wich included also Finland). He was in war with Denmark and he decided to fight Denmark by attacking Norway, which belonged to Denmark. He failed as almost his solders died in snowstorms in Northern Norway and he was shot by his own soldier. I was in conflict with Luke, even if Luke did acknowledge that such confilict existed because Luke insisted that in a legal sense, I did not even exist in this **J** affair. I thought to conquer Norway with nice **J** codes. Renovation of my co-operation with GAYA group suited their plans as they were rebuilding their simulator.

I wrote to Johanna Buchert, director general of Luke (who was already in the group of Luke leaders hustling in the publication of **J** when I was in Luke) Antti Asikainen, research director of Luke, and Kari T. Korhonen,

who has been involved in this **J** affair and was the superior of Reetta, and told that I will not send to Luke new **J** codes including working factory optimization codes without a written co-operation agreement. I wrote also to registry-office and asked for the opening decision, and to a lawyer of Luke and asked what kind of legal actions Luke will take agaisnt me if I started to publish factory optimization things with GAYA group using codes which I would not give to Luke. The leaders did not comment in any way my co-operation proposal. The research director just wrote that all codes done in METLA are property of Luke. Buchert did not respond, but the the Senior Vice President Ilkka P. Laurila answered that Luke regrets for the delays in the publication of **J** caused by personell changes (a funny expression for the fact that the publication process could just start in 2020 when prof Packalen left Luke) and that Luke has started to prepare the publication and will publish the **J** on its web pages. Kari T Korhonen did not respond.

Thus the leaders of Luke insisted that Luke will publish tens of thousands code and manual lines I had written, and insisted that it has no reason or obligation to discuss with me what and in what form it will these lines publish. It is evident that Luke had not been able to publish **J** in my life time (I'm over 70 yrs and I have a heart disease). But if I had been alive when Luke had published **J** under my name, I had made a lawsuit againt Luke for shaming my researcher honor by publishing something I had not accepted for publication. Or perhaps Luke had followed the previous custom of Romania where all chemistry research was published under the the name of Elena CeauÈescu, and had published the the software and manuals under the name of some equivalent to Elena. Also in that case I had made a lawsuit against Luke (if I had been alive). When the director general of Luke, the research director of Luke, the Senior Vice President of Luke and the lawyers of Luke decided that Luke has absolutely no interest in working factory optimization **J** codes or other up to date **J** codes, did they discuss with any researchers?

After getting such a reponse to my co-opertation suggestion, it was evident that I would not give Luke any new codes, and would require Luke to remove all codes I had submitted after my retirement as Luke would not akckowledge that it had got any codes from me after my retirement, and it would publish my codes as if they were developed in Luke. So I would continue my Charles XII operation. Unfortunately NMBU did not have factory and transportation cost data ready. In Norway the transportation costs can not be estimated as easily from coordinates as in Finland.

Laurila did send the decision to publish **J** as a open source software after the lawyers of Luke had considered two weeks whether Luke should do that. The decision was signed by the research director Johanna Buchert

30.8. 2017. The process of trying to get permission to use Fletchers codes after his death was not described properly. It was written that after Fletchers death the university of Dundee did not respond to queries of Luke. As decribed above, the things did not go that way. When Luke was not satisfied with the corrected permission sent by the university, the university had probably decided that it is vain to continue correspondece with Luke. However, the university wanted to communicate with me, when they asked my statement about the importance of Flethers code in planning of natural resources. The government wanted to figure out whether the university is doing something useful. I think that my statement was more valuable for the university than the euros the university would loose when not getting payments for the bqbd code.

The decision told that **J** is published under licence AGPLv3. The licence is specially planned for programs which provide computing services through the net. Luke has not done anything to develop such net work services, as it has not developed **J** in any way after my retirement in 2017. The leaders of Luke did not discuss with me the license matters, of cource not.

The decision nominated prof Packalen to take care of the launch of the software. Unfortunately she could not organize the lauch, as **J** was not ready to publication because prof Packalen did not allow Reetta to do the necessary preparations with respect to the code and manuals.

The lawyers of Luke did not tell anything of the plans of Luke to make lawsuits against me if I would co-operate with Norway and would not send new codes to Luke.

Lauri MehtÃ¤talo became Professor in Mathematical Modeling for Forest Planning in Luke in Mai 2021. Coming from an university environment, he is interested in the development of science and research. He wanted to get **J** finally published after 6 year deadlock. Lauri did have anything againts that I would take care of the publication. Lauri even thought that it would be good for science and research that I would work without payment. It was a refreshing new attitude in Luke to put science into priority. Lauri told that he would try to convince the leaders of Luke that it would not harm Luke's interests that I would take care of the publication of the software I had made (for most part). Lauri told initially that he wants to persuade me to give all new codes to Luke. When I protested his choice of words, he apologized, and used thereafter the word 'negotiate'. When Lauri started to 'negotiate' with me, I consider that Luke first time in its history acknowledged that I do exist as an legal entity is this **J** affair.

Lauri was able to rotate the head of the research director 180 degrees. The research director even told that he would be interested to become a coauthor in the publication dealing the factory optimization. It was, how-

ever, not mentally feasible for me, as he did not even answer my previous co-operation suggestion, but just told that Luke has everything it needs. Being a researcher, Lauri understood what were my aims and was able to get approval of the research director. So it seemed that the agreement was about ready. But then the agreement was stuck on the table of the research director.

1.6 Last battle, victory and peace in Olavinlinna

I was invited to the 100 year anniversary celebration of the Finnish National Forest Inventory as I had worked also in the NFI group. I was interested to see my former colleagues, but it started to disturb me that I would not know whether I would be a guest of my enemy or my co-operations partner. I had a very exciting video meeting with the GAYA people. They promised me access to the Norwegian supercomputers needed in large factory optimizations, funding for traveling and for any consulting work. I started to think that have been stupid when I spent years of my short residual life waiting for such co-operation with Luke. As Lauri had tried to establish such co-operation between me and Luke, I decided to give Luke a last chance. I made an ultimatum that if we do not get a agreement with Luke within a week, I would break all negotiations with Luke and would continue only with Norwegian people, even if Luke might try to make a lawsuit against me. Twenty-four hours before the time limit Lauri called that Luke is willing to make treaty with me. The lawyers of Luke did finalize the treaty two hours before the NFI celebration.

Antti Asikainen, the research director of Luke and I signed the co-opertation treaty in the king's hall in Olavinlinna (St. Olaf's Castle) in Savonlinna 7.10. 2021. In the treaty I got almost everything I had fighted for: The **J** software will be published, and I have full responsibility of the code and manuals. Luke will provide me data of factory locations, simulated treatment schedules for all inventory plots for the whole country, and the matrix of distances between the factories and inventory plots. Luke will also provide me access to CSC supercomputers which are necessary for the whole country optimizations. CSC does not allow independent retirees to use its computers. I'm allowed to make a methodological paper from developments, whereafter I will provide Luke all the new and improved **J** functions and the developed **J** scripts. I had hoped that Luke had promised me access to Mela so I could study how Mela is doing interest rate computations, as I have suspects that these computations are not in order. I had asked Olli Salminen some Mela results to study my suspects, but he told, after some day considerations, that Luke is not willing to make such computations. I

suspect that the reason is Luke wants to keep all possible Mela problems secret. I did not insist to get such permissions as Lauri thought that this would delay and complicat the **J** process. In the treaty it is said that the code files state that Juha Lappi and Natural Resources Institute Finland have the copyright for the code. In the last minutes before the signatures the lawyers tried to reverse the order of copyright owners. I did not, of course, accept the reveresed order, bacause I have made the **J** really operable after my retirement, Now, the recent developments have made Lukes or Metlas contribution even smaller, and the overall influence of Luke has been negative. It must, however to be taken into account that Pekka Mäkinen has done valuable work in collecting data of factories and transportation costs, and this data greatly helps my work when I'm now allowed to start to develop factory optimization for the whole Finland.

1.7 Total balance with Luke

At the signing of the peace treaty, Johanna Buchert told that there has not been any disagreement in this **J** busisness, and let bygones be bygones. From my point of view, I had six and half years fight to get permission to publish the results of my research. For Luke, there was no disagreement because Luke did not acknowledge that I even existed as a legal entity. How could Luke have disagreement with something which does not even exist? For me, the behavior of the leaders of Luke was a 'me too' experience. In the heart of my researcher identity is the conviction that every researcher, at least in a government research institute, has the right and even the obligation to publish the results of his research let it be a stastistical method or computer code or ordinary research publication. The leaders of Luke denied me the right to publish the results of my statistical work in our biometry book and the results of my software development. In this **J** case, the reason was that they wanted to publish my results themselves. This has been a very traumatic harrasment experience. All the victims of sexual harassment are also told that 'let bygones be bygones'. As in the peace treaty I was given all responsibility of manuals, I'm now utilizing this responsibility in this preface hoping the leaders of Luke and perhaps also others would be more careful when making decisions which may not tolerate daylight, similarly as bosses are slowly learning to keep better control of their hands which would like to grope the bottoms of their subordinates.

From the legal perspective, the leaders of Luke have insisted that Luke has full copyright to all work I've done in Mela or Luke. So they can freely follow their whims and waste hundreds of thousands of euros taxpayers money used in the **J** project. With Lukes overhead multipliers we may

speak about millions. Perhaps they should also pay attention to the by-law telling what is the duty of Luke. How has Luke promoted forest research and its utilization by prevented the development and utilization of **J** software for over six years? Perhaps the ministry should give the leaders of Luke training about the duties of Luke. The leaders of Luke never gave any rational justification why Luke should prevent me to take care of the publication.

If I had been allowed to publish and develop **J** as I wanted, Luke and forest industry would have now available methods to analyze e.g. what can be done when timber import from Russia suddenly stops. Luke is also still doing static regional sustainability analyzes, which are not reasonable as they do not take factories into account. Mentally Luke is stuck to the German foresters from 1800's who rotated a fixed cutting area for each year. If Luke had let me publish the open-source software in 2015 Luke had been a forerunner in the open source publication, and I had praised it for its progressive attitude. Now Luke has has establised its status in the history of open-source as an institute which decided to open a software but keeps the decision and the code locked for several years because it did not allow the principal author to take care of or even participate in the publication.

1.8 Freedom

After the Olavinlinna peace treaty, I thought initially that I would like to publish the software before Christmas 2021. But when the iron cage around me disappeared, the pressure which had increased in my mind for seven years did burst in intensive code development. This development had already started when negotiations with Lauri started to look promising, and I developed the program to make Latex files for manuals and script files for examples automatically from comments embedded within the code or from a separate text files.

After the peace treaty I have rewritten all the central parts of the software using simpler and more easily maintained solutions, and added many new properties. The most important was the rewriting of the interpreter which interprets the **J** code generated using the input programming capabilities. I had been shamed if the previous interpreter had be offered to the eyes of professional programmers. I did rewrite the precompiler so it takes only some seconds to do the precompilation, and if the precompiled file does not change the new precompiled file is deleted so that the compiler does not do any more unnecessary compilation. I did put the matrix computations into a new level implementing all properties I had found useful in Matlab when providing a method for a client using Matlab codes. I

added new tools for debugging and code development. Now **J** can treat nicely submatrices and do all arithmetic operations and logical operations on matrices in addition to scalars. The previous version of **J** could make rough figures using a separate program written in Intel fortran, which I do no more have available. Publication level figures could be done using R sripts generated by J. Now **J** is producing figures using Gnuplot which is automatically run from within J. Currently only basic 2D and 3D figures can be done, but this Gnuplot connection makes it possible to develop **J** into full scale interface to all Gnuplot properties. **J** can now generate random numbers from any continuous or discrete distibution providing new possiblities to study the effects of random errors in the simulation of treatment schedules or in optimization. It is now possible to run R scripts from within J. Now **J** can be used easily also as a subroutine. Lauri has utilized this property by calling **J** from R. This offers many new possiblities, which we will utilize when continuing the co-operation with the GAYA-group.

I made first cleaning of the important jlp() function in the spring of 2021 when correcting the fragility which prevented the solution of large problems which take several hours in a ordinary PC. There are many things to be done in jlp() function, but I start to work with them when developing factory optimization in CSC supercomputers. I know already how parallel computations can be utilized in jlp()-function. Previous versions of **J** had special functions for developing forest simulators and computing the simulated schedules. Now all simulators can be developed and utilized using standard properties of **J** transformations either for using stand level or tree level variables I have not yet made good examples to demonstrate these properties. Because the new **J** provides a platform for making all kind of new devlopments, including integer optimization (i.e preventing treatment unit to be divided among several schedules), nonlinear objective function (which would utlize the capability of **J** to compute derivatives according to analytic derivation rules), it was difficult to breathe a little and make the new codes available for users. The codes now published contain many errors. I invite the users to help me to correct them. First users should have some patience with respect to errors. I claime that the bug percentage is smaller than estimated 23-27% bug percentage in the first version of Windows 1, which accintally happened be the same as the bug percentage Soviet masons used in the mortar they used when masoning the US embassy in Moscow. Currently I can correct most bugs causing system crash in 5-10 minutes. It should be emphasized that Luke does not have any kind of responsibility of the errors. I take the full moral responsibilty of the errors. Similarly, it should be emphasized that Luke has no responsibility of the code which works correctly. I take the main responsibility of the

correctly working code, giving Reetta her fair share. It is easier for me to correct the errors while alive. This fundamental fact of life was ignored by Luke when it prevented the publication for seven years.

During the intensive work period after the Olavinlinna peace treaty, my deep 'me too' traumas have started to heal slowly. Probably this healing process takes a long time, and I cannot very soon say let bygones be bygones.

Juha Lappi, Suonenjoki 22.4.2022

1.9 Changes from previous versions

The software is almost completely rewritten. There are many new functions and properties. All the previous users should look at the new properties. If an old user has difficulties to run their old scripts they should contact J.L. Note that graphics now assumes that Gnuplot is installed.

2 Introduction

The **J** software can be used as is, i.e. using exe files. The User guide concentrates on using binary files, but some reference is aslo made to additional possiblities offered by the open **J** code which Luke finally after 7 years deadlock allowed to publish.

2.1 Using exe files

J is a general program for many different tasks. In one end, **J** is a programming language which can be used to program several kind of applicatios 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 **J** functions and **J** trasformations can be combined in many ways. ! The development of new **J** applications is made easier with special input programming. Input programming can generate efficiently several **J** code lines with few lines so that any part of object names can be have indices. **J** scripts can be read from include files which can contain sections, so that all code of a project can be stored in one file. Different sections can be executed by just writing the name of the section at sit> prompt. ! The code development is easy using efficient debugging tools. The output of a **J** function is printed if the

line ends with ';' or ';;'. After the code development, the printing can be put off without changing the code by telling with Printresult variable whether ';' or ';;' or - results are printed. During pause() the user can do any computations and print objects and then either continue or interrupt the process. With Debugconsole variable the user can tell whether pause() is generated after each command line read from an include file. With Debugcode the user can tell wther pause() is generated after each line when **J** excecutes code made by the interpreter and which is packed into a transformation object.

J is now efficient program to do many kind matrix computations. After retirement I sold a client an algorithm which I dad developed using J. I submitted the algorith using Matlab code, and thus learned Matlab. Then I implemented all Matlab properties I found useful. all arithemetical, trigonometric and logical funtions cand operate both on scalars and matrices. Some useful extensios to normal computation rules were addewd. Matrices can be made from submatrices, submatrices can be extracted from matrices and replaced by other matrices. ! The list object is used to transimit and acces object lists. If the list elements are objects called REAL variables, the the list can be used in matrix computaions in the same way as column vectors. ! The input programming has the following functions and capabilities.

- ;incl(file,form->) include code from file starting from a label, e.g. as ;ad1;
- ;do(i,ia1,ia2,step) generates input records in a loop
- ;if();then
  !includes conditionally code
- objstart"i"end ! replaces i in the text with the integer value of i. Combined with ;do() this provides possiblitesd to generate large numer of lines using only some linse ;goto() ! code is obtained from a different place of an include file.
  ;elseif();then
  ;else
  ;endif

J functions can have arguments and options. Options are used to trasnmit optional arguments to a function. Options are expressed as option->. If the option does not refer to a single object then the value of the option must be put into parentrhesi.In some cases the prence or absence of an argument passes an logical argument into a function. There tow groups of options, ordinaryy options and code options. Ordinalry options treansmit

either object names or numeric values into a function. code options contain a short peice of code which is computed several times form within the function. E.g. func->sin(x) can tells the draw() function waht is the function to be drawn. If computation of the codeoption rquires several lines, then thes lines can be then the func->tr(res) is used to tell how the function value is computed.

2.2 Github

The package at github.com/juhalappi/j contains the following components

- **J** Data: folder for data files
  - **J** test.cda example unit data file for small jlp() example
  - **J** test.xda example schedule file for small jlp() example
  - **J** test.inc include file for the jlp() test
- **J** J_R: using **J** from R, courtesy of Lauri Mehtätalo. !
  - **J** j.par decaul include file for starting **J**
  - **J** JR_0.0.tar.gz File neede to us fortran subroutines in R
  - **J** testr.cda
  - **J** test.xda
  - **J** testr.inc include file an jlp-example
  - **J** testr.inc
- **J** Jbin: binary .exe files and dll files
  - **J** j3d.exe Debug version of J, release version is provided after some testing period
  - **J** jindent.exe indentation
  - **J** jmanual.exe makes the latex code file jmanual.tex
  - **J** jpre.exe the precompiler which generates the code for accessing variables in module
  - **J** 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
- **J** Jdoc_demo documents and include file for running examples fro User's guide

- **J** hyvonenetal2019.pdf facotory optimiazation paper
- **J** J3.0_userguide_2021.pdf old version of the manual, explains better some functions thant j3.pdf
- **J** J3.pdf Users guide made with Latex
- **J** J3_setup_development.docx not up-to-date manual for developers
- **J** jexamples.inc include file which can be used to run all examples in the manual and which is generated with jmanual.exe
- **J** jlp92.pdf Manual of old JLP which explains the theory behind the jlp algorithm
- **J** lappilempinen.pdf

J Jmanual: Source files for making Latex code for the manual and the include file for running examples

- **J** jmanual.f90 source for making the Latex code and jexamples.inc
- **J** jmanual.tex Latex code genrated with jmanual.exe
- **J** jsections.txt describes such manual sections not in source files
- **J** jsections2.txt tells in what order sections foun in jsections.txt and source files are put into the manual and what is the level of the sections
- **J** main.tex main tex code containd Latex definitions
- **J** Makefile_debug Makefile for making jmanual.exe

!

J Source : source code before precompilation

- **J** fletcherd.for Fletchers subroutines turned into double precision
- **J** j.f90 code for **J** functions
- **J** jlp.f90 code for linear programming
- **J** j.main main prorgam for calling **J** when used as is, if **J** is used as a subroutine then this must be made a subroutine
- **J** jmodules.f90 data structure definitions
- **J** jutilities.f90 subroutines for handling objects etc.

- **J** jsysdep_gfortran.f90 system dependent routines
- **J** matsub.f subroutines obtainde from other sources, e.g. from Netlib
- **J** other subroutines for setting up users own function

J Source2 : source code files after precompilation in addition to files in Source (such files which are not precompiled are put in both folders)

- **J** makefile_debug makefile for making debug- version !
- **J** makefile_release makefile for making release- version

J LICENSE:the license file

J README.m :readme file

!

2.3 J functions

The general (non arithmetic) **J** functions are used either in statements
func(arg1,â¦,argn,opt1->value1,â¦.,optm->valuem)
or
output=func(arg1,â¦,argn,opt1->value1,â¦.,optm->valuem)
If there is no output for a function in a statement, then there can be three different cases: i) The function does not produce any output (if an output would be given, then **J** would just ignore it
ii) The function is producing output, and a default name is used for the output (e.g. Result for arithmetic and matrix operations, Figure in graphic functions).
iii) The function is a sub expression within a transformation consisting of several parts including other function or arithmetic operations. Then the output is put into a temporary unnamed object which is used by upper level functions as an argument (e.g. a=inverse(b)*t(c)) If the value of an option is not a single object or numeric constant, then it must be enclosed in parenthesis.
It is useful to think that options define additional argument sets for a function. Actually an alternative for options would be to have long argument lists where the position of an argument determines its interpretation. Hereafter generic term 'argument' may refer also to the value of an option.

When **J** is interpreting a function, it is checking that the option names and the syntax are valid, but it is not checking if an option is used by the

function. Also when executing the function, the function is reacting to all options it recognizes but it does not notice if there are extra options, and these are thus just ignored.

An argument for a **J** function can be either functional statements producing a **J** object as its value, or a name of **J** object. Some options can be without any argument (indicating that the option is on).

An essential feature in **J** functions is that the driver subroutine which is computing the functions is recursive. This recursion is used when code options SEE? are used to compute the the output of an code option or when transformation objects are explcitly called within an transformation object or a function dealing with data is computing trans-> transforamtion for each observation.

```
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,â¦valn, and objects by obj1,â¦,objn, or the names of real variables are indicated by var1,â¦,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.

```
stat(D,H,filter->(sin(D).gt.cos(H+1)) !
only those observations are
accepted which pass the filter
```

```
draw(func->(sin($x)+1),x->$x,xrange->(0,10,1)) ! the func-> option
```
transmits the function to be drawn not a single value.

2.4 Structure of general J functions

The general (non arithmetic) **J** functions are used either in statements

func(arg1,â¦,argn,opt1->value1,â¦.,optm->valuem)

or

output=func(arg1,â¦,argn,opt1->value1,â¦.,optm->valuem)

If there is no output for a function in a statement, then there can be three different cases:

- **J** The function does not produce any output (if an output would be given, then **J** would just ignore it
- **J** The function is producing output, and the default name Result is used for the output for arithmetic and matrix operations.
- **J** The function is a sub expression within a transformation consisting of several parts including other function or arithmetic operations. Then the output is put into a temporary unnamed object which is used by upper level functions as an argument (e.g. a=inverse(b)*t(c))

If the value of an option is not a single object or numeric constant, then it must be enclosed in parenthesis.

It is useful to think that options define additional argument sets for a function. Actually an alternative for options would be to have long argument lists where the position of an argument determines its interpretation. Hereafter generic term 'argument' may refer also to the arguments of an option. When **J** is interpreting a function, it is checking that the option names and the syntax are valid, but it is not checking if an option is used by the function. Also when executing the function, the function is reacting to all options it recognizes but it does not notice if there are extra options, and these are thus just ignored. An argument for a **J** function can be either functional statements producing a **J** object as its value, or a name of **J** object. Some options can be without any argument (indicating that the option is on). [jfuncex]Examples of J-functions

```
 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.
Dm=matrix(do->(0.1,40))
nob=nrows(Dm)
```

```
e=matrix(nob)
e=rann()
Hm=0.5+Dm**0.7+e
dat=newdata(Dm,Hm,read->(D,H))
stat(D,H,min->,max->) ! Here arguments must be variable names
plotyx(H,D) !arguments of the function are variables
h=matrix(5,5);
h=rann();
g=matrix(5,do->5);
c = inverse(h+t(g)); ! The argument can be intermediate result from matrix computations.
```

2.5 J objects

J objects have a simple yet efficient structure. Each object is associate with a double precision variable, 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 even if some predefined objects are locked so that users cannot change them.

2.6 System requirement

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

See J3.0 Development Guide to start develop the software or to add own functions. The development package contains, in addition to source code for the standard **J** software, program Jmanual which can be used to generate Latex code for the manual, program Jindent to make indentations for Fortran source files and a precompiler Jpre which writes necessary Fortran statements to access all globa **J** data structures.

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:Files˙J will strat Gnuplot automatically when plotting figures.

2.7 Set-up of J

The maximum number of available objects cannot be changed during a **J** session. It is determined during the initialization. When **J** is started it tries to read first file j.par from the default directory (see 'During the first use' chapter): The first line must look like *2000 where the number gives the maximum number of named objects. If j.par is not available, the default number of objects is 5000. Thereafter there can be in j.par file any number of **J** commands executed directly (e.g. you can give shortcuts for commands which are handy e.g. when including repeatedly certain sections from include files). If you want to go directly into a specific J-application, you can put into j.par the corresponding include command. If **J** is started from command prompt, there can be an include file name in the command line. This include file is run after commands read from j.par if it is available. ?TESTAA

2.8 During the first use

It is reasonable to have the exe versions and dll's found in **J**bin folder in one folder, and to make shortcuts for exe-files into all working folders. Edit the properties of the shortcut (right click the shortcut icon) so that the starting directory is the working directory. Or alternatively you can set the path to the folder containing **J** executable and run **J** in your working directory. Copy also the file j.par into each working directory. It is recommended that **J** is run in the Command Prompt window even if it can be run also directly from the program shortcut. Edit first the properties of the I/O window if you are using **J** directly or from the Command Prompt window. The properties of the I/O window can be changed by right-clicking the-icon at the upper left corner. It is reasonable to make the screen buffer rather large (large height) so that the whole history of the **J** session can be seen (this is done in the layout sheet of the shortcut properties). The default height of the I/O window is also probably too small. The width should be at least 81. If you would like to use mouse in copy and paste, put quick edit option on. Also the colours of the text and background of the **J** window should be made healthier for eyes (dark text, bright background). To see that **J** is running properly, give your first commands at sit> prompt: sit>a=7.7; ! The result

should look a=7.700000 sit> It is possible to use arrow keys to gwt previous command lines. ! All input lines entered or generated by input programming at sit> prompt are called commands. Commands are either input programming commands (input commands) or commands that define operations in the **J** working environment (operation commands). Input commands and operation commands may read and interpret more input lines before returning control to the command level. It is most convenient to develop **J** applications using include files. There is available in the jdoc-demo folder an include file jexamples.txt which can be used to run all examples in this manual. If the include file is in the working directory you can run any example by wrting the name of the example in the J-window if you have given ';incl(jexamples.txt)' at the sit> pronpt. The working environment of **J** consists of named objects, temporary objects, constants, functions, arithmetic operations and text paragraphs. Operation commands define simple arithmetic operations or more complicated operations on the data structures. Operation commands are defined using a transformation language. In addition to operation commands, the same transformation language is used to define transformation objects which are computed as a group, usually severl times, and which can be linked in different ways to data structures or other transformation objects.

2.9 Exiting J

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

```
sit>end
```

2.10 Typographical conventions

In this manual function names are written in red, option names in blue, object types in capital letters, object names within the text are written in this color: Object. Names like ob1 are used as generic names for object and names like var1 are generic names for REAL variables. If there is no output for an operation command line, the object Result is used as the default output. if the line ends with a single semicolon ';' or double semicolon ';;', then the output may be printed depending MIHIN KOHTAAN In many cases there is no output object, and a possible output given is ignored. If an explicit output is necessary, then '=' is put in front of the function name. Notation â[=]â means that the output can be given but it is not necessary. In most cases the default output Result is then used: In some cases no output is then generated (this will be indicated). For functions that return real values or

matrices which can be used directly in arithmetic or matrix operations, the existence of output is not indicated.

Subroutines obtained from other sources The following subroutines are obtained from other sources.

- **J** 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: http://pages.cs.wisc.edu/∼deboor/pgs/tautsp.f
- **J** real function ppvalu used in **J** function value from Carl de Boor (1978) A practical guide to splines. Springer, New York, p.310-314 No licence restrictions known distribution: http://pages.cs.wisc.edu/∼deboor/pgs/tautsp.f
- **J** subroutine interv , used in function ppvalue from Carl de Boor (1978) A practical guide to splines. Springer, New York, p.310-314 No licence restrictions known obtained from: http://pages.cs.wisc.edu/∼deboor/pgs/tautsp.f Lapack matrix routines
- **J** Several subroutines from www.netlib.org/lapack licence : http://www.netlib.org/lapack/L

2.11 Operation of J

In this section the main tools in code devlopment in a project are presented. It is usefule 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 eaxample of a script file. !The script files should start with shortcuts for each section. If it is 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 defintiosn for different sections. the same For instance, if the the section label is // ;thistask:// then the shortcut definition could be thistaskh=';incl(this,from->thsitask)'//

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 **J** code line, either in the input paragraph defining a transforamtion object or outside it, ends with ';' or ';;', the 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 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 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 transforasmtion 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 8teh 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 jexaples.inc to demosntrate possible error conditions so that the computation continues as fi no error had occured.

[operexr]Exapmple of operation of J.

3 Command input and output

J has two programming levels. First level, called input programming, generates text lines which are then transmitted to the interpreter 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.

3.1 Input record and input line

J reads input records from the current input channel which may be terminal, file or a text object. When **J** 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 ',' ,'+', '*Â´, '-', '(', '=' 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. 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). The continuation line cannot start with â*â or â!â because these are reserved to indicate comments. Note: '/' (division)cannot be used as last character indicating the continuation of the line because it can be legal last character indicating the end of an input paragraph.

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 **J** 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 **J** 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.

3.2 Input Paragraph

Many **J** functions interpreted and executed 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 ends 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. An input paragraph can contain input programming commands; the resulting text lines are transmitted to the **J** function which interprets the input paragraph [inpuparag]Example

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

3.3 Command 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, **J** excutes the command. The command can be either input programming command or ??? command. The file jexamples.inc shows an useful way to organize shortcuts and include files. [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>'
;return
//
write('shortex.txt',$,te)
;incl(shortex.txt)
ju1
ju2
delete_f('shortext.txt')
te=0 !delete also text object te
```

3.4 Input programming

The purpose of the input programming is to read or generate **J** commands or input lines needed by **J** 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 **J** functions, but the input programming functions cannot have an output. There are also controls structures in the input programming. An input paragraph can contain input programming structures.

3.4.1 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 cannot be other text but not commands on the label line.

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

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 addresses and code labels. It is up to input programming what code alabels become part of the transformation object.

3.4.2 Changing âiâ 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:\j\' name='area1' extension='svs' then in->'"directory""name"."extension"' is equivalent to in->'D:\j\area1.svs' If the "-expression is not a character variable then **J** 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 ää 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->äy=ä[f5.2]coef(reg,x1)ä*x1+ä[f5.2]coef(reg,1)ää) See file jex.txt and Chapter 8 for an ex

3.4.3 ;incl()

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().he
from	N|1	Ch gives the starting in-> label for the inclusion, label is given without starting ';' and ending ':'.
wait	N|0	 **J** waits until the include file can be opened. Useful in client server applications. See chapter **J** as a server.

Include files can be nested up to 4 levels.evels See Chapter Defining a text object with text function and using it in ;incl how to include commands from a text object. When editing the include file with Notepad ++, it is reasonable to set the language as Fortran (free form). [inpuincl]Example of ;incl()

```
 file=text()
i=1;
```

```
goto(ad1)
i=2;
ad1:i=66;
goto(ad2,ad3,2)  !select label from a label list
ad2:
i=3;
ad3:i=4;
goto(5)  !select label from the list of all labels
ad4:i=5;
ad5:i=6;
//
write('file.txt',file)
close('file.txt')
;incl(file.txt)
;incl(file.txt,from->ad2)  The adress line can contain comment starting with
'!'.
```

3.4.4 ;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:	

```
[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')
```

3.4.5 ;return returns from include file

;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 address 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 either an explicit ;incl or ;goto command or generating these commands commands with command shortcut.

3.4.6 ;do()

Generates new input records and replaces text with other text using " " to generate numbers, @list to generate lists of object names and @list(elem) to pick the names of the elements of a list, or ;sum() to generate sums and ;dif() to generate differences.

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.	

```
[inpudoex]Examples of ;do()
 ;do(i,1,2)
x"i"="i"*10
print('Greetings from iteration "i"')
;enddo
print(x1,x2)

!After dropping out extra text about the processing we get:
<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
```

3.4.7 ;sum()

J can 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. [inpusumex]Example of ;sum()

```
 su=';sum(i,1,5)(a"i"*x")'
print(su)

<print(su)
'a1*x1+a2*x2+a3*x3+a4*x4'

prob=problem()
;sum(i,1,5)(a"i"*x"i")==max

<prob=problem()
prob< a1*x1+a2*x2+a3*x3+a4*x4+a5*x5==max
```

;dif()() works similarly for minus

3.4.8 ;dif()

J 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.

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

3.4.9 ;pause

Including input from an include file can be interrupted using an input programming command ;pause promt or the **J** function pause('<prompt>'). In both cases the user can give **J** 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 **J** 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> **J** 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 **J** procees similarly as if an error had occured, i.e. print error messages and returns control to sit> -promt.

3.4.10 ;return returns from include file

;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:

- **J** The conrol came to the starting address 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.
- **J** The control came to the starting label from within the same include file using either an explicit ;incl or ;goto command or generating these commands commands with command shortcut.

3.5 Generating sequences with ... -construct

It is often natural to index object names, and often we need to refer object names having consecutive index numbers or index letters. In **J** 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 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. [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
```

4 J functions: an overview

J functions are organized into 21 groups. Some of of the functions are such that the interpreter converts the code given by the user into new function names. For instance the code 1+2 is first changed into PLUS(1,2), and code a(3) is changed into form getelem(a,3). The following list contains also such implicit function names to support also those users who are interested to start looking at the open source code. The ordering of functions in this manual is the same as in the file jmodules.f90 in order to make it easier to start looking the source code, even if the order is not the most logical from the application point of view. When adding new functions into any group is easy when the new function is put to the end of group so new functions are not inserted in the middle of group even if it would more logical. !

- Special functions
  - setoption
  - getelem
  - setelem
  - list2
  - o1_funcs
  - o2_funcs
  - o3_funcs

  - setcodeopt
- Objects
  - type
  - delete_o
  - exist_o
  - name
- Transformations
  - trans
  - call
  - pause
  - noptions
  - R
- Loops and controls structures
  - do
  - if
  - ASSIGN
  - sit
  - which
  - errexit
  - goto
  - itrace
  - trace
  - tracenow
  - itraceoff
  - traceoff
  - tracetest
  - assone
  - enddo
  - assmany
  - goto2

  - goto3

- Arithmetic and logical operations after converting to the polish notation
  - HMULT
  - HDIV
  - IPOWER
  - MULT
  - DIV
  - PLUS
  - MINUS
  - EQ
  - NE
  - LE
  - LT
  - GE
  - GT
  - NOT
  - AND
  - OR
  - EQV
  - NEQV
  - POWER
- Arithemetic functions which can operate on scalars or on matrices
  - max
  - sign
  - mod
  - nint
  - int
  - ceiling
  - floor
  - sqrt

  - sqrt2
  - log10
  - exp
  - sin
  - sind argument is in degrees, also in the following
  - cos
  - cosd
  - tan
  - tand
  - cotan
  - cotand
  - asin
  - asind
  - acos
  - acosd
  - atan
  - atand
  - acotan
  - acotand
  - sinh
  - cosh
  - tanh
  - fraction
  - abs
- Special arithemetic functions
  - der
  - gamma
  - loggamma
  - logistic
  - npv
- Probability distributions

  - pdf
  - cdf
  - bin
  - negbin
  - density
- Random numbers
  - ran
  - rann
  - ranpoi
  - ranbin
  - rannegbin
  - select
  - random
- Interpolation
  - interpolate
  - plane
  - bilin
- List functions
  - list
  - merge
  - difference
  - index
  - index_v
  - len
  - ilist
  - putlist
- Creating a text object
  - text
  - txt

- File handling
  - exist_f
  - delete_f
  - close
  - showdir
  - setdir
  - thisfile
  - filestat
  - read
  - write
  - print
  - ask
  - askc
  - printresult
  - printresult2
- Matrices
  - matrix
  - nrows
  - ncols
  - t
  - inverse
  - solve
  - qr
  - eigen
  - sort
  - envelope
  - ind
  - mean
  - sum
  - var
  - sd

  - minloc
  - maxloc
  - cumsum
  - corrmatrix
- Data functions
  - data
  - newdata
  - exceldata
  - linkdata
  - getobs
  - nobs
  - classvector
  - values
  - transdata
- Statistical functions
  - stat
  - cov
  - corr
  - regr
  - mse
  - rmse
  - coef
  - r2
  - se
  - nonlin
  - varcomp
  - classify
  - class
- Linear programming
  - problem

  - jlp
  - weights
  - unit
  - schedcum
  - schedw
  - weight
  - partweights
  - partunit
  - partschedcum
  - partschedw
  - partweight
  - priceunit
  - weightschedcum
  - priceschedcum
  - priceschedw
  - weightschedw
  - integerschedw
  - xkf
- Plotting figures
  - plotyx
  - draw
  - drawclass
  - drawline
  - show
  - plot3d
- Splines, stem splines, and volume functions
  - tautspline
  - stemspline
  - stempolar
  - laasvol

  - laaspoly
  - integrate
- Bit functions
  - setbits
  - clearbits
  - getbit
  - getbitch
  - bitmatrix
  - setvalue
  - closures
- Misc. utility functions
  - value
  - properties
  - cpu
  - seconds

5 Special functions

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

5.1 setoption(): set option on

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

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

5.2.1 Get or set a matrix element or submratrix

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

One can get or set matrix elements and submatrices as follows. If the expression is on the right side of '=' then **J** gets a REAL value or submatrix, if the expression is on the left side of '=', the **J** 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 sepcified using explicitly 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 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. [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,sum->2);
```

C=A(1,3,4...2);
H=matrix(4,4,diag->,do->3);
H(3,-4,diag->)=matrix(2,values->(4,7)); When giving range, the lower and upper limit can be equal.qual

5.3 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)=..). If someone starts to use the own function property of the open source J, she/he probably would like to get the possiblity to put information into from her/his object types also. To implement this property requires some co-operation from my side. In effect the getelem() and setelem() functions are excuted in the same getelem() subroutine, because bot functions can utilize the same code.

5.4 list2()

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

5.5 setcode(): Initialization of a code option

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

5.6 Own functions: o1_o1_funcs(), o2_funcs() and o3_funcs()

The users of open **J** can define their own functions using three available own-function sets. In addition to own functions open **J** is ready to recognize also own object types and options which are defined in the source files controlled by the users. The main **J** does not know what to do with these own object types and options, they are just transmitted to the own-functions. In the main **J** the control is transmitted to the own functions using **J** functions o1_o1_funcs(), o2_funcs() and o3_funcs().

6 Functions for handling objects

The following functions can handle objects.

6.1 Get type of an object: type()

! The index of any object can be access by type(object). ! If the argument is a character variable c
! 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 taht name, then type() returns
-1 and no error is generated. [typeex]Example of type

```
 ttt=8;  !REAL
type(ttt);
type('ttt'); !type is CHAR
type('ttt',content->);
cttt='ttt'
type(cttt);
type(cttt,content->);
```

6.2 Deleting objects: delete_o()

When an object with a given name is created, the name cannot be removed. With delete_0() function one can free all memory allocated for data structures needed by general objects: delete_o(obj1,â¦,objn) After deleting an object, the name refers to a real variable (which is initialized by the delete_o() function into zero). Other objects except matrices can equivalently be deleted by giving command obj1,â¦,objn = 0 This is because the output objects of any functions are first deleted before defining them anew. Usually an object is automatically deleted if the object name is as an output object for other functions. One can see how much memory each object is using print(Names).mes Deleting a compound object deletes also such subobjects which have no meaning when the main object is deleted. But e.g. if a data object is deleted then the as-sociated transformation object is not deleted as the transformation can be used independently. Files can be deleted with delete_f(file). See IO-functions for details. If the user has defined own new compound objects in the open source **J** software she/he needs to define the associated delete function.

6.3 $_o$*exist_o*() : *doesanobjectexist*

looks whether an object with the name given in the character constant argument exists. In the prvious versions of **J** same function was used for files and objects.

6.4 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.

7 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.

7.1 Generating a transformation object trans()

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. **J** 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\ob[. 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 objec.

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.

!

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

Options input->, local->, matrix->, arg->, result->, source-> of previous versions are obsolte. 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. The value of Printresult can be changed in other parts of the transformation, or in other transforamations called or during execution of pause().

Output variables in maketrans-> transformations whose name start with $ are not put into the new data object. [transex]Demonstrates also error handling

```
 tr=trans()
$x3=x1+3
x2=2/$x3;
/
tr%input,tr%output,tr%source;
x1=8
call(tr)
tr2=trans(x1,x2)
$x3=x1+3
x2=2/$x3;
x3=x1+x2+$x3;
/
tr2%input,tr2%output,tr2%source;
call(tr2)
tr23;  !x3 is now local
tr3=trans()
x1=-3
call(tr) !this is causing division by zero
/
Continue=1  ! continue after error
call(tr3)
```

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


<tr=trans()
<$x3=x1+3
```

```
<x2=2/$x3;
</
<tr%input,tr%output,tr%source;
tr%input is list with            2  elements:
x1 $x3
tr%output is list with            1  elements:
x2
tr%source is text object:
1 $x3=x1+3
2 x2=2/$x3;
3 /
///end of text object

<x1=8


<call(tr)
x2=0.18181818

<tr2=trans(x1,x2)
<$x3=x1+3
<x2=2/$x3;
<x3=x1+x2+$x3;
</
<tr2%input,tr2%output,tr2%source;
tr2%input is list with            1  elements:
x1
tr2%output is list with            1  elements:
x2
tr2%source is text object:
1 $x3=x1+3
2 x2=2/$x3;
3 x3=x1+x2+$x3;
4 /
///end of text object

<call(tr2)
x2=0.18181818

tr2\x3=19.1818181
```

```
<tr2\x3;
tr2\x3=19.1818181

<tr3=trans()
<x1=-3
<call(tr)
</
<Continue=1
<call(tr3)
*division by zero
*****error on row            2  in tr%source
x2=2/$x3;
recursion level set to    3.0000000000000000

*****error on row            2  in tr3%source
call(tr)
recursion level set to    2.0000000000000000

*err* transformation set=$Cursor$
recursion level set to    1.0000000000000000
****cleaned input
call(tr3)
*Continue even if error has occured
<;return
```

7.2 Excuting transformation object explicitly call()

Interpreted transformations in a transformation object can be automatically executed by other **J** functions or they can be executed explicitly using call() function.

Arg	1	TRANS
	The transformation object executed.	

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. [recursion]Recursion produces system crash.

```
 tr=trans() !level will be initialized as zero
level;
level=level+1
```

```
call(tr)
/
Continue=1 !error is produced
call(tr)
Continue=0
```

7.3 Pause: pause()

Function pause() stops the execution of **J** 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 **J** 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. 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.

7.4 Number of options in the current function: noptions()

noptions() returns as a REAL value the number of options currently active. This may be used for testing purposes. When **J** processes options in different funtions, it varies at which point options are cleared and number of options decreases

8 Co-operation between J and R

Is is possible to run R scripst from **J** and **J** scripts from R.

8.1 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. [Rex]Example of Rscript

```
 rscript=text()
# A simple R-script that generates a small data to file mydat.txt
wd<-"C:/j3/"
```

```
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')
```

8.2 Calling J-scripts from R

File JR_0.0.tar.gz in the folder J_R contains R tarball for taking **J** subroutines into R. With R command JR(âtestr.incâ) the example jp problem can be solved from R. Later lauri Mehtatälo will develop this cooention further so tha R can directly access also matrices in the **J** memory. For further information contakt lauri.mehtatalo@luke.fi

9 Loops and control strucures

This section describes nonstadard functions.

9.1 do() loops

The loop construction in **J** looks as follows:

```
do(i,start,end[,step])
enddo
```

cycle and exit are implemented in the current **J** version with goto()oto Within a doâloop there can be cycleand exitdostatements There can be 8 nested loops. do-loop is not allowed at command level.evel [doex]do-loop

```
 'begin';
tr=trans()
do(i,1,5)
i;
ad1: if(i.eq.3)goto(cycle)
i;
if(i.eq.4)goto(jump)
cycle:enddo
jump:i;
!goto(ad1) ! it is not allowed to jump into a loop
```

/
call(tr)

9.2 if()

if()j_statementâ¦
The one line if-statement.

9.3 assignment: 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.

[assignex]Examples of assignments
a=matrix(2,3);
a=4;
a=rann();
v1...5=2...6;
v1...5=77;
Continue=1 !ERROR
v1..3=1,5;
Continue=0

9.4 Selecting a value based on conditions.

Usage://
output=which(condition1,value1,...,conditionn,valuen) // or// output=which(condition1,value1
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. [whichex]Example of which()

```
 c=9
which(a.eg.3.or.c.gt.8,5,a.eq.7,55);
a=7
which(a.eg.3.or.c.gt.8,5,a.eq.7,55);
a=5
which(a.eg.3.or.c.gt.8,5,a.eq.7,55);
which(a.eg.3.or.c.gt.8,5,a.eq.7,55,108);
```

9.5 errexit()

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

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

9.6 goto()

Control can be transfered to a line in a transformation set with goto(). [gotoex]

```
 tr=trans()
i=0
if(i.eq.0)goto(koe)
'here';
```

```
koe:ch='here2';
/
call(tr)
ch;
```

It is not allowed to jump in to a loop or into if -then structure. This is checked already in in the interpreter. It is not yet possible to continue within an include file using Continue=1. It is not recommended to use goto() according to modern computation practices. However, it was easier to implement cycle and exitdo with goto(), especially if cycle or exitdo does not apply hte innermost do-loop.

10 Arithmetic and logical operations

!The arithmetic and logical operations are first converted into the polish notation. 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 +.

10.1 Minimum and maximum: 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

11 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:

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

11.1 mean(): means or weighted means of matrix columns

See section matrixstat for details

11.2 sd(): sd's or weighted sd's of matrix columns

See section matrixstat for details

11.3 var(): Sample variances or weighted variances of matrix c

See section matrixstat for details

11.4 sum(): sums or weighted sums of matrix columns

See section matrixstat for details

12 Special arithmetic functions

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

12.1 Derivatives with 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.

Output

The der() function does not have an explicit output, but der() accompanied with the function produces REAL]d[[] variable for each of the argument variables.

Args 1- REAL

]d[[Argi] variable will get the value of the derivative wiht respect to the argument Argi.

```
[derex]Derivatives with der()
 tr=trans()
der(x)
f=(1+x)*cos(x)
/
fi=draw(func->tr(d[x]),x->x,xrange->(0,10),color->Blue,continue->)
fi=draw(func->tr(f),x->x,xrange->(0,10),color->Cyan,append->,continue->fcont)
[derex2]2
 X=matrix(do->(0,1000,10))
e=matrix(nrows(X))
e=rann(0,2);
A,Pmax,R=0.1,20,2
A*Pmax*1000/(A*1000+Pmax);
Y=A*Pmax*X/.(A*X+Pmax)-R+e !rectangular hyperbola used often for photosynthesis
rect=trans()
der(A,Pmax,R)
f=A*Pmax*I/(A*I+Pmax)-R
/
fi=draw(func->(rect(f)),x->I,xrange->(0,1000),color->Orange,width->2,continue-
>,show->0)
da=newdata(X,Y,e,extra->(Regf,Resid),read->(I,P,er))
stat()
fi=plotyx(P,I,append->,show->0,continue->fcont)
A,Pmax,R=0.07,17,3 !initial values
fi=draw(func->(rect(f)),x->I,xrange->(0,1000),color->Green,width->2,append-
>,show->0,continue->)
reg=nonlin(P,f,par->(A,Pmax,R),var->,corr->,data->da,trans->rect)
```

```
reg%var;
reg%corr;
corrmatrix(reg%var);
fi=draw(func->(rect(f)),x->I,xrange->(0,1000),color->Violet,append->,continue->fcont)
```

12.2 gamma function: gamma()

Function gamma() produses the value of gamma funtion for a positive argument. The function utilises gamma subroutine from library dcdflib in Netlib.

12.3 logistic(): value of the 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 1-1/(1+exp(x)). Because the logistic function can be needed in the nonlinear regression, also the derivatives are implemented. Note, to utilse 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 [logisticex]Example of logistic function

```
 tr=trans()
der(a,x0)
f=logistic(a*(x-x0));
/
x,x0,a=10,5,0.1
d[a],d[x],tr(d[x0]);
```

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.

12.4 npv(): net present value

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

13 Functions for probality distributions

There are currently the following functions relate to probality distributions. function density() can be used define density or probality function for any continuous or discrete distribution which can then be used to generate random numbers with random() function.

13.1 Density for normal distribution: pdf()

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.	

See example drawclassex for an utilization of pdf()

13.2 Cumulative distribution function for normal and chi2: cdf

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

13.3 bin(): binomial probability

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

13.4 negbin():: negative binomila distribution

negbin(k,myy,theta)// Gives the probability that a negative binomial random variable has value k when the variable has mean myy and variance myy+]theta[*]myy[**2. negbin(k,n*p,0)= bin(k,n*p). Sorry for the parameter inconsistency with rannegbin().

13.5 density() define

any discrete or continues distribution 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

Actually the function generates a matrix having towo rows which has values for the cumulative distribution function. When defining the density function, the user need not care about the scaling constant which makes the integral to integrate up to 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)
```

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

14 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.

14.1 ran(): uniform random number

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().

```
[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;
```

14.2 rann(): normal random variate

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

Output	1	REAL|MATRIX The matrix to be generated bus be defined earlier with matrix().cm er
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.

[rannex]Random normal variates, illustrating also find

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

! 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.

14.3 ranpoi(): random Poisson variables

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

14.4 ranbin(): random binomial values

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. (J 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. !

```
[ranbinex]x
 ranbin(10,0.1);
ranbin(10,0.1);
!
A=matrix(1000,2)
A(All,1)=ranbin(20,0.2)
A(All,2)=ranbin(20,0.2)
da=newdata(A,read->(s1,s2))
stat()
cl=classify(1,x->s1)
fi=drawclass(cl,histogram->,color->Blue,continue->fcont)
cl=classify(1,x->s2)
fi=drawclass(cl,histogram->,color->Red,append->,continue->fcont)
```

14.5 rannegbin(): negative binomial variates

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

Output	1	REAL | MATRIX the number of successes in independent Bernoul trials before râth failure when p is the probability of success. ranbin(r,1)returns 1.7e37 and ranbin(r,0)returns 0.
!Args		REAL r=Arg1 and p=Arg1

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 âoverdispersed Poissonâ variables. Sorry for the (temporary) inconsistency of parameters with function negbin(). 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.

14.6 select(): Random selection of elements

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

[selectex]Random selection
```
 S=select(500,10000)
mean(S),sum(S),500/10000;
```

14.7 random(): random variates from any distribution

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

15 Functions for interpolation

The following functions can be used for interpolation

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

15.2 plane() interpolation from 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.

15.3 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

16 List functions

The following list functions are available

16.1 Object lists

An object list is a list of named **J** 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.

16.2 list() generates a LIST object

Output	1	LIST
	The generated LIST object.	

Args	0-	named objects. If an argument is LIST it is ex+panded
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.

The same object may appear several times in the list. (see merge())ge There may be zero arguments, which result in an empty list which can be updated later. The index of object in a LIST can be obtained using index().ex

```
li=list(x1...x3);
index(x2,li);
Continue=1
index(x4,li); ! error
Continue=0

    [list2ex]x
 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
```

16.3 merge()

merge() will produce of list consisting of separate objects in argument lists and argument objects.

Output	1 LIST	A list which is produced by first putting all elements of argument lists and non-list arguments into a vector, and then duplicate objects are dropped.

Args	2-	LIST|OBJ	LIST and separate non-list objects.

```
[mergex]Merging list
 x1...x3=1,2,3
mat=matrix(3,values->(4,5,6))
lis0=list(x2,x1)
lis2=merge(x1,mat,lis0)
print(lis2)
```

```
<print(lis2)
lis2 is list with            3  elements:
x1 mat x2
```

16.4 Difference of LIST objects

difference() removes elements from a LIST

Output	1	LIST	the generated LIST.
Args	2	LIST|OBJ	The first argument gives 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 remode from the lisrt.

```
[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
```

16.5 index(): index of a variable in a data object

To be documented later

16.6 index(): index of a variable in a data object

To be documented later

16.7 len() lengths of different lists or vectors

len(arg) gives the following lenghts for different argument types

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

16.8 ilist(): list of integers

Generates a list of integers which can be used as indexes. This function is used implicitly with .

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 object 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

ILIST is a new object whose all utilization possiblities are not yet explored. It will be used e.g. when developing factory optimization. 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. [ilistex]ILIST examples
1,4,5;
4...1;
A=matrix(4,4)
A(1,5,3=

16.9 putlist()

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

17 Creating two types of text objects

The are now two object types for text.

17.1 text() creates the old TEXT object

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

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)

17.2 txt() generates the new TXT object.

Works as text(), to be documented later. The new TXT object is used to implement ;incl and Gnuplot -figures.

18 File handling

The following function can handle files.

18.1 $_o$*exist_o*() : *doesanobjectexist*

looks whether an object with the name given in the character constant argument exists. In the prvious versions of **J** same function was used for files and objects.

18.2 Deleting objects: delete_o()

When an object with a given name is created, the name cannot be removed. With delete_0() function one can free all memory allocated for data structures needed by general objects: delete_o(obj1,â¦,objn) After deleting an object, the name refers to a real variable (which is initialized by the delete_o() function into zero). Other objects except matrices can equivalently be deleted by giving command obj1,â¦,objn = 0 This is because the output objects of any functions are first deleted before defining them anew. Usually an object is automatically deleted if the object name is as an output object for other functions. One can see how much memory each object is using print(Names).mes Deleting a compound object deletes also such subobjects which have no meaning when the main object is deleted. But e.g. if a data object is deleted then the as-sociated transformation object is not deleted as the transformation can be used independently. Files can be deleted with delete_f(file). See IO-functions for details. If the user has defined own new compound objects in the open source **J** software she/he needs to define the associated delete function.

18.3 close() closes a file

close(file) closes an open file where file is either a character constant or character variable associated with a file. No open(9 function is needed. An file is opened when it is first time in write().te if the file exists, it is asked whether the old file is dleted.

18.4 showdir() shows the current directory

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

18.5 setdir() sets the current directory

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

18.6 thisfile() returns the name of the current include 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

18.7 filestat() gives 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

19 Io-functions

Theree are following io functions

19.1 read() read from a file

read(file,format[,obj1,â¦,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.// '(â¦.)' a Fortran format. Does not work when reading matrices. $ the * format of Fortran// obj1,â¦,objn **J** 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 **J** is waiting until the file can be opened. Useful in client-server applications. See chapter **J** as a server. Use ask() or askc() to read values from the terminal when reading lines from an include file. When reading matrices, their shapes need to be defined earlier with matrix() hfunction.

19.2 write()

write(file,format,val1,â¦,valn[,tab->][,rows->])! case[1/6] Writes real values to a file or to the console. If val1 is a matrix then this matrix (or usually vector) is written or at most as many values as given in the values-> option. Arguments: file variable $ (indicating the console), or the name of the file as a character variable or a character constant, or variable $Buffer format $ indicates the '*' format of Fortran, works only for numeric values. A character expression, with the following possibilities: A format starting with 'b' will indicate binary file. Now 'b' indicates ordinary unformatted write, later there will be other binary formats A Fortran format statement, e.g. (∼the values were ∼,4f6.0), with this format pure text can be written by having no object to write (e.g. write('out.txt','(∼kukuu∼)')). For these formats, other arguments are supposed to be real variables or numeric expressions or there is a matrix argument. If they are not, then just the real value which is anyhow associated with each **J** object is printed (usually it will be zero). If the val1 argument is a matrix, then all values are printed. val1,â¦,valn real values Options: tab 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. rows If val1 is a matrix or a vector and rows-> has one argument then at most as many values are written as given in this option, if there are two arguments then the option gives the range of written rows in the form rows->(row1,-row2). If the upper limit is greater than the number of rows, no error is produced, all available rows are just written. write(file,'t',t1,val1,t2,val2,â¦,tn,valn[,tab->]) ! case[2/6] Tabulation format. positive tab position values indicate that the value is written starting from that position, negative tab positions indicate that the value is written up to that position. The values can be either numeric expressions or character variables or character constants. Tab positions can be in any order. Arguments: file variable $ (indicating the console), or the name of the file as a character variable or a character constant, or variable $Buffer 't' tabulation format t1,val1,t2,val2,â¦,tn,valn KUVAUS Options: tab option indicates that sequences of spaces are replaced by tab character so that writ-ten text can be easily converted to Ms Word tables.

19.3 print() prints objects

print(arg1,â¦,argn[,maxlines->][,data->][,row->] [,file->][,func->][,debug->])// Print values of variables or information about objects.// Arguments: arg1,â¦,argn arguments can be any **J** objects or values of arithmetic or logical expres-

sions Options: maxlines the maximum number of lines printed for matrices, default 100. 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(âfileâ)) 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 â(15f6.2)â 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 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 **J** objects. The printing format is dependent on the object type. print() function can be executed for the output of a **J** command by writing ';' or ';;' at the end of the line. The excution of implied print() is dependent on the value of Printoutput. If printoutput =0, then the output is not printed, If printoutput =1, then ';' is causing printing, if Printoutput =2 then only ';;'-outputs are printed, and if Printoutput =3, the bot ';' and ';;' outputs are printed.

19.4 ask() asks a value for 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. 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)

[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>')
```

19.5 askc() asks a value for a character variable

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

19.6 printresult() and printresult2() e

The **J** interpreter translates ';' at the end of the line to a call to printresult() function and ';;' to a call to printresult2() The output of a function is printed by writing ';' or ';;' at the end of the line. The excution of implied print() is dependent on the value of Printoutput. If printoutput =0, then the output is not printed, If printoutput =1, then ';' is causing printing, if Printoutput =2 then only ';;'-outputs are printed, and if Printoutput =3, the bot ';' and ';;' outputs are printed. printresult() and printresult2() are simple functions

which just test the value of Printoutput and then call the printing subroutine, if needed.

20 Matrix functions

J contains now the following matrix functions.

20.1 Matrices and vectors

Matrices and vectors are generated with the matrix() function or they are produced by matrix operations, matrix functions or by other **J** 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.

20.2 matrix(): create a 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 gives 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, **J** 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) 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

---

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->

[matrixex]Example of generating matrices
A=matrix(3,

20.3 nrows(): number of rows in MATRIX, TEXT or BITMATRIX

can be used as:

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

If the argument has another object type, and error occurs

20.4 ncols(): number of columns in MATRIX or BITMATRIX

can be used as:

- nrows(MATRIX)
- nrows(BITMATRIX)

If the argument has another object type, and error occurs

20.5 t() gives 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. Multiplying a matrix by the transpose of a matrix can be made by making new operation '*. The argument matrix can also be a submatrix expression.

20.6 inverse()

inverse(A) computes the inverse of a square MATRIX A. 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 A is not a square matrix or REAL, or A is singular according to dgesv. instead of writing c=inverse(a)*b, it is faster and more accurate to write c=solve(a,b)

20.7 solve() solves a linear equation A*x=b

A linear matrix eqaution A*x=b can be solved for x with code// x=solve(A,b) x=solve(A,b) is faster and more accurate than x=inverse(A)*b solve works also if A and b are scalars. This is useful when working with linear systems which start to grow from scalars.

20.8 qr()

Makes QR decomposition of a MATRIX This can be used to study if columns of a are linearly dependent. **J** 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 kâth 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 **J** has not properly interpreted which diagonal elements are zero. In **J** 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.	

20.9 eigen()

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.	

20.10 sort() sorts a 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. It is currently assumed that if there are two keys then the values in first key column have integer values. If key2 is not given and key1 is positive, then the syntax is: sort(a,key->key1). If there is no output, then the argument matrix is sorted in place.lace The argument can be the data matrix of a data object. The data object will remain a valid data object.

20.11 envelope() computes the 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
	gives the number of points if not all points of the input matrix are used	

The transpose of the output can be directly used in frawline() function to draw the envelope The function is using a subroutine made by Alan Miller and found in Netlib

20.12 find()

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 should be seaeched, use submatrix() 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
		Gives 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

[findex]Example of find, illustrating also rann()

```
 !Repeating the example, different results will be obtained
rm=matrix(100)
m,s=2,3
rm=rann(m,s)
print(mean(rm),sd(rm),min(rm),max(rm))
```

```
=    2.4564691829681395
=    3.2549002852383477
=   -5.6481685638427734
=    10.714715003967285
```

```
first=find(rm,filter->($.ge.m+1.96*s))
large=find(rm,filter->($.ge.m+1.96*s),any->)
large2=find(rm,filter->($.ge.m+1.96*s),expand->)
print(first,100*nrows(large)/nrows(rm),100*sum(large2)/nrows(rm))
```

```
first=    12.000000000000000
=    4.0000000000000000
=    4.0000000000000000
```

20.13 mean(): means or weighted means of matrix columns

See section matrixstat for details

20.14 sum(): sums or weighted sums of matrix columns

See section matrixstat for details

20.15 var(): Sample variances or weighted variances of matrix c

See section matrixstat for details

20.16 sd(): sd's or weighted sd's of matrix columns

See section matrixstat for details

20.17 minloc() : locations of the minimum values in columns

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.

20.18 maxloc() : locations of the minimum values in columns

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.

20.19 cumsum(): cumulative sums of matrix columns

cumsum(MATRIX) generates a MATRIX with the same dimesnions as the argument, and puts the cumulative sums of the columsn into the output matrix. If the argument is vector, the cumsum makes a vector having the same form as the argument.

20.20 Making correaltion matrix from variance-covariance amtrix:

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)

21 Working with DATA objects

Data can be analyzed and processed either using matrix computations or using DATA objects. A DATA object is compound object linked to a data MATRIX and LIST object containing variable (column) names, some other information. When data ere used via DATA object in statistical or linear programming functions, the data are processed observarion by observation. It is possible to work using DATA object 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 **J** functions which create DATA objects from files, data() and exceldta(). data() can create hierarchical data objects. Function newdata() creates DATA object from matrices, which temselfs can be picked from data objects. Function linkdata() can link two data sets to make a hierarchical data. 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 matrix operations and newdata() to make DATA object with variable names and matrices. See Simulator section for an example. transdata() function goes through DATA object similarly as statitical functions, but does not serve a specific purpose, just transforamtions defined in the TRANS object refreed with trans-> option are computed. See again the simulator

section. 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.

21.1 data()

Data objects are created with the data() function. Two linked data objects can be created with the same function call (using option subdata-> and options thereafter in the following description). It is recommended that two linked data objects are created with one data() function call only in case the data is read from a single file where subdata observations are stored immediately after the upper data observation. Data objects can be linked also afterwards with the linkdata() function. A data object can created by a data() function when data are read from files or data are created using transformation objects. New data objects can be created with newdata() function from previous data objects and/or matrices. If data objects can created using transformation objects either with data() function or by creating first data matrix by transformation and then using newdata() to create data object.

Output 0|1 Data
Output 0|1 Data Data object to be created. If there is no output then the default is $Data$. It is recommended that this default is used only when only one data object is used in the analysis. !

---

read 0|1- REAL|List
Variables read from the input files or the name of the list containing all variables to be read in. If no arguments are given and there is no readfirst-> option then the variables to read in are stored in the first line of the data file separated with commas.?? Also the â¦-shortcut can be used to define the varaible list. If no arguments are given and there is readfirst-> option then the variable names are read from the second line.

---

in 0- Char
input file or list of input files. If no files are given, data is read from the following input paragraph. If either of read-> or in-> option is given, then both options must be present.

form	-1|1	Char
	Format of the data as follows $ Fortran format '*', the default b Single precison binary bs Single precison binary opened with access='stream' Needed for Pascal files in Windows. B Double precison binary. Char giving a Fortran format, e.g. '(4f4.1,1x,f4.3)' d4 Single precison direct access for Gfortran files. d1 Single precison direct acces for Intel Fortran files.	
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.	
obs	-1|1	REAL
	Variable which gets automatically the observation number when working with the data, variable is not stored in the data matrix, default: Obs. When working with hierarchical data it is reasonable to give obs variable for each data object.	
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.	
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. subdata the name of the lower level data object to be created. This option is not allowed, if there are multiple input files defined in option in->.

subread,â¦,subobs sub data options similar as read->â¦obs-> for the upper level data. (subform->'bgaya' is the format for the Gaya system). The following options can be used only if subdata-> is present

---

nobsw -1|1 REAL
A variable in the upper data telling how many subdata observations there is under each upper level observation, necessary if subdata-> option is present.

---

nobswcum -1|1 REAL
A variable telling the cumulative number of subdata observations up to the current upper data observation but not including it. This is useful when accessing the data matrix one upper level unit by time, i.e., the observation numbers within upper level observation are nobswcum+1,â¦,nobswcum+nobsw

---

obsw -1|1 REAL
A variable in the subdata which automatically will get the number of observation within the current upper level observation, i.e. obsw variable gets values from 1 to the value of nobsw-variable, default is 'obs_variable%obsw'.

---

duplicate -1|2 TRANS
duplicate -1|2 TRANS The two transformation object arguments describe how observations in the subdata will be duplicated. 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 transmit-

ted to the variable Duplicate. These transformations are called also when Duplicate=0. This means that when there is the duplicate-> option, then all transformations for the subdata can be defined in the duplicate transformation object, and submaketrans-> is not necessary.

---

oldsubobs -1|1 REAL

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

---

oldobsw -1|1 REAL

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

---

nobs -1|1 Real

There are two uses of this option. First, a data object can be created without reading from a file or from the following input paragraph by using nobs-> option and maketrans-> transformation, which can use Obs variable as argument. Creation of data object this way is indicated by the presence of nobs-> option and absence of in-> and read-> options. Second, if read-> option is present nobs-> option can be used to indicate how many records are read from a file and what will be the number of observations. Currently reject-> or filter-> can not be used to reject records (consult authors if this would be needed). If there are fewer records in file as given in nobs-> option, an error occurs. There are three reasons for using nobs-> option this way. 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 virtual memory overflow occurs, then it may be possible to read data in as linked buffers are not needed. In case nobs-> option is present and read-> option is absent either maketrans-> or keep-> option (or both) is required.

---

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|0 When reading data from a text file, the first line can contain a header which is printed but othewise ignored
rfcode	-1|0 The data file can contain also J-code which is 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. rfsubhead-> works for subdata similarly as rfhead-> for data. rfsubcode works for subdata similarly as rfcode-> for data If there are both rfhead-> and rfcode-> then rfhead-> is excuted first. rfhead-> and rfcode-> replace readfirst-> option of previous versions which was too complicated.

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 single precision matrix. The number of observations can be obtained by nobs(Data) or by nrows(Data%matrix). See common options section for how data objects used in other **J** functions will be defined. The in-> and subin-> can refer to the same file, or if both are without arguments then data are in the following input paragraph. In this case data() function read first one upper level record and then nobsw-> lower level records. When reading the data the obs->variable (default Obs) can be used in maketrans-

> transformation and in reject-> option and filter-> option, and the variable refers to the number of observation in resulting data object. The variable Record gets the number of the read record in the input file, and can be used in maketrans-> transformations and in reject-> and filter-> options. If subdata-> option is given, variable Subreject gets the number of record in the sub file, and it can be used in submaketrans-> transformations and in subreject-> option and in subfilter-> option. 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. If no observations are rejected, obs variable and Record variable get the same values. If virtual memory overflow occurs, see nobs-> optio. This should not happen easily with the currrent 64-bit application. Earlier versions contained trans-> and subtrans->options which associated a permanent transformation object with the data object. This feature is now deleted because it may confuse and is not really needed. If tranformations are needed in functions they can always be included using trans-> . [dataex]data() generates a new data object by reading data.

```
 data1=data(read->(x1...x3),in->)
1,2,3
4,5,6
7,8,9
/
```

21.2 newdata()

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.

It is not yet possible to drop variables. An error occurs if the same variable is several times in the variable list obtained by combining variables in data sets and read-> variables. An error occurs if the numbers of rows of matrices and observations in data sets are not compatible. Output variables in maketrans-> transformations whose name start with $ are not put into the new data object. [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
/
newtr=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->newtr)
print(new)
```

21.3 exceldata()

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 **J** 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 fot 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;Species c
```

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;species0
```

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.	

21.4 linkdata() links hierarchical data sets

linkdata(data->,subdata->,nobsw->[,obsw->])// links hierarchical data sets.

data	1	DATA
	the upper level data set object	
subdata	1	DATA
	the lower data set object	
nobsw	1	REAL
	the name of variable telling the number of lower level observations for each	
obsw	0|1	REALV
	variable which will automatically get the number of lower level observation within each upper level observation. If not given, then this variable will be the Obs-variable of the upper level data.	

In most cases links between data sets can be either made using sub-options of data() function or linkdata() function. If there is need to duplicate lower level observations, then this can be currently made only in data() function. Also when the data for both the upper level and lower level data are read from the same file, then data() function must be used. When using linked data in other functions, the values of the upper level variables are automatically obtained when accessing lower level observations. Which is the observational unit in each function is determined which data set is given in data-> option or defined using Data list. In the current version of **J** it is no more necassary to use linked data sets in jlp() function, as the treatment unit index in data containing both stand and schdedule data can be given in unit-> option

21.5 getobs() loads an 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.

dataset	1	DATA
	the DATA object	
obs	1	REAL
	row number in the data matrix of the dataset	
trans	-1|1	TRANS
	these transformations are also executed.	

21.6 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().

21.7 classvector()

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 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 Â§Arg be the generic name for arguments.	
class	1	REAL
	.oindent class 1 REAL	
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 Â§Class changes, the class changes.	
data	0|1	Data

	Data object used. Only one data object 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 Â§Class%%first if expand-> is present and Â§Class%first if expand-> is not present.
last	0 The the number of lastt observation in class is stored in vector Â§Class%%last if expand-> is present and Â§Class%last if expand-> is not 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 Â§Arg#Class%%mean with expand-> and without ext-> Â§Arg#Class%%meanExt with expand-> and with ext-> are Â§Arg#Class%mean without expand-> and without ext-> Â§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.

Numbers of observations in each class can be obtained by Class%nobs=Class%%last-Class%%first+1 when expand-> is present, and Class%nobs=Class%%last-Class%%first+1 [newclassdata]Making class level data object

```
 classvector(x1,x2,class->stand,data->treedata,mean->,min->)
standdata=newdata(x1#stand%mean,x2#stand%mean,x1#stand%min,x2#stand%min,
read->(x1,x2,x1min,x2min))  [addingclass]Adding class means and deviations from class means
 classvector(x1,x2,class->stand,data->treedata,mean->,expand->)
tr=trans()
relx1=x1-x1mean
relx2=x2-x2mean
/
treedata=newdata(treedata,x1#stand%mean,x2#stand%mean,read->(x1mean,x2mean),
maketrans->tr)
```

21.8 values() extracts values of class variables

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.

The values found will be sorted in an increasing order. After getting the values into a vector, the number of different values can be obtained using nrows() function.

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

21.9 transdata() own transformations for data

transdata() is useful when all necassy computions are put into a TRANS object, and a DATA object is gone throug obsevation by observation. This is useful e.g. when simulating harvesting schdules using a simulator which is defined as an ordinary TRANS object. The whole function is written below to indicate how users' own functions dealing with data could be developped. @@data

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

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

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

return
```

22 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 20.2

22.1 stat()

Computes and prints basic statistics from data objects.

Output	0-1 kokopo	REAL
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 variables of the tran
@@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 are com
transformation or it can be a variable in the data object
@@seecom

---

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.

---

max -1,99 REAL
maxima are stored, works as min->

---

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 filter-> or reject-> opt

---

trans	-1 | 1	TRANS transformation object which is executed for each observation. If there is a transformation object associated with the data object, those transformations are
filter	-1 | 1	Code logical or arithmetic statement (nonzero value indicating True) describing which observations will be accepted. trans-> transformations are computed before u
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->-option.

1: stat() function prints min, max, means, sd and sd of the mean computed as sd/sqrt(number of observations) 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. [statex]stat() computes minimums, maximums, means and std devaitons
;if(type(data1).ne.DATA)dataex
stat()
stat(area,data->cd,sum->bon20,filter->(site.ge.18.5))
stat(ba,data->cd,weight->area)
stat(vol,weight->(1/dbh***2))

22.2 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
weight	-1|1	CODE

Codeoption for weight of each observation.

the output is not automaticall printed, but it can be printed using ';' at the end of line. The covariance matrix can changed into correaltion matrix with corrmatrix() function. If variable w in the data is used as the weigth, this can be expressed as weight->w [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
dat=newdata(X1...X6,read->(x1...x5))
co=cov(x1...x5);
co=cov(dat%keep);
```

22.3 corr() computes a correlation matrix.

corr(1) works similarly as cov()

22.4 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	
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.

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

22.5 nonlin():: nonlinear regression

To be raported later, see old manual

22.6 varcomp(): variance and covariance components

TO BE RAPORTED LATER, see old manual

22.7 classify()

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. @@data
x	1 REAL The first variable defining classes. minobs 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 **J** 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.
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 **J** 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 @@filter @@reject	
print	-1|1	Real
	By setting print->0, the classification matrix is not printed. The matrix can be utilized directly in drawclass() function.	

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 Variable Accepted gives the number of accepted obsevations.

22.8 class()

Function class() computes the 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.

Either dx-> or classes-> must be given. If both are given, dx-> dominates. If stat() is used earlier for variables including Var1 and options min-> and max-> are present, then xrange->(Var1%min,Var1%max) is assumed.

23 Linear programming functions

The linear programming (LP) functions are called jlp-functions. The jlp-functions can be used to define linear programming problems, solve them and access the results. The jlp-problems are defined using problem() function, and the problems can be solved using jlp() function. There are several ways to access the results. The main interest in jlp-functions may be in the forest planning problems, where a simulator has generate treatment schedules. There are four different applications of jlp() function.

- Small ordinary LP problems with text input.
- Large ordinary LP problems with MATRIX input.
- Forest planning problems without factories.
- Forest palnning problems with factories.

The problem definition for each case is generated with problem() function. These applications are presented in different subsections

23.1 Problem definition object

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

23.2 problem() defines a Lp-problem

An LP-problem is defined in similar way as a TEXT object// problem([repeatdomains->])// /

```
Define a lp problem for jlp() function.
Output:
a problem definition object
OOption:
repeatdomains
if this option is given then the same domain definition can be in several pla
```

the problem definition, otherwise having the same domain in different places
causes an error (as this is usually not what was intended). If the same domai
definition is in several places is slightly inefficient in computations, e.g
function computes and prints the values of x-variables for each domain defin
even if the same values have been computed and printed for earlier occurrenc
of the domain definition.
The problem definition paragraph can have two types of lines: problem rows a
Examples of problem definitions showing the syntax.

```
pr=problem() !ordinary lp-problem
7*z2+6*z3-z4==min
2*z1+6.1*z2 >2 <8+ !both lower and upper bound is possible
(a+log(b))*z5-z8=0
-z7+z1>8
/
prx=problem() ! timber management planning problem
All:
npv.0==max
sitetype.eq.2: domain7:
income.2-income.1>0
/
```

In the above example domain7 is a data variable. Unit belongs to domain if th
variable domain7 is anything else than zero.
The objective row must be the first row. The objective must always be present
is to just get a feasible solution without objective, this can be obtained by
function.
In problems having large number of variables in a row it is possible to give
a vector and variables as a list e.g.
ESIMERKKI
In problems with x-variables it is possible to maximize or minimize the objec
constraints. In factory problems this would also be quite straightforward to
In problems with x-variables it is possible to maximize or minimize the objec
constraints. In factory problems this would also be quite straightforward to
does not come as a side effect of computations as in the case of maximization
and thus it has not been implemented. The maximization of a factory objective
constraints can be obtained by adding to the problem constraints which requi
amounts of transported timber assortments to different factories are greater
zero.
Function problem() interprets the problem paragraph, and extracts the coeffi
variables in the object row and in constraint rows. The coefficients can be c
arithmetic statements utilizing the input programming "-sequence or enclosi
in parenthesis. The right hand side can utilize arithmetic computations with

The values are computed immediately. So if the variables used in coefficient
values later, the problem() function must be computed again in order to get u
coefficients. Note that a problem definition does not yet define a JLP task.
is possibly only when the problem definition and simulated data are linked in
function. At the problem definition stage it is not yet known which variables
which are x-variables and which are factory variables (see Lappi 1992).
NNote that < means less or equal, and > means greater or equal. The equality
part of linear programming constraints.
The logic of jlp() function is the same as in the old JLP software. There is o
which makes the life a little easier with J. In J the problem definition can u
are defined in the stand data. These are used similarly as if they would beco
constraints like
vol#1-vol#0>0
where vol#0 is the initial volume, i.e. a c-variable, and vol#1 is the volume
In old JLP these initial values had to be put into the x-data.
NNote also that problem definition rows are not in one-to-one relation to the
the final lp problem. A problem definition row may belong to several domains
taking multiple domains in domain definition rows into account is called exp
Domain definitions describe logical or arithmetic statements indicating for
units the following rows apply. Problem will generate problem definition obj
described below.
Starting from J3.0 it is also possible to specify the period of the row for ea
#2# income.2-income.1 >0
If the row contains x variables from several periods, the period of the row i
the x variables. If the period is given for some rows containing x variables,
all except for the objective row. The period of the objective is assumed to b
having any other period for the objective would not make any sense. If wrong
for a row, J computes the correct solution but not as efficiently as with cor
If periods are given for rows, J is utilizing the tree structure of schedules
This leads to smaller amount of additions and multiplications as the computa
of a branch of the tree can for each node utilize the value of branch before t
Unfortunately this was not more efficient e.g. in test problems with five pe
NNote 1: Only maximization is allowed in problems including factories. To ch
minimization problem to a maximization problem, multiply the objective funct
*** We may later add the possibility to define also minimization problems.
NNote 2: If optimization problem includes factories (see chapter 11.2 Optimi
including factories), there have to be variables in the objective function o
constraint row. Example of problem definition including factories can be fou
JLP Examples.
NNote 3: An ordinary linear programming problem contains only z-variables.
NNote 4: It is not necessary to define problem() function if the problem incl

```
For more information see chapter 11.8 Solving a large problem with z-variabl
NNote 5: If the problem contains harvest/area constraints for several domair
if the constraints are written in form
harvest < area_of_domain*constant
instead of
(1/area_of_domain)*harvest < constant.
The latter formulation takes the number of domains times more memory than th
formulation.
```

23.3 Solving large problems without schedule data.

When solving problems including only a large number of z-variables, it is possible to feed the coefficients as a matrix with zmatrix-> option. Unit and schedule data (c- and x-variables) are not allowed when zmatrix-> is used. =jlp(zmatrix->,max->|min->[,rhs->][,rhs2->][,tole->] [,print->][,maxiter->][,test->][,debug->]) Output: Function jlp() generates output row vectors output%zvalues, output%redcost and output column vectors output%rows, output%shprice output is the name of the output. Options: zmatrix Matrix containing coefficients of z-variables for each constraint row. max Vector containing coefficients of z-variables for the objective row of a maximization problem. Either max-> or min-> option has to be defined but not both. min Vector containing coefficients of z-variables for the objective row of a minimization problem. Either max-> or min-> option has to be defined but not both. rhs Vector containing lower bound for each constraint row. Value 1.7e37 is used to indicate the absence of the lower bound in a row. Either or both of the bound options (rhs->, rhs2->) has to be defined. rhs2 Vector containing upper bound for each constraint row. Value 1.7e37 is used to indicate the absence of the upper bound in a row. Either or both of the bound options (rhs->, rhs2->) has to be defined. Other options described above in chapter Solving a problem: jlp( ). NNote. When zmatrix-> option is used, the solution is not automatically printed. Use jlp solution objects to access solution. For more information see chapter 11.10 Objects for the JL enEdnote

There are two versions of jlp() function call: one with problems-> option for problems defined by problem() function and the other with zmatrix-> option for large ordinary linear programming problems with z-variable coefficients defined by matrix. A lp problem defined by problem() function can be solved using jlp() function: [=]jlp(problem->[,data->][,z->][,trans->][,subtrans->] [,tole->][,subfilter->][,subreject->][,class->] [,area->][,notareavars->][,print->][,report->] [,maxiter->][,refac->][,slow->][,warm->][,finterval->] [,fastrounds->][fastpercent->][,swap->][,test->][,debug->] [,memory->]) Output: Necessary for factory problems, otherwise optional. If output is given then function

generates several matrices and lists associated with the solution (e.g. the values of the constraint rows, the shadow prices of the rows, the values of the z-variables, the reduced costs of z-variables, the sums of all x-variables of the data in all domains and their shadow prices, lists telling how problem variables are interpreted. See Objects for the JLP solution for more detailed description. Options: problem problem definition generated by problem() function data data set describing the stand (management unit) data or the schedules data. The unit data set must be linked to schedule data either using sub-options in the data() function or using linkdata() function. Following the JLP terminology, the unit data is called cdata, and the schedule data is called xdata. The jlp() function tries if it can find a subdata for the data set given. If it finds, it will assume that the data set is the unitdata. If subdata is not found, it tries to find the upper level data. If it finds it, then it assumes that the data set given is the schedules data. If data-> is not given, then the problem describes an ordinary lp-problem, and all variables are z-variables. If data-> option is given but no variable found in problem is in the schedules data set, then an error occurs. z If the data-> option is given then the default is that there are no z-variables in the problem. The existence of z-variables must be indicated with z-> option (later the user can specify exactly what are the z variables, but now it is not possible). The reason for having this option is that the most jlp-problems do not have z variables, and variables which **J** interprets as z-variables are just accidentally missing from the data sets. trans transformation set which is executed for each unit. subtrans transformation set which is executed for each schedule. NNote: the subtrans transformations can utilize the variables in the unit data and the output variables of trans-> transformations. NNote: transformations already associated with cdata and xdata are taken automatically into account and they are executed before transformations defined in trans-> or subtrans-> options. *** Later we may add the possibility to have several data sets (note that several files can be read into one data object in the data()function) tole the default tolerances are multiplied with the value of the tole-> option (default is thus one). Smaller tolerances mean that smaller numerical differences are interpreted as significant. If it is suspected that jlp() has not found the optimum, use e.g. tole->0.1 ,tole->0.01 or tole->10. subfilter logical or arithmetic statement (nonzero value indicating True) describing which schedules will be included in the optimization. If all schedules are rejected, an error occurs. Examples: filter->(.not.clearcut) , filter- >(ncuts.ge.5), filter->harvest (which is equivalent to: filter- >(harvest.ne.0)). If the subfilter statement cannot be defined nicely using one statement, the procedure can be put into a transformation set which can be then executed using value() function. subreject logical or arithmetic

statement (nonzero value indicating True) describing which schedules will not be included in the optimization. If subfilter-> is given then test applied only for such schedules which pass the subfilter test. If the subreject statement cannot be defined nicely using one statement, the procedure can be put into a transformation set which can be then executed using value() function. Kommentoinut [LR(46]: Repon puolelle Kommentoinut [LR(47]: Repon puolelle[Natural resources and bioeconomy studies XX/20XX] 95 class class->(cvar, cval) Only those treatment units where the variable cvar gets value cval are accepted into the optimization. The units within the same class must be consecutive. area gives the variable in cdata which tells for each stand the area of the stand. It is then assumed that all variables of cdata or xdata used in the problem rows are expressed as per are values. In optimization the proper values of variables are obtained by multiplying area and per area values. Variables of cdata used in domain definitions are used as they are, i.e. without multiplying with area. Variables which are not treated as per area values are given with the notareavars-> option. notareavars If area-> option is given then this option gives variables which will not be multiplied with area. print of output printed, 1 => summary of optimization steps, 2=> also the problem rows are printed, 3=> also the values of x-variables are printed, 9= the pivot steps and the point of code where pivoting is done and the value of objective function are written to fort.16 file (or similar file depending on the operating system). The value 9 can be used in the case where jlp seems to be stuck. From fort.16 file one can then infer at what point debugging should be put on. Some cycling situations are now detected, so it should be rather unlikely that jlp is stuck. report the standard written output is written into the file given in report-> option (.e.g. report->'result.txt'). The file remain open and can be written by several jlp-functions or by additional write() functions. Use close() function to close it explicitly if you want to open it with other program. maxiter maximum number of rounds through all units (default 10000). refac after refac pivot steps the basis matrix is refactorized. The default value is 1000. New option since J3.0. Actually refactorization was present in the first version of **J** but it had to be dropped because the refactorization corrupted some times the factors of the basis matrix. The reason was found and corrected for J3.0. The reason for misbehavior of the refactorization algorithm of Fletcher was such that it never caused problems in ordinary linear programming problems for which Fletcher designed his algorithms. slow if improvement of the solution in one round through all units is smaller than value given in slow-> option, then **J** terminates under condition slow improvement. New option since J3.0. Earlier slow improvement is computed from the tolerances of the problem, and if the slow-> option is not present

these tolerances are still used. If slow-> option gives negative value, then the absolute value of the option indicates per cent change. Note that in large problems the solution is often very long time quite close the actual optimum, and hence the optimization time can be decreased with rather low loss in accuracy using the slow-> option. If slow-> option is give value zero it is equivalent to omitting the option and hence the slow improvement is determined from the tolerances. warm If option is present in ordinary problems with x-variables then the key schedules of previous solution are used as the starting values of the key schedules. In factory[Natural resources and bioeconomy studies XX/20XX] 96 problems also the previous key factories are used as starting values. If there is no previous solution available or the dimension of the key schedules vector (number of treatment units) or the dimensions of the key factories matrix (number of treatment units and number of factories) do not agree with the current problem, warm option is ignored. Thus it is usually safe to use the option always, the only exception is that the factories and number of units do agree with the previous problem even if factories or schedule data are changes. The warm start may reduce solution time perhaps 40-80%. finterval In factory problems the transportations to new factories are checked if the round number is divisible with finterval. The default value is 3. When there have not been improvements during the last round, the value is changed into 1. fastrounds The shadow prices are used to select an active set of schedules which are considered as entering schedules. fastrounds gives the number of rounds using the same active set. The default is 10. When there have not been improvements during the last round, all schedules are used as the active set. fastpercent A schedule belongs to the active set if its shadow price is at least fastpercent % of the shadow price of the best schedule. The default is 90. swap If option is present, the schedules data matrix is written to a direct access binary file. This option may help if virtual memory overflow occurs. The data needs to be used from the file until all inquiry function calls are computed. Thereafter the data can be loaded again into memory using function unswap(). If this does not happen ordinary functions using this data object work a little slower. If a new problem is solved with the swap-> option, there is no need to unswap() before that. In an Intel Fortran application the swap-> option is given without arguments. In a Gfortran application the option must be given in form swap->4. If there is shortage of virtual memory, read note 5 for problem() function before starting to use swap->. test If option is present then jlp() is checking the consistency of the intermediate results after each pivot step of the algorithm. Takes time but helps in debugging. debug determines after which pivot steps jlp() starts and stops to print debugging information to fort.16

file. If no value given, the debugging starts immediately (produces much output, so it may be good to use step number which is close to the step where problems started (print variable Pivots at the error return). debug->(ip1,ip2,ip3) indicates that debugging is put on at pivot step ip1, off at pivot ip2 and the again on at pivot ip3. memory gives the amount of memory in millions of real numbers that can be used to store data needed in solving the problem. In factory problems also the xdata stored in a direct access file are loaded into memory as much as possible. It is not possible to figure out how large number memory option can give, so it must be determined with experimentation. Using disk-> option in data() function and memory-> option in jlp() function makes it in principle possible to solve arbitrary large problems. In practise the ability of double precision numbers cannot store accurately the needed quantities in very huge problems. Kommentoinut [LR(48]: Vain gfortran käännös? Kommentoinut [LR(49]: Lisätty. Oliko tämä mukana julkaistavassa versiossa?[Natural resources and bioeconomy studies XX/20XX] 97 Function jlp() is generating output (amount is dependent on the print-> option) plus a **J**LP-solution stored in special data structures which can be accessed with special **J** functions described below and if output is given then several output objects are created (see Objects for the JLP solution). NNote 1: a feasible solution (without an objective) can be found by minimizing a z-variable (remember z-> option), or by maximizing a unit variable (which is constant for all schedules in a unit). NNote 2: If virtual memory overflow occurs, see first Note 5 for problem() function and then

23.4 weigts() weights of schdules

TO BE RAPORTED LATER

23.5 partweight() weights of split schedules

TO BE RAPORTED LATER

24 Plotting figures

The graphiscs of the current version of **J** is produced Gnuplot. **J** offers nowan alternative interface to Gnuplot, and it is quite easy to add more plotinng routines later.

24.1 Figures

Figures are made using Gnuplot. **J** transmits information into Gnuplot using text files. **J** generates the files using .jfig extension. Necessary files are generated by deleting old files without asking permission. If fig is the output of a figure function, then **J** creates always, except in 3D, file with name fig.jfig and possibly other files with .jfi0, jfi1 etc extensions. All figure functions use the following options.

Output	1	FIGURE
	The FIGURE object created or updated @@figure	

is possible to show a figure using show() function. It can have either the fig object as the argument or the file name of fig.jfig -file. Thus it is possible to edit the file using Gnuplot capabilites. 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'. The default writte by Gnuplot does not look nice and is not implemented. The user can write own legends using label-> option in drawline(). Easiest way to delete an nonmatrix object fi is fi=0, which makes it possible to use append-> also for an fi wihtout the need to construct if then- structures.

24.1.1 Figure object

Graphic functions produce FIGURE objects. Each FIGURE object can consist of several subfigures. Each FIGURE object stores information of x- and y axes, the range of all x- and y-values, and for each sub-figure information of the ranges of x and y in the subfigure plus the subfigure type and the needed data values. Currently, when Gnuplot is used for graphics, most data values are stored in text files which Gnuplot reads. Function plot3d() is plotting 3-d figures without making a FIGURE object. See Plotting figures for more information.

24.2 Scatterplot: plotyx()

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(). @@figu
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.

```
[plotyxex]plotyx()
 xmat=matrix(do->(0,10,0.001))
tr=trans()
y=2+3*x+0.4*x*x+4*rann()
/
da=newdata(xmat,read->x,maketrans->tr,extra->(Regf,Resid))
fi=plotyx(y,x,continue->fcont)
fi=plotyx(x,func->tr(y),mark->3,color->Orange,continue->fcont)
reg=regr(y,x)
fi=plotyx(y,x,show->0)
fi=plotyx(Regf,x,append->,continue->fcont)
fir=plotyx(Resid,x,continue->fcont)
```

With data with integer values, the default ranges of Gnuplot may be hide point at borderlines. fi=plotyx() produces or updates file fi.jfig] which contains Gnuplot commands and file fi.jfi0 containg data.

24.3 Draw a function: draw()

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. @@draw

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	

```
[drawex]Example of draw()
 fi=draw(func->sin(x),xrange->(0,2*Pi),color->Blue,continue->fcont)
fi=draw(func->cos(x),xrange->(0,2*Pi),color->Red,append->,continue->fcont)
if(type(figyx).ne.FIGURE)plotyxex
show(figyx,cont->fcont)
reg0=regr(y,x)
stat(data->datyx,min->,max->)
figyx=draw(func->reg0(),x->x,xrange->,color->Violet,append->,continue->fcont)
tr=trans()
x2=x*x
fu=reg2()
/
reg2=regr(y,x,x2,data->datyx,trans->tr)
figyx=draw(func->tr(fu),xrange->,color->Orange,append->,continue->fcont)
Continue=1 !Errors
fi=draw(func->sin(x),x->x)
fi=draw(xrange->(1,100),func->Sin(x),x->x)
Continue=0  fi=draw() produces or updates file fi.jfig] which contains Gnu-
plot commands and file fi.jfi0 containg data.
```

24.4 Draw values in a matrix generated with 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-calss standard deviations are drawn.
var	N | 0 Within-class sample variances are drawn.
histo	N | 0 Within-class sample variances are drawn.abto5cm
freq	N | 0 Cumulative frequences are drawn.
area	N |0 the histogram is scaled so that that it can be overlayed to density function

[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->,continue->fcont)
cl=classify(Resid,x->x,xrange->,classes->5)
fi=drawclass(cl,color->Blue,continue->fcont)
fi=drawclass(cl,se->,continue->fcont)
fi=drawclass(cl,sd->,continue->fcont)
fi=drawclass(cl,var->,continue->fcont)
fi=drawclass(cl,histo->,area->,continue->fcont)
fi=draw(func->pdf(0,rmse(reg)),x->x,xrange->,append->,continue->fcont) ! xrange comes from s
```

In previous versions of **J** if se-> and sd-> were both present, the error bot bars were plotted. This possibility will be included later.

24.5 Draw a polygon through points: drawline()

drawline() draws a function through points.

Output	1	FIGURE
	The FIGURE object created or updated.	
Args	1-	REAL | MATRIX
	The points which are connected:	
	J x1,...,xn,y1,...,yn The x-coordinates and y-coordinates, $n \geq 1$*Ifthereisonlyoneargumentwhichisamatrixobjecthavingtworows, the*	
J 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.		
	@@figure	
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.	
mark	N | 1	REAL | CHAR
	The mark used in the plot.	
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 mar 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.	

```
[drawlineex]Example of drawline()
 fi=draw(func->sin(x),xrange->(0,2*Pi),color->Blue,continue->fcont)
fi=drawline(Pi,sin(Pi)+0.1,label->'sin()',append->,continue->fcont)
xval=matrix(do->(1,100))
mat=matrix(values->(xval,xval+1,xval,xval+2,xval,xval+3))
fi=drawline(mat,color->Red)
fi=drawline(mat,break,color->Orange,break->,continue->fcont)
x=matrix(do->(0,100,1)
e=matrix(101)
e=rann()
y=2*x+0.4+e
da=newdata(x,y,read->(x,y))
reg=regr(y,x)
if(type(figyx).ne.FIGURE)plotyxex
show(figyx)
reg0=regr(y,x)
stat(data->datyx,min->,max->)
figyx=draw(func->reg0(),x->x,xrange->,color->Violet,append->,continue->fcont)
tr=trans()
x2=x*x
fu=reg2()
/
reg2=regr(y,x,x2,data->datyx,trans->tr)
figyx=draw(func->tr(fu),xrange->,color->Orange,append->,continue->fcont)
Continue=1 !Errors
fi=draw(func->sin(x),x->x)
fi=draw(xrange->(1,100),func->Sin(x),x->x)
```

Continue=0 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->.

24.6 Show figure: show()

An figure stored in a figure object or in Gnuplot file can be plotted. If the argument is FIGURE, the parameters of the figure can be changed. If the argument is the name of Gnuplot file, the file must be edited.

Args	1	FIGURE | CHAR
		The figure object or the name of the file containg Gnuplot commans, @@figure

If the argument is the file name with .jfig extension, and you edit the file, its is safe to change the name, becase if an figure with teh same name is generated, the edited fike is autimatically deleted. If the file refers other files, it is wise to rename also these files and change the names in the beginning of the .jfig file. You may wish to use show also if you cnange the window size

[showex]Example of show()

```
fi=draw(func->sqrt2(x),x->x,xrange->(-50,50),continue->fcont)
show(fi,xrange->(-60,60),xlabel->'NEWX',ylabel->'NEWY')
show(fi,axes->10)
show(fi,axes->01)
show(fi(axes->00)
Window='400,800'
show(fi)
Window='700,700'
fi=drawline(1,10,3,1,color->Red,continue->fcont)
show(fi,xrange->(1.1,11)) !the line is not visible
dat=data(read->(x,y),in->)
1,4
2,6
3,2
5,1
/
stat()
fi=plotyx(y,x,continue->fcont) ! Gnuplot hides points at border
show(fi,xrange->(0,6),yrange->(0,7))
```

24.7 Plot 3d-figure: plot3d()

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.

```
[plot3dex]plot3d() example see p. 328 in Mehtatalo  Lappi 2020
 mat=matrix(1000000,3)
mat2=matrix(1000000,3)
tr=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
/
```

```
call(tr)
fi=plot3d(mat,sorted->)
fi=plot3d(mat2,sorted->)
```

25 Splines, stem splines, and volume functions

There are several spline functions.

25.1 tautspline()

tautspline(x1,â¦,xn,y1,â¦,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,â¦,xn the x values// d1,â¦,dn the y values.// There must be at least 3 knot point, i.e. 6 arguments.// Options:// par gives the 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 xâs 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.

25.2 stemspline(): splines for stems

TO bE REPORTED LATER, see old manual

25.3 stempolar(): polar coordinates

TO bE REPORTED LATER, see old manual

25.4 laasvol(): svolume eqaitions of Laasasenaho

TO bE REPORTED LATER, see old manual

25.5 laaspoly(): polynomila stem curves of Laasase-naho

TO bE REPORTED LATER, see old manual

25.6 integrate(): integrate

TO bE REPORTED LATER, see old manual

26 Bit functions

bit functions help to store large amound of binary variables in small space. These functions are used in domain calcualtions

26.1 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 set-value() function. The values of bitmatrix elements can be accessed with value() function. Also ordinary real variable can be used to store bits. See bit functions.

26.2 setbits(): setting bits on

To be reported alter

26.3 clearbits(): clearing bits

To be reported alter

26.4 getbit() : get bit value

To be reported later, see old manual

26.5 getbitch() : get bit value

To be reported later, see old manual

26.6 bitmatrix() : define a matrix for bits

To be reported later, see old manual

26.7 setvalue() : set value for a bitmatrix

To be reported later, see old manual

26.8 closures() :convex closures

To be desrribed later, see old manual

27 Misc. functions

There are some functions which do not belong to previous classes.

27.1 properties(): defining properties of some subjects.

This function has been used to define properties of factories. It will replaced with othe means in later versions.

27.2 cpu() gives the elapsed cpu time

```
[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;
```

27.3 seconds() gives the elapsed clock time

```
[secondsex]Example of elapsed time
 cpu0=cpu()
sec0=seconds()
a=matrix(100000)
a=ran() !uniform
mean(a),sd(a),min(a),max(a);
```

```
cpu1=cpu()
sec1=seconds()
elapsed=cpu1-cpu0;
selapsed=sec1-sec0;
```

27.4 Object names

Object names start with letter or with '$'. Object names can contain any of symbols '#.%Â§\_'. **J** is using '%' to name objects related to some other objects. E.g. function stat(x1,x2,mean->) will store means of variables x1 and x2 into variables x1%mean and x2%mean. Objects with name starting with '$' 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 â[â and â]â, 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 objec. There are many objects intitilized automatically. Some of these are locked so that the users cannot change them. Names of objects having a predefined interpretation start with capital letter. The user can freely use lower or upper case letters. **J** is case sensitive. All objects known at a given point of a **J** session can be listed by command: print(Names)

27.5 Copying object: a=b

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

27.6 Deleting objects: delete_o()

When an object with a given name is created, the name cannot be removed. With delete_0() function one can free all memory allocated for data structures needed by general objects: delete_o(obj1,â¦,objn) After deleting an object, the name refers to a real variable (which is initialized by the delete_o() function into zero). Other objects except matrices can equivalently be

deleted by giving command obj1,â¦,objn = 0 This is because the output objects of any functions are first deleted before defining them anew. Usually an object is automatically deleted if the object name is as an output object for other functions. One can see how much memory each object is using print(Names).mes Deleting a compound object deletes also such subobjects which have no meaning when the main object is deleted. But e.g. if a data object is deleted then the as-sociated transformation object is not deleted as the transformation can be used independently. Files can be deleted with delete_f(file). See IO-functions for details. If the user has defined own new compound objects in the open source **J** software she/he needs to define the associated delete function.

27.7 Object types

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

27.7.1 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->(x1â¦x4)) ! x1, x2 ,x3, x4 are variables in the data set, and get their values when doing operations for the data.

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.

In this manual 'variable' refers to a **J** object whose type is REAL.

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

27.7.3 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

27.8 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 REAL A special variable used in data() function when duplicating observations
LastaData List A list object referring to the last data set made, used as default data set.
$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
$
textcolortealtabto25mm REAL
Object name used to indicate console and '*' format in reading and writing
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 REAL Variable used to indicate how much output is printed. Not properly used.
$Debug REAL Variable used to put debugging mode on.
Accepted REAL The number of accepted observations in functions using data sets.
Arg REAL The default argument name when using transformation object as a function.

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.

27.8.1 Transformation object

A transformation object groups several operation commands together so that they can be used for different purposes by **J** functions and **J** objects. A transformation object contains the interpreted transformations. For more details see **J** 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 **J** functions use also implicitly transformations object $Cursor2$.

27.9 Code options

There are some special options which do not refer to object names or values. Some options define a small one-statement transformations 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. [codeoptex]Codeoptions

```
 dat=data(read->(D,H),in->)
3,2
2,4
4,1
/
stat(D,H,filter->(H.gt.D)) ! only those observations are accepted which pass the filter
fi=draw(func->(sin($x)+1),x->$x,xrange->(0,10),color->Red,ylabel->'sin(x)+1',xlabel->'x',width->2)
```

Interpreted transformations in a transformation object can be automatically executed by other **J** functions or they can be executed explicitly using call() function.

Arg	1	TRANS
	The transformation object executed.	

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. [recursion]Recursion produces system crash.

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

27.10 J transformations

Most operation commands affecting **J** 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 **J** objects. General **J** 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 **J** 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 **J** 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.

27.11 Generating a transformation object trans()

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. **J** 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\ob[. 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 objec.

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.

Options input->, local->, matrix->, arg->, result->, source-> of previous versions are obsolte. 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. The value of Printresult can be changed in other parts of the transformation, or in other transforamations called or during execution of pause().

Output variables in maketrans-> transformations whose name start with $ are not put into the new data object. [transex]Demonstrates also error handling

```
 tr=trans()
$x3=x1+3
x2=2/$x3;
/
tr%input,tr%output,tr%source;
x1=8
call(tr)
tr2=trans(x1,x2)
```

```
$x3=x1+3
x2=2/$x3;
x3=x1+x2+$x3;
/
tr2%input,tr2%output,tr2%source;
call(tr2)
tr23;  !x3 is now local
tr3=trans()
x1=-3
call(tr) !this is causing division by zero
/
Continue=1  ! continue after error
call(tr3)

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


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

<x1=8


<call(tr)
x2=0.18181818

<tr2=trans(x1,x2)
<$x3=x1+3
<x2=2/$x3;
```

```
<x3=x1+x2+$x3;
</
<tr2%input,tr2%output,tr2%source;
tr2%input is list with            1  elements:
x1
tr2%output is list with            1  elements:
x2
tr2%source is text object:
1 $x3=x1+3
2 x2=2/$x3;
3 x3=x1+x2+$x3;
4 /
///end of text object

<call(tr2)
x2=0.18181818

tr2\x3=19.1818181

<tr2\x3;
tr2\x3=19.1818181

<tr3=trans()
<x1=-3
<call(tr)
</
<Continue=1
<call(tr3)
*division by zero
*****error on row            2  in tr%source
x2=2/$x3;
recursion level set to    3.0000000000000000

*****error on row            2  in tr3%source
call(tr)
recursion level set to    2.0000000000000000

*err* transformation set=$Cursor$
recursion level set to    1.0000000000000000
****cleaned input
call(tr3)
```

```
*Continue even if error has occured
<;return
```

27.12 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.
[transfunc]Transformation as a function
```
 delete_o(a,c)
tra=trans()
a=8;
c=2;
/
trb=trans()
a=5;
c=1;
/
c=2
a=c+trb(a)+c+tra(a);
```

27.13 Transformation control structures

Within **J** transformations, there can be similar controls structures as in the input programming. The difference is that these will remain as part of the transformation set. Only the 'if()output=â¦'structure is allowed at the command level, other are possible only within a transformations set.

27.13.1 if()

if()j_statementâ¦
The one line if-statement.

27.13.2 if() elseif() else endif

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

```
    if()then
â¦.
elseif()then
â¦
else
â¦.
endif
```

27.14 Loops and control strucures

This section describes nonstadard functions.

27.15 return

Return from a transformation set to the transformation object where call() function was, or to the include file with call(), or to the sit> promt, is call() was at sit>. [retex]example of return and goto ()

```
 tr=trans()
ad1:r=ran();
if(r.lt.0.2)return
goto(ad1)
/
```

call(tr) return is automatically put to the end of transformation object. Labels in a transformation object are without ';' as the addresses in an include file start with ';'.

27.16 errexit()

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

```
    [errexitex]itex
 tr=trans()
if(a.eq.0)errexit('illegal value ',a)
s=3/a; ! division with zero is teste automatically
/
a=3.7
```

```
call(tr)
tr(s); !tr can also be used as a function
a=0
Continue=1 !Do not stop in thsi seflmade error
call(tr)
Continue=0
```

27.17 goto()

Control can be transfered to a line in a transformation set with goto(). [gotoex]

```
 tr=trans()
i=0
if(i.eq.0)goto(koe)
'here';
koe:ch='here2';
/
call(tr)
ch;
```

It is not allowed to jump in to a loop or into if -then structure. This is checked already in in the interpreter. It is not yet possible to continue within an include file using Continue=1. It is not recommended to use goto() according to modern computation practices. However, it was easier to implement cycle and exitdo with goto(), especially if cycle or exitdo does not apply hte innermost do-loop.

27.18 Numeric operations

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

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

J - 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 addtional opertations.

J *. elementwise product (Hadamard product)

J /. elementwise division

The matrix operations are explained in section ?. Their operation rules extent the standard rules.

27.19 Logical and relational expressions

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

J .eq. == equal to

J .ne. <> not equal

J .gt. > greater than

J .ge. >= greater or equal

J .lt. < less than

J .le. <= less or equal

J .not. ~ negation

J .and. conjunction

J .or. disjunction

J .eqv. equivalent.

J .neqv. not equivalent

The relational and logical expressions produce value 1 for True and value 0 for False. Note: Testing equivalence can be done also using 'equal to' and 'not equal', as the same truth value is expressed with the same numeric value. when the truth value of an expression is tested with if(), then all nonzero real values means that the expression is true.

27.20 Arithmetic functions

The arithmetic functions return single REAL value or a MATRIX. sqrt(), sqrt2(), exp(), log(), log10(), abs()

- **J** 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.
- **J** 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.
- **J** exp(x) e to power x. If x>88, then **J** produces error in order to avoid system crash.
- **J** log(x) natural logarithm
- **J** log10(x) base 10 logarithm
- **J** abs(x) absolute value

Real to integer conversion

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

The following rules apply both for min() and max(). The rules are presented here only for min().

- **J** min(x1,...,xn): $n>1$ and all arguments are REAL, Result is REAL.
- **J** min(A): If A is matrix then the result result is row vector containg minumum of each column. If A is a column vector, Result is REAL.
- **J** min(A,any->) If A is matrix, then Result minimum over all elements.

- **J** min(x,A) If x is REAL and A is matrix, then Result is matrix with the same dimensions as A and Result(i,j)=max(x,A(i,j)). The order of arguments does not matter.
- **J** min(A,B), A and B compatible matrices. Result is amatrix with the same dimesnions containg elementwise minimums.

27.20.1 Text objects

Currently there are two text object types, the old text object TEXT and the new TXT. The TEXT object stores text in a long vector of single characters. The TXT object stores text in lines of 132 characters. The TEXT objects save memory but are not so easy to modify and use. Several **J** functions create associated text objects. **J** functions text() and txt() can be used to create text objects directly. All the names of **J** objects are also stored in a TEXT object called Names. The number of lines in a text objects can be obtained with nrows() function and the total number of characters can be obtained with len() function.

27.21 The main components of J program

The structure of **J** program

- **J** Input programming which generates text input for the interpreter (subroutine j_getinput). **J** commands are obtained:
  - **J** from sit> prompt
  - **J** form possibly nested include files
- **J** Interpreter which generates from text lines integer vectors containing function indices, option indices and object indices (subroutine j_interpret).
- **J** Function driver which executes the code in the interpreted integer vector (subroutine dotrans). The the function driver is using:
  - **J** **J** functions which operate on arguments which are determined either as formal arguments or via options.
  - **J** **J** objects
  - **J** Global variables and matrices
  - **J** Utility subroutines

A user of **J** needs only input programming and **J** functions, but understanding of the other properties may help to understand waht is going on in a **J** session.

28 Future development

I think that the current version of **J** provides many possibilities for future developments. For instance:

- **J** The 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 muc more efficient ways to develop simulators. Examples will be provided shortly.
- **J** Using the possibility to compute derivatives using the analytic derivates makes it quite straightforward to make it possible to have a nonlinear objective function
- **J** It would be quite easy to develop **J** so that integer solution is produced with respect to the schedule weight.
- **J** It would be interesting to see how **J** can put to work with Heureka.
- **J** The possiblity to run R scripts from **J** and **J** scripts from R provide new possiblities.
- **J** **J** can now be used as an interface to Gnuplot. Google search show how many possiblities Gnuplot provides. It is quite straightforward to implement these grpahs if it is not currently possible.
- **J** The possibility to generate random numbers from any discrete or continuous distribution provide new possibilities to stdy the effects of random errors in the optimization.
- **J** The new tools for analyzing grouped data are usefulf 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 multimarket timber supply model. Usersâ 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 â 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 178â-184",

DOI = http://dx.doi.org/10.1080/02827581.2014.886714