User's Guide for Positive Matrix Factorization programs PMF2 and PMF3, Part 2: reference

Last changed on February 25, 2000

Contents

Introduction

These instructions are for the two programs PMF2 and PMF3. The reason for combined instructions is that the programs have similar control structures and similar usage patterns. The notation PMFx means either PMF2 or PMF3. We try to reserve the word "matrix" for two-dimensional arrays, and correspondingly the word "block" for 3-way arrays.
Before reading these instructions, you should become familiar with the information in the file or booklet "Getting started with PMF". This user's guide does not repeat all the "getting started" text.

The program PMF2.EXE solves approximately (in the Least Squares sense) the matrix equation

$$\mathbf{X} = \mathbf{GF} \qquad (1)$$

where **X** is known and **G** and **F** are unknown. Writing in component form and showing the residual matrix explicitly we get

$$x_{ij} = \sum_{h=1}^{p} g_{ih} \; f_{hj} \; + e_{ij} \qquad (2)$$

where std-dev(**X**) = std-dev(**E**) = **S** and where some or all elements of **G** and **F** are required to be non-negative. There are p "factors" in this model. One column of **G** and the corresponding row of **F** represent one factor. They correspond to the 'scores' and 'loadings' of the customary factor analysis.

When the residual matrix **E** is defined by

$$\mathbf{X} \; = \; \mathbf{G}\,\mathbf{F} + \mathbf{E}, \qquad (3)$$

the task of PMF2 may be expressed as minimizing the sum of squares

$$Q = \sum_{i=1}^{n} \sum_{j=1}^{m} (e_{ij} \; / \; s_{ij})^2 \qquad (4)$$

In the robust mode, this expression is modified so that the s_{ij} are dynamically readjusted (=iterative reweighting). We are also calling the value of Q by the name "chi2" but this is not quite correct. Strictly speaking, Q is not distributed according to the chi-squared distribution, although the distribution in many cases approximates chi2.

The "PARAFAC" model solved by the program PMF3 is best described in the following component form,

$$x_{ijk} = \sum_{h=1}^{p} a_{ih} b_{jh} c_{kh} + e_{ijk} \qquad (5)$$

Again there are p factors in the model, but in this case there are three entities (columns of **A**,**B,C**) forming one factor. And again the model is solved as a non-negatively constrained weighted (iteratively reweighted) Least Squares task.

Notation: we call the array of observed data by the symbol **X** both in PMF2 and in PMF3. Anticipating later needs, we denote the "fit" array by **Y** so that equation (3) may be written as **X**=**Y**+**E**, where the matrix **Y**=**GF**, and equation (5) is also **X**=**Y**+**E**, but now the block **Y** may be written symbolically as **Y**=**ABC**.

Special enhancements of the factor analytic models

For PMF3, two enhancements have been available. These enhancements have been difficult to use and the documentation has not been adequate. The same functionality is much better obtained by using the new flexible program "Multilinear Engine" (ME-1). For this reason, the explanation of these features has been deleted from this User Handbook.

Automatic background in PMF2. "Background" is an extra component which is fitted to the matrix X together with the factors. This feature may be useful for certain spectroscopic tasks. It is *not useful for typical environmental problems.* It is not available on the F side. The support of this feature will be dropped after the current version. Similar functionality is better obtained by using the multilinear program ME1. Details of this option have been removed from the user handbook.

Modelling of "errors"

Sometimes the residuals e_{ij} are caused by the errors of a measuring process in a straightforward manner. Then it may be possible to specify the standard deviations s_{ij} for measured data x_{ij} before running PMFx. This is the "easy" case. If the errors are caused by weighing the samples, then we have this easy situation.

It is more usual that the randomness is inherent in the process to be studied. Then it is not possible to deduce a reliable "error estimate" for an isolated measured value. An example is given by Poisson-distributed data, e.g. a set of low-intensity spectra measured by counting the radiation quanta, or an ecological study where a number of species are counted. If the count "2" has been observed, then the standard deviation for this value could be less than one unit or it could be over two units. The expectation values are needed in order to estimate the standard deviations or "error estimates". The unknown "true array" which is to be represented by the factor model approximates the expectation values or the "true values". We cannot solve the problem if we don't know the solution!

The programs PMFx solve this dilemma in the following iterative way: first rough approximations for the standard deviations s_{ij} are formed and an initial solution of the problem is computed. This initial factorization gives approximate expectation values (or other similar parameters) for the random variables. Then the program is able to compute better approximations for the s_{ij}, then a better factorization, and so on. The computations in PMFx are iterative in any case, thus the need to iterate the error model does not add significantly to the computational workload.

Another example of this kind are lognormally distributed data which are common in environmental studies. Each observed value x_{ij} may be regarded as a sample from a unique lognormal distribution. The geometrical mean values $\mu_{ij} = \sum_{h=1}^{p} g_{ih} \; f_{hj}$ of these distributions obey the factor model, thus

$$x_{ij} = \mu_{ij} + e_{ij} = \sum_{h=1}^{p} g_{ih} \; f_{hj} + e_{ij} \; . \qquad (6)$$

We view this as a maximum likelihood (ML) problem: the factor model should be determined so as to maximize the likelihood of the observed array **X**. It is possible to determine a Least Squares (LS) problem which is equivalent to the ML problem in the sense that the two problems have the same solution. When PMFx solves this equivalent LS problem then it in fact solves the ML problem and computes ML estimates of the original factorization problem. Equivalent LS problems for the ML solution of Poisson distributed and lognormally distributed factor models have been programmed in PMFx. The equivalence requires that PMFx computes iteratively the std-dev values s_{ij} so that the LS problem based on these s_{ij} is equivalent to the original ML problem. The equations for these std-values are given in the section describing the error models. Deriving these equations is by no means trivial, thus the end user may just take them as given.

Solving the lognormal model. The user has to specify the logarithm of the "geometrical standard deviation" (log(GSD)) for each measured value (often the same value is good for all measured points). This is sufficient if the distributions are really pure lognormals. Then the result is indeed a ML estimator. But here is a catch: the value zero cannot occur in a lognormal unless the distribution is degenerated to zero. In practice zero values often occur, e.g. because of concentrations below the detection limit. These values make the factorization unreliable if they are processed under the assumption of pure lognormality. Then it is necessary to modify the assumption of lognormality. PMFx offers the option to assume that there is an additional normal error superimposed onto the lognormal distribution. The detection limit of the measured values might be used as this additional error. The statistical properties of this model are not fully known, the solution is probably a good approximation of a ML solution, but it has not been proved to be strictly ML.

The expectation of the Q value does not equal its usual value (the degrees of freedom) in lognormal models. PMFx estimates the increase of the expectation value of Q and writes it in the .log file.

Installation of PMF2 and/or PMF3

Currently the PMF programs for 486-Pentium platforms are based on the Fortran 90 compiler LF90 by Lahey Computer Systems. The present instructions are for LF90.

The .EXE file(s) for PMF program(s) and the file lf90.eer should be downloaded from the ftp site and copied into any suitable directory which is included in the PATH. Depending on the version, the .EXE files may have different names, but the most usual names are e.g. PMF2OPT.EXE or PMF3TST.EXE. If you wish, you may rename the files to any other names when copying to your fixed disk. The file lf90.eer enables the Fortran system to write plain-language error messages. Also there is the Fortran error message list file rterrmsg.txt which contains the same information in a human-readable text file. You may look at this file with any text editor to determine the meaning of the numerical error codes. You will also need an authorization file or "key file", usually it will be emailed to you. The key file name should be pmf2key.key and/or pmf3key.key. The key file should preferably be copied to the directory C:\PMF or D:\PMF. It one file contains a licence both for PMF2 and for PMF3, then the file should be copied to the directory twice, under both of these names. Other possible places for the .key files are your working directory and two directory levels above it. The key file contains licencing information, e.g. information about the licencee and about the licencing period. This information is sealed with a numerical check code.

If you wish to run PMFx in Windows Dos box, you might perhaps set up the Dos startup command DOSPRMPT.PIF so that PMFx continues running while you switch away from Dos in order to perform editing or other tasks. Start up the "PIF EDITOR" and open for editing DOSPRMPT.PIF. Put an x mark in the box "Execution: Background", perform a "SAVE", and then "EXIT". If a Dos box is open, make an "EXIT" in it, then reopen the Dos box. Now Windows should continue running PMFx even while you are working elsewhere! But be careful **not to work** on those files which PMFx is either reading or writing to! In the readme file you may find additional information, e.g. about how to configure the dos box of Windows95.

PMFx auxiliary files (for PMF2, e.g. PMF2DEF.INI, GE.INI, GE2.INI, GE.DAT) and your own INI files should be copied into your own working directory. Last-minute information (e.g. warnings for known errors of PMF programs) may be found as a readme file or "release notes". Such a file might be called relnotes.txt or readme.txt or something like that. Check for the presence of such a file!

See the disclaimer at the end of this user's guide for important information!

The general working of PMF2 and PMF3

The following four stages can be identified. Stages 2 and 4 are described in detail in sidebars, below.

1. Reading of the .INI file, opening of input and output files
2. Reading arrays from input files
3. Calculations (usually with 3 different limit repulsion values or "penalty levels")
4. Writing results to output files (steps 2 to 4 may be repeated any number of times)

The complete sequence of all these steps, as outlined above, is seldom needed. These steps are controlled in detail by the file "mypmf.ini" where "mypmf" stands for any name selected by the user. This file is called "the .INI file" in this document. The PMF programs do not interact with the user at all, they only obey the definitions in the .INI files.

The .INI file is not simple, and it would be an error-prone task to write one from scratch. For this reason PMF programs first write a template .INI file called "PMF2DEF.INI" or "PMF3DEF.INI". This file contains definitions for an "average" PMFx run. It is suggested that the user should edit PMFxDEF.INI with a text editor and change some details according to his/her preferences, e.g.:

- shall the arrays be written in transposed layout or straight layout, how many digits are needed in the output: change the output layout and the formats accordingly

- are array headers useful or a nuisance in the output: they might help the human user, but if the data are to be input to a spreadsheet, say, the headers may be a nuisance. Arrange for writing or no writing of headers.

- how many different files are needed for input and for output. Is it desired that old info in output files is conserved or should old info be deleted. Arrange file designations accordingly. The only limitation is that any single file must be either for input only, or for output only. Otherwise one may configure input and output flexibly, e.g. the titles might be written to one file, and the arrays to another. In fact, each step a to d, A to C for each array in stages 2 and 4 is individually controlled.

Reading an input array (stage 2) may comprise the following steps

a. Reading a title for the array from an input file
b. Writing the title of the array to an output file
   c1. Reading the array itself from (the same or another) input file or
   c2. Generating the array internally within the program PMF2 or PMF3
d. Writing the input array itself to an output file

Writing a result array (stage 4) may comprise the following steps

A. Reading a title for the array from an input file
B. Writing the title of the array to an output file
C. Writing the array itself to (the same or another) output file

For unrelated tasks one should prepare separate .INI files. Under the control of one .INI file one may process a sequence of related tasks having common dimensions and common control information. It is possible to have some arrays input only once, to be used repeatedly in the sequence, and to have other arrays read or generated individually for each task in the sequence.

The .INI file for controlling PMFx

Maintaining your .INI files

There are two kinds of lines in any .INI file: comment (or title) lines, beginning with characters ##, and data lines, which must not contain the characters ##. When you update an .INI file, you should normally not change the number of lines in the file except for the section "optional parameters" . The program only makes use of the contents of the data lines. The comment lines are intended for helping the user; the program reads them but is not interested in their contents. Thus you *could* make your own annotations near the ends of the comment lines, but we recommend that you do not delete their original essential information. The very first ## line is intended as a general title of your task and PMFx copies it in the .log file and on the screen. It is recommended that you write descriptive information in this line after the initial key text "##PMF2" or "##PMF3". This key text must not be changed.

General run control items in the .INI file

"Monitor"

Default: 1
This parameter controls the amount of monitoring output generated by PMFx. With the default value every step is reported. But if Monitor=M>1 then the program writes (on screen and in the .log file) only on every M^{th} step. In routine work, one could perhaps have M=5. When trying to find an error, one might set M=0 or M=-1. In addition to the usual monitoring output, these two alternatives create diagnostic output, intended for the end user. Most of it go both to screen and to .log file, but some data is only written to the .log file. Other negative values of M produce special printouts, intended for finding errors in the program. The general principle is that more positive values produce less output. The value M=13 writes timing information at the end of .log output. The time value on the line marked "iterall" represents all the time spent in the iteration, excluding reading and writing of arrays. This time value might be used for comparing speeds of different computers and/or operating systems.

"Version of PMFx", compatibility

Default is currrently 4.2 for PMF2 and 4.2 for PMF3
This is a "version stamp". It will prevent old .INI files from being misunderstood by a newer version of PMFx programs. If you have an old .INI file and wish to use it with a newer PMFx program, then you might figure out how to update the file. Having performed suitable modifications, you should also change the version stamp. Otherwise, do not touch it! Usually new versions of PMFx also understand .INI files written for the previous version, see the release notes on the diskette. However, compatibility with still older .INI files is usually not possible. For this reason you should always update your .INI files when upgrading PMFx. If you use Monitor M=-1 then PMFx tries to write a partly updated .INI to the .log file (not guaranteed!). Extract/update the file by using a suitable text editor.
The current version PMF3 v.4.2 does understand .INI files written for the two previous versions 4.1 and 4.15. Similarly the current PMF2 v.4.2 understands versions 4.0 and 4.1

"Dimensions"

Defaults for PMF2: 40, 20, 4
Defaults for PMF3: 0, 0, 0, 0
The PMF2 default values are for the Gauss-Exponential test case. For other cases, you will have to change them. "Rows" and "Columns" are the dimensions n and m of the matrix X to be factorized. "Factors" is the number p of factors to be used by PMFx. Even one factor (p=1) may be a meaningful selection in some cases.
PMF3: the 3-dimensional data block "X" is visualized as a book: there are rows and columns on pages. The data is read and written page by page. The dimensions are: numbers of rows, columns, pages, and factors.

"Repeats"

Default: 1
PMF programs are able to compute several cases in one run, provided that they are controlled by a single .INI file. The .INI file is only read once, and the data files are also only opened once. Based on these, PMFx may process a number of similar cases, their number is given by "Repeats". Default is (Repeats=1), compute a single case. Different aspect may be varied between the repeated cases. Typical examples: different data sets may be analyzed. Or, one data set may be analyzed with different std-dev values or starting from different initial factor values. During the repeated runs, those lines of the I/O control table (see below) are performed again (= reading again, or forming random values again) whose (R) code has the value T or *true*.

"FPEAK"

(PMF2 only) Default: 0.0
Fpeak exerts control on the rotational state of the solution. The default prefers a "central" rotation.

"Robust mode"

Default: True
ROBUST is T (true) or F (false). Use T unless you know that the errors in your data are approximately normally distributed and that there are **no** outliers and **no** non-representative values in your data.

"Outlier threshold distance"

Default: α=4.0
When running in robust mode a measured value x_{ij} (or x_{ijk}) is processed as an outlier if

$$|x_{ij} - \Sigma_k\ g_{ik}f_{kj}\ |\ /\ s_{ij}\ > \alpha\ ,$$

in other words, if the residual exceeds α times the standard deviation. The "processing as an outlier" means that the std-dev value s_{ij} or s_{ijk} is increased so that the "pull" or influence of the outlying value x_{ij} or x_{ijk} is no more than the pull of a value which is on the limit of being classified as an outlier. This corresponds to the Huber estimation principle. —We suggest that the following values should be used as outlier threshold distance: α= 2.0, 4.0, or 8.0. By adhering to these standard values one makes it easier to compare PMF results obtained by different researchers. —Separate threshold distance values for positive and negative residuals may be specified by using the optional parameter "*outlimits* α_p α_n" (see below).

"Codes C1, C2, C3 and Errormodel (EM)"

The Errormodel code EM, the three codes C1, C2, C3 and the input arrays X_std-dev /T, - /U, and -/V work together in determining how the program reads and/or computes standard deviations S for the observed array X. The various alternatives are explained later in a dedicated section.

Pseudorandom "Seed"

The program may generate different sequences of pseudorandom numbers, depending on the chosen *Seed* value. There is no limit to the lengths of these sequences. Repeated analyses may be performed so that the generation of initial values is repeated. Then the new values are derived by continuing the sequence that was originally initiated by the *Seed* value.

"Stabilizer" and "Accelerator"

(PMF3 only) Defaults: 0.0, 0.0.
These parameters are leftovers from earlier PMF3 versions. Now they are ignored and should be =0.0.

Iteration control table for 3 levels of limit repulsion

The computation runs in three stages, with different limit repulsion values "lims". In fact, "lims" is a weight coefficient for a logarithmic penalty function acting on non-negatively constrained elements of the factors (G and F, or A, B, and C). Also, "lims" is a weight coefficient for regularization terms

which tend to prevent "wild" values and which take care of the implicit scaling of the factors during the iterations.

For each stage there is an end test. In the .INI file there are three lines of four values, one line for each stage. The values on the first line influence the ending of the first stage, the values on the third line end the whole computation. The last line of values influences the final result, whereas the first and second lines usually only influence the step count needed to reach the final result. One may experiment with the values in order to find a fast convergence. With typical .INI values the end test works as follows: Each stage is ended if there have been 4 (=Ministeps_ required) consecutive steps where the absolute value of the change of Q ("chi2") value was less than 0.1 (=Chi2_test) on each step. Also, the stage ends if the maximum allowed cumulative step count is exceeded for the stage (=Max_cumul_count). In order to study if the iteration has really converged to a true local minimum, one could run with a small third value for the Chi2_test, e.g. with 0.01. Large problems require larger values for Chi2_test!

The first two values for "lims" do influence the route which the minimization process follows in the many-dimensional space. If there are several local solutions to your problem, then changing the first (and perhaps also the second) lims value may result in a change of the result from one local optimum into another. It is tentatively suggested that large values of lims should be avoided in such cases.

The last (third) lims value affects how near to zero the constrained components of the solution may get. Possible values are from 0.1 to 0.001, say. Smaller values cause that the results may go nearer to zero.

Optional parameters

Near the end of the .INI file there is a place for optional parameters or "special information". This is used because of flexibility, in this way the format of the .INI files may be kept unchanged although new features are introduced. In later versions of PMFx some of the optional parameters may get their own standard places in the .INI file. The use of optional parameters is explained in detail elsewhere in the User's Guide. The following optional parameters are available now:

sortfactorsg Before output PMF2 orders the factors in a systematic order, based on values EV(G)
sortfactorsf PMF2 bases the ordering of factors on EV(F), in effect ordering the F factors.
Do not use both sort options (g and f) simultaneously!

sortfactorsa, sortfactorsb, sortfactorsc Similar sort options for PMF3, based on factors A, B, and C.

normfactorsac PMF3 normalizes the average values of A and C factors to unity.

missingneg r All negative values in **X** represent missing data. PMFx increases their specified std-dev values internally by the factor r. Typically use r=10.0 to r=100.0.

BDLneg r1 r2 (r1<0.0, r2>1.0) PMFx interprets negative values below r1 as missing values (r2 gives the desired increase of their std-dev values). Negative values between zero and r1 represent BDL indicators.

minreg r (PMF2 and PMF3) Normally both the logarithmic penalty and the regularization are controlled by "*lims*". The optional parameter "*minreg* r" controls the regularization so that regularization strength is max(*lims*, r).

outlimits α_p α_n The two decimal values α_p and α_n are used as two different outlier threshold distances, separately for positive and negative residuals. (Then the common outlier threshold distance α, specified earlier in the .ini file, is not used at all.)

Input/Output of arrays

Specifying file properties in the .INI file

The programs PMF2.EXE and PMF3.EXE may read from and write to several files, as specified by the user in the .INI file. Code numbers 30 to 39 have been reserved for input and output files. In addition, the input code 4 denotes the PMFx.INI file itself. Similarly, the output code 24 denotes the output file PMFx.LOG . Properties of the files 4 and 24 cannot be changed by the user.

Properties of the files 30 to 39 (in increasing numerical order!) are set by the .INI file:

1. each file is designated either input (T) or output (F).
2. by using the "file opening status" attribute of F90, one may specify how each file is opened. The alternatives and their requirements are: (see any Fortran 90 handbook):

   NEW a file with the specified name must not already exist on the disk. A new file is created (only for output files, of course)

   OLD a file with the specified name MUST already exist on the disk. If output, the old file is opened so that new output will be written at the end of the file. If input, the old file is opened for reading from the beginning of the file. For input files, OLD should always be used!

   UNKNOWN a combination of NEW and OLD: works both ways. Not for input, good for output

   REPLACE (only for output files): If there is an old file with the same name on the disk, it is deleted. The results are written into a new file! This is good if previous results are of no permanent value, and also if the previous results have already been copied to another file or directory.

3. the maximum length of lines in each file is specified. If the matrix F is stored straight, it may need long lines. So may the data array X and the std-dev arrays T,U,V.

   WARNING: depending on the properties of the compiler, there may be no error message if your input file contains long records (2500 characters, say) and you specify a shorter record (2000, say). It is possible that the extra characters (500) are simply chopped off the input records, and you cannot read the whole array correctly!

4. a name for each file is specified. If needed, you may also specify the path as part of the name. By using the character $ as one of the characters of a file name, you request file name substitution from command line. Assume that PMFx has been started by the command "PMFx mypmf monday", and that there is a file name "my$.dat" in the mypmf.ini file. Then name substitution will change the name into "mymonday.dat". This technique makes it possible to use one copy of the .INI file although several input or output files are needed at different times. File name substitution may not be available on some platforms.

In the default PMFxDEF.INI the definitions are such that files 30 to 33 are reserved for input and 34 to 39 for output. You are free to change this distinction, e.g. having 3 files for input and 7 for output. But we suggest that the smaller numbers should be reserved for input and the larger for output.

Using formats

There are 10 formats specified in the .INI file, with code numbers 50 to 59 (in increasing numerical order!). You may change them, but it requires knowledge of the FORTRAN90 language format system (essentially the same as in FORTRAN77). Each row of an array (or column, if transposed layout) is input by a separate READ statement or written by a separate WRITE statement. Thus the format should cover reading/writing of one row (not the whole array). Each new row will start using the format from beginning. For input it is often practical to use list-directed (="free-form") input. This is achieved by using the code number FMT=0 instead of the format numbers 50 to 59. — If there is an

error in your format specification, the error is only found when the program attempts to use the format, not earlier.

If there are quantities of widely differing magnitudes in different columns of the arrays, then write the results in E or G formats, e.g. E13.5E2, instead of F formats, such as F9.4. (The formats in PMFxDEF.INI are examples!) Otherwise zeroes appearing in the output may mislead you when the true value nevertheless could be significantly non-zero. In air pollution studies, the concentrations of lead are typically such small quantities; for Si there are large values.

Control of details of reading and writing

There are separate FIL codes for each possible input and output operation. The first alternative is that FIL = one of (4,24,30,31,..38,39). This means that the reading/writing is performed from/to the file whose code number =FIL. The second alternative is FIL=0. For headers and for output of arrays this means that this reading/writing is not performed at all. For input of arrays the marking FIL=0 indicates a special instruction, as detailed in the following table. And the third alternative FIL=1 (only for input of arrays) is also a special instruction, causing the array to be filled with the value 1 or 1.0.

If FIL=0 for	It causes the following operation instead of reading the array
X	the data array X is simulated
X_std-dev /T, /U, or /V	the array in question (T,U, or V) is set equal to the corresponding code C1, C2, or C3. See the section "Specifying the X_std-dev" for more details, e.g. correspondence analysis.
G and/or F	random initial values are generated into G and/or F. Similarly for A,B, and C.
Gkey and/or Fkey	zero values are generated into key matrices: all components of G and/or F are constrained to positive values. Similarly for Akey,Bkey, and Ckey.
rotcom	zero values are generated into "rotcom" matrix: no rotations are commanded.

If several arrays are to be read from one file, they must be present in the file in the same order as they appear in the array list in the .INI file and in the following tables. If something is read from the .ini file (FIL=4), those values must be at the end of the .ini file.

The marker FMT indicates format. FMT=0 means list-directed input or output. For input, FMT=0 is usually the best choice. The formats used for arrays "xkey" (x means G, F, A, B, or C) should be based on the integer conversion, e.g. I3. The F format is not possible for xkey. The FMT codes 50 to 59 specify format strings stored earlier in the .ini file.

Below the marker (T) one specifies if the array should be read or written "as is" ("straight") (T)=F, or transposed, (T)=T. For looking at results on the screen it may be good to have F transposed and G straight. For input arrays, use (T)=F or (T)=T depending on the layout of data in the original file.

Matrix processing order in PMF2

There may be echo output of 'IN' data immediately after input if so specified in the .ini file.

IN(+out)	the matrix of measured data (not input if X is to be simulated)	X
IN(+out)	the standard deviations for X	X_std-dev T, U, and V
IN(+out)	initial values for the left factor matrix	G
IN(+out)	initial values for the right factor matrix	F
IN(+out)	the key matrices for G and F	Gkey, Fkey
IN(+out)	the rotation command matrix	rotcom

OUT	simulated data matrix X and its true std_dev (output here only if X was simulated)	X, S
OUT	computed left and right factor matrices	G, F
OUT	standard deviations for G and F	G_std-dev, F_std-dev
OUT	explained variations of G and F	EV(G), EV(F)
OUT	the matrix of residual values	X - GF
OUT	the residuals scaled by std-deviations	(X - GF) / S
OUT	the robust/scaled residual matrix	(X - GF) / S(modif)
OUT	standard deviations of rotations	rotmat
OUT	the computed standard deviations for X (not output here if X was simulated)	S
OUT	coefficient matrix for the subtracted background, 1 or 2 rows of length n	

Array processing order in PMF3

There may be echo output of any 'IN' data immediately after input if so specified in the .ini file.

IN(+out)	the block of measured data (not input if X is to be simulated)	X
IN(+out)	data for computing the std-dev of X	X_std-dev T, U, and V
IN(+out)	initial values for three factor matrices also used as target values for binding factor elements to non-zero values	A, B, and C
IN(+out)	key matrices for factors	Akey, Bkey, Ckey
OUT	simulated data block X and its true std_dev (output here only if X was simulated)	X, S
OUT	the three computed factor matrices	A, B, C
OUT	standard deviations for computed factors	A_std-dev, B_std-dev, C_std-dev
OUT	the array of residual values	X - "ABC"
OUT	the residuals scaled by std-deviations	(X - "ABC") / S
OUT	the robust/scaled residual array	(X - "ABC") / S(modif)
OUT	the computed standard deviations for X (not output here if X was simulated)	S
OUT	the joint covariance matrix of A,B,and C	cov(vec(A|B|C))

For each array, there are first the specifications for reading the header: (FIL (R) FMT). The code (R) indicates whether the header should be read again during repeated tasks, (R)=T. The header of X is often an important bookkeeping tool.

The second group is for writing the header: (FIL FMT). If there is no reading but only writing of a header, then the default header is written. The default header is the last item on the line, you may change it.

The third group is for reading the array. The codes (R) and (C) control repeated tasks: If both are false, then the originally read array is also used in repeated tasks. If (R)=T, then another reading (or generating random values if FIL=0) happens when repeating the task. And if (C)=T, then the computed

factor matrix (G, F, A, B, or C) is used as the starting value for the next task. If there are no repetitions, then the codes (R) and (C) have no significance. (C) is short for "Chain".

The last group (FIL FMT (T)) controls the writing of arrays. For input arrays, this 'echo' writing may be meaningful for documentation ("what was being analyzed") or for error finding: if there is a read error when reading X, say, then by looking at the echo output of X, one may see where the good values end. The "unread" values contain the characters -9! But normally it is not necessary to write the input arrays, then their output FIL codes may be zero.

Specifying initial values of factors

1. PMFx may read initial values for factor matrices (G and F, or A, B, and C) from a file. Then the FIL codes for them must be >1. If there is no *a priori* information for the factors, then one may set all the initial values equal to 1.0 by having FIL=1. The value 0.0 is not suitable as a general-purpose initial value for factors.
2. PMFx may also generate initial random values for these factor matrices. Then their FIL codes must be zero. This is the default.
3. In repeated computations PMFx may read a new set of initial values, or it may generate another set of random values (according to FIL) if the repeat-code (R) is true. There is also a third alternative: if (C) "Chain" is true, and (R) is false, then the results (factor values) from the previous computation are used as starting values for the next. And if both (R) and (T) are false for G and F, then the original starting values are used for all the repeated computations.

The reasons for using random values. Use random starting values of factors in order to	Drawbacks connected with random starting values of factors:
1. suppress any personal preferences towards a certain solution. This maximizes the objectivity of the analysis. 2. generate many different starting points for repeating the same computation. This attempts to find out if there are local minima of the Q function.	1. The factors appear in random order. This may be partly corrected by using one of the optional parameters "sortfactorsx" where x stands for g, f, a, b, or c. 2. The iteration may need more steps when started from a "poor" starting point.

Generally one should use random values for such analyses which are not part of a sequence or group. Also one should investigate a few samples from each group of analyses by re-running several times with different random values. This requires having FIL=0, (R)=true for factors in a repeated analysis.

When analysing a set of similar arrays, there are two alternatives to random starting:

1. Take the result file(s) (G and F) from one typical analysis and read them (FIL>1) as starting values when doing all the other analyses. This is fast and creates the factors in the same order.
2. Form artificial "skeleton factors" where a few key variables have non-zero values. Use these arrays as starting values. This is slower but one may create the factors in any order one likes.

Array headers

The headers are intended for two different purposes: help-like documentation and bookkeeping. The default headers document what is the meaning of any matrix or array, read or written. Variable headers (input from data files) identify individual measured data sets from each other. If you wish, you may work without any headers. Then you must set all the FIL codes for header input and output to zero in the .INI file. Your data must be pure arrays, without any text lines. This may, however, create confusion: what is the meaning of all the output arrays.

You might wish to include headers for all output. Then the FIL codes for header ouput should be the same as the FIL codes for the output of the corresponding arrays. If you wish to direct an array to another file, then you will have to change two things: the output FIL code of the array, and the output

FIL code of the header. In certain applications, it would be good to have application-oriented headers for output arrays. This is achieved by editing the default headers of arrays in the .INI file.

Timing of header operations. The headers are read and/or written immediately before the time when the array itself might be read and/or written. The header operations may take place even if the array itself is not read and/or not written.
Exception: if the array X is simulated (not input), then header operations for X take place immediately before X is written, i.e. after all input of other arrays. Then also the header operations for S take place after writing X but before writing S.

The programs PMFx store only a maximum of 40 characters of a header. By default the program replaces the default header (as specified in your .INI file) by the header that was read from the data file. It is possible, however, to keep the tail part of the default header intact: if there are the characters ".." (two periods) in the default header, they act as a separator: only the characters before .. are replaced by the corresponding characters from the data file. Thus you may have a mixture of input header and default header in the output.

Headers of input arrays (mainly for array X) are useful for bookkeeping purposes. Be systematic: if there is a header in the data file, then you must read it by specifying the correct code for the header input of X, and vice versa. If the header is read with "A" format (as in PMF2DEF.INI) then (max 40 characters from) one whole row in the file constitutes the header. The only requirement is that there must be at least one non-white character on the row: the program skips all-white rows until it finds a non-white row. If the header is read without format (FMT=0) then the header should be surrounded by 'apostrophes' or by "quotation marks".

If you forget to insert a header into your input file, then (part of) the first row of the array will be used as a header. There will probably be an error message when the array ends too soon (because the first row was lost to the header reading process).

If you do have a header in the file, but forget to read it (FIL=0 for header input) then there will be an error message when the program attempts to read the first row of the array but finds letters which are part of the header. If the header is all numerical characters, there is no error message. One should avoid using such all-numeric headers.

It is good practice to write the header of X to one of the output files even if X itself is normally not written. If there are any problems in reading X then it is useful to write the X array to a temporary file (such as file 38 in PMF2DEF.INI) by setting the FIL code for array output of X correctly (e.g. FIL=38). Then one may inspect which values have been correctly read and which have remained unread (unread values are = -9.999 or some such special value). Also one should check if the data are correctly divided onto rows and columns of X, e.g. are the elements (2,1) (row=2, column=1) and (1,2) correct in X.

One should keep an eye on the Q values of the tasks. They are included in the file PMFx.LOG. But it is also possible to print the Q value as part of the header of the computed G, F, rotmat, or any other result array. This is achieved so that you insert the two characters "Q=" near the end of the default header of the desired array in your .INI file (also remember to turn on writing of that header, otherwise the characters "Q=" have no effect). When PMFx writes the header, it replaces 12 characters of the header after Q= by the current value of Q. In this operation the header is allowed to grow longer than the normal limit of 40 characters.

Arranging arrays in files

The "straight" layout

```
aaaaaaaaaaaaaaa
bbbbbbbbbbbbbbb
ccccccccccccccc
```

In input files the rows of a matrix (or columns, if transposed) must appear on separate lines of the file. One row (column) of a matrix may be split to several lines of the file, if necessary. The diagrams in the boxes illustrate various possible layouts of a data file containing an F matrix of size 3x15 (3 factors). One line in the box represents one line in a file. The elements on the 1st, 2nd, and 3rd rows of the matrix are denoted by a, b, and c, respectively.
The straight layout is good if matrix rows are short enough so that they fit into the lines of the file.

The "folded straight" layout is necessary if the length of lines in the file is limited. This example shows the situation if only 6 matrix elements fit onto one line.
- This layout may be input without any special concern.
- It is output by PMFx if a format is used which specifies 6 elements on a line
- This layout may be difficult to import to spreadsheets or to matlab.

The "folded straight" layout

```
aaaaaa
aaaaaa
aaa
bbbbbb
bbbbbb
bbb
cccccc
cccccc
ccc
```

The transposed layout is often practical for the F matrix, especially when exporting results to spreadsheets or to matlab. Then one has to set the transpose indicator for the matrix as (T)=T (True).

Binary files, e.g. matlab .mat files, cannot be read by PMFx. By using the matlab command "save" with the switch "-ascii" it is possible to export data from matlab in text form so that PMFx may read them.

Layout of factor matrices in files

The trans-posed layout (15 lines in this example)

```
abc
abc
abc
...
...
abc
```

For the factor matrices G and F, the "natural" layout is defined as straight. Thus in the straight layout of G, the factors run along the columns of G. Similarly, in the straight layout of F, the factors run along the rows of F. The same definitions hold for the other matrices which have the dimensions of these factors: Keys, EV, and computed std-dev matrices.

For the factor matrices A, B, and C, there is no natural layout (the basic equation of PMF3 is not in standard matrix notation). We adopt the definition that in the straight layout of the three factor matrices A, B, and C the factors run along the columns of the matrices (similarly as for G). The same shall be true for the Key matrices and for the computed std-dev.

Reading/writing three-way data blocks

page 1			page 2		
a	b	c	m	n	o
d	e	f	p	q	r
g	h	i	s	t	u
j	k	l	v	x	y

(PMF3 only)
A three-way array or block is to be visualized as a book. The first index should generally have the largest dimension (because of efficiency reasons), it is the row index for data points. The second index is the column index. The third dimension should be the smallest. The third index corresponds to the number of page. The pages of the block may be read either so that the second index varies most rapidly (the straight layout, by rows) or so that the first index varies most rapidly (the transposed layout, by columns). In either case, the last (page) index varies most slowly. The example in the box shows a 4 by 3 by 2 data block, the 2 pages are shown side by side. If this data is to be read in straight layout, the values a to y should be in the file in the following order ("/" denotes a new line):

a b c / d e f / g h i / j k l / m n o / p q r / s t u / v x y

If the data is to be read in transposed layout (by columns), then the order should be

a d g j / b e h k / c f i l / m p s v / n q t x / o r u y

Extra line breaks may be inserted if necessary, if the rows/columns are too long to fit on single lines.

Special read layout for X and X_std-dev

(PMF2 only)
In some cases the original data material for X and/or for S=X_std-dev consists of an extra large array of numbers so that one has to pick the desired data values from this large array. An example is shown in the box: the data file consists of pairs of values (X_{ij}, S_{ij}) plus one extra value in the beginning of each line. Then it would be necessary to write an auxiliary program for extracting the desired data.

```
D D D
D X S X S X S X S
D X S X S X S X S
D X S X S X S X S
D X S X S X S X S
D X S X S X S X S
```

The special reading option helps in these picking situations. This is best explained with the example. Assume that the data shown at right are stored in a file so that each line of the box corresponds to one line of the file. The

dimensions of the model are n=5, m=4. Each of the letters D, X, and S denotes one value in the file. We would like to read all the values X into the 5x4 data matrix X, and all the values S into the 5x4 matrix X_std-dev/T. It would be rather laborious to extract these values by using a text editor. In this example, the values D are "something else", a nuisance.

There are four integer parameters near the end of the .ini file for controlling the special reading of matrix X. The first parameter indicates the size of the data block to be processed when reading X. (The value 0 indicates no special reading). In this case, we have 48 data values. The second parameter indicates the position of the first value x_{11} within the data block. In this example, x_{11} is the fifth value in the block. The third parameter indicates the "horizontal" distance from x_{11} to x_{12} to x_{13} ,... = two units. The distance must be the same everywhere in the data block. Similarly, the fourth parameter indicates the "vertical" distance between consecutive rows of X in the data block, (=the distance from x_{11} to x_{21} to x_{31} to x_{41},...) = nine units. The parameters for special reading of X are thus: 48 5 2 9.

Similarly, the parameters for reading X_std-dev/T are 48 6 2 9. But in order to read T, there must be another block of data in the file. After the reading of X, the data block of X is no longer available when starting to read T. If the values of X_std-dev/T are interspersed with values of X (as in this example) there must be a second copy of the same block in the file. This is easily done with a text editor.

The special reading utilizes the user-defined formats as all other readings made by PMF2. However, all the data values in a block are input in one operation, as one very long "row". Thus the division of values onto lines in the block need not correspond to the rows or columns of the matrix to be read. The "Transpose" indicator has no meaning for the special reading of a matrix.

We imagine that the special reading will mostly be used so that FIL codes for the matrices **U** and **V** are =0. This need not be so, however. But there is the restriction that only one set of parametrs is available for controlling the reading of all the matrices **T**,**U**, and **V**. Thus all matrices must have the same layout in their respective blocks.

Normalization options in PMF2

1. no normalization

2. maximum abs. value in each column of G (=in each G factor) equals unity

3. sum of absolute values of elements in each G factor equals unity

4. mean value of absolute values of elements in each G factor equals unity

5. maximum abs. value in each row of F (=in each F factor) equals unity

6. sum of absolute values of elements in each F factor equals unity

7. mean of absolute values of elements in each F factor equals unity

Normalization of factor matrices

In PMF3, the default normalization is fixed: the sum of absolute values of elements in each column of factor matrices **B** and **C** is normalized to unit value. This corresponds to the sixth normalization option for PMF2. There is a new alternative normalization in PMF3: The optional parameter "normfactorsac" commands PMF to normalize A and C factors (instead of B and C). Furthermore, the average absolute values are normalized to unity (instead of sums of absolute values). This option is recommended for environmental studies where B mode corresponds to element concentrations.

Normalization is omitted by PMF3 for such factors where any element in any of the factor matrices A, B, or C is *bound to non-zero value* (xkey<-1), as follows:

> No specific normalization is performed. The norms of such factors remain at those values which happen to be produced during the computation. The user should assume responsibility of producing well-defined norms by binding at least one significant element in exactly two modes (of each such factor) to non-zero values.

PMF2. The seven alternatives for normalization are shown in the sidebar on right. The normalization affects the output of the following five matrices: **G**, **F**, std-dev(**G**), std-dev(**F**), and rotmat.

Only one of these choices may be selected. Immediately before output, PMF2 determines p coefficients which are needed for achieving the desired normalization. Then it divides columns of **G**, and multiplies rows of **F** with these coefficients. Thus the product **GF** does not change when the normalization is performed.

After normalizing (and writing) **G** and **F**, PMF2 performs necessary scalings for the other three matrices so that their new values are correct with respect to the normalized matrices **G** and **F**, and writes these scaled matrices according to the instructions in the .ini file.

Normalizing the factors does not influence factorization computations in any way. The 'chi-2' value Q is not changed.—The command for normalization is near the end of the .ini file of PMF2.

Technical details

Marking true and false

As written by the Fortran system, true and false are represented by "T" and "F". Such tables are visually hard to read. It is suggested that if you have to change these default tables, you write the changed values with two or more letters, e.g. as "TR" and "fa". This makes it easier for you to visually spot changed entries in the tables. The Fortran system is only interested in the first letter which must be one of "f F t T".

Unwanted characters in data files

Some measurement systems produce such data arrays where there are extra characters (e.g. line numbers) in the beginning of each line. These characters may be bypassed by using a format for reading the data, (FMT>0) and by inserting a suitable nX code in the format. This requires help from somebody who knows how to use the format system of Fortran90 (essentially the same as in Fortran77).

Avoiding too large files

Often it is good to accumulate the important results in a file which grows constantly. But if all results are directed to such a file it grows fast and fills the disk. It is difficult to find editors for handling a file once it has grown to half a megabyte, say. In order to limit the growth one might only direct the essential results (e.g. the F factor, "rotmat" matrix) to an accumulating file (status = UNKNOWN). Other results (e.g. error estimates and EV values for the G factor) might go to a temporary file, opened with "status=REPLACE". Such a file will only contain the results of the latest run.—The growth of the file PMFx.LOG is slower if "monitor" is set to M=5, say, in the .INI file.

Seldomly needed details

Systematic naming of files

Assume that you are analysing air pollution data from several cities, and you have the file AIR.INI for controlling the analysis. In the .INI file, you could specify the file names as X$.dat, G$.RES, and F$.RES, and specify reading of X from X$.dat, and writing G and F to G$.RES and F$.RES. Then you would start the analysis of Rome data by the command "PMF2 AIR ROME", meaning that the character $ shall be replaced by the letters "ROME" wherever it occurs in file names. The program would read data from file named XROME.DAT and write results to GROME.RES and FROME.RES. Similarly, Paris data would be run by the command "PMF2 AIR PARIS", reading from the file XPARIS.DAT. - This feature is not available on some processors.

Controlling pseudorandom numbers

The parameter "SEED" initiates the pseudorandom generator. Changing SEED enables one to study the convergence of PMFx from different pseudorandom initial values of factors **G**, **F** or **A**, **B**, **C**. In "usual" runs the value of SEED need not be considered, the default value SEED=1 is adequate. — Different sets of pseudorandom initial values may also be obtained by running with "*Repeats*">1, see above.

If SEED=0 then the pseudorandom generator of the host Fortran 90 system is used "as such". On different platforms (PC, DEC, ...) there may be different sequences and there is only one sequence available on any platform.

If SEED=k>0 (k is an integer!) then a simple portable random-number generator is used, it is the same on all platforms. Different values of k select different positions of the sequence for starting (the integers k=1, 2, ... do *not* select first, second, etc. element for starting, but "randomly chosen" locations within the full sequence).

The pararameter "Initially skipped" indicates how many first elements of the pseudorandom sequence are to be skipped when PMFx is started. The skipping is performed *after* choosing the starting position according to SEED. (This parameter is not useful and may be dropped from future versions.)

Advanced options for PMF2 and PMF3

The key matrices Gkey,..., Ckey

In PMF2, the values **G**key$_{ij}$ and **F**key$_{ij}$ control which one of the first three alternatives (shown on the right) applies for each factor element g_{ij} and f_{ij}. The strength of the bond is quite weak if key=3 and strong if key=10, say. The matrices **G**key and **F**key have the same shapes as **G** and **F**. Currently it is not possible to bind the factors to non-zero values in the program PMF2.

key=0	element is constrained to non-negative values only
key=1	element is free to take any values, >0, =0, or <0
key>1	element is bound to zero
key<-1 (PMF3)	element is bound to a non-zero value (given as "initial value")

In PMF3, the matrices **A**key, **B**key, and **C**key have a similar control function but there is also the possibility of binding selected factor elements to non-zero target values. The more negative key values effect a stronger binding, the value -10 might be a good first try. The target values are to be specified as initial values for the factor elements in question. Using the option "key<-1" is somewhat complicated. See the file readme.txt.

PMFx generates synthetic data arrays

It is permissible to set the FIL code for input of the data array **X** equal to zero. Then the normal processing of PMFx is changed in the following ways:

First PMFx generates the std-dev array **S** as specified by the input FIL codes for X_std-dev (**T**,**U**, and **V**) (after reading those arrays (**T**,**U**,**V**) which have FIL code >1) and by the codes C1, C2, and C3. If the corresponding output FIL codes for any of **T**,**U**, or **V** are non-zero then those arrays are also output ("echoed"). The computed array **S** is output later, after output of **X**. (In normal runs **S** is output at the end of the computation).

Next PMFx reads the initial values for factor matrices **A**, **B**, and **C** (and their key matrices if specified in the .ini file). If echo output for these matrices has been specified, it is performed, too. (If PMF2, then also rotcom is input). Then PMFx computes the theoretical data array **X** which exactly corresponds to these initial factor values.

Then PMFx computes the "simulated" error-containing data array **X** by adding to the exact **X** the computed pseudorandom error values according to the computed array **S**, and outputs the simulated **X** according to the **X** output FIL code. Then PMFx outputs the array **S**, according to the FIL code and other specifications given for **S** near the end of the table of array I/O specifications.

After this, the processing continues normally, the simulated **X** is fitted by the factors in the usual way, but the final computed std-dev array **S** is not output.

Synthetic data arrays are useful for systematic testing of the program. Then one knows the correct solution and the corresponding Q value from the generating run.

Specifying standard deviations for the array X. Significance of the Q value. Outliers.

The array of standard deviations

The wording and the equations in this part are written for PMF2. Similar equations are also valid for PMF3. In theoretical descriptions of PMF, the array of standard deviations is often denoted by **S**. In connection with the programs, we also denote it by X_std-dev.

The significance of the array **S** (standard deviations for elements of the array **X**) has been discussed in more detail in the references. This array is the means for communicating problem-specific *a priori* information to PMFx.Generally one should attempt to specify the std-dev values so that they indicate the agreement which is expected between the (bilinear or trilinear) model and the data array. A few

unrealistically small std-dev values (e.g. smaller than the data values by a factor of 10000) may lead to useless results or even to breakdown of the computation because of apparent matrix singularity. Often it might be simple to specify the std-dev values as a fixed percentage of data values. This must be avoided, however, if there may be (near-)zero values in your data. Sometimes the std-dev values can only be computed when there exists a (preliminary) fit to the data. When the fit converges towards a good solution, then also the computed std-dev values converge towards a good model. An example is when the data obey Poisson distribution and the counts are low: the measured data as such don't give satisfactory estimates for **S**. Another example is environmental data where outliers are common: std-dev values are best based on the larger one of the measured and fitted values, thus lessening the effect of both kinds of outliers: contamination and loss.

Sometimes one has hardly any idea of std-dev values for **X**. How is one to use PMFx then? A simple rule of thumb might be applied, such as: std-dev(x_{ij}) is = 5% of x_{ij} plus two units of the least significant digit reported for x_{ij}. (In cases where all the rows and all the columns of **X** are of same magnitude, this rule may be implemented very simply by using the ad-hoc equation for X_std-dev in PMFx.) If x_{ij} = 123, then this rule gives 6.15 + 2 = 8.15. If x_{ij} = 0.006, we get std-dev(0.006) = 0.0023. This is simple but it is better than nothing (or better than assuming that all values are of the same absolute accuracy, as is inherent in the classical PCA). If rows and columns are of different order of magnitude then the simplest ad-hoc equation is not useful; one must model the std-dev values by using the three coefficient arrays **T**,**U**, and **V**.

Specifying the X_std-dev

Computation of **S** is based on three codes C1, C2, and C3, on the code "Errormodel" (EM), and on three arrays **T**, **U**, and **V**. **S** is always computed according to equations (7), (8), (9), (10) or similar equations. It is possible to read the array **T** from a file (by having its FIL code >1) but in simple cases (e.g. all data are of same accuracy) one may leave its FIL code=0. Then the program uses the code C1 instead of the array values t_{ij}. Similarly one may read the array **V**, containing relative errors of data points.. But if the relative error of all points is the same, then one may have the FIL code for **V** =0 and insert the relative error in the code C3. This means that the value C3 is used in place of all array values v_{ij}. Usually both the FIL code for **U** and the value C2 are =0 but in rare cases they are used similarly as the other two error terms. If any of the arrays **T**, **U**, or **V** is input from a file then the value(s) of the corresponding code(s) C1, C2, or C3 has no meaning. The values C1 and t_{ij} are expressed in same units as the data values x_{ij}. The coefficients C2 and C3 and the arrays **U** and **V** are dimensionless.

In the simplest case the std-dev array is ready in a file and should be input as such. Then it is read as the array **T**. The FIL codes should be =0 for **U** and **V**, and one should set C2=0, C3=0, EM=-12.

In repeated runs the array **S** is always recomputed when another analysis is started, irrespective of the repeat-codes (R) of arrays **T**, **U**, and **V**. These (R) only controls re-reading of the three arrays.

The code EM controls how the **S** is computed. There are two possibilities: either the array **S**=X_std-dev is computed once before the iterative computation ("iterative least squares fit") is started (EM=-12), or else approximations for **S** are computed iteratively during the fit. In those cases where **S** is based on the fitted array **Y** (for definition of **Y**, see Introduction), the values max(y_{ij},x_{ij}) are utilized instead of the fitted values y_{ij} during the first few iteration steps when the fit is still not well defined. This is true for the EM values -10, -13, and -14.

A complete list of the alternatives for computing **S**=X_std-dev follows. The two first alternatives are the simplest. Preferably one should first learn to use the program by using these simple techniques; sometimes one may simply assemble the array with a text editor so that for each column (row) there is only one value which is repeated for the whole column (row) (see the remark at the end of next section). In other cases, one has to compute the array in a preprocessing program.

EM=-12, C1=t, C2=0,C3=0, FIL(T)=0, FIL(U)=0, FIL(V)=0. The value t of C1 is used as the std-dev value for all elements of the array **X**. This value is not changed during the iteration.

EM=-12, C1=0,C2=0,C3=0, FIL(T)>1, FIL(U)=0, FIL(V)=0. The std-dev array is read from a file (as array **T**) and used as such, not changed during the iteration. Repeated reading of std-dev is controlled by the repeat-code (R) of X_std-dev /**T**.

EM=-10. *Lognormal distributions*. It is assumed that each data value x_{ij} comes from a lognormal distribution with geometric mean equal to the fitted value y_{ij} and log(geometric-standard-deviation) = v_{ij}. It is further assumed that there is "measurement error" having std-dev=t_{ij} in each measured value x_{ij}. The factors **G** and **F** (or **A**,**B**,**C**) are determined so that **Y**=**GF** (or **Y**=**ABC**) maximizes the likelihood of **X**, given the arrays **T** and **V**. This implies that PMFx computes s_{ij} as $s_{ij} = \sqrt{t_{ij}^2 + cv_{ij}^2 \left|y_{ij}\right| (\left|y_{ij}\right| + \left|x_{ij}\right|)}$, or $s_{ijk} = \sqrt{t_{ijk}^2 + cv_{ijk}^2 \left|y_{ijk}\right| (\left|y_{ijk}\right| + \left|x_{ijk}\right|)}$ if PMF3. (c =0.5). The computed **G** and **F** (or **A**,**B**,**C**) are Maximum Likelihood (ML) estimates of the true unknown factors.— If all the log(GSD) values v_{ij} are equal, then the coefficient C3 may be used instead of the array **V**. Similarly C1 may be used instead of the array **T** if all the absolute errors have the same std-dev. The array **T** is intended for representing typical measurement error, which may cause zero or negative values to appear although the genuine lognormal distribution is always strictly positive. If there is no such "lab error" in the data, then **T** is not needed and one sets C1=0. — The coefficient C2 and the array **U** are not used when EM=-10.— In log-normal modelling the expectation value of the Q as computed by PMFx increases from the usual degrees-of-freedom value (approximately the number of elements in the data array). The coefficient of increase is written by PMFx to the .log file.

EM=-11. *Correspondence analysis.* It is assumed that each data value x_{ij} comes from the Poisson distribution with a parameter μ_{ij}. Factors are determined so that they represent the array μ: μ=**GF** (PMF2) or μ=**ABC** (PMF3). During the iteration, PMF2 computes s_{ij} as $s_{ij} = \sqrt{\max(\left|\mu_{ij}\right|, 0.1)}$, (PMF3: $s_{ijk} = \sqrt{\max(\left|\mu_{ijk}\right|, 0.1)}$) resulting in maximum likelihood (ML) estimation for the array μ. The "max" protects against ever having s_{ij} =0. FIL codes for **T**,**U**,**V** should be =0, and C1=0, C2=0, and C3=0.

EM=-12 The values s_{ij} (s_{ijk}) are computed according to equation (7), (8) or (9) by using the coefficients C1,C2,C3 and/or arrays **T**,**U**,**V** and the data array **X**. This is done before the iteration is started. The std-dev values are not changed during the iteration. **This is the default.**

EM=-13. The values s_{ij} (s_{ijk}) are computed by using the coefficients C1,C2,C3 and/or arrays **T**,**U**,**V** and the array **Y** of fitted values. This is done iteratively during computations.

EM=-14. The values s_{ij} (s_{ijk}) are computed according to equation (10) by using the coefficients C1,C2,C3 and/or arrays **T**,**U**,**V**, the array **X**, and the array **Y** of fitted values. For each element (ij), the computation is based on the larger of the values x_{ij} and y_{ij}. This is done repeatedly during the iterations.

The option EM=-14 is recommended for general-purpose environmental work. The reason for recommending this alternative instead of the "standard" (EM=-12) is as follows: The (EM=-12) computes the std-dev essentially as a percentage of the observed value. For a large value (a possible contamination-type outlier) a large std-dev is obtained, thus a contaminated value never gets an unduly large weight. But for a small observed value the standard technique computes a small std-dev value; if the small value is in fact a loss-type outlier then it gets a too large weight because the std-dev is based on the near-zero erroneous value. In such a case the fitted y_{ij} is significantly larger than x_{ij}. The alternative EM=-14 avoids generating too small std-dev values by taking the larger of y_{ij} and x_{ij} as the basis for std-dev. — For good (non-outlier) observed values the observed and fitted values agree and there is little difference between selecting one or the other. Thus the alternative EM=-14 should never be a bad choice, except that the option EM=-12 runs faster than the other ones. —The alternative EM=-10 (lognormal) is also a good choice for environmental data. However, it has been found out that the options EM=-10 and EM=-14 are practically similar unless the size of residuals is very large in comparison to the size of data. Thus EM=-10 is only needed for such data where the parameter C3 is larger than 0.3, say.

Ad-hoc computation of std-dev values.

If FIL=0 for all arrays **T**,**U**,**V**, and if EM=-12, then all values s_{ij} are computed with a simple ad-hoc formula, based on the corresponding data values x_{ij}. The equation is

$$s_{ij} = \text{std–dev}(x_{ij}) = C1 + C2\sqrt{\left|x_{ij}\right|} + C3\left|x_{ij}\right| \qquad (7)$$

The values C1, C2, and C3 are input as part of the file PMFx.INI. This alternative is practical if all rows and columns (and planes) of **X** represent the same physical quantity having the same error characteristics. Then it is not necessary to have individual treatment of rows and/or columns. Then C1 should be chosen so that small values of **X** get a good std-dev value (typically = the detection limit in environmental work). Similarly, C3 should be chosen so that the relative uncertainty of large values is reasonable, typically C3 is between 0.01 and 0.1. The value of C1 should be expressed in same units as the data values x_{ij}. The value of C2 is usually zero, except in Poisson-like situations.

In environmental work, different columns of the array **X** typically contain concentrations of different elements, having different error characteristics. Then the simple ad-hoc formula (above) is not adequate because different columns would need different values for the value of C1 (detection limit). Then one has to use the arrays **T,U,V**. In PMF2, each of **T,U,V** is a matrix of the same size as the observed matrix **X**. Analogously, in PMF3, each of **T**,**U**,**V** is a three-dimensional block of the same dimensions as the block of observed data **X**. The equation for **S** in PMF2 is

$$s_{ij} = t_{ij} + u_{ij}\sqrt{|x_{ij}|} + v_{ij}|x_{ij}| . \qquad (8)$$

For PMF3, the corresponding equation is

$$s_{ijk} = t_{ijk} + u_{ijk}\sqrt{|x_{ijk}|} + v_{ijk}|x_{ijk}| . \qquad (9)$$

These two equations are valid for the code EM=-12. For the choice EM=-13, the fitted values Y replace X. And for the last choice EM=-14, equation (8) gets the form

$$s_{ij} = t_{ij} + u_{ij}\sqrt{\max(|x_{ij}|,|y_{ij}|)} + v_{ij}\max(|x_{ij}|,|y_{ij}|) , \qquad (10)$$

and similarly for equation (9). Usually the square-root term is unnecessary: set FIL(U)=0, C2=0.

In typical environmental work, each column of **X** contains similar values (concentrations for a certain compound or element). Then each of the columns of **T** and **V** is simply a repetition of a fixed value. It is not necessary to write such arrays in full form. It is practical to write them in the input file in the transposed layout. Then each column may be represented by a repeat code, e.g. the notation 1000*0.02 represents the value 0.02 for one column of **T** or **V** (of any length not larger than 1000).

Robustness

We recommend that you generally run PMFx in the robust mode, at least in environmental research. This guarantees that occasional outliers do not ruin your results. Only in exceptional cases it is possible to know that there are **no** non-representative data (no 'outliers') in the array. In robust mode it is not quite so essential to specify increased std-dev for unreliable values but if you suspect some points, then you should increase their std-dev even in robust mode. It is important to note that using the robust mode does not protect against the harmful effects caused by using extremely small std-dev values for some data points.

The combination of robust mode and systematic gross under-estimation of std-dev for many points makes the program slow! Try to provide realistic std-dev even in robust mode!

Missing values

Missing values occur in real data. In PMF they are best handled so that the user specifies large std-dev for them. However, in order to avoid non-physical factor values one should avoid too large std-dev $\mathbf{S}_{ij}$ which satisfy the inequalities $s_{ij} >> x_{ik}$ and $s_{ij} >> x_{hj}$ for all k and all h.

By using the optional parameter "*missingneg* r" (r is a decimal value) one commands PMF to decrease the significance of all negative entries of the array **X.** This is realized so that PMFx increases internally their std-dev values by the factor r and uses the value zero as the data value. With a suitable r=10...r=100 this may cause that these negative x_{ik} values have a negligible effect on the factors. This option cannot be used if there are "good" negative values in **X**. This technique should be regarded as a "trick", it should only be applied with caution.

The optional parameter "*BDLneg* r1 r2" combines missing value handling (as presented above) with Below-Detection-Value handling. If a data value x_{ik} is more negative than r1, it is treated as missing

and its std-dev is increased internally by the factor r2. The value zero is used in the least squares fit, instead of x_{ik}. If, however, $r1 < x_{ik} < 0$, then it is assumed that $|x_{ik}|$ is the Detection Limit (DL) for a Below- Detection-Limit measurement. The corresponding std-dev value s_{ik} is computed normally but dynamic weighting is applied in order to achieve correct BDL handling.

Sometimes missing values are represented in a spreadsheet table by empty cells. Such a table should not be written in text form, then the information about empty cells is lost. However, the table may be written in comma-separated (.csv) form. Then an empty cell is indicated by two consecutive commas. Whenever PMFx encounters two consecutive commas while reading with the usual format code =0, it inserts the special value -999.9 in the empty slot in the data array. By suitable use of the commands "*missingneg* r" or "*BDLneg* r1 r2", this special value may be interpreted as a missing value indicator.

Judging the values obtained for Q

If there are no outliers, and if the error modelling is correct, then the final Q or chi2 value should be approximately equal to the number of entries in your data array **X**. This is also true for the Poisson model but **not** for the lognormal model.

If there are outliers then the increase of Q may be estimated as follows: Denote by e_{ij} the residual and by s_{ij} the std-dev specified by the user. Let α be the specified outlier threshold distance. Define d so that

$$e_{ij} = s_{ij} \alpha d,$$

i.e. the residual is d times the outlier threshold distance. Let (ij) be an outlier point so that d>1.0. Such an outlier increases the customary non-robust Q by the amount

$$(e_{ij} / s_{ij})^2 = \alpha^2 d^2 .$$

The robustized or "downweighted" Q is increased only by the amount $\alpha^2 d$. This corresponds to the apparent increase of s_{ij} by the factor $\sqrt{d}$. This increase of s_{ij} is performed by PMFx dynamically during the iteration. The heuristic behind this downweighting is that the downweighted point (ij) exerts the same influence on the fit as an observed value with d=1, i.e. an observed value at the outlier threshold distance from the fitted value.

In the presence of outliers (even in the robust mode) it may be difficult to know if the observed value of Q is "normal" or too large. It may be more helpful to investigate the distribution of scaled residuals (e_{ij} / s_{ij}). If the vast majority of these values obey $|e_{ij} / s_{ij}| < 2.0$, then the increase of Q value is probably due to the outliers.

Possible reasons for an excessively large Q (chi-2) value. How to correct.

1. the original X_std-dev (as specified by you) are too small. Specify larger values.

2. (if lognormal model) the geometric std-dev are too small and/or you need to add an additional normally distributed error term (C1 or array **T**).

3. more factors are needed, i.e. the original value of p is too small. Increase the number of factors.

4. the data do not obey a bi(tri-)linear model, i.e. PMF2(PMF3) is not a suitable model.

5. (in non-robust mode) there are outliers. Switch to the robust mode!

6. the iteration did not converge, or converged to a local minimum. Rerun with increased iteration count limits and/or from several different starting values.

If you observe a large Q or chi2, you should try to find a cure from the list in the sidebar.

There are three alternatives for writing the residual array to the file. In the third alternative the residuals are scaled by the apparently increased s_{ij} values, thus the third array consists of the values

$$e_{ij} / (s_{ij} \text{ sqrt}(d)) \quad (d \geq 1.0)$$

(If the analysis was run in non-robust mode, then the third residual array is the same as the second.) Usually it is sufficient to write one alternative only, the scaled residuals .

The problem of rotations; error estimates of results

Using FPEAK to control rotations in PMF2

There is rotational ambiguity in the results of all two-way factor analytic programs, including PMF2. The question of rotations is also discussed in the tutorial part of this User's Guide. One means for choosing between different possible solutions is "FPEAK", a "peaking parameter". By setting a positive value to FPEAK (0.1...2.0, say) one forces the routine to search for such solutions where there are many (near-) zero values among the F factor values, and also many large (i.e. as large as allowed by the data) values, but few values of intermediate size. In some cases one could say that there are "peaks" on the F side. This is similar to using the varimax tecnique on F. Technically speaking, a positive FPEAK forces the routine to try to subtract the F factors from each other (meaning that the G factors are added to each other).

Correspondingly, FPEAK < 0.0 generates "peaks" on the G side. This would correspond to using varimax on G, i.e. using varimax after solving the transposed problem with usual PCA. A negative FPEAK causes factors to be subtracted from each other on the G side. In the Gauss-exponential example, this tends to produce clean peaks. Try the example with FPEAK = -0.1 or -0.5, say! Very large (negative or positive) values of FPEAK may not lead to the desired solution at all, especially if "lims" is also very small. This is probably caused by some factor values being pushed so near to zero that there is no room for necessary rotations without making some factor values negative.

A large value (positive or negative) of FPEAK leads to worsening of the fit. This is caused by the changing of factor shapes: additional rotations are then only possible if the shapes of the factors change (in order to avoid violating the non-negativity constraints) and this makes the fit worse. This may be observed in the GE example: look carefully at peak shapes, especially at the tails of the peaks. With FPEAK=-1.0, say, the peak tails are not as nicely rounded as they should be but they fall abruptly to zero.

Rotational matrices, rotational freedom, and error estimates of results

PMF3. Estimation of std-dev values for factors is based on a global Least Squares fit where all three factor matrices **A**, **B**, and **C** are determined simultaneously (this differs from PMF2!). Thus the obtained "error estimates" or std-dev values A_std-dev, B_std-dev, and C_std-dev, reflect *both* individual random uncertainty *and* uncertainty due to global rotation-like transformations of the factors.

If the factor model is *identifiable* (obeying the so-called Kruskal conditions) then the solution of PMF3 has no rotational freedom. Then the error estimates of the factors A, B, and C only reflect the individual random uncertainties of the factor elements (similarly as in PMF2).

In the opposite *non-identifiable* case, there is similar rotational freedom in PMF3 as is typical for two-dimensional factor analysis. Different possibilities for influencing the rotation of the result are discussed in Part 1 of the User's Guide. The most important technique is to use the Akey, Bkey, and/or Ckey matrices to constrain some factor elements to zero or non-zero values. In the *non-identifiable* case, the error estimates for A, B, and C contain the rotational ambiguity (unless the ambiguity has been removed by using the Keys). The presence of rotational freedom is indicated by large values in the std-dev matrices for A, B, and/or C.

Borderline cases are likely to occur in practice. This means situations where one of the factor matrices **A** or **B** is almost rank-deficient, i.e. one of the singular values is almost down to the noise level, i.e. one factor may almost be expressed as a linear combination of other factors. This will probably be indicated by somewhat increased values in the error matrices. More experience is needed!

PMF2. The uncertainty of the factors G and F is conceptually attributed to two reasons:

- individual randomness of the values, caused by individual errors (noise) in the observed data. This uncertainty is estimated by G_std-dev and F_std-dev.
- rotational uncertainty, often limited by the shape of factors and the non-negativity constraints. The rotational uncertainty is estimated by the matrix "rotmat".

The rotational state of the result by PMF2 may be controlled by a matrix called "rotcom", it is "a matrix of rotation commands". The matrix rotcom should be all=0 if/when special rotations are not needed. Otherwise rotcom is a pxp matrix (often mostly zeroes) commanding additions and subtractions of factors according to the table where $g_{\#j}$ denotes the j^{th} column of G.

> The matrix rotcom has not proved to be a practical tool, and it has probably never been used successfully in real-life PMF2 applications. It will probably be deleted from the program PMF2, but it is still explained here as it is part of the current program. Using it is not recommended any more. Instead we recommend using FPEAK and/or pulling factor elements to zero or non-zero values.

The scale of rotcom values corresponds to the scale of FPEAK values, the same value should have an effect of comparable size in rotcom and in FPEAK. Good first trial values are between 0.1 and 1. Note, however, that one should normally use either FPEAK or rotcom, but not both.

Using FPEAK (or rotcom) is not equivalent to making rotations *a posteriori*, after running PMF2. When FPEAK(rotcom) influences PMF2, it changes the rotational state the solution, but it may also change the solution itself (causing an increase of the Q value). Thus FPEAK(rotcom) may succeed in producing a rotation in a case where rotations appear impossible if attempted *after* running PMF2.

Together with G_std-dev and F_std-dev (see below) "rotmat" is the basis for estimating the uniqueness and accuracy of the computed factor values. "rotmat" indicates the rotational uncertainty. The values rotmat(i,j) and rotcom(i,j) are related to the same rotation. "rotmat" is a pxp matrix of standard deviations of rotational coefficients: rotmat(i,j) (row=i, col=j) indicates the uncertainty or freedom of rotation where simultaneously

$rotcom_{ij}>0$	$g_{\#j}$ is added to $g_{\#i}$ $f_{i\#}$ is subtracted from $f_{j\#}$
$rotcom_{ij}<0$	$g_{\#j}$ is subtracted from $g_{\#i}$ $f_{i\#}$ is added to $f_{j\#}$

> $g_{\#j}$ is added/subtracted to $G´g_{\#i}$ and
> $f_{i\#}$ is subtracted/added to $f_{i\#}$.

The final (third) value of "lims" influences the values of rotmat directly and indirectly. With a *large* final "lims" all elements of rotmat are of medium size, because limit repulsion is soft but extends to all components. All rotations are then in effect determined by limit repulsions, but because no elements are near to the boundaries, the repulsions are "soft".

If the final "lims" is *small* the distinction of locked and free rotations becomes apparent. Some factor elements tend to approach near to the limits, and they become "stiff" with respect to rotations (there is no distinction between rotations towards the limits and away from the limits). Then the corresponding element of rotmat becomes small, indicating a locked rotation. On the other hand, some factors stay away from the limits, and the rotmat elements for their rotations increase, indicating free rotations and non-unique factor values.

The rotmat values should only be interpreted qualitatively. It would be tempting to assume that a value of 0.2 in rotmat would mean that 20 % of one factor can be added/subtracted from another. This is not true, unfortunately. If one is solving many similar problems, then one might get a feeling about the quantitative interpretation, too. As an example, in one case the value 0.2 occurred in rotmat when it was possible to rotate 2% of one factor into another.—*Utilization of rotmat values is only possible if correct std-dev values have been specified for measured data.*

If all elements in the i^{th} row of rotmat are "small", then one knows that the i^{th} G factor is uniquely determined without "any" rotational uncertainty. Similarly, a column of small elements in rotmat indicates a uniquely determined F factor.

Forced rotations also affect the rotmat values: if FPEAK(rotcom) is used to press some factors tightly against the wall, then there remains less freedom of further rotation, and the corresponding rotmat values decrease. See the example GE2.INI.

Forcing rotations with rotcom may be surprisingly tricky: when one sets up a certain "pull" for one factor, it may happen that other rotations also happen at the same time, if they are free. It is as if cutting a slippery piece of food with a dull knife, the piece rotates in all possible ways in order to avoid being mutilated. It is also possible that large values in rotcom change the order of factors in the factor matrices. The risk of this is minimized if one uses a "good" starting point for a factorization where rotcom has large values; a good starting point tends to freeze the identity of factors.

Quantitative estimation of rotational freedom has not yet been attempted in a systematic manner. It is not possible by simply looking at rotmat elements. It should be possible by systematically trying different rotcom elements and by observing how much rotation is possible without increasing the Q too much. The author would be interested in joint research along these lines!

The result matrices G_std-dev and F_std-dev

G_std-dev gives the uncertainty of each element of factor G under the assumption that F is kept fixed. Vice versa, F_std-dev is the uncertainty of F when G is kept fixed. *These values are invalid if correct values have not been specified for X_std-dev.*

These matrices contain the standard deviations of G and F as determined from such Alternating Least Squares ("Alternating Regression") fits where each row of G is determined while keeping F constant, and each column of F determined while keeping G constant. Mainly, the std-dev values are determined by values in X and X_std-dev. But the presence of a penalty function also influences these uncertainties, especially for those factor elements which approach zero with decreasing lims. Currently we think that these std-dev estimates are quantitative by their nature, but this remains to be proved.

Miscellaneous details

Locations of error messages

During their operation, PMF programs write log information on the screen and also to the end of the file PMFx.LOG. They also write error messages to both places. However, the error messages created by the Fortran system (e.g. complaints about data/format errors) are only written to the screen! Every now and then, one should delete the file PMFx.LOG, otherwise it would occupy too much of disk space.

Explained Variation EV

(PMF2 only)
The original definition of EV came from the papers of Paatero and Tapper, 1994, and Juntto and Paatero, 1994. There are several problems with this definition, especially if negative factor values are allowed to occur. The current recommended definition for EV was introduced for PMF2 v.4.1.

The quantity EV is dimensionless, it summarizes how important each factor element is in explaining one row or column of the observed matrix. The values of EV range from 0.0 to 1.0, from no explaining to complete explanation. This value is not quadratic: if measured data is explained so that the residuals are typically 10 % of data, then the corresponding EV value for the explanation is 0.9. The customary "explained variance" and the original Paatero-Tapper-Juntto definition of EV are quadratic quantities, they would indicate 0.99 in the same situation!

The new definition considers that all of X is explained jointly by the p factors and by the residual, as if the residual would be an extra (p+1st) factor. Taken together, these p+1 "factors" by definition explain 100% of X. The equations for the variation of the factor G are

$$\mathrm{EV}(G)_{ik} = \frac{\sum_{j=1}^{m} \left| g_{ik} f_{kj} \right| / s_{ij}}{\sum_{j=1}^{m} \left(\sum_{h=1}^{p} \left| g_{ih} f_{hj} \right| + \left| e_{ij} \right| \right) / s_{ij}} \qquad \text{(for k=1,...,p)} \qquad (11)$$

$$\mathrm{EV}(G)_{ik} = \frac{\sum_{j=1}^{m} \left| e_{ij} \right| / s_{ij}}{\sum_{j=1}^{m} \left(\sum_{h=1}^{p} \left| g_{ih} f_{hj} \right| + \left| e_{ij} \right| \right) / s_{ij}} \qquad \text{(for k=p+1)} \qquad (12)$$

Equation (11) defines how much each element g_{ik} (k=1,...,p) of the factor G explains of the ith row of the matrix X. Equation(12) defines similarly how much the residual values e_{ij} do explain of the ith row. By definition, the sum of these p+1 values equals unity.

The contributions of different points x_{ij} (j=1,...m) are scaled by the values s_{ij}=X_std-dev$_{ij}$. These are the original user-defined standard deviations for x_{ij}, not the adjusted values generated during a robust fit. If there is a huge outlier on the ith row it may slash the EV-values for that row, even if it does not influence the fit significantly in the robust mode. If one wishes to calculate such EV-values without the influence of the large outlier then one must manually increase the std-dev value for that outlier.

The values EV(G)$_{ik}$ are represented as a matrix of dimensions n by p+1, i.e. of the same shape as G but with one extra column added for representing the "explanation by residual" (in fact telling what part of X is unexplained by the p factors).

Similar equations are also defined for the F side. The matrix EV(F) is a (p+1) by m matrix, where the extra row (p+1) corresponds to the residual.

The covariance matrix of A, B, and C.

Optionally, the program PMF3 computes the full covariance matrix of all the factor elements. This is a very large square matrix. If the dimensions of A,B,C are (m×p),(n×p),(o×p) then both dimensions of this matrix are = p×(m+n+o). The order of elements on the rows and columns of this matrix is:
A: 11,12,...1p, 21,22,...2p, ... m1,m2,...mp
B: 11,12,...1p, 21,22,...2p, ... n1,n2,...np
C: 11,12,...1p, 21,22,...2p, ... o1,o2,...op
This matrix may be useful in determining significances of trends or other collective features visible in factors. More experience is needed, however. -- If only the significance of individual factor elements is to be investigated, then it is sufficient to compute the std-dev matrices of the individual factor matrices. These std-dev matrices are in fact the square roots of the diagonal of the covariance matrix. However, computing them is faster because the whole of the covariance matrix is not needed.

Observing the process of iteration

The algorithms output some information on each step of the process. The first thing to note are the decreasing "Q" or "chi2" values. In PMF2 the leftmost chi2 values result from fitting the G side, the rightmost ones from fitting the F side. Usually the chi2 values tend to decrease. There are, however, several reasons for apparently increasing values:

1. The algorithms minimize the sum of the chi2 and the penalty. Sometimes the penalty is decreasing and chi2 increasing while their sum is decreasing. This is perfectly normal.
2. Iterative reweighting is in use in robust mode and also if the std-dev values are computed dynamically during the fit (the code EM is -10, -11, -13, or -14). In these cases the minimization task is different in each step, and the sum of chi2 and penalty may increase. By running PMF2 with Monitor=-2 one sees the chi2 and penalty also *before* each step, in addition to the values computed *after* the step. This allows one to check if each step achieves a decrease.
3. Forcing the process by FPEAK, rotcom, or by the Key arrays causes a worse fit and thus a larger chi2. One should observe such an increase of chi2 and avoid such forcing values which cause too high an increase. Presently it is not yet known with certainty how much is "too much of increase". However, an increase which is less than 10 units is almost certainly "not too much".

The algorithms also output "Flags". These are a trace of what happens in computations. In error situations it may be helpful to note what flags have appeared before the error. The flag G indicates that the step was about to violate one or several of the constraints $G_{ij}>0$ and it was necessary to take special precautions. Similarly for the flags F, A, B, and C. The flag characters "0" indicate that the rotational or norm-balancing substeps only made a small change. More zeros mean a smaller change. In PMF3, S indicates a simplified faster step. —Other flags do occur but they are not documented.—It is normal that the first few main steps are quite complicated, with many different flags. This indicates that the optimization is repeatedly hitting against the non-negativity boundaries or against the non-linearity of the problem. Gradually the penalty functions gets better and better approximated until a smooth flow of the process sets in, with only an occasional constraint violation every now and then. If the process does not reach this smooth phase in some ten or fifteen steps, it may indicate that the first penalty weight factor (the first "lims" value) is much too small. Run again with a larger first lims value. However, sometimes it may be good to try with a smaller first penalty factor.

The penalty function values are output on each step. When the algorithm nears convergence, both the chi2 and the penalty values stop changing.

In PMF3, "RelSt" values illustrate the "raw" Relative Step as computed for factor elements. The leftmost value is the largest relative decrease of any positively constrained factor element. The value 1.0 would indicate that some factor element attempts to decrease down to zero. Such a large step is truncated, of course. The rightmost value is the mean relative change of unconstrained factor elements. These values may occasionally be > 1.0 without causing problems. The value "Trim" shows how much the originally computed "raw" step is shortened (or lengthened) when the step is actually made. The reason for shortening the step is either an attempted violation of constraints, or a failure to (optimally)

decrease the sum of chi2 and penalty. The latter is caused by the non-linear nature of the problem and becomes apparent if long steps are tried; it is indicated by the flag ">".

When a peaking rotation is requested, the PMF2 algorithm sometimes seems to proceed "by stages": the chi2 changes, then stays approximately constant during several steps, again changes, and so on.— Current experience indicates that if there is practically no change of chi2 during 10 or 20 steps, then one may be confident that the true convergence has been reached. Note, however, that there are local optima for some problems. In order to ascertain that the global minimum has been found, one has to rerun the analysis ten or twenty times, starting from different pseudorandom values.

The algorithm used by PMF3 may sometimes show the message "matrix is singular" followed by a code number. This is not an error. These messages need not be reported to the author. However, if there are tens of them so that the program is unable to proceed, then one should report.

Using PMF3 for solving 2-way problems

The program PMF3 may also be used for solving 2-way problems. Then one has to specify the third dimension equal to 1. Examples of solving the Gauss-Exponential test case with PMF3 are to be found in the subdirectory GE-3way.

In principle the algorithm of PMF3 is more powerful than that of PMF2, thus it should also solve 2-way problems well. The main advantages of using PMF3 are the following:

- The std-dev values computed for factor elements by PMF3 also include all rotational indeterminacy
- If desired, PMF3 computes the full covariance matrix for all factor elements. Performing an SVD of this matrix will show possible free rotations as singular components with very large singular values.

There are, however, the following two drawbacks when solving 2-way problems with PMF3:

- For large data matrices, PMF3 may need so much memory that it cannot be run, or also the computation may become very slow for large matrices.
- There is no ready-made control for rotations in PMF3. One has to control the rotations (if they are needed) by hand, either by binding selected factor elements to zero or to non-zero values, or by using target factor shapes (see Part 1 of the User's Guide).

Avoiding degenerate factorizations with PMF3.

For some arrays, the PARAFAC model produces degenerate factorizations when non-negativity constraints are not employed. This means that some factor elements grow without limit towards large positive and negative values. Then positive and negative contributions cancel each other to a large degree for some or for all of data points. Such solutions are mathematically correct but useless in practice.

With PMF3, degenerate factorizations may be avoided by using a non-negligible regularization during all three stages of iteration. Normally, using smaller and smaller "*lims*" parameter values decreases both regularization and the logarithmic penalty that is needed for implementing non-negativity constraints. The optional parameter setting "*minreg* r" controls the regularization so that the strength of regularization is = max(*lims*,r). Thus *minreg* defines the smallest regularization to be used for all levels of *lims*. In order to avoid degeneracy, use *lims* as usual but introduce the optional parameter as "*minreg* 1.0", say. Then experiment with the numerical value of *minreg*, in order to find the smallest value that still prevents degeneracy. Use this smallest value in order to minimize the distortion that is caused by the regularization.

Efficiency considerations

PMF computations tend to be time consuming. Thus it pays to arrange them in an efficient way. The following points should be considered.

Convergence criteria. In the default setup convergence criteria are tight for all three penalty levels. This setup is only intended for a safe start, not for permanent use. Different arrays have very different convergence characteristics. The "activity" of non-negativity constraints is the most important question; if many factor components try to go negative then the case is "difficult". Thus it is impossible to give specific instructions about good parameter values. Instead, we try to point out things to watch.

If the first two stages have tight convergence limits then the program finds two precise solutions which are then immediately discarded when the computation proceeds to the next stage. This precision goes wasted. In most cases one may set the chi2 limit of the first stage to something like 5 units, and it may be sufficient to require 2 small steps. On the other hand, the iteration often proceeds more slowly with the smaller penalty values. Thus it may be undesirable to end the first stage(s) too early. Sometimes the iteration may get stuck to a local minimum more easily if the first stage ends early. It is good to first compute with tight limits and then relax them. Use the relaxed limits if they result in faster computing.

The convergence limit for the final stage influences the accuracy of the result. Statistical considerations seem to indicate that it is sufficient to reach a chi2 value which is within one unit from the true minimum. Theoretically it is incorrect to use a solution having a chi2 value which is more than a few units above the true minimum. Practical results may be quite good even if the distance to the true minimum is somewhat more.

Monitor level M. When testing the program, the .ini file, or the data it is good to have M=1 or even M=-1. In routine runs one might specify a larger M, perhaps M=5 or M=10. In this way the programs may run a little faster when certain diagnostic computations may be omitted.

Penalty coefficients "lims". If there is one very weak (low-intensity) factor, then it may happen that it is not visible at all when the penalty coefficient is too large. This was observed e.g. with the 4-factor PMF3 example by Rob Ross. In such cases it is useless to run with a large first lims value, because the factor structure changes when the penalty is decreased and thus whatever is computed with a large penalty gets soon discarded.

The std-dev values specified for the array have an indirect influence for the convergence. If you specify the std-dev values so that they are too small by a factor of 2, then the resulting chi2 values are too large by a factor of 4. Then also the convergence criterion is too strict and the computation takes longer than necessary. Also the std-dev values computed for the factors are unrealistically small.—If you notice that the computed chi2 values are too large, you should find out the reason and select at least one of the remedies suggested in the section on specifying the standard deviations.

Robust mode. Running in the robust mode is slower than in the non-robust mode. Having a larger outlier threshold distance parameter (8.0 or 4.0) may be faster than a smaller value, such as 2.0. In robust mode, specifying too small std-dev values may slow the convergence dramatically.

Matrix layout. In PMF2 it is slightly better to have the larger dimension in the vertical direction, i.e. the number of rows should be larger than the number of columns. This difference is not significant. — In PMF3 it may be important to order the dimensions optimally. Denote the dimensions of the data block as rows x columns x planes=mxnxq (m is the number of rows). Generally one should have m>n>q.

Computing the covariance matrix of factors. In PMF3, computing the full covariance matrix of all factor elements is extremely slow and requires a lot of memeory. Thus you should request this computation only if you really need the result. Otherwise, let the FIL code for the covariance matrix be =0.

The computing environment. The programs are compiled/linked so that if the RAM memory is too small, disk memory will be used for storing such data segments which cannot fit in the RAM memory. A shortage of memory may cause that the operating system continually exchanges data segments

between RAM memory and disk. This causes severe slowing down of the program, possibly to the extent that no computations are possible. (Buy more memory or simplify your model!) In Windows a memory shortage may be caused by having too many other tasks open simultaneously. —Use a Pentium or better if possible! And remember that it is easy to set up a "batch job" (start PMF runs in a .BAT file) so that PMF runs overnight or over the weekend!

Fine-tuning the algorithms of PMF2 and PMF3. There are a large number of unpublished tuning parameters that the end user might adjust in order to enhance the convergence rate of PMF2 or PMF3. In this way one might obtain a speed improvement of 10% to 30%. However, changing these parameters is risky because the programs might not converge properly in some cases. Their use is like trimming the family car: you might get to your destination a little faster, but the car is more difficult to drive and the engine may stop if not handled properly. Thus this technique will only be useful in situations where the program is used for analyzing tens or hundreds of similar large data sets. If this is your situation, contact the author for details.

Fortran errors, weak points of PMF programs

The language Fortran 90 is used as the basis of these programs. This is a new and advanced language and there are still errors in the compilers. Please report suspected errors to the author. — It is impossible to test the programs PMFx with all different data arrays. It is likely that the programs will behave in strange ways with some special arrays. Possible candidates for trouble are e.g.:

- arrays where some elements have *extremely small specified std-dev values.* Such arrays may lead to singularity when solving for a step, or they may lead to a bad solution.
- arrays where some elements have extremely large specified std-dev values. It was observed that in such cases one may obtain solutions which are mathematically correct but nonsense from the practical viewpoint: A factor element may get a huge value corresponding to the huge std-dev value. As a simple rule, no std-dev value $\mathbf{S}_{ij}$ should exceed both the largest value on the i^{th} row and the largest value on the j^{th} column of matrix **X**. See Paatero and Tapper, 1994.
- arrays where all values are very small or very large
- arrays where some columns and/or rows contain large values while others contain small values
- tasks where some factor elements are bound to zero. These cases may have many local optima. It may be necessary to use "good" starting values instead of random numbers.

If you suspect strange behaviour, you might try to change your task (e.g. change units used for expressing matrix rows or columns, change error estimates a little, try another starting point, and so on). If you find something, please inform the author. Don't be satisfied with just eliminating the problem from your work, help us in eliminating the problem from the program itself!

Errata

1. There is a typo in the Paatero-Tapper paper of 1994: on page 116, the element 3,3 of matrix X should be 1 instead of 0. **2**. In a few places we have used the term "alternate regression", which is wrong. The correct term is "Alternating Regression" (AR) or better yet, "Alternating Least Squares", ALS. **3**. Currently the PC versions of PMFx are based on the Lahey Fortran 90 compiler version 4.00a. Errors caused by the compiler are explained in the PMF readme file.

Requesting support and updates

Instructions for requesting support, contact information, and a list of references are at the end of Part 1 of User's Guide. Newest versions of PMF2.EXE and PMF3.EXE, of the necessary readme files, and of useful examples are stored in the ftp server of the Physics Computation Unit, University of Helsinki.

Licenced users may freely download updated program versions and run them with their previous authorization keys. The program files are packed by the program PKZIP (currently version 2.04), possibly protected by a password which will be changed infrequently, perhaps once a year.

The ftp approach is through anonymous login to "rock.helsinki.fi, directory /pub/misc/pmf". Alternatively you may approach through www: connect to ftp://rock.helsinki.fi/pub/misc/pmf/ .

Disclaimer

The programs PMF2 and PMF3 are not guaranteed to be error-free. Also, using a Dos-extender (PharLap in the current versions) is a complicated process and in rare cases can lead to complications. The programs PMF2 and PMF3 are licensed on the condition that the end user in all circumstances bears all risk of all possible damages and losses to his/her computer system and data files, related to the use or attempted use of these programs.

In order that this not be empty talk, consider if your back-up practices are sufficient. Never have important data residing on your hard disk without first making a backup copy of them on another disk or on a diskette. It is not sufficient to backup onto another file or partition on the same physical disk!