

This manual is in doc and pdf form. The pdf version is easier to read and navigate, but *embedded files cannot be read in it*.
All embedded files are found in the zipfiles' Rapp\Dok.

Rapp - programming language manual

2026-08-01 Stig Rosenlund

Table of contents

General info about the programming language Rapp	2
Reasons for using Rapp	2
Location of program and how to write and run Rapp code .	6
Examples of running Rapp	8
The language's syntax and components	8
Limitations, summary	9
Proc Acctra	10
Proc Alarm	10
Proc Bich	10
Proc Calend	17
Proc Chaall	17
Proc Cmd	18
Proc Compar	18
Proc Coofil.....	19
Proc Copy	21
Proc Copyfo	22
Proc Data	23
Proc Ddist	24
Proc Diskcm	25
Proc Durber	26
Proc Excel	27
Proc Figadj	30
Proc Filsta	30
Proc Ftp	30
Proc Gpdml	30
Proc Graf	31
Proc Grafb.....	37
Proc Hipseu	38
Proc Init	39
Proc Linreg.....	41
Proc Livr	42
Proc Map	44
Proc Match	51
Proc Matrix	53
Proc Mbasic	54
Proc Ovelim	75
Proc Percen	76
Proc Print	77
Proc Pseu	77
Proc Reschl	77
Proc Restea	86
Proc Restri	86
Proc Rskilj	89
Proc Sample	89
Proc Sas	90
Proc Sasin	90
Proc Sasut	90
Proc Sort	91
Proc Split	91
Proc Sum	92
Proc Svg2co	92
Proc Taran (Proc Jung)	94
Proc Taran multiclass: OJ2010, SR 2018	107
Norming	112
Proc Xlmerg	112
Swedish to English glossary for reserved words	112
Examples of Rapp-programs (not multiclass analysis)	113
Examples of graphs in PDF	123
Quick guide - short manual by example	127
Appendices - confidence intervals, multiclass analysis .	132

General info about the programming language Rapp

Web site for Rapp is <http://www.stigrosenlund.se/rapp.htm>, also with the C# application Rappmenus. The programs are downloaded as Rapp.Exe and Rappmenus.Exe. Make a shortcut to Rappmenus on the desktop or start menu. (But a shortcut to Rapp is of no use.) Input and output to Rapp.Exe are simple text files. Use of the graphics embedded in Rapp needs MiKTeX or Adobe Acrobat to translate PostScript to PDF. Rappmenus has graphical user interfaces for several Rapp procedures.

Rapp is written in C. But no special software for C is needed, because Rapp.Exe is a compiled and linked C program. Rapp.Exe is an interpreter for the programming language Rapp, ie it reads a Rapp-program and interprets the instructions in it and uses C code to solve systems of equations and make PDF and Xml files, etc. Programming languages are usually built in C. Eg SAS is written in C. It is the fastest way. **Rapp would be slower in C++.**



Here is an explanation.  WhyC.jpg. It is found also on the Rapp web page.

In Appendix 1 are e. g. confidence intervals described mathematically and Appendix 2 describes my multiclass method. The main purpose is tariff (price rating) analysis, but there are also procedures for maps and claim reserve calculation, random samples, matching, data mangling, etc.

I will denote the book "Non-Life Insurance Pricing with Generalized Linear Models" by Esbjörn Ohlsson and Björn Johansson (2010), Springer, Berlin by OJ2010.

Reasons for using Rapp

Proc Taran was the first proc constructed. By default it makes tariff analysis by MMT (Method of Marginal Totals), but the methods Standard GLM and Tweedie are also available. Factor estimates are made for claim frequency and risk premium. For mean claim, no factor estimates are in the listfile, but they are given in a semicolon-separated textfile and displayed in Proc Graf with parameter m. What is then given are factor estimates and confidence intervals derived from the frequency and risk premium via (mean claim factor) = (risk premium factor)/(claim frequency factor) and an essentially similar calculation of confidence intervals. Mean claim factors are of interest to provide background information as to why the risk premium is as it is. Given the fact that MMT solutions, for at least four arguments, mostly are the best for both frequency and risk premium, a separate analysis of frequency and mean claim is best done by (mean claim factor) = (risk premium factor)/(claim frequency factor) and its confidence intervals as a basis. See <http://www.tandfonline.com/doi/abs/10.1080/03461238.2012.760885> or Appendix 1.

Built-in hypothesis testing options are not available in Rapp. OJ2010 contains instructions on how to perform e. g. F-tests with the facilities in Sas Proc Genmod. Tests in SAS for mean claim factors are completely dependent on both the gamma distribution and the homoscedasticity assumption for the claim amounts in the standard-GLM. Since the assumption of gamma distribution is never even remotely true in reality, it would be wrong to build such facilities in Rapp. (The misguidedness of using the specific gamma distribution assumption in standard-GLM is also shown by the research conducted on the LF for different ϕ -estimation techniques, resulting in the dismissal of all gammalikelihood-based estimates, with the conclusion that Pearson's ϕ -estimation of non-aggregated claim data is the only acceptable one.) Hypothesis tests for the argument classes' risk premium factors are best done by studying graphs with confidence intervals.

If certain levels (= classes) miss claims, or even insurances, the equation solutions go through anyway, in contrast to SAS, with 0 in the estimated factors for the levels.

In non-mathematical respects such as

- ◆ Ease of use
- ◆ The speed with which the results reached
- ◆ Output information richness
- ◆ The impact of the graphic images obtained

the programming language Rapp has clear benefits, which is shown below. Especially the latter aspect is usually considered to be very important.

It is easy to write and run a Rapp-program. Selection and grouping of frequently occurring types can be done in Proc Taran and thus reduce the need to create new input

for each new angle of analysis. You can combine multiple variables into one, such as sex and age. For example, if Sex has values 1=Male / 2=Female / 3=Company, and Age values 0-120, then the variable Sexage is calculated and used as argument with

```
dvar(Sexage = 1000 * Sex + Age)
arg(Sexage) niv( (1,1000-1019 'M -19') (2,2000-2019 'K -19') ... (9,3000-3999 'Company')
);
```

Rapp interacts easily with SAS. A SAS table for Proc Genmod can, with a few simple statements inside Rapp, be exported to a textfile for Rapp. Output from Rapp can be easily transferred to a SAS or Excel table for further processing for tariff simulation. Rapp is also considerably more flexible than SAS concerning the structure of the input.

By optimized calculation algorithms Proc Taran runs go through much faster than SAS and is sometimes the only way to get to a result in reasonable time. The difference in speed is greatest with many free parameters. There, SAS can use weeks or years, while Rapp goes through in minutes using the classical method for numerical solution of equations. But even in normal tariff analysis the difference is significant. A test of a SAS table with approximately 4 million lines, about 1700 million combinations, 15 arguments and 70 free parameters was made. The Newton-Raphson method for the numerical solution of equations is here better than the classical method. The SAS-run, with "Proc Genmod / Dist=Poisson Link=log", was optimized by first using Proc Summary. Thereafter, the factor solution was performed on both claim frequency and risk premium like in Rapp. SAS and Rapp was running in Windows on a local PC with 1 gigabyte of RAM and processor speed of 3.2 GHz. Outcome:

SAS: 60 minutes.

Rapp: 3.7 minutes, of which 2.2 minutes to export the table to a textfile and 1.5 minutes to solve the equations from that textfile.

Informative text in text blocks, and in graphs are produced easily. Several key ratios and univariate (marginal) accounting concepts are produced at the same time as factor and variance estimates. The easily produced graphic images are extremely powerful.

Input is one or more textfiles with fields that are separated by a space or other delimiter such as semicolon or tab character. No special computerfile formats like SAS tables are designed for Rapp, because it would make data more closed and difficult to port between platforms. For visual inspection of data one should read the textfiles into SAS, Access or Excel. Reading of the numeric fields display the form of textfiles is slower than reading binary stored fields such as in a SAS table, but still fast enough to be acceptable in this context. Even with millions of input lines there is only a few seconds delay. In the internal processing of Rapp are used, however, files stored with binary fields, in sorting, aggregation and multiple input of data during the iterations of the equation solution.

Output is a listfile in text format with factor estimates for claim frequency and risk premium, uncertainty rates, the marginal risk premium, claim percent of premium, and other marginal totals and ratios. In addition is made a textfile with the factor estimates and sums in semicolon-separated fields, which can easily be transferred to a table in SAS, Access or Excel. With the listfile as the only input is produced graphics with point estimates, confidence and portfolio accounts in PDF format. SAS can be run inside Rapp. Arbitrary Exe files, BAT files and other applications that can be called from the Command prompt can be run from within Rapp.

Columns in Swedish in the listfile (which are not self-explanatory)

Antal försår	= duration = number of insurance years
Skkost 1000-tal	= claim cost in thousands of units of currency (eg USD or EUR)
Marg. skfr.	= 1000×(number of claims)/(number of insurance years)
Osäkerhet	= uncertainty of the claim frequency (relative standard error)
Marg. medsk.	= (claim cost)/(number of claims)
Osäkerhet	= standard error för mean claim
Marg. riskpr.	= (claim cost)/(number of insurance years)
Marg. rp/fbel	= (claim cost)/(sum insured under yearly risk)
Osäkerh %	= relative standard error in percent for marginal risk premium
Premint 1000-tal	= earned premium in thousands of currency units
Medelpremie	= average premium = (earned premium)/(number of insurance years)
Skadeproc	= 100×(claim cost)/(earned premium)

Faktorer frekvens = claim frequency factor estimate solved with GLM

Faktorer riskprem = risk premium factor estimate solved with GLM

Ffaktospct = relative standard error as a percentage of the frequency factor estimate
Rfaktospct = relative standard error as a percentage of the risk premium factor estimate

Tariff faktor = factors in an existing or recommended multiplicative tariff
Omrfakt = tariff factor multiplied by a constant to make the average Omrfakt 1, weighted by the duration, or sum insured under yearly risk if sum insured is used. Normed duration ndur is used in the same way as sum insured.

Translation of the column headers depending on the parameter lan() in Proc Init:

Swedish	English	German
Antal försår	Number insyears	Summe Versdauer
Antal skador	Number claims	Anzahl Schaden
Skkost 1000-tal	Clcost 1000:s	Schhöhe 1000:n
Marg. skfr.	Marg. clfreq	Marg. Schfrz
Osäkerhet	Uncertainty	Unsicherheit
Marg. medsk.	Marg. meancl	Marg. Mittels
Osäkerhet	Uncertainty	Unsicherheit
Marg. riskpr.	Marg. riskpr.	Marg. Risikpr
Marg. rp/fbel	Marg. rp/suin	Marg. RP/Vsum
Osäkerh %	Uncertainty %	Unsicherheit %
Premint 1000-tal	Premium 1000:s	Präm.ein 1000:n
Medelpremie	Mean prem	Mittelprämie
Medelfbel	Mean suin	Mittelvsum
Medelp/fbel	Average pr/suin	Mittel Pr/Vsum
Skadeproc	Claim perct	Schadprozt
Faktorer frekvens	Factors frequency	Faktoren Frequenz
Faktorer riskprem	Factors riskprem	Faktoren Risikpräm
Ffaktospct	Ffactucpct	FfaktusPzt
Rfaktospct	Rfactucpct	RfaktusPzt
Tariff faktor	Tariff factor	Tarif Faktor
Omrfakt	Recfact	Umrfakt

The semicolon-separated textfile gives units of currency instead of thousands of units of currency. With base factor for each of claim frequency, mean claim, risk premium is meant a constant that the factors for the right argument classes for a policy should be multiplied with to give the factor smoothed estimate of the parameter.

Columns in the semicolon-separated textfile

Argnamn = the argument name
Nivnamn = level's name (class name)
Anr = argument consecutive numbers 1, 2, 3, ...
Ninr = level consecutive numbers 1, 2, 3, ...
Dur = duration = number of insurance years
Fbelndur = sum insured under yearly risk or normed duration
Prem = earned premium in currency units
Antskad = number of claims
Skkost = claim cost in currency units
Ospmu = relative standard error in percent for marginal (univariate) mean claim
Osp = relative standard error as a percentage of the marginal risk premium
Basff = base factor for smoothed claim frequency (equal in all lines)
Basfm = base factor for smoothed mean claim (equal in all lines)
Basfr = base factor for smoothed risk premium (equal in all lines)
Faktf = claim frequency factor estimate
Faktm = mean claim factor estimate
Faktr = risk premium factor estimate
Ospf = relative standard error in percent of the claim frequency factor estimate
Ospm = relative standard error in percent of the mean claim factor estimate
Ospr = relative standard error in percent of the risk premium factor estimate
Tarf = tariff factor

Translated column headings in the textfile depending on parameter lan() in Proc Init:

Swedish	English	German
Argnamn	Argname	Argname
Nivnamn	Classname	Klassename
Anr	Argno	Argno
Ninr	Classno	Klasseno
Dur	Exposure	Versicherungsdauer
Fbelndur	Suminsexposure	Vsumversicherungsdauer

Prem	Premium	Prämie
Antskad	Claimnumber	Schadenanzahl
Skkost	Claimcost	Schadenhöhe
Ospmu	Uncpctmclu	Unspztschade
Osp	Uncpct	Unspzt
Basff	Baseff	Basisff
Basfm	Basefm	Basisfm
Basfr	Basefr	Basisfr
Faktf	Factf	Faktf
Faktm	Factm	Faktm
Faktr	Factr	Faktr
Ospf	Uncpctf	Unspztf
Ospm	Uncpctm	Unspztm
Ospr	Uncpctr	Unspztr
Tarf	Tarf	Tarf

Ffaktospct = Ospf and Rfaktospct = Ospr were calculated from the GLM theory for claim frequencies, as in SAS "Proc Genmod / Dist=Poisson Link=log". These identities apply:

Basfr = Basff*Basfm
Faktr = Faktf*Faktm
Ospr² = Ospf² + Ospm²

Let level (class) j be a base level specified with bas(), see below, or the level of those with claim cost not 0, which has the greatest duration if bas(0) indicated that no level should be a base level. Then level j has the same value for Ospf, Ospm, Ospr as in a univariate account with only one argument. For the risk premium factors for those levels of the respective arguments, Rfaktospct is equal to Osäkerh % (= Uncertainty %).

Other levels are adjusted upwards by means of the diagonal elements of the inverses of the Fisher information matrices for claim frequency and risk premium.

If in Proc Graf is given F2_bas=n1_n2_ ... where n1, n2 are base levels specified with bas(n1), bas(n2) in Proc Taran, then graphs are obtained that give confidence intervals for the frequency factors exactly as GLM theory, i.e. without confidence intervals for the base levels.

If you give R_bas=n1_n2_ ... respectively M_bas=n1_n2_... you get graphs of risk premium respectively mean claim factors without confidence intervals for the base level. This is from a solution I designed for a model where no assumption is placed on the claim amount distribution in a Compound Poisson process. The solution is partly strictly mathematical and partly adhoc. It gives a better approximation to the strict confidence intervals than the adhoc method to transfer "Osäkerh %" above for the marginal risk premiums to the risk premium factors and "Osäkerhet" above to mean claim factors. The portfolio must be very non-multiplicative (the arguments very dependent) for the difference between the partial and the coarse adhoc method to be noticeable. The partly adhoc method is denoted MVW and the coarse adhoc method is denoted 1984 in Appendix 1.

If you enter in Proc Graf R, F, M or R_bas=0_0_ ... F_bas=0_0_ ... M_bas=0_0_..., then intervals are given for all levels. Then preferably bas(0) should have appeared in Proc Taran for easier interpretation of the report and graphs. To give confidence intervals for all levels is important in applications, although such intervals may be difficult to interpret from a mathematical-statistical point of view. When they are shown to laymen, ie product specialists, however, these intervals are intuitively correctly interpreted directly! Confidence interval with positive length for all levels can be described as relative confidence intervals for marginal risk premiums with respect to duration normed for other arguments. That is, we should multiply the lower bound, point estimate and upper limit by a common factor such that the resulting estimate shows this marginal risk premium. Another way to interpret the confidence intervals with positive length for all levels is that they are literally approximately correct if all factors have been multiplied by a factor such that the average factor over the portfolio is 1.00, provided no argument dominates.

With parameter T to Proc Graf Omrfakt (Recfact) is illustrated by the black bar. To the right of it, like a flag on a flagpole, is displayed a confidence interval for the argument level as a bar hanging in the air. The midpoint is the risk premium factor, the whole bar is the 90%-interval and the inner bar with a denser pattern is the 60%-interval. If in the underlying reality the risk premium factor is equal to the tariff factor, then on the average in nine cases out of ten the top of the flagpole will be within the flag's height. But in one case out of ten the top will be either below or



above the flag. Alternatively Omrfakt (Recfact) is given as a broken line with parameter `tarline[C[F]]|[L]|[S[F]]`.

Location of program and how to write and run Rapp code

An all-in-one package is available at <http://www.stigrosenlund.se/rappzip.htm> as a zip file. It contains the essentials and example Rapp programs.

Rapp is written in MSVC = Microsoft Visual C++, C-part only. There are two Rapp exe files for different environments in <http://www.stigrosenlund.se/rappexes.htm> and in the zip file, namely Rapp32Vc2017.Exe for 32 bit and Rapp.Exe for 64 bit. In a 64-bit system Rapp.Exe will run faster than Rapp32Vc2017.Exe. Rapp32Vc2017.Exe will work in Windows XP, although I have compiled and linked it in Windows 7 Professional 64-bit. Do not use Rapp32Vc2017.Exe in a 64 bit system, although it is possible.

Rename the chosen exe file after download to Rapp.Exe and put it eg in C:\Rapp\Pgm.

In addition you find on the site a special zip file Rappmultiprecision.zip with variants of Rapp.Exe for different levels of multiple precision in Proc Mbasic. The precision below denotes the maximum number of correct digits in the mantissa of the mouble data type of Proc Mbasic. These exe files work as Rapp.Exe with the exception of precision. They are only for 64-bit. Rapp.Exe with maxprecision 306 is not included in the special zip file.

Exe file	Maximum precision	Storage size in bytes of a mouble
Rapp0027.Exe	27	32
Rapp0054.Exe	54	40
Rapp0108.Exe	108	64
Rapp1008.Exe	1008	464
Rapp1800.Exe	1800	816
Rapp5004.Exe	5004	2240
Rapp9000.Exe	9000	4016
Rapp.Exe	306	152

To run SAS from within Rapp of course requires that SAS is installed. Graphic images in PDF requires MiKTeX with the PS2PDF.BAT program, or Acrobat Distiller (comes with the larger Adobe Acrobat, which you pay for). Rapp needs no other special programs. You can bring home the program on a USB stick and run it at home, and make graphs if you have MiKTeX alternatively Acrobat Distiller. MiKTeX is a free program that can be easily downloaded from the Internet - search for MiKTeX on Google, or use the link in <http://www.stigrosenlund.se/AllInOne.htm>.

Expected technical lifetime of Rapp.Exe (personal independence and invulnerability)

Rapp.Exe can be expected to last at least 25 years, given that the way described above to re-compile and link the program can be remembered. If the source code cannot be compiled and linked, then maybe sustainability is only 15 years.

Rapp.Exe is very simple in Windows technique though mathematically advanced. Rapp.Exe is not "installed", but simply copied to a disk that the computer has access to. No advanced windows with combo boxes, etc., no environmental variables set permanently, no file types that are reserved (the user can make that herself). There is no reason to believe that future Windows versions will be backward incompatible with the existing parts of MSVC, which are used for Rapp.Exe. MSVC is Microsoft's own product for C/C++, which is a guarantee of continued operation.

The graphics part of Rapp.Exe needs some software to convert a PS file to PDF. Since Adobe Systems has invented both PostScript and PDF such programs will exist in the future, even if the free program PS2PDF.BAT stops working. See under Proc Init for initial settings suitable for PS2PDF.BAT and Acrobat Distiller, respectively.

There is no risk for disturbances in the PC from running Rapp.Exe. Long ago, there were risks with exe files from C-programs of the kind that we are talking about, but no longer. Both the operating system and the C-compiler/linker have built-in safety guards that prevent such disturbances. If for example Rapp.Exe would try to write data in forbidden memory, the only thing that happens is that the program aborts with a message that a forbidden operation has been attempted. So it is safe to run Rapp.Exe. - Another thing is that one should be careful with exe files of unknown origin, since they can contain viruses.

Editing Rapp programs and how to run Rapp.Exe

Parameter to Rapp.Exe is a textfile with Rapp-code. No limit exists for the line length or size of the program. To write Rapp-programs you can use a text editor you're used to, such as the SAS editor or Wordpad. I have recommended SPF Source Editor. However, the creator CTC has folded. I will anyway use SPF as long as it works. I can offer others the text editor accessed with the grey button in Rappmenus. Also my editor Sred, which is found in the Pgm folder of the Rapp zip file.

The SAS source editor can also be used. Rapp is adapted to that editor by treating tab characters as blanks.

In Rappmenus, choice "Text editor for Mbasic and Rapp.", there is an editor with coloring of Mbasic (part of Rapp, see Proc Mbasic later in this manual) and Rapp special syntax words. There is also syntax-sensitive coloring for C and for some of the keywords of SAS (those I used at Länsförsäkringar). I have emulated some of the functionality of SPF. An Mbasic- or general Rapp-program is run from the editor with a button. There are some useful template parmfiles to get you started in the folder Mbparm of the zip files Rapp.zip and Rapp.zipx. Place this folder on your computer as C:\Rapp\Mbparm. There are special features which facilitate debugging your Mbasic-program. Find, replace and sort are extensively implemented.

The editor is not as good as SPF, but I think it is better than Wordpad and many other editors also for general files.

Running Rapp - short description

From the Command Prompt

Prerequisite is that you have entered C:\Rapp\Rpp by writing Br for the execution of Br.Bat. (Search Br.Bat in this document.)

Rapp Rapp-programname

where the extent can be omitted if it is .Rpp, and the full path can be omitted if the Rapp program resides in C:\Rapp\Rpp. For example

Rapp Taran-demo

From Windows Explorer

Open the Rapp program with Rapp

This can be done in two ways. Either by double-clicking the Rapp program after having checked "Always open with this program" earlier. Or by right-click / Open with and choosing Rapp.Exe.

Running Rapp - long description

From an arbitrary Command prompt, you can run the command

C:\Rapp\Pgm\Rapp.Exe

Shortened to only Rapp if the folder C:\Rapp\Pgm is in PATH, for example with a BAT file with the statement

PATH=C:\Rapp\Pgm;%PATH%

being executed at the entrance of the Command prompt.

If you write Rapp in the Command Prompt prompt or double click Rapp.Exe in Windows Explorer you are asked for the Rapp-codefile. If you write Rapp followed by Rapp-codefile, or open a Rapp-codefile with Rapp in Windows Explorer, it runs directly. Extent may be excluded if it is .Rpp. With a word that begins with n (for nolog) as a second parameter, the messages on the screen telling how the program progresses in different Procs are suppressed. Error messages appear however if necessary. With only one parameter, or with another word, such as x, as the second parameter this information is not suppressed.

With only the Rapp program as parameter this happens when Rapp has finished its task: If Rapp is run from Windows Explorer by opening a Rapp-codefile, the opened window is retained until Enter, unless the first line's first word is N or n. If N or n the program runs as "Rapp prog n" even from the Explorer, ie the program gives no progress reports and the window closes without Enter at termination. With X or x instead the program is run as "Rapp prog x", ie with progress reports and window closing after termination. See Calc.Rpp further down here for an example where the N increases the programs usefulness.

If Rapp is run from the Command Prompt it depends on how you went into the prompt. Suppose you went by the command Br, after having made Br.Bat with this statement among others

set "FromCmdPrompt=Yes"

(See further down here by searching Br.Bat). Then Rapp exits. If you did not go by Br, then Rapp pauses and waits for the Enter key. If Rapp is run from a bat file, where other commands follow the Rapp command, it is necessary that Rapp exits.

You can change the Rapp-program and run the changed program. The third argument to Rapp shall then be cha or CHA. It is case insensitive. Example:

```
Rapp rprog01 x cha $01 T $02 "Trafik 1995-1999"
Rapp rprog01 nolog cha keep(rprogt.Rpp) $01 T $02 "Trafik 1995-1999"
Rapp "rprog 01" x CHA "keepnorun(rprog avs D.Rpp)" $01 D $02 "Delkasko 1995-1999"
Rapp B{\aa}t cha keep(B{\aa}t01.Rpp) $01 A $02 "B{\aa}tf{\oe}r{\ae}kring 2000-2006"
```

At execution from a BAT file, å - ö come out wrong, so they must be represented like this
 å ä ö Å Ä Ö
 {\aa} {\ae} {\oe} {\AA} {\AE} {\OE}

Program file name and change strings after cha or keep() or keepnorun() must be given within double quotes " if they contain blanks, not within single quotes ', because a single quote is taken as part of a change string. A double quote as part of a change string is given as \". Up to 100,000 changes are admitted after cha, keep(), keepnorun().

If keep() or keepnorun() (case insensitive) is not indicated after CHA, a temporary file is created with name containing the date and time, which is removed after execution. At keepnorun() the modified file is not run. The purpose is to enable checking that the change strings, e. g. \$01 \$02 \$03, are right in the Rapp-program.

You can right-click a file with extent .Rpp in Windows Explorer, select Open With, select C:\Rapp\Pgm\Rapp.Exe, and check that this program continues to be used to open files with extent .Rpp. Then you can double click (highlight + return) on a file with extent .Rpp to run it. Upon such execution the Command window is retained after run, as described above.
End running Rapp - long description

Examples of running Rapp-programs

The Rapp-program is Rapp01.txt. Respond Rapp01.txt on the program's question, or type the command "Rapp Rapp01.txt" in the Command prompt or in a BAT file. Command "Rapp Rapp01.txt nolog" suppresses screen printing.

The Rapp-program is Rapp01.Rpp. Respond Rapp01 on the program's question, or type the command "Rapp Rapp01" or "Rapp Rapp01 n" in the Command prompt or in a BAT file.

Examples of running in a BAT program with changeable content in the Rapp-program

```
@ECHO OFF
REM This program modifies $01 to the right company in a temporary Rapp-program and runs
REM it in a loop in the companies. For example, the Rapp-program can contain
REM rub62 ('Company $01') and urval(company=$01). The parameter x means that the loop
REM goes on, rather than pause at the end of the Rapp-execution.
set rae= C:\Rapp\Pgm\Rapp.Exe
set bol=02 03 04 08 09 10 11 14 15 16 21 24 27 28 29 31 32 33 34 35 37 42 43 50
set rap=F:\B99sti\xxx\Vip01.Rpp
for %%b in (%bol%) do "%rae%" "%rap%" x cha $01 %%b $02 "rub62 'Riskanalys Hem'"
ECHO Press a button!
pause
```

Example of execution in a SAS program

```
x C: & CD \Mappl\Map2 & C:\Rapp\Pgm\Rapp.Exe Rapp01 nolog & exit;
```

The language syntax and components

It is case insensitive, ie large cap or small cap letters do not matter, except in text strings within single or double quotes. An exception is the keyword TEXT alone on a line in Proc Taran. You can have any number of blanks between statements and parameters. Line breaks are irrelevant. Comments are written as in C, REXX and SAS, ie between /* and */. They can be nested in each other, ie as /* ... /* ... /* ... */ ... */ ... */. Also // comments are recognized, ie those that end with the line it is written on.

A caret ^ is continuation symbol last on a line, except in strings and in the block TEXT in Proc Taran. This has a use in Proc Graf to split long words for easier editing, eg:

```
1_pie_color=LGREEN,0.7/0.7/1/rgb,GOLD,1/.4/.4/rgb,CYAN,^
```



```
BROWN, .85/1/.8/hsb, DBLUE, DRED, YELLOW_pattern=L1,X2,L2,SOLID,L1,X2,L2,SO^
LID,L1,X2,L2,SOLID,L1,X2,L2
```

Do not put blanks first on the line following a caret, if you want to preserve the word.

Strings can be split in several lines without a caret. A caret would appear in string.

Include: Rapp-code in another file can be included by writing include or #include or %include first on a line or after N or X, then the file name. Example:

```
N Include C:\Rapp\Rpp\Init.Rpp
```

See under Running Rapp - long description above for the N or X.

Unlimited many levels of include files are allowed. An include file is included in the Rapp-program at the time point where it is needed. Thus it is possible to run a file creation program in a system()-statement or a Proc Data procedure and take in the created file with include in a later step. See Calc.Rpp below in this manual. Two file names can be given with | in between. If Rapp cannot find the first file, it reads the other one. After the included file only a comment can stand.

Exit: Rapp is terminated. Handy if you create Rapp programs dynamically, which shall be closed at some point under certain conditions. This is the case sometimes when you run from Rappmenus.Exe

Say: A text is written on the screen with Say (Say, say) outside procedures and system()-statements. What is on the same line after Say is written, but not the following lines. For example, the information that a particular procedure has finished. Proc Alarm before or after Say gives audio info also. A comma last after Say makes the text written by the next Say-statement appear on the same line. Say is case independent.

System(): Another program can be run with a statement system() outside procedures and say-sattements. Case independent. The command within system() must have as many right parentheses as left ones. Eg

```
system(C:\xx\aaa.bat), SYSTEM(C:\xx\aaa.bat), System(C:\xx\aaa.bat).
```

The remainder of a Rapp is a collection of procedures according to the table of contents.

First is given Proc (or PROC or proc or pRoC, etc., ie optional uppercase or lowercase) followed by the proc name. The end of the Proc is given with Endproc.

Example of a work process in a Rapp-program:

```
Init   : Set parameters needed for the rest of the processing.
Cmd    : Perform a set of commands in the operating system.
Sasut  : Export SAS tables to textfiles that are input in Proc Data.
Data   : Mangle textfiles - create new indexes, etc.
Durber : Calculate duration.
Taran  : Make a tariff analysis report.
Sasin  : Import an outputfile with estimates from Proc Taran to a SAS table.
Graf   : Makes graphic images in PDF format from an outputfile from Proc Taran.
```

Limitations, summary

Filename: For externally named files å, ä, ö, Å, Ä, Ö is OK, but files created in Rapp-programs should not have these characters in the name.

nummetod(K), Classical Approach: A maximum of 20 arguments and 2147483646 argument combinations.

nummetod(N), Newton-Raphson: Up to 99 arguments.

Heading lengths: up to 30 for rub30, 62 for rub62 and 110 for rub110.

Level (class) names in arguments: 10 characters.

Number of variables in the inputfiles: 600, including derived variables in dvar().

Length variable name: 30 characters.

Number of levels (classes) for arguments: 65535.

Length alphanumeric variable: 10 for arguments, otherwise according to `alphasize()`.

Interval numeric argument, variant 1 and 2: [1,65535].

Interval numeric argument, variant 3 with `niv(...)`: [-2100000000,2100000000].

Interval numeric argument, variant 4 without parameter `niv()`: [-999999999,2147483647].

Number of selection conditions per set of inputfiles: 250.

Number of enumerated values for selection after selection condition = or ^=: no limit.

Number of ID fields for multiclass analysis with OJ2010:s method: 99.

Description of the procs in alphabetical order:

Proc Acctra

Example:

```
Proc Acctra infil(Acctable1.Txt) utfil(Acctable2.Txt) recfil(Acctable2.Rpp) Endproc
```

Transforms a textfile exported from Access. The export must be a textfile with tab separated fields, a first line of field names, and strings delimited by double quotes ". The proc transforms alphanumeric dates on the form YYYY-M-D to numerical ones on the form YYYYMMDD. First character in each field name is assumed to specify the datatype: 'c' and 'm' for character, 'd' for dates as above. Other initial characters are assumed to mean numeric field. Up to 150 alphanumeric characters per field are transmitted to the output-file `utfil()`. In `recfil()` a record description in Rapp syntax is given, where numeric fields that need be floating-type are given type R. The outputfile is a valid input file for Proc Taran et al, with parameters `dml(9)` and `firstobs(2)`.

Proc Alarm

Example 1:

```
Proc Alarm Endproc
```

Gives a signal immediately.

Example 2:

```
Proc Alarm time(1500) Endproc
```

Gives a signal at 15:00. The time can be given as `hhmm`, `hh:mm` or `hh.mm`.

Useful between other procs to show how a long calculation progresses. Or - as a somewhat extraneous application for Rapp - to be reminded that it is lunchtime.

The way sound is made changes for each new computer and Windows version. Previously Rapp could vary the type of signal and its length, but now it can only make an instantaneous sound.

Proc Bich

Example:

```
Include C:\Rapp\Rpp\Init.Rpp
```

```
Proc Bich Timeconv(Y) Nrep(2000) /* Nsvans(4) Nmseinf(6) */ Utfil(Bootst1.Txt)
  Infil-boo(C:\S\Boot1.Txt) Firstclaimtime-boo(19940101) Lastclaimtime-boo(19961231)
  RDC Openuse(2) maxW(4) quantlimit(999) quantno(500)
  Colnpaybel-boo(2) Colnpaytime-boo(3) Colnsettime-boo(4)
  Colnreptime-boo(5) Colncltime-boo(6)
  Urval-boo(1_2_1_1_44_9_0_49999)
  Pricefile-boo(Kpi.Txt) Basepricetime-boo(2007) Colnpricetime-boo(1) Colnprice-boo(14)
  Infil-akt(C:\S\Boot1.Txt) Firstclaimtime-akt(19970101) Lastclaimtime-akt(20071231)
  Colnpaybel-akt(2) Colnpaytime-akt(3) Colnsettime-akt(4)
  Colnreptime-akt(5) Colncltime-akt(6)
  Urval-akt(1_2_1_1_44_9_0_49999)
  Pricefile-akt(Kpi.Txt) Basepricetime-akt(2007) Colnpricetime-akt(1) Colnprice-akt(14)
  Inx-method(1) graf-akt
ENDPROC
```

```
Proc Excel listfil(Bootst1.txt) xlmfil(C:\Rapp\Ut\Xlm\Bootst1.xml) cb visa endproc
Proc Graf listfil(Bootst1.txt) pdfil(C:\Rapp\Ut\Pdf\Bootst1.Pdf)
  genhead('Prediction intervals by BICH in boxplots') sw boxplot nosubtitle
  pos[ 0 A_vbar_pattern=solid_color=green
    47-(1.96*71) $ 47+(1.96*71) 95$%$with$normal$approx$(+1.96*stderr)]
Endproc
```

The ordinary use of this proc will be as part of Rapp programs generated dynamically in Rappmenus / BICH. I hardly expect any other uses.

Bootstrap simulation of reserve development outcomes with development histories of individual claims as input. The reserve is calculated by one of seven methods.

A. Chain ladder and optionally an exponential tail as in Proc Reschl.

B1. Statistical reserve conditioned on report delay, development period and paid to date. The method is chosen with parameter RDC.

B2. As B1 with smoothing by simple linear regression over the Quantno quantiles, separately for each combination of the other variables K and A, as described at Quantlimit().

C. Bornhuetter-Ferguson as described in Wüthrich and Merz (2008), Stochastic Claims Reserving Methods in Insurance, 2.2. Is selected by parameter Bornhuetter-Ferguson. Priorultimate() is parameter for a file with the preliminary estimates of the final claim cost.

D. Benktander as described in the same book, 4.1.1. Also known as Benktander-Hovinen. Is selected by parameter Benktander. Priorultimate() has the same role as for Bornhuetter-Ferguson.

E. Cape-Cod as described in the same book, 4.1.2. Is selected by parameter Cape-Cod. Premium() is parameter for a file with earned premium per claim period.

F. Schnieper as described in the same book, 10.2. Is selected by parameter Schnieper. Schnieperexposure() is parameter for a file with exposure per claim period.

Method B (1 and 2) provides a reserve for any known claim and a reserve per report delay period for not yet reported. The numbers per report delay period for the latter are predicted with chain ladder. Known and predicted numbers are multiplied by their reserves, giving a total reserve per claim period. See Appendix 6 for more details. This method requires that payments are adjusted for inflation to the same level in each file for object and bootstrap claims. It can be done with Pricefile-akt() and Pricefile-boo(). Furthermore, any known claim must be found in the infiles. If no payment has been made on a claim, make a line for it with payment period = report period and payment amount = 0. Parameter RDC is used to apply the method B. Additional parameter Regr for B2.

In experiments, B2 has not given another variability than B1 in reserve estimates, and the bias against the outcome has not changed. If you give the parameter Resfil, reserves for individual claims in that file can however be better smoothed with B2 than B1.

For methods Benktander, Bornhuetter-Ferguson and Cape-Cod, intermediate lower triangle cumulated values, before the last development period, are raised by the same percentage as the ultimate claim cost is raised from chainladder.

Which method gives the best results varies with the situation.

Claims are assumed to exist as textfiles with one line per payment amount or change amount of known claim cost. A line must contain the words

claim-ID, claim-time, pay-time, payment/change-amount

blank-, semicolon- or tab-separated. There must a line with pay-time = report-time. If there is none naturally, make a such a line with Payment/change-amount = 0.

Here claim-ID must be the first word and it must have the same length for all claims in an infile. A field for report-time is also recommended. This is indicated by parameters Colnreptime-akt() and Colnreptime-boo(). For metod B are needed also settlement periods, which are indicated by Colnsettime-akt() and Colnsettime-boo(). An open claim shall have settlement period 0.

Rapp determines if the fields are separated by spaces, semicolons or tabs from the presence of a semi-colon or tab or not. Time can be given as date YYYYMMDD date, month YYYYMM, quarter YYYYQ, year YYYY or non-negative integer index. The same kind of text-files as to Proc Restri, with a first column for the claim-ID, eg 28-12345-1-03.

A number Nrep() simulations are made. Let claim period i for the selected lines from the inputfile Infil-akt() have $N(i)$ claims occurred in the period and reported as of now. For each simulation s ($s = 1, \dots, Nrep()$) and i ($i = 1, \dots, n$), where n is determined by the selected lines from the inputfile Infil-akt(), Rapp draws $f(i)N(i)$ claims from the inputfile Infil-boo().

Here $f(i)$ is an IBNR-factor, which can be determined in three ways.

1. By drawing a random number $M(i)$ claims from Infil-boo() until $N(i)$ ones with report period $\leq n - i + 1$ were obtained. Then $f(i) = M(i)/N(i)$. Furthermore, the distribution of $M(i)$ is a bootstrap version of the predictive distribution of the corresponding number that will appear in Infil-akt() after full development. This is the default, provided Colnreptime-boo is given > 0 .
2. By chainladder estimates of $f(i)$ using Infil-akt(). This method is the default if Colnreptime-boo = 0 and Colnreptime-akt > 0 .
3. By input of IBNR-factors in the parameter Ibnrfakt(), such that $f(n)$ is the first word of Ibnrfakt() etc. This method is used when the two other methods cannot be used due to missing reportdates.

The $f(i)N(i)$ claims are to be imagined having occurred in the period i , ie claim-date and payment-/change-date are shifted to the right time for this purpose. On those bootstrapped claims, which may represent the actual claims for object reserve, we can calculate the reserve by method A or B and also the final outcome in terms of what each claim really was worth. Observed MSE (mean square error) of the predicted residual claim cost from the outcome of residual claim cost can be calculated from these Nrep() simulations. The relative MSE can then be transferred to a prediction-MSE-estimate for the current object-chainladder, given a model assumption that we have time homogeneity except for a price adjustment factor. The output report Ufil() provides a collection of such prediction-MSE-estimates.

Optionally simulations can be made in three stages, where in the second and third stage bootstrapped triangles differing from the object triangle by more than a certain number or percentile are thrown away. The parameters Nrep2(), Nrep3(), Rholimit() and Rhopercentile() are set for this purpose. See Appendix 6, section 6.2 for more details.

Two claim files, which can be the same file, shall be specified

Infil-akt()	Claims for object (actual) chainladder to be given reserves per claim period with chain ladder and optional exponential tail.
Infil-boo()	Shall contain a sufficient number of claims that occurred sufficiently long ago to be fully developed. For the bootstrap simulation.

Other parameters

Argname()	Indicates name of segment.
Basepricetime-akt()	Base period for inflation adjustment, to which the claims in Infil-akt() shall be recomputed. The adjustment uses pay-time.
Basepricetime-boo()	Ditto for the bootstrap claims in Infil-boo().
Benktander	That method, of Gunnar Benktander (1919-2018) is used, see above.
Bornhuetter-Ferguson	That method is used, see above.
Cape-Cod	That method is used, see above.
Colncltime-akt()	Column number for claim time in Infil-akt() inclusive of ID. Eg if claim time comes after ID, then give Colncltime-akt(2).
Colncltime-boo()	Ditto for Infil-boo().

Colnpaybel-akt()	Column number for payment in Infil-akt() inclusive of ID.
Colnpaybel-boot()	Dito för Infil-boo().
Colnpaytime-akt()	Column number for payment time in Infil-akt() inclusive of ID.
Colnpaytime-boo()	Dito för Infil-boo().
Colnprice-akt()	Column number (space separated) for price index in Pricefile-akt().
Colnprice-boo()	Ditto for bootstrap in Pricefile-boo().
Colnpricetime-akt()	Column number (space separated) for time period in Pricefile-akt(). The time periods may be specified in any format, eg year SSAA, although time periods of Infil-akt() is in the form eg YYYYMMDD.
Colnpricetime-boo()	Ditto for bootstrap in Pricefile-boo().
Colnreptime-akt()	Column number for report-time in Infil-akt() inclusive of ID.
Colnreptime-boo()	Ditto for Infil-boo().
Colnsettime-akt()	Column number for settlement-time in Infil-akt() inclusive of ID.
Colnsettime-boo()	Ditto for Infil-boo().
DiscountToClaimperiodRate()	See button Inf01 in first screen of Rappmenus.Exe.
Firstclaimtime-akt()	The first claim period to be included in Infil-akt().
Firstclaimtime-boo()	Ditto for bootstrap.
Futureprices	See button Inf01 in first screen of Rappmenus.Exe.
genhead()	General heading within single or double quotes.
graf-akt	Indicates that the object claims part of the report is to be used in Proc Graf. See example above. The parameter nosubtitle in Proc Graf prevents Rapp from taking the first of three header lines as subtitle in the graph.
graf-boo	The same for the bootstrapped claims part of the report.
Ibnrfakt()	Optional sequence of IBNR-factors for number of occurred claims, starting with the latest development period. Eg Ibnrfakt(1.23 1.10 1.05) means that number of claims in the latest period is adjusted with a factor 1.23, the next latest with a factor 1.10 and the one before that with a factor 1.05. Remaining periods are not adjusted.
Infl-method()	0: As described in Appendix 6, ie a common factor for all claim periods for method RDC, otherwise separate factors. Default. 1: No adjustment for inflation. 2: Separate factors per claim period. 3: A common factor for all claim periods.
Inx-method()	Relevant for method RDC and if Pricefile-akt() was given. If Inx-method is not given, then all payments in the reports are indexed by the price file. If given as Inx-method(1), then payments are indexed only internally in Rapp for calculation of reserves. Known payments are shown in the reports with nominal values, and these are the same as in a report without price file.
Lastbetper()	Last payment period index (1,2,3, ...) to be included in both files. If not specified, all available payments of the bootstrap file are included. Suitable to set if there are small payments at the end that do not mean anything but disturb the

model. For example `lastbetper(12)` causes payments only in development periods with indices 1-12 to be included. If not given, the parameter is taken as the development index for a payment on the first claimdate made on the last claimdate. If a tail is wanted in the computations, set `Lastbetper(999)`.

`Lastclaimtime-akt()` Last time period to be included in `Infil-akt()`.

`Lastclaimtime-boo()` Ditto for bootstrap.

`LaTeX()` If given by two numbers in brackets, eg `LaTeX(7 3)`, Rapp creates a file parallel to the output file with L before the extent, with tables for LaTeX. The first number is the first table number and the second is example number. For example `utfil(Motor-200912.Txt)` gives the file `Motor-200912L.Txt`.

`Low()` Lower bound `b_1` by section 4.1 in Appendix 6. Default none.

`MaxW()` Maximal report period counted from the claim period (report period = 1 means reported at claim period), for conditioning with respect to report period. At `maxW(w)` are calculated separate reserves for report period $W = 1, \dots, w-1$, while report periods $W = w, w+1, \dots$ are lumped together. Conditioning refers to the expected remaining payment amount given $J = \text{elapsed duration from the report date}$. The duration J is calculated exactly from the report date, but the distribution given J is assumed independent of W for $W \geq w$. With `maxW(1)` you indicate that W is not important except for conditioning with respect to J . Default is maximal possible w .

`MergeSegm` If given segments are merged before reserving, as opposed to separate reserving per segment as described at the end of section 5 in Appendix 6. The object reserves will then be equal to the ones obtained without giving `Segments-akt()`. The bootstrap reserves will however be different, due to the mechanism of bootstrapping per segment. Normally this parameter would not be appropriate.

`Nmseinf()` Same meaning as `Nmseinfssk()` in `Proc Reschl`.

`Nrep()` Number of simulations of $f(1)N(1), \dots, f(n)N(n)$ claims. The larger `Nrep()` the more certain prediction-MSE-estimates. The standard errors for the prediction-MSE-estimates are given as a basis for assessing appropriate `Nrep()` for the next simulation. If not given or as `Nrep(0)`, then no bootstrapping is done and no boo-parameters are needed.

`Nrep2()` Number of simulations of $f(1)N(1), \dots, f(n)N(n)$ claims in a second stage. Bootstrapped triangles with `rho_n(object triangle,bootstrap triangle) > rho_0` are thrown away. See Appendix 6. If `Rhopercentile()` is given, then this percentile from the first stage is taken as `rho_0`. Otherwise `rho_0` is the number `Rholimit()`, if given. If not given, then no bootstrapped outcomes are thrown away in the second stage.

`Nrep3()` See Appendix 6.

`Nsvans()` Same meaning as `Nsvansssk()` in `Proc Reschl`. Default 0.

`Openuse()` Has effect with parameter `RDC`. Which open claims to use for inference and how they are to be used:

- 0 Closed (finalized) claims only.
- 1 Closed claims and open claims with known settlement period.
- 2 Closed claims and all open claims. For any fixed payment period, the mean payment per settlement period for open claims, for any possible settlement period, is taken to be proportional to mean payment per settlement period for closed claims. This is the default.
- 3 Closed claims and all open claims. For any fixed payment period, the mean payment per settlement period for open claims, for any possible settlement period, is taken to be the

same value.

Percentiles()	Optional. A sequence of at most 11 percentile values which overtake the default values 0.5 1.0 ... 99.0 99.5. Example: Percentiles(0.5 1 25 50 75 80 90 95 97.5 99.55 99.6034)
Premium()	A file of earned premium per segment and claim period for method Cape-Cod. The same structure as the file Schnieperexposure(). If the file is not given or is given as D, then Rapp uses (chainladder computed claim number per segment and claim period)* (total chainladder computed mean claim per segment)
Pricefile-akt()	File with price index for claims in Infil-akt().
Pricefile-boo()	Ditto for bootstrap.
Priorultimate()	A file of preliminary estimates of ultimate claim costs per segment and claim period for the methods Bornhuetter-Ferguson and Benktander. Same structure and default values as for Premium().
Quantlimit()	Maximum for period K calculated from the report period for which conditioning shall be made for claims open at K with respect to hitherto paid period 1, ... , K calculated from W = report period. Hitherto paid is quantile divided. For example, quantlimit(3) indicates that for claims open at K = 3, 4, 5,... the conditioning shall be made for the payment sum over pay periods [W-1]+1, [W-1]+2, [W-1]+3, ie W, W +1, W +2. With quantlimit(999) is indicated that conditioning should be carried out with respect to the last known pay sum so far. With quantlimit(0) is indicated that no conditioning should be done with regard to paid so far.
Quantno()	Number of quantiles for paid so far. For example quantno(20) gives quantiles 0.05, 0.10, ... , 0.95, 1.00 for 20 quantile intervals. Quantno(1) indicates no conditioning with regard to paid so far.
Resfil()	If specified with RDC, this file gets one line per claim in Infil-akt() with claim-ID, claim period (1, 2, ...), report delay w, quantile q, segment and statistical reserve. Also Rapp creates a file with -IB before the extent with IBNR calculation of the numbers of unknown claims, reserve per unknown claim and reserve = product of these two, and RBNS-reserve in the same way. Eg resfil(Res-TPL-200912.Txt) gives IBNR file Res-TPL-200912-IB.Txt. A file with -IBtri before the extent, giving partitions of IBNR and RBNS on development periods, is also created.
RDC	If specified, method B (Reserve by Detailed Conditioning) is used.
Regr	If specified together with RDC, method B2 is used. RDC only gives method B1.
Rholimit()	See under Nrep2().
Rhopercentile()	See under Nrep2().
Schnieper	If given, the method used is the one by R. Schnieper "Separating true IBNR and IBNER claims", Astin Bulletin 1991, vol 21(1), p. 111-127. Colnreptime() is required for this.
Schnieperexposure()	A file with exposures E_i for Schnieper's method. Dates are given as in the claim files. Exposures in the file are added up to the right level. Segment, date and exposure are given blank separated on each line. If this file is not given or given as D, the numbers of claims reported in the first development periods are used as exposures.
Segname()	Synonym for Argname().
Segments-akt()	Position of segmenting variable, see under Segments-boo().
Segments-boo()	If given it shall contain two numbers denoting startposition and

number of positions, separated by underscore. For example segments-boo(12 3). Then Infil-boo() will be divided according to a segmenting variable, with alphanumeric values that in the example starts in position 12 and is 3 characters long in the file. For each claim period, the sampling will be proportional to the numbers of claims in Infil-akt() in the different segments. For example, say that in claim period 5 there are 300 claims belonging to segment 7 in Infil-akt(). Then, to represent this segment 300 claims will be sampled from segment 7 in Infil-boo(). Lines in Infil-boo() with segments not found in Infil-akt() will not be used. Segments will be merged so that segments with no claims in Infil-boo() will be merged with adjoining segments. For each segment, after merging, in Infil-akt() there will thus be at least one claim in the corresponding bootstrap segment, provided there are claims at all in Infil-boo() that belong to a segment in Infil-akt(). If the composition of the claim collection has changed from the time covered in Infil-boo() after selection wrt Urval-boo() and the dates to the time covered in Infil-akt() after selection, then the sampling will be more representative. Segmenting can for example be wrt line of business or percentiles of the first claim periods result or both.

Timeconv() Same as in Proc Restri. Valid alphanumeric values D,H,I,M,T,Q,Y,H. Default Y. Numerical values 1,3,4,6,12 only are supported.

Upp() Upper bound b_1 by section 4.1 in Appendix 6. Default none.

Urval-akt() Same meaning as in Proc Restri. Urval-akt(1_2_1_1_44_9_0_49999) means that the field in position 1 with length 2 (business line) shall be 1 and that the field in position 44 with length 9 (original reserve) shall be maximum 49999.

Urval-boo() Ditto for bootstrap.

Utfil() File for the report.

Uttri() Optional. File for triangles as from Proc Restri, with predictions in the future lower triangles.

W-method() 0: As described in Appendix 6, ie drawing bootstrap claims until the numbers of reported ones are the same as for the object claims.
1: The number of drawn bootstrap claims shall be as for the object claims for each known report period.
2: The number of drawn bootstrap claims shall be as for the object claims for each report period, known or unknown.

Colnreptime-akt(), Colnreptime-boo(), Colnsettime-akt(), Colnsettime-boo(), Nsvans(), Nmseinf(), Urval-akt(), Urval-boo() are optional. Also all parameters for adjustment with price indexes are optional; if none is given there will be no price adjustment.

The first lines of C:\S\ Boot1.Txt (made with Easytrieve in z/OS)

```
03021994000001 19940102 19941206 00001200â 000012000
03021994000001 19940102 19940103 000012000 000012000
03021994000002 19940101 19941206 00002000â 000020000
03021994000002 19940101 19940103 000020000 000020000
```

Negative numbers are represented here with Zoned Decimal according to Easytrieve. Also minus signs are accepted by Rapp.

The first lines of Kpi.Txt (from Statistics Sweden's site)

```
2011 306.15 308.02 310.11 311.44 312.02
2010 299.79 301.59 302.32 302.36 302.92 302.97 302.04 302.06 304.60 305.57 306.58 308.73 303.46
2009 297.88 297.95 298.80 299.26 299.45 300.17 298.80 299.42 300.35 301.11 301.03 301.69 299.66
2008 294.09 295.28 298.08 299.67 300.99 302.45 302.11 301.98 305.08 305.56 303.06 298.99 300.61
2007 285.01 286.45 288.33 289.79 289.48 289.95 289.49 289.41 292.30 293.85 295.75 296.32 290.51
```

See Appendix 6 for the mathematics.



Proc Calend

Produces a calendar for any year from 1582. Week numbers by ISO 8601 effective 1972.

Example:

```
N
Include C:\Rapp\Rpp\Init.Rpp
Proc Init lan(e) Endproc
Proc Calend Pdffil(C:\Rapp\Pdf\al.Pdf) CalendarYear(1889) charsperday(1) visa ENDPROC
Proc Calend Pdffil(C:\Rapp\Pdf\a2.Pdf) prompt visa charsperday(1) ENDPROC
```

Parameters

pdffil()	Output file.
visa	Displays pdf.
prompt	Makes Rapp ask for year, if calendarYear() not given.
calendarYear()	Calendar year, default present year.
charsperday()	Number of characters per day, max 14. If language is English, for Monday the value 1 gives M and the value 2 gives Mo.
indonesian	If given the language of the calender will be Indonesian (Java). Indonesian is not (yet) generally available as language in Rapp.

Proc Chaall

Example:

```
Proc Chaall root(C:\Webb\Egna mallar) files(*.asp webb*FE???.htm*) lines(1 5 12 16)
cha(Sakförsäkring Livförsäkring 'Stig Rosenlund' 'Karl Svensson') Endproc
```

Parameters

cha()	Required. Change commands in the form of pairs (string newstring). A change string with blanks or parentheses must be within single or double quotes. A change string with single quotes must be within double quotes. A change string with double quotes must be within single quotes.
ci	Case independence - string is changed to newstring regardless of its case. The default is case dependence, called MatchCase in other languages.
cols()	Blank separated number pairs, firstcolumn lastcolumn, to change. They take effect for successive change instructions. E.g. cha(x1 y1 x2 y2) cols(1 5 30 35) changes x1 to y1 in columns 1-5 and x2 to y2 in columns 30-35. The line length can be smaller than lastcolumn; if you want to change one or more blanks to something, the last characters are set to blanks before the change. Default all columns.
files()	File name patterns separated by < eg *.asp<*.htm. Default *, ie all. With * is meant any number (≥ 0) of arbitrary characters. With ? is indicated as many arbitrary characters as the number of ?. File name pattern can also be a specific file name without * and ?. Files without a file type, ie no point in the name, are indicated by << (two angles). Blanks can be part of patterns. 2020-06-08. Note. I have terminated the uses of comma as separators between file name patterns and of % as wildcard for one arbitrary character.
folders()	Folder name patterns separated by left angles < eg *xyz,*asp,*Steinberg*. With * is meant any number (≥ 0) arbitrary characters. With ? is indicated as many arbitrary characters as the number of ?. Folder name pattern can also be a specific folder without * and ?. Blanks can be part of patterns. These rules are as those for files(), but << is not relevant here. Letting folders() = root() is the same as nl.
lines()	Blank-separated number pairs, firstline lastline, to change. Analogous to cols(). Give a pair for each change up to and including the last change that shall be restricted to certain lines. Eg lines(1 999999999 2 12) if only the second change shall be delimited. And eg if five changes shall be made in lines 2 - 12, give lines(2 12 2 12 2 12 2 12 2 12). Default all lines.
nl	Files in subfolders of the root folder are not searched.
nochange	The files are not changed. All lines that would be affected are written on

the screen.

nomatchcase Synonym of ci.

notlower Synonym of nl.

root() Required. Start folder eg C: or C:\ or C:\Webb\Sak försäkring. With the flag /K last, eg C:\Webb\Sak/K, a question on modification is given.

wholeword Only strings that are whole words are changed. For example, say a string is foot, to be changed to Hand. If footprint is found in the file to be changed, it will not be changed if wholeword is set. Otherwise it will be changed to Handprint. Strings preceded or succeeded by digits or letters are not whole words. Strings preceded and succeeded by other characters are whole words. For example, foot is a word in the string [foot]. But foot(is not a word in the string foot(x. I follow the C# conventions.

If the flag /K was set: You get a screen print of each change and a question if the file shall be changed, for each file to be changed.

Example of use beside the obvious: Say you want to find all SAS programs in the folder E:\Sas\Pgm that use a specific function, say func01(), in order to modify the use of the function in a way that a simple change will not attain. Then run

```
Proc Chaall root(E:\Sas\Pgm) files(*.sas) cha('func01(' 'fu#c0$(') Endproc
Proc Chaall root(E:\Sas\Pgm) files(*.sas) cha('fu#c0$(' 'func01(') Endproc
```

Sort the file list of E:\Sas\Pgm by date descending and you find the programs that need your attention.

Note that 'func01(' etc must be framed by single or double quotes, since otherwise Rapp's parsing of the parameters by parentheses gets confused.

Proc Cmd

Example:

```
Proc Cmd
@echo off
hf listal.txt b99sti.cdoci(listal) w
copy listal.txt listalx.txt
Endproc
```

Here you write code as in a CMD or BAT program. The code is executed where it stands.

Proc Compar

Example:

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Compar fill(Rappg8.Exe) fil2(Rapp.Exe) Endproc
Proc Compar fill(Rappg8.c) fil2(Rapp.c) text visa Endproc
```

Compares two files. For text files, indicated by the parameter Text below, the proc aims to find the longest sequence of matching lines in old file and new file and to list the lines outside of this sequence. This is not a trivial problem, although not as difficult as the traveling salesman problem. I have probably not succeeded completely. If you find a counter-example, with two files where the longest such sequence is not what follows from the proc's listing, the send me a mail with those files.

Lines that are different in Old file and New file are displayed within two ' (single quotes).

Parameters

Fill	Old file.
Fil2	New file.
Firstcol()	The first column for comparing with parameter Text. Default 1.
Maxlen()	The last column for comparing with parameter Text. Default 255.

Nomatchcase Case independence - the comparison is case independent.

Pdfconv If given, and if Fill1 and Fil2 are pdfs, they will be converted to text in temporary text files, provided that the converter program ps2ascii from MiKTeX exists. Line numbers will mostly be wrong, and some other imperfections will also appear.

Quick If given, the comparison stops at the first inequality found and gives the discrepant characters, with the line where they are found if Text was given. If the files are equal, that is stated together with their sizes. If Text was given, the number of lines, minimum and maximum line length and average line length are stated.

Text If Quick was given: A stated for Quick.
If Quick was not given:
The comparison gives a listing of the lines that were deleted from Fill1 and the new lines that were inserted into Fil2, and reformatted lines. A reformatted line is one that has a different number of *leading* blanks in New file than a corresponding line in Old file, but is equal to that if both lines have their leading blanks removed. Lines with different numbers of *trailing* blanks are considered to be non-matched and non-reformatted.
If Text is not given, Quick is in effect without line statements.

Utfil() Text file for comparison results at parameter Text. Default Rappcompare.Txt in folder tempmapp.

Visa Shows the comparison at parameter Text.

Proc Coofil

Examples:

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Coofil merge infil(Sverige-forsaml.Txt) utfil(Sverige-kommun.Txt)
  InfilNewIdStartpos(1) InfilNewIdNumofpos(4) colim(4) newid ID4prefix(Komm)
Endproc
Proc Coofil merge infil(Sverige-forsaml-land.Txt) utfil(Sverige-bol-land.Txt)
  colim(4) masterfil(Sverige-forsaml-Bol.Txt)
Endproc
Proc Coofil Bordersplit Infil(Sverige-kommun.Txt) Utfil(Sverige-kommun-borders.Txt)
  Infilid(Sverige-kommun-ID.Txt) Coordidcol(1) Coordsecol(2) Coordxcol(3) Coordycol(4)
  Rpidcol(1) Labelidcol(3) alfaid(j) newid ID4prefix(Kngr)
Endproc
```

Makes a new coordinate file for Proc Map from another coordinate file. Three uses:

1. Merge areas to larger and fewer ones.
2. Split borders by which areas share which border segments.
3. Transform ID to form A + B*j or PREFnnnnnnnn. See alfa2num() and Id4prefix().

Use 1 merge can be combined with use 3. Use 3 is implied automatically for use 2 split.

See Proc Svg2co for making coordinate files from Scalar Vector Graphics files.

Parameters for use, at least one must be given

Merge
Bordersplit
Newid

Parameters for all uses

Alfa2num() Optional. Can be given as one or two numbers, alfa2num(b) or alfa2num(a b). For example alfa2num(0.001) and alfa2num(0.2 0.0001). If given, ID is converted to numerical values a+b, a+2*b, a+3*b, a+4*b, See like-named parameter of Proc Svg2co. For Bordersplit every new created distinct border segment will get its own ID. Eg, if you give alfa2num(0.5 0.01), the third created new ID will be 0.53. You can treat the new ID as alphanumeric or numeric in subsequent uses in Proc Map. Possibly you want to change the ID:s with an edit operation in the oufiles.

Id4prefix() If given ID will begin with these four characters in upper case, followed by an 8-digit number starting with Startno(). Overtakes Alfa2num().

Infil() Coordinate infile.

Startno() See under Id4prefix(). Default 1.

Utfil() Coordinate outfile.

Utfilid() Outfile with one line per ID of the borders. If not given this outfile will get name as Utfil() with -IX before the point. With parameter Merge it will contain ID, midpoint-x, midpoint-y. With parameter Bordersplit it will contain ID, bordersegment-name. With parameter Newid it will contain the new ID where the old one was while the old one is discarded.

Merging areas

It is assumed that the column numbers for ID, segment, x-coordinate, y-coordinate are 1,2,3,4. Merges areas of inputfile to the larger and fewer areas in the outputfile, according to an algorithm that gives the right result, given the condition that two corners in succession in an area of the inputfile also are two corner points in a line in a different area of the inputfile that has the distance between corner points as border, if such a different area in the inputfile exists. This condition has been satisfied in the coordinatefiles in Sverigel000plus from Statistics Sweden and the Land Survey.

Special parameters for Merge

Colim() Optional. Merged areas with a maximum of colim points (including one end point equal to the starting point) are not included in the outputfile. If not given 3 is assumed, which means that areas that are only a point or a line are not included.

Epsilon() Optional. If given > 0 coordinates x and y of the inputfile are changed to the lowest value in a group nearby. If there is a value x1 with $x - \text{epsilon} \leq x1 < x$, then x is changed to x1. The smallest of the possible x1 is selected. If given < 0 then -epsilon is used in the same way, but the outputfile will in other respects be equal to the inputfile.

Masterfil() Optional. If given it contains a table with ID and parent ID, and then spoid and npoid are irrelevant.

MasterNewIdcol() Optional. Column number of parent identity of the master file. If not given assumed 2. Can also be written as mcolnoid().

MasterOldIdcol() Optional. Column number for the ID of the master file. If not given assumed 1. Can also be written as mcolnid().

InfilNewIdNumofpos() Optional. If given > 0 the number of positions within the ID for the parent ID. Can also be written as npoid(). Default 2.

InfilNewIdStartpos() Optional. If given > 0 the starting position within the ID for the parent ID. Can also be written as spoid(). Default 1.

Example of the use of a master file with an infile that is to be merged

Master file content

```
011401 COMP00000028
011402 COMP00000028
...
258402 COMP00000024
258403 COMP00000024
```

Infile content

```
011401 0 1616982.000 6605402.000 1616391.00000 6602314.50000
011401 0 1616952.000 6605297.000
...
258403 0 1809735.000 7581527.000
258403 0 1810042.000 7581377.000
```

We want to make all coordinates for IDs 011401, 011402 etc to have new ID COMP00000028 and coordinates for IDs 258402, 258403 etc to have new ID COMP00000024. So the outfile obtains these lines. New midpoints x and y per new ID have been computed.

```
COMP00000028 0 1616982.000 6605402.000 1650412.50000 6597995.00000
COMP00000028 0 1616952.000 6605297.000
...
COMP00000028 0 1809735.000 7581527.000
COMP00000028 0 1810042.000 7581377.000
```

The second column is segment, which alternates between 0 1 to distinguish geographically separate areas within the ID.

Splitting borders

Give keyword Bordersplit somewhere in the Proc statement. Input is a file with areas, some of which might have common borders with one or more other area. For example a file satisfying the condition for merging described above.

Newid is implied. If neither of Alfa2num() and ID4prefix() was given, then Alfa2num(0 1) is assumed as default.

Output is a file of border segments, to be used with _AC=none or TP=S in Proc Map.

Special parameters for Bordersplit

Alfaid() Set j or J if the coordinate infile has an alphanumeric ID.

Coordidcol() Column number of ID in coordinate infile. Default 1.

Coordsegcol() Column number of segment in coordinate infil. Default 2.

Coordxcol() Column number of x-coordinate in coordinate infil. Default 3.

Coordycol() Column number of y-coordinate in coordinate infil. Default 4.

Rpfil() Infile with one line per ID with the columns above. Synonym Infilid().

Labelidcol() Column number of name of area in Infilid(). Default 2.

No-bd -bd will not be added to the border names.

Rpidcol() Column number of ID in Infilid(). Default 1.

Copy Utfilid() to an rpfil() for Proc Map, with risk premiums (possibly dummy values) added as a new column in each line. If the names of the areas areal, area2, etc, are name1, name2, etc, the names of the border segments will be name1-bd if the segment is border for areal only and areal/area2-bd if the segment is border between areal and area2. You can change these names, eg to areal/area2_borderline. Blanks in names are represented by underscores.

Special parameters for Newid

Alfaid(), Coordidcol(), Rpfil() and Rpidcol() as for Bordersplit.

Proc Copy

Examples:

```
Proc Copy infil(f1.txt) utfil(f3.txt) from(10) for(1000) Endproc
Proc Copy infil("f1.txt" "f 2.txt") utfil(f3.txt) for(1000) append Endproc
```

Parameters

app, append Infil() shall be added to an existing utfil(), not create it new. If utfil() does not exist, it is created new.

for() Number(s) of lines to be copied, default all.

forcol() Number of columns (positions) to be copied, default all.

from() First line(s) to be copied, default 1.

fromcol() First column (position) to be copied, default 1.

infil() Infile(s). Anyone can be equal to utfil(), it is then copied to a tempfile.

utfil() Outfile.

If multiple inputfiles are to be merged, give each in double quotes. If only infil() and utfil() and possibly append are given, then a fast binary copy is made. If at least one of the other parameters is given, the infile is supposed to have 10, 13 or (13,10) as linebreaks (LF, CR, CRLF), and the outfile will get (13,10) as linebreaks. For instance from(1) will accomplish this. If files with linebreaks to be merged have different linebreaks, eg some have 10 and some have (13,10), then give from(1) to get an OK outfile. Otherwise the outfile might become bungled.

Several from- and for-numbers can be given, eg From(10 2000) For(1000 1500).

If the security program annoys you by saying that a downloaded file, eg Rapp.Exe or Rappmane.doc, is suspect: You can run it through this proc, thereby copying it byte for byte without copying the catalogue info. Eg

```
Proc copy infil(C:\Rapp\Dok\Rappmane.doc) utfil(C:\Rapp\Dok\a.doc) Endproc
system(del C:\Rapp\Dok\Rappmane.doc & ren C:\Rapp\Dok\a.doc Rappmane.doc)
```

Another use of for() and forcol() is to copy a limited number of lines and columns of a large file to a smaller file that can be inspected in Notepad. For example an svg-file as input to Proc Svg2co to determine its attributes.

If a file exported from Excel is to be used by Rapp, it might have 13 (CR) as linebreaks. This might cause problems in rare instances. Avoid these by making a copy with from(1), thus giving the new file (13,10) as linebreaks.

Proc Copyfo

Examples:

```
Proc Copyfo infil(C:\Rapp\Xmlnew) utfil(C:\Rapp\Xml) mode 3 Endproc
Proc Copyfo infil("C:\Rapp\Xml1" "C:\Rapp\Xml2") utfil(C:\Rapp\Xml) mode 5 Endproc
```

The parameters infil() and utfil() are as the ones of Proc Copy but shall contain folders. Mode defines priorities of the folders. The outfolder needs not exist. Infolders can be postfixed with a \ and a wildcard for file names, thus copying only files satisfying the wildcard. Example: C:\Rapp\Rpp\Resv*.

Mode values

With Des (destination) folders and files are meant the outfolder and those already present in the outfolder. The infolders and -files are denoted Sou (source) folders and files. Sou folders and files without corresponding Des are added in all modes.

1. Delete destination outfolder if existing and create it new. Of Sou infolders and infiles with the same paths, the ones with the latest date-times are copied.
2. Keep the outfolder if existing. Keep Des folders and files if there are no Sou ones with the same paths, otherwise replace them. Sou infolders and infiles that are listed later in your enumeration replace earlier listed Sou infolders and infiles with the same paths.
3. Keep the outfolder if existing. Keep Des folders and files if there are no Sou ones with the same paths, otherwise replace them. Sou infolders and infiles with later date-times replace earlier listed Sou infolders and infiles with the same paths.
4. Keep the outfolder if existing. Keep Des folders and files if there are no Sou ones with the same paths. If there are Sou ones with the same paths, they replace the Des ones only if they are newer than the Des ones. Of Sou infolders and infiles with the same paths, the ones with the latest date-times are copied.
5. Keep the outfolder if existing. Replace Des folders and files with the same paths as Sou ones only after confirmation. You must run as administrator; right-click on the Rappmenus startmenu and choose that.
6. Keep the outfolder if existing. Never replace Des folders and files. Of Sou infolders and infiles with the same paths, the ones with the latest date-times are copied.
7. Keep the outfolder if existing. Never replace Des folders and files. If there are Sou files with the same paths, they are copied as name(n).ext, where name.ext is

the Des file and n is the lowest integer ≥ 2 such that name(n).ext did not exist before the copy. Sou folders and files are added. Of Sou infolders and infiles with the same paths, the ones with the latest date-times are copied.

8. The same as 7, but uses a more dangerous method of renaming the Sou files forth and back. This process must be allowed to complete. It can also be thwarted if the Sou files are occupied by you in some other application. If the total volume of the files copied is large, you might want to take the risk of messing up the Sou files for the benefit of faster execution.

Mode 5 uses Xcopy. The other modes use Robocopy. It is an inconsequential peculiarity of the Windows utilities that Xcopy needs administrator permission while Robocopy does not.

Proc Data

Example:

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Data;
Infiler fil(Skador.Txt) var(kkl Sex frdk birtdate Living $ Car_age bmkl mvikteff skkost)
urval(Skkost > 0 & Age >= 40 & Age <= 49 & kkl ne 0)
dvar(
  Age      = min(99,|[(frdk-birtdate)/10000]|)
  Sexage   = 100*(Sex-1) + age
)
delimiter(';') ;
Utfil fil(Skador2.Txt) Headerline delimiter(';')
  Var(kkl Sex Age Sexage Living Car_age bmkl mvikteff skkost)
  Sort(Car_age/D Sexage Living) Textfilexk(Skador-bortvalda.Txt) ;
ENDPROC
```

You can omit var() for the outputfile, and then all variables in the inputfile are included in the outputfile.

Here Skador.Txt is washed and mangled into a new file Skador2.Txt with the fields
kkl Sex Age Sexage Living Car_age bmkl mvikteff skkost

and with as many lines as there are in Skador.Txt which satisfy the selection condition
Skkost > 0 & Age >= 40 & Age <= 49 & kkl ne 0

The outputfile Skador2.Txt is sorted on Car_age, Sexage, Living in that order.

Example with norming

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Data arr(0(0, 0.75 , 0.93874 , 0.63 , 1.17)) ;
Var(kkl kon Ålder Könålder boende fordald bmkl mvikteff dur ndur) ;
Infiler fil(Forsakr.Txt) var(kkl kon frdk föddat boende $ fordald bmkl mvikteff dur)
urval(kkl ne 0)
dvar(
  boende_medblanka = " "!!boende!!" "
  boinx = Indci(boende_medblanka," 01 03 ") + 2*Indci(boende_medblanka," 06 ")
  + 3*Indci(boende_medblanka," 04 ") + 4*(1-Indci(boende_medblanka," 01 03 04 06 "))
  ndur = dur*arr(0,boinx)
  Ålder   = min(99,|[(frdk-föddat)/10000]|)
  Könålder = 100*(kon-1)+ålder
) dlm(';') ;
Utfil fil(Forsakr2.Txt) Headerline dlm(';')
  Var(kkl kon Ålder Könålder boende fordald bmkl mvikteff dur ndur) ;
Endproc
```

Here normed duration is calculated in a new file intended as input to Proc Taran. For example, when boende = '06 duration is multiplied by 0.93874 to give ndur.

The parameters of the main clause Proc Data has the same meaning as those with same name in the main clause of Proc Match.

The syntax for the statement Infiler is the same as for the statement Transfil in Proc Match apart from Key() and Timekey(). Ie as for the statement Infiler in Proc Taran with the elimination of the parameters associated with utfilink(). Parameters Dvar() and Urval() can be used, but Urval() cannot contain variables in dvar() made with the

function `inx()` or depending on such variables. Trying to make such a selection Rapp interrupts with an error message, so this is not necessary to have in mind.

Sort(): The syntax of statement `Utfil` is the same as in `Proc Match`. In addition, there is an optional parameter `sort()` as shown in the first example. A sort field needs not appear in the outputfile. If `/d` is suffixed a sort field, eg `Company/D`, the sort is in descending order, otherwise ascending. Numerical sort fields have to be integer-valued. They can have negative values.

Further, any number of `arg()`-statements with syntax as in `Proc Taran`, can be used with `inx()` to create new variables. The above ways to create a new variable `boinx` with the function `Indci()` can be simpler, however.

Proc Ddist

Example:

```
N
Include C:\Rapp\Rpp\Init.Rpp
Proc Ddist n(12) values(-1.30 0.22   -.1 0.24   .2 0.46   9 , -0.25 0.08   1.3 )
utfil(C:\Rapp\x.Txt) ENDPROC
```

The proc computes the probability distribution of a sum $S = X_1 + \dots + X_n$ of n independent discrete random variables X_i distributed according to `values()`. Successive distributions for $X_1, X_2, \dots$ are delimited by comma characters. If less than n distributions are given, the remaining distributions are taken to be the last one. If no comma characters are present, the random variables are thus identically distributed. In the example $n = 12$ and $P(X_1 = -1.3) = 0.22$, $P(X_1 = -0.1) = 0.24$, $P(X_1 = 0.2) = 0.46$, and $P(X_1 = 9) = 1 - 0.22 - 0.24 - 0.46 = 0.08$. For $i > 1$ the distribution is: probability 0.08 for the value -0.25 and 0.92 for 1.3. If there are at most nine decimals for the values -1.30, -.1 etc, and if they are not in E-notation, and if the values are not too far apart, the computation employs a fast algorithm. Otherwise a slower algorithm. Probabilities can be in E-notation for the fast algorithm. A possible probability for the last value per comma-separated distribution is disregarded by Rapp. Any number of blanks between numbers are admitted.

Parameters

Digits Optional. Maximum number of decimal digits with which values and probabilities can be given, and the number of digits for the output distribution of S . The latter is given as a set of mouble numbers, if digits is set to more than 15. Without digits or with digits at most 15, the computation is in double. See `Proc Mbasic` for the datatypes double and mouble and for the parameter digits.

N() Number of independent random variables.

Utfil() Outputfile.

Values() Comma-separated sequences of pairs of discrete values and their probabilities.

Visa Optional. Show outputfile in Notepad after computation.

Give n between 1 and 40. Rapp computes and adds all possible probabilities for combinations of the X_i . Max 1,100,000,000,000 combinations are allowed, so as to keep run time within an hour or two. If there are only two possible values for each X_i , then there are $2^{40} = 1,099,511,627,776$ combinations for the S distribution with $n = 40$.

The calculation is time-consuming. But it admits all discrete distributions on a finite support, within the limitations. If there is an interest in possibly faster algorithms, using inversion formulas for the Fourier- or Laplace-transform of X_i , I might start to develop those. See my paper "Practical Inversion Formulas", Scandinavian Actuarial Journal 1983: 29-38:

<http://www.stigrosenlund.se/matstat/Practical%20Inversion%20Formulas.doc>

The result is listed in `utfil()`. With parameter `visa` the list is displayed as text. Every possible value for S is shown with its probability and accumulated probability up to and including the value. Numbers in both fixed point notation and scientific notation are given if the fast algorithm was used, otherwise only in scientific notation.

Proc Diskcm

Example:

```
Proc Diskcm Spacesort(K) Numshow(All)
  Root(C:\Rapp)
  Folders(C:\Rapp\Pdf,C:\Rapp\Xml)
  files(B*.pdf,B*.Xml)
  Sorttyp /O-DN
  Textfind Any Findstr('"String">2016-' ModifyDate>2016)
  numshow(10) // numshow(all)
  Utfil(C:\Rapp\Uparm\DiskcmResult.Txt)
Endproc
```

Mnemonics for diskcm is DISK CoMputation.

The proc lists the total space, and space per folder and file, for folders, subfolders and files satsifying the name patterns of the parameters below.

Parameters

AlllDownSum Collect all space in root under direct subfolders. See info in Rappmenus.

cols() A pair firstcolumn and lastcolumn to search with findstr().

files() File name patterns separated by an angle < eg *.asp<*.htm.

With * is meant any number (≥ 0) arbitrary characters.

With ? is indicated as many arbitrary characters as the number of ?.

File name pattern can also be a specific file name without * and ?.

Files without a file type, ie no point in the name, are indicated by << (two angles). Blanks can be part of patterns. The default is files(*), ie all files.

2020-06-08. Note. I have terminated the uses of comma as separators between file name patterns and of % as wildcard for one arbitrary character.

findlines() The number of lines per file with colored search results in Rappmenus / Utilities / Disk listing. Max is 255, default is 15.

findstr() A sequence of strings, in syntax as cha() in Proc Chall, that are to be searched for in the files listed. Only files satisfying the search criteria are listed. Only their space is included in the space per folder shown.

2020-08-18. Note. Also pdf files can now be searched, although with some imperfections, such that words are sometimes not separated.

folders() Subfolder name patterns separated by left angles < eg *xyz<*asp<*Stein*.

With * is meant any number (≥ 0) arbitrary characters.

With ? is indicated as many arbitrary characters as the number of ?.

Folder name pattern can also be a specific folder without * and ?.

Blanks can be part of patterns. These rules are as those for files(), but << is not relevant here. The default is folders(), meaning all subfolders.

lines() A pair firstline and lastline to search with findstr().

nomatchcase Case independence - the find strings are searched regardless of their case.

numshow() The number of files per subfolder to show. Default is All.

pdfconv If given, ps- and pdf-files are converted to text before findstr().

renewatfind If given, the files satisfying findstr() get new date-times.

root() Start folder eg C: or C:\ or C:\Webb\Sak försäkring. Not given or given as root() means the work folder.

sorttyp After this parameter can be given one of the following parameters, to be used as paraameters to the command line dir command. The default is /ON.

Parameter	Listing sort order
/ON	Name Ascending
/O-N	Name Descending
/ODN	Time A
/O-DN	Time D

```

/OSN      Size A
/O-SN     Size D
/OEN      Extent A, Name A
/OE-N     Extent A, Name D
/O-EN     Extent D, Name A
/O-E-N     Extent D, Name D
/OEDN     Extent A, Time A
/OE-DN     Extent A, Time D
/O-EDN     Extent D, Time A
/O-E-DN     Extent D, Time D
/OESN     Extent A, Size A
/OE-SN     Extent A, Size D
/O-ESN     Extent D, Size A
/O-E-SN     Extent D, Size D

```

spacesort() Disk space unit. Values B for bytes, K for Kilobytes, M for Megabytes and G for Gigabytes. If omitted, Rapp uses the largest unit that makes the total disk space at least 1, given the name patterns.

subfolders If given, a subfolder name pattern without * and ? will get all subfolders searched also. For an empty name pattern folders() all subfolders of the root will then be searched, and if not given only the root.

textfind After this parameter can be given Any or All, determining whether at least one or all strings in findstr() shall be found in a file for it to be listed. Any file can be searched.

utfil() Required. The file with the report.

wholeword Only strings that are whole words are searched. See Proc Chaall.

Rappmenus / Utilities can be used to generate a Rapp program with a Proc Diskcm and a Proc Graf to display a listing.

2020-06-08. Note. Some more possibilities added for limiting searches for folders and files with Folders() and Files(). Use Rappmenus / Utilities and study the generated Rapp program Utmp1.Rpp.

Proc Durber

Example:

```

Proc Durber infil(Fors1.txt) utfil(Fors2.txt)
  var(falt1 fromdat falt3 falt4 4 R falt5 intilldat falt7)
  frdkvar(fromdat) todkvar(intilldat) ypremvar(falt4) datum(20020101 360 5) firstobs(2)
Endproc

```

Main function: For an inputfile with insurance versions (ie containing start-date and day-after-ending-date for successive policy premium periods), 360-day durations (banking day durations) are computed for a sequence of periods. For each read in-line is output as many out-lines as there are periods in the in-line with duration > 0. Can also be used simply to shuffle the variables in the inputfile to the outputfile with the same number of lines. That will be the case when frdkvar() and/or todkvar() is omitted. Delimiter space, semicolon or tab character is determined by the first line of the inputfile.

Parameters

datum() Optional. There are two formats:

1. datum(startdate number-of-days-per-period number-of-periods).

Example:

```
datum(20020101 360 5)
```

If the number-of-periods is omitted, for example datum(20020101 360), then it will be as many periods as are entirely in the past at today's date. If number-of-days-per-period is omitted, for example datum(20020101), it becomes 360, ie whole-year-durations, and number-of-periods is also here as many periods as are entirely in the past at today's date. Pename equals the period's starting-date.

2. datum(startdate1 enddate1 name1 [startdate2 enddate2 name2] ...)

Example:

```

datum(
  20020101 20021231 2002
  20030101 20031230 2003
  20040101 20041231 2004
  20050101 20051230 2005
)

```

An in-line with (start-date,day-after-ending-date) = (20040701,20050501) provides two lines, one with period name 2004 and duration 0.5 and one with period name 2005 and duration 0.333... = 4/12 = 120/360. It is immaterial whether the day number in the last two characters of the end dates is 30 or 31, under the principle of banking day durations. If datum() is not given, then 360 days per period and as many full-year periods as fully cover the in-line's data are used.

firstobs()	Optional. First observation of the inputfile to be processed, eg firstobs (2) if the first line is a header. Default 1.
frdkvar()	Optional. The variable in var() containing the version's start-date YYYYMMDD. If not given, there will be as many lines out as in and no duration is calculated.
headerline	A header line with field names shall be first in the outputfile.
infil()	Inputfile/files. If multiple files are to be merged, give each one within double quotes, eg infil("HelfC.Txt" "HelfD.Txt" "HelfR.Txt").
todkvar()	Optional. The variable in var() containing the version's day-after-ending-date YYYYMMDD. If not given the result will be the same as when frdkvar() is omitted.
utfil()	Outputfile.
uvar()	Optional. Outputfile variables. In addition to inputfile variables these can be listed: pername = field where the name of the period, determined by datum(), is dur = computed duration prem = computed earned premium from yearly premium and dur. If uvar() is not given, the output variables will be the same as the input variables with the addition of pername, dur and - if ypremvar() was given - prem.
var()	The infile variables as in Proc Taran et al. Type declarations are not required.
ypremvar()	Optional. The variable in var() containing the annual premium. If given, then prem = earned premium can be obtained in the outputfile.
zerodur	If given, then dur = 0 in the outputfile without prem being affected.
zeropre	If given, then prem = 0 in the outputfile without dur being affected.

If a file for each period is desired, use Proc Split afterwards with the period name as split column.

Proc Excel

Examples:

```
Include C:\Rapp\Rpp\Init.Rpp
```

```
Proc Excel listfil(EBL-listal.txt) xmlfil(EBL-listal.xml) cb visa endproc
```

```
Proc Excel textfil(EBL-skattn.txt) xlmfil(EBL-skattn.xml) strip visa endproc
```

Parameter listfil(): Makes an XML file from a listfile, including crosslist and English or German language listfile, from Proc Taran. Output file can be given as either xmlfil() or xlmfil(). It can be opened in Excel 97-2003 and later versions, but shall sometimes have extent .xml and sometimes .xlm. The following requirements 1-3 must be met for a listfile to be treated. Requirement 4 should be satisfied for the Excel output to be informative.

1. Position 1 shall be '1' when a page break is to occur. If position 1 is '+' the line can indicate a subsequent total line, but is not itself used in Excel. Otherwise position 1 shall be blank.
2. Lines with numerical list output must include
 - A. A column describing the class of the distribution argument, starting at position 2 and 10 characters long. Opening, closing and interspersed blanks may occur.
 - B. A sequence of numerical columns.
3. Each block of list output must include
 - A. One or more lines by specification 2.
 - B. Subsequently, a line that begins with '+__' or '==', which marks the border between lines for different levels of the distribution argument and an total line.
 - C. A line for the total. It must comply with specification 2 and also start with " Total " or " Totalt ".
4. The following can (should) also be provided before list output blocks to provide output column headings:
 - A. A sequence of lines at least as long as the shortest of the subsequent lines of numerical list output before the total line. Contains text describing the following lines.
 - B. Between those lines and numeric-list-output lines can be boundary lines starting with '+__' or '=='.

Part of a listfile that meets the requirements:

1							p.	2
Company:02 Blekinge								
Breed	Number insyears	Number claims	Clcost 1000:s	Risk premium	Uncer- tnty %	Premium 1000:s	Mean prem	Claim perct
+								
Angora	7420	620	1190	160	7.15	2873	387	41.4
Persian	1694	157	355	209	14.87	644	380	55.0
Oth breeds	275	9	17	60	57.85	91	332	18.1
+								
Total	9390	786	1561	166	6.44	3608	384	43.3
Company:03 Dalarna								
Breed	Number insyears	Number claims	Clcost 1000:s	Risk premium	Uncer- tnty %	Premium 1000:s	Mean prem	Claim perct
=====								
Angora	17404	1049	2211	127	5.45	7155	411	30.9
Persian	5138	421	842	164	8.35	1789	348	47.1
Oth breeds	703	51	87	124	23.72	217	309	40.1
=====								
Total	23246	1521	3141	135	4.49	9162	394	34.3

Parameter textfil(): Without other parameters than textfil() and xmlfil(), or xlmfil(), the proc makes an XML file from a textfile with a first line containing column headings. The file must have a constant number of columns per line. The textfile that is made with parameter textfil() in Proc Taran can thus be treated, but also other textfiles with a maximum of 600 columns can be treated. The XML file displays the first line with yellow background. Fields can be separated by blanks, semicolons or tab characters (x'09').

More parameters for textfil()

- Headerline()** Can be given with blank separated column headings. If some heading contains a character + or other character that can be interpreted as Rapp syntax, then enclose the line in apostrophes or citation marks. Then the first line of the input file is taken as data, not column headings. Example:
Headerline("Column_1 Column2 Column_3").
- Sp2() Sp3()** If given, the start columns for field 2 and 3. Fields 1 and 2 have fixed startpositions and can have embedded blanks.
- Strip** Blanks at the beginning and end of each field are removed.
- Types** Can be given with a sequence of characters N or \$. Example: Types(N \$ N). This means that columns given the \$ type always will be alphanumeric in Excel. Without the parameter columns with numerical content will be

numeric.

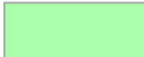
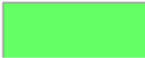
Visa Gives an Excel open of the Excel file, which then must have extent .xml.
Visa means Show in Swedish.

Parameters for listfil()

Cb Results in blank-separated thousands in Excel, for integer numbers in colored fields with comma-separated thousands in the listfile. For example, 1,234,567 will be 1 234 567.

Blanksep Gives blank-separated thousands in Excel, for integer numbers in colored fields with or without comma-separated thousands in the listfile. Eg 1234567 will be 1 234 567.

Colcomb() Provides different colors for different columns. Start with the first column after the class name, eg Bondkatt above. Put 0 for default colors. Valid other values are 1-5. For example Colcomb(0 1 1 2) gives default for Number insyears, color combination 1 for Number claims and Clcost 1000:s, and 2 for Riskpremium. Other columns get default colors. Try different combinations to see how they fit the report's purpose. For every code-value there is a color for detail lines and a color for total lines. Different Colcomb() can be given for different blocks. See TemplateB.Rpp or contact me. The colors are as follows.

0			Numerical list output, default
1			Paid values discounted to 'now'
2			Paid values discounted to 'then'
3			Prediction error in BICH
4			Premium
5			Risk premium(factor) in Tariff analysis

Kilo Integer numbers in colored fields are given in thousands. Eg 1234567 will be 1235 if Blanksep was not given, otherwise 1 235.

Mega As Kilo but in millions. Eg 1234567 will be 1.

In order to make the proc more durable, the parameter Excelproepilog() has been introduced in Proc Init. For example Excelproepilog(Proepilog-v7.xml). If given, the prologue of the XML file until its first blank line is replaced with the first paragraph of the given file, and the epilogue after the last blank line is replaced with the second paragraph of the given file. Below is the content of such a file corresponding to Excel in November 2011.

```
<?xml version="1.0"?><?mso-application progid="Excel.Sheet"?>
<Workbook
  xmlns="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:x="urn:schemas-microsoft-com:office:excel"
  xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet">
<ExcelWorkbook xmlns="urn:schemas-microsoft-com:office:excel">
<AcceptLabelsInFormulas/><ProtectStructure>False</ProtectStructure>
<ProtectWindows>False</ProtectWindows>
</ExcelWorkbook>

<WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
<Unsyncd/>
<Print>
  <ValidPrinterInfo/>
  <PaperSizeIndex>9</PaperSizeIndex>
  <Scale>21</Scale>
  <HorizontalResolution>600</HorizontalResolution>
  <VerticalResolution>600</VerticalResolution>
</Print>
<PageBreakZoom>60</PageBreakZoom>
```



```
<Selected/>
<DoNotDisplayGridlines/>t
<FreezePanes/>
<FrozenNoSplit/>
<SplitHorizontal>1</SplitHorizontal>
<TopRowBottomPane>1</TopRowBottomPane>
<ActivePane>2</ActivePane>
<Panes>
  <Pane>
    <Number>3</Number>
  </Pane>
  <Pane>
    <Number>2</Number>
    <ActiveRow>0</ActiveRow>
  </Pane>
</Panes>
<ProtectObjects>False</ProtectObjects>
<ProtectScenarios>False</ProtectScenarios>
</WorksheetOptions>
</Worksheet>
</Workbook>
```

Note. The opening program for XML files might not be excel.exe. If so, search excel.exe with Windows Explorer in the root, note its location, right-click an XML file for Open with, and go to excel.exe.

Proc Figadj

Examples:

```
Proc Figadj Endproc
```

Helps to adapt PostScript figures for use in Rapp as logos and map symbols. Run the example program for instruction. See Rappmenus / Info3 for more explanations.

Proc Filsta

Examples:

```
Proc Filsta infil(f1.txt) Endproc
Proc Filsta infil("f1.txt" "f 2.txt") Endproc
```

Applicable only to text files, including C-code, etc. Provides file statistics for the inputfile/files - the number of lines, number of bytes, the minimum and maximum line length, average line length. If multiple inputfiles, give each within double quotes. Then statistics are given for the files as if they were one file combined.

Proc Ftp

File transfer.

With s the file(s) are sent from Windows, otherwise it/they are downloaded. With b a binary transfer is made, otherwise a textfile transfer ASCII <-> EBCDIC. If multiple files are to be transferred, give each within double quotes.

Example 1. Sends a file from Windows binary to an MVS file given with whole dataset name.

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Ftp s b winfil(Hf.exe) hostfil('B99STI.EXE(HF)')
  host(lfsasp.lfnet.se) user(b99sti) password(sr47wy) Endproc
```

Example 2. Retrieving six files from MVS to Windows with prefix B99STI implied.

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Ftp host(lfsasp.lfnet.se) user(b99sti) password(sr47wy)
  winfil("For 02.Txt" "For 03.Txt" "For 04.Txt" "For 08.Txt" "For 09.Txt" "For 10.Txt")
  hostfil("Fors02.Dat" "Fors03.Dat" "Fors04.Dat" "Fors08.Dat" "Fors09.Dat" "Fors10.Dat")
Endproc
```

The proc might not work, depending on the configuration of the network. If so, you have to use some other method of file transfer.

Proc Gpdml

Example:

```
Proc Gpdml Infil(C:\Rapp\Data\Catastro-claims.Txt) Utfil(C:\Rapp\Ut\Txt\C01.Txt) Endproc
```

Produces ML estimates for the Generalized Pareto Distribution from claims in Infil(). Write claims in Infil in display format, one claim per line. See Appendix 9. Adaptation of a program originally written in 2000 for the Reinsurance Department, LFAB, Stockholm.

Proc Graf

Example:

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Graf listfil(C:\s\Agria\Agr001.Txt) listfilut(C:\s\Agria\Agr001-withgraphs.Txt)
  pdfil(C:\s\Agria\Agr001.Pdf) r u s a ENDPROC
Proc Graf listfil(C:\s\Agria\Agr001.Txt) pdfil(C:\s\Agria\Agr001.Pdf)
  R_bas=1_2_3_max=0_3_0 f_bas=1_2_3 m_bas=1_2_3 u_bas=1_2_3 s_bas=1_2_3 a ENDPROC
```

Parameters, they can be given in any order

listfil() listfilut() pdfil()

A listfile from Proc Taran as listfil() or crosslist() can be treated. For a crosslist the multivariate parameters r, f and m are irrelevant. Also lists from Proc Reschl.

Three files can be given: listfil, listfilut, pdfil. Here listfil was created with Proc Taran or in some other way, see below. The role that listfilut plays is to be a middle file with embedded SAS code that gives graphs in the PDF file pdfil. If the SAS code, created from listfil and the parameters r etc below, doesn't need to be changed, then listfilut is unnecessary. The SAS code is interpreted by a limited SAS-emulator.

Files that the user gives in Proc Graf are marked with x in the table below for different cases.

Case	listfil	listfilut	pdfil	Action if no preceding Proc Taran has been executed
1	x	-	x	from listfil is made pdfil with a temporary middle file
2	x	-	-	from listfil is made a pdfil with the same name as listfil but with extent .Pdf, with a temporary middle file.
3	-	-	x	not accepted
4	-	-	-	not accepted
5	x	x	x	from listfil is made listfilut and from listfilut pdfil
6	x	x	-	from listfil is made listfilut
7	-	x	x	from listfilut is made pdfil
8	-	x	-	from listfilut is made a pdfil with the same name as listfil but with extent .Pdf.

If a preceding Proc Taran was executed, then listfil in Proc Graf in cases 3 and 4 will be listfil in Proc Taran. Then case 3 will be case 1 and case 4 will be case 2. Cases 5-8 are normally not interesting, but if one wants to make some additions to the graphics, one can use case 6, change the SAS code in listfilut, and then use case 7.

a, a_pie A percent account of exposure (number insyears) and number of claims is given. Classes below a certain percentage p can be grouped to an othergroup with _p%, eg A_pie_2.5%.

b, b_... Gives a regression analysis of risk premium in monetary units as a linear or broken linear function of fbel or ndur. The analysis uses the series of pairs (mean sum insured, risk premium factor) that is given in listfil. Only one argument is allowed, namely an interval partitioning of fbel or ndur, in listfil, that has been created by editing away the remaining arguments that appeared in a original listfile.

b_X1=x1_X2=x2_X3=x3_X4=x4, e. g. b_X1=3_X2=9_X3=759.25_X4=900

X1 and x2 can discretionarily be given as the first and the last class for the mean sum insured points that the regression shall pertain to. If not given, the argument's first respectively last class are used.

X3 is a discretionary upper limit for x-values that is used for regression line n:o 1 in broken linear regression.

X4 is a discretionary lower limit for x-values that is used for regression line n:o 2 in broken linear regression, determined after line n:o 1 and so that the broken line is continuous and is broken in x4. If x4 is omitted but x3 is given, Rapp takes $x_4 = x_3$.

All classes get confidence intervals, and exact normal distribution does not hold.

- c, c_... As b, but gives risk premium per thousand instead of monetary units.
- dec() Optional. Number of decimal places at writenum. Default = appropriate choice.
- f, m, s, u Refers to respectively frequency (= claim frequency), mean claim, claim percent and univariate (marginal) risk premium. As r (see below), with variants f2, f2_bas= etc. For the univariate key ratios claim percent and univariate (marginal) risk premium, s_bas=... respectively u_bas=... yield confidence intervals for the quotient between claim percent respectively univariate (marginal) risk premium for a certain class and the base-class.
- fontsize() Optional. Font size at writenum. If given > 0, that size in points is used. One point is 0.352778 mm. If given as one of the negative values. -1, -2, -3, -4, -5, -6, -7, -8, -9, a size appropriate to the bar width is used. The minimum is -1, which writes the bar values with width equal to bar width plus a small margin. The value -5 gives the maximum size that guarantees that the staple values do not overlap and obscure each other regardless of the chart structure. The values -6, -7, -8, -9 give progressively larger font size, which sometimes may cause the staple values to overlap. Value -6 can be used without overlapping, if there is a maximum of two consecutive bars without separating space. Default value is -5.
- genhead() General heading for the graphs within single or double quotes. Maximum 80 characters.
- landscape The output is made as "landscape", ie with larger width than height.
- leftfoot() A footnote at bottom left. Example: leftfoot('2015, Stig Rosenlund').
- lefthead() A header text at top, like leftfoot().
- logo Relevant if t or t_bas was given. The tariff bar is built up by the company logo, if defined by logo() in Proc Init, instead of black. For instance logo(L) in Proc Init gives the LF logo.
- logo() Relevant if t or t_bas was given. The tariff bar is built up by that logo, regardless of logo() in Proc Init. Eg logo(t) gives a number of Trygg-Hansa life-buoys stacked on each other.
- nosubtitle Stops Rapp from making a graph title of a header line in a report. To be used for reports from Proc Bich made with parameters graf-akt or graf-boo.
- pos[parameter1 ... parameter10]
 where the parameters state discretionary columns for different types of graph. Default for graph form is vbar (vertical bar). At least four parameters must be given:
parameter1
 startpos for duration, arbitrary e. g. 0 if duration does not exist or exists but is not needed. In the normal case it is 13.
parameter2
 The first character = 1, A, F, N
 1: sum of column value = 100, i. e. the share of the class in percent is shown
 A: no recalculation
 F: the first column value = 100
 N: $(\text{sum of (ndur per class)} * (\text{column value})) / (\text{sum of ndur}) = 1$, where dur, appearing from pos parameter 1, is used
 Characters n:o 2-4 or 2-5 can be given as PIE or _PIE for pie charts

instead of vertical bars. At most four bar types and at most two pie charts per page. After the first underscore, or the second underscore if _PIE was given, options without blanks and separated by underscores can be written. The refer to colors and, for vertical bars, patterns, e. g.

```
A_pie_color=red,green,blue,yellow,0.7/0.7/1/rgb,lgreen
  l_color=red,green,blue,yellow,gold_pattern=L1,L2,solid,X2
```

where color and pattern are given values separated by comma (,). The colors and patterns are assigned in the order of the given columns with vertical bars and the given classes with pie. If no such options are given, default colors and patterns are used. Logos can be used as patterns. They shall be written with at least two characters, eg pattern=if for if....

Classes below a certain percentage can be grouped to an othergroup, e. g.

```
A_pie_2.5%_color=red,green,blue,yellow
```

parameter3

startpos for column 1 = the first pos after the blanks following the preceding column. Several start positions can be separated by the arithmetic operators + - * /. Those columns are then combined. E. g. 13*85*95 gives the product of exposure, mean sum insured and mean premium/(sum insured) = earned premium. With 42-66 the difference between the columns is obtained.

parameter4

column name for column 1, blanks must be represented by \$-characters

```
parameter5, parameter6 (discretionary): startpos and column name for col 2
parameter7, parameter8 (discretionary): startpos and column name for col 3
parameter9, parameter10 (discretionary): startpos and column name for col 4
```

One can use Proc Graf with the parameter pos[] on listfiles made in some other way than with Rapp, provided each table block is preceded by a line on the form

```
xxx...xxx (argument n:o xxx), xxx...xxx
```

and the heading that immediately precedes the number lines begins with " Class " (exact case) and the number lines are immediately followed by a line that begins with " ==". Lines that begin with + can be inserted. If a table of contents exists on the form

```
Table of contents
+
xxx...xxx
xxx...xxx ..... p. 2
xxx...xxx ..... p. 3
...
```

then it is supplemented with page references to the graphs.

Example:

```
Include C:\Rapp\Rpp\Init.Rpp
PROC Graf listfil(Lonk7.Txt) pdfpil(Lonk7.Pdf) s
  pos[ 13 A_color=red_pattern=l2 56 Claim freq$perthousand ]
  pos[ 13 A_color=red_pattern=x1 70 Meanclaim ]
  pos[ 13 A_color=red_pattern=solid 86 Riskpremium ]
  pos[ 13 A_color=green_pattern=x1 111 Meanprem ]
  pos[ 13 A_color=blue_pattern=solid 126 Meanprem-Riskprem ]
  pos[ 13 1 13 Number$customeryears 100 Earned$premium ]
  pos[ 13 Apie_color=red,yellow,green,blue_pattern=if
    13 Number$customeryears 100 Earned$premium ]
ENDPROC
```

r, r... Optional. States that graphic pictures for risk premium factor shall be created in PDF-format. The PDF file contains both the text tables and the pictures.

r All classes get confidence intervals in all arguments. See above under the introduction.

r_bas=3_4_5 In argument n:o 1 class 3 will be base-class, in n:o 2 class 4 will be base-class, in n:o 3 class 5 will be base-class. (If class 0 is given as base-class all classes get confidence intervals in the argument.) Then bas(3) should have been given for argument n:o 1 in Proc Taran, and the corresponding numbers for n:o 2 and n:o 3. Remaining classes that are not given after r_get base-class 0. Analogously with bas(xx/label) in Proc Taran one can give ..._xx/label... for the class name xx that shall be base class. No blanks may

occur in the string `r_bas=...`; where a blank is in a class name, write a `␣`-character instead. See below for alternative use of the parameter in order to give a blowup factor.

`r2, r2_bas=...` The 2 states that exact normal distribution is assumed for the factor estimates, with confidence degree 95 % for a coefficient 1.96 applied to standard error in the familiar way. If 2 is not given, or if `r1, r1_bas=...` is given, then Rapp gives confidence degree 90 % for a coefficient 2.00. The latter is my normal way to give confidence intervals with respect to that reality always is considerably more irregular than the nice normal distribution assumption. The exact normal distribution is however useful if the language Rapp is used for simulated data that in fact give normally distributed factor estimates.

`rightfoot()` A footnote at bottom right. Example: `rightfoot('2015, Stig Rosenlund')`.

`righthead()` A header text at top, like `rightfoot()`.

`sw` Provides Swedish standards at writenum, eg 23 456.78 instead of 23,456.78.

`t, t_bas=...` As `r`, but recalculated tariff factor is also given and compared to recalculated risk premium factor. Use `t_o0` if a list block for a specific argument has more than 99 levels and therefore is divided by you into several listfiles, where each file must have the two header lines, of which the second one begins with " Nivå" (" Class"). In such cases the computation in Proc Graf of mean tariff factors and risk factors will not be correct. To get a computation of the recalculated tariff factor Omrfakt (Recfact), use parameter `listfilut()` and delete in edit the other arguments and the graphics part at the bottom of the created file. The edited file does not have to have a line beginning with "====" and shall then not have a total line either.

Fixed factors: Example: `T_FIX=() (3_1.1_1) () (7_0.8_1.2) ()` gives
in argument 2 for class 3: 1.1 = risk premium factor, 1.0 = tariff factor
in argument 4 for class 7: 0.8 = risk premium factor, 1.2 = tariff factor
Can be combined with MAX and BAS with underscore as separator, eg
`T_FIX=() (3_1.1_1) () (7_0.8_1.2) () _MAX=0_5_0_6_BAS=1_0_2_0_1`

`tarline` Relevant if `t` or `t_bas` was given. The tariff is given by a broken line instead of by vertical bars.

`tarlinec` As `tarline` with circle contours at the tariff factors.

`tarlinecf` As `tarlinec` with filled black circles at the tariff factors.

`tarlinel` As `tarline` with the logo, if given, at the tariff factors.

`tarlines` As `tarline` with square contours at the tariff factors.

`tarlinesf` As `tarlines` with filled black squares at the tariff factors.

`visa` Discretionary. Used only in Windows. Shows the PDF file in Acrobat Reader.

`writenum` If given for bar chart type `pos[]`, then values for the bars are written in the graph.

If the listfile was produced from Proc Taran with parameter `s-GLM`, which is identified by the texts "Table of Contents" and "standard-GLM." first on a line, then Proc Graf uses `Graf x2_bas=...` `Graf x2_bas =...` (`x = r, ... , u`) where the base levels, if not explicitly given, are taken from the lines with factor 1 and uncertainty 0.

When a base class `> 0` is given by `_bas =...`, then confidence intervals are calculated in the GLM-way for the logarithms of the factors. They are then converted to the intervals for the factors. For example, Rapp computes $\log(\text{factor}) \pm 1.96 \times \sqrt{[\text{Baseuncertainty}/100]^2 + [\text{Uncertainty}/100]^2}$, whereafter the confidence interval $\text{factor} \times \exp\{\pm 1.96 \times \sqrt{[\text{Baseuncertainty}/100]^2 + [\text{Uncertainty}/100]^2}\}$ is calculated and given in the graph. This is for `r, f, m, s, u` and `r2, f2, m2, s2, u2`. With suffix 2 the coefficients 0.8416 and 1.96 are used. Without suffix 2 the coefficients 1 and 2 are used.

For R,R2,F,F2,M,M2,S,S2,U,U2,T can be given maxvalues for the height of the confidence vertical bar. Give 0 if no maxvalue is wanted. Examples: R_max=0_0_4, F2_bas=1_2_max=5_5, U_max=4_7_bas=0_1.

Should there be a base-class or not? The following can be said.

No base-class, comments

Without the parameter bas= in Proc Graf the factors for risk premium etc, univariate (marginal) risk premiums and claim percentages in Proc Graf are shown precisely as they are in the listfile from Proc Taran. All classes get confidence intervals with positive lengths. This no matter whether any base-class exists in Proc Taran or not. In order to make the interpretation easier, it is an advantage if no bas()-parameter exists in Proc Taran, but it is not necessary.

Advantages with having no base-class: One can clearly see how the classes' factors differ from the average. The comparison between any two classes is not disturbed by the confidence intervals being enlarged with the uncertainty of the base-class. The risk premium factors' confidence intervals can directly be compared with the confidence intervals for univariate risk premiums and claim percentages.

Certain base-class, comments

With the parameter bas= in Proc Graf the factors for risk premium m m, univariate risk premiums and claim percentages in Proc Graf are shown recalculated so that the base-class gets value 1 and the remaining ones are recalculated. This no matter whether any base-class exists in Proc Taran or not. The base-class gets no confidence intervals, while the remaining ones get confidence intervals with positive width that is relatively larger than the widths without the parameter bas=. This is in order to add the uncertainty for the base-class. In order to make the interpretation easier it is an advantage if the bas()-parameters in Proc Taran are set in accordance with Proc Graf, so that the numbers in the pictures are recognizable.

Advantages with having a base-class: One can compare Rapp's output with output from SAS Proc Genmod and other GLM-programs. (For more exact comparison the second variant R2 etc for the assumption of exact normal distribution should also be used.) For an argument with only two classes (e. g. Stoneground J or N) a graphic test of the hypothesis that they differ is made easier. If the tariff by tradition departs from a 1-class (100-class), the comparison between the tariff and the risk premium factor estimates is made easier. (But estimates without base-class can easily be recalculated, of course.)

Mathematically the interpretation of confidence intervals without base class is described in Appendix 1.

Alternate use of _bas in r, f, m, s, u, b, c: one can instead of base-class give a blowup factor, such that all factor estimates are multiplied with the blowup factor, after the factor estimates have been computed so that the mean value will be 1. It must be written with a decimal point in order to distinguish it from a base-class. Example:

m_bas=7_2856.0 means that in the mean claim factor graph argument, n:o 1 shall get base-class 7, but for argument n:o 2 all factor estimates shall be multiplied with 2856. Because, it can sometimes be suitable for a pedagogical purpose to give the graph in mean claim scale instead of around 1. Preferably in the mean claim case one should take the blowup factor as the ratio between the list text's constants for risk premium and claim frequency, given that all arguments have base-class average in the listfile, i. e. bas(0) which is default.

Making pdf from text

If as listfil is given a program-codefile with extent someone of .bas .bat .c .cmd .cpp .exec .ezt .h .hpp .jcl .rpp .sas .vb then the PDF file is coloured depending on the syntax of the language. Example:
PROC Graf listfil(PRISAS-tst1.rpp) pdffil(a.Pdf) ENDPROC

Formating parameters in listfil

The listfil is first made into a PostScript file, which is then made into a pdf file.

Lengths below are given mixed in inches, points and millimeters.

1 inch	= 72 points	= 25.4 mm
1 point	= 0.0140817 inch	= 0.352778 mm
1 mm	= 0.0393701 inch	= 2.83465 points

For an ASA file the only characters in column 1 are: blank 1 0 - + s *

1 new page.

0 feed a blank line before this one.

- feed two blank lines before this one.

+ do not feed a line, put the characters following the + on top of the preceding line with a right displacement. If the line is identical to the preceding line except for the +, the effect will be bolder text.

s do not feed a line, put the characters following the s at the end of the preceding line instead.

* treat the line as PostScript code to be inserted in the generated PostScript file as it is.

!R! is the start of a parameter sequence. The parameters are case dependent. Do not start it in the first column. A parameter has effect on the preceding text on the same line and after it, until it is changed. Text after the parameter sequence is not included in the pdf.

ASApageno Set page numbers if the file is ASA.

LPP number of lines per page. If not given, Rapp computes a suitable number. The font size is set sufficiently small to show all lines on a page.

SAW factor for shrinking blank space between characters, default 0.1.

SBR factor for character width, so that width is SBR*SFS, default 1.

SCS space between characters in inches, default the built-in one of the current font, 0 = default.

SFF means that line feed shall be inhibited for the line where the parameter appears.

SFO font, default Fon1.

Fon1 Courier

Fon1b Courier Bold

Fon1o Courier Oblique

Fon1bo Courier Bold Oblique

Fon2, Fon2b, Fon2o, Fon2bo Times Roman

Fon3, Fon3b, Fon3o, Fon3bo Helvetica

Fon4, Fon4b, Fon4o, Fon4bo Helvetica Narrow

The following fonts no 5, 6, 7 presuppose that Ps2pdf2.Bat, which employs Ghostscript and not MiKTeX, is used with the help of a Rapp statement

Proc Init ps2pdf(C:\Rapp\Pgm\Psf2pdf2.Bat) Endproc

You have to download Ghostscript. Search it on the internet.

Fon5, Fon5b, Fon5o, Fon5bo Arial (not much difference from Helvetica)

Fon6, Fon6b, Fon6o, Fon6bo Calibri

Fon7, Fon7b, Fon7o, Fon7bo Consolas a monospace font as an alternative to Courier. To get the same character width as with Fon1 for given SFS, set SBR 1.09 (see below).

Other PostScript fonts can be used, but å, ä, ö, Å, Ä, Ö will not come out right.

SFS font size in points, default determined by maximal line length and page length, 0 = default. Max value under default is 11.

SLM left margin in inches, default = 0.653, 0 = default. For a small value x, set -x.

SLS line space in inches, default SFS/72, 0 = default.

SLU factor for character slope. 0 = default = no slope.

SOR orientation, 1 = portrait (default), 2 = landscape.

SRL change of vertical position in number of lines, positive => higher. An !R!-line feeds one line, set SFF in the parameter sequence to avoid it.

SRV change of vertical position in inches (Set Relative Verticalpos), positive => higher.

STH factor for right displacement at first column + in an ASA file, default 0.050.

SVP vertical position in points. It can be between 1 and 830, where 830 is the uppermost position.

For literary text, use Fon2 and Fon2b with SAW 0 as in this example. The file is ASA. PostScript code is inserted to change colors with the RGB system (Red Green Blue). The s marker stops line feed so that several colors can appear on the same line.

```

Once upon a time far far away!R!SFO Fon2b SAW 0 SFS 16 ASApageno
there lived a!R!SFO Fon2 SFS 12
*1 0 0 setrgbcolor % red
s red
*0 0 0 setrgbcolor % black again
s-haired prince in a
*0 0 1 setrgbcolor % blue
s beautiful kingdom
*0 0 0 setrgbcolor % black again
s. In another kingdom nearby
lived a
*0.8 .6 0 setrgbcolor % gold
s golden
*0 0 0 setrgbcolor % black again
s-hair princess.
lNew page here.
```

This would be a somewhat extraneous application for Rapp; you would normally use Word.

Proc Grafb

1. Vertical bars and pies

Example:

```

n
Include Init.Rpp
/* Possible colors: WHITE BLACK GREEN RED BLUE GOLD PINK LGREEN
CYAN BROWN GREY SCARLET DBLUE DRED YELLOW
where D first means Dark and L first means Light. Also colors
in the form x/y/z/rgb or x/y/z/hsb as in Proc Map.
Possible patterns: SOLID L1 L2 X2 and logos. */
Proc Grafb Infil(C:\Rapp\Data\Grafb-demo1.Txt) Pdffil(C:\Rapp\Pdf\a1.pdf) // Sw
Patterns(Fo LF TR if Mo di X2) writenum Ylabel(Antal) Dec(2)
Colors(black black black black black black .7/.7/1/rgb) /* marin */ visa
Endproc
Proc Grafb Infil(C:\Rapp\Data\Grafb-demo2.Txt) Pdffil(C:\Rapp\Pdf\a2.pdf) // Sw
writenum Ylabel(Insurances) PatternsPerArg Percentstap
Patterns(di Fo if LF Mo TR X2) Colors(black black black black black black .7/.7/1/rgb)
Endproc
Proc Grafb Pie Piepctdecimals 1 Infil(C:\Rapp\Data\Grafb-demo2.Txt) Percentonly // Sw
Pdffil(C:\Rapp\Pdf\a3.pdf) landscape
Patterns(di Fo if LF Mo TR X2) Colors(white white white white white white .7/.7/1/rgb)
Endproc
```

Produces two pdf's with vertical bar graphs and one pdf with pies, one page per section delimited by .p.

Indata files:

Grafb-demo1.Txt

.p

Antal försäkringar

Separat hemförsäkring

Datum	Folksam	Länsför	Trygg-H	if...	Moderna	dina_Group
2010-03-31	1450696	735523	328514	287813	63959	40701
2011-03-31	1463403	740589	322897	290311	100720	44435

Källa: Svensk Försäkring
.p

etc with more insurance branches.

Grafb-demo2.Txt

Number of insurances Mars 31

Separate home insurance		
Company	2010	2011
dina_Group	40701	44435
Folksam	1450696	1463403
if...	287813	290311
Länsförsäk	735523	740589
Moderna	63959	100720
Trygg-Hans	328514	322897

Source: Svensk Försäkring
.p

etc.

The argument listed downwards, ie Datum in Grafb-demo1.Txt and Company in Grafb-demo2.Txt, is written i columns 1-10. The accounting concepts listed from left to right, ie insurance company in Grafb-demo1.Txt and date in Grafb-demo2.Txt, are written blank separated in columns 12-. Max 10 characters for the argument header and values, and max 38 characters for the headers of the accounting concepts. Write underscores in the latter headers for blanks. Prepend or append numeric such headers with an underscore, eg _2011.

Data lines with values for the accounting concepts are identified by having only numeric characters in columns 12-.

If parameter PatternsPerArg is given, patterns and colors are given to the argument values. If not, they are given to the accounting concepts.
If patterns() or colors() are not stated, default values are used.

Maxvalues: 99 argument values, 6 titles and 5 footnotes. For vertical bars max 99 accounting concepts. For pies max 6 accounting concepts.

2. Polygons - piecewise linear functions, max 300 points (x,y)

Example:

```
Include Init.Rpp
Proc Grafb Infil(Polygon1.Txt) Pdfil(a.pdf) polygon Ylabel(y) Endproc
```

Polygon1.Txt

```
.p
Piecewise linear function
x      y
.1      0.1
.15     0.2
.25     0.35
.3      0.36
.5      0.25
.9      0.1
```

Example n:o 1

Proc Hipseu

Example:

```
Include C:\Rapp\Rpp\Init.rpp
Proc Hipseu listfil(Hicredinsurlist.Txt) infil(Hicredinsur.Txt) p 1 Method B long ENDPROC
Proc Hipseu listfil(Hicredclaimlist.Txt) infil(Hicredclaim.Txt) p 2 Method A ENDPROC
```

Performs herarchical credibility analysis freestanding from Proc Taran. While in the latter proc insurances and claims are analyzed simultaneously, in Proc Hipseu only one of them is analyzed. The proc gives all estimators in the model. In particular the proc gives my variance pseudo-estimators according to my forthcoming article "Hierarchical credibility pseudo-estimators". My estimators are scale invariant. They have a zero as subindex to distinguish them from those of OJ2010. The correspondence to OJ2010 is this.

$$\sigma_0^2 = \sigma^2/\mu^p \quad v_0^2 = v^2/\mu^2 \quad \tau_0^2 = \tau^2/\mu^2$$

In Proc Taran these are obtained with the parameter hipsRO or hipsRO(p method p method), where method can be A, B or C. Here A is the most elaborate method, while B uses a non-pseudo algorithm for τ_0^2 and C uses a non-pseudo algorithm for v_0^2 . In the Rappmenus menu for freestanding herarchical credibility with Proc Hipseu, however, for the time being only method A is used. If another method is desired, the user has to modify the generated Rapp program and run that. The article is in Appendix 12.

The infile shall be a text file with fields

```
sector
group
exposure
exposure*claimrate
```

For p = 1, insurance file

```
sector
group
exposure
number of claims
```

For p = 2, claim file

```
sector
group
1
claim cost
```

Thus a superfluous 1 shall be present in the file for p = 2, for clarity and uniformity. One line for each claim.

The fields shall be separated by one or more blanks, or by a semicolon or tab character. If a sector or group can contain embedded blanks, the delimiter must be semicolon or tab.

Parameters

```
infil      input file as given in the example, conforming to general Rapp usage
listfil    list output file as given in the example, conforming to general Rapp usage
long       if present a long list with all parameters, otherwise only the general ones.
nosector   variable order number for sector, default 1.
nogroup     variable order number for group, default 2.
noexposure variable order number for exposure (exposure = 1 for p = 2), default 3.
noexprate  variable order number for number of claims and claim severity, respectively,
           default 4.
method     A (default), B or C
p          1 for insurances, 2 for claims
```

For how to make Excel files from the direct text output listfil, look at a generated Rapp program from Rappmenus.

Proc Init

Example:

```
Proc Init
  pathdir(C:\Rapp\Pgm)
  // SAS interpreter: example - remove if SAS not available.
  sasexe(C:\Program Files\SAS\SAS 9.1\sas.exe)
  logo(i) tempmapp(C:\Rapp\Jt) antmb(3000) priooffset(0) erralarm(5)
  pdfoffset(14) xgfact(1.03) ygfact(1.03) xgtran(-15) ygtran(3)
Endproc
```

Parameters that can be given in the proc (the order is arbitrary):

```
antmb()    RAM-memory MB available to the program. For old computers, where Windows
           functions for determining available RAM do not work. Rapp tries three
```

different methods for this determination. If all fail `antmb()` is used.

`erralarm()` If given > 0 Rapp makes a sound at error with that type as in Proc Alarm.

`excelproepilog()` See under Proc Excel.

`figuretab()` See Proc Figadj and Rappmenus / Info3.

`lan()` Language that the output is given in. Swedish = s. English = e, German = d, g, t.

`logo()` Logo in PDF files. Default RA for Rapp. Other values are DI,FO,GO,IF,LF,MO,RA,RI,SI,SW,TR. Case independent. Meaning:
DI = dina
FO = Folksam
GO = Gothaer
IF = if...
LF = LF (Länsförsäkringar Alliance)
MO = Moderna
RA = Rapp detailed picture
RI = Rapp icon, simpler picture
SI = Sirius
SV = Svenska Sjö (a blue anchor - available as Svsym in Proc Map)
SW = Swiss Mobiliar
TR = Trygg-Hansa
Use two characters in Proc Graf `pos[` after `pattern=`, eg `pattern=Mo`. In `logo()` one character is enough if unambiguous. `logo(0)` means no logo.

`pathdir()` Folders for bat-, cmd- and exe-files to be run by name only.

`pdfoffset()` Offset in points for PDF files, integer value.

`priooffset()` Offset in points for paper printout, integer value.

`ps2pdf()` Command name max 256 chars for converting a PS file to a PDF file.

`ps2pdfp()` Parameters between `ps2pdf` and `psfil` `pdffil`, default nothing.

`printer()` Printer that takes PostScript for printout.

`sasexe()` The name of the SAS exe that runs SAS.

`tempmapp()` Folder for temporary files, gives faster execution if on the C-disk. Folder name can contain blanks from July 2018.

`TempMonthCl` This will clean `tempmapp` montly from all files not used the last 14 days, depending on the content of the file `TempMonthCl.Txt` in `tempmapp`. If this file does not exist, or if it contains a date in the form YYYYMMDD that is one month or more earlier than today's date, then files unused for two weeks in `tempmapp` will be deleted, except that possible subfolders in `tempmapp` with contents will be kept. After the cleanup `TempMonthCl.Txt` will be created anew with today's date. The purposes of the parameter are to delete temporary files left after abends.

`xgfact()` X-coordinate scale factor for graphs including maps. See explanation below.

`ygfact()` Y-coordinate scale factor for graphs including maps.

`xgtran()` Amount of right translation in points for graphs including maps.

`ygtran()` Amount of upwards translation in points for graphs including maps.

The proc can be omitted. No parameter is mandatory. Default `antmb` is 1500, `lan` Swedish, `tempmapp` C:\Rapp\Jt. For `ps2pdf` the default is `ps2pdf`, for `pdfoffset` and `priooffset` 0. If `printer` is not given, it will be the default printer. It needs not take PostScript.

A point is 0.352778 mm and a mm is 2.83465 points. The placement of the text in the PDF file respectively on the paper is raised with the value given in `pdfoffset()` respectively `priooffset()`, or lowered if a negative value is given. E. g. `pdfoffset(-28)` lowers the text with one cm. Useful if the settings in PDF or the printer are changed without the

user's control. The offset parameters do not however work for the graphic pictures. How to change these is described next.

The parameters `xgfact()`, `ygfact()`, `xgtran()`, `ygtran()` are used to change graphs. Defaults are 1, 1, 0, 0, i.e. no change. The size in X is changed by a factor `xgfact()`, in such a way that the left border of the graph (paper) is unchanged. E.g. `xgfact(2)` will double the horizontal size and push the rightmost half of the graph outside the paper. In Y (vertically) `ygfact()` works analogously and leaves the bottom border unchanged. With `xgtran()` the graph is pushed, to the right if positive and to the left if negative. E.g. `xgtran(-10)` will push the graph to the left 10 points. Analogously for `ygtran()`, with a positive value pushing the graph upwards and a negative value downwards. Since new versions of `ps2pdf` in MiKTeX seem to change the placement of graphs randomly, this way of correcting the placement is needed.

The following parameters are suitable for MiKTeX 2.9.

```
pdfoffset(14)
xgfact(1.03)
ygfact(1.03)
xgtran(-15)
ygtran(3)
```

Alternatively install Ghostscript. Go to Ghostscript Downloads. Use via `Pgm\Ps2pdf2.Bat`. It is included in the zip files. Parameter `ps2pdf(C:\Rapp\Pgm\Ps2pdf2.bat)`.

If you use Acrobat Distiller, make a bat file with name e.g. `C:\Rapp\Pgm\Ps2pdfAD.bat` and the right exe file and parameters:

```
@echo off
"C:\Program Files (x86)\Adobe\Acrobat 10.0\Acrobat\acrodist.exe" --deletelog:on /F /N /Q /O %2 %1
```

alternatively on two lines with a caret as continuation symbol

```
@echo off
"C:\Program Files (x86)\Adobe\Acrobat 10.0\Acrobat\acrodist.exe" ^
--deletelog:on /F /N /Q /O %2 %1
```

Set these Proc Init parameters:

```
xgfact(0.97) ygfact(0.97) xgtran(9) ps2pdf(C:\Rapp\Pgm\Ps2pdfAD.bat)
```

What is given in Proc Init works for the rest of the run until a new Proc Init is given with other values, which supersede the previous values. Parameters that are not given in a new Proc Init are not changed. New folders in `pathdir()` are added to the previous ones.

If `tempmapp` is put on a server instead of on a disk within the PC, then the executions will be much slower.

Proc Linreg

Example, with a subsequent Proc Graf for a graph of the confidence intervals:

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Linreg infil(Lintst1.Txt) gutfil(Lingraf.txt) log level(95) Endproc
Proc Graf listfilut(Lingraf.txt) Endproc
system(del Lingraf.txt & Lingraf.Pdf)
```

Performs regular multivariate linear regression, based on H Cramér, *Mathematical Methods of Statistics*, Ch 37.3. Provides point estimates, confidence intervals and p-values. Has no specific mathematical features - use SAS (or R) for more advanced methods. The proc appears in Rapp so you can avoid paying for SAS, if you do not need more of linear regression than what the proc provides. The model is:

$Y_i = \alpha + \beta_1 X_{1i} + \beta_2 X_{2i} + \dots + \beta_p X_{pi} + \varepsilon_i$, where X_{ri} are deterministic and ε_i IID $N(0, \sigma^2)$

Also least-squares estimates of β_j are given for the model without a constant α

$Y_i = \beta_1 X_{1i} + \beta_2 X_{2i} + \dots + \beta_p X_{pi} + \varepsilon_i$

but without confidence intervals or other variance estimates. I needed such estimates in a non-statistical context, where I really wanted to minimize absolute deviations, but where least-squares were more practical.

Parameters

cols()	Optional. Column number of the variables to be used. The first given column must relate to the dependent Y-variable. If col() was not given, the first column is the dependent variable and the remaining ones the independent variables perceived as deterministic. Blank separated columns are assumed. For example "cols(4 7 3 9 5)" indicates that the line's fourth column shall be Y-variable and columns 7, 3, 9, 5 shall be four x-variables.
gutfil()	Optional. Inputfile to Proc Graf for boxplots of the confidence intervals.
level()	Optional. Confidence level in percent for confidence intervals. Default 95.
log	Optional. Indicates that the natural logarithm of each variable shall be used. Then Rapp gives both estimate and exp(estimate) for the multiplicative model in the original data resulting from the taking of logarithms.
infil()	Required. Textfile with blank separated numeric input data.
utfil()	Optional. Reportfile. If not given the infile name is used with "-ut" appended before the point. The example outputfile is Lintst1-ut.Txt

Example of infil. Col 1 is house price Y(i) and the following explanatory variables.

```

68900 5960 44967 1873
48500 9000 27860 928
55500 9500 31439 1126
62000 10000 39592 1265
116500 18000 72827 2214
45000 8500 27317 912
38000 8000 29856 899
83000 23000 47752 1803
59000 8100 39117 1204
47500 9000 29349 1725
40500 7300 40166 1080
40000 8000 31679 1529
97000 20000 58510 2455
45500 8000 23454 1151
40900 8000 20897 1173
80000 10500 56248 1960
56000 4000 20859 1344
37000 4500 22610 988
50000 3400 35948 1076
22400 1500 5779 962

```

Proc Livr

Example:

```

Include E:\Riskanalys\Rapp\Init.Rpp
Proc Livr
Include E:\Livrgrund.Rpp
perdat(20081231) /* perdat(Ovfdat) */ livrid(Livn)
var(Livn Skadenr $ Grupp Kon Foddatt Får Ovfdat
    Belopp1 Termin1 Tomdat1 Belopp2 Termin2 Tomdat2 Kapbel Kapdat)
infil(Livran.Txt) listfil(Livrskr.txt)
ENDPROC

```

where the include file Livrgrund.Rpp has the following contents:

```

grupper (
/*
----- Man -----
Kat Utbf delta      a      b      k2
999(1.01 0.0143988  0.000362 0.00001377 0.0472
11(1.01 0.0293588  0.000362 0.00001377 0.0472
12(1.01 0.0293588  0.000362 0.00001377 0.0472
13(1.01 0.0293588  0.000362 0.00001377 0.0472
21(1.01 0.0293588  0.000362 0.00001377 0.0472
22(1.01 0.0293588  0.000362 0.00001377 0.0472
31(1.01 0.0143988  0.000362 0.00001377 0.0472
----- Female -----
a      b      k2      */
0.000362 0.000008181745 0.0472) /* unknown */
0.000362 0.000008181745 0.0472) /* Cas indiv */
0.000362 0.000008181745 0.0472) /* Cas indiv */
0.000362 0.000008181745 0.0472) /* Cas indiv */
0.000362 0.000008181745 0.0472) /* Cas coll */
0.000362 0.000008181745 0.0472) /* Cas coll */
0.000362 0.000008181745 0.0472) /* Liabil eg */

```

```

32(1.01 0.0143988 0.000362 0.00001377 0.0472 0.000362 0.000008181745 0.0472) /* Liabil eg */
33(1.01 0.0143988 0.000362 0.00001377 0.0472 0.000362 0.000008181745 0.0472) /* Liabil eg */

41(1.01 0.0342014 0.000362 0.00001377 0.0472 0.000362 0.000008181745 0.0472) /* Traf old */
42(1.01 0.0342014 0.000362 0.00001377 0.0472 0.000362 0.000008181745 0.0472) /* Traf old */

51(1.01 0.0143988 0.000362 0.00001377 0.0472 0.000362 0.000008181745 0.0472) /* Traf new */
52(1.01 0.0143988 0.000362 0.00001377 0.0472 0.000362 0.000008181745 0.0472) /* Traf new */
53(1.01 0.0143988 0.000362 0.00001377 0.0472 0.000362 0.000008181745 0.0472) /* Traf new */
54(1.01 0.0143988 0.000362 0.00001377 0.0472 0.000362 0.000008181745 0.0472) /* Traf new */

61(1.01 0.0143988 0.000362 0.00001377 0.0472 0.000362 0.000008181745 0.0472) /* Liab bol */
63(1.01 0.0143988 0.000362 0.00001377 0.0472 0.000362 0.000008181745 0.0472) /* Liab bol */

/* 88 = Buyback. Then Termin shall be 8 = quarter afterwards. */
88(1.00 0.0487902 0.000362 0.00001377 0.0472 0.000362 0.000008181745 0.0472) /* Buyback */
)
terminer(
999 1.00 /* Unknown */
1 1.020 /* full-year advance */
2 1.010 /* half-year advance */
3 1.002 /* months advance */
4 1.005 /* quarterly advance */
5 0.998 /* months afterwards */
6 0.980 /* full-year afterwards */
7 0.990 /* half-year afterwards */
8 0.995 /* quarterly afterwards */
)

```

Example of a line in the input file:

```
1250 '34-8-00047-96' 53 2 19560824 2006 20070101 26900 5 20210823 14998 5 0 0 0
```

Calculates capital values of (claim) annuities or death one-time amounts in the file `infil()`. The inputfile must have a variable description `var()` with syntax according to Proc Taran. Delimiter between variables can be blank, semicolon, or tab character. The proc determines which it is from the first input line.

Necessary variables in `var()`

```
belopp1 foddatt1 fomdat1|fomdaf1 grupp kon termin1 tomdata1|tomdael1.
```

Meaning:

```

belopp1  annual amount
foddatt1  date of birth
fomdat1   from-and-inclusive-date for the annual amount
fomdaf1   day before from-and-inclusive-date, alternative to fomdat1
grupp     integer valued distribution argument och table entry to grupper()
kon       sex: 1 and M is man, other values are woman
termin1   payment term, table entry to terminer()
tomdata1  to-and-inclusive-date for the annual amount
tomdael1  day after to-and-inclusive-date, alternative to tomdata1

```

More specific variables

```

kapbel    lump sum if the annuitant lives at kapdat
kapdat     date of lump sum payment
ovfdat     transfer-date, which is used for fomdat1 if does not exist
dkapflg    flag for life- or death insurance:
            0: Life only, incl liability annuities. The amounts are annual amounts.
            1: Death Insurance. The amounts are paid for death in [fomdat,tomdat].
            2: Life + Death. As 1, plus payment of the amount at tomdata if the
               policyholder lives then.
            For Death Insurance the payout ratio Utb is used, but no payment term factor.

```

All dates shall be in the form `YYYYMMDD`.

Max 50 sets (`beloppj,fomdatj|fomdafj,terminj,tomdataj|tomdaej`), $j = 1, \dots, 50$. We write now below `fomdat(1)`, `fomdat(2)`, ... instead of `fomdat1`, `fomdat2`, ... , etc.

Derivation of missing variables from other ones

If neither `fomdat(j)` or `fomdaf(j)` exist for $j > 1$, then is set `fomdat(j) = tomdata(j-1)` if the variable exists, otherwise the day after `fomdaf(j-1)`.

If neither fomdat1 or fomdaf1 exists, then is set fomdat1 = ovfdat if the variable exists, otherwise fomdat1 = perdat.

By analogy, when neither tom-dat(j) or tom-dae(j) exist, then if possible is used either fomdat(j+1) or fomdaf(j+1) adequately.

If termin(j) does not exist, then is used termin(i) for the highest $i < j$ where termin(i) exists.

Parameters beyond var()

firstobs() The first line of data to be processed, eg 2 if the first line is a header

grupper() See the example in Livgrund.Rpp. Group number, followed in parentheses by:
 payment factor
 $\delta = \text{interest intensity} = \log(1 + [\text{yearly interest rate in percent}]/100)$
 a, b, k_2 for man
 a, b, k_2 for female
 where the death intensity is $\mu(x) = a + b \cdot 10^{(k_2 \cdot x)}$

headerline If given the outputfile gets a first header line.

infil() File with data with variable description var().

listfil() Summary of capital values per group and the calculation parameters.

livrid() Variable providing ID for an annuity such as social security number, affecting listfil().

perdat() Date at which capital values are calculated, either a numeric date YYYYMMDD or the name of a variable that contains perdat, eg ovfdat.

terminer() See the example in Livgrund.Rpp. Term number and factor for this.

utfil() File with the output, content as infil with a last variable capital value.

If you give a group number in grupper() respectively a term number in terminer() that does not appear in the data, such as 999, then those parameters are used for data lines that have group number respectively term number not found in the parameter set for grupper() respectively terminer().

The equity value calculated from δ, a, b, k_2 is multiplied with the term factor as far as annuities are concerned. For a lump sum kapbel there is no such multiplication. Finally, capital value is multiplied by the payout factor belonging to the group.

Proc Map

Example:

Include C:\Rapp\Rpp\Init.Rpp

Proc Map intervals(1 3 4 5 6 7 8 9 10 12 13 14 17 18 19 20 21 22 23 24)

coordfil(Sweden.txt) coordidcol(4) coordsecol(5) coordxcol(6) coordycol(7) alfaid(j)

rpfil(Sverige2.txt) rpidcol(3) rpcol(2) labelidcol(5)

labelfil(Sverige2.txt) labelrpcol(2) rp(j) labelcol(4)

colors(GREEN1 .5/.5/0/rgb GREEN3 brown Green5 GREEN6 .3/.6/.9/hsb RED4 GREY5 0/.6/1/Rgb BLUE1 BLUE3 0/255/153/rgb255 scarlet YELLOW RED1 RED3 RED5 PINK CYAN BROWN GOLD)

titles('Sveriges kommuner l nsvis' '2007-01-01') // max 4 titles at the top

scalefact(5e-3) midx(1097867) midy(2341169) pdfil(a.pdf) visa

ENDPROC

where the first lines of the specified files are

Sweden.txt

1	1	850	114	1	1613700	6601000
1	0	850	114	1	1615000	6602700

Sverige2.txt

850	1	114	STOCKHOLMS_L�N	UPPLANDS-V�SBY
850	1	115	STOCKHOLMS_L�N	VALLENTUNA

The proc is used to make a map in PDF with different colors for different areas. In addition, roads, railways and rivers can be plotted. Text files are input. In all those, blank lines and lines starting with a % character can be inserted; they are bypassed by Rapp. This makes development of input files easier.

Parameters

- p>
- alfaid() Optional. If given with the value j or J, the ID field is alphanumeric, otherwise a floating point number with max 15 digits in the mantissa.
- colors() Optional. Colors for risk premium intervals n:o 1, 2, If not given, default colors are used.
The following are defined in Rapp. BLUE1 - BLUE6 are in order from lighter to darker, ditto GREEN, GREY, RED.
BLACK BLUE BLUE1 BLUE2 BLUE3 BLUE4 BLUE5 BLUE6 BROWN CYAN DARKBLUE
DARKRED GOLD GREEN GREEN1 GREEN2 GREEN3 GREEN4 GREEN5 GREEN6 GREY GREY1
GREY2 GREY3 GREY4 GREY5 GREY6 LIGHTGREEN MARIN PINK RED RED1 RED2 RED3
RED4 RED5 RED6 SCARLET WHITE YELLOW
In addition, colors by RGB or HSB-principle (RED, GREEN, BLUE respectively (HUE,SATURATION,BRIGHTNESS) can be given in the form x/y/z/rgb and x/y/z/hsb respectively, by the example. Here x, y, and z are between 0 and 1, unless rgb255 or hsb255 is given for integers <= 255. If a number is given, it will be the index for the subsequent color, eg 7 Scarlet 12 Marin. BLUE1 is actually rather dark, equal to 0/0/1/rgb. To get still lighter blue, give for example .7/.7/1/rgb. Similarly for RED1 and GREEN1.
- consolas Optional. If given Fon7 will be used for texts instead of Fon1 (Courier). Search Fon7 in this document.
- coordfil() Required. One or more files with ID-field, segment, x-coordinate, y-coordinate, Files with blanks inside the name shall be surrounded by double quotes. Example:
Coordfil(Bolgrans.Txt "Sjöar i Sverige.Txt" "Gator i Sverige.Txt")
If areas or lines overlap, later appearing coordinates have priority, so that "Gator i Sverige.Txt" have priority over Bolgrans.Txt.
- coordidcol() Optional. Column number 1, 2, ... for ID i the coordinate file, default 1. An alphanumeric ID can be max 40 characters. It is case dependent.
- coordsegcol(), coordxcol(), coordycol(): Optional column numbers for segment, x-coordinate, y-coordinate. If not given assumed to follow directly after the ID-column.
- intervals() Optional. Upper limits for grouping of a numerical accounting value. The names below starting with rp are chosen because the accounting values of applications are often factors or risk premium or similar concepts. A last group are values exceeding the last given. In the example the 21:st and last group is for values larger than 24. If intervals() was not given, then all existing values are displayed.
- labelcol() Mandatory if labelfil() was given. Column number 1, 2, ... for label-text in labelfil. Represent blanks with underscore.
- labelfil() Optional. File with texts for the accounting values.
- labelidcol() Optional. Provides column number in rpfil for text describing the ID. If given > 0 texts are displayed in the map with the beginning (TP=R), the end (TP=L) or the middle (TP=C) of the text on the approximate midpoints of the areas. The midpoints must then exist as two additional columns in the coordinate file in the first line of each area. Alternatively, with TP=S the text is written repeatedly on the borderline (road, river etc).
- labelrpscol() Required if labelfil() was given. Column n:o in labelfil() of accounting value, alternatively indices 1, 2, ..., by intervals(). See also rp().
- labels() Optional. Texts, given within single quotes, in the bottom for the accounting value in ascending order. If labels() and labelfil() were given simultaneously labelfil() has priority.
- lakecolor() Optional. Color of lakes and the sea according to landfil(), default ocean blue. Valid colors according to colors().

landfil() Optional. A file like coordfil with coordinates. The area not in landfil is colored with lakecolor. Intended for a height contour that gives a certain height above sea level, such as five meters, and illustrates the area still land following a rise in sea level to that height. The column numbers must be according to overcoordidcol, overcoordsegcol, overcoordxcol, overcoordycol.

landscape The map is made as "landscape", ie with larger width than height. While non-landscape ("portrait") is best for eg Sweden, landscape is best for eg Indonesia and the United States.

leftfoot() A footnote at bottom left. Example: leftfoot('2015, Stig Rosenlund').

lefthead() A header text at top, like leftfoot().

meterperunit(): Optional. Number of meters for one unit in the x- or y-coordinate. If given, a scale with text eg 100 kilometer will be given in the bottom right corner. In the coordinate system RT90 one unit is one meter.

midx() Optional. The map's center will get that x-coordinate value if given.

midy() Optional. The map's center will get that y-coordinate value if given.

noborders Optional. Suppresses the border lines of areas.

nolabels Optional. Suppresses texts for the accounting value at the bottom.

nonlogo Optional. Suppresses company logo at top left.

nonsymprop Optional. Symbols and line widths keep their sizes in mm after a change of scale instead of being changed by the same factor as the surfaces.

Notruncstreetname Optional. If given, then Rapp tries not to write shortened street names. For instance, Oxford_Street will not be written Oxford Stre.

overallfaid() Optional. If given with value j or J, the ID field in overcoordfil is alphanumeric, else numeric.

overcolor() Optional. Color of areas given by overcoordfil. No color if not given.

overcoordfil(): Optional. A file like coordfil with coordinates whose district boundaries are drawn with dotted lines - - - -.

overcoordidcol(), overcoordsegcol(), overcoordxcol(), overcoordycol(): Optional. Provide column numbers for overcoordfil and landfil. If those files were given but not these column numbers, then the same column numbers as for coordfil are adopted.

parmc() Optional. Column number (word number) in rpfil for the parameters described below under rpfil. Alternatively, the parameters can be given in the Rapp-program as addition to rpfil after a slash, eg, for all lines in Roads2-ID.Txt with
rpfil(Roads1.Txt Roads2-ID.Txt/_AL_AC=none_LC=grey4_LW=0.0118_LT=0)
Parameters in rpfil have priority over such parameters in the program.

pdf() Required. The PDF file that the map is created in.

rightfoot() A footnote at bottom right. Example: rightfoot('2015, Stig Rosenlund').

righthead() A header text at top, like rightfoot().

rpfil() Required. One or more files with the ID concept and a numerical accounting value, such as a risk premium factor, which shall be illustrated in the map. Files with blanks inside the name shall be surrounded by double quotes. Additional optional parameters for accounting are given in one blank-delimited word in the parmc:th column in the file or in the program after a / according to
_AL_RD_AC=areacolor_LC=linecolor_LER_LT=linetype (the word continues)

`_LW=linewidth_TC=textcolor_TF=font_TFS=charwidth_TP=textplace`

With `_AL` is indicated that this ID is plotted on the map after the area not in landfill was colored with lakecolor.

With `_RD` is indicated that duplicate line segments need to be removed. Two line segments are considered duplicates if they have the same endpoints and the same linewidth. If only same endpoints are to be used for identifying duplicates, write `_RDI`, where I is for Independently of linewidth. Later appearing line segments have priority.

With `AC=none` (or `ac=NONE`, etc) is indicated that the field with that ID shall not be colored. `AC=(ID=x)` indicates color as for areas with ID x. Give x in single or double quotes if ID is alphanumeric and can include right parentheses but not quotes. Eg `AC=(ID="N1:(37)")`. `AC=(RP=x)` indicates color as for areas with risk premium x. If the area is not an area and its coordinates give the corners of a road, railway, river, brook etc, give `AC=none`. Alternatively, `TP=S` for the same effect.

After `AD=` you can give L1, L2 and X for diagonal lines in the area. (Mnemonics: AD = Area Diagonals.) L1 produces going from down left to up right. L2 produces lines from up left to down right. X produces both sets of lines. At `ADC=` give line color in x/y/z/rgb or x/y/z/hsb form. At `ADLW=` you give line thickness. At `ADLD=` you give distance between lines. Measure units as for `LW=`, with an m last for meters.

After `_LC=` you give the color that the road etc gets in the map. The colors above and arbitrary rgb/hsb-colors in the forms x/y/z/rgb and x/y/z/hsb can be used.

Give `_LER` (Line Ends Round) for rounded line ends. Otherwise the lines will be squared off at the end. The line length will be somewhat larger with rounded ends. The parameter causes the execution of the PostScript statement "l setlinecap" for the line or area boundary.

After `LT=` you give line type, where 0 is a solid line, a number `x < 0` gives alternating gray and colored lines with length `-x`, and a number `x > 0` gives alternating colorless and colored bars with length x. If the first character after the number x is m or M, eg 1.23m, then x (or -x) is taken to be meters in the real world. Otherwise x is millimeters in the map on A4-paper, ie `|x|/2.83465` points, provided `xgfact()` and `ygfact()` in Proc Init were given as 1. Negative x and white line color may be suitable for railways. For meters, the size of one coordinate unit must be known. If `meterperunit()` is given, this is used. If not, one coordinate unit is supposed to be one meter, as in RT90.

After `_TC=` give color for text describing the ID. As for all colors in proc Map, it can be given as x/y/z/rgb or x/y/z/hsb.

After `LW=` give line width. As for `LT=` an m or M after the number means real-world meters for real proportions, otherwise millimeters in the map, depending on `xgfact()` and `ygfact()` in Proc Init.

After `_TF=` give b, o, bo for respectively Bold, Oblique, BoldOblique text. Give u for uppercase. The order of b, o and u has no effect. The monospace font Courier is used.

After `_TFS=` give text character width in millimeters, depending on `xgfact()` and `ygfact()` in Proc Init. Then all texts describing the ID will have the same font size, unless a U is given. Otherwise the texts will have the same total length, so that eg Landskrona with 10 characters will have half the font size of Malmö with 5 characters. Give a U after the width, eg `_TFS=0.5u`, if the text shall be no wider than the width computed without the `_TFS=` parameter. Do not normally use this parameter for routes and water flows which have `TP=S`, for then the text within the lines of the ID might not fit. Without `_TFS=` Rapp computes the proper fontsize for exact fit for routes.

After `TP=` give placement of the text in column no labelidcol: C = Center for centering around the midpoint, L = Left for extension to the left of

center, R = Right for extension to the right, and S = Street for placement in the line of route.

Example railway line, with the statement: `parmc(3) labelidcol(2):`

```
0.01 _ AC=none_LC=white_LW=2.5m_LT=-0.3937
```

Example European route:

```
0.02 E4 TP=S_LC=white_LW=1.2_LT=0_TF=Bu
```

`rp(icol(), ricol())` Required. Column no for ID and a numeric accounting value in `rpfil`. See below for how to give column numbers specific to a particular file.

`rp(j)` Optional. Value `n` or `N` indicates that the indices 1, 2, ... are in the accounting value column of `labelfil`. If another value was given, then the values in the column `n:o labelr(icol())` are divided into intervals according to `intervals()`.

`scalefact()` Optional. Scale factor = (number of points per coordinate unit) to be used instead of the default computed by Rapp and shown at parameter `scalewrite`.

`scalewrite` Optional. If given, Rapp writes on the screen the factor by which Rapp converts the coordinates to points in PostScript, and `x-` and `y-`coordinates for the midpoint.

`symbfil()` Optional. One or more files with coordinates for items to be displayed with a particular symbol, and possibly a text that is located close to the symbol. Give one of NE, N, NW, W, SW, S, SE, E for placement of text northeast, north etc of the symbol after the `textwidth` parameter in `symbols()` without blanks in between. Default is NE. See the syntax for `symbols()` below). Files with blanks inside the name shall be surrounded by double quotes. Example of a line in `symbfil`:

```
1414017 6249652 1 Åköping
```

With the `symbol()`-statement as the example below, the text Åköping will be displayed with overall width 10 mm, just northwest of a green triangle with 0.4 meters height.

The parameters can also be given as line-parameters in the 5:th blank-separated word in `symbfil()`. See below. They will then be applied only to the line they are written, while parameters in `symbols()` apply to all symbols with the same `symindex`, eg 1 above and below for a green triangle. Line-parameters have priority over parameters in `symbols()`.

`symbols()` Optional. Provides color, shape, `symbolheight` in mm or meters, `textwidth` in mm, angle counterclockwise in 360-degrees-scale, for points in `symbfil`, if no scale change with `scalefact()` was made.

At scale change symbol height and `textwidth` are changed proportionately. unless the parameter `nonsymprop` was given.

Syntax:

```
symbols(symindex (symcolor[_textcolor] form symbolheight[m]
[textwidth[c][place][_b][o][u]] angle) ... )
```

where `symindex` must be an integer ≥ 0 . If `symbolheight` ends with `m` or `M`, it is real-world meters, otherwise millimeters in the map on A4-paper, depending on `xgfact()` and `ygfact()` in Proc Init. See above at `LT=`. Give `c` or `C` after `textwidth` if the width of one character is given. Place is one of NE, N, NW, W, SW, S, SE, E. Give `b`, `bo`, `o` after an underscore for bold and/or oblique text. Give `u` for uppercase. It is not necessary to have a text. Give an underscore in column 4 if none is wanted but you have line-parameters in column 5.

Example:

```
symbols(
0(blue_.1/.2/.3/hsb circle 1.2 7W_bu) 1(green triangle 0.4m 10Nw)
2(red bar) 3(blue bar 0.4 10s -45) 4(- Risym 2 10cSW) 5(- Fig7 1 5 90)
)
```

symbol line-parameters: They are written in one word, the 5:th. If you do not want any

line parameters for a line, but want a comment in it, then write a single underscore as the 5:th word and your comment in the 6:th etc.

Syntax:

```
x-coordinate y-coordinate symindex text _SA=angle (the word continues)
_SC=color _SH=symbolheight[m] _TC=color _TF=font[u] _TP=place (cont.)
_TW=textwidth[c] _textxadd=factor-on-fontwidth-rightshifting-text (cont.)
_textyadd=factor-on-fontwidth-raising-text
```

where angle, color, symbolheight, place and textwidth work as the like-named parameters of symbols(), while font works as described above under rpfil(). Give _TF=0 for non-bold, non-oblique text. Give a u for uppercase. The _TC= color is for the text, eg red for provinces. _textxadd and _textyadd can be negative.

Example:

```
1414017 6249652 1 Åköping _TP=nw _SH=0.4m _TW=10 _SC=green _TF=ub _SA=-45
```

Colors as shown above under colors().

The forms, including logos for certain insurance companies and Rapp, are: airplane arrow bar check circle clubs cross diamonds disym eight five flower flower2 flower3 flower4 flower5 flower6 Fosym four Gosym halfcircle hand hand2 hand3 heart hearts ifsym LFsym Mosym nine one pen pencil Rasym Risym scissors seven Sisym six snow spades square square2 star star2 star3 star4 star5 star6 starcircle Svsym Swsym tape telephone ten three triangle triangle2 Trsym two. Also 1001- and user-defined PostScript figures Fignnn. See Proc Figadj. Most symbols are ZapfDingbats letters.

2023-02-08. As form you can now give the number 1 or 2 followed by a file name of a PostScript file. The number 1 is for a simple PostScript file which can be embedded in a proc, while 2 is for a complicated file with images. If 1 works, the 2 will also work, but 1, if possible, gives often a smaller and faster output file. The figure of the file should fit in a square with middle 0 0 and width = heigth = 92 points. For example 1C:\Rapp\Txt\Sym01.Ps and 1 C:\Rapp\Txt\Sym 02.Ps.

Default values:

Color blue, form circle, symbolheight 0.8 mm, textwidth 0 mm, angle 0°. If angle is given, then the symbol is turned counterclockwise around its center, for example 90 for the symbolen HAND makes the hand point up instead of right. Also logos can be rotated (as of 2013-07-21).

Note. Text placement can be adjusted with underscores. These will be invisible but take space on the map. Eg, if the text Åköping northwest of the symbol obscures something at the ending ng, then give Åköping__.

titles()	Optional. Max four strings each max 94 characters at the top of the page.
titletran()	Optional. Moves titles and logo the given number millimeters up.
txtprop	Optional. Text sizes are changed after a change of scale.
visa	Optional. Shows pdf file in the default Pdf reader.

More on symbols: If text only is desired, give the symbol blank. Other symbols airplane ... two are shown below in PDF. Parameter S was given after textwidth in 2.2cS.



Symbols.pdf

You can obtain the PostScript code for symbols in the pdf with numbers 1001 etc. Say you want it in the file a1.ps. Put the following line right before Proc Map

```
Proc Init ps2pdf(copy) Endproc
```

In Proc Map, write

```
pdf fil(a1.ps)
```

To obtain a pdf file, with name a1.pdf: After the Endproc for Proc Map, write

```
system(ps2pdf "a1.ps" "a1.Pdf")
```

To look at the code eg. for symbol 1099 (Dogs on leash), search the file for %Figure 1099 and %Figure 1099 End.

■

The files shall have blank separated columns up to and including the highest column number given by the *col()-parameters for the file in question.

The position and scale of the coordinates have no effect, provided the same scale factor applies to x and y. That is, if (x,y) is replaced by (a1+b*x,a2+b*y) in the coordinate file the result will be the same.

Specific column numbers per riskpremium file

You may want to combine different rpfil:s where ID, risk premium, ID label and parameters in parmcol are located in different columns. Then you can give the column numbers in the same word (= sequence of characters without a blank) as the rpfil after an angle < with an underscore before all column numbers. An example, where _parmcol(99) indicates that there are no parameters in the file:

```
rpfil(
...
Slpva-ID.Txt/_AL_LW=.001<_rpidcol(1)_rpcol(5)_labelidcol(2)_parmcol(99)
Slphy-ID.Txt<_rpcol(5)_labelidcol(2)_parmcol(99)/_AL_TP=S_LC=marin_LW=0.3937_LT=0
)
```

The column numbers not specified for a certain file are replaced with the general parameter. For example rpidcol for Slphy-ID.Txt becomes that indicated by rpidcol() preceded by at least one blank.

Purpose of the parameters scalewrite, scalefact, midx, midy

You may wish to determine the scale yourself, either to enlarge the map and show some part of it or to make a series of maps of various areas with a common scale factor, which may usefully be the smallest of the series of default scale factors.

Example:

We want to show magnified the municipality Heby 0331. Run a program for all Swedish municipalities with scalewrite set. Note the computed scale factor s1 and run again with midx(1097867) midy(2341169), which is Heby's approximate center coordinates, and with scalefact(s2) where s2 = s1*(height of map)/(height of Heby municipality).

More examples

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Map visa coordfil(Sverige-bol-land.Txt) overcoordfil(Sverige-lan-land.Txt)
    rpfil(Bol.Txt) rpidcol(1) rpcol(1) labelidcol(1)
    labelfil(Bol.txt) rp(j) labelrpcol(1) labelcol(2)
    titles('Sweden per county company' 'County border lined with - - - ') pdfil(a.pdf)
ENDPROC
Proc Map coordfil(Sverige-bol-gator.Txt) overcoordfil(Sverige-lan-land.Txt)
    symbfil(symtst.txt) symbols( (red starcircle 1) (green circle 1 5)
    (brown flower3 1 5) (cyan bar 1 5) (_ 1 C:\Rapp\Txt\Usflag.Ps _ _ ) )
    rpfil(Sverige-bol-gator-ID.Txt) rpidcol(1) rpcol(1) parmcol(4)
    labelidcol(3) nolabels pdfil(a2.pdf) visa
ENDPROC
```

In the example below surfaces, lines and points are drawn in the following order, as determined partly by the order in which the coordinate files are provided, and partly by the parameter _AL in the rpfil:s.

Sverige-bol-land.Txt	County companies including Gotland.
Roads.Txt	Railways.
Roads1.Txt	European routes and highways.
Land1.Txt	What is not in the land contours is white, including Gotland.
Lakes.Txt	Lakes.
Islands.Txt	Islands, including Gotland and Öland which are now shown again.
Roads2.Txt	Road class under highways.
Slport.Txt	Points for urban areas.

```

Include C:\Rapp\Rpp\Init.Rpp
// Proc Init ps2pdf(Copy) Endproc activate to make pdf file a PS-file.
Proc Map coordfil(Sverige-bol-land.Txt Roads.Txt Roads1.Txt
    Lakes.Txt Islands.Txt Roads2.Txt)
    landfil(Land1.Txt) lakecolor(white)
    rpfil(Bol.Txt Roads-ID.Txt Roads1-ID.Txt
        Lakes-ID.Txt Islands-ID.Txt Roads2-ID.Txt)
    rpidcol(1) rpol(1) labelidcol(2) parmc(3)
    labelfil(Bol.txt) rp(j) labelrpol(1) labelcol(4)
    symbfil(Slport.Txt) symbols( 0(red square 0.41 1.2) 1(grey5 star 0.4 1)
        2(grey5 circle 0.25 0.5) 3(x lfsym 0.6 2) )
    titles('Sweden per company' 'With roads, railroads, municipalities'
        '@Stig Rosenlund 2008')
    pdf file(Sverige.Pdf) visa
Endproc

```

Examples of lines in the rpfil:s. Underscore in an ID label will be blank.

Bol.Txt	Roads-ID.Txt
02 02 _ 02_Blekinge	0.401 _ AC=none LC=black LW=0.0397 LT=-0.157
03 03 _ 03_Dalarna	0.402 _ AC=none LC=black LW=0.0397 LT=-0.157
04 04 _ 04_Älvsborg	0.403 _ AC=none LC=black LW=0.0397 LT=-0.157

Roads1-ID.Txt	Lakes-ID.Txt
0.001 E10 TP=S LC=white LW=0.0591 LT=0	0.0008702 _ _AL AC=white
0.002 E10/RV_45 TP=S LC=white LW=0.0591 LT=0	
0.003 E12 TP=S LC=white LW=0.0591 LT=0	

Islands-ID.Txt	Roads2-ID.Txt
0.000001 _ _AL AC=GREY LW=.00787	0.15001 _ _AL AC=none LC=black LW=.0157 LT=0
0.000002 _ _AL AC=GREY LW=.00787	0.15002 _ _AL AC=none LC=black LW=.0157 LT=0
0.000003 _ _AL AC=GREY LW=.00787	0.15003 _ _AL AC=none LC=black LW=.0157 LT=0

```

Slport.Txt
1514213.590 6803407.980 2 Alfta
1306520.980 6427537.350 1 Alingsås

```

Proc Match

Example:

```

Include C:\Rapp\Rpp\Init.Rpp
Proc Match unmatched(t) umvalue(X) stats ;
Masterfil fil(Forsakr.Txt)
    var(Xkod Bilmärke $ Bolag Forsnr Hj $ Kon Radnr Geografi $ Fromdat Foddad Dur Prem)
    dvar(
        Ålder = min(99,|[(Fromdat-foddad)/10000]|)
        Könåld = 100*(kon-1)+ålder
    )
    Key(Bolag Forsnr Hj Radnr) Timekey(Fromdat)
    firstobs(2) delimiter(';') ;
Transfil fil(Skador.Txt) var(Skkost Xkod Skadedat Bolag Forsnr Hjsiffra $ Radnr)
    dvar(Antskad = 1 Kvadr = Skkost*Skkost)
    Key(Bolag Forsnr Hjsiffra Radnr) Timekey(Skadedat)
    firstobs(2)
    urval(Skkost > 0)
    delimiter(';') ;
Utfil fil(Skadormatchade.txt) Headerline delimiter(9) noq
    var(Skkost M.Xkod Antskad Kvadr Bilmärke Kon Geografi $ Ålder Skadedat Könåld Fromdat);
ENDPROC

```

This proc matches a transfil (transaction file), such as claims, against a masterfil (master file), such as insurance versions. Transfil and masterfil need not be sorted by key fields. The proc is not as universal as eg SQL, but in return, adapted to claim statistics. There are certain restrictions on the match keys, see below.

Syntax, similar to Proc Taran, and how the proc works is shown in the example. The general rules for Rapp apply, such as case insensitivity, and that line breaks and the number of blanks between words is irrelevant.

Parameters of the main clause Proc Match, shall be completed with semi-colon
 alphasize() Optional. As in Proc Taran.

arr(), begperiod(), endperiod() Optional. As in Proc Taran.

q
 quick Optional. Specifies that the match shall be made in memory with hashing instead of alternating reads of sorted files. The masterfil keys and outfields are placed in RAM, which must be sufficiently large. If it is not, Rapp will interrupt with a message about it. This match type can be significantly faster than alternating reads of sorted files, provided the masterfil is small enough. For a sufficiently large masterfil it is slower. The outputfile has the same sort as the transfil. The parameter is not used with parameter timekey(). If the latter is given, then alternating reads of sorted files is used regardless of Quick.

stats Prints on the screen: The number of lines in the masterfil and transfil, number of matched lines in the transfil and the number of lines in the outputfile.

umvalue() Field value for unmatched transfil-lines, default "0".

unmatched
 unmatched(t) The outputfile obtains one line per line in the transfil if unmatched is given. Otherwise, one line per line in transfil that was matched against the masterfil on the keys in Key(). Unmatched transfil lines get the value in umvalue() in the outputfile's masterfil fields.

unmatch(m) If given, then unmatched lines in the masterfil are written to the outputfile, with 0 in the transfil fields. But only one of several unmatched masterfil lines with the same combination of keys is output. Thus not symmetrical treatment of master and trans, because all matched transfil lines, and unmatched ones if unmatched or unmatch(t) was given, are output. Unmatched master lines obtain the corresponding values of the masterfil key fields in the transfil key fields, for those of the latter that have been included in the outputfile. The parameter is not intended for use when timekey() (see below) is given.

Parameters and syntax in the statements Masterfil and Transfil, closed with a semicolon
 Same as for the statement Infiler in Proc Taran, with the addition of parameters Key() and Timekey() and removal of the parameters associated with utfilink(). The latter are generated internally in the creation of a temporary Proc Taran used for matching.

key() Required. The keys for matching. They shall correspond word for word in the masterfil and transfil. An alphanumeric key in one file must be alphanumeric in the other file. The datatype of a numeric key in one file can be different from that of the corresponding key in the other file. The datatypes integer and floating point are OK, but the values must be integers. If a numeric key field can be of absolute value larger than 2147483647, it must be declared as floating point with R and then it may have a maximum of 15 digits (or 16 digits with a leading figure not more than 5), since the mantissa of an 8-digit floating point number allows a maximum of 15 significant digits without risk of error. Also with negative numeric key values the outputfile will get the order from lower to higher key values (new from 2015-04-04). With alphanumeric keys containing å, ä, ö, Å, Ä, Ö, the outputfile is not in order from lower to higher key values in the Swedish order of characters, because the key comparison concerns the characters alphanumeric scheme, where for example Å < Ä. The files need not be sorted on keys. Rapp performs sorting of temporary files if necessary.

timekey() Optional. If given it shall relate to an 8-digit date in each file. Then the transfil line is matched against the masterfil line, with the right keys in the Key(), which has the greatest date that is less than or equal to the transfil line's date. If no such date exists, the matching masterfil line with minimum date is used. The matched or unmatched condition is determined solely by Key(). This is the proper way to match claims with Timekey = claimdate. against insurance versions with Timekey = fromdate of the version, if you want get the best insurance information transmitted to the claims. 8-digit dates from the year 1000 are positive and fill eight characters, so this is right even with the alphanumeric comparison described above.

Parameters and syntax in the statement `Utfil`, preceded by and completed with a semicolon

<code>delimiter()</code>	Optional. Delimiter between fields in the outputfile. If not given the
<code>dlim()</code>	fields will be blank separated. For example <code>dlim(9)</code> for tab separated fields.
<code>fil()</code>	Required. Outputfile in text format.
<code>headerline</code>	Optional. If given the outputfile gets an initial header. It must then be skipped. at a later reading in <code>Proc Taran</code> with <code>firstobs (2)</code> .
<code>noempty</code>	An alpha-numeric field will never be empty. If it would have been empty without the parameter, it will contain one blank. The purpose is to admit input into SAS in <code>Proc Sasin</code> .
<code>noq</code>	Optional. Removes the qualifiers M. and T. from the header line, if <code>headerline</code> was given.
<code>var()</code>	Required. The outfile fields. Must be in <code>transfil</code> or <code>masterfil</code> . If a field is in both of these files, it is taken from the <code>transfil</code> , if it was not stated with M. before the field name that it is to be taken from the <code>masterfil</code> . In the example, <code>Xkod</code> was given as <code>M.Xkod</code> so that it is taken from the <code>masterfil</code> . Type declarations may be present.

Proc Matrix

Example:

```
Include C:\Rapp\Rpp\Init.Rpp
```

```
Proc Matrix Inv
```

```
  Infil(Matr1.Txt) Utfil(Matr2.Txt) Detfil(MatrDet.Txt) Sympos Format(S)
```

```
Endproc
```

```
Proc Matrix Inv Infil(Matr1.Txt) Utfil(Matr1.Txt) Detfil(Matr1.Txt) Endproc
```

```
Proc Matrix Inv Infil(Matr1.Txt) Utfil(Matr1.Txt) Detfil(Matr1.Txt) LU Endproc
```

```
Proc Matrix Mul
```

```
  InfilL(MatrLeft.Txt) InfilR(MatrRight.Txt) Utfil(C:\Rapp\Txt\Result.Txt) Format(S)
```

```
Endproc
```

```
Proc Matrix Trp Infil(Matr1.Txt) Utfil(Matr1.Txt) Endproc
```

Performs matrix inversion, multiplication, transpose and determinant computation on matrix infiles in text format. Matrix elements can be delimited by blanks, semicolons or tab characters.

Parameters

<code>Detfil()</code>	File for determinant, if <code>Inv</code> was given. If the same as <code>Utfil()</code> , the first line of it will contain the determinant.
<code>Format()</code>	If <code>Format(S)</code> or <code>Format(s)</code> is given, the elements of the out matrix are written in the shortest possible form that preserves precision, eg -0.001350923141374722. Otherwise the elements are written in scientific notation with 16 digits in the mantissa, eg -1.350923141374722E-003. Elements are right justified.
<code>Infil()</code> <code>InfilL()</code>	Matrix infile. For multiplication it contains the left matrix.
<code>InfilR()</code>	Right matrix for multiplication.
<code>Hh</code>	Given <code>Inv</code> , the inversion is made with Householders method. It might be preferable for some dense matrices.
<code>Inv</code>	Inversion to be performed. The Gauss-Jordan method is used, unless parameters <code>HH</code> , <code>LU</code> or <code>Sympos</code> are given. If the matrix is singular, Rapp exits with an error message.
<code>Lu</code>	Given <code>Inv</code> , the inversion is made with LU decomposition. A determinant can now be computed, from 2017-10. If <code>Sympos</code> is given, Rapp tests for symmetric positive-definiteness and gives a messages if this is not the case. The matrix is inverted anyway if it is possible.
<code>Mul</code>	Multiplication to be performed.
<code>Sympos</code>	Given <code>Inv</code> and that <code>Lu</code> was not given, the inversion is made by

Choleski's method. Works if the matrix is symmetric positive-definite, such as Fisher's information matrix I. If Rapp finds that this is not the case, the Gauss-Jordan method is tried after a message is given.

Sympostest Tests for symmetric positive-definiteness and writes the answer in utfil. If symmetric positive-definiteness holds and detfil was given, the determinant is written there. This takes shorter time than performing an inversion with keywords Lu Sympos or only Sympos.

Trp Transpose to be performed.

Utfil() The out matrix is written here. Blank delimited elements.

Assume that the matrix is symmetric positive-definite and you want the fastest method. Use Choleski's method if the number of rows is at most 1230, otherwise LU decomposition without parameter Sympos. If you are not sure of symmetric positive-definiteness, besides Sympostest you can use LU decomposition with parameter Sympos if the number of rows is more than 1380, otherwise Choleski's method with parameter Sympos without LU. The numbers 1380 and 1230 should maybe be some other numbers depending on the matrix.

For a complete programming package with matrix calculus, see the next Proc Mbasic.

Proc Mbasic

Syntax

```
[N] Proc Mbasic
statements
Endproc
```

Proc Init needs not be included. An N as first non-blank character in the file suppresses printout of begin- and end-times.

Comments in the code are written as in Rapp elsewhere, ie with // last on a line or enclosed in /* and */.

Way of running: In addition to running Rapp with a file as above, run Rapp with a file with extent .Mba that does not include Proc Mbasic and Endproc. Then Rapp will understand from the extent that those first and last lines have to be included. Eg *Rapp Name1.Mba*. Giving the command *Rapp Name1* gives priority to Name1.Rpp, but if that file does not exist, then *Rapp Name1* will run Name1.Mba.

Also, a file is run as an Mba-program, if its first line has the first word digits.

Line numbers given in error messages start with 1 for the first line of the proc (or of the file for extent .Mba), not with the line number of the Rapp program if there are other procs preceding Proc Mbasic.

See also the info on the special editor for Mbasic and Rapp in general given at the beginning of this manual. Found in Rappmenus / Text editor for Mbasic and Rapp.

Example 1:

```
N Proc Mbasic
double i1 i2
mouble m1 m2 Ma[3][3]
m1 = 0 m2 = 1
Ma = { 1.23 4.56 7.89 2.34 5.67 8.90 3.45 6.78 9.01 }
/* Accomplishes the same as:
   Ma[1][1] = 1.23 Ma[1][2] = 4.56 Ma[1][3] = 7.89
   Ma[2][1] = 2.34 Ma[2][2] = 5.67 Ma[2][3] = 8.90
   Ma[3][1] = 3.45 Ma[3][2] = 6.78 Ma[3][3] = 9.01 */
For i1 = 1 to 100 by 3
  m1 = m1 + 1
  m2 = m2*(m1+4)
  if m2 > 1234567890123456789012345678901234567890 then
    print 'm2 too large.'
    break
  endif
Next i1
```

```

print m1\n
print to 'C:\Rapp\Txt\moutput.txt' // print statements directed to file.
m2 = log(Ma[3][1]) // log(3.45)
print m2\n // prints +1.23837423104326838876677449E+0

m2 = exp(1E19) print m2 \n
m1 = probnorm(0.975)
print m1 \n
m2 = probnin(m1)
print 'm2 =',%1.3 m2 \n // prints m2 = 0.975
print to
close 'C:\Rapp\Txt\moutput.txt'
m1 = probnin(0.975)
print 'm1 =',%1.9 m1 \n // prints m1 = 1.959963985
Read m1 // Reads from keyboard. If preceded by Read from file, the input is from file.
m2 = probnin(m1)
print 'm2 =',%1.3 m2 \n
i1 = 0 i2 = 20
do
  i1 = i1+1
  i2 = i2 - 3
  if 10 <= i1 then break
  elseif i1 = 5 6 7 8 or i2 < 4 then i1 = i1 + 3
  else continue endif
  i1 = i1 + 2
enddo
print i1 \n
Endproc

```

That example does nothing useful. The following example does.

Example 2. File with extent .Mba:

```

// Computes Eulers constant 0.5772156649015... . Computation time
// increases rapidly with n, but n <= 15 will not take too long.
// Due to rounding errors the last few (about six) digits will be wrong.
// If you want k correct digits, set digits to k + 9 or so.
digits 306 // Maximum with Rapp - larger maximum with Rapp1008 etc.
double m n
mouble Hm One Sum Sumold Two Two_raised_to_n e_raised_to_Two_raised_to_n m01
Read n // The computation will have a remainder term  $O(1/(2^{**n} * \exp(2^{**n})))$ .
One = 1
Two = 2
Sum = 1
Hm = 1
m01 = 1
Two_raised_to_n = Two**n
e_raised_to_Two_raised_to_n = exp(Two_raised_to_n)
For m=1 to 9999999999999999 // Infinite loop interrupted by condition below.
  // Mouble variable One is needed for mouble precision.
  Hm = Hm + One/(m+1) // Harmonic series 1 + 1/2 + 1/3 + 1/4 + 1/5 + ...
  m01 = m01*Two_raised_to_n/(m+1)
  Sum = Sum + m01*Hm
  if Sum = Sumold then break endif // When the addition of m01*Hm is too small.
  Sumold = Sum
Next m
// log(Two) is needed since log(2) will only yield double precision.
Sum = Sum*Two_raised_to_n/e_raised_to_Two_raised_to_n - n*log(Two)
print "Approximate value of Euler's constant is:"\n,%1.300 Sum \n
print to 'Euler.Txt'
print %1.300 Sum
print to

```

Here is an Mbasic program that computes the Gauss integration constants for mouble, and a sample integration program. See under "Built-in constants" later.

Example 3:

```

N Proc Mbasic
// Computes Gauss x- and w-values by simple interval halving. See Abramowitz & Stegun
// p. 887. Epsilon cannot be too small. The last one or two digits of the results are
// generally in error, so choose somewhat more digits than required for the results.

```

```

digits 72
double i signleft signmidl
mouble a b epsilon ten x x02 x04 x06 x08 x10 y z Gaussxm[10] Gausswm[10]

ten = 10

For i = 6 to 10
  a = Gaussxd[i]*0.9999
  b = Gaussxd[i]*1.0001
  epsilon = a*ten**(1-digitsnum)
  x = a Gosub P10 if y > 0 then signleft = 1 endif else signleft = -1 endif
  Do
    x = (a+b)/2 Gosub P10 if y > 0 then signmidl = 1 endif else signmidl = -1 endif
    if signmidl = signleft then a = x endif else b = x endif
    if b - a < epsilon then break endif
  Endo
  x = (a+b)/2
  Gaussxm[i] = x
  Gosub P10w
  Gausswm[i] = y
Next i

Gaussxm[1] = - Gaussxm[10]
Gaussxm[2] = - Gaussxm[ 9]
Gaussxm[3] = - Gaussxm[ 8]
Gaussxm[4] = - Gaussxm[ 7]
Gaussxm[5] = - Gaussxm[ 6]

Gausswm[1] = Gausswm[10]
Gausswm[2] = Gausswm[ 9]
Gausswm[3] = Gausswm[ 8]
Gausswm[4] = Gausswm[ 7]
Gausswm[5] = Gausswm[ 6]

print to 'Gaussxw.Txt'
print Gaussxm \n,Gausswm \n
Close 'Gaussxw.Txt'

Stop

P10:
  x02=x*x x04=x02*x02 x06=x02*x04 x08=x04*x04 x10=x04*x06
  y = -252 + 13860*x02 - 120120*x04 + 360360*x06 - 437580*x08 + 184756*x10
  y = y/1024
Return

P10w:
  x02=x*x x04=x02*x02 x06=x02*x04 x08=x04*x04
  y = 27720 - 480480*x02 + 2162160*x04 - 3500640*x06 + 1847560*x08
  y = x*y/1024
  y = 2/((1-x02)*y*y)
Return
Endproc

```

Example 4:

```

N Proc Mbasic
// Uses Gauss x- and w-values for numerical integration. See Abramowitz & Stegun p. 887.
digits 63
double i j n
mouble a0 a b b0 bmaB2 bpaB2 h sqr2pi_inv Result x y Gaussxm[10] Gausswm[10]

sqr2pi_inv = 1/sqrt(2*Pim)
Read from 'Gaussxw.Txt'
Read Gaussxm,Gausswm
Close 'Gaussxw.Txt'
Read from
Read a0,b0, n // n is the number of intervals. The larger n the more exact.
h = (b0-a0)/n

For j = 1 to n

```

```

a = a0 + h*(j-1)
b = a0 + h*j
bmaB2 = (b-a)/2    bpaB2 = (b+a)/2
For i = 1 to 10
  x = bmaB2*Gaussxm[i] + bpaB2
  Gosub FUNC
  Result = Result + Gausswm[i]*y
Next i
Next j

```

```

Result = Result*h/2
print 'Result from',%9.5 a0,'to',%9.5 b0,'is',nb Result,'.' \n
print 'Comparison with exact value:      ',probnorm(b0) - probnorm(a0) \n

```

Stop

FUNC: // You can put the function computation in an include file and use Rapp cha.

```

y = sqr2pi_inv*exp(-x*x/2)
Return
Endproc

```

The following program performs portfolio optimization. It is very simple. I assume that programs made by commercial financial software companies are much better. See Appendix 11 for the formulas used.

Example 5:

```

N Proc Mbasic
double E1 alfa1 alfa2
  F_4 G_4 H_4 H1_4[1][4] H2_4[1][4] V_4[4][4] Vinv_4[4][4]
  c_4[4][1] e_4[4] etr_4[1][4] m01_4 mu_4[4] mutr_4[1][4]
  F_3 G_3 H_3 H1_3[1][3] H2_3[1][3] V_3[3][3] Vinv_3[3][3]
  c_3[3][1] e_3[3] etr_3[1][3] m01_3 mu_3[3] mutr_3[1][3]

e_4 = { 1 1 1 1 }
mu_4 = { 1.100 1.100 1.175 0.850 }
V_4 = {
  0.030000 0.030000 0.030000 -0.030000
  0.030000 0.060000 0.030000 -0.030000
  0.030000 0.030000 0.076875 -0.030000
  0.030000 -0.030000 -0.030000 0.037500
}
etr_4 = trp(e_4)
mutr_4 = trp(mu_4)
Vinv_4 = invsp(V_4)
H1_4 = etr_4*Vinv_4
H2_4 = mutr_4*Vinv_4
F_4 = H1_4*e_4
G_4 = H1_4*mu_4
H_4 = H2_4*mu_4
m01_4 = G_4*G_4 - F_4*H_4

e_3 = { 1 1 1 }
mu_3 = { 1.100 1.100 1.175 }
V_3 = {
  0.030000 0.030000 0.030000
  0.030000 0.060000 0.030000
  0.030000 0.030000 0.076875
}
etr_3 = trp(e_3)
mutr_3 = trp(mu_3)
Vinv_3 = invsp(V_3)
H1_3 = etr_3*Vinv_3
H2_3 = mutr_3*Vinv_3
F_3 = H1_3*e_3
G_3 = H1_3*mu_3
H_3 = H2_3*mu_3
m01_3 = G_3*G_3 - F_3*H_3

print to 'Portopt.Txt'
print '          Standard' \n

```

```

print 'Expected deviation      Distribution percent' \n
print '  yield      percent      Ass1  Ass2  Ass3  Ass4' \n
for E1 = 1.005 to 1.175 by 0.005
  alfa1 = (G_4*E1 - H_4)/m01_4
  alfa2 = (G_4 - E1*F_4)/m01_4
  c_4 = Vinv_4*(alfa1*e_4+alfa2*mu_4)
  if not(c_4[1][1] >= 0 and c_4[2][1] >= 0 and c_4[3][1] >= 0 and c_4[4][1] >= 0) then
    // Asset no 4 is excluded. The one to exclude was simply determined by running
    // first without these conditional statements, which gave negative c_4[4][1].
    alfa1 = (G_3*E1 - H_3)/m01_3
    alfa2 = (G_3 - E1*F_3)/m01_3
    c_3 = Vinv_3*(alfa1*e_3+alfa2*mu_3)
    c_4 = c_3
  endif
  print %8.3 E1, %10.2 100*sqrt(trp(c_4)*V_4*c_4), ' ', %6.2 100*trp(c_4)
next E1
print to
Endproc

```

Overview

Mbasic is a mathematical language within the language Rapp. The M in Mbasic stands for Multiple precision, ie you choose the number of correct decimal digits up to 306 for a floating data type called mouble with Rapp.Exe. (I made such a language in 1981 with the technology then available.) The last digit in a mouble result of an operation is correctly rounded. Internally Rapp uses arithmetic with some more digits than the chosen number, in order to compensate for rounding errors. This holds also if you choose 306 digits. The common double data type with an 8-byte storage is also available.

See the section "Location of program and how to write and run Rapp code" for other variants of Rapp with other maximums than 306. Computing speed is not much affected by choosing a larger variant than needed for a given precision, but the number of array elements you can store with a given RAM is of course affected.

Also M stands for Matrix Basic, since it has powerful matrix operations.

All variables and arrays are visible between Proc Mbasic and Endproc regardless of their place of declaration. There are no classes such as global, static and automatic as in C, although all such classes are used in the C program interpreting Mbasic code. Mbasic is simple in this respect, which however does not hinder advanced numerical calculations.

You can largely use the same syntax as in classical Basic (such as For / Next, Gsub etc). Some C syntax is also employed. Conditional statements if / elsif / else can be stated as complex as you like. I have implemented all elementary and inverse circular functions and some more. Common matrix operations, including four methods of inversion, are also implemented.

The purpose of Mbasic is to furnish mathematical capabilities. It has no string handling facilities, except for printing texts within single or double quotes and for executing system command strings. For string manipulation while transforming and matching text files, use Proc Data and Proc Match.

Mbasic is case independent, like the rest of Rapp. It does not use semicolons or other delimiters between statements, only one or more blanks. A statement can span over arbitrarily many lines of code. Several statements can be written on a line; only ease of code understanding motivates linebreaks.

The Mbasic code is given between Proc Mbasic and Endproc. A variable in the code cannot be named Proc, Endproc or Include. Keywords for statements like break are not either allowed as names of variables.

Below a scalar is a real-valued constant, variable, array element or function return value. A constant is written in the standard way, with decimal point or in scientific E-notation. Both upper- and lower-case E is accepted.

An item in a syntax description within [and] is optional. (Might possibly cause confusion since [and] are used in arrays, but the context should make clear what is intended.) An | separating items denotes choice. These conventions are standard.

Description in alphabetical order

Arithmetical operations for arrays
 Arithmetical operations for matrices
 Arithmetical operations for scalars
 Break statement
 Built-in constants, variables and arrays
 Close statement
 Complex variables, arrays, matrices and functions
 Conditional statements
 Continue statement
 Data types
 Declarations
 Digits statement
 Do loops
 Error messages
 Exit statement
 For/Next loops
 Free statement
 Functions of one scalar
 Functions of two scalars
 Functions of three scalars
 Gosub statement
 Goto statement
 Helpmatricesfree statement
 Print to statement
 Printing arrays
 Printing scalars
 Printing text
 Pseudo-random numbers
 Read from statement
 Read line statement
 Reading data from a file
 Reading data from keyboard
 Redim statement
 Return statement
 Sort statement
 Sortd statement
 Stop statement
 System statement

Arithmetical operations for arrays

These are defined for arrays with arbitrary dimensions up to 30. See Declarations later. An array with at most two indices (at most dimension 2) is called a matrix. More operations are defined for these. See next section.

The operations, their notations and restrictions are standard.

As far as the operations below are concerned, the first index for all dimensions is considered to be 1 regardless of the declaration, as described under Declarations below. For example A[3:5] is considered to be the same as A[1:3] and A[3]. But for assignments of scalars to array elements or vice versa, the declarations matters. Eg for A declared A[3:5][-2:4] and B declared B[3][7], the statement B[1][1] = A[3][-2] transfers the upper left corner element of A to the corresponding element of B. Assignment type IDN, defined for matrices below, will however be performed with all first indices considered to be 1. Letting the first index be 0, eg with declaration B[0:2][0:6], might sometimes be convenient.

The indices are scalars; variables, constants or array elements.

An array declared A[n] is regarded as the same as A[n][1] and an array declared A[n][1] is regarded as the same as A[n] for the purpose of matrix calculus. However, printing an array (see Printing arrays later) declared A[n] will print the elements on the same line, while an array declared A[n][1] will have the elements printed on separate lines.

Assignment of scalars to array elements

1. Elements one by one. Eg A[2][3]=1.23 A[2][3]=x1 A[2][3]=D[7][3]
2. In the form A = { a11 a12 a13 a21 a22 a23 }, where the elements within { and } are constants (not variables) listed in order first index varying most slowly, etc. For matrices this is called row-major order. The elements are separated by any

- number of blanks. The braces need not be followed or preceded by a blank.
3. A = CON fills the array with 1's.
 4. A = ZER fills the array with 0's. (Initially all arrays are filled with zeroes.)

Operation	Example
Assignment	B = A
Addition	C = A + B
Subtraction	C = A - B
Negation	B = -A
Multiplication with scalar	B = A*x1 or B = x1*A

The dimensions and index limits need not be the same. The array of all involved arrays with the smallest number n of elements decides which elements go to the out array. Elements n+1 etc in the out array are set to 0, if the out array has more than n elements. The statement B = A with declarations A[2][3][4] and B[2][4][4] might not put the elements in B where you want them, though.

My intention is that array operations can be combined in ANY manner. Should you find an example where this is not true, please report it to me: stig.ingvar.rosenlund@gmail.com.

Ordinary priority rules for operations hold. These can be overridden by parentheses.
Example:

```
A=(-B-C+D)*1.23
```

Double and mouble arrays can be freely mixed.

The operations are defined also for complex arrays. Arrays can be mixed real and complex.

Arithmetical operations for matrices

These are defined for arrays with at most two indices. Such an array is called a matrix. All operations for arrays described above can be used for matrices. Below we describe the extra operations for matrices.

The operations, their notations and restrictions are standard, except for the notation for inversions. There are four inversion operations. The differences between these are explained after the operations table.

In this description Lim1(A) denotes the number of allowed integer indices for the first dimension. The first index is 1. Lim2(A) is the number for the second dimension. For example, a matrix declared as double A[5] with one dimension has Lim1(A) = 5, where A[1], ... , A[5] can be accessed as simple variables. A matrix declared as mouble B[4][9] with two dimensions has Lim1(B) = 4 and Lim2(B) = 9, where B[1][1], ... , B[4][9] can be accessed as simple variables.

Assignment of scalars to matrix elements

A = IDN fills the matrix with 1 in the diagonal and 0 elsewhere; identity matrix.

Operation	Example	Restrictions
Multiplication	C = A*B	Lim2(A) = Lim1(B), Lim1(C) = Lim1(A), Lim2(C) = Lim2(B)
Transpose	A = trp(B)	Lim1(A) = Lim2(B), Lim2(A) = Lim1(B)
Inversion 1	A = invgj(B)	Lim1(A) = Lim2(A) = Lim1(B) = Lim2(B), ie square matrices
Inversion 2	A = invsp(B)	same as Inversion 1
Inversion 3	A = invl1(B)	same as Inversion 1
Inversion 4	A = invl2(B)	same as Inversion 1
Inversion 5	A = invhh(B)	same as Inversion 1
Symptest	t = symptest(B)	B a square matrix

The return value t of symptest() is a scalar. The function tests if B is symmetric positive-definite. The return is 0 if that is the case, 1 otherwise. If B is double and t is mouble, then B is converted to mouble in a temporary matrix on which the test is performed. When rounding errors present a problem, letting t be mouble and setting digits high enough can overcome the problem.

A symptest expression can be combined in arithmetics with other expressions, such as "t=1+symptest(B)". It can be used in an if/elsif-clause. Eg "if symptest(A+B) = 0 then ...". The data type of the expression will be mouble if at least one of the operands in symptest() is mouble.

A determinant is computed after each inversion and, for real matrices, is found in the variable `d_determinant` if the left side matrix is double, and as `m_determinant` if the left side matrix is mouble. For complex matrices (see below) it is found in `d_determinantc` = `d_determinantc[1] + i*d_determinantc[2]`, and `m_determinantc`, respectively.

A determinant is also computed with `sympostest()`, provided the return value is 0. If both `B` and `t` is double, the result is in `d_determinant`. If at least one of `B` and `t` is mouble, the result is in `m_determinant`.

Properties of inversion operations

Inversion 1 `Invgj()` Gauss-Jordan elimination.
 Inversion 2 `Invsp()` Choleski inversion for a symmetric positive-definite matrix.
 Inversion 3 `Invll()` LU decomposition. Right side matrix `B` is preserved, unless `A = B`.
 Inversion 4 `Invl2()` LU decomposition. Right side matrix `B` is changed by the operation.
 Inversion 5 `Invhj()` Householder method.

Note. It can happen that `Invgj()` gives an error message that the determinant is 0, ie the matrix is singular, but `Invll()` goes through.

`Invgj()`, `Invsp()` and `Invhj()` are not available for complex matrices.

My intention is that matrix operations can be combined in ANY manner. Should you find an example where this is not true, please report it to me: stig.ingvar.rosenlund@gmail.com

Ordinary priority rules for operations hold. These can be overridden by parentheses.
 Example:

```
A=(-B*C*(Invsp(A) - Trp(D)))*1.23
```

A scalar variable can be given on the left side of a matrix expression, if the right side has dimension `[1]` or `[1][1]`. Such matrix expressions can also be used as scalars in some instances, such as functions or one or two variables where these variables are matrix expressions with dimension `[1]` or `[1][1]`. But I have not managed to cover all cases of use of matrix expressions as scalars. If such use is not possible, Rapp will give an error message, so that you can modify the program to use an intermediate scalar.

Apart from some instances, a scalar as left side and a matrix expression with dimension `[1]` or `[1][1]` as right side must stand by itself and cannot be combined in arithmetics with other expressions. Eg `"t=1+B"` is not allowed, but `"t=B t=t+1"` goes through, if `B` was declared `[1]` or `[1][1]`. Such an expression cannot either be used in an `if/elsif`-clause.

Double and mouble matrices can be freely mixed, if the dimension requirements are met.

The operations are defined also for complex matrices, with the exception of `Invgj()`, `Invsp()` and `Sympostest()`. Matrices must be either all real or all complex.

Arithmetical operators for scalars

The basic operations addition, subtraction, multiplication and division, their notations and priorities are standard. Priorities can be overridden by parentheses. Optional number of blanks between operators. Up to 999 operations can be combined in a statement.

Exponentiation is made with `**`, eg `2**5` is `32 = 25`. It has priority. It can also be accomplished with the function `pow( , )`, see the section on functions of two scalars.

As in Proc Data, the absolute value of `x` can be obtained by `|x|` and the floor value by `[x]`. Thus `[ ]` has several uses, but Rapp can tell which one is intended. You can also accomplish `| |` with the function `abs()` and `[ ]` with the function `floor()`.

Division by zero will give zero and not cause a crash. Exponentiation `x**y = xy` gives the right answer for any `x`, if `y` is an integer in `(... ,-2,-1,0,1,2 ...)`. For non-integer `y` and negative `x` the result is strictly undefined, but 1 will be returned.

Example:

```
m4 = |x1| - ([m2**63] - 3.14*A[1][1]**x2/m3)
which is the same as
m4 = abs(x1) - (floor(pow(m2,63)) - 3.14*pow(A[1][1],x2)/m3)
```

Break statement

Causes the execution of the For- or Do-loop to end, ie the first statement following the loop is executed and so on.

Built-in constants, variables and arrays

The number $\pi = 3.14159265358979323846\dots$ is available as the double variable `Pid` and the mouble variable `Pim`. The double variable `digitsnum` holds the number of digits. It is a multiple of 9. The abscissas `xi` and weights `wi` for $n = 10$ in Abramowitz & Stegun (25.4.29) p. 887, for Gaussian type integration, are available as the double arrays `Gaussxd[10]` and `Gausswd[10]`. See Example 3 and 4 above for how to make and use mouble variants. Results from the function `bessjy()` are in the double variables `d_bessJ`, `d_bessY`, `d_bessJp`, `d_bessYp` and the mouble variables `m_bessJ`, `m_bessY`, `m_bessJp`, `m_bessYp`. Results from `bessik()` are given in analogous variables. Determinants of matrices are given in `d_determinant` and `m_determinant` at matrix inversions. The complex variables `cuberootd1`, `cuberootd2`, `cuberootd3`, `cuberootm1`, `cuberootm2`, `cuberootm3` will hold the results of the function `cube()`, see below.

Close statement

Syntax

```
Close 'file'
```

If the file closed is a print file, then printing to it again with 'print to ' will overwrite its content. If it is a read file, then reading from it again with 'read from ' will read it from the beginning.

If you want to copy the content of a print file to another file, then close it first.

Complex variables, arrays, matrices and functions

A complex variable or array of type double or mouble (see Data types and Declarations later) is declared with the keyword `complex` after it. For example:

```
mouble C complex D[66][77] complex
```

You can mostly treat these as real arrays with [2] appended, as if declared, using the example:

```
mouble C[2] D[66][77][2].
```

The real part is in the first number and the imaginary part is in the second number of the last dimension. Eg $C = C[1] + iC[2]$ and $D[33][44] = D[33][44][1] + iD[33][44][2]$, where i is the imaginary unit such that $i^2 = -1$. The last [1] can be written `.re` and the last [2] can be written `.im`. Eg $D[33][44].re = 3.14$ is the same as $D[33][44][1] = 3.14$.

The difference, between writing `complex` or [2] after a variable or array, is in how they are treated in matrix operations.

The operations `+` `-` `*` `/` and `**` are implemented for complex variables and follow the standard rules. Eg

```
mouble Y complex    double X[3] complex
X[2] = {1 -3}    Y = {1-i*3}    Y = X[2]**Y
```

Assignment to a complex variable or array element with complex constants within { and } can be in the form of two real numbers, like `X[2]` above, or in the form of a word `x+i*y` or `x-i*y` without embedded blanks, where `y` is non-negative and `i` is in lower case, like `Y` above. Or as `Complexd()` or `Complexm()`. (If the array is real, two real numbers are assigned. See 'Reading data from a file' below. Such constants can only be used this way. They cannot be used generally as constants in expressions with several constants. For example ~~`Y = {1-i*3}*(1+i*3)`~~ does not work. For this, use this form: `Y = Complexm(1 - 3)*Complexm(1 3)`.

All operations defined for arrays that are not matrix operations can be performed on complex variables or arrays as if the were real arrays with [2] declared last, except for the `CON` assignment which sets every complex member to $1 + i*0$ (not $1 + i$). Eg

However, only one complex array member (real and imaginary part) can be assigned within braces { and }, unless the whole array is filled as previously described.

The operations + and -, and multiplication with a real scalar or with a complex number, are defined for a complex array of any dimension. Eg $X = 3 \cdot X$ or $X = Y \cdot X$.

All operations defined for real matrices can be performed on complex matrices, with the exception that only LU decomposition inversion is implemented for complex matrices, as of now. That is, `invl1(A)` and `invl2(A)` are implemented, but not `invgj(A)` and `invsp(A)`, for a matrix A declared `double|double A[r][r] complex`. If you would have use for `invgj()`, send me a request.

`A = IDN` fills the matrix with $1 + i \cdot 0$ in the diagonal and 0 elsewhere; identity matrix.

The data types `double` and `double` can be freely mixed. In matrix operations in addition to those defined for all real arrays, all matrices must be either all real or all complex.

The polar representation $z = re^{i\theta}$ of `xy2pol()` uses $-\pi < \theta \leq \pi$, affecting `log()`. All arithmetics and functions in `Mbasic`, except `pol2xy()`, assume the ordinary form $z = x + i \cdot y$.

Functions of complex variables (see descriptions later for real arguments and return)

<code>abs</code>	Absolute value is put in the real part and the imaginary part is set to 0.
<code>asin</code>	Formula by Abramowitz & Stegun, (4.4.37) with $k = 0$, but with change of sign of output imaginary part, if imaginary part of input is negative. See Appendix 10.
<code>asinh</code>	<code>asinh(z) = -i * asin(i * z)</code>
<code>atan</code>	Formula by Abramowitz & Stegun, (4.4.39) with $k = 0$.
<code>atanh</code>	<code>atanh(z) = -i * atan(i * z)</code>
<code>ceil</code>	Ceiling is applied to both parts.
<code>conjugate</code>	Input $x + i \cdot y$, output $x - i \cdot y$.
<code>cos</code>	
<code>cosh</code>	
<code>cot</code>	
<code>coth</code>	
<code>exp</code>	
<code>floor</code>	Floor is applied to both parts.
<code>log</code>	
<code>pol2xy</code>	Input is number in polar form, output is number in ordinary form.
<code>pow</code>	<code>pow(z1, z2) = z1 ** z2 = exp(z2 * log(z1))</code> .
<code>sin</code>	
<code>sinh</code>	
<code>sqrt, sqr</code>	<code>sqrt(z)</code> is the same as <code>pow(z, 0.5 + i * 0)</code> .
<code>tan</code>	
<code>tanh</code>	
<code>xy2pol</code>	Input is a number in ordinary form, output is a number in polar form with r in the real part and θ in the imaginary part.

Some functions not listed above can be obtained from the ones listed. Eg

$\text{acos}(z) = \pi/2 - \text{asin}(z)$, $\text{acot}(z) = \text{atan}(1/z)$ for $z \neq 0$ and $\text{acot}(0) = \pi/2$.

The function `pow(z1, z2)` computes faster and more accurately when z_2 takes one of the special values -1.5, -1, -0.5, 0, 0.5, 1, 1.5. Eg `pow(z, -1.5 + i * 0) = 1 / (z * sqrt(z))`.

Conditional statements

Syntax

```
if condition1 then
    statement(s)
[endif]
elseif condition2 then
    statement(s)
[endif]
elseif condition3 then
    statement(s)
[endif]
else
    statement(s)
endif
```

None or any number of elsif blocks can be given. The else statement is optional. For convenience you can write elseif instead of elsif (but not else if, since that starts nested conditional statements). Also end if instead of endif. Note that else is concluded by endif. Statements under an if or elsif must be concluded by endif if no elsif or else comes directly after, but is optional otherwise, since in the latter case Rapp can understand what is meant. See the [endif]:s above. Nesting conditional statements within conditional statements is allowed.

The condition operators, with their synonyms, are the same as in Proc Data / Urval().

Relational operators	Symbol
equality	= == EQ
inequality	^= != <> >< NE NQ
less than:	< LT
greater than	> GT
less than or equal to	<= LE LQ
greater than or equal to	>= GE GQ
Logical operators	Symbol
conjunction	AND &
disjunction	OR !
negation	NOT ^ !

The exclamation mark ! has several meanings, but Rapp can tell which one is intended.

Several values can be stated after equality and inequality operators.

Any number of conditions can be combined with the logical operators above, with parentheses to override normal priorities. Scalar variables and constants can appear on both the left and right side of a relational operator. (As opposed to Proc Data / Urval(), where the left side of a relational operator must be a variable and the right side cannot contain a variable.)

As you can see, conditional statements can be written as in Visual Basic.

Continue statement

Causes the execution of the For- or Do-loop to go to its beginning, thus bypassing the remaining statements before Next or Endo.

Data types

There are two data types, double and mouble. Both are floating point types.

Below ** means "raised to".

Double is the same as double in most C-compilers and also as the SAS Rb8. format, namely with an 8-byte storage. The storage is binary. Translating the number of binary digits in the mantissa to decimal ones gives approximately 15.8 digits.

Approximate range for a double variable d is this. It depends on the implementation in the C math library. This one is for Microsoft C.

$$1.1125369292536E-308 \leq |d| \leq 1.797670120615949E+308$$

An operation with a double result that would be smaller in absolute value than the left side will give 0 as result. If the result would be larger in absolute value than the right side, then ±(the right side) will be given as result.

Mouble is specific to Mbasic. In Rapp.Exe a mouble occupies 152 bytes of storage regardless of the number of chosen digits, see section Digits later. The chosen number of digits affects the speed of execution. The number will be at least 27, ie 71 % more than with double. This is the default. And it will be at most 306. In Rappmultiprecision.zip there are additional variants of Rapp for other maxvalues for digits. The storage is decimal. The exponent is stored in 8 bytes. These are the limits for a mouble variable m.

$$10^{*(-9223372036854775800)} \leq |m| < 10^{*9223372036854775816}$$

Maximal value m such that exp(m) and exp(-m) are computed correctly is
21237598959199934491.4100942 = 2.12375989591999344914100942*10**19

There are no integer data types; double variables have to be used for integer-valued variables. The loss in speed from this omission is negligible, since Microsoft C computes doubles very fast.

A constant, eg 3.14159, is stored in a double if it has ≤ 15 digits in the mantissa (ie not including digits in an exponent) and has absolute value inside the range for double stated above. Otherwise it is stored in a mouble. This will mostly store the constant with its exact value, but for many non-integer values, such as 0.009, there will be a difference due to the binary representation of a double. A statement `m01 = 0.009` will give m01 the value 0.008999999999999999, since 0.009 is first passed to a double and then to a mouble. Avoid this by writing ≥ 16 digits in the number, apart from an exponent. Writing `m01 = 0.0090000000000000`, `m01 = 00.0090000000000000`, `m01 = 00000000000000.009` will give m01 the value 0.009.

For an expression where the left side is mouble and the right side contains double variables, including constants assigned to double, the following holds. In a simple expression with only one of the `+` `-` `*` `/` operators or only a single function computation, the double variables will be converted to mouble before the operation. With several such operators this is not guaranteed, so make sure that you do not lose precision by operations on doubles that cannot have an exact result within double. Examples:

```
m01 = 1/3           will give digitsnum correct digits
m01 = 10*(1/3)      will not give digitsnum correct digits
m01 = 10*(1/3.0000000000000000) will give digitsnum correct digits
m01 = log(2)        will give digitsnum correct digits
m01 = 10*log(2)      will not give digitsnum correct digits
m01 = 10*log(00000000000000002) will give digitsnum correct digits
```

Declarations

A statement beginning with double declares double variables and arrays. The variable names are blank delimited. A name can have several thousand of characters. It shall start with a letter and contain only letters and digits. An array is declared with its index limits within `[` and `]`. Eg `[2:7]` means that you must have $2 \leq \text{index} \leq 7$. If only one limit is given, then it is the upper limit and the lower limit will be 1. Eg `[3]` means `[1:3]`. Th limits can be given in E-notation, eg `[2:1e2]`. The dimension can be up to 30, ie there can be up to 30 indices. Eg `A[2][3][4][2][8][5]` defines the dimension as 6 with 1920 elements. An element is accessed with eg `A[2][1][3][2][6][3]`.

A statement beginning with mouble declares mouble variables and arrays in the same way.

The dimension limits can be given as variables. However, with no such declarations and with no Redim-statements (see later) more errors in your program can be detected at an early stage (corresponding to compile stage in C) instead of at a time when the program has been running for a while.

If an index value falls outside the limits range, Rapp interrupts with an error message. (As opposed to C programs where speed is more important than informative error messages.)

There can be any number of declaration statements placed anywhere in the program, except that declarations with variable dimension limits must be placed after statements where those variables are assigned to valid numbers. The variables and arrays declared are available also before the point where they are declared, except that variable dimension limits are not in effect before the variables are assigned; before that point the dimension limits are 1:1. In the example below only `Mc[1][1]` is available before `i1` and `i2` are set.

An array is allocated the memory space it needs when it is first used. If the first use is as input, of the whole array or only one element, all elements will be 0. If the first use is as output, all elements not referenced will be 0. If it is never used, no space will be occupied. If the total memory use would be very large, consider freeing arrays when they are not needed. See the Free statement later.

Example:

```
double i1 i2
mouble m1 m2 Ma[-1:3][0:3] Mb[3][4][5]
i1 = 7 i2 = 3
mouble Mc[2:i1][i2]
```

Digits statement

Syntax

digits number of digits

Gives the number of correct digits in the mantissa of a mouble.

Example:

digits 54

At least 27 digits will be set by Rapp. It will be a multiple of 9. If you do not give a multiple of 9, the number of digits will be adjusted upwards to closest multiple that is ≤ 306 . In Rappmultiprecision.zip there are variants for other maxvalues of digits than 306. See section "Location of program and how to write and run Rapp code". If you give several digits statements, the largest one will take effect. But it is normally best to give one digits statements at the top.

Do loops

Syntax

```
Do
    statement
    statement
    ...
endo
```

The statements between do and endo are repeated indefinitely until a break, goto or exit statement is executed. You can write end do and enddo instead of endo, if that is more convenient by resembling other languages you use.

Error messages

Errors are described in messages and the faulty code line(s) listed. Split the code where the error occurs on several lines to better understand what the error is.

Exit statement

Causes the execution to end. Stop is a synonym.

For/Next loops

Syntax

```
For variable = startvalue to endvalue [by increment]
    statements
Next variable
```

See initial example. Increment will be 1 if omitted. This is as classical Basic. Variable after Next is really superfluous, but I haven't come around to admit its omission.

These examples are equivalent:

```
For i1 = 3 to 100 by 3
    statements
Next i1

i1 = 3
Do
    if i1 > 100 then break endif
    statements
    i1 = i1 + 3
Endo
```

Free statement

Syntax

```
Free array
Free helpmatrices
```

Example:

```
Free Ma
```

Execute this to free all memory occupied by an array after you no longer need it, to make more space for other arrays. You can use the array again, but all previous content was discarded.

The word `helpmatrices` after `free` has a special meaning, namely that all help matrices created for non-trivial matrix expressions are freed. Use this after a loop with convoluted matrix calculations. See also statement `Helpmatricesfree`.

All arrays will anyway be freed at termination of `Proc Mbasic`.

Functions of one scalar

Argument and return value can be both double and mouble, and the data types can be mixed.

Explanations are given for functions with non-standard notation.

Function	Explanation / comment
<code>abs</code>	
<code>asin</code>	$\arccos(x) = \pi/2 - \arcsin(x)$
<code>asinh</code>	
<code>atan</code>	$\operatorname{acot}(z) = \operatorname{atan}(1/z)$ for $z \neq 0$ and $\operatorname{acot}(0) = \pi/2$.
<code>atanh</code>	
<code>ceil</code>	smallest integer value that is $\geq$ argument
<code>ceiling</code>	synonym to <code>ceil</code>
<code>cos</code>	
<code>cosh</code>	
<code>cot</code>	$1/\tan$ - somewhat unnecessary but saves you a division
<code>coth</code>	$1/\tanh$
<code>erf</code>	
<code>erfc</code>	$1 - \operatorname{erf}$ **
<code>erfin</code>	inverse of <code>erf</code>
<code>exp</code>	
<code>floor</code>	largest integer value that is $\leq$ argument
<code>gam</code>	gamma function $\Gamma(x)$ *
<code>log</code>	negative argument gives result 0
<code>log10</code>	negative argument gives result 0
<code>loggam</code>	logarithm (base e) of gamma function $\Gamma(x)$ *
<code>probnin</code>	inverse Φ^{-1} of normal distribution function <code>probnorm</code> . Remark: Not guaranteed to give 15 correct digits if both input and output are declared double
<code>probnorm</code>	normal distribution function Φ with mean 0 and variance 1. Same Remark.
<code>probnormc</code>	$1 - \operatorname{probnorm}$ **
<code>probnormmp5</code>	$\operatorname{probnorm} - 0.5$ **
<code>psi</code>	derivative of <code>loggam</code> ; $\psi(x) = \Gamma'(x)/\Gamma(x)$ *
<code>readline(x)</code>	word number x in string <code>readline</code>
<code>sgn</code>	sign: $\operatorname{sgn}(x) = -1$ if $x < 0$, 0 if $x = 0$, 1 if $x > 0$
<code>sin</code>	
<code>sinbyx</code>	$\sin(x)/x$; the difference is a slight gain in accuracy and value 1 for $x = 0$
<code>sinh</code>	
<code>sinhbyx</code>	$\sinh(x)/x$; same difference as for <code>sinbyx</code>
<code>sqr</code>	square root
<code>sqrt</code>	synonym to <code>sqr</code>
<code>tan</code>	
<code>tanh</code>	

* Remark on `gam(x)`, `loggam(x)` and `psi(x)` for mouble return value. Return has maximal precision 306 digits, since the functions become increasingly difficult to implement as precision increases, unless x is a positive multiple of 0.5 (which is easy to implement). For $x < 0$ and $\sin(x) < 0$, $\log(|x|)$ is given for `loggam(x)`, while `gam(x)` will be negative as it should. If the absolute value of the return value for `gam(x)` would be larger than $10^{9223372036854775800}$, that number or its negative is returned. For $0 < x < 5.333669791026345338999251\text{E}17$ (approximately), `gam(x)` will be returned OK. If `loggam(x)` and `gam(x)` are not defined, eg for $x = 0$, 0 is returned. For $x = n + k/10$ ($n = 0, \dots, 100$; $k = 1, \dots, 10$), $\Gamma(x)$ are computed once and stored for rapid returns. The purpose is

to facilitate simulations, where these decimal values 0.1, 0.2, ... , 0.9, 1.0, ... 100.9, 101.0 are sufficient for the study.

The function erf is standard. It holds $\text{erf}(x) = 2 \cdot \text{probnorm}(\sqrt{2} \cdot x) - 1$.

****** $Y1 = 1 - \text{probnorm}(x)$ and $Y2 = \text{probnormc}(x)$ will generally be different numbers. The precision for $Y2$ will be related to itself, but the precision of $Y1$ will be related to $\text{probnorm}(x)$. For example, with default digits 27 and $x = 15$ you get $Y1 = 0$ since $\text{probnorm}(x)$ will be $1 - \text{epsilon}$ with epsilon so small that it does not affect the 27:th digit of $\text{probnorm}(x)$. You get $Y2 = \text{epsilon} = 3.67096619931275088578608966\text{E-}51$, however. Analogously for $\text{probnorm}(x) - 0.5$ versus $\text{probnormmp5}(x)$ and $1 - \text{erf}(x)$ versus $\text{erfc}(x)$.

Functions of two scalars

Arguments and return value can be both double and mouble. The data types can be mixed. The order of the first and second argument might at places seem inconsequential, but that is the way it often is with mathematical function notation. For example $\text{bessik}(x, v)$ and $\text{gamdist}(a, x)$, where in applications x varies for fixed v and a , respectively. Trying to have a unified order will lead to other problems, such as inconsistency with notation in the literature.

Explanations are given for functions with non-standard notation.

Function	Explanation / comment
$\text{atan2}(x, y)$	$\text{atan}(y/x)$ if $x > 0$. For $x \leq 0$ see Wikipedia.
$\text{bessik}(x, v)$ $\text{bessiks}(x, v)$	Modified Bessel functions. The "Numerical Recipes in C" algorithm used. For negative x all results are set to 0. The return value $\text{bessik}(x, v)$ is 0 if the computation succeeded (albeit with possibly wrong results if x and/or v are too large). Return value is 1 if it failed, and then all results are set to 0. The results are given in built-in variables of Mbasic. If both x and v are double, the results from $\text{bessik}()$ are in

$$\begin{aligned} d_bessI &= I_v(x) & d_bessK &= K_v(x) \\ d_bessIp &= I_v'(x) & d_bessKp &= K_v'(x) \end{aligned}$$

The results from $\text{bessiks}()$ are scaled with e^x , ie

$$\begin{aligned} d_bessI &= I_v(x)e^{-x} & d_bessK &= K_v(x)e^x \\ d_bessIp &= I_v'(x)e^{-x} & d_bessKp &= K_v'(x)e^x \end{aligned}$$

If at least one of x and v is mouble, the results are given in likenamed variables with prefix $m_$ instead of $d_$, ie m_bessI etc; ≤ 306 digits.

$\text{bessjy}(x, v)$	Bessel functions of the first and second kind. Built-in functions $\text{jn}()$ and $\text{yn}()$ of C are sometimes used for integer v and both x and v double, otherwise the "Numerical Recipes in C" algorithm. Both x and v can be negative, but 0 is given as result if the function is not defined. The return value $\text{bessjy}(x, v)$ is 0 if the computation succeeded (albeit with possibly wrong results if x and/or v are too large). Return value is 1 if it failed, and then all results are set to 0. The results are given in built-in variables of Mbasic. If both x and v are double, results are in
-----------------------	---

$$\begin{aligned} d_bessJ &= J_v(x) & d_bessY &= Y_v(x) \\ d_bessJp &= J_v'(x) & d_bessYp &= Y_v'(x) \end{aligned}$$

If at least one of x and v is mouble, the results are given in likenamed variables with prefix $m_$ instead of $d_$, ie m_bessJ etc; ≤ 306 digits. Use x and v double is speed is important and accuracy is not. Often there will be only 12 correct digits in the mantissas of the double results.

<code>chi2(r,x)</code>	χ^2 distribution, see <code>gamdist()</code> below.
<code>chi2c(r,x)</code>	$1 - \text{chi2}(r,x)$; see remark for <code>probnormc()</code> .
<code>chi2inv(r,p)</code>	χ^2 distribution inverse
<code>chi2invc(r,p)</code>	<code>chi2inv(r,1-p)</code> ; not analogous to <code>betadistinvc(a,b,x)</code>
<code>gamdist(a,x)</code>	gamma distribution function of x with mean and variance a ; the χ^2 distribution with r degrees of freedom is <code>gamdist(r/2,x/2)</code> ; ≤ 306 digits unless a is a multiple of 0.5.
<code>gamdistc(a,x)</code>	$1 - \text{gamma distribution function}$; see remark for <code>probnormc()</code> .
<code>gamdistinv(a,p)</code>	gamma distribution function inverse
<code>gamdistinvc(a,p)</code>	<code>gamdistinv(a,1-p)</code> ; not analogous to <code>betadistinvc(a,b,x)</code>
<code>hypot(x,y)</code>	$\sqrt{x^2+y^2}$; a trivial function, but included in the C math library.
<code>max(x,y)</code>	
<code>min(x,y)</code>	
<code>poisson(λ,x)</code>	Poisson distribution function of x with mean λ .
<code>poissonc(λ,x)</code>	$1 - \text{poisson}(\lambda,x)$; see remark for <code>probnormc()</code> .
<code>poissoninv(λ,p)</code>	Poisson distribution function inverse; Returns $\min\{k:P(X \leq k) \geq p\}$ for X Poisson distributed (λ).
<code>pow(x,y)</code>	$\exp(\log(x)*y)$; see comment for $x**y = x^y$
<code>readline(x,y)</code>	number starting at column x for y columns of string <code>readline</code>
<code>tdist(x,v)</code>	Student's t-distribution function of x with v degrees of freedom defined for $-\infty < x < \infty$, and approximately <code>probnorm(x)</code> if v is large. The result has mouble precision if the variable receiving the return is mouble.
<code>tdistc(x,v)</code>	$1 - \text{tdist}(x,v)$; see remark for <code>probnormc()</code> .
<code>tdistinv(p,v)</code>	inverse of Student's t-distribution. Full precision per 2016-11-28.

Functions of three scalars

Arguments and return value can be both double and mouble. The data types can be mixed.

Explanations are given for functions with non-standard notation.

Function	Explanation / comment
<code>betadist(a,b,x)</code>	distribution function for the frequency function $Cx^{a-1}(1-x)^{b-1}$, where $0 \leq x \leq 1$, $a > 0$, $b > 0$. Here $C = \Gamma(a+b)/[\Gamma(a)\Gamma(b)]$. The return value is 0 if invalid values ≤ 0 are given for a and/or b . Max 306 digits unless both a and b are multiples of 0.5.
<code>betadistc(a,b,x)</code>	$1 - \text{betadist}(a,b,x)$; see remark for <code>probnormc()</code> .
<code>betadistinv(a,b,x)</code>	inverse of <code>betadist()</code>
<code>betadistinvc(a,b,x)</code>	$1 - \text{betadistinv}(a,b,x)$
<code>bindist(n,p,x)</code>	$P(N \leq x)$ where N is binomial (n,p).
<code>bindistc(n,p,x)</code>	$1 - \text{bindist}(n,p,x)$; see remark for <code>probnormc()</code> .
<code>bindistinv(n,p,z)</code>	inverse of binomial (n,p); Returns $\min\{k:P(N \leq k) \geq z\}$. Due to rounding errors $z = \text{bindist}(n,p,x)$ and $x1 = \text{bindistinv}(n,p,z)$ will not always give $x1 = x$.
<code>cube(A,B,C)</code>	solves the equation $x^3 + Ax^2 + Bx + C = 0$. The roots are given in the built-in complex variables <code>cube rootd1</code> , <code>cube rootd2</code> and <code>cube rootd3</code> if all of A , B and C are double. If at least one of them is mouble the roots are given in <code>cube rootm1</code> , <code>cube rootm2</code> and <code>cube rootm3</code> . If all roots are real they are sorted in ascending order. If two roots are complex, the first root is the real one, and the following two are the complex ones sorted in ascending order of the imaginary parts. The cases exactly zero or exactly two real roots do not exist. The imaginary part <code>cube rootd1[2]</code> will always be 0. Return value is 1 if there are three distinct real roots, 2 if there are three real roots whereof at least two are equal, and 3 if there is only one real root. A 2-degree equation $x^2 + Dx + E = 0$ is solved from $x^3 + Dx^2 + Ex = 0$, ie you set $A = D$, $B = E$, $C = 0$, and disregard a root 0.
<code>fdist(x, v1,v2)</code>	F-distribution function with $v1$ and $v2$ degrees of freedom
<code>fdistc(x,v1,v2)</code>	$1 - \text{fdist}(x,v1,v2)$; see remark for <code>probnormc()</code> .
<code>fdistinv(x, v1,v2)</code>	inverse of <code>fdist()</code> .
<code>max(x,y,z)</code>	maximum of x , y and z
<code>min(x,y,z)</code>	minimum of x , y and z

So `max()` and `min()` are defined for two or three scalars. At present it is too complicated to admit more than three scalars as arguments. You will have to combine functions, ie `max(max(x,y,z),max(t,u))`.

For these functions except `cube()`: If all arguments and the variable receiving the return value are double the calculation is carried out with only double variables and functions. If at least one of them is mouble, the calculation is done in mouble and, if the variable receiving the return value is double, the result is converted from mouble to double. For `cube()` this holds for the arguments as stated above, but the type of the return variable does not influence this.

Some explanations of the algorithms behind functions

These are designed to give the right answer with the chosen precision, including correct rounding, in the shortest possible time. This often means using several algorithms for a function, choosing the best one depending on the arguments and the precision.

Example: For `probnorm()`, `probnormc()` and `probnormmp5()` which compute the normal distribution Φ , Rapp uses the four formulas (26.2.10), (26.2.11), (26.2.12) (26.2.14) on p 932 of Abramowitz & Stegun (1955), Handbook of Mathematical Functions, Dover Publications, New York. (26.2.10) is used for small positive x. (26.2.11) is used for somewhat larger x. (26.2.14), even part as described in "Numerical Recipes in C", is used for still larger x in `probnormc(x)`. (26.2.12), finally, is used for the largest x in `probnormc(x)`, where it can give the right answer considering the remainder term R_n . These four intervals depend on the chosen precision. "Numerical Recipes in C" formula (6.2.8) for `erf()`, using the gamma distribution, can be adapted to compute $\Phi()$. But I do not use it for mouble, since it is much slower than the combination of methods above. For double I use it for large enough x, since it is then more accurate.

For `probnin()`, which computes the normal distribution function inverse, Rapp uses a combination of Abramowitz & Stegun (26.2.23) and Boris Moro's and Peter J. Acklam's method for an initial aproximation. After that starts an iterative solution by the Acklam method of a Taylor approximation up to degree 2 around successive approximations. If this iterative solution does not converge, Rapp switches to rootfinding of $\Phi(y) = x$ by the bisection. (Note. For very small x, decreasing below the double range, the Moro-Acklam method deteriorates, so that the simpler formula Abramowitz & Stegun (26.2.23) actually becomes preferable.)

For $\Gamma(x)$ and its logarithm, Abramowitz & Stegun formulas (6.1.15), (6.1.17), (6.1.33) and (6.1.40) are used. (6.1.15) and (6.1.40) are used together. From (6.1.15) it follows that $\log \Gamma(x) = \log \Gamma(x+K) - \log((x)(x+1)...((x+K-1)))$ for any positive integer K. Applying (6.1.40) to $z = x+K$ we can get the error arbitrarily small in absolute value for any n in (6.1.42) by choosing K large enough. The maximal n used is 169. Rapp is optimized to use the best pair (n,K) if this method is used. Also Rapp is optimized to use the best of this method and (6.1.33). The latter can be applied to any x by using (6.1.15), but for too large x the number of multiplications will be too time-consuming. (6.1.33) must be used for x close to 1 or 2 in order to get the desired relative accuracy in a reasonable time. The maximal n used in (6.1.33) is 335.

For `bessik(x,v)` and `bessjy(x,v)` the functions `beschb()` and `chebev()` of "Numerical Recipes in C", §6.7, are, in the mouble algorithm, replaced by series developments obtained from Abramowitz & Stegun (6.1.33). This satsifies the demand that the calculation is sufficiently accurate for v close to 0.

For more info on the ways the functions are computed, send me a mail.

Gosub statement

Syntax
gosub label

Meaning as in classical Basic. Transfers execution to the statement after label, where label shall be a word ending with a colon. After that statement shall be a return statement that transfers execution to the statement after the gosub statement.

Example:
d01 = 2 Gosub Routine print d01

```
goto L01
Routine: d01 = exp(d01) Return
L01:
```

To prevent execution while passing Routine, put a goto and label around Routine, as in the example. Or put Routine last and terminate the program with Exit or Stop before it.

Goto statement

Syntax
goto label

Meaning as in classical Basic. Transfers execution to the statement after label, where label shall be a word ending with a colon.

Example:
goto L01

L01:

Helpmatricesfree statement

Syntax
Helpmatricesfree

If you do not use matrix calculus for very large matrices, you do not need to read this. If you do create large matrices, Rapp will give you instructions to read this section of the manual in case Rapp aborts due to too much memory use. The statement frees the memory occupied by help matrices, created by Rapp, directly after their use. If desired, place the statement at the top before or after the digits statement. See statement Free with parameter helpmatrices for freeing at particular points.

Help matrices are created for storing intermediate results in non-trivial matrix expressions, such as $A = \text{trp}(\text{inv}(\text{g}(\text{A} \cdot \text{B} + \text{C} \cdot \text{E})))$. Rapp does not automatically apply Helpmatricesfree, since such a matrix expression might appear in a loop that is executed many times. If Helpmatricesfree is in effect, then Rapp increases and decreases the memory for helpmatrices each time the loop is executed. The calculation goes through, but takes unnecessarily long time. If such loops that are executed more than a few times do not exist, then you should use Helpmatricesfree. If such loops exist, you can use the statement Free helpmatrices after the loops, see statement Free above.

Print to statement

Syntax
print to [file]

If a file within single or double quotes is given, the output from print statements will be directed to this file until another print to is given. Print to without a file directs the output (again) to the screen. If you run a program with Mbasic progfile > file, the output (but not all of it depending on your program) will be to this file.

Printing arrays

Syntax
print [nb|complex] [%[+][0][n[.m]]] array [\n]

Examples:
print Ma
print %9.2 Ma\n,Mb \n
print %+09.2 Ma\n
print %09.2 3*(Ma-MB) \n

Several array expressions are printed with the same print statement if separated by commas.

All elements of Ma will be printed with sort order first index, second index, ... , last index. When the last index reverts to 1 a linebreak takes place. When the next last index reverts to 1 one more linebreak takes place.

If a format string `%n.m` is given, the number will print in at least `n` columns right-justified, of which `m` will be decimals preceded by a point. The format `%n` is the same as `%n.0`, unless `n = 0`. See below for info on scientific format printing.

If `nb` (meaning no blank) is not given directly after `print`, printed numbers on a line will be separated by one blank, in addition to blanks filled in left to have numbers occupy `n` positions. This is mostly what you want, but otherwise you give `nb` to print items with no blanks between them, not counting those filled in to the left of items.

If the keyword `complex` is written directly after `print`, complex variables and arrays have their values printed as `re+i*im` if `im ≥ 0` and as `re-i*|im|` if `im < 0`. The values in the last complex dimension (the next last dimension if considered as real arrays) will be written on the same row. This printing is for easier understanding of a complex variable or array; it is not intended for subsequent use in the program. Real variables and arrays are not affected. The keyword cannot be given together with `nb`.

If `r = (number of integers including leftfill) + (one position for decimal point) + (number of decimals) > n`, the `r` positions will print and the following numbers on the same line will be printed `r-n` columns more to the right.

If a format string `%0n.m` is given, the left filling to obtain right-justification will be with zeroes. If a plus sign `+` is given directly after `%`, possibly followed by a zero `0`, all numbers will be printed with the sign `+` or `-` first. If not, only negative numbers will be printed with the minus sign `-` first.

These rules are those governing printing in C with `%n.mf`. Here the `f` last is omitted.

If no format string is given, all numbers will print in scientific format concluded by an exponent. All numbers will be printed with the sign `+` or `-` first. The format string `%0` causes scientific format printing with a leading sign only for negative numbers.

If `\n` is given a linebreak takes place. `\n` needs not be preceded by a blank.

The program

```
mouble Ma[4][2][3]
Ma = {
  01 02 03.16
  04 05 06

  07 08 09
  10 11 12.04

  13 14 15
  16 17 18

  19 20 21
  22 23 24
}
print %6.1 Ma \n
```

will produce the output

```
  1.0    2.0    3.2
  4.0    5.0    6.0

  7.0    8.0    9.0
 10.0   11.0   12.0

 13.0   14.0   15.0
 16.0   17.0   18.0

 19.0   20.0   21.0
 22.0   23.0   24.0
```

Printing scalars

Syntax

```
print [nb] [%+][0][n[.m]] scalar [\n]
```

Examples:

```
print x
print %9.2 x,%10.0 y \n
print 3.14\n,6.28\n
```

Several scalars are printed if separated by commas. Printing scalars, matrices and texts can be combined in the same print statement.

Here only one number is printed. See Print arrays above for formats and linebreaks. With nb, no blank will be put after the number.

Example with printing of array elementwise:

```
mouble Ma[3]
Ma = { 1.22 2.33 3.44 }
print %02.2 Ma[1] print %02.2 Ma[2] print %02.2 Ma[3] // same output as print %02.2 Ma
```

Printing text

Syntax

```
print [nb] ['text'] [\n]
```

Several text strings are printed if separated by commas.

Up too 5000 character long texts can be printed.

Single or double quotes can be used around the text. Use single quotes if the text contains double quotes and vice versa. Without \n for linebreak the next printed number or text will be on the same line. With one blank in between unless nb was given.

The text can be omitted, and then only a blank or a linebreak is printed.

Only constant texts can be printed, not alphanumeric variables. (Mbasic is intended for mathematics, not text handling. It has no alphanumeric variables.)

Example:

```
print 'Result of operation :' print m2 \n
```

Pseudo-random numbers

The double variable RND will take a new value, uniformly and randomly distributed in (0,1), every time it is used. Successive values will be independent.

The mouble variable MRND works in the same way and should have all digits independent and randomly distributed, but I am not sure about this.

Although these numbers are pseudo-random like all such numbers generated by computers, in practice they can be regarded as random. The mechanism period is said to be $> 2 \cdot 10^{18}$. The mechanism is the same as that used for Procs Bich and Sample, namely the one of "Numerical Recipes in C", Ch. 7, routine ran2 in double and mouble, respectively.

Read from statement

Syntax

```
Read from [file]
```

If a file within single or double quotes is given, the input from read statements will be from this file until another read from is given. Read from without a file causes input (again) to come from the keyboard = what you type.

Example:

```
Read from 'C:\Rapp\Data\minput.txt'
Read m1
Read from
```

Read line statement

Syntax

Read line

Reads a line from keyboard or a file depending on the last preceding Read from. The line content is accessed with the functions `readline(x)` and `readline(x,y)` described above.

Reading data from a file

Syntax

Read scalar | array

Before the statement, a statement Read from file shall have been executed. Reads the scalar or array with blank-separated elements from the file.

For a complex variable or array, the file can have either two words for the real and imaginary parts, or a word (without embedded blanks) in the form $x+i*y$ or $x-i*y$, where x and y are numbers and y is non-negative. The imaginary unit i must be given in lower case. For a complex array with n elements, the file can have either n numbers or $n/2$ words in the form $x+i*y$ or $x-i*y$. This allows you to store data in a file with print complex in the form $x+i*y$ and read them again from the same file, thus removing uncertainty as to the type of content in the file. Rapp does not check that the variable or array is complex. If it is a real array, two numbers x and y or $-y$ will be read.

The next Read will get the next blank-separated number in the file, unless the file is closed in between.

Several items (scalars or arrays) can be read with the same read statement if separated by commas.

Example: See above.

Reading data from keyboard

Syntax

Read variable | array

Several items (scalars or arrays) can be read with the same read statement if separated by commas in the statement.

Complex numbers and arrays can be given as a sequence of numbers or a sequence of words in the form $x+i*y$ or $x-i*y$, as stated above under 'Reading data from a file'.

Either no Read from statement shall have been executed before this, or the last Read from preceding this shall have been without a file.

You are prompted for the input by the name of the variable(s) or array followed by $\Rightarrow$. Press enter after giving the number of items prompted for, blankseparated.

Redim statement

Syntax

Redim array[scalar][scalar] ... [scalar]

Example:

Redim Ma[d1][3]

Expands or shrinks the array to a new size, taking up more memory or freeing some if the array has been used.

The number of scalars within [and] shall be the same as the number of dimensions initially declared for the array.

If the array was used before, elements with indices $\geq$ than largest of the old and new dimension lower limits and $\leq$ than the smallest of the old and new dimension upper limits are preserved. That is, elements with indices valid both in the old and the new array. New elements get the value 0.

Return statement

Syntax
Return

Example: See under GOSUB statement.

Causes the statement after the calling GOSUB to be executed.

Sort statement

Syntax
Sort array
Sort array[scalar]

Examples:

```
Sort Mc
Sort Mc[2]
Sort Mc[d1]
```

Sorts the elements of the array in ascending order from and including the scalar index given. If no index is given it will be the lower index limit, which is 1 if you did not write [lower:upper]. Valid for one-dimensional arrays.

Sortd statement

As Sort with descending order.

Stop statement

Synonym to Exit.

System statement

Syntax
system 'text'

where text shall be enclosed in single or double quotes. If the text contains single quotes, enclose it in double quotes and vice versa.

Executes the system command in the text.

Example:

```
system 'copy "file no 1.Txt" C:\Rapp\Txt\moutput.txt'
```

Proc Ovelim

Example:

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Ovelim
  prog(C:\Rapp\Rpp\Ov.Rpp) run
  Infil("C:\Rapp\Data\temp2.Txt" "C:\Rapp\Data\temp3.Txt") dlm(9)
  Var(F01 $ F02 $ F03 $ F04 $ F05 $ F06 $ F07 $ F08 $ F09 $ F10 $ F11 $
      F12 F13 F14 $ F15 $ F16 $ F17 $ F18 $ F19 $ F20 $ F21 $ F22 $)
  Fromdatevar(F12) Untodatevar(F13) key(F01 F03)
  Utfil(C:\Rapp\Data\temp4.Txt) Noempty
Endproc
```

Overlap Elimination: Creates a Rapp-program for eliminating policy period overlaps.

In insurance production systems successive policy cover periods often overlap. For example, a policy period might have begun 20120101 with an intended untodate (date after last day of cover) 20130101. Say that in the middle of the year the policy holder changes the policy terms by expanding the cover. This generates a new policy cover period starting at 20120701, with untodate 20130101 or 20130701. But the previous period's untodate remains 20130101 in the production system. A statistical data warehouse would be better served by changing it to 20120701, thus eliminating the overlap. This proc performs such overlap elimination by lowering untodates if warranted.

The created Rapp-program consists of two Proc Data steps. The first one sorts the indata to a temporary file by the ID-variables and Fromdate. The second one reads the temporary file and compares the given Untodate with the Fromdate of the next line, provided that the next line has the same ID. If this next Fromdate of the same ID is lower than Untodate, then the latter is lowered to this Fromdate in the outfile. The Next() function, described under Statement infiler of Proc Taran, is used. The function is applicable also to Proc Data, Proc Match and Proc Sum.

The input files temp2.Txt and temp3.Txt are merged into temp4.Txt, where untodate F13 has been replaced by a corrected, ie sometimes lowered, untodate.

Parameters

Dlm()	Delimiter, as in Proc Taran et al.
Dvar()	Derived variables, as in Proc Taran et al. If Untodate is used it will be the corrected date.
Firstobs()	First line included, as in Proc Taran et al.
Fromdatevar()	The variable for fromdate.
Headerline	Gives a header in the outfile, as in Proc Taran et al.
Infil()	Infiles. Syntax as in Proc Durber, ie if there is more than one file each must be enclosed in double quotes.
Key()	This parameter contains the ID-variables defining a specific policy.
Noempty	As in Proc Taran et al.
Prog()	The generated Rapp-program. If not given, a temporary Rapp-program is created and deleted after the proc.
Run	If given the generated program is run directly. If not given, then Prog() must be given, since it is meaningless to generate and delete a program without running it.
Untodatevar()	The variable for untodate.
Urval()	Selection of records as in Proc Taran et al.
Utfil()	Outputfile.
Uvar()	Outfile variables. If not given, all infile variables, including derived ones, will be moved to the outfile. If given, untodate must be included and will be replaced by the corrected date in the outfile. (The proc is meaningless without inclusion of the corrected untodate.) Type declarations are not required.
Var()	The infile variables as in Proc Taran et al. No type means integer numeric. For other types type declarations are required. Fromdate and untodate have to be numeric, ie cannot be marked by \$ as alphanumeric. A numeric ID-variable can be marked as alphanumeric, if it is not used in dvar() and if a specific value cannot have different appearances, such as "2", " 2", "02" for the value 2. Apart from the dates and the ID-variables, any variable not used in dvar() can be taken as alphanumeric. In the example all variables except the dates are marked by \$ for simplicity.

There is a menu for overlap elimination in Rappmenus, using the proc.

Proc Percen

Example:

```
Proc Percen infil(Fors1.Txt) utfil(Result.Txt) columns(5 11) firstobs(2) Endproc
```

Adds the values in the columns specified in columns() - separated by delimiter space, tab, or semicolon - and calculates 100 percentile values where percentile 0 is the lowest

percentile and percentile 100 the maximum value for the sum per line of the column values. Number of lines and average value is also given. Percentile values can differ slightly from those of a strict calculation.

The example has all parameters. See Proc Durber for explanation of like-named parameters.

Proc Print

Example:

```
Proc Print listfil(C:\xx\lista1.txt) Endproc
```

If a printer has been given in Proc Init, the listfile listfil is sent to that one, otherwise to the default Windows printer. In the former case the printer must be able to take PostScript. The listfile can be an outputfile from Proc Graf with embedded SAS code for the graphics. Can also be an arbitrary textfile such as a SAS program. At difficulties to get this proc to work, print inside a PDF file created by Proc Graf.

Proc Pseu

Example:

```
Include C:\Rapp\Rpp\Init.rpp
Proc Pseu listfil(C:\Rapp\Txt\Credlist1.txt) infil(C:\Rapp\Data\HierInsur.Txt)
  p 1 Nogroup 1 Noexposure 3 Nocclaimno 4 long ENDPROC
Proc Pseu listfil(C:\Rapp\Txt\Credlist2.txt) infil(C:\Rapp\Data\HierClaim.Txt)
  p 2 Nogroup 1 Nocclaimcost 4 long ENDPROC
```

Performs credibility analysis freestanding from Proc Taran. While in the latter proc insurances and claims are analyzed simultaneously, in Proc Pseu only one of them is analyzed. The proc gives my variance pseudo-estimators according to my article "Credibility pseudo-estimators", SACT 2018. My estimators are scale invariant. See "Proc Taran multiclass: OJ2010, SR 2018" further down in this manual.

The infile shall be a text file with fields

```
group
  exposure for p = 1
  number of claims for p = 1
  claim cost (severity) for p = 2
```

The parameters starting with No are the order numbers of the corresponding variables. For p = 2, one line for each claim.

The fields shall be separated by one or more blanks, or by a semicolon or tab character. If a group can contain embedded blanks, the delimiter must be semicolon or tab.

Parameters

```
infil    input file as given in the example, conforming to general Rapp usage
listfil  list output file as given in the example, conforming to general Rapp usage
long     if present a long list with all parameters, otherwise only the general ones.
p        1 for insurances, 2 for claims
```

For how to make Excel files from the direct text output listfil, look at a generated Rapp program from Rappmenus.

Proc Reschl

Example:

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Reschl Infil(Tril.Txt) Utfil(Utl.Txt) Futfil(Futjl.Txt) Textfil(Datafil.Txt) Endproc
```

Example of syntax, where the parameters in [] are optional, and | indicates alternatives. Any order. All parameters are case independent. More parameters are described below.

PROC Reschl

```
infilskk(fil1)|infil(fil1) [infilbet(fil2)] [parmfil(pfil)]
utfil(fil3)
[textfil(fil4)]
[xutfil(fil5)]
[futfil(fil6)] [ftextfil(fil7)] [gamma]
[ackskk|ack] [ackbet]
```

```
[exd]
[nsvanssskk(m)|nsvans(m)] [nsvansbet(m)]
[svanssskk(A|E|G|W)|svans(A|E|G|W)] [svansbet(A|E|G|W)]
[nmseinfsskk(m)] [nmseinfbet(m)]
[discountToClaimperiodRateI(0)] [discountToClaimperiodRateP(0)]
[interestratefile(File/Lastclaimdate/FlagIncurredFlagPaid)]
[tidssort(D|W|M|Q|H|Y)] [munich] [sigPnml(s1)] [sigInml(s2)]
ENDPROC
```

Parameters

ackbet	If specified, the payments are cumulative.
ackskk ack	Ditto for known claim cost (Incurred).
cnmseinfbet()	Number of claim back periods for least squares estimation, where the last is (maximum claim period index) - cxmseinfbet(). Default all.
cnmseinfsskk()	Ditto for known claim cost.
cnmseusebet()	Number of claim back periods for curve fitting, where the last is (max claim period index) - cxmseusebet(). Default all. The fitted curve is a continuation backwards of the tail determined by svansbet(m), m = A, E, G, W. The curve replaces IBNS-values in the unknown triangle, which would otherwise have been calculated by chainladder or smoothing of factors with GLM.
cnmseuseskk()	Ditto for known claim cost.
cxmseinfbet()	Number of claim periods backwards not to be used for least squares estimation. Default 0.
cxmseinfsskk()	Ditto for known claim cost.
cxmseusebet()	Number of claim periods back where curve fitting is not to be used. Default 0.
cxmseuseskk()	Ditto for known claim cost.
dec()	Number of decimals in outputfiles utfil() and xml file from utfil().
discountToClaimperiodRateI() and discountToClaimperiodRateP()	Interest rates for discounting. Overruled by interestratefile(). See this parameter and Rappmenus / Reserves.
dnmseinfbet()	Number of pay development periods backwards for least squares estimation, with the last being (maximum payment period index) - dxmseinfbet(). Default 6.
dnmseinfsskk()	Ditto for known claim cost.
dnmseusebet()	Number of pay development back periods for curve fitting, where the last is (max payment period index) - dxmseusebet(). Default 0. The fitted curve is a continuation backwards of the tail determined by svansbet(m), m = A,E,G,W. The curve replaces IBNS-values in the unknown triangle, which would otherwise have been calculated by chainladder or smoothing of factors with GLM.
dnmseuseskk()	Ditto for known claim cost.
dt()	Specifies the time for discounting to 'then'; where in the claim period. Values B, M, E for Beginning, Middle, End. Default E.
dxmseinfbet()	Number of pay development periods of backwards not to be used for least squares estimation. Default 0.
dxmseinfsskk()	Ditto for known claim cost.
dxmseusebet()	Number of pay periods of development back where curve fitting is

	not to be used. Default 0.
dxmseuseskk()	Ditto for known claim cost.
exd	Specifies that the inputfiles were created by copying triangles in Excel and paste in textfiles. A line of such a textfile is either a line of text that clearly shows that it does not contains the numerical triangle values, or else it contains triangle values separated by blanks, semicolon or tab character.
expbet()	if given it is used for tail computation as p in the algebraic method. If not given it will be estimated. See Appendix 5.
expskk()	Ditto for known claim cost.
ftextfil()	Semicolon separated textfile for output of GLM smoothing factors.
futfil()	Report with the output based on smoothing factors.
futfilipc()	File for use as listfilut() in Proc Graf.
fxutfil()	Report output in reverse order of arguments and claim period.
gamma	If the specified gamma loglink is used in smoothing factors.
genhead()	General heading for the reports within single or double quotes. Maximum 80 characters.
incpaid()	Specifies what the input data represent. Default 0. Needed for the right headlines in graphs. 0 incurred and paid 1 number of claims: Payment-date shall then be report-date and the paid amount set to 1. Then the reports are interpreted as IBNR analysis of the number of claims. 2 Paid only 3 incurred only With value of 1,2,3, the same inputfile shall be specified for payment and changes in known claim cost.
infilbet()	Triangle file for payments in the form in which they are made by Proc Restri.
infilskk() infil()	Ditto for known claim cost.
interestratesfile()	A file containing interest rates per month with lines on the form Month Rate, eg 201801 2.75. A Lastclaimdate that is the now time epoch shall be given after a slash. After yet a slash shall be 1 if the file applies to Incurred, otherwise 0. After that 1 or 0 if the file applies to Paid or not. This overrules the parameters discountToClaimperiodRateI, P. For more explanation, use the application Rappmenus / Reserves, read the right-click documentation there and look at the generated Rapp program.
kilo	Excel-inputfile xlmfilinput() has monetary amounts in thousands.
macksefultimbet()	Manual estimate of the standard error for the development factor from the last column of the triangle to the tail. Applies to Paid. The parameter that was set to 0.02 in the final section of Mack (1999). Default 0.02.
macksefultimskk()	Ditto for Incurred.
mackseperiodbet()	Manual estimate of the standard error of individual development factor for an average claim period. Applies to Paid. The parameter was set to 0.03 for claim period 3 in Mack (1999). Default 0.03.
mackseperiodskk()	Ditto for Incurred.
mackvolbet()	Accumulated end value of the triangle for the average claim period. The one that Mack (1999) set to 5608000. Default value is

	the average over all claim periods. Applies to Paid.
mackvolsskk()	Ditto for Incurred.
mega	Excel-inputfile xlmfilinput() has monetary amounts in millions.
munich	Munich Chain Ladder (MCL) is used.
nsvansbet()	Number of pay development periods forward when the tail is computed. If 0 no tail is calculated. Default 0.
nsvanssskk() nsvans()	Ditto for known claim cost. Default 0.
parmfil()	Gives parameter estimates for MCL to be used in place of the computed ones. Also gives the combinations of claim time index and development period index (both with base 1, not 0) to be excluded in the calculation of variance estimates and correlation coefficients.
sigInml()	Value of $I(n-1) \rightarrow n$, if not given it will be set to 0. (Quarg & Mack p. 627)
sigPnml()	Value of $P(n-1) \rightarrow n$, if not given it will be set to 0.
svansbet()	Method for tail estimates for Incurred = known claim cost. Values, where c_{ij} is the increment for claim period i and development period j and $V[i] = \sum_j c_{ij}$ over those j used for the estimates. A $E[c_{ij}] = \alpha V[i] / [1 + q(\beta + j)^p]$ E $utvecklingsfaktor = 1 + \beta \cdot \exp(-\lambda j)$ G $E[c_{ij}] = \alpha V[i] \cdot q^j$ W $E[c_{ij}] = \alpha V[i] \cdot [(j+1)^{(\beta-1)}] \cdot [q^{((j+1)^\beta)}]$ Algebraic Exponential Geometric Weibull For A p can be chosen by the user with $expskk(p)$ respectively $expskk(p)$, eg $expskk(8)$. Default is p that minimizes the sum of square deviations. Default method is E. For adjustment of the payment tail to the right level taking into account the known claim cost, see below. See also Appendix 5.
svanssskk() tail()	Ditto for known claim cost.
tailcorr()	Indicates the method for adjustment of the tails so that this equality is obtained: $Incurred(known + IBNS + tail) = Paid(known + IBNS + tail)$ 0: None. 1: Paid tail adjusted. 2: Incurred tail adjusted.
tailfact()	Factor for adjusting the Incurred-tail, default 1. Equal factor for all claim periods and development periods $> n$. The factor applies to the tail values computed in the first place. See below for order of priority.
tailpctlastbet()	Provides the required percent of the final cumulative value including IBNS for Paid-tail. See below for priority with other. parameters that affect the tail size.
tailpctlastsskk()	Ditto for Incurred-tail.
textfil()	Semicolon separated textfile for the report output.
tidssort()	Period length for the inputfiles' claim and development periods in months, or code D, W, M, Q, T, H, Y. Eg $tidssort(4)$ is the same as $tidssort(T)$, ie Tertial = four months. Needed for correct summary to rolling-12-month figures and for discounting() if the annual interest rate $arsranta() > 0$.
userfil()	Optional. Textfile with additional data shown in (f)utfil(), and (f)textfil() after regular calculated data per claim period. First word of a line of data must be either an index $0 \dots n$ ($1 \dots n+1$) or a notation such as 2003H1 occurring as first word in the files mentioned. Thereafter, any number of blank-separated columns. Data for different blocks (such as different business

lines) are separated by one or more non-data lines, such as a blank line. Can eg contain book value per claim period. See the instruction further down for adapting a report file.

`usernames()` Optional. Field names for userfil, eg `usernames(Premium Clsetcst)`.

`utfil()` Report output with various undiscounted and discounted values. This parameter determines also the name of an outputfile with standard errors, according to Thomas Mack, "The standard error of chain ladder reserve estimates: Recursive calculation and inclusion of a tail factor" Astin Bulletin. Vol. 29. No 2. 1999. 361-366. The report with standard errors is always created and gets `-mack` in the name before the point. Eg `utfil(Reslist01.Txt)` gives `Reslist01-mack.Txt`.

`utfilapc()` File as `utfilipc` below, but with accumulated values.

`utfilipc()` File for use as `listfilut()` in `Proc Graf`. For example,

```
Proc Reschl ... utfilipc(Inc-per-skper.Txt) ... Endproc
Proc Graf listfilut(Inc-per-skper.Txt) Pdf(Inc1.Pdf) Endproc
```

gives graphs of increments with different colors for true values, chainladder estimated values, tail values. One graph per claim period.

If `utfilapc()` but not `utfilipc()` is given, then Rapp creates a file with content as `utfilapc()` and name `utfilipc()` with `-Ack` appended. Conversely, if `utfilipc()` but not `utfilapc()` is given, then Rapp creates a file with content as `utfilipc()` and name `utfilapc()` with `-Inc` added.

`visachl` If given, IBNS by chainladder is shown simultaneously with curves for parametric values. The parametric values are the ones used in the reports.

`xmlfilinput()` Input from previously created xml file at parameter `xmlinput`.

`xmlinput` Input is taken from previously created xml file.

`xutfil()` Report output in reverse order of arguments and claim period.

This proc works with at most two triangle files such as those made with `Proc Restri` or manually produced. If there are pure upper development triangles in the files, then (Munich) chainladder calculation is made with computation of tails after the last available development period.

If also lower future prediction triangles are in the files, then those values are taken at face value, shown and summed up in the output as IBNS. Thus the RDC method can be used with `Proc Reschl` (despite the 'chl' in the proc name referring to chainladder, with which it was originally made). With parameter `Uttri()` to `Proc Bich` a file is created that can be used by `Proc Reschl` in that way.

Manually made files

Option 1

Mark a line of data by letting it begin with two integers and then a space and a colon (:). The first integer is just text information in the outputfiles and can be for example a quarter notationn such as 20083. The second integer is the index of claim periods, which may begin with 0 or 1. Examples from "Munich Chain Ladder" by Gerhard Quarg & Thomas Mack (2003), first payment file, then known-claim-cost file.

Paid accumulated

20011	1 :	576	1804	1970	2024	2074	2102	2131
20012	2 :	866	1948	2162	2232	2284	2348	
20013	3 :	1412	3758	4252	4416	4494		
20014	4 :	2286	5292	5724	5850			
20021	5 :	1868	3778	4648				
20022	6 :	1442	4010					
20023	7 :	2044						

Incurring accumulated

20011	1 :	978	2104	2134	2144	2174	2182	2174
20012	2 :	1844	2552	2466	2480	2508	2454	
20013	3 :	2904	4354	4698	4600	4644		
20014	4 :	3502	5958	6070	6142			
20021	5 :	2812	4882	4852				
20022	6 :	2642	4406					
20023	7 :	5022						

Option 2

Copy one or more triangular blocks in Excel, only the columns with known-claim-cost-changes and payments. Between two blocks there should be at least one line of text, ie a line that cannot be data. Paste it into a textfile such as Notepad. Rapp then sets the index 1, 2, 3, ... on the claim and payment periods. Blank fields may occur separated by delimiter semicolon or tab after the diagonal of the triangle.

Examples with the same numbers as above:

```
576;1804;1970;2024;2074;2102;2131
866;1948;2162;2232;2284;2348;
1412;3758;4252;4416;4494; ;
2286;5292;5724;5850; ; ;
1868;3778;4648; ; ; ;
1442;4010; ; ; ; ;
2044; ; ; ; ; ;
```

```
978;2104;2134;2144;2174;2182;2174
1844;2552;2466;2480;2508;2454;
2904;4354;4698;4600;4644; ;
3502;5958;6070;6142; ; ;
2812;4882;4852; ; ; ;
2642;4406; ; ; ; ;
5022; ; ; ; ; ;
```

Examples of parmfil at MCL

Parameters as the first word on a line (possibly preceded by tab or ;) means new block. After fP etc you give a sequence of development period indices and replacement value. The lines need not appear in the order below.

```
lambdaP 0.636 lambdaI 0.436
fP 1 2.437 3 1.029
fI 1 1.652 3 1.000 6 0.996
sigP 1 13.456 2 3.666 3 0.482 4 0.210 5 0.479 6 0.100
sigI 1 9.727 2 2.544 3 1.004 4 0.120 5 0.860 6 0.100
q 1 0.533 2 0.849 3 0.928 6 0.960 7 0.980
rhoP 2 4.990 3 2.167 4 1.619 5 1.791 6 0.236
rhoI 1 5.711 2 3.819 3 1.918 4 1.461 5 1.637 6 0.222
```

Under Ex (exclusions) is given on a line for a specific time period

Claim-period-index i = 1, 2, ...

[Development-period-index s = 1, 2, ...]. If no development-period-index is given, the word after claim period index refers to all development periods.

Words containing:

I if $I(i,s+1)/I(i,s)$ shall be excluded from the calculation of fI and sigI.

P if $P(i,s+1)/P(i,s)$ shall be excluded from the calculation of fP and sigP.

Q on $Q(i,s)$ shall be excluded from the calculation of q, rhoP and rhoI.

Ex

```
3 1 IP 3 Q
5 1 IPQ 2 IPQ 3 IPQ 4 IPQ 5 IPQ 6 IPQ 7 IPQ
7 IP
```

End of example of parmfil at MCL

Also without actual Munich Chain Ladder exclusions can be made. Namely, if only one inputfile infil is given, which is interpreted to contain payments = known-claim-cost-changes, at the same time as parameter Munich is given.

If futfil() is given then is made, parallel with the chain ladder estimates, also reserve estimates with factor smoothing by gamma loglink if gamma was given, otherwise the marginal-totals-method, on

1. the distribution argument that the triangles in the inputfile are separated on,

- such as line of business.
2. claim period.
 3. development period.

Here are used all the cells of the distribution argument, claim period and development period with known outcomes, with the exception of the value Total of the distribution argument. The latter is supposed to give a summation of the previous triangles. Risk outcome, corresponding to claim cost in tariff analysis, is here payment amount alternatively known-claim-cost-change. Exposure is set to the risk outcome of development period 0 for the claim period that such a cell with known outcome belongs to. This smooths random fluctuations in the triangular cells over the distribution argument. If some risk outcome is negative, then the classical iterative marginal-totals-method is used. If RDC or Schnieper was given to Proc Restri, also predicted outcomes are used.

Factor smoothing with marginal-totals for only claim period and development period is the same as chain ladder calculation. So if there is only one triangle in addition to the Total-triangle, then this gives no new info. This even if some risk outcome is negative.

With two inputfiles one shall be a file of payments and the other a file of known-claim-cost-changes, identified by respective infilbet() and infilskk(). Without parameter Exd, they must be correspond line by line. Thus, line r in one shall refer to exactly the same claim period and the same development periods as line r in the other, or be a line of text without payment/changeamounts in both files. With parameter Exd, there may be any number of lines between the blocks, different numbers for the two files; however, the number of blocks shall be equal. Then the last non-blank line of text before a block is taken as the title for the block. The intended use of the proc is partly for files produced by Proc Restri without parameter Exd, and partly for files copied and pasted from Excel with parameter Exd.

With only one inputfile specified as infil() or infilskk(), it is assumed to relate to known-claim-cost-changes in terms of column headers in the outputfiles. However, the output is relevant for a payment file as input if the column headers are interpreted properly.

If ackbet respectively ackskk is given, then a triangle-value for development period j is assumed to be the cumulative amount up to j. Otherwise, the triangle-values are assumed relate to payments respectively known-claim-cost-changes during period j.

Yearly interest rate is expressed as a percentage, eg arsranta(3) gives the discounted value at that interest rate. Default is 0.

With tidssort is indicated length of claim and development periods in the inputfiles, where D is for days, M for months, Q for quarter, T for tertial and Y for years. Default is years. Also, it is used for summation to rolling 12-month figures in columns I and J.

The outputfile() will contain a list of undiscounted and discounted reserves. The discounted values are meaningful only if there is a payment file, because if only one file with known-claim-cost-changes exists, then one cannot know how they are distributed as payments and adjustments of claim-handler reserve.

If textfil() is given, then a semicolon delimited textfile is made that is suitable for input into Excel or SAS, containing: Distribution arguments (if any), period of time, claim period indices 0,1,2, ..., reserve factor for the final accumulated value, reserve, claim cost, tail, and several other concepts that are available in utilil().

For reserve calculation with factor smoothing, futfil() has the same role as utilil(). And ftextfil() has the same role as textfil().

Tail adjustment

If tailcorr(1) or tailcorr(2) is given, then the tails are adjusted so that this equality holds, for each time period separately, for undiscounted values in three terms regarding known triangle, unknown triangle and tail after unknown triangle:

$$\begin{array}{l} \text{(Known Incurred)} + \text{(IBNS Incurred)} + \text{(Tail Incurred)} = \\ \text{(Known Paid)} + \text{(IBNS Paid)} + \text{(Tail Paid)} \end{array}$$

Shorter written $KI + II + TI = KP + IP + TP$

tailcorr(1): Paid tail is adjusted and Incurred tail is left unchanged.

tailcorr(2): Incurred tail is adjusted and Paid tail is left unchanged.

Proc Reschl thus fills out a possible gap between the known claim cost and payments so that the total claim cost including IBNS and tail will be the same regardless of whether known claim cost or payments are considered.

At tailcorr(1) the following rule is used. Analogously at tailcorr(2).

If (Tail payments) is 0 before this adjustment, then is set

$$TP = KI + II + TI - KP - IP$$

Otherwise, ie if (Tail payments) is not 0 before this adjustment, then is set

$$F = (KI + II + TI - KP - IP) / (TP \text{ before adjustment})$$

and

$$TP = F \times (TP \text{ before adjustment})$$

The factor F is multiplied to all individual tail values (before this adjustment) for each development period to provide adjusted individual tail values.

Priority order low to high of tailcorr(), tailfact(), tailpctlastbet(), tailpctlastskk()

1. tailfact() is applied to Incurred if given. Equal factor for all claim periods.
2. tailpctlastbet() and tailpctlastskk() are applied if given. For example tailpctlastskk(5) implies total Incurred tail for claim period $j = 0.05 \times (\text{Incurred} + \text{IBNS claim period } j)$. Then tailfact() is superseded.
3. tailcorr() is applied if given 1 or 2. For example, at 2 the Incurred tail is adjusted and tailpctlastskk() becomes void.

Example, where Mclp.Txt is payment file and Mcli.Txt is the known-claim-cost file above.

```
Proc Reschl Infilbet(Mclp.Txt) Infilskk(Mcli.Txt) ackbet ackskk
  nsvansskk(0) nsvansbet(0) tidssort(q) arsranta(4.5)
  Munich sigPnml(0.100) sigInml(0.100)
  Utfil(Mcllist1.Txt) Xutfil(Mcllist2.Txt) parmfil(Mclparm01.Txt)
ENDPROC
```

An example with selection of small claims in preparatory stages in Proc Data

```
/* The first lines of Resb0.txt. Company and business line written
   together with comma in between.
28,30 19940103 19940121 000002840;000006000;000005000;000005430;
28,30 19940109 19940225 000004247;000001500;000001300;000007250;
28,30 19940109 19940616 000046423;000125645;000033000;000051675;
*/
Include C:\Rapp\Rpp\Init.Rpp
Proc Data textfil(Resb0-sma.Txt) var(f1) ;
infil fil(Resb0.txt) var(f1 $ Ures Kres Utbet) dlm(';')
  dvar(Skkost = Kres + Utbet) urval(Skkost < 1000000) ;
Endproc
Proc Data textfil(Resk0-sma.Txt) var(f1) ;
infil fil(Resk0.txt) var(f1 $ Ures Kres Utbet) dlm(';')
  dvar(Skkost = Kres + Utbet) urval(Skkost < 1000000) ;
Endproc
Proc Restri
  infil(C:\CKOD\Resb0-sma.Txt) utfil(C:\CKOD\Resb30t.Txt) argname(Bb) urval(4_2_30_30)
  Firstclaimtime(19940101) Lastclaimtime(20090630) Lastpaytime(20090630) Timeconv(h)
Endproc
Proc Restri
  infil(C:\CKOD\Resk0-sma.Txt) utfil(C:\CKOD\Resk30t.Txt) argname(Bb) urval(4_2_30_30)
  Firstclaimtime(19940101) Lastclaimtime(20090630) Lastpaytime(20090630)
  Timeconv(h)
Endproc
Proc Reschl Infilbet(C:\CKOD\Resb30t.Txt) Infilskk(C:\CKOD\Resk30t.Txt)
  svansskk(a) svansbet(a) expskk(1) expbet(1)
  nsvansskk(16) nsvansbet(16)
  dnmseinfskk(12) dxmseinfskk(0) dnmseinfbet(12) dxmseinfbet(0)
  //cnmseinfskk(10) cxmseinfskk(2) cnmseinfbet(10) cxmseinfbet(2)
  tidssort(h) arsranta(3.5) Munich sigPnml(0.100) sigInml(0.100)
  Utfil(Mclt7-1.Txt) Xutfil(Mclt7-2.Txt) Utfilipc(Mclt7-3.txt)
Endproc
Proc Graf visa listfilut(Mclt7-3.txt) Endproc
```

Graphics

Proc Graf can be used with parameter pos[], where listfil() = utfil(), xutfil(), futfil(), fxutfil(). From column B the startpos is increased by 15 for each column.

Startpositions s in pos[... .. s text]

Undiscounted values

```

16  B = Paid
31  C = Claimhandlers reserve
46  D = IBNS:Incurred
61  E = Tail:Incurred
76  F = B+C Incurred
91  G = B+C+D+E Totcost
106 H = F accumulated 1 year
121 I = G accumulated 1 year
136 J = IBNS:Paid
151 K = Tail:Paid
166 L = J + K

```

Discounted values to 'now' = end of last claim period in triangle

```

181 M = IBNS:Paid
196 N = Tail:Paid
211 O = M + N

```

Discounted values to 'then' = beginning, middle or end of respective claim period as determined by dt()

```

226 P = Paid
241 Q = IBNS:Paid
256 R = Tail:Paid
271 S = P + Q + R

```

User data in userfil()

```

286 T = User01 (User01, User02, ... are column headings in Excel unless names are given)
...

```

Example:

```

Proc Graf listfil(Resout1.Txt) pdfil(al.Pdf) visa
  pos[0 1_pie_1% 46 IBNS-reserv 16+31 Känd$skadekostnad$sexkl$IBNS 286 Bokförd$reserv]
  pos[0 A_vbar_color=black,green,red,blue_pattern=x1,l2,solid,solid
       76 Känd$skadekostnad$sexkl$IBNS 46 IBNS-reserv
       61 Svans$över$IBNS 211 Diskonterad$reserv$still$'nu']
Endproc

```

```

Proc Graf listfil(Resout1-mack.Txt) pdfil(Resout1-mack-skk.Pdf)
  genhead('Prediction intervals Mack (1999) KNOWN CLAIMCOST') sw boxplot
  pos[ 0 A_vbar_pattern=solid_color=green 16-(1.96*54) $ 16+(1.96*54)
       95$%$with$normalapprox$(+1.96*stderr)]
Endproc

```

Userfil() instructions

Take an Utfil from a previous run and modify it. Example of the left part of a summary block up to and including the field Paid:

Claimperiod	B Paid
201201 1 :]	25852045
201202 2 :]	25001332
201203 3 :]	25425739
201204 4 :]	25241922
201205 5 :]	23878859
201206 6 :]	23833887
201207 7 :]	22644009
201208 8 :]	19093987
201209 9 :]	17705484
201210 10 :]	12636243
201211 11 :]	9955345
201212 12 :]	5118419
Total	236387271

Delete the Total line and everything to the right of :]. You can blank out or keep the period index (1, ... ,12) and :]. Put your variables after the location of :]. Make as many blocks as there will be argument values with one block per value. You can omit the block for the Total. Eg if there are two argument values denoting for example two regions, make two blocks with your variables for each one. The report will have three blocks, where the Total will have the sum of your variables. Make at least one line without period data

between the blocks. You can put the argument values above the blocks for clarity. Example - the right alignment and the headings are only for clarity. They are not necessary.

```
[Region: 07
Claimperiod    Premium Clsetcost
201201    1 :] 34567890    3456789
201202    2 :] 45678901    3333333
201203    3 :] 56789012    2222222
201204    4 :] 12345678    1111111
201205    5 :] 23456789    2222222
201206    6 :] 87654321    3333333
201207    7 :] 76543210    4444444
201208    8 :] 65432109    5555555
201209    9 :] 54321098    6666666
201210   10 :] 22223333    5555555
201211   11 :] 55556666    4444444
201212   12 :] 33331111    3333333
```

```
[Region: 38
Claimperiod    Premium Clsetcost
201201    1 :] 14567890    1456789
201202    2 :] 25678901    2333333
201203    3 :] 36789012    3222222
201204    4 :] 42345678    4111111
201205    5 :] 53456789    5222222
201206    6 :] 67654321    6333333
201207    7 :] 56543210    5444444
201208    8 :] 45432109    4555555
201209    9 :] 34321098    3666666
201210   10 :] 22223333    2555555
201211   11 :] 15556666    1444444
201212   12 :] 3331111    333333
```

Proc Restea

Extends a triangle with chain ladder and aggregates. Triangle Extend Aggregate. Intended use is for calendar year reserving before the year is over. Example:

1. A triangle file with monthly values 200401-200910 exists, ie, 70 claim and development periods. The proc then extends to 72 claim- and developmental periods, so that 200911 and 200912 are added as further periods with 0-values. Then the development factors for the 71st and 72nd development period are set to 1. Then this is aggregated to one out-period per 12 in-periods. Example:

```
Proc Restea infil(f1.txt) utfil(f2.txt) inpad(2) inpperutp(12) Endproc
```

2. A triangle file with two-month values 200401-200910 exists, ie, 35 claim and development periods. The proc then extends to 36 claim- and developmental periods, so that 200911+200912 is an additional period with 0-value. Then the development factor for the 36th period of development is set to 1. Then this is aggregated to one out-period per 6 in-periods. The inputfile f1.txt is outputfile. Example:

```
Proc Restea infil(f1.txt) inpperutp(6) Endproc
```

Proc Restri

Example:

```
Include E:\Riskanalys\Rapp\Init.Rpp
```

```
Proc Restri Infil(Indat1.Txt) Utfil(Triangel.Txt) Argname(Produkt) Urval(1_2_2_50)
```

```
Firstclaimtime(19910101) Lastclaimtime(20051231) Lastpaytime(20051231) Timeconv(q)
ENDPROC
```

```
Proc Restri Infil(Bichsim5x.Txt) Utfil(b222.Txt)
```

```
segments(71_1) argname(Season)
```

```
Timeconv(M) Firstclaimtime(20090101) Lastclaimtime(20091231) Lastpaytime(20091231)
```

```
Colnpaybel(2) Colnpaytime(3) Colnsettime(4) Colnreptime(5) Colncltime(6)
```

```
RDC maxW(999) quantlimit(999) quantno(100)
```

```
ENDPROC
```

Syntax, where parameters in [] are optional, and | indicates alternatives. Any order. All parameters are as always in Rapp case independent.

```

PROC Restri
infil(fil1) utfil(fil2) [argname(name)] [Firstclaimtime(timepoint1)]
[Lastclaimtime(timepoint2)] [Lastpaytime(timepoint3)] [Colncltime(colnno1)]
[Urval(m1_m2_a3_a4_n1_n2_b3_b4)] [Timeconv(D|I|M|Q|H||Y|)]
[Pricefile([fil3[/interestratefile]])]
[Basepricetime(timepoint4)] [Colnpricetime(colnno2)] [Colnprice(colnno3)]
[colnreptime(colnno4) utfild(fil4) utfiln(fil5)]
ENDPROC

```

Creates one or three triangle files from a textfile with one line per payment amount or change amount of known claim cost. A line must contain claim time, payment time and payment amount blank-, tab- or semicolon-separated in that order. Rapp determines if the fields are separated by spaces, tabs or semicolons from the presence of a semi-colon or tab or not. Time can be given as date YYYYMMDD, month YYYYMM, quarter YYYYQ, halfyear YYYY00H, year SSAA or non-negative integer indices. If argname() is given, by default the first column is taken as a distribution argument, such as product code. If the distribution argument is not the first delimiter separated word, then you can enter sparg() and nparg() or segments() for the starting position and number of positions for it. If colncltime() is given, claim timepoint is assumed to be in that column. Otherwise, in the first column if argname() is not given, else the second column.

Three files are created if Colnreptime() but not RDC and not Schnieper is given, to make possible an analysis according to R. Schnieper "Separating true IBNR and IBNER claims", Astin Bulletin 1991, vol 21(1), p. 111-127. See below under Colnreptime(). If parameter Schnieper is given, the parameter estimates and reserve predictions are computed in Rapp.

Parameters

Argname()	Indicates that a distribution argument with that name shall be used for splitting into several triangles.
Basepricetime()	Time period to which payments are up-indexed using price index in the textfile PriceFile. Eg Basepricetime(2007) converts all amounts. in the inputfile to the year 2007 from the price level at payment time. (This is, however, perhaps not appropriate when there are changes in known claim cost in Infil, for those changes may be affected by predicted inflation.)
Benktander	See Proc Bich.
Bornhuetter-Ferguson	See Proc Bich.
Cape-Cod	See Proc Bich.
Colncltime()	Column no (blank-, tab- or semicolon-separated) for claim time.
Colnpaybel()	Column number for payment in Infil().
Colnpaytime()	Column number for payment time in Infil().
Colnprice()	Column number (space separated) of the price index in the textfile Pricefile.
Colnpricetime()	Column number (space separated) for time period in Pricefile. The time period can be given in any format of YYYYMMDD, YYYYMM, YYYYQ, SSAA; it is converted by Rapp to format according to Timeconv.
Colnreptime()	If given, the column no (blank-, tab- or semicolon-separated) for customer claim report time. The format shall be as the format of claim time. Ie YYYYMM if claim time is given as month in that format, YYYYMMDD if claim time is given as date in that format, etc. If also utfild() and utfiln() are given (unless also RDC or Schnieper are given) utfild() will contain the D-triangle(s) and utfiln() the N-triangle(s) according to Schnieper (1991). The claim report time will determine this; if it is within the development period, the payment will be added to the N-triangle for that development period.
Colnsettime()	Column number for settlement-time in Infil(). Necessary for RDC.
Delimiter()	Delimiter as in Proc Taran. If not given, then delimiter is

Dlm() determined from the inputfile's first line. Is there a tab or semicolon, then this becomes the delimiter.

DiscountToClaimperiodRate() See button Inf01 in first screen of Rappmenus.Exe.

Firstclaimtime() Lines with claim-time < Firstclaimtime are not included.

Futureprices See button Inf01 in first screen of Rappmenus.Exe.

Infil() Inputfile with detailed data on payments / claim cost changes.

Lastclaimtime() Lines with claim-time > Lastclaimtime are not included.

Lastpaytime() Lines with pay-time > Lastpaytime are not included.

Listfil() Report listfile for RDC and Schnieper.

Ndec() The number of decimals for increments. Default 0.

Nparg() If given, it is the number of positions for the distribution argument.

Premium() See Proc Bich.

Pricefile() Textfile with price index. A line is eg "2007 290.1" or "2007 290,1".

Priorultimate() Se Proc Bich.

RDC If given the RDC method is employed as in Proc Bich, see this proc. A first variable with a unique claim-ID, Colnreptime() and Colnsettime() must then be present for the infile. Other parameters relevant for object claims in Proc Bich can be used, with -akt optionally removed. Then quadrats with upper development triangles and future prediction triangles by RDC are produced. These can be input to Proc Reschl, which then takes the predictions for good. See example above. Timeconv(month) with month not 1, 3, 4, 6, 12 cannot then be given. A report file as Utfil() in Proc Bich is also created. If Listfil() was given it will be the report file. Otherwise a file with -Report inserted before the point in the name of Utfil().

Resfil() If specified with RDC, this file gets one line per claim in Infil() with claim-ID, claim period (1, 2, ...), report delay w, quantile q, segment and statistical reserve. Also Rapp creates a file with -IB before the extent with IBNR calculation of the numbers of unknown claims, reserve per unknown claim and reserve = product of these two, and RBNS-reserve in the same way. Eg resfil(Res-TPL-200912.Txt) gives IBNR file Res-TPL-200912-IB.Txt. A file with -IBtri before the extent, giving partitions of IBNR and RBNS on development periods, is also created.

Schnieper If given, the method used is the one by R. Schnieper "Separating true IBNR and IBNER claims", Astin Bulletin 1991, vol 21(1), p. 111-127. Colnreptime() is required for this. Listfil() works as for RDC.

Schnieperexposure() A file with exposures E_i for Schnieper's method. If not given or given as D, the first development period reported claim numbers are exposures. See Proc Bich.

Segmname() Synonym for Argname().

Segments() As Segments-akt() in Proc Bich. Alternative to Sparg() and Nparg(). Instead of Sparg(63) Nparg(5) you can give Segments(63_5).

Sparg() If given, the starting position for the distribution argument.

Timeconv() Time conversion. The times are converted from the format of indata to the right period length and notations in the triangles.
A no conversion
D date YYYYMMDD
H half year YYYYHj, j=1,2
I integer index

M month YYYYMM
Q quarter YYYYQj, j=1,2,3,4
T tertial (four months) YYYYTj, j=1,2,3
Y year YYYY (default)
Mnemonics for A is that it means ASIS ("as it is"). One can also give period length in number of months. For example Timeconv(4) is the same as Timeconv(T), ie four months. Any integer > 0 accepted for chain-ladder. For Schnieper and RDC only 1,3,4,6,12 are supported for now.

Urval() If the strings a3 and a4 have the same length m2, then only lines with 'a3' <= X <= 'a4' are included, where X is a string that is at start-position m1 and m2 positions forward. Otherwise a3 and a4 are interpreted as integer numeric values, between which the numeric value at start-position m1 and m2 positions forward must be. X is positional, not determined by column number for space separated columns. Likewise optionally for n1, n2, b1, b2. If given, the selections are combined with logical AND.

Utfil() Outputfile with triangles.

Utfild() See above under Colnreptime().

Utfiln() See above under Colnreptime().

When Lastclaimtime() is not the end-date for a normal period according to Timeconv(), the payments (known-claim-cost-changes) are summarized on broken periods, where the last one's last date is Lastclaimtime(). Then the input is assumed to be dates on the form YYYYMMDD. Eg at Timeconv(Q) for quarter, Lastclaimtime(20090331) is the end-date of a normal period, but Lastclaimtime(20090131) is not. The last period is then 20081101-20090131. Then Firstclaimtime() should be the start of a broken period of the same displacement, such as Firstclaimtime(19940201). Period names in the outputfile are then given as the first included month followed by an f for following. In the example 199402f, 199405f, ... , 200811f.

Lastpaytime() may be later than the Lastclaimtime(), for example:

Firstclaimtime(19940101) Lastclaimtime(20081231) Lastpaytime(20090630) Timeconv(h)

Example semicolon separated input fields

02 01;19880523;20020715; 514
02 02;19880101;19880103; 302

Proc Rskilj

Example:

Proc Rskilj listfil(C:\xx\lista1.txt) Endproc

Listfil must be an outputfile listfil from Proc Taran, or an edited file with layout as a listfile from Proc Taran. In each argument block are placed at the far right under the double-bars (====) the measures Rskilj and Rsjbst. The denote risk separation capacity, on the one hand Rskilj for real portfolios and on the other Rsjbst, which is what it would be at quite evenly distributed portfolio with the same medelfbel or medelndur. If the risk premium factor estimates have large sampling errors, it is advisable to use a listfile where the "Factors riskprem" were replaced with estimates modified with assessments. This is an approximate method that works well, however, unless some arguments covary very strongly with each other. In the latter case, these measures exaggerate risk separation ability. Bengt Eriksson's method is better from the covariation viewpoint but is, on the other hand, difficult to modify in the light of sampling errors and using assessments. Neither is there an equivalent to Rsjbst in Bengt's method. Bengt's method is implemented as parameter Rskilj() in Proc Taran.

Proc Sample

Example:

Include C:\Rapp\Rpp\Init.Rpp

Proc Sample infil(Skador.Txt) utfil(Skador-urval.Txt) firstobs(2) size(100) Endproc

Makes a simple random sample of size lines from a textfile in Windows, according to a random number generator I found on the Internet described as the best in the world. A "long period (> 2×10¹⁸) random number generator of L'Ecuyer with Bays-Durham shuffle and

added safeguards". It is not the fastest of all acceptable generators, but fast enough; 10 million random numbers uniformly distributed in (0.1) takes a second to produce on a computer with 2GHz processor. Parameter firstobs has same meaning as in other places.

Proc Sas

Example:

```
Include C:\Rapp\Rpp\Init.Rpp
PROC Sas
libname Bibl 'F:\B99STI\SAS\Data';
filename in 'skattningar.txt';
DATA Bibl.Rappskattn;
  INFILE in DLM=';' FIRSTOBS=2;
  INPUT Argnamn $ Nivnamn $char10. Anr Ninr Dur Fbelndur Prem Antskad Skkost
         Ospmu Osp Basff Basfm Basfr Faktf Faktm Faktr Ospf Ospm Ospr Tarf
  ; RUN; quit;
Endproc
```

In Proc Sas you write code as in a SAS program.

Proc Sasin

Example:

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Sasin libname(C:\xx\sas\data) tabell(parmtab1) textfil(parmfill1.txt) lrecl(500)
satser(if Antskad ^= 0; keep Anr Ninr Dur Prem;)
firstobs(2) dlm(';') Var(
  Argnamn $char20. @32 Nivnamn $char10. Anr Ninr Dur R Fbelndur Prem Antskad Skkost
  Ospmu Osp Basff Basfm Basfr Faktf Faktm Faktr Ospf Ospm Ospr Tarf
) rboot endproc
```

```
Proc Sasin libname(C:\xx\sas\data) tabell(parmtab1) textfil(parmfill1.txt)
satser(if Antskad ^= 0;) Endproc
```

```
Proc Sasin libname(C:\sas) tabell(Ag1) textfil(Ag1.txt) ctyp(7 9 21 22) Endproc
```

Imports a textfile to a SAS table. Parameter libname is given without quotes. A delimiter can be given as above. Delimiter, firstobs() and the list of variables var() may be omitted if the textfile has a first header (eg if it is created with Proc Match, Proc Sum or Proc Data with parameter Headerline). Rapp then reads the first and second lines to determine names and datatypes. If an alphanumeric variable can have a numeric value in the textfile's second line, you indicate with parameter ctyp() that it is alphanumeric. The last example indicates that the 7th, 9th, 21st and 22nd variables are alphanumeric.

In satser() can be written an optional set of SAS statements, which is executed after loading the textfile into SAS but before addition to the table. Above are included only lines with Antskad not 0. It is understood that any parentheses in the SAS-statements will, as usual, appear as (followed by), such as in conditions. A) alone without a corresponding (leads to Rapp stopping the generation of SAS-statements there.

At some lengths for Argnamn (above with \$char20.), the scanning of Nivnamn will be wrong if you do not position to Nivnamn with @32. A bug in SAS (2009 - possibly corrected since then). Nivnamn always begins in column 32 in a semicolon delimited textfile produced by Proc Taran. With the parameter RBOOT the type declaration R can occur in var().

Proc Sasut

Example:

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Sasut libname(C:\xx\sas\data) tabell(tabell1) textfil(Rappdat1.txt) lrecl(500)
satser(if zzz = 'ABC';) var(bolag dur R riskpremie skfrek msk kvadr a01 $ a02 a03)
dlm(';') rboot endproc
```

Exports a SAS table to a textfile. Parameter libname given as in Proc Sasin. The advantage of the proc rather than produce an entire SAS program and use Proc Sas is that it becomes easier to make a textfile with fields separated by a delimiter such as semicolon. It is a bit cumbersome to do it in pure SAS. Unlike Proc Sasin one can besides a character within quotes provide a binary code, such as 9 for the tab character with delimiter(9), in the same manner as in Infiler in Proc Taran. Type declarations \$ may be

found in var(). Type declaration R is allowed, if the parameter RBORT is given in the proc. If not, Rapp believes that R is a SAS variable.

In satser() can be written an optional set of SAS statements, which is executed before writing to textfile from SAS. Above are included only lines with zzz = 'ABC'. Variables can be changed and new variables can be made here and be included in var(). Regarding possible parentheses in the SAS statements, Rapp assumes as above in Proc Sasin.

Example 1:

```
satser(if hv_pool = ' ' then hv_pool = 'N';
if boyta = 0 then biprom = 0; else biprom = floor(1000*biyta/boyta + 0.5);)
```

Example 2:

```
satser( if hv_pool = ' ' then hv_pool = 'N';
  if boyta = 0 then biprom = 0; else biprom = floor(1000*biyta/boyta + 0.5); )
```

Proc Sort

Examples:

```
Include C:\Rapp\Rpp\Init.Rpp
```

```
Proc sort Stats sortin(file n:o 1.txt) sortout(f2.txt) parms(12 7 nd 1 5) Endproc
```

```
Proc sort sortin("f1.txt" "f 2.txt") sortout(f2.txt) parms(12 7 nd 1 5) Endproc
```

Sorting of textfile with fixed positions for sorting fields. Initial order for lines with the same value of the sort keys can not be guaranteed.

Parameters

Collatingsequence() Optional. Either a string with a collating sequence, where the first word starts with a colon (:), or a file with a collating sequence, divided in arbitrarily many lines. Rapp reads the sequence either as characters written as they are, binary with three characters, or hexcode. Three characters are needed for binary not to be taken as hexcode. Those characters are placed first in a collation sequence. Thereafter are put the characters not given by the user, in their original order. Sorting fields marked with n (see below) are converted with the help of the collating sequence before sorting. E.g, if the character & is written first, then the lines whose first sort field begins with & get to be on top. Represent blanks with its hexcode 20. For example

```
Collatingsequence(: & 20 61 A 246 :)
```

```
Collatingsequence(:26 20 a 41 ö 3a)
```

These two examples accomplish the same thing.

Parms() Required. A sequence of starting positions and number of positions, in sort order. After the number of positions can be written a third parameter for the sort field as d, n, nd, dn, optionally with lowercase or uppercase. Here d means sorting in descending order and n that sorting shall be done with å ä ö Å Ä Ö according to Swedish standards (National) if Collatingsequence() is not given. If Collatingsequence() is given, n means sort according to this.

Sortin() Required. Infile(s). If multiple files are given, give each in double quotes.

sortout() Required. Outfile, can be the same as infile.

Stats Optional. Gives file and RAM statistics, including longest line length.

Proc Split

Splits a file by values (maximum 30 characters) in a given column. That column can be given with splitcol() or with var() and splitvar(), where var() follows the syntax of Proc Taran, Proc Data et al. Delimiter is determined by the first line of the inputfile. The first observation that shall be read is given with firstobs(), default 1.

Example:

```
Proc Split infil(fil.txt) utfil(abc .txt) splitcol(7) firstobs(2) Endproc
alternatively
```

```
Proc Split infil(fil.txt) utfil(abc .txt) splitvar(F12) firstobs(2)
  var(f1 $ f3 f5 4 f77 r v9 C x14 f12 a0 a7) Endproc
```

Creates abc0.txt abc2.txt abcVV03M370.txt abc36.txt if 0 2 VV03M370 36 is in column 7.

Proc Sum

Example:

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Sum Textfil(Skadsum.txt) Headerline stats delimiter(9) Sumvar(Antskad Skkost Kvadr);
Infiler fil(Skadormatchade.txt)
  var(Skkost Xkod Antskad Kvadr Bilmärke $ Kon Geografi $ Ålder Skadedat Könåld Fromdat)
  Key(Geografi Ålder Kon Bilmärke) firstobs(2) delimiter(';') ;
ENDPROC
```

or

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Sum;
Infiler fil(Skadormatchade.txt)
  var(Skkost Xkod Antskad Kvadr Bilmärke $ Kon Geografi $ Ålder Skadedat Könåld Fromdat)
  Key(Geografi Ålder Kon Bilmärke) firstobs(2) delimiter(';') ;
Utfil fil(Skadsum.txt) Headerline stats delimiter(9) Sumvar(Antskad Skkost Kvadr);
ENDPROC
```

Creates an aggregated file Skadsum.Txt with fields:
Geografi Ålder Kon Bilmärke Antskad Skkost Kvadr

The file has one line per combination of (Geografi,Ålder,Kon,Bilmärke) [= (geography, age, gender, carmake)] and contains the sums of Antskad, Skkost and Kvadr [= square] in Skadormatchade.txt for that combination. The inputfile is the outputfile that was made in the example of the Proc Match.

The parameters of the main clause Proc Sum in the example has the same meaning as like-named parameters in Proc Match. Instead of Sumvar() one can write Var() as in Proc Match.

Other parameters

q	Optional. Indicates that the summation shall be made in memory with hashing
quick	instead of reading with break logic of a sorted file. The summation file's keys and summands are placed in RAM, which must be sufficiently large. If it is not, Rapp will interrupt with a message about it. Given that the memory requirement can be met, quick summation is usually much faster than sort/break. To ensure that the outfile is sorted on the keys, give parameter sort. Compare with parameter q = quick in Proc Match.
sort	The outputfile is sorted on the keys.

The syntax for the statement Infiler is the same as for the statement Transfil in Proc Match except Timekey(). That is to say as for the statement Infiler in Proc Taran plus Key parameter() and the elimination of the parameters associated with utfilink(). Dvar parameter() can be used. Statements arg(), arrays and functions ind0() and inx() can be used, as in Proc Data.

Proc Svg2co

Examples:

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Svg2co svgfil(Komm.svg) coordfil(Sverige-kommun.Txt) kk idnum ENDPROC
Proc Svg2co svgfil(Fors.svg) coordfil(Sverige-forsaml.Txt) ff idnum ENDPROC
// 1. Points (circle) - use Idfil() as a symbfil() for Proc Map.
Proc Svg2co svgfil(Slport.svg) coordfil(Slport.Txt)
  ID4prefix(Orte) Idvar(x y typ namn) Idfil(Slport-ID.Txt) visaattribut
ENDPROC
// 2. Lines (roads, railroads, rivers)
Proc Svg2co svgfil(Slpvg1.svg) coordfil(Roads1.Txt)
  Left2RightText fnonl ID4prefix(Road) Idvar(ID IDold typ)
ENDPROC
Proc Svg2co svgfil(Slpvg2.svg) coordfil(Roads2.Txt)
  fnonl ID4prefix(Road) Startno(105) Idvar(ID typ)
```



```

ENDPROC
Proc Svg2co svgfil(Slpjv.svg) coordfil(Railroads.Txt)
  fnonl ID4prefix(Rroa) Idvar(ID)
ENDPROC
Proc Svg2co svgfil(Slphy.svg) coordfil(Slphy.Txt)
  Left2RightText fnonl ID4prefix(Rive) Idvar(ID namn)
ENDPROC
// 3. Areas
Proc Svg2co svgfil(Slpva.svg) coordfil(Slpva.Txt) ID4prefix(Vatt) Idvar(ID namn) ENDPROC
Proc Svg2co svgfil(Slpmk.svg) coordfil(Mark.Txt) ID4prefix(Mark) Idvar(ID) ENDPROC
Proc Svg2co svgfil(Slpto.svg) coordfil(Slpto.Txt)
  ID4prefix(Tato) Symindex(7) Idvar(ID x y Symindex namn befolkning)
ENDPROC

```

Converts an SVG file (Scalar Vector Graphics) to a coordinate file according to the requirements of Proc Map. Creates also an ID-file, eg Sverige-forsaml-IX.Txt. You can edit it eg for better names.

Parameters

Alfa2num()	Optional. Can be given as one or two numbers, alfa2num(b) or alfa2num(a b). For example alfa2num(0.001) and alfa2num (0.2 0.0001). If given, ID is converted to numerical values a+b, a+2×b, a+3×b, a+4×b, ...
Attribut()	Optional. Can be given arbitrarily many times for attributes in svgfil. Eg attribut(name) if svgfil contains attribut:name, for example as in attribut:name="Steiermark". These will be given last in the ID-file. To know the attributed, use Proc Copy with for() and forcol() to make a smaller file which can be inspected in Notepad.
Ff	If given svgfil() is assumed to have been made from Sverigel000plus\ArcView8.x\slpfg.shp and relate to the Swedish parishes. Cannot be used with parameter Idvar().
Fnonl	If given no final record is created with x, y as the first per ID, segment.
Id4prefix()	If given ID will begin with these four characters in upper case, after which an 8-digit number starting with Startno() follows. Has priority over Alfa2num().
Idnum	If given, any non-digits are removed from the ID in svgfil.
Idfil()	If given the name of the output ID-file. If not given, the ID-file will have name as Coordfil() with -IX before the extension, eg Slport-IX.Txt.
Idvar()	If given, it has priority over Attribut(). Gives the variables in the output ID-file. The variables that can be given are ID, IDold, Symindex, x, y and any attributes. x and y are the midpoint x and y. IDold is the ID of the SVG file. ID is IDold if neither one of Alfa2num() or Id4prefix() was given. Otherwise the ID determined by those parameters.
Infil()	Input SVG file.
Kk	If given svgfil() is assumed to have been made from Sverigel000plus\ArcView8.x\slpkn.shp and relate to the Swedish municipalities. Cannot be used with parameter Idvar().
Left2RightText	If given, the order of certain items is swapped so that embedded texts are read from left to right. Refers to roads, rivers, etc. in Proc Map. For such files Fnonl should be given.
Startno()	See under Id4prefix(). Default 1.
Symindex()	An arbitrary integer that can be a variable in Idfil().
Util()	Outfile with ID, segment, x, y. Midpoint x and midpoint y are added for the first line of an ID.
Visaattribut	If given the attributes are shown on the screen. Can only be used with parameter Idvar().

Column numbers for ID, segment, x-coordinate, y-coordinate will be 1, 2, 3, 4. The first line for an (ID,segment) will contain in column 5 and 6 $(\text{max-x-coordinate} + \text{min-x-coordinate})/2$ and $(\text{max-y-coordinate} + \text{min-y-coordinate})/2$. That usually provides an approximate center of the area (ID,segment) where labels for ID can be written if `labelidcol > 0` is given in Proc Map. For hook-shaped areas, however, the point might fall outside the area.

To convert an shp-file (Shape-file) to an svgfil, use free programs on <http://www.carto.net/papers/svg/utils/shp2svg>

There is found a manual for these two programs (subject to change):
<http://www.carto.net/papers/svg/utils/shp2svg/ogis2svg.exe>
http://www.carto.net/papers/svg/utils/shp2svg/binary_win/shp2pgsql.exe

Proc Taran

Synonym: Proc Jung

Example:

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Taran listfil(a1.txt) textfil(a2.txt) nummetod(N) rub62 'Test example n:o 01';
infiler fil(Rappdat1.txt) fil(Rappdat2.txt) delimiter(9)
  var(bolag Kod Bilmärke $ Geografi $ dur fbel/dur antskad skkost Kvadr Byggtyp a2)
  urval(skkost>0 & bolag = 4 10 & kod ^= 7) ;
arg(Code) rub30 'Code' rub110 'Some code' bas(1) antniv(7);
arg(A2) rub30 'Kod2' rub110 'Another code' bas(1)
  niv('Kod 01' 1.00 'Kod 02' 1.17); /* Class name 'Kod 01' etc max 10 characters. */
  /* 1.00 och 1.17 tariffen */
arg(byggtyp) rub30 'Btyp' rub110 'Bebyggelsetyp' bas(1)
niv(
  ( 4,1-9 'Övrigt' 1.00)
  ( 1,1:2 'Enplanshus' 1.00)
  ( 2,3-4 'Tvåplanshu' 1.32)
  ( 3,5 'Treplan' 1.40)
  ( 2,7 9 'Tvåpl spec' 1.32) ) ;
arg(Bilmärke) rub30 'Bilmärke' rub110 'Bilmärke Volvo eller Jaguar' bas(2);
arg(Geografi) rub30 'Tät- eller glesbygd' rub110
'Enligt SCB:s klassifikation från 1998' bas(0) tar(1 1.23 0.86 0.56);
```

TEXT

"Factors frequency" and "Factors riskprem" have been obtained from a system of equations, which clears the influence of other arguments than the tabulated. "Factors riskprem" is suitable for rating, albeit with regard to the column Rfactucpct (uncertainty in %). Few and/or unevenly size distributed claims give a high percentage of uncertainty, making the risk premium factors less usable for commercial rating.

Endproc

The proc consists of a number of statements, each with different parameters in any order.

Main statement directly after Proc Taran, ending with a semicolon

Parameters specified in the statement: they can be given in any order:

- alphasize() Optional. Default 2000 for 32-bit and 100000 for 64-bit. Minimum value 10. All alphanumeric fields are truncated to a maximum of alphasize characters in string constants and selection conditions in the Rapp-program and the outputfile. If less than 10 is set, it will be 10. Arguments always have at most 10 characters.
- arr() Optional. Sets up a two-dimensional floating point number array. The minimum array index value is 0. The numbers are separated by commas or spaces. Intended for eg norming or price-indexing. Example:

```
arr(0(12.3,14.4, 77.89e-1) 3(1.3 1.45 55 123.67 1.87255))
```

Used in dvar() with arr(index1,index2). For example, arr(0,0) is assigned the value 12.3 and arr(3,2) is assigned the value 55 above. If an element is not assigned a value, there will be a warning, and value 0.
- assm,assmL Optional. Provides association measures. AssmL gives list form. See Appendix 7.

- `begperiod()` Optional. First date of the period for which duration shall be calculated, if any of `durb1()` or `durb2()` is used for calculating duration in `dvar()` in the statements `Infiler`. Example 20020101. If not given, `dur` is calculated from the beginning of time. However, if a sequence of variable `firstdate` and `enddate` [see below under `endperiod()`] are to be used, then this can be done in parameter `dvar()` on the form
`durb1(frdk + firstdate/100000000 , todk + enddate/100000000)`
and likewise for `durb2()`.
- `BE-metod`
`EVW` Optional. Can be given at the same time as `S-GLM` and `Tweedie()`. If given, then before the risk premium estimates are made the observations are weighted with estimated real variances per tariff cell according to a model proposed by Bengt Eriksson. If the `listfil` is given as `nnn.Txt`, then a `listfil` is produced without the parameter and is given as `nnn-orig.Txt`. For example, if `listfil(LL01.txt)` was given, then `LL01.txt` will contain the estimates worked out with `BE-metod` and `LL01-orig.Txt` will contain the estimates worked out without `BE-metod`. All columns in the `listfil` except `Factors riskprem` are the same in the two `listfils`. Eg the variance estimates have not been adjusted. A `textfil` given with `textfil()` will contain parameter estimates from the `-orig listfil`. Recommended only for at most three arguments.
- `chi2min` Optional. Gives the multiplicative solution that minimizes the exposure-weighted sum, over all tariff cells, of

$$[(\text{observed riskprem}) - (\text{estimated riskprem})]^2 / (\text{estimated riskprem})$$
The method is historically interesting, but does not provide useful solutions. Restrictions as for `nummetod(K)` described below.
- `complfil()` Optional. A file with content as the insurance `infile(s)` and three added fields last, namely multiplicatively computed expected number of claims, mean claim and multiplicatively computed expected claim cost. Delimiter between fields as the `infile's`. Only lines that were selected for tariff analysis, i.e. with excluded lines removed.
- `crossarg()` Optional. The arguments to be cross-tabulated in `crosslist()`. Here the tabulation order has the first given argument innermost etc. If `crosslist()` but not `crossarg()` is given, then the first four arguments are tabulated, or those who exist if there are fewer than four arguments.
- `crossblocks()` Optional. Number of blocks ,for different values of the next innermost argument, per page. If not given `Rapp` calculates an appropriate value.
- `crosslines()` Optional. Number of lines per page. Default 78. `Crossblocks` is computed from that.
- `crosslist()` Optional. File in text format with univariate cross-tabulation of claim statistics for a maximum of four arguments given in `crossarg()`. Without parameter `crossvar()` the tabulation gives the univariate columns in the usual `listfile listfil()`, and then `Proc Graf` works. Run a few variations of this parameter and `crossarg()` to see how it works. Example:
- ```
Proc Taran crosslist(cross01.txt) crossarg(Bolag Hustyp) nummetod(u) ...
```
- The listfile is made to PDF with `Proc Graf`, just as other listfiles, eg
- ```
Proc Graf listfil(cross01.txt) pdfpil(a.pdf) s u visa ENDPROC
```
- The parameter can be used with all other parameters. However, if the only purpose of the execution is to obtain a cross-tabulation, then `nummetod(u)` should be set to minimize run time.
- `crossout()` Optional. Indicates together with `crossvar()` the variables to be tabulated and an optional format indication on the form `,#ofchars.#ofdecimals`. If a format indication is not specified, it will be 0 decimals and the maximum number of characters found for printing of the variable, plus 1 for the initial blank. Number of chars shall include an initial blank and a possible decimal point. With a comma character in a format indication, thousands are comma separated. In the report, the level names of the innermost argument are given in 11 characters that are preceded by a blank.

This format shall not appear in crossout() but is to be understood.

- crossrub() Optional. Not more than four headers for the blocks in crosslist() when crossvar() was given.
- crossvar() Optional. Indicates summation variables. If there are two infile-sets in two statements Infiler they must be prefixed by 1. or 2. where 1 or 2 gives the set of files that the variables are in. See example below. Furthermore one can give, after all the above summation variables and on the form $bervar = \dots$, computed variables that are functions of the summation variables, of the six special variables Dur Fbel Prem Antskad Skkost Kvadr, and of variables with prefix tot. which are the sum over the innermost argument of the variable after the prefix. Eg tot.1.simulerad_skkost is the sum (innermost argument) of the variable simulerad_skkost in the first set of files. Then $100 \cdot 1.simulerad_skkost / tot.1.simulerad_skkost$ is the percent of the total for respective argument value. The percent for ordinary claim cost is, in the same way, $100 \cdot skkost / tot.skkost$.
- endperiod() Optional. Final date of the period that duration shall be calculated for, if any of durbl() or durb2() is used for calculating duration in dvar() in statements Infiler. Example 20041231. If not given, dur is computed to the end of time. See also above under begperiod() for ways to handle a sequence of variable firstdate and enddate.
- hprec Optional. Gives six decimals in the risk premium factors in the listfil.
- inffil() Optional. File in text format in which the Fisher information matrix and its inverse are put, for claim frequency and risk premium (mean claim), if the parameter is specified.
- listfil() Required. File in text format, where the report will come.
- mellagg() Optional. Values J, N. At J an insurance middle file is aggregated to a new file used in the iterations of the equation solution, but not at N. If mellagg not is given, such aggregation occurs only when Rapp estimates that the total CPU time would shrink by aggregation. Give this parameter if you found that Rapp's assessment of sorting needs to be replaced by your own assessment.
- mse Optional. Produces the square root of an MSE (mean square error) estimate in listfil after the text "MSE-estimate square root:". This MSE estimate is the exposure weighted sum, over all tariff cells, of $[(\text{observed risk premium}) - (\text{estimated risk premium})]^2$. Our experience is that the marginal totals method will mostly give the smallest such estimate. In simulations this was found to be true, even if the Standard GLM method had the smallest expected value of an estimate as this one, but with observed risk premium replaced by true risk premium.
- msemin() Optional. Gives the multiplicative solution that minimizes the exposure-weighted sum, over all tariff cells, of $[(\text{observed riskprem}) - (\text{estimated riskprem})]^2 (\text{observed riskprem})^p$ where the exponent p is given in parentheses, eg msemin(0). With $p = 0$ is minimized the MSE-value described in the preceding parameter. The method, however, will usually not give the minimum expected value of an estimate like this, but with the observed risk premium replaced by true risk premium. Restrictions as for nummetod(K) described below.
- nofreq Optional. Factor smoothing not made for claim frequency. Can be applied to shorten run times for simulations of eg the Tweedie methods' confidence intervals, or to other cases where frequency smoothing is not needed.
- noriskp Optional. Factor smoothing not made for risk premium. Shortens run times for simulations where only the claim frequency is interesting.
- nummetod() Optional. Values:
K = Classical method.
N = Newton-Raphson method is used.
U = No GLM factor smoothing is made, only univariate accounting.
If the parameter is not specified, N is selected if its running time is max

the one of K plus a few minutes, or if K can not be used due to more than 20 arguments or more than 2147483646 combinations. Otherwise K, ie if Newton-Raphson's method would give more than a few minutes longer running time than the classical method and the number of arguments and number of argument combinations fall within the technical limitations of the classical method. If K is chosen the confidence intervals will be according to the coarse adhoc method (1984) instead of the partial adhoc method (MVW). In most cases the difference has no practical significance.

phi() Optional. At s-GLM and the Tweedie model that value is used, if given, as dispersion parameter ϕ . If not given ϕ is estimated with Pearson's χ^2 -estimate for s-GLM. For the Tweedie model the Pearson estimate is used if PEARSON was given, otherwise the expression (8) in Appendix 4. If the sum of squares Kvadr per tariff cell is available, as is the case if individual claims are given as input to Proc Taran, the best estimate is calculated for s-GLM. The Tweedie model Pearson estimate depends on the input data: If idgrupp() is given as in Rapp multiclass analysis, the data are aggregated on the arguments and the ID-variables before the ϕ -calculation. If idgrupp() is not given, data are aggregated on the arguments if there are separate insurance- and claim-files, but if there is a set of combined claim-insurance files, there is no aggregation before the ϕ -calculation.

rskilj() Optional. There one can give arguments for which a measure of the risk separation capability according to Bengt Eriksson shall be calculated, eg Rskilj(KKL Zon Försäkringstagarålder). Rskilj(all) gives all. Shown in listfil just before the table of contents. The measure is $50(\text{sum}|\text{premium difference}| \text{ with and without the argument})/(\text{total premium}) = 100(\text{total premium reallocated because of the argument})/(\text{total premium})$. All factor solutions with one argument removed are made with marginal-totals, but the one with all arguments is made with the selected primary method, such as S-GLM. In the latter case the whole risk premium level flows, so one would need to recompute all total risk premiums to the same value = $(\text{total claim-cost})/(\text{total exposure})$. That has not been done in Rapp. Therefore, it is preferable to select as primary method marginal totals, which gives the same total risk premium automatically. Otherwise, the interpretation is not clear.

rub62 Optional. The main heading in the report, at most 62 characters. Default "Rubrik" (Swedish for Heading). Example rub62 'xxx' and rub62('xxx'), ie optionally with and without parentheses around the title in the way of the examples. The same applies to rub30 and rub110 relating to the arguments.

s-GLM Optional. If given, standard GLM (Poisson claim numbers and gamma-distributed claim amounts) is used, which gives a special listfile and graphs in Proc Graf in English in accordance with the GLM theory. Graphs for the univariate concepts of claim percent and marginal risk premium are shown as relations (ratios) between the levels and the base level. For example, if level 1 is the base level with claim percent 53 and level 2 has claim percent 84, then Claim perct 1.5849 = $84/53$ is shown for level 2.

textfil() Optional. File in text format for the factor estimates and sums in semicolon-separated fields. Imported easily into SAS and Excel.

Tweedie() Optional. If given, a Tweedie model for risk premium with exponent p is used, where p is given in parentheses, eg Tweedie (1.5). Has priority if given at the same time as s-GLM. Claim frequency is analyzed with Poisson loglink. List layout as for MMT method. For graphs of mean claim factors, the square of the coefficient of variation is calculated as the difference of the corresponding squares for risk premium and claim frequency. However, note that I advice against using the Tweedie model in my article <http://www.tandfonline.com/doi/abs/10.1080/03461238.2012.760885>.

Examples of the use of crosslist() with crossout(), crossrub() and crossvar():

```
Proc Taran crosslist(cross01.txt) crossarg(Bolag Hustyp) nummetod(u)
  crossvar(
    2.skkost1000 2.skkost2000 1.premproposal
    clpct = 100*skkost/prem
```

```

ucpct = 100*sqr(kvadr)/skkost
sjrpct1000 = 100*2.skkost1000/skkost /* assumed < 99.5 */
sjrpct2000 = 100*2.skkost2000/skkost /* assumed < 99.5 */
)
crossout(dur 10. prem 1.premproposal 14 clpct 9.2 ucpct .2
sjrpct1000 sjrpct2000)
/* 12345678901 234567890 12345678901234 23456789 12.45 12 12 */
crossrub('          Number          Claim Uncer Sj Sj'
' Bolag          ins yrs Premium Premiumpropos percent tntpc p1 p2') ;

```

For standard-GLM and Tweedie there must be base levels with uncertainty 0. If no base level is set, Rapp sets the levels with largest exposure to base levels. Newton-Raphson method is always used with Fisher's "scoring" method to replace **H** with **-I**. (Rapp is so fast that it no use to experiment with more complex algorithms.)

The Tweedie model's Pearson estimation can cause lost claims if you give an ID group at too low level, eg idgrupp (company,forsnr,hj,lineno,fromdat) where lineno is defined by dvar(lineno = Recno). A claim can have been matched against an insurance version that has never been valid, ie with fromdat = tomdat (up-to-not-incl-date), or against a version in force before the period covered. In those cases the point estimates will be more correct if no idgrupp() is given. It may be useful to first run with idgrupp() to get a ϕ -estimate, and then set that ϕ -value of phi() in a run without idgrupp().

Statement infiler, ending with a semicolon

One or two such statements must be given. If only one, it shall include both insurance- and claim-information. If two, then it is most natural that one contains insurance- and the other one claim-information. Uou can also use the other one for eg only premium information. The words variable and field below indicate the same thing.

String constants can be given as x' followed by hex numbers and concluded by '. Eg x'23BD2BBC' is the same as '#½+¼'.

Parameters specified in the statement; they can be given in any order:

delimiter() Optional. If not specified or empty, the fields in the inputfiles of the dlm() statement shall be blank separated. If a character in quotes is specified, eg. delimiter (';'), then the fields are separated by that character. If a numeric value is givent, such as delimiter(9), then the fields are separated by the character with that binary code. Tab has binary code 9 in Windows. Proc Sasut can export a SAS table to a textfile that is separated by delimiters such as tab or semicolon.

delimiterut() Optional. Delimiter between fields in the outputfile utfilink(). Default dlm() is the same delimiter as for the inputfile. Eg delimiterut (9) for tab separated fields.

dvar() Optional. Provides derived variables from expressions in fields of the inputfile or from other variables in dvar(). A derived variable is a floating point number in 8 bytes, if it is clear that it is numeric from the variables and functions it uses, otherwise alphanumeric with maximum alphasize characters like other alphanumeric variables. These variables are used in the same way as fields in the inputfile, for all purposes: arguments, selection variables, summation variables, ID variables for multiclass analysis. As arguments numerical derived variables are truncated to integers like other floating-point arguments. The names of these variables and the fields they use may not contain + - ! , * / so that arithmetic expressions are not misinterpreted. Otherwise the same rules as for var(). Division by 0 produces result 0. If the terms are complex, Rapp generates internally additional derived variables for intermediate results. If, however, the expressions use no more than three quantities (constants or variables), uses at most one parenthesis pair, and the functions are alone or preceded by minus sign in the expression, then no additional variables are generated. Total number of ordinary and derived variables shall be maximum 600. If not, Rapp interrupts with an error message.

Format

dvar(variable = expression variable = expression ...)

Example, where frdk, todk, birthdate, sex (1/2/3) are numeric fields in the inputfile. As is seen, the order does not matter; Rapp computes anyway Age

before sexage, because sexage requires that these are computed.

```
dvar(Carm3 = Carm2 sexage = 100*(sex-1) + age
      Carm2 = substr(Carmake,1,2) Age = min(99,|[(frdk - birthdate)/10000]|)
      Carmodel1='M-'!!Carm2!!Carmkod Datum = 'D'!!chr(indatum)
      dur = durbl(frdk,todk) Månad = num(substr(Datum,6,2)) )
```

Operators and functions for numerical derived fields

+ - * / []	plus minus times division absoluteval integerpart (floor)
arr()	array eg arr(0,inx(Age))
chr()	converts a number to string
durb1() durb2()	functions for duration calculation
ind0()	ind0(x) = 1 if x = 0, else ind0(x) = 0
indi() two arg	indi(x,y) = 1 if 0 <= x <= y, else 0
indi() three arg	indi(x,y1,y2) = 1 if y1 <= x <= y2, else 0
indp()	indp(x) = 1 of x > 0, else indp(x) = 0
For indsil()	... indslt(), y1 y2 ... are numeric constants, not variables
For indslt()	indslt() is assumed y0 = -oo (minus infinity)
indsil()	indsil(x,y1 y2 ... yn) = 1 if x = yk for some k, else 0
indsik()	<u>Simple use</u> indsik(x,y1 y2 ... yn) = k if x = yk for some k, else 0 <u>Use with special number &IX as first y</u> (&IX = 899441271765394E271, a random number not appearing naturally in data/constants) and a pair sequence. indsik(x,&IX y11 y12 y21 y22 y31 y32 ... yn1 yn2) gives order number of of last pair k such that yk1 <= x <= yk2. Set y11 = y12 if a hit for one single number is appropriate for index i. It is not required that y11 <= yj1 for j > i, iepairs need not be ordered. Returns 0 if no hit. <u>Use with x interspersing values/valueintervals.</u> Then indsik() returns the order number of the last x-delimited sequence where a hit occurs. A colon between two values denotes an interval. Parallel1 to indci() for alphanumeric constants given below. Example indsik(x, x 1 2 3 4 x 5:8 12:15 x 16:22 x 25:28) is 1 if x = 3, 2 if x = 12, 3 if x = 17, 4 if x = 28. Otherwise as above. Returns 0 if no hit.
indsle()	indsle(x,y1 .. yn) = k om y(k-1) < x <= yk for some k, n+1 else. Mnemonics: INDEX in Sequence Less than or Equal.
indslt()	indslt(x,y1 .. yn) = k if y(k-1) <= x < yk for some k, n+1 else. Mnemonics: INDEX in Sequence Less Than.
min() 2-3 arg	minimum of the arguments
max() 2-3 arg	maximum of the arguments
maxv() one arg	maximum value of the argument of records read so far and not excluded. Eg maxv(Yearlyprem) gives the largest occurring yearly premium. Used together with endshow.
maxv() two arg	as above but initiated to -10 ⁷² when 2:nd argument is 0.
perinx()	gives period index. Use: perinx(date,startdate,daysperperiod) Example: daysperperiod = 90 means quarterly periods and perinx then gives quarter index ≥ 1 for date when startdate is the first date in period 1. Thus perinx(20050101,20050101,90) = 1 perinx(20050315,20050101,90) = 1 perinx(20050401,20050101,90) = 2 If date < startdate then perinx ≤ 0. I.e. perinx(20041231,20050101,90) = 0 perinx(20041001,20050101,90) = 0 perinx(20040930,20050101,90) = -1 perinx(20030930,20050101,90) = -5
sumv() one arg	sum of values of the argument of records read so far and not excluded. Eg sumv(Yearlyprem) gives premium portfolio. Used together with endshow.
sumv() two arg	as above but initiated to 0 when 2:nd argument is 0.

The function durb1() refers to 360- and durb2() to 365-day calculation of duration in two arguments fromdat and todk (up-to-not-including-date). Addition of begperiod/100000000 to fromdat and endperiod/100000000 to todk

gives duration within [begperiod,endperiod], eg
durb1(fromdat+.20080101,todk+.20081231).

Functions beginning with ind are used to derive indices of various kinds.

Mathematics for the numerical derived fields

asin() atan() cos() exp() log() log10() loggam() max() min() probnln()
probnorm() sin() tan() sqr() sqrt() $x^{**}y$ where
 $\text{loggam}(x) = \log \Gamma(x)$ $\text{probnln}(x) = \Phi^{-1}(x)$ $\text{probnorm}(x) = \Phi(x)$ $x^{**}y = x^y$
The number π is represented by &PI (case independent - &pi also possible).

Random numbers and simulations

A random number uniformly distributed on (0,1) is obtained by ranuni. With probnln() you obtain then a normally distributed random number, because $F^{-1}(X)$ has distribution function F if X is distributed U(0,1). Example with $E[x3] = 10$, $\text{Var}[x3] = 4$:

```
dvar( x1=ranuni x2=probnln(x1) x3=10+2*probnln(ranuni) )
```

Operators and functions for alphanumeric derived fields

!! Concatenation
cha() cha(str,tostr,fromstr) cha(f1,"AAO","ÅÄÖ") ÅÄÖ to AAO
dat() gives a numerical date YYYYMMDD from a string with the first word
Y-M-D (ISO), M/D/Y (USA) or D.M.Y (EUR). M and D can have one or
two figures. Eg dat("2009-5-7 13:24:45") = 20090507. Will also
come out right, if month is given as Jan, jan, etc, in a field.
indc() indc(str1,str2) = 1 if str1 = str2, else 0
indci() After comma is given a sequence of alphanumeric constants, not
variables.

Simple use

indci(str1,str2) = 1 if str1 exists in str2, else 0
Use with special character x (hex D7) first in str2 followed by a
sequence of alphanumeric values within single quotes, where a
colon between two values means an interval. Enclose the whole of
str2 in double quotes.

indci(str1,"x'value1'[:'value2'] ...") gives order number of
last value/valueinterval equal to/enclosing str1.

Eg: indci(Geo,"x'G-000' 'G-075' 'G-030' :'G-060' 'G-100'") =
1 if Geo = 'G-000'
2 if Geo = 'G-075'
3 if 'G-030' <= Geo <= 'G-060', eg 'G-043'
4 if Geo = 'G-100'
0 if no hit for any of the conditions above

Swedish comparison, ie 'z' is less than 'Å', 'ä', 'ö' and 'Z'
is less than 'Å', 'Ä', 'Ö'. Otherwise the Windows ASCII
character representation order. Ie digits come before letters,
and capitals before lower case letters. Eg '9' < 'A' < 'a'.

Use with x, in addition, interspersing values/valueintervals.
Then indci() returns the order number of the last x-delimited
sequence where a hit occurs.

Eg: indci(Geo,"x'G-000'x'G-075' 'G-030' :'G-060' x 'G-100'") =
1 if Geo = 'G-000'
2 if Geo = 'G-075'
2 if 'G-030' <= Geo <= 'G-060', eg 'G-043'
3 if Geo = 'G-100'
0 if no hit for any of the conditions above

The special uses might not be employed by many Rapp users
directly, but they are employed in the C" application Tariff
analysis.

left() x = left(str,il) gives il chars from the beginning of str. If
str's length is less than il, then x is padded to the right with
blanks. At x = left(str,il,'z') x is padded to the right with the
character z.

len() gives length of the field

num() gives a numeric variable from an alphanumeric expression

remove() remove(str,tkn) removes the char tkn from str, eg remove(f1,'-')

replace() replace(str,tostr,fromstr)
replace(f1,"AAO","ÅÄÖ") replaces Å,Ä,Ö with respectively A,A,O

right() x = right(str,il) gives il chars from the end of str. If
str's length is less than il, then x is padded to the left with


```

blanks. At x = right(str,i1,'z') x is padded to the left with the
char z.
strip() x = strip(str,'B',' ') x = strip(str,'B') x = strip(str)
Third argument: char that shall be removed from beginning
and/or end av str, default blank.
Second argument, default 'B':
'B','b' (Both) remove from beginning och end
'L','l' (Leading) remove from beginning
'T','t' (Trailing) remove from end
substr() x = substr(str,i1,i2) gives i2 chars from and including n:o i1
subword() x = subword(str,i1,i2) gives i2 blank separated words from and
including n:o i1

```

Operators for Look-back and Look-ahead, numeric and alphanumeric

prev(variable) returns the value of a variable in the previous line
next(variable) returns the value of a variable in the next line

Can be used to identify first and last line of several with equal values
of ID fields. Values 0 if no previous respectively next line exists.

Functions for arguments, numeric and alphanumeric

inx() gives the level number 1, 2, ... in the listfile for the argument as
it is determined in the statement arg(). For example, in the introductory
example

```

arg(byggtyp) ... niv( ... ( 2,3-4 'Tvåplanshu' 1.32) ... ) ;
inx(byggtyp) gets the value 2 when byggtyp is 3, 4. Intended for norming of
specific arguments in multiclass analysis, in conjunction with arr(), in
Proc Data. See the section on norming in the multiclass analysis section.

```

With derived variables and the above functions Proc Taran can be used for
mathematical calculations. Make a file Dum1.Txt with only one line of
content "1 1 1" and a Rapp-program that looks something like this:

```

Include C:\Rapp\Rpp\Init.Rpp
Proc Taran listfil(a.txt) nummetod(u) ;
infiler fil(Dum1.Txt) var(Dur Skkost Dum)
dvar( x1 = log(sqrt(2*3.141592653589793)) )
utfilink(Ut.txt) utfilinkp( headerline drop(Recno Dur Skkost Dum) ) ;
arg(Dum);
ENDPROC

```

The file Ut.txt then gives the value of x1.

Use of Rapp as calculator. Write this Rapp-program:

Calc.Rpp

```

N
Include C:\Rapp\Rpp\Init.Rpp
system(Kalk.Txt)
Proc data ;
infiler fil(Dum1.Txt) var(Dum) // Dum1.Txt is one line "1 1 1".
dvar( x =
Include Kalk.Txt
);
utfil fil(Ut.Txt) var(x);
ENDPROC
system(Ut.Txt)

```

Make the file Kalk.Txt with a blank line.

With the execution of Calc.Rpp is obtained a screen, where Kalk.Txt is
filled out with a mathematical expression consisting of the operators and
functions above. After close of Kalk.Txt the result is obtained on the
screen.

endshow() Optional. In that parameter you give variables whose final value you want
to get on the screen. Intended primarily for variables defined with sumv()
or maxv(). Example: Endshow(Recno maxkund maxobj sumskkost) together with
Dvar(maxkund = maxv(kund) maxobj = maxv(obj) sumskkost = sumv(skkost))

`fil()` Mandatory with at least one, but any number can be entered which are then concatenated. Specifies an inputfile in text format.

`firstobs()` Optional. If specified, lines with consecutive number < firstobs are not used. Eg firstobs(2) if the first line contains headers, firstobs(3) if the first and second line contains headers.

`urval()` Optional. If given only those records that meet the selection conditions are used. Conditions are specified for variables given in `var()` and `dvar()`, in the form

```
variable condition-operator value [value [value ...]].
```

A negative value is written in one word, eg -3 and not - 3. Conditions for multiple variables can be combined with AND or & (logical AND) and OR or ! (logical OR). Parentheses can be used in arbitrarily many nestings. Negation of condition is written ! or NOT before a condition, a simple one or an expression in parentheses. A variable can be used several times in the same `urval()`. Max 250 selection conditions. After equality = or inequality ^= can be specified one or more values. There are several condition-operators for the same thing, so that users of eg SAS can write like they use to. EQ GE GT LE GQ LQ LT NE NQ shall be surrounded by blanks.

```
equality:      =    ==    EQ
inequality:    ^=    !=    <>  ><  NE    NQ
less than:     <    LT
greater than:  >    GT
less than or equal to:  <=    LE    LQ
greater than or equal to: >=    GE    GQ
```

Example: `urval(not(Bolag=4 10 OR (not Sktyp = 'TP' 'TE' ! ras>="KATT") and !skkost=0) and promr ne 'A1' 'A2')`

Alphanumeric selection values are given, separated by at least one blank, within single or double quotes. Swedish standard is used for å - ö in comparisons. Trailing blanks are not included in a comparison, for instance Rapp believes that "A" is equal to "A ". All condition-operators given above can be used.

`utfilink()` Optional. Specifies an output file of selected records in text format. Used to see if `urval()` is right and to create a new file with all or certain fields, including derived ones, dependent on the parameter `utfilinkp()`. Delimiter is given by `delimiterut()`, otherwise it the file gets the delimiter between fields that is used in the input file - blank if no delimiter was given or semi-colon if `delimiter(';')` was given. Floating point numbers are represented as compactly as possible given maximum accuracy, in order to optimize the file for further use in Proc Taran.

`utfilinkp()` Optional. Here you give the fields for `utfilink()` and if a first header shall be given. If not given, all fields including derived ones are put in `utfilink()`. With `keep()` are indicated fields that shall be included. With `drop()` are indicated fields that shall not be included. A first header is included if headerline is given. One can give `keep()`, `drop()` and headerline in any order. The fields come in the order they are in `keep()` if given, otherwise in the inputfile order.

Syntax:

```
utfilinkp([headerline] [keep(Field1 ...)] [drop(Field1 ...)]).
```

Examples

```
utfilinkp(headerline keep(Age brand dur prem))
utfilinkp(drop(date_of_birth frdk todk))
```

With `dvar()`, `utfilink()` and `utfilinkp()` one can come a long way in mangling data like SAS. With headerline and `delimiter(';')` is obtained a file that can be entered into Excel with Proc Excel. Without headerline is obtained a file for further use in Proc Taran. For example, one can calculate dur and age once and for all in a Proc Taran with `nummetod(u)` and some arbitrary argument.

`utfilexk()` Optional. Specifies an outputfile with non-selected lines in text format.

var()

Required. Record description of the file's fields. Each field shall be given with a word of up to 30 characters. Up to 600 fields. No distinction on lower and capital letters, ie Field1 is the same as fiEld1. A field name may not be N,I,R,D,F,A,C,\$, may not contain the characters () ; & < > = ' " and must not only contain numbers and comma point + - e E. Synonym to var() is post().

The fields can be divided into five categories:

1. Accounting fields for duration, etc
2. Arguments
3. Pure selection fields
4. ID fields for multiclass analysis - see that section
5. Fields not used by Proc Taran

The system field Recno is also available and can be used for any purposes - as argument, for selection, etc. It is of integer typ and is increased by 1 for each line in the inputfile. It has the value 1 for line no firstobs. For example, line 3 gets Recno = 1, if there are two headers in the inputfile and therefore one has set firstobs(3). The field is included in a possible utfilink, unless keep() is given for a collection of fields where Recno is not included or drop() is given for a collection of field where Recno is included.

After a field can be specified one or two of these parameters, optional lowercase or uppercase letters:

Datatype N,I, R,D,F, A,C,\$. Optional, N if not specified.

N,I indicates an integer in 4 bytes between -2147483648 and 2147483647.

R,D,F indicates a floating-point number in 8 bytes (Real, Double, Float).

A,C,\$ indicates an alphanumeric variable. It may not have embedded blanks if the fields are blank separated, but may otherwise have blanks, eg at tab separation.

Number of decimal places, eg, 3. Optional, 0 if not set. For example,

Field1 r 3 means that 123456 is loaded as the float number 123.456.

Field2 3 means that 123456 is read as the integer 123.

Decimal comma and decimal point accepted, but not thousands delimiter. For example, space between thousands must be removed with remove(field,' '). Integer values with datatype N in format Zoned Decimal made with COBOL or Easytrieve in the mainframe are accepted after transfer of data to Windows. For floating point data with type R, the format Zoned Decimal is accepted only for negative integer values. (Non-negative integer values in format Zoned Decimal are never produced by Easytrieve.) Selection fields (see below) can be numeric or alphanumeric, and they may be floats. Accounting fields and arguments can be used for selection.

Accounting fields: The following seven concepts are available for the report's account, where the optional ones are is given within []. Optional lowercase or uppercase.

Dur [Fbel] [Ndur] [Prem] [Antskad] Skkost [Kvadr]

There may be several lines with the same set of fields. Data need not be aggregated.

The insurance concepts

Dur = sum of insurance years for the fields of category 2-5 on the line.

Fbel (= sum insured under yearly risk) can not be given simultaneously with Ndur (= normed duration). They are treated the same way, ie as the exposure measure that applies in place of Dur. Here Fbel, Ndur, Prem shall have been calculated under yearly risk. For example Prem shall be earned premium, so that a certain policy is added to the Prem with its Dur*(annual premium) when indata is made to Proc Taran. If sum insured repectively annual premium are in the infiles, they shall be given as Fbel/Dur repectively Prem/Dur. In that case Rapp performs the multiplication with Dur. See below.

One can calculate Dur and other summed variables, see above under dvar(). Premium would then be given as Prem/Dur (annual premium) and (Sum insured) as Fbel/Dur. Alternatively in dvar(), for example:

```
dvar( Dur=durbl(frdk,todk)      Fbel=Fbelopp*Dur      Prem = annual_prem*Dur )
```

Here then the inputfile has one line per insurance version. A line can be eg. frdk = 20011201, todk = 20030301, giving duration 1.25. If begperiod and/or endperiod is not given, then duration is computed from the beginning and / or to the end of [frdk,todk]. Note however how Proc Durber computes duration.

The claim concepts

Antskad (= number of claims), Skkost (= claim cost for the Antskad claims) and Kvadr (= square of claim cost) shall be calculated separately for each claim, in which Kvadr = Skkost×Skkost, and can be aggregated up to the set of fields of category 2-5 that the line refers to. The line does not have to be aggregated: One can let each line refer to a single policyholder or a single claim, with the only difference that the run time will be slightly longer. The claim concepts can be recorded together with the insurance concepts in one line or in a separate file for claims specified in one of two statements Infiler. They can be calculated in dvar().

Antskad may be omitted, if the specified set of files given with a statement Infiler, where Skkost is, has one line per claim. This holds also for Kvadr, which in that case is computed as Skkost*Skkost for each line.

The accounting fields, except Dur, can alternatively be given as average values if the inputfile is structured so. These averages can be given:

```
Fbel/Dur
Ndur/Dur
Prem/Dur
Antskad/Dur
Skkost/Dur
Kvadr/Dur
Skkost/Antskad
Kvadr/Antskad
```

For example, if a file has been produced for SAS Proc Genmod it contains a field for claim frequency. That field is then given in Rapp as Antskad/Dur. A field for risk premium is given as Skkost/Dur. Mean claim is given as Skkost/Antskad. This means for example that SAS-tables for input to SAS Proc Genmod with Pearson's ϕ -estimate on non-aggregated claim data, with one line per claim, can be used directly in Rapp through a short program section. A file with one line per claim is then given as input in one of two statements Infiler.

Statement arg(), ended with semicolon

Between 1 and 99 such statements must be available. They describe arguments for factor smoothing with GLM or the classical method.

Parameters specified in the statement, they can be given in any order:

- antniv() Optional. Specifies the number of levels (classes) for numeric arguments. Should be used only when niv() is not given, which is variant 1 below. Execution will then be faster.
- arg() Required. A field given in the descriptions of all inputfiles. Arbitrary datatype N,R,A. It shall be understood to have the same meaning in both of two infile sets, if two are given.
- bas() Optional. Value 0 if not set. Specifies the base level in the argument that will be given factor 1 for risk premium, mean claim and claim frequency. If 0 is given through bas(0) or by default, the average value of the factors will be 1, weighted with fbel or ndur if any of those fields exists, otherwise weighted with dur. If the argument has only two levels (Yes or No), it may be advantageous to account for both modes by two runs of Proc Taran on the same input data and arguments. With one of the levels as base level, the hypothesis that the levels differ in risk is easier examined graphically.
- bas(/label) If label is specified last within bas(), such as bas(Yes/label), bas(25-29 yrs/label), bas(99/label), then what is before label is the

actual value of the input data that shall have the base level. For example, if there are seven values 1 5 8 12 14 88 99 and we want to have value 88, the 6th, as base level. With `bas(88/label)` this is achieved, and it is synonymous with `bas(6)`.

Alternative use of `bas()`: Like the parameter `bas=_` in `Proc Graf`. One can instead of a base class provide a special factor, such that all factor estimates for the risk premium are multiplied by the special factor, after the factor estimates are calculated so that the mean becomes 1. It is written with a decimal point to distinguish it from a base class. Example: `bas(2856.0)` means that all risk premium factor estimates are multiplied by 2856 while the constant in the list is divided by 2856. The frequency factors are not affected. (A practical problem with this may be that the risk premium factors in the list become too large so that the text protrudes too far to the right.) In the semi-colon separated textfile and in the graphs the mean claim factor estimates are also multiplied by the special factor, since they are ratios of risk premium and frequency factor estimates. Was that done we should not have a special factor in `Proc Graf`, for that would be cake on cake.

`niv()` Optional. Describes levels and possible tariff factors. See below.

`rub110` Optional. Long title <= 110 characters within single or double quotes. Becomes equal to "Argument argument name" if not set. Example `rub110('Bebyggelsetyp som de klassifierats av SCB 2001-01-01')`

`rub30` Optional. Short title not exceeding 30 characters within single or double quotes. Becomes equal to the argument name if not set. Example `rub30 'Bebyggelsetyp enl. SCB-klasser'`

`tar()` Optional. Provides tariff factors. Takes over possible tariff factors given within the `niv()`-parameter.

The parameters `antniv()` and `niv()` are related. There are four variants.

- Variant 1-2 requires for numerical argument values that they be between 1 and 65535.
- Variant 3 requires for numerical arguments that the diff between the largest and lowest values are not too large, depending on available RAM.
- Variant 4 requires either integer values between and -999999999 and 2147483647 or that the argument has datatype A (C,\$) and has values up to 10 characters.

1. Parameter `niv()` not given, `antniv()` given

See introductory example:

```
arg(Code) rub30 'Code' rub110 'Some code' bas(1) antniv(7);
```

If the argument is numeric, lines with the argument < 1 or > `antniv` are thrown away. If alphanumeric, lines with values of the argument over the `antniv:th` value are thrown away.

2. Parameter `niv()` given on the form below, `antniv()` given or not given

```
niv('nivnamn' tarfakt 'nivnamn' tarfakt ... )
```

See introductory example: `niv('Kod 01' 1.00 'Kod 02' 1.17)` Here `nivnamn` is a string of 10 characters giving level names and, optionally, `tarfakt` = tariff factor in an existing or recommended multiplicative tariff. With `N = antniv` if given, otherwise the number of class names given within single or double quotes, the following applies: If the argument is numeric, lines with the argument < 1 or > `N` are thrown away. If alphanumeric, lines with values of the argument over the `N:th` value are thrown away.

3. Parameter `niv()` given on the form below, `antniv()` given or not given.

```
niv( (n, argv argv ... argv:argv argv argv ... 'nivnamn' tarfakt) ... )
```

See introductory example:

```
niv(
  ( 4,1-9 'Övrigt' 1.00)
  ( 1,1:2 'Enplanshus' 1.00)
  ( 2,3-4 'Tvåplanshu' 1.32)
  ( 3,5 'Treplan' 1.40)
  ( 2,7 9 'Tvåpl spec' 1.32)
)
```

and this example for an alpha-numeric argument:

```
niv(
  ( 4,'A0'-'Z9'   'Övrigt'      1.00)
  ( 1,'E11': 'E12' 'Enplanshus' 1.00)
  ( 2,'E21'-'E29' 'Tvåplanshu' 1.32)
  ( 3,'E31'      'Treplan'     1.40)
  ( 2,'E27' 'E29' 'Tvåpl spec' 1.32)
)
```

Also here nivnamn is a string of 10 characters that provides level name and, optionally, tarfakt = tariff factor in an existing or recommended multiplicative tariff. If tarfakt is not given it will be 0. If given it must be preceded by a nivnamn within single or double quotes, to not be confused with a number as argument value.

Subsequently written levels have priority over the previous ones. Eg in the first example above only the argument values 6 and 8 remain in the Övrigt (Other) group with index value 4. If(4,1-9 'Övrigt' 1.00) had been written last, the whole argument had ended up in the Övrigt group.

The difference from Variant 2 is that

- Each level is indicated in parentheses.
- The level index 1,2, ... can be given after the first left parenthesis relating to the level. If not given the index counts up from 1 for first level and by 1 for each new level. In this case, a comma shall be given after the first left parenthesis relating to the level, eg (, 3-4 'Tvåplanshus').
- Level-name nivnamn, within single or double quotes, is optional for numeric arguments but mandatory for alpha-numeric arguments. It receives as its value the first given argument value in the last line that gives the index level, if not given.
- The argument is assumed given with integer values ... -2, -1, 0, 1, 2, ... after the first comma if numeric. Alphanumeric argument values shall be given within single or double quotes. If a sequence of characters gives two integers or strings that have a hyphen, without surrounding blanks, or colon between them, such as -69--60 or -69:-60 or -69 : -60 or 'CBS': 'NIS', then it means that these two values and all values in between shall give the level's index. Eg (7, 3 6 123:457 36) means that the values 3, 6, 36, 123, 124, ... 456, 457 shall provide index 7. The integers must have values between -2100000000 and 2100000000, and the difference D between maximum and minimum value should not be too large. How large D that can be accepted depends on available RAM. There is need for D bytes of memory if the number of levels is at most 254, and 2D if the number of levels is 255-65535. Sum of D or 2D or for all arguments by Variant 3 shall be accommodated within the RAM.

Lines in the input files with different values from those given in the niv()-parameter are thrown away.

Number of levels is antniv if this is given and is less than the highest given level within parentheses, otherwise the highest level given within parentheses. Up to 65535 levels.

4. None of the parameters niv() and antniv() given.

See introductory example:

```
arg(Bilmärke) rub30 'Bilmärke' rub110 'Bilmärke Volvo eller Jaguar' bas(2);
```

The argument shall be either an integer with values between -999999999 and 2147483647, or be alpha-numeric (datatype A,C,\$) and have values with a maximum of 10 characters. The presence of the arguments of this type causes Rapp to first scan the insurance file(s) to determine the value set, in order to subsequently be able to place the argument levels into indices 1.2 The order of levels in the report will be Swedish with å,ä,ö last in the alphabet, if the argument datatype is A. At most 65535 levels are allowed.

Statement TEXT, must be last in the proc

Optional. Shall begin with TEXT in capital letters alone on a line. What is written in the following lines before Endproc appear as text in the report after a table of

contents. The concluding Endproc is case insensitive but shall be alone on its line and shall not be preceded by a blank.

Proc Taran multiclass: OJ2010, SR 2018

Syntax in statement Proc Taran, where parameters in [] are optional and | indicates alternatives. Any order, also before or embedded among other parameters given in the manual for Proc Taran. All parameters are case independent.

```
Proc Taran
  (previously listed parameters in the manual, excluding listfil which is given below)
  multiarg(multiargname) | multiarg(multiargname1 multiargname2)
  [multigrupp(argname1 argname2 ...)]
  multimetod( // The indented parameters below shall appear inside multimetod().
    e|s [srkorr] [Bsaps] [Bsxzw] [Poisson]
    [hipsRO | hipsRO(1|2 B|C 1|2 B|C)] // hipsRO new parameter 2019-09-13.
    [fr-exp number] [fr-maxit number]
    [ms-exp number] [ms-maxit number]
    [rp-exp number] [rp-maxit number]
    [regrfexp number] [q number] [qM number]
    [Freqmcl] [Nonpseudo|NonpseudoM] [NonpseudoF]
    [Distfree | Gamma | Lognormal | Mix] // New parameters 2015-07-27 for SR:s method.
  )
  listfil(listfil-excl-multiarg)
  multilist1(listfil-incl-multiarg-univariate)
  multilist2(multiarg-accounting-with-texts)
  multilist3(multiarg-accounting-without-texts) multiheader // For column headings A,B,..
  multilist4(listfil-excl-multiarg-after-iterations-with-multiarg)
  [multilist5(semicolon-separated-textfile-content-as-multilist4)]
  [idgrupp(variablename1 variablename2 ...)] ;
```

There must be at least one argument in addition to the one or two credibility arguments. This can be a dummy, e.g. the variable dummyvar set by dvar(dummyvar = 1) for the insurance-, claim- or insurance-claim infiles.

Important parameters

multimetod()	First character: e for one of Esbjörn Ohlssons methods, s for Stig Rosenlunds method as described in Appendix 2. Also s is for the hierarchical method with a slight twist.
multiarg()	One or two credibility arguments with few or no claims in many classes. Two arguments apply to Esbjörn Ohlssons hierarchical method, of which the first one is the argument denoted Sector and the second one is the argument denoted Group.
multigrupp()	Optional. Argument(s) that are functions of the multi-argument(s), ie two lines of data in the insurance infile with the same value for the multi-argument(s) can not have different values for an argument in multigrupp(). Otherwise, Rapp gives an error message and stops. These arguments are called "auxiliary rating factors" in OJ2010.
freqmcl	Separate analysis of claim frequency and mean claim in the Stig Rosenlund method.
regrfexp	Exponent for flattening the factor ladder for the credibility argument in the Stig Rosenlund method. This is needed due to the regression effect - regression to the mean.
fr-exp, ms-exp, rp-exp	Exponents in Tweedie-like variance assumptions. See below. For Stig Rosenlunds method with parameter freqmcl for separate analysis of claim frequency and mean claim, these parameters are not used. The exponents will then always be 1 for claim frequency and 2 for mean claim, since in practice no other values are suitable.
fr-maxit, ms-maxit, rp-maxit	Maximal number of iterations between credibility and ordinary GLM. Set 0 for no iterations.
Poisson	If given for multimetod e claim numbers are assumed Poisson, meaning that $\sigma^2/\mu^p = \sigma^2/\mu$ will be set to 1. New parameter 2019-01-27.

See the **Applicability table** below for these and other parameters.

Below the expression `riskp,meancl` means mean claim, if parameter S-GLM for Standard GLM was given to Proc Taran. Otherwise it means risk premium.

The `listfiles'` meaning

`listfil` Tariff analysis results with the multi-argument(s) completely excluded

`multilist1` Univariate accounting, ie without estimated factors, for all arguments

`multilist2` multiclass analysis report with estimates, normed exposure, credibility weights and credibility factors, with texts

`multilist3` multiclass analysis report with credibility factors, without texts, for further processing with programs. For the methods below are given

1 OJ2010/4.2 A Index $j = 1, 2, \dots$ for the multiclass argument's levels
 B Name as in `multilist1` for the multiclass argument's levels
 C $\mu^{(-1)}Y^{-(.j.)}$ experience-value for claim-frequency by (4.23)
 D $\hat{U}_j$ for claim-frequency by (4.24)
 E $c_{0c}(j) = \hat{U}_j(\text{factorproduct for auxiliaries})$ for claim-frequency by Table 4.3 including a constant
 F $\mu^{(-1)}Y^{-(.j.)}$ experience-value for `riskp,meancl` by (4.23)
 G $\hat{U}_j$ for `riskp, meancl` by (4.24)
 H $c_{1c}(j) = \hat{U}_j(\text{factorproduct for auxiliaries})$ for `riskp,meancl` by Table 4.3 including a constant
 I $\text{fr-}\hat{U}_j \times \text{ms-}\hat{U}_j$ if `riskp,meancl` = mean claim under S-GLM
 J $\text{fr-}c_{0c}(j) \times \text{ms-}c_{1c}(j)$ if `riskp,meancl` = mean claim under S-GLM

2a OJ2010/4.4 A Index $j = 1, 2, \dots$ for sector level
 hierarchical. B Name as in `multilist1` for sector level
 2b with C Index $k = 1, 2, \dots$ for group level with start 1 in each sector
 multimetod s. D Index $1, 2, \dots$ for group level independent of sector
 E Name as in `multilist1` for group levels independent of sector
 F $\hat{U}_j$ for claim-frequency
 G $\hat{U}_{jk}$ for claim-frequency
 H $\hat{U}_j \hat{U}_{jk}$ for claim-frequency
 I $c_{0c}(jk) = \hat{U}_j \hat{U}_{jk}(\text{factorproduct for auxiliaries})$ for claim-frequency by section 4.4.2 including a constant
 J $\hat{U}_j$ for `riskp,meancl`
 K $\hat{U}_{jk}$ for `riskp,meancl`
 L $\hat{U}_j \hat{U}_{jk}$ for `riskp,meancl`
 M $c_{0c}(jk) = \hat{U}_j \hat{U}_{jk}(\text{factorproduct for auxiliaries})$ for `riskp/meancl` by section 4.4.2 including a constant
 N $\text{fr-}\hat{U}_j \hat{U}_{jk} \times \text{ms-}\hat{U}_j \hat{U}_{jk}$ if `riskp,meancl` = mean claim with S-GLM
 O $\text{fr-}c_{0c}(jk) \times \text{ms-}c_{0c}(jk)$ if `riskp,meancl` = mean claim with S-GLM

3 Rosenlund 2018 A Index $j = 1, 2, \dots$ for the multiclass argument's levels
 B Name as in `multilist1` for the multiclass argument's levels
 C $Y(j)$ = risk premium per normed exposure = experience-value by (C.1)
 D $\mu^{(k_j)} = \text{factor-smoothed risk premium} = \text{constant} \times (\text{factor-product for auxiliaries})$ by (C3)
 E $\Lambda^{(j)}$ = predictor (estimate) of credibility factor by (C6)
 F $\Lambda^{(j)}$ = predictor corrected for regression effect by (C7)

`multilist4` och `multilist5`: See the syntax description above.

1 If the first non-blank character following `multimetod(` is e or E, and also `multiarg(multiargname)` was given, then the technique used is the one in section 4.2 of OJ2010. Both claim frequency and `riskp,meancl` are treated, where `riskp,meancl` = mean claim if S-GLM was given. For the marginal totals and the Tweedie methods `riskp,meancl` = risk premium. The parameter p of equation (4.19) for frequency is specified after `fr-exp`, p for mean claim after `ms-exp`, and p for risk premium after `rp-exp`. If p is not given,

then default 1 is used for frequency, 2 for mean claim and parameter p in Tweedie(p) for that method. For marginal totals the default is $p = 1.5$ for risk premium. After `fr-maxit`, `ms-maxit` and `rp-maxit` are specified the maximum number of iterations between on one hand the determinations of $\hat{U}$ -factors and on the other hand factor estimates according to the chosen GLM method on all arguments, except the multiclass argument that is written in `multiarg()`. Default values are `fr-maxit = ms-maxit = rp-maxit = 0`.

The variables in `idgrupp()` are the ID variables used for the OJ2010 method. If S-GLM was given it is not used for mean claim, only for claim frequency. They must be numeric. If such a variable has more than nine digits, it must have type R ie float. A maximum of 15 digits can be stored in a floating point number. In OJ2010 there is an index t , which refers to any repetition within the cell. The classical one in credibility is calendar year, but here is meant insurances or insurance versions. In OJ2010 variance estimates are made by summaries of squared deviations over t in a certain way. If you write eg in home insurance `idgrupp(company forsnr)`, data are first summarized over that concept of identity, ie different versions with different helpnumbers and fromdates are pooled. With `idgrupp(company forsnr helpno fromdate)`, each version and time period gets its own t . If `idgrupp()` is omitted and there are separate insurance- and claim-infiles, there will be a maximum aggregation within the method of OJ2010, which means that it will be at most one t per combination of all arguments, including ordinary arguments such as policy-holder age, "auxiliary rating factors" and the multi-argument(s). If the insurances are aggregated on the multi arguments and there is only a dummy ordinary variable with value 1, the estimated σ^2 for claim frequency will be 0. With separate insurance- and claim-infiles, you can avoid problems by the parameter `Poisson`, to get $\sigma^2 = \mu$ for claim frequency, and by having one line per claim in the claim file.

If you have one combined insurance-claim-file, then its line number is taken as the id in `idgrupp()`. This is parallell to the treatment of the Tweedie model in ordinary GLM.

The notation in OJ2010 has a base-factor μ which depends on the base levels defined for the arguments. The OJ2010 method can however be applied without such a base-factor defined. The parameters that are estimated in Rapp will then be

$$\sigma^2/\mu^p \quad \tau^2/\mu^2 \quad \text{The square root estimates given (truncated to } \geq 0 \text{): } \sigma/\mu^{0.5p} \quad \tau/\mu$$

2018-09-11. The scaling to σ^2/μ^p instead of σ^2/μ^2 is better and was effected this date. For $p = 2$ (normal mean claim value) they are the same.

In the formula (4.27) for mean claim τ^2 Rapp uses the number of groups with claims as J in the factor $(J-1)$ for the σ^2 -estimate. This was changed 2014-01-04.

For the OJ2010 method are specified `Srkorrr`, `Bsaps`, `Bsxzw` for three corrections. `Srkorrr` is a recalculation of the credibility factors after an iteration. `Bsaps` is that $\mu^{-2}\tau^2$ is estimated with Bichsel-Straub's estimator $\hat{A}_{p\text{seu}}$ and not with med OJ2010's expression (4.27). `Bsxzw` means that the z-weighted mean value X_{zw} is used instead of $Y \sim (\dots)$ for the computation of $\hat{U}_j$. See F. E. De Vylders bok "Advanced Risk Theory" (1996), Editions de l'Université de Bruxelles, III.Ch.3 - Theorem 15 for X_{zw} and Theorem 18 för $\hat{A}_{p\text{seu}}$. Here we have $\hat{A} = \tau^2$. See Appendix 3.

2a, 2b If the first non-blank character following `multimetod()` is e, E, s, S, while at the same time `multiarg(multiargnamn1 multiargnamn2)` was given, then Rapp uses the hierarchical model in section 4.4 of OJ2010. Here `multiargnamn1` is OJ2010/4.4:s sector and `multiargnamn2` is OJ2010/4.4:s group. These arguments may have datatype and value set as other arguments. For example `multiarg(company LKF)`. Rapp recalculates company to sector 1,2,3, .. and LKF to group 1,2,3, ... starting with 1 in each sector, in line with OJ2010/4.4:s notation. Both sector and group are written to the listfiles `multilist2` and `multilist3` with both these positive integer indices and the original values, eg 011401, 011402, ... for LKF. Both frequency and riskp,meancl are treated. Parameters `idgrupp()`, `fr-exp`, `ms-exp`, `rp-exp`, `fr-maxit`, `ms-maxit`, `rp-maxit`, `srkorrr` have the same meaning and default values as for the OJ2010/4.2 method. However, `bsaps` and `bsxzw` do not (yet) apply to the hierarchical model. OJ2010/4.4 does not describe iterations between determinations of the $\hat{U}$ -factors and the factor estimates for all arguments excluding sector and group, but such iterations are implemented in Rapp. The norming factors that correspond to $\hat{U}_j$ in Step 1 of section 3.1 is for the hierarchical case $\hat{U}_j\hat{U}_{jk}$, which I perceive to be the

correct interpretation of the model. A warning however that one may get unstable factor estimates at iterations in the hierarchical case.

2b, which takes effect with `multimetod s`, is the same as **2a**, with the exceptions that parameter Poisson is always implied, `riskp`, `meancl` means mean claim, zero cost claims are counted, and the iterations use MMT. The parameter S-GLM has no effect. With **2a** you have to give Poisson explicitly. **2b** is essentially by Esbjörn Ohlsson, but these twists motivate to denote this method, with two multi arguments, `multimetod s`. If `hipsRO` or `hipsRO(p method p method)` is given, then `fr-exp` is set to 1, `ms-exp` is set to 2 and the pseudo-estimators of my forthcoming article "Hierarchical credibility pseudo-estimators" will be used. σ^2 and μ will also be affected. Give the `hipsRO` parameter within `multimetod()` as the following examples show.

```
hipsRO use method A defined in "Hierarchical credibility pseudo-estimators" for both p
hipsRO(1b) use method B for p = 1 and A for p = 2
hipsRO(2c) use method A for p = 1 and C for p = 2
hipsRO(1c2b) use method C for p = 1 and B for p = 2
hipsRO(1 c 2 b) use method C for p = 1 and B for p = 2
```

OJ2010/4.4 contains like OJ2010/4.2 a base factor μ . For application of the OJ2010/4.4 method without the need to define such a base-factor, Rapp estimates the parameters

σ^2/μ^p v^2/μ^2 τ^2/μ^2 Square root estimates given (though ≥ 0): $\sigma/\mu^{0.5p}$ v/μ τ/μ

2018-09-11. The scaling to σ^2/μ^p instead of σ^2/μ^2 is better and was effected this date.

3 If the first non-blank character following `multimetod()` is `s` or `S`, then Rapp uses the method of Credibility pseudo-estimators, Appendix 2 (Scandinavian Actuarial Journal 2018). The number after `rp-maxit` is used in the same way as for OJ2010:s method. The number after `q` is the factor `q` present in an earlier version of Stig's paper, default 0. The number after `qM` is the factor `QM` in the earlier paper, default 0. The number of `rp-exp` is the exponent `p` in assumption A4 in the earlier paper, default 1.5. If `fregmcl` (see below) is given, exponents `pF = 1` and `pM = 2` are used. I now recommend `fregmcl`, so `q`, `qM` and `p` are obsolete. The number after `regrfexp` (default 0.8) is δ in (C7) and is used for a corrected column "Regression fallcorr rp" which is expression (C7). With `0 < regrfexp < 1` then is obtained a flatter factor sequence, which is intended to be a better factor estimation with regard to the "Regression fallacy". By that I mean that the factor values to a part are influenced by multiclass levels, which in reality are rather normal, by chance happening to get lower or higher values than they deserve. Selecting `regrfexp` is (so far for me) entirely a matter of intuition. In my practice, I have clearly seen that there is a regression effect, in the sense that a subsequent analysis of multiclass classifications on new data independent of those used for classification gave flatter ladders for factor estimates than the original, if these were not corrected in this way.

τ_F^2 and τ_M^2 are the claim frequency and mean claim versions, respectively, of the generic parameter τ^2 in Assumption 1 of Appendix 2. The generic estimator of the parameter is given in Theorem 5.2.

In Appendix 2, Section 1.3, it is stated that I haven't been able to show that the equations for the pseudo-estimators of τ^2 for claim frequency and mean claim, respectively, have at most one positive solution. While my intuition tells me that this is so, I have nevertheless defined the psedo-estimate as the largest of possibly several solutions. Despite this, Rapp gives a solution that possibly might be lower than the largest one. I have not found it worthwhile to seek a possibly larger solution.

Method 3 is now implemented for the MMT and the S-GLM methods of estimating μ_{k_j} .

New functionality 2014-02-26 and 2014-04-24: Parameter `Nonpseudo` or `NonpseudoM` means that the τ^2 -estimator for mean claim will be τ^2/μ^2 by OJ2010, as described under **1**. I have found that it is sometimes preferable to use my pseudo-estimator and sometimes preferable to use the OJ2010 non-pseudo-estimator. This is described in Appendix 2. Parameter `NonpseudoF` means that the claim frequency τ^2 -estimator will be τ^2/μ^2 by OJ2010, but modified by taking $\sigma^2 = \mu$, thus using the Poisson

assumption. This is sometimes preferable over the pseudo-estimator, but less often than in the mean claim case. -- In Rappmenus, give CrRonpF, CrRonpM or CrRonpFM for Method. Freqmcl is implied with any of these three Methods.

New functionality 2015-07-27: Parameters Distfree or Gamma or Lognormal or Mix added. They have effect with Freqmcl. See below.

A summary of how the parameters are applicable (marked with Y) in the three methods:

Applicability table

Applicable in	Method				
Parameter	Default	1	2	3	Remark
bsaps	not used	Y	-	-	
bsxzw	not used	Y	-	-	
distfree	not used	-	-	Y	Distribution-free estimator of τ_M^2 , See Appendix 2.
fr-exp	1	Y	Y	-	
fr-maxit	0	Y	Y	Y	
freqmcl	not used	-	-	Y	Separate claim freq and mean cl analyses, exponents 1 and 2.
gamma	not used	-	-	Y	Gamma-assuming estimator of τ_M^2 , See Appendix 2.
lognormal	not used	-	-	Y	Lognormal-assuming estimator of τ_M^2 .
mix	not used	-	-	Y	Mixed Gamma-lognormal-assuming estimator of τ_M^2 .
ms-exp	2	Y	Y	-	
ms-maxit	0	Y	Y	Y	
nonpseudoF		Y	-	-	See above. Non-pseudo-estimator used for claim frequency.
nonpseudoM		Y	-	-	See above. Non-pseudo-estimator used for mean claim.
q	0	-	-	Y	Q
qM	0	-	-	Y	QM
regrfexp	0.8	-	-	Y	δ
rp-exp	1.5	Y	Y	Y	
rp-maxit	0	Y	Y	Y	
srkorr	not used	Y	Y	-	

Example

```
Proc Taran listfil(L1.Txt) nummetod(n) rub62 'Villaförsäkring version 15'
  multiarg(Bolag LKFKod) multimetod( Esbjörn fr-exp 1 rp-exp 1.5 fr-maxit 5 rp-maxit 4)
  multigrupp(Tätortsgrad Medianinkomst boendeform Befintligt-Område) idgrupp(bolag
  forsnr) multilist1(M1.Txt) multilist2(M2.Txt) multilist3(M3.Txt) multilist4(L2.Txt);

Proc Taran listfil(L1.Txt) textfil(Texttest1.Txt) nummetod(n)
  rub62 'PPF Lastbil Vagnskada version 15' Multiarg(LKFKod)
  Multimetod(Stig rp-maxit 4 regrfexp 0.6) multigrupp(Tätortsgrad Medianinkomst
  boendeform Befintligt-Område) multilist1(M1.Txt) multilist2(M2.Txt) multilist3(M3.Txt)
  multilist4(L2.Txt) multilist5(T5.Txt) ;
```

Normed exposure exposure = sum of exposure*(factor product for the ordinary variables) is written in multilist2 for all methods. The factor product is exclusive base premium factor. All factor ladders for an argument has average 1 weighted with exposure. The base premium factor is instead in the "Factorest riskpremium = multiplicative risk premium from Auxiliaries, per unit normed exposure", that is written in the reports by Stig's method Appendix 2.

Example program

```
Include C:\Rapp\Rpp\Init.Rpp
Proc Taran listfil(t0.txt) S-GLM multiarg(kundnr) multimetod(e /*srkorr*/ bsaps bsxzw
fr-maxit 5 ms-maxit 5) multilist1(t1.txt)
multilist2(t2.txt) multilist3(t3.txt) multilist4(t4.txt) multilist5(t5.txt) ;
infil fil(C:\s\busscase2.txt) var(zon bussald kundnr antavt dur antskad
skkost dum1 dum2) urval(kundnr ne 145);
arg(bussald) bas(5); arg(zon) bas(4); arg(kundnr);
ENDPROC
```

Rapp-program that makes a PDF file from the Rapp-program above, shown in colour below

```
Include C:\Rapp\Rpp\Init.Rpp
Proc graf pdfil(a.pdf) listfil(Mtst1.Rpp) visa endproc
```

Include C:\Rapp\Rpp\Init.Rpp

```
Proc Taran listfil(t0.txt) S-GLM multiarg(kundnr) multimetod(e /*srkorr*/ bsaps
bsxzw
fr-maxit 5 ms-maxit 5) multilist1(t1.txt)
multilist2(t2.txt) multilist3(t3.txt) multilist4(t4.txt) multilist5(t5.txt) ;
infiler fil(C:\s\busscase2.txt) var(zon bussald kundnr antavt dur antskad
skkost dum1 dum2) urval(kundnr ne 145);
arg(bussald) bas(5); arg(zon) bas(4); arg(kundnr);
ENDPROC
```

Note that multiclass parameters are brown, so that you can verify that you spelled it right - a misspelled parameter becomes black.

The first three lines of the input file busscase2.txt

1	0	15	2	1.585216	0	0	0	0
1	0	145	47	12.062971	377	294556	377	377
1	0	184	6	3.321013	4	22152	4	4

Norming

When dealing with arguments, ordinary or in multigrupp() (ie "Auxiliary rating factors"), with many levels and uncertain outcomes, you may want to replace point estimates obtained in multilist4 with assessments. For example, if five years of age sticks up by chance (large claim), you can take a risk premium factor value between them for ages 4 and 6 years but let other risk premium factors be left unchanged. Then we norm for age according to the example of Proc Data. In multiclass analysis with Proc Taran then Ndur is used as exposure, and the argument age is omitted as arg()-clause.

Proc Xlmerg

Example: Proc Xlmerg Infiler("C:\Ut\A1.xml" "a2.xml") Utfil(a2.xml) Names(- Sh02) Endproc Copies all sheets to Utfil(). Eg if C:\Ut\A1.xml has 2 sheets and a2.xml has 3 sheets, then a2.xml will get 5 sheets. If Names() is omitted the original sheet names are kept, unless Rapp needs to change duplicate names. A hyphen, like the first entry of Names() in the example, also means that the original name is kept. Works for all Xml files that were created by Rapp and not changed manually after that.

Swedish to English glossary for reserved words

anmdat	report-date when a customer reports a claim
antmb	number of megabytes; antal = number
antniv	number of levels (classes)
Antskad	number of claims
arsranta	yearly interest rate
bet	relating to payment; betalning. In parameters in Proc Reschl.
Dur	duration; exposure unless Fbel is given
durber	duration computation; ber = short för beräkning = computation
Fbel	sum insured under yearly risk
fil	file
Kvadr	square
livr	relates to life annuity; livränta = life annuity
mellagg	aggregation in between; mellan = between
Ndur	normed duration
niv	level; nivå = level (class)
Prem	earned premium
riktnh	direction to the right
rskilj	risk differentiating; skilja = differentiate
rub	header; rubrik = header
s	claim cost in percent of premium; skadeprocent. Parameter in Proc Graf.
sats	statement
satser	statements
skadat	date when the claim occurred
skadedat	date when the claim occurred
skk	relating to incurred claim cost; skadekostnad. In parameters in Proc Reschl.
Skkost	claim cost
sludat	date when the claim was settled
slutdat	date when the claim was settled
svans	tail. In parameters in Proc Reschl.
tal	number

```

tid      time
tidssort time unit
tutb     total paid up to now
urval    selection
ut       out
utfil    outfile
visa     show

```

Note. Rapp expects numbers without blanks or commas. Give decimal points if appropriate. Scientific notation such as 1.23e-45, 98E+7 and 65.4e3 is accepted.

Examples of Rapp programs (not multiclass analysis) for Windows

Example 1:

Edited in an editor (SPF) with syntax dependent colors that recognize the extent .Rpp. (Here is written Proc Jung, who is an older term for Proc Taran.).

```

SPF A15.jng - e:\Riskanalys\Jung\
COMMAND ==>
*** Top of File ***
000001 %Include Init.jng
000002
000003 Proc Sasut libname(J:\SAK\Data\B99STI) tabell(Vi01fs2)
000004     textfil(C:\s\al5in.txt) var(
000005         a01 a02 a03 a04 a05 a06 a07 a08 a09 a10 a11 a12 a13 a14 a15
000006         dur skfr riskp
000007     )
000008 ENDPROC
000009
000010 /* Kvadratsumma finns ej, så då beräknas kvadr som skkost*skkost.
000011     Om högst en skada per rad i infilen ger det rätt resultat. */
000012 Proc Jung listfil(a15.txt) textfil(a15s.txt) nummetod(N)
000013     rub62 'Test körning 6 antarg = 15, antfria = 70';
000014 infiler fil(C:\s\al5in.txt) post(
000015     a01 a02 a03 a04 a05 a06 a07 a08 a09 a10 a11 a12 a13 a14 a15
000016     dur antskad/dur skkost/dur
000017 ) ;
000018 arg(a01) bas( 2); arg(a02) bas( 5); arg(a03) bas(24); arg(a04) bas( 2);
000019 arg(a05) bas(10); arg(a06) bas( 7); arg(a07) bas( 9); arg(a08) bas( 2);
000020 arg(a09) bas( 2); arg(a10) bas( 4); arg(a11) bas( 3); arg(a12) bas( 2);
000021 arg(a13) bas( 3); arg(a14) bas( 4); arg(a15) bas( 5);
000022 ENDPROC
000023
000024 PROC Sasin libname(F:\B99STI\SAS\Data) tabell(Vi01fs2skattn)
000025     textfil(a15s.txt) delimiter(',') firstobs(2)
000026     satser(drop Argnamn Nivnamn Fbelndur Prem Tarf;)
000027     var(Argnamn $char3. @32 Nivnamn $char10. Anr Ninr Dur Fbelndur Prem
000028         Antskad Skkost Ospmu Osp Basff Basfm Basfr Faktf Faktm Faktr
000029         Ospf Ospm Ospr Tarf)
000030 ENDPROC
000031
000032 system(del C:\s\al5in.txt)
*** Bottom of File ***

```

Example 2:

```
SPF PRISAS-tst1.jng - c:\CKOD\
COMMAND ==>
000001 %Include Init.jng
000002 Proc Sasut libname(G:\Sak\Forsakringsekonomi\B99sti\sas\data) tabell(Skadetabell)
000003 textfil(z1.txt) var(skadkost standard01 standard02 standard03) ENDPROC
000004 Proc Sasut libname(G:\Sak\Forsakringsekonomi\B99sti\sas\data) tabell(Forsskad)
000005 textfil(z2.txt) var(dur standard01 standard02 standard03) ENDPROC
000006
000007 Proc Jung listfil(Ptst1.txt) textfil(Ptst1s.txt) nummetod(N)
000008 rub62 'PRISAS-exempel':
000009 /* Argumenten har bara siffror och kan vara numeriska här. */
000010 infiler fil(z1.txt) var(skkost standard01 standard02 standard03) ;
000011 infiler fil(z2.txt) var(dur standard01 standard02 standard03) ;
000012 arg(standard01) bas(1) niv( (1,99 'Man') (2,2 'Kvinna') (3,3 'Jur-person') )
000013 rub30 'Kön / juridisk person'
000014 rub110 'Från personnummer/orgnummer, månadssiffrorna och näst sista siffran'
000015 tar(1 0.85 1.1) ;
000016 arg(standard02) bas(2) niv( (1,1 'Kkl 1-2') (2,99 'Kkl 3') (3,3 'Kkl 4-5') )
000017 rub30 'Körsträckeklass' rub110 ''
000018 tar(0.90 1 1.25) ;
000019 arg(standard03) bas(4) niv(
000020 (1, 1 'Breobj 00')
000021 (2, 2 'Breobj 01')
000022 (3, 3 'Breobj 02')
000023 (4,99 'Breobj 03')
000024 ) rub30 'Boende' rub110 'Breobj 00 = ej boendeförsäkrad i Länsförsäkringar'
000025 tar(1.2 1.1 1.2 1.0) ;
000026 TEXT
000027 "Faktorer frekvens" och "Faktorer riskprem" har fått ur ekvationssystem, som
000028 rensar bort inflytandet från andra argument än det tabulerade. Kolumnen
000029 "Faktorer riskprem" är lämplig att använda för premiesättning, dock med hänsyn
000030 till kolumnen Rfaktospot (osäkerhet i %). Få och/eller ojämnt storleksfördelade
000031 skador ger en hög osäkerhetsprocent, vilket gör riskpremiefaktorerna mindre
000032 användbara för premiesättning.
000033 ENDPROC
000034 system(del z1.txt & del z2.txt)
000035 Proc Graf listfil(Ptst1.txt) pdfil(Ptst1.pdf) /* Normalfördelning utom för T */
000036 T R2_bas=1_2_4 f2_bas=1_2_4 m2_bas=1_2_4 u2_bas=1_2_4 a visa ENDPROC
*** Bottom of File ***
```


The program in PDF from Rapp statement: PROC Graf listfil(PRISAS-tst1.Rpp) pdfil(a.pdf) ENDPROC If you do not have an editor with the ability to color adjust file types to a language syntax, then PROC Graf offers instead the possibility to check the syntax of a Rapp-program under development in this way.

Adobe Reader - [a.pdf]

Bokmärken

Signaturer

Lager

Sidor

PRISAS-tst1.jng

1

```

%Include Init.jng
Proc Sasut libname(G:\Sak\Forsakringsekonomi\B99sti\sas\data) tabell(Skadetabell)
textfil(z1.txt) var(skadkost standard01 standard02 standard03) ENDPROC
Proc Sasut libname(G:\Sak\Forsakringsekonomi\B99sti\sas\data) tabell(Forsskad)
textfil(z2.txt) var(dur standard01 standard02 standard03) ENDPROC

Proc Jung listfil(Ptst1.txt) textfil(Ptst1s.txt) nummetod(N)
rub62 'PRISAS-exempel';
/* Argumenten har bara siffror och kan vara numeriska här. */
infiler fil(z1.txt) var(skadkost standard01 standard02 standard03) ;
infiler fil(z2.txt) var(dur standard01 standard02 standard03) ;
arg(standard01) bas(1) niv( (1,99 'Man') (2,2 'Kvinna') (3,3 'Jur-person') )
rub30 'Kön / juridisk person'
rub110 'Från personnummer/orgnummer, månadsiffrorna och näst sista siffran'
tar(1 0.85 1.1) ;
arg(standard02) bas(2) niv( (1,1 'Kkl 1-2') (2,99 'Kkl 3') (3,3 'Kkl 4-5') )
rub30 'Körsträckeklass' rub110 ' '
tar(0.90 1 1.25) ;
arg(standard03) bas(4) niv(
(1, 1 'Bredbj 00')
(2, 2 'Bredbj 01')
(3, 3 'Bredbj 02')
(4,99 'Bredbj 03')
) rub30 'Boende' rub110 'Bredbj 00 = ej boendeförsäkrad i Länsförsäkringar'
tar(1.2 1.1 1.2 1.0) ;
TEXT
"Faktorer frekvens" och "Faktorer riskprem" har fått ur ekvationssystem, som
rensar bort inflytandet från andra argument än det tabulerade. Kolumnen
"Faktorer riskprem" är lämplig att använda för premiesättning, dock med hänsyn
till kolumnen Rfaktospct (osäkerhet i %). Få och/eller ojämnt storleksfördelade
skador ger en hög osäkerhetsprocent, vilket gör riskpremiefaktorena mindre
användbara för premiesättning.
ENDPROC
system(del z1.txt & del z2.txt)
Proc Graf listfil(Ptst1.txt) pdfil(Ptst1.pdf) /* Normalfördelning utcm för T */
T R2_bas=1_2_4 f2_bas=1_2_4 m2_bas=1_2_4 u2_bas=1_2_4 a visa ENDPROC

```

215,9 x 279,4 mm
1 av 1

Start
C:\CKOD
SPF 3270 c:\CK...
Tolk komm...
Adobe Rea...
Jung

Example 3:

```

/* Rapp-program to check a few things, including how SAS imports and
   exports an estimate file. */
Include C:\Rapp\Rpp\Init.Rpp /* Innehåller en Proc Init */
/* Namn på nivåer skall alltid ha högst 10 tecken.
   Namn på argument skall alltid ha högst 30 tecken. */
Proc Cmd
@echo off
G:\b99sti\hf.exe Rappskadl.txt b99sti.Jskadl.dat w
G:\b99sti\hf.exe Rappforsl.txt b99sti.Jforsl.dat w
Endproc
Proc Taran listfil(rappl.txt) textfil(skattningar.txt) nummetod(N)
  rub62 'Test exempel n:o 01';
infil fil(Rappforsl.txt) delimiter() var(Kod Bilmärke $ Geografi $ dur prem) ;
infil fil(Rappskadl.txt) delimiter() var(Bilmärke $ Geografi $ Kod Skart A
  antskad skkost/antskad kvadr/antskad) urval(!skkost=0 & skart >='110');
arg(Kod) rub30 'Kod' rub110 'Någon kod' bas(1) niv('Kod 000001' 1.00
  'Kod 02' 1.17); /* 1.00 och 1.17 = tariffen */
arg(Bilmärke) rub30 'Bilmärke text' rub110 'Bilmärke Volvo eller Jaguar' bas(1);
arg(Geografi) rub30 'Tät- eller glesbygd'
  rub110 'Enligt SCB:s klassifikation från 1998' bas(0)
  tar(1 1.23 0.86 0.56); /* Alternativt sätt att ge tariffaktorer */
TEXT
"Faktorer frekvens" och "Faktorer riskprem" har fått ur ekvationssystem, som rensar
bort inflytandet från andra argument än det tabulerade. "Faktorer riskprem" är lämplig
att använda för premiesättning, dock med hänsyn till kolumnen Rfaktospct. Få och/eller
ojämnt storleksfördelade skador ger hög osäkerhetsprocent, så att riskpremiefaktorerna
blir mindre användbara för premiesättning.
ENDPROC
PROC Rskilj listfil(rappl.txt) Endproc
PROC Graf listfil(rappl.txt) pdfil(a.pdf)
  T_bas=1_1_0 R_bas=1_1_0 F_bas=1_1_0 M_bas=1_1_0 U S A visa ENDPROC
PROC Graf listfil(rappl.txt) pdfil(b.pdf) T_bas=1_1_0 R_bas=1_1_0 F_bas=1_1_0
M_bas=1_1_0 U S A ENDPROC
PROC Sas
  libname Bibl 'F:\B99STI\SAS\Data';
  filename in 'skattningar.txt';
DATA Bibl.Rappskattn;
  INFILE in DLM=';' FIRSTOBS=2;
  INPUT Argnamn $ Nivnamn $char10. Anr Ninr Dur Fbelndur Prem Antskad Skkost
    Ospmu Osp Basff Basfm Basfr Faktf Faktm Faktr Ospf Ospm Ospr Tarf
    ; RUN; quit;
ENDPROC
PROC Sasut libname(F:\B99STI\SAS\Data) tabell(Rappskattn)
  textfil(skattningar2.txt) delimiter(';')
  var(Argnamn Nivnamn Anr Ninr Dur Fbelndur Prem Antskad Skkost Ospmu Osp
    Basff Basfm Basfr Faktf faktm Faktr Ospf Ospm Ospr Tarf)
ENDPROC
PROC Sasin libname(F:\B99STI\SAS\Data) tabell(Rappskattn2)
  textfil(skattningar.txt) delimiter(';') firstobs(2)
  var(Argnamn $ @32 Nivnamn $char10. Anr Ninr Dur Fbelndur Prem Antskad Skkost
    Ospmu Osp Basff Basfm Basfr Faktf Faktm Faktr Ospf Ospm Ospr Tarf)
ENDPROC
PROC Sasut libname(F:\B99STI\SAS\Data) tabell(Rappskattn2)
  textfil(skattningar3.txt) delimiter(';')
  var(Argnamn Nivnamn Anr Ninr Dur Fbelndur Prem Antskad Skkost Osp
    Basff Basfm Basfr Faktf Faktm Faktr Ospf Ospm Ospr Tarf)
ENDPROC
/* system(del rapp1.txt) behöver sparas */
system(del a.pdf) /* behöver inte sparas */

```

Example 4:

Data mangling in the two Rapp-programs, based on text files directly from statistical databases. The second statement Proc Init ... Endproc is there to enable execution with double click on Mangla.Rpp in Windows Explorer.

Mangla. Rpp

```

Include E:\Riskanalys\Rapp\Init.Rpp
Proc Init pathdir(C:\Tariffanalys\Rapp) Endproc /* Där Rapp.Exe finns. */
system(Rapp Mangla1 x cha $01 HAST $02 20050401)
system(Rapp Mangla1 x cha $01 HUND $02 20050501)
system(Rapp Mangla1 x cha $01 KATT $02 20050501)

/* Gör SAS-tabeller av de större försäkringsfilerna och skadefilerna. */
Proc Sasin libname(C:\sas) ctyp(6 14 15 16 17 18 21 33 35 47 48) tabell(Agrskad_HAST)
  textfil(Agrskad_HAST.txt) Endproc
Proc Sasin libname(C:\sas) ctyp(6 14 15 16 17 18 21 33 35 47 48) tabell(Agrskad_HUND)
  textfil(Agrskad_HUND.txt) Endproc
Proc Sasin libname(C:\sas) ctyp(6 14 15 16 17 18 21 33 35 47 48) tabell(Agrskad_KATT)
  textfil(Agrskad_KATT.txt) Endproc

system(del tmpfl.txt & del tmpf2.txt & del tmps1.txt & del tmps2.txt & del tmps3.txt)

```

Mangla1. Rpp

```

/* Exekveras från Mangla.Rpp */
Include E:\Riskanalys\Rapp\Init.Rpp

/* 1. Försäkringsversioner. Extrahera vissa fält och gör om datum, t ex 2005-05-01 till
20050501. Sortera på Kund Avt Obj Momrad Fromdat Åtgdat. */
Proc Data ;
Infiler fil(UUIG_STATBESTÅND_$01.TXT) dlm(9)
var(
  Kund Avt Obj Åtgdatin $ Åtgkod $ Momrad Avttyp $ Objtyp $ Fp $ Mtyp $ Mvillk $
  Ffdatin $ Fromdatin $ Slutdatin $ Nytedatin $ Tdatin $ Föddatin $ Postnr $ Termin $
  Bolag $ Ombud $ Provisionsbolag $ Provisionsombud $ Kön $ Anvkod $ Ras $ Annuorsak $
  Tlän $ Tariffkommun $ Premiekl Antdekl $ Regnr $ Sjrp Rabtyp1 $ Rabtyp2 $ Rabtyp3 $
  Rabtyp4 $ Rab_kr Fbel Premiekod $ Årspremie Antal_djur_areal R Vikt_år_0 R
  Vikt_år_m1 R Vikt_år_m2 R Vikt_år_m3 R Vikt_år_m4 R Vikt_år_m5 R Vikt_rull_tolv R
)
dvar(
  Fromdat = remove(Fromdatin, '-')
  Åtgdat = remove(Åtgdatin, '-')
  Ffdat = remove(Ffdatin, '-')
  Nytedat = remove(Nytedatin, '-')
  Tdat = remove(Tdatin, '-')
  Föddat = remove(Föddatin, '-')
)
;
Utfil fil(tmpfl.txt) dlm(9)
var(
  Kund Avt Obj Momrad Fromdat Åtgdat
  Åtgkod $ Avttyp $ Objtyp Fp $ Mtyp Mvillk
  Ffdat Nytedat Tdat Föddat Postnr
  Termin Bolag Kön Anvkod Ras $
  Annuorsak $ Sjrp Fbel Årspremie
)
sort(Kund Avt Obj Momrad Fromdat Åtgdat)
;
Endproc

/* 2. Försäkringsversioner. Unik ID för en post i Statbestånd är
(Kund,Avt,Obj,Momrad,Fromdat,Åtgdat). Av poster med samma
(Kund,Avt,Obj,Momrad,Fromdat) skall man endast använda den
med högst Åtgdat och kasta bort övriga. */
Proc Data ;
Infiler fil(tmpfl.txt) dlm(9)
var(
  Kund Avt Obj Momrad Fromdat Åtgdat
  Åtgkod $ Avttyp $ Objtyp Fp $ Mtyp Mvillk
  Ffdat Nytedat Tdat Föddat Postnr
  Termin Bolag Kön Anvkod Ras $
  Annuorsak $ Sjrp Fbel Årspremie
)
dvar(

```

```

/* Sammaidindikator = 1 om nästa rad har samma värden på Kund ... Fromdat, 0 annars. */
Sammaidindikator = Ind0(Kund-next(Kund)) * Ind0(Avt-next(Avt)) * Ind0(Obj-next(Obj)) *
Ind0(Momrad-next(Momrad)) * Ind0(Fromdat-next(Fromdat))
)
urval(Sammaidindikator = 0)
;
Utfil fil(tmpf2.txt) dlm(9)
var(
Kund Avt Obj Momrad Fromdat
Åtgkod $ Avttyp $ Objtyp Fp $ Mtyp Mvillk
Ffdat Nytedat Tdat Föddat Postnr
Termin Bolag Kön Anvkod Ras $
Annuorsak $ Sjrp Fbel Årspremie
)
;
Endproc

/* 3. Försäkringsversioner. Härledning av det datum som en försäkringsversion gällt
intill i Statbestånd. Det kallas nedan för Todk (TomDatumKorrigerat).
Unik ID i tmpf2.txt är (Kund,Avt,Obj,Momrad,Fromdat).
Todk sätts enligt följande:
A) Om det finns en annan post med samma (Kund,Avt,Obj,Momrad)
och högre Fromdat, sätt Todk = den andra postens Fromdat.
B) Om inte A) gäller, kolla om åtgärds-koden Åtgkod = 040. Sätt
i så fall Todk = Fromdat, i annat fall Todk = Ffdat.
Åtgärds-koder Åtgkod
020 ändring
040 annulation
070 förnyelse
090 nyteckning */
Proc Data;
Infiler fil(tmpf2.txt) dlm(9)
var(
Kund Avt Obj Momrad Fromdat
Åtgkod $ Avttyp $ Objtyp Fp $ Mtyp Mvillk
Ffdat Nytedat Tdat Föddat Postnr
Termin Bolag Kön Anvkod Ras $
Annuorsak $ Sjrp Fbel Årspremie
)
dvar(
/* Sammaidindikator = 1 om nästa post har samma värden på Kund t o m Momrad, 0 annars.
Annulationsindikator = 1 om posten är annullerad, 0 annars. */
Sammaidindikator = Ind0(Kund-next(Kund)) *
Ind0(Avt-next(Avt)) *
Ind0(Obj-next(Obj)) *
Ind0(Momrad-next(Momrad))
Annulationsindikator = Indc(Åtgkod,"040")
Todk = Sammaidindikator*next(Fromdat) + (1-Sammaidindikator)*
(Annulationsindikator*Fromdat+(1-Annulationsindikator)*Ffdat)
Ålder = min(99,[(Fromdat-Föddat)/10000])
)
;
Utfil fil(Agrfors_$01.txt) dlm(9) headerline
noempty /* med noempty läggs en blank ut i stället för ett tomt fält, för Proc Sasin */
var(
Kund Avt Obj Momrad Fromdat Todk
Avttyp $ Objtyp Fp $ Mtyp Mvillk
Ffdat Nytedat Tdat Föddat Postnr
Termin Bolag Kön Anvkod Ras $
Annuorsak $ Sjrp Fbel Årspremie Ålder
)
;
Endproc

/* 4. Skador. Extrahera vissa fält och gör om datum, t ex 2005-05-01 till 20050501.
Sortera på Kund Avt Obj Momrad Skadedat Diagnos3, där sista sortfältet är de tre
första tecknen i Diagnos. */
Proc Data ;

```

```

Infiler fil(UUIEX_STATSKADOR_$01.TXT) dlm(9)
var(
  Ersår Skadenr Hjs Momrad Delmom Delersnr Kund Avt Obj Skadedatin $
  Bruttoers Nettoers Sktyp $ Avttyp $ Objtyp Fp $ Mtyp Mvillk Orsakskod $
  Diagnos $ DjurklinikB $ DjurklinikO $ Avvisad $ AvvisadRad $ Kön Anvkod $
  Ras $ Bolag $ Provisionsbolag $ Provisionsombud $ Postnr $ Tlän $
  Tariffkommun $ Premiekl Sjrpr Åldergr Utbetdatin $ Statskadedatin $ Ffdatin $
  Fromdatin $ Nytedatin $ Tdatin $ Födelsedatin $ Betäcktdatin $ Fbel
  Fölatdatin $ Manber $ Antal_djur_areal Sjrkr Utvslaktv Proratap Reduktionp
  Skadad_gröda_kg Sjrantal Ant_ersbara_djur Regnr $ Objverdatin $ Objverkod $
  Uppläggningsdatin $ Län $ Kommun $ Församling $ Justering $ Sjrrab
)
dvar(
  Skadedat      = remove(Skadedatin, '-')
  Utbetdat      = remove(Utbetdatin, '-')
  Statskadedat  = remove(Statskadedatin, '-')
  Uppläggningsdat = remove(Uppläggningsdatin, '-')
  Diagnos3 = substr(Diagnos, 1, 3)
)
;
Utfil fil(tmpls1.txt) dlm(9)
var(
  Kund Avt Obj Momrad Skadedat Diagnos3 $
  Ersår Skadenr Hjs Delmom Delersnr Bruttoers Nettoers Sktyp $ Orsakskod $
  Diagnos $ DjurklinikB $ DjurklinikO $ Utbetdat Statskadedat Manber $ Sjrkr
  Proratap Reduktionp Sjrantal Uppläggningsdat Sjrrab
)
sort(Kund Avt Obj Momrad Skadedat Diagnos3)
;
Endproc

/* 5. Skador. Summera på ersättningsbeloppen på ID Kund Avt Obj Momrad Skadedat Diagnos3.
   Skador med samma ID skall betraktas som samma skada. Fälten får namnen Bruttoerstot
   och Nettoerstot i nästa proc. En indikator Raknas visar vilka poster som skall
   representera den enda skadan. */
Proc Sum ;
Infiler fil(tmpls1.Txt) dlm(9)
key(Kund Avt Obj Momrad Skadedat Diagnos3)
var(
  Kund Avt Obj Momrad Skadedat Diagnos3 $
  Ersår Skadenr Hjs Delmom Delersnr Bruttoers Nettoers Sktyp $ Orsakskod $
  Diagnos $ DjurklinikB $ DjurklinikO $ Utbetdat Statskadedat Manber $ Sjrkr
  Proratap Reduktionp Sjrantal Uppläggningsdat Sjrrab
)
;
Utfil fil(tmpls2.txt) dlm(9) Var(Bruttoers Nettoers) ;
Endproc

/* 6. Skador. Summerade ersättningsbeloppen överförs till alla skadeposter. */
Proc Match;
Masterfil fil(tmpls2.Txt) dlm(9)
key(Kund Avt Obj Momrad Skadedat Diagnos3)
var(Kund Avt Obj Momrad Skadedat Diagnos3 $ Bruttoerstot Nettoerstot)
;
Transfil fil(tmpls1.Txt) dlm(9)
key(Kund Avt Obj Momrad Skadedat Diagnos3)
var(
  Kund Avt Obj Momrad Skadedat Diagnos3 $
  Ersår Skadenr Hjs Delmom Delersnr Bruttoers Nettoers Sktyp $ Orsakskod $
  Diagnos $ DjurklinikB $ DjurklinikO $ Utbetdat Statskadedat Manber $ Sjrkr
  Proratap Reduktionp Sjrantal Uppläggningsdat Sjrrab
)
dvar(
  /* Raknas = 1 för en enda post, den första, av flera med samma
     Kund Avt Obj Momrad Skadedat Diagnos3. Annars 0. */
  Raknas = 1 - Ind0(Kund - prev(Kund)) *
    Ind0(Avt - prev(Avt)) *
    Ind0(Obj - prev(Obj)) *

```

```

Ind0(Momrad - prev(Momrad))*
Ind0(Skadedat - prev(Skadedat))*
Indc(Diagnos3,prev(Diagnos3))
)
;
Utfil fil(tmpos3.txt) dlm(9)
var(
  Kund Avt Obj Momrad Skadedat Diagnos3 $
  Ersår Skadenr Hjs Delmom Delersnr Bruttoers Nettoers Sktyp $ Orsakskod $
  Diagnos $ DjurklinikB $ DjurklinikO $ Utbetdat Statskadedat Manber $ Sjrkr
  Proratap Reduktionp Sjrantal Upplägningsdat Sjrrab
  Raknas Bruttoerstot Nettoerstot
)
;
Endproc

/* 7. Skador. Överför försäkringsinfo och beräkna periodindex. */
Proc Match ;
Masterfil fil(Agrfors_$01.txt) dlm(9)
key(Kund Avt Obj Momrad) timekey(Fromdat)
var(
  Kund Avt Obj Momrad Fromdat Todk
  Avttyp $ Objtyp Fp $ Mtyp Mvillk
  Ffdat Nytedat Tdat Föddat Postnr
  Termin Bolag Kön Anvkod Ras $
  Annuorsak $ Sjrpr Fbel Årspremie Ålder
)
firstobs(2)
;
Transfil fil(tmpos3.Txt) dlm(9)
key(Kund Avt Obj Momrad) timekey(Skadedat)
var(
  Kund Avt Obj Momrad Skadedat Diagnos3 $
  Ersår Skadenr Hjs Delmom Delersnr Bruttoers Nettoers Sktyp $ Orsakskod $
  Diagnos $ DjurklinikB $ DjurklinikO $ Utbetdat Statskadedat Manber $ Sjrkr
  Proratap Reduktionp Sjrantal Upplägningsdat Sjrrab
  Raknas Bruttoerstot Nettoerstot
)
dvar( Periodinx = Indi(Skadedat,$02,$02+9999) +
      2*Indi(Skadedat,$02+10000,$02+19999) + 3*Indi(Skadedat,$02+20000,$02+29999) )
;
Utfil fil(Agrskad_$01.txt) dlm(9) headerline noq
noempty /* med noempty läggs en blank ut i stället för ett tomt fält, för Proc Sasin */
var(
  /* 1. Ursprungliga skadefält */
  Kund Avt Obj Momrad Skadedat Diagnos3 $
  Ersår Skadenr Hjs Delmom Delersnr Bruttoers Nettoers Sktyp $ Orsakskod $
  Diagnos $ DjurklinikB $ DjurklinikO $ Utbetdat Statskadedat Manber $ Sjrkr
  Proratap Reduktionp Sjrantal Upplägningsdat Sjrrab
  /* 2. Härledda räknas-indikator och aggregerade skadebelopp */
  Raknas Bruttoerstot Nettoerstot
  /* 3. Försäkringsversionsfält */
  Fromdat Todk
  Avttyp $ Objtyp Fp $ Mtyp Mvillk
  Ffdat Nytedat Tdat Föddat Postnr
  Termin Bolag Kön Anvkod Ras $
  Annuorsak $ Sjrpr Fbel Årspremie Ålder
  /* 4. Periodindex 1-3 */
  Periodinx
)
;
Endproc

/* 8. Skador. Ny fil med bara de poster som har Raknas = 1. Var() utelämnad i
  utfilsatsen och då tas alla fält. Filen skall gå in i Proc Taran. */
Proc Data;
Infiler fil(Agrskad_$01.txt) dlm(9)
var(

```

```

Kund Avt Obj Momrad Skadedat Diagnos3 $
Ersår Skadenr Hjs Delmom Delersnr Bruttoers Nettoers Sktyp $ Orsakskod $
Diagnos $ DjurklinikB $ DjurklinikO $ Utbetdat Statskadedat Manber $ Sjrkr
Proratap Reduktionp Sjrantal Uppläggningsdat Sjrrab
Raknas Bruttoerstot Nettoerstot
Fromdat Todk
Avttyp $ Objtyp Fp $ Mtyp Mvillk
Ffdat Nytedat Tdat Föddat Postnr
Termin Bolag Kön Anvkod Ras $
Annuorsak $ Sjrp Fbel Årspremie Ålder
Periodinx
)
firstobs(2)
urval(Raknas = 1)
;
Utfil fil(Agrskad_raknas_1_$01.txt) dlm(9) headerline ;
Endproc

```

Example 5

```

Include C:\Rapp\Rpp\Init.Rpp

Proc Match ;
Masterfil fil(C:\Rapp\Data\Breeds.Txt) dlm(9) firstobs(2)
var(cBreedDescription $ Breedgroup Groomn Lifeage) key(cBreedDescription) ;
Transfil fil(C:\Rapp\Data\Fors0.Txt) dlm(9) firstobs(2)
var(
  Include PolicyPremiumYear3.Rpp
  cPostcode $
  Area
)
key(cBreedDescription)
;
Utfil fil(C:\Rapp\Data\tmpf1.Txt) dlm(9) headerline unmatched noq stats
var(
  Include PolicyPremiumYear3.Rpp
  cPostcode $
  Area
  Breedgroup
  Groomn /* 0: Okänd 1: Little 2: Moderate 3: Considerable */
  Lifeage
)
;
Endproc

Proc Data ;
Infiler fil(C:\Rapp\Data\tmpf1.Txt) dlm(9) firstobs(2)
var(
  Include PolicyPremiumYear3.Rpp
  cPostcode $
  Area
  Breedgroup
  Groomn
  Lifeage_in
)
dvar(
  Age = min(99,|[(nfrdk-nAnimalDateOfBirth)/10000]|)
  IncAge = min(99,|[(nPolicyInceptionDate-nAnimalDateOfBirth)/10000]|)
  Kastrat = min(1,max(0,|lAnimalSpayed|,|lAnimalNeutered|))
  /* 0 ej kastrat, 1 kastrat */
  Sexx = replace(cAnimalSexID,"MF","mf")
  SexCastr = Sexx!!chr(kastrat)
  /* M0 hane ej k, M1 hane kastrat, F0 hona ej k, F1 hona kastrat, */
  Birthyear = [nAnimalDateOfBirth/10000]
  Renewalyear = [nFrkd/10000]
  Lifeage = 10*Ind0(Lifeage_in) + Lifeage_in
  /* Om Lifeage_in = 0 är det en omatchad rad och då skall värdet vara 10 år. */
  Lifecover = Indp(Birthyear+Lifeage-Renewalyear)
)
;

```

```

Utfil fil (C:\Rapp\Data\tmpf2.Txt) dlm(9) headerline
var(
  nAnimalTypeID /* 1 Hund, 2 Katt */
  nFrdk
  nTodk
  nArsprem R
  /* Premie för API + BO före skatt. 0.95*nArsprem är premien efter skatt. */
  Breedgroup
  Groomn /* 0: Okänd 1: Little 2: Moderate 3: Considerable */
  Age
  IncAge
  SexCastr $
  Area
  cPaymentFrequencyID $
  Lifecover
)
;
Endproc

Proc Durber infil(C:\Rapp\Data\tmpf2.txt) utfil(C:\Rapp\Data\Fors1.txt) firstobs(2)
headerline
var(
  nAnimalTypeID
  nFrdk
  nTodk
  nArsprem R
  Breedgroup
  Groomn
  Age
  IncAge
  SexCastr $
  Area
  cPaymentFrequencyID $
  Lifecover
)
  frdkvar(nFrdk) todkvar(nTodk) ypremvar(nArsprem) /* datum(20000101 360 9) */
Endproc

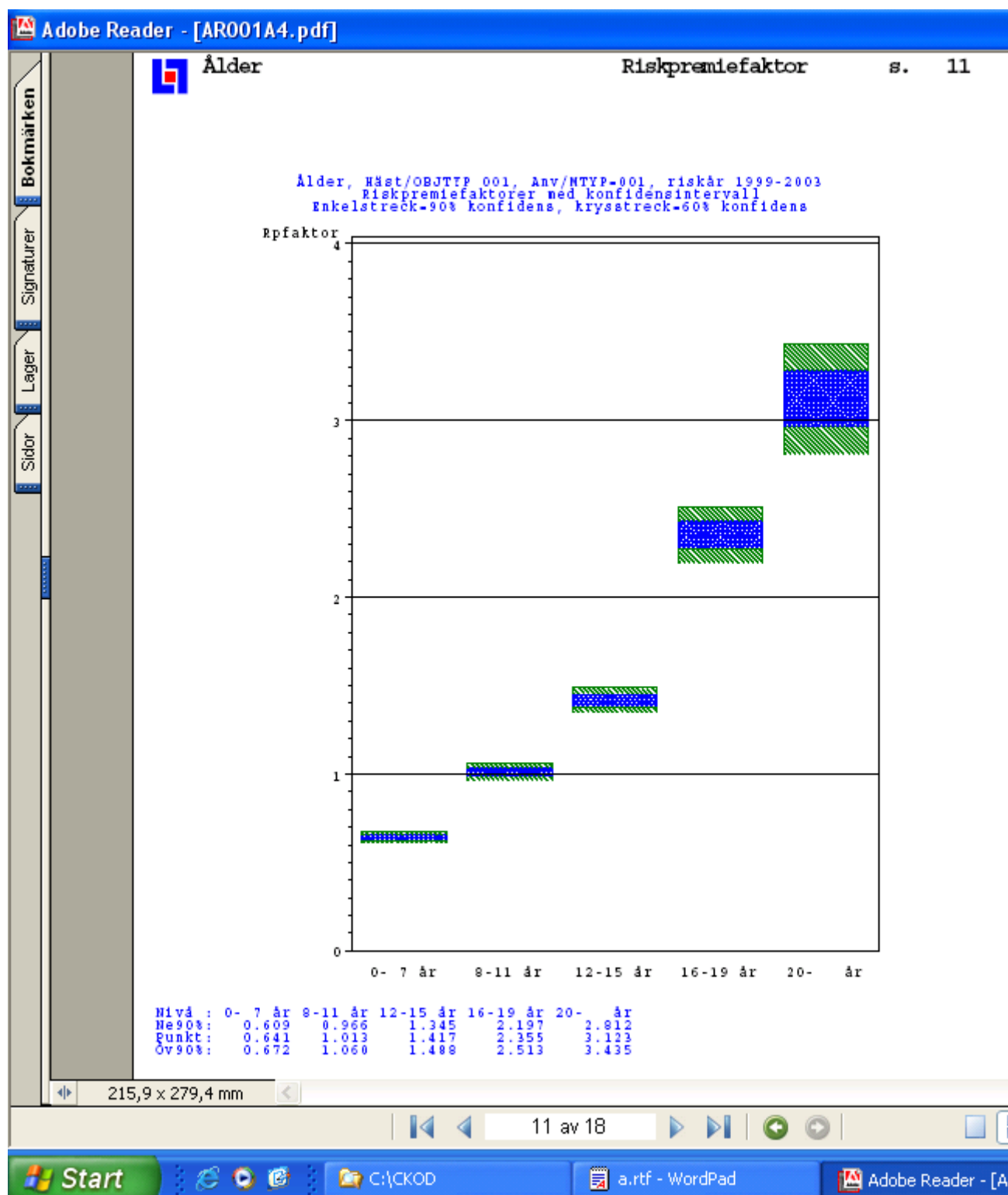
system(del C:\Rapp\Data\tmpf1.Txt & del C:\Rapp\Data\tmpf2.Txt)

```

Examples of graphs in PDF

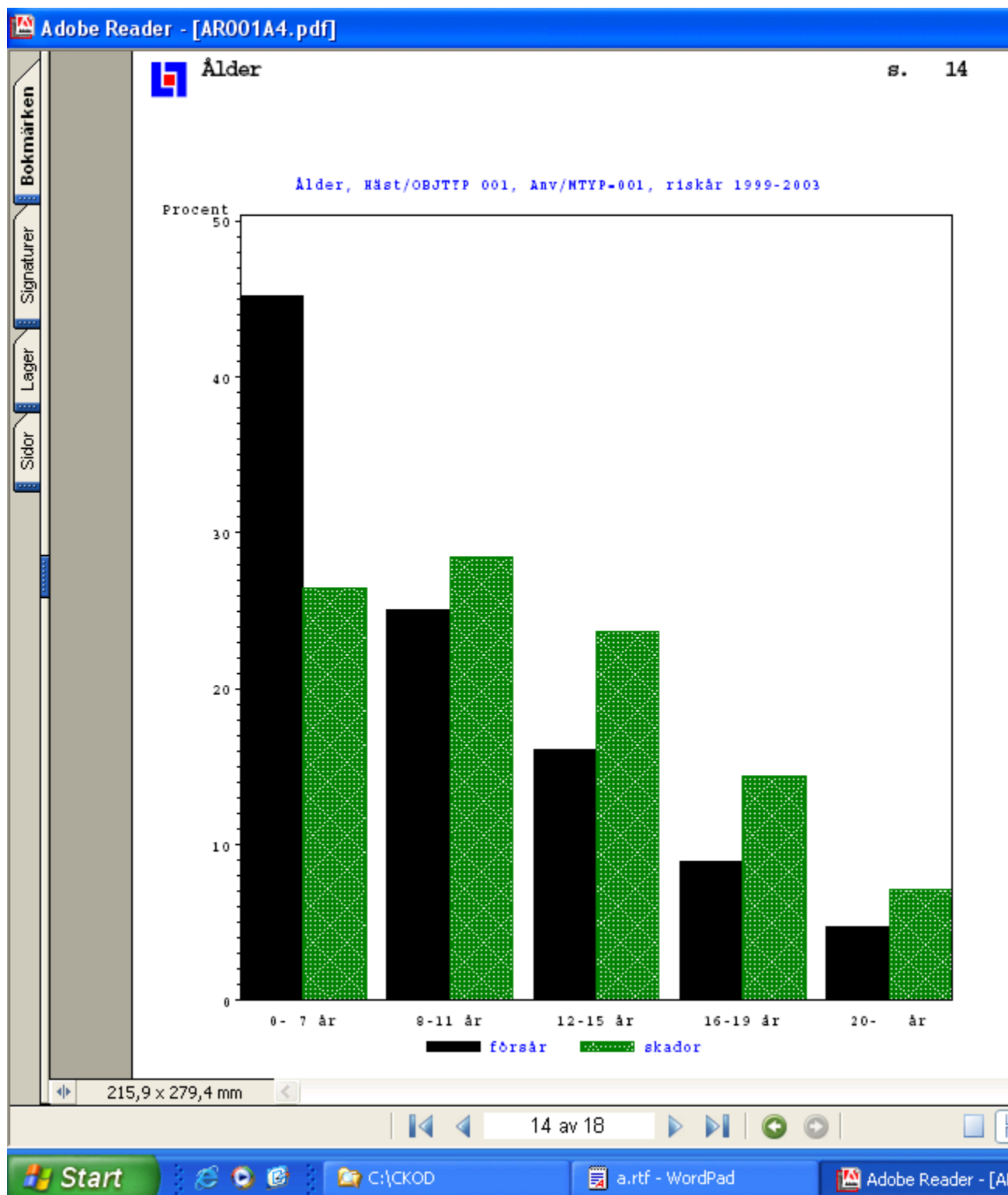
Example 1:

Risk premium factors for age ranges



Example 2:

Portfolio exposure breakdown of age ranges



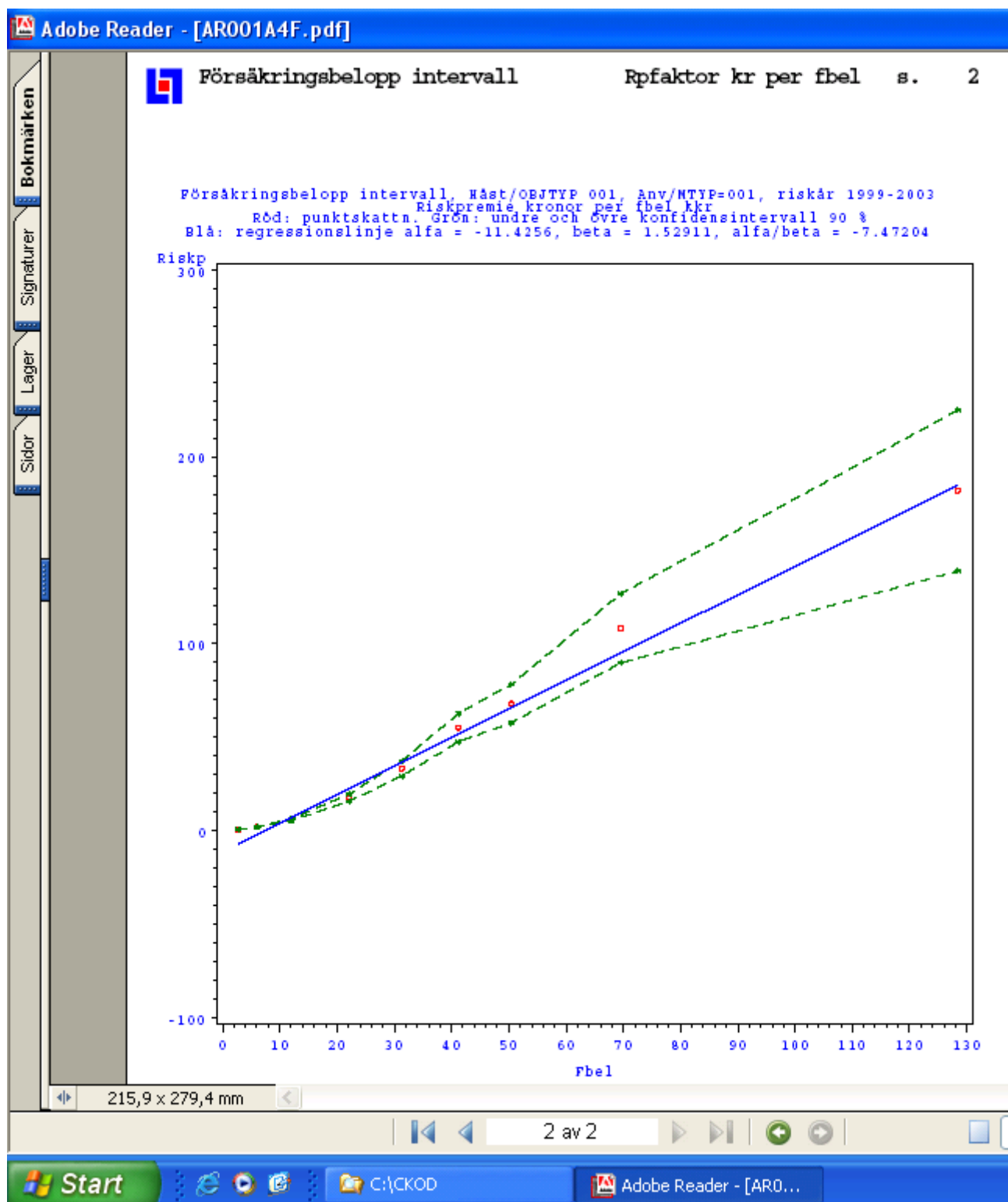
Example 3:

Interval assignment of insurance coverage, list section for Proc Graf with parameter B, regression analysis of risk premium SEK as a linear or broken linear function of fbel. Here linear, with all points (medelfbel, risk premium factor) included.

Påskäringsbelopp intervall (indelning nr 4), Rikt/CROTIP 001, Års/MTD=001, riktår 1999-2003														
Intervallindelning av Påskäringsbelopp Rikt														
Rikt	Antal Rikt	Antal Rikt	Antal Rikt	Antal Rikt	Antal Rikt	Antal Rikt	Antal Rikt	Antal Rikt	Antal Rikt	Antal Rikt	Antal Rikt	Antal Rikt	Antal Rikt	Antal Rikt
0- 3.999	10210	159	466	15.6	1.2	2056	44	17.47	8.0630	1.7229	57.59	50.3	2.6174	0.2019
4- 9.999	78990	1248	7967	16.1	0.5	6262	43	17.16	1.9054	6.0289	26.59	44.0	1.5011	0.2340
10-19.999	121461	2404	20469	16.3	0.4	11674	55	19.29	1.0025	11.5521	39.40	49.1	1.1674	0.4021
20-29.999	72904	2238	51597	21.5	0.7	11716	56	22.26	1.0071	11.0901	54.42	59.5	1.1126	0.7996
30-39.999	47785	2016	61675	42.1	0.9	21694	55	42.63	1.2229	21.2542	61.62	66.6	1.0592	1.0527
40-49.999	21211	1102	45720	32.0	1.6	41451	68	32.34	2.0125	41.2597	67.79	77.1	1.0179	1.2281
50-59.999	15973	750	21225	45.5	1.7	50561	60	45.56	2.6341	50.4308	67.12	67.9	0.8452	1.2474
60-69.999	10284	604	41597	58.7	2.4	62515	401	58.70	4.1099	69.5459	75.72	73.6	0.7103	1.2973
100 -	7150	285	22062	40.0	2.4	115672	2774	26.02	6.2194	116.4129	64.26	56.1	0.2696	1.4171
Totalt	295979	10905	214796	27.6	0.2	18865	221	26.84	1.2262	11.5925	57.54	63.6		

Example 4:

Interval assignment of sum insured, graph.



To not have negative premiums we recommended a minimum insurance amount 10 thousand SEK.

Quick guide - short manual by example

A. Getting started

1. Installing LaTeX at your employer

Make sure that LaTeX is approved as supplementary program and install it.

2. Installing LaTeX at home

See AllInOne.htm at my site. Or Google MiKTeX and install MiKTeX, which contains LaTeX, from one of the sites that come up.

3. Make C:\Rapp\Rpp\Init.Rpp

Make the file C:\Rapp\Rpp\Init.Rpp with this or modified content:

```
Proc Init
  pathdir(C:\Rapp\Pgm)
  // SAS interpreter: example - remove if SAS not available.
  sasexe(C:\Program Files\SAS\SAS 9.1\sas.exe)
  logo(ri) tempmapp(C:\Rapp\Jt) antmb(3000) priooffset(0) erralarm(5)
  pdfoffset(14) xgfact(1.03) ygfact(1.03) xgtran(-15) ygtran(3)
Endproc
```

Rapp creates the folder within tempmapp() if it does not exist. Change tempmapp(C:\Rapp\Jt) to another one if C:\Rapp\Jt is not appropriate. The tempmapp should reside on the C-disk, otherwise Rapp will run much slower.

Within sasexe() shall be the program used to run SAS with, if you need to run SAS inside Rapp via the procs Sas, Sasin, Sasut.

4. Make Rapp.Exe executable with command Rapp at the Command prompt

Go to the Command prompt. In the folder you come to, make Br.Bat (mnemonics for Begin Rapp) by writing Notepad Br.Bat.

```
C:\Users\Youruser\Br.Bat
-----
@echo off
echo You run exe-files primarily from C:\Rapp\Pgm.
set "FromCmdPrompt=Yes"
path=C:\Rapp\Pgm;%PATH%
CD C:\Rapp\Rpp
```



Rappenv.pdf

An instruction how to do it and a way to write Rapp programs:



See also C:\Rapp\Dok\Rapp-run-and-edit.pdf for more info.

Can be accomplished with C:\Rapp\Pgm\Adapt.Exe after Rapp.Zip is unpacked:

Running C:\Rapp\Pgm\Adapt.Exe requires administrator status. It is harmless to run, provided you answer N at the prompt. Answering Y accomplishes what is described below. I cannot guarantee that it will not harm the computer, although I have run it myself without problems. Possible harm might result from bungled authorizations, but should not occur if Adapt.Exe and the generated program Assoc.Bat are run by a user with full administrator authorizations.

5. Associate extent .Rpp to Rapp.Exe in Windows Explorer

(Right)click in Windows Explorer on a Rapp-program with extent .Rpp (for example Init.Rpp), select Open / Choose Program / Browse / locate Rapp.Exe and tick Always use this program. Then you can run a Rapp-program by click + return in Windows Explorer.

6. Change opening program for .rpp at click and return in Windows Explorer
If you moved Rapp.Exe to a different folder than where it was at step 4, do this. Take care so nothing is sabotaged. Double-click icon for pictures:



Changepath.pdf

Description in words. Assume we made Rapp.Exe default opening program for files type .rpp.

1. Right-click cmd.exe in C:\Windows\System32. Select Run as administrator.
At a workplace it must probably be an IT-person.
2. Command Regedit.
3. In the list that pops up, select HKEY_CLASSES_ROOT.
4. Then you get a list of a number of filextents where .rpp is and then a number of names in more plain language where Rpp_auto_file is.
5. Locate Rpp_auto_file\shell\open\command.
6. Right click the entry (Default) and select **Modify** to change the path in Edit String.
7. Close a few times.

7. Change icon and description for extension .rpp in Windows 7

This is not needed for the function but can make the appearance in the Explorer look flashier. Since Rapp is not installed this task is up to the user. See picture. The icon file Rapp.ico is in the folder C:\Rapp\Pgm.



Chaicon.Pdf

Change alternative opening program shown at right-click / Open with

Assume we made Rapp.Exe alternative opening program for a collection of file types such as .txt and .sas.

1. As above.
2. As above.
3. In the list that pops up, select
HKEY_CLASSES_ROOT\Applications\Rapp.Exe\shell\open\command
4. Right click the entry (Default) and select **Modify** to change the path in Edit String.
5. Close a few times.

End - can be accomplished with C:\Rapp\Pgm\Adapt.Exe.

If you want your own icon, make a BMP file and rename it to extent .ico. For making icons that can be favicon.ico on a web site you can use this link. It seems to work best with PNG files. <http://ico.bradleygill.com/index.php>

B. Making input data for Rapp from Excel



Making-data-for-Rapp.pdf

C. Using Proc Taran and Proc Graf - user manual with examples

Example 1, combined insurance and claim data

A SAS table Forsskad in the folder J:\AA\BB contains combined insurance and claim data and has among others the variables

kkl kon frdk foddad boende fordald bmkl mvikteff

dur durprem skadkost skadkvadr skfr Major durprem skadkost skadkvadr skfr

Here skadkvadr is summed squares of individual claim costs. For example, if antskad = 2 and skadkost = 20000, which is the sum of 12800 and 7200, then

$skadkvadr = 12800^2 + 7200^2 = 215680000$. The variable skfr contains antskad/dur, ie claim frequency.

The variable boende is alphanumeric and can contain letters, while all other variables are either numeric or alphanumeric with only figures as content.

Of the arguments we want to group the vehicle age fordald to fewer classes than in the data, while the other arguments are presented with all values in the data.

In Proc Sasut we then create a text file with semicolon separated fields, which is necessary if certain fields can contain only or interspersed blanks.

In Proc Taran we name the sum- and average-fields by the right reserved field names, which are

Dur	duration
Fbel	sum-insured-duration (insurance coverage under yearly risk)
Ndur	normed duration
Prem	duration premium (earned premium)
Antskad	number of claims
Skkost	claim cost
Kvadr	square sum of individual claim costs

and some ratios of these, for example Skkost/Antskad.

Required of these are only Dur and Skkost. If Antskad does not exist as a field, then is assumed that each line contains one claim. Similarly if Kvadr does not exist.

Rapp-program

```
Include C:\Rapp\Rpp\Init.Rpp
```

```
Proc Sasut libname(J:\AA\BB) tabell(Forsskad) delimiter(';')
textfil(C:\s\z1.txt) var(kkl kon frdk foddatt boende fordald bmk1 mvikteff
    dur durprem skadkost skadkvadr skfr)
Endproc
```

```
Proc Taran listfil(v1.txt) textfil(v2.txt) crosslist(v4.txt) crossarg(kkl bmk1)
nummetod(n) rub62 'Exempel 1';
```

```
infil fil(C:\s\z1.txt) var(kkl kon frdk foddatt boende $ fordald bmk1 mvikteff
    dur prem skkost kvadr antskad/dur)
dvar(
    Ålder      = min(99,|[(frdk-foddatt)/10000]|)
    Könålder   = 100*(kon-1)+ålder
)
delimiter(';') ;
arg(kkl) ;
arg(Könålder) ;
arg(boende) ;
arg(fordald) rub30 'Fordonsålder'
niv(
    ( , 0:2 '0-2')
    ( , 3 '3')
    ( , 4 '4')
    ( , 5 '5')
    ( , 6 '6')
    ( , 7 '7')
    ( , 8 '8')
    ( , 9 '9')
    ( ,10 '10')
    ( ,11:99 '11-')
```

```

) ;
arg(bmkl) ;
arg(mvikteff) ;
TEXT
"Faktorer frekvens" och "Faktorer riskprem" har fåttts ur ekvationssystem, som
renskar bort inflytandet från andra argument än det tabulerade. "Faktorer
riskprem" är lämplig att använda för premiesättning, dock med hänsyn till
kolumnen Rfaktospct (osäkerhet i %).
ENDPROC

```

```

Proc Graf listfil(v1.txt) pdfil(v1.pdf)
  r f m u s pos[13 1_pie_2% 13 Antal$förår]
ENDPROC

```

Example 2, separate insurance and claim data

A SAS table Forsakr in the folder J:\AA\BB contains insurance data and has the variables

```

kkl konald boende fordald bmkl mvikteff dur durprem

```

A SAS table Skad in the folder J:\AA\BB contains claim data and has the variables

```

kkl konald boende fordald bmkl mvikteff antskad mskad mskadkvadr
where mskad is mean claim = skkost/antskad and mskadkvadr is mean claim square =
kvadr/antskad.

```

Rapp-program

```

Include C:\Rapp\Rpp\Init.Rpp

```

```

Proc Sasut libname(J:\AA\BB) tabell(Forsakr) delimiter(';')
textfil(C:\s\z1.txt) var(kkl konald boende fordald bmkl mvikteff dur durprem)
Endproc

```

```

Proc Sasut libname(J:\AA\BB) tabell(Skad) delimiter(';')
textfil(C:\s\z2.txt) var(kkl konald boende fordald bmkl mvikteff
  antskad mskad mskadkvadr)
Endproc

```

```

Proc Taran listfil(v1.txt) textfil(v2.txt) nummetod(n) rub62 'Exempel 2';
infil fil(C:\s\z1.txt) var(kkl konald boende $ fordald bmkl mvikteff dur prem)
  delimiter(';') ;
infil fil(C:\s\z2.txt) var(kkl konald boende $ fordald bmkl mvikteff
  antskad skkost/antskad kvadr/antskad) delimiter(';') ;
arg(kkl);
....

```

Example 3

Here an example that can be run to verify that the technical stuff works after LaTeX was installed and C:\Rapp\Rpp\Init.Rpp was made. Export the following as text files to the working directory with the names given.

f1.txt

Kod	Bilmärke	Tätgrad	Dur	Intjprem
1	Jaguar	Tätgr-000	2500	3345090
1	Jaguar	Tätgr-025	5000	5578000
1	Jaguar	Tätgr-050	750	600000
1	Jaguar	Tätgr-100	2345	1678050
1	Volvo	Tätgr-000	4000	4000000
1	Volvo	Tätgr-025	2000	1500000
1	Volvo	Tätgr-050	500	750000
1	Volvo	Tätgr-100	1000	223450
2	Jaguar	Tätgr-000	3500	1489045
2	Jaguar	Tätgr-025	1234	254500
2	Jaguar	Tätgr-025	3300	2254500

2	Jaguar	Tätgr-050	850	445000
2	Jaguar	Tätgr-100	2800	1906508
2	Volvo	Tätgr-000	4060	4500000
2	Volvo	Tätgr-025	1400	1345000
2	Volvo	Tätgr-050	400	250000
2	Volvo	Tätgr-050	400	250000
2	Volvo	Tätgr-100	500	750000

s1.txt

Kod	Bilmärke	Tätgrad	Antskad	Skkost	Kvadr
1	Jaguar	Tätgr-000	250	1500000	180000000000
1	Jaguar	Tätgr-000	250	1500000	180000000000
1	Jaguar	Tätgr-025	1500	18000000	4320000000000
1	Jaguar	Tätgr-050	560	7345000	192675089285
1	Jaguar	Tätgr-100	240	7645000	487050208333
1	Volvo	Tätgr-000	400	4000000	800000000000
1	Volvo	Tätgr-025	310	6000000	232258064516
1	Volvo	Tätgr-050	150	4312000	247911253333
1	Volvo	Tätgr-100	240	6000000	300000000000
2	Jaguar	Tätgr-000	500	3000000	360000000000
2	Jaguar	Tätgr-025	1120	9500000	161160714285
2	Jaguar	Tätgr-050	560	7005000	175250089285
2	Jaguar	Tätgr-100	240	7645000	487050208333
2	Volvo	Tätgr-000	300	3000000	600000000000
2	Volvo	Tätgr-025	310	6000000	232258064516
2	Volvo	Tätgr-050	120	2342000	914160666666
2	Volvo	Tätgr-100	120	3000000	150000000000
2	Volvo	Tätgr-100	120	2345500	124780000000

a.Rpp

```

Include C:\Rapp\Rpp\Init.Rpp
Proc Taran listfil(a.txt) rub62 'Test av Rapp';
infiler fil(f1.txt) var( Kod Bilmärke $ Geografi $ dur prem )
    urval( Bilmärke ne 'Bilmärke' );
infiler fil(s1.txt) var(Kod Bilmärke $ Geografi $ antskad skkost kvadr)
    urval(Bilmärke ne 'Bilmärke');
arg(Kod) tar(1.05 1.23); arg(Bilmärke) tar(0.6 1.2);
arg(Geografi)
    niv(
        (,'Tätgr-000': 'Tätgr-025' 'Glesbygd')
        (,'Tätgr-050' 'Tätgr-100' 'Tätort')
    )
    tar(0.75 1.25)
;
TEXT
Test av Rapp
Endproc
Proc Graf listfil(a.txt) pdfil(a.Pdf) t r f m u s visa
    pos[ 13 1_pie_2% 13 Antal$försår]
    pos[ 13 A_vbar 39 Skadefrekv 53 Medelskada 68 Riskpremie ]
    pos[ 13 1_color=red,green,blue,yellow 13
        Antal$försår 85 Intjänt$premie
        22 Antal$skador 30 Skadekostnad ]
Endproc
Proc Excel listfil(a.txt) xlmfil(a.xlm) visa Endproc

```

Example 4

Deductible simulation.

```

Include C:\Rapp\Rpp\Init.Rpp
Proc Taran listfil(a1.txt) nummetod(u)
    crosslist(a2.txt)
    crossarg(Ålder Kön)

```

```

crossvar(
  skkost1000 skkost2000 Bruttoerstot /* Primära summationsvariabler */
  sjpct1 = 100*skkost1000/Bruttoerstot /* Sekundära härledda variabler */
  sjpct2 = 100*skkost2000/Bruttoerstot
  Skk_kkr = Bruttoerstot/1000
  Antpct = 100*Antskad/Tot.Antskad
  Skkpct = 100*Bruttoerstot/Tot.Bruttoerstot
)
crossout(Antskad 7, Antpct 8.2 Skk_kkr 10 Skkpct 8.2 sjpct1 7.2 sjpct2 7.2)
// 12345678901 234567 2345.78 234567890 2345.78 234.67 234.67
crossrub('      Antal Procent      Skkost Procent      Proc      Proc'
  ' Ålder      skador av tot.      kkr av tot.      1000      2000')
rub62 'Självrisksimulering demo';
Infiler fil(C:\s\Agria\Agrskad_raknas_1_HAST2.txt)
var(
  Skadedat Diagnos3 $ Ersår Delmom Sktyp $ Orsakskod $ Diagnos $ Sjrkr
  Sjrrab Bruttoerstot Nettoerstot Fromdat Todk Mtyp Mvillk Postnr Bolag
  Kön Anvkod Ras $ Sjrp Fbel Ålder Periodinx Fbelintervall Omfinx
)
delimiter(9)
dvar( dur = 1 skkost = 1 // Dummies since dur and skkost must be present.
  skkost1000 = max(0,Bruttoerstot-1000)
  skkost2000 = max(0,Bruttoerstot-2000)
)
;
arg(Kön) ; arg(Ålder) ;
ENDPROC
Proc Graf listfil(a2.txt) pdfil(a2.Pdf) visa ENDPROC

```

D. Enter estimates into SAS

```

Include C:\Rapp\Rpp\Init.Rpp
PROC Sasin libname(J:\AA\BB) tabell(Skattn)
  textfil(v2.Txt) delimiter(';') firstobs(2)
  satser(drop Fbelndur Tarf;) /* Fbelndur, Tarf (tariffaktor) är 0 här. */
  var(Argnamn $char30. @32 Nivnamn $char10. Anr Ninr Dur Fbelndur Prem Antskad
    Skkost Ospmu Osp Basff Basfm Basfr Faktf Faktm Faktr Ospf Ospm Ospr Tarf)
ENDPROC

```

E. Enter estimates into Excel

```

Include C:\Rapp\Rpp\Init.Rpp
Proc Excel textfil(v2.txt) xlmfil(Skattn.xml) visa endproc

```

F. Other GLM-methods than MMT / marginal totals

Standard-GLM
Proc Taran s-glm ...

Tweedie model for risk premium
Proc Taran Tweedie(1.5) idgrupp(riskar bolag forsnr) ...

Proc Graf works also for list files from these methods.

Appendices - confidence intervals, multiclass analysis

Appendix 1 is from Scandinavian Actuarial Journal 2014:8.
Appendix 2 is from Scandinavian Actuarial Journal 2018:9, with an additional appendix 3.
Appendix 4 is from Astin Bulletin 40(1), May 2010. Statement by ASTIN's rules: *Copyright 2010 by ASTIN Bulletin and PEETERS s.a. Reproduced with Permission.*
Appendix 6 is from Astin Bulletin 42(1), May 2012. Same statement applies with year changed to 2012. Some elaborations after printing of the paper are included.

Appendix 8 treats Poisson approximation, no new results however.
 Appendix 12 will be published (as of November 2021) online on a web site of Taylor & Francis concerning Scandinavian Actuarial Journal. The last Appendix F is for Rapp use and will not appear on Taylor & Francis' site. Appendix 12 replaces an earlier appendix.

Appendices 1, 2, 4 and 6 are made by the author. They have the same page breaks but not the same line breaks as the Pdf's made by Taylor & Francis and PEETERS s.a., respectively. A few misprints have been corrected in Appendix 6. They are listed at the end.

Open jpg's by double-click.

Open Pdf's by either right-click / Acrobat document or double-click.

If the embedded Pdf's do not open, follow this procedure. Or some other way appropriate for new versions of Word.

Disabling Protected Mode in Adobe Reader X to enable reading embedded objects

1. Open Adobe Reader X.
2. From Edit (Redigera) menu select Preferences (Inställningar). Preferences dialog box appears.
3. Select General category (Allmänt) on the list, uncheck or remove tick mark for "Enable protected mode at startup (Aktivera skyddat läge vid start)".
4. Close and reopen the application and the embedded object should now open.

Appendix 1



Imp.Pdf

Appendix 2



Cpse4manual.Pdf

Appendix 3



Bs.Pdf

Appendix 4



Dpesttw.Pdf

Appendix 5



Tailmethods.pdf

Appendix 6



Bich.Pdf

Appendix 7



Assm.Pdf

Appendix 8



Polim.Pdf

Appendix 9



Gpdparm.Pdf

Appendix 10



AoS80-corr.jpg

Appendix 11



Portfolioopt.Pdf

Appendix 12



Hcr4manual.Pdf