End User's Guide to Multilinear Engine Applications

File name for distribution: ME2_EndUsrGuid.pdf
Archive file name for this version: ME2EUG12.doc

Introduction

This guide is one of a set of three guides. The other two are "Multilinear Engine User's Guide" (MEUG) and the file me2scrip.txt. MEUG mainly concentrates on the mathematical problem that is to be solved. Regarding practical details, MEUG is somewhat obsolete and will soon be updated. On the other hand, me2scrip.txt gives a complete description of the script language and the special variables that are used in order to specify the ME-2 run. Together, these two files attempt to give a complete and thorough description of the program ME-2, so that new applications can be developed based on these two documents. For this reason, these two files are not suitable (being too long and too complicated) for somebody who receives a ready-made script from colleagues. When using a ready-made script, one only needs to update a few details in the script, such as file names, data dimensions, number of factors, specifications of errors, output formats, and so on. The present guide has been written in order to help the "End User" who receives a ready-made script from others. Thus this guide does not discuss all details of the scripts. – This guide discusses the modeling task from the viewpoint of environmental science. Some of the statistical conclusions may not be valid in other applications.

The Multilinear Engine (ME-2) program is different than the programs PMF2 and PMF3. Those two programs solve certain well-defined tasks. By modifying their .ini files, the user can fine tune the action but the task is always the same, i.e. the solution of a certain bilinear or trilinear problem. In contrast, the program ME-2 by itself does not do anything at all. The actions of ME-2 are defined in a "script file", a special complicated .ini file that is written in a special-purpose programming language, constructed specifically for the needs of ME-2. This guide will discuss such properties of the script file that each user should understand.

This guide also discusses standard conventions that are adhered to in many different ME-2 scripts. In this way, the instructions supplied with individual scripts need not repeat the same explanations. As an example, the notation **XX** indicates the (main) data matrix in all 2-way data analysis scripts. The conventions are explained from the viewpoint of the basic (2-way) factor analysis, where the data to be analyzed consist of a matrix of values, often enhanced by some auxiliary data.

Although the program ME-2 itself has been written in the Fortran language, the users of ME-2 do not use Fortran at all. The user only works in the script language. The script language loosely resembles the language of Matlab, and also the language Fortran. The script language is documented in the file me2scrip.txt. The original ME publication (Paatero 1999) also describes separate auxiliary Fortran programs, used for setting up the model description table for the program ME-1. Those auxiliary programs are not needed any more. The program ME-2 creates the needed tables itself, without needing auxiliary programs.

Installing and running ME-2

It is recommended that the distributed files me2wopt.exe, ME2libr.txt, me2key.key, me2scrip.txt, and this file be copied to directory \me2 (or alternatively to \pmf). Data files or

script files should not be stored in this directory. Instead, they should be kept in problem-specific directories in your directory tree. In this example, it is assumed that both the data and the corresponding script files are in \mydata\test1. (It is possible to keep data and script files in different directories but this guide does not cover such arrangements.)

The actual name of the executable file (.exe file) of ME-2 may be different from "me2wopt.exe". Sometimes, several different .exe files, with different file names, may be available, to be used on different computer architectures. In this guide, the generic name "me2wopt.exe" is used for representing all names of .exe files.

In order to run ME-2, first prepare the data file(s) and a script file named "`myscript.ini`" in the directory \mydata\test1. Then open a "Dos window" or "Command prompt" window. In this window, type the command "`cd \mydata\test1`" (do not type the quotes, they are used here for clarity). Then start ME-2 by typing the command

"`\me2\me2wopt myscript`"

or, if ME2 files have been stored in \pmf, type the command as

"`\pmf\me2wopt myscript`"

Now ME-2 should start up, so that you may follow its operations in the Dos window. When the run has finished, the result files will appear in the current directory, i.e. in \mydata\test1.

Working with the scripts

The scripts should *not* be viewed or edited with Word or Word-Perfect. A good programmer's editor, such as "Source Edit" by Joacim Andersson (free, at least so far) should be used. Probably the best available editor might be Ultraedit-32. This program is not free but the price is low in comparison to the quality. The program Notepad is better than Word but it has several drawbacks. A new free program "Notepad2" is much better than the original (Microsoft) Notepad. If using Notepad, be sure to save the files with the option "all files", not with "text files". Also, be sure to set up your Windows system so that all file name extensions are displayed. (This rule is also important for fighting viruses. If you do not know how to do this, ask your computer-aware colleagues!)

The scripts and some document files, such as me2scrip.txt, have been formatted so that they look good when using monospaced fonts, such as Courier New. Do not use proportionally spaced fonts (e.g. Times Roman) when working with these files! Do not use tabulators (TAB) in the scripts! Although the script might look good on *your* computer when TABs are used, the script would not be portable any more. On a different computer, with different sets of tabulators in use, the script would look terrible enough!

The first line of the script contains the name (including path) of the licence file that authorizes the use of ME-2. The name of the licence file is usually either PMF2key.key or ME2key.key. The path should be set up according to the directory tree of the user. The distributed script files usually contain the path-and-file as: `\me2\me2key.key` . Change this according to your situation!

Most ME-2 scripts import code segments from a standard library file. This is achieved by a line such as

```
$include $Xwrite2
```

and similar follow-up lines, usually near the end of the first script section "defines". This line indicates that routine `Xwrite2` should be imported from the standard library file (ME2libr.txt) residing in the same subdirectory as the .key file. If the library file resides elsewhere in your computer, or if a different library file is used, then the complete form of the `$include` command must be used, e.g.

```
$include $Xwrite2 '\PMF\ME2libr.txt'
$include $Mytrick '\ME2\Mylibr.txt'
```

It may be a good idea to store all general ME2-related files (licence, library, .exe files, and document files) in the subdirectory \me2. In this way, distributed scripts can be used with a minimum of modifications.

The scripts contain a command such as

```
version=1.204;
```

This command indicates that the script needs an ME-2 .exe file that is of version 1.204 or higher. Or, that the author of the script believed that the script needs such an .exe version. If the version of ME-2 in your computer is older, the script will not run. Then you should download a newer version of me2wopt.exe from
`ftp://rock.helsinki.fi/pub/misc/pmf/me2/`
(soon may be: `ftp://rock.it.helsinki.fi/pub/misc/pmf/me2/`)
or from the mirror site `ftp://ftp.clarkson.edu/pub/hopkepk/pmf/`.

If it is impossible to get a newer version of me2wopt.exe, try to change the version command in the script. This may or may not work, depending on what new features the script needs.

The scripts contain user instructions and comments on lines that begin with the percent sign "%". Also, many command lines contain end-of-line comments that are separated by a "%" character from the command proper. The comments are intended for the human user, the program does not need them. Nevertheless, it is vital that the comments are kept up-to-date when the script is modified. The user of the script should be able to trust the comments, otherwise confusion will result. The distributed scripts are licenced to ME-2 users on the following conditions:

The scripts may be modified and used in all licenced use of ME-2. No guarantee is given for the correctness of the scripts, everybody uses the scripts at their own risk. Unmodified scripts may be freely passed on to other ME-2 users. Modified scripts may only be passed on under the following four conditions: (1) the modified script has a different file name than the original one, (2) the original unmodified script is also passed on together with the modified one, (3) the comments in the modified script have been updated so that they do not contain obsolete or conflicting information about the script, and (4) whoever modifies the script, inserts his/her name and the date of the last modification in the script but otherwise does not change the author information and modification history in the script.

Trivial changes, such as changing file names, data dimensions, number of factors, robust/non-robust, etc should not be documented as modifications, and they do not require renaming the script. If the script already contains modification information from a previous user, do not delete such information. Add your own modification information after the previous one.—If you detect an error in the script, please correct the error and also inform the author of the script about the error and its correction!

Standard notation used in ME-2 scripts

Standardized notation is helpful in order that different scripts can be quickly understood and modified. The following are the most important standardized notations:

`n1`	number of rows in data matrix, *m*
`n2`	number of columns in data matrix, *n*
`np`	number of factors, *P*
`XX`	the data matrix
`30`	file number for reading the data matrix XX. The file name for XX is specified in the command "`openfile 30 ...`". This command is usually located near the beginning of the second section "equations".

`c1` and/or `c3` C1 and C3 coefficients, if/when one value applies for the entire matrix (these values will be copied to XX.C1 and XX.C3, either by the equ> commands or by explicit copying commands in the script)

`XX.C1` the matrix of std-dev coefficients C1 (each data value has its own C1)

`XX.C3` the matrix of std-dev coefficients C3 (each data value has its own C3)

`XXC1` sometimes, the std-dev coefficients C1 are first stored in the matrix `XXC1` before being copied into `XX.C1`.

`em` The errormodel code. For environmental data, usually `em=-14;` is OK.

`AA` the left (time series) factor matrix, sometimes denoted by `GG` in ME-2, denoted by **G** in PMF2

`BB` the right (source composition) factor matrix, sometimes denoted by `FF` in ME-2, denoted by **F** in PMF

`normc1` In up-to-data ME2 scripts, this value is not used because normalization is based on EM code -23 which maintains the C1 values automatically. In older scripts, `normc1` is the std-dev coefficient for normalization equations, typically `normc1=0.02;`

`aasmooc1` `aasmooc3` Std-dev coefficients C1 and C3 for equations that impose smoothness on columns of AA. Use smaller values if more smoothing is needed. Too much smoothing may distort the results, too little may cause that unrealistic wiggles appear in time series factors.

`pullc1` Std-dev coefficient C1 for equations that pull some factor element(s) towards some user-specified target values. Use smaller values for more strong pulling. Observe how Q values increase with stronger pulling.

`contrun` A selector constant whose value ... chooses between

...0 random starts

...1 a start from previously computed results

...2 a start from specified initial values, while also specifying some special conditions, e.g. that some BB factor elements be kept fixed at their initial values

`seed1` The initial value of a pseudorandom seed, used for generating random starting values of factors AA and BB. If a different random start is needed, then the `seed1` value must be changed. If several random starts are computed in one run, then the script varies `seed1` automatically so that the same result is not computed again and again.
Other seeds (`seed2, seed3,...`) may be used for other purposes in more complicated scripts.

Commonly used control values in ME-2 scripts

When the script reads values (either data or control values) from files, it will also accept “null values”. A null value is an “empty” place in a file, between two commas that are used to delimit the values. Thus in the sequence `“3, 4, , , , 5,”`, there are three null values between the values 4 and 5. The same sequence can also be expressed as `“3, 4, 3*, 5,”` by using the repeat notation. See below for special features in input files. When a null value is encountered in a file, then the variable in question will retain its previous value. Despite its name, a “null” value does not mean the value of zero. Null values are useful e.g. in situations where the previous value must be preserved but one does not know the previous value when composing the file.

Dealing with missing data values

There are three main alternatives for dealing with missing values.
(1) The user may assign some typical values to such data where a true measurement is not

available, and in addition assign large standard deviations for such data points. The standard deviation should be at least equal to half of the difference between the largest and smallest data value for the variable in question. This alternative should only be used if otherwise unrealistic variation (peaks or drops) of factors occur at the locations of missing values.
(2) The data value may be set equal to an indicator value, such as –999, in order to indicate that this is not a real data value. The indicator value must be chosen so much negative that no real data can ever be so much negative. In addition, the control value `missdatlim` must be set to a suitable threshold value, e.g. by setting "`missdatlim=-990;`". In this way, the program will not use the indicator value for fitting, even if an equation has been specified for the data point.—This alternative is recommended if the number of missing values is small.
(3) The indicator values can be identified by the script when generating the equations. The script may check each data value against some specified limit: if the value is more negative than the limit, the corresponding equation is not created at all. This alternative is useful with extremely large numbers of missing values because unneeded equations are not wasting memory space. This alternative is only practical if the script already contains the required coding for equation generation, see the script-specific instructions.

If the number of missing values is not large, then the indicator values may be inserted by hand. However, some Excel spreadsheets contain large numbers of missing values, encoded simply so that the cells have been left blank. In this case, manual insertion is not practical. The following procedure is recommended:
(1) In Excel, create a comma-separated file (.csv file) that contains the data matrix. In order to be able to see the .csv option, the list of Excel output formats has to be scrolled down from its default position. In the .csv file, the empty cells appear as two consecutive commas. The program ME-2 interprets two adjacent commas as representing one missing (i.e. "null") value.
(2) Unfortunately, Excel does not always write a comma after the last cell on the row. Thus a problem arises if the last cell is blank on any line. The simplest solution is to inspect the spreadsheet and to manually insert the indicator values to all empty cells in the last (rightmost) column before creating the .csv file.
(3) Before reading, the script must insert missing value indicators to all positions of the data array, e.g. by the command "`XX[0,0]=-999;`". This causes that the indicators will remain in those positions of XX where no real data is input. The command for insertion of missing-value-indicators may already be present in the distributed script but it is best to check the situation.
(4) Sometimes, a TAB-separated file is available instead of a comma-separated one. The instructions for reading a comma-separated file also apply for a TAB-separated file, with the following addition: Use the command "`tabiscomma=1;`" in the script, in order that the TAB be interpreted similarly as the comma.

Other control values

A group of control values occur in the scripts right after the initial comments, within the first section "defines". Some of the values have already been discussed. The following list discusses some of the remaining variables in their order of occurrence in a typical script. The user should be aware of all these controls and adjust them when needed.

`monitor` controls the amount of monitoring information written to file me2.log. The value of monitor does not have any influence on the computed results. Typically use `monitor=5;` to `monitor=20;` Note that the file me2.log is cumulative: most recent information is found at the end of the file.

`robust` Setting `robust=1` selects robust mode of fitting, see (Paatero 1997). For most environmental data sets, robust mode is recommended. Setting `robust=0` selects the non-robust mode where outliers are not

downweighted.

`posoutdist=4;` Limit for *positive residuals*: if the scaled residual exceeds 4, then the data point is considered an outlier and downweighted. (residual is the difference between measured and fitted values)

`negoutdist=4;` Limit for *negative residuals*: if the scaled residual is <-4, then the data point is downweighted. Depending on the distribution of residuals, smaller limits (e.g. 2) may also be used.

`bdlneg=1;` Used if Below Detection Limit (BDL) data values have been encoded so that the limit, with a minus sign, is entered into the data table. Then ME-2 will interpret *all* negative data values (except for missing data indicators) as indicating BDL limits and fit them accordingly.

`precmode=15;` This value can be left as it is.

`numtasks=5;` This indicates that 5 different random starts are to be computed. Because of the risk of arriving at a "local minimum", one should not rely on one random start. To be quite sure, 20 random starts should be computed. One of the computed results (usually the one with lowest Q) should be chosen for further computations, see next item `numoldsol`.

`numoldsol` This number is used when the results from a previous run are used as the starting point. Assume that the 3rd random start was chosen. Then `numoldsol=3;` indicates for the continuation run that the 3rd result is to be used.

Mathematical-statistical adjustments in multilinear models

Many mathematical adjustments are used for optimizing the solution process, i.e. the *algorithm*. Those parameters will not have a significant influence on the numerical values of the solution while they may significantly improve the efficiency of the computations. In general, the end user does not need to be concerned with these parameters: the ready-made scripts should contain reasonable values for these parameters.

Other mathematical parameters are of different character: they are part of the *model* that is used for modeling the reality. The number of factors *P* is the most fundamental of these parameters. Handling of these parameters cannot be automated. Choosing good values for these parameters is an essential part of the professional skills of the analyst, even of the required skills of the "End User". Physical and chemical properties (known or assumed) of the real-world situation must be taken into account. Also, statistical properties of the measurements and of the physical situation must be considered. Even the purpose of the model at hand influences the selection of parameters. As an example, the optimal number of factors *P* may be smaller if the goal is to determine well the strongest sources of pollution at a receptor site, and larger if a weak source, e.g. a distant smelter, is to be studied.

Judging the model by the obtained *Q* value

Sometimes, users of PMF models argue that a certain *P* is correct because it leads to a *Q* that is close to the theoretically predicted *Q* value. Such reasoning is often false because the obtained *Q* values depend strongly on assumed standard deviations of data values. Both measurement errors and modeling errors contribute to residuals and hence to *Q* values. Even if measurement errors can be reliably estimated (usually not the case!), modeling errors remain largely unknown. Thus the usual situation is that standard deviations of measured values are specified for PMF models on the basis of obtained *Q* values.

The differences of Q, as obtained from different variants of a model, are often a main indicator for preferring one of the variants. When the number of factors P is increased by one, then the number of free parameters in the model increases approximately by $(n+m)$. Increase of the number of free parameters corresponds to a similar (non-significant) decrease of the Q value. In order that the increase of P by one should be deemed useful, the decrease of Q should be "significantly" more than $(n+m)$. More specific criteria cannot be given because the statistical properties (distributions and covariances) of residuals are largely unknown for environmental data.

Using the FPEAK rotational option

Mathematical formulation of FPEAK rotations has been developed in January 2007. For details, see Paatero (2007, to be submitted). This formulation will be used in EPA PMF and later in other ready-made scripts. As seen from the user's side, there are three alternatives: (1) FPEAK is not used, (2) FPEAK is used in its original PMF2-like form, and (3) an enhanced FPEAK is used, so that individual rotational parameters are defined for all elementary rotations.

In (2), the overall rotational parameter ϕ (similar to Fpeak in PMF2) is specified by the user. Best values should be determined by trial-and-error, the order-of-magnitude range of ϕ may be from -2 to 2. If no rotation is desired, then alternative (1) should be selected rather than (2) with $\phi=0$.

In (3), the user specifies a matrix "FPEAK" of dimensions $P \times P$. The values of off-diagonal elements $FPEAK_{pq}$ indicate how strongly column p of **G** is to be added to (if $FPEAK_{pq} > 0$) or subtracted from (if $FPEAK_{pq} < 0$) column q of **G**. (The diagonal elements should be =0.) Regarding rows of **F**, the transformation follows the rules of rotations: row q of **F** is subtracted from row p of **F** if $FPEAK_{pq} > 0$ and added to row p of **F** if $FPEAK_{pq} < 0$.

With FPEAK rotational forcing, the outcome may not always be as expected, however. Non-negativity constraints may couple rotations with each other: in order that a certain rotation may be possible, another rotation (or rotations) may also need to happen. Setting an element of FPEAK to zero does not prevent the corresponding rotation from occurring. The zeros in FPEAK only mean that the corresponding rotation, should it happen, is neither rewarded by a decrease of Q^{aux} nor penalized by an increase of Q^{aux}.

Auxiliary equations, rotations, and pulling of factor elements

A more detailed discussion of these topics is available as the manuscript Paatero (2007, to be submitted).

In programs PMF2 and ME-2, the object function Q consists of two parts: $Q = Q^{main} + Q^{aux}$. The first part Q^{main} consists of residuals of data fitting equations. When judging a model, one should mainly look at this first part. The second part consists of auxiliary equations, such as equations that normalize G factors to average=1. There are no set rules for the expected or allowed size of Q^{aux}. In "pure" 2-way PMF models, normalizing equations do not contribute anything to Q^{aux}. If there are pulling equations or FPEAK rotations, then normalization equations may contribute up to some hundreds of units to Q^{aux}. If there is a normalization conflict or excessively strong pulling in the model, then normalization might contribute thousands. If this occurs, the reason should be examined and the situation corrected. Pulling equations may contribute varying amounts, depending on how the equations are set up. Smaller or larger values of pulling Q^{aux} should not be taken as a signal of success or failure.

Sometimes, available *a priori* information tells us that in a certain source, the true concentration of a certain chemical element should be zero or should have a known value. Then it makes sense to impose this known value on the model. In PMF2 and in ME-2, it is easy to force chosen factor elements to zero. In ME-2, it is also possible to fix chosen factor

elements to non-zero values. Such forcing of the solution causes an increase of the value of Q^{main}. Then one must decide if the increase is acceptable or if it is too large. And again, there is no solid answer because the statistical properties of errors and hence of residuals are not known well enough. For a larger data set, the increase can probably be larger than for a smaller data set. If the data set is known to contain problems (e.g. incompatible instruments, or variation of true source profiles with time, say) then a larger increase of Q may be acceptable. As a simple rule of thumb, increases of hundreds may be acceptable while thousands appear questionable.

Instead of fixing factor elements to chosen values, it is often safer to use "softer" means, viz. "pulling" factor elements up or down. Denote by f a factor element that is to be pulled towards a value ("anchor") a. The pulling equation introduces into Q^{aux} the contribution $(f\text{-}a)^2/s^2$, where s is the "softness" of the pull. The smaller the value of s, the larger the introduced Q^{aux} and the stronger the pull. When the program minimizes the overall Q, it will accept an increase of Q^{main} if a larger decrease of Q^{aux} is achieved in return.

Detailed discussion of pulling equations is in (Paatero 2007). The formalism for pulling equations in ME-2 is such that the left-hand side must always be a constant value (not a variable factor element). It follows that if the purpose of the equation is to make the two factor elements f_{pj} and f_{pk} equal, then the natural form $f_{\text{pj}} = f_{\text{pk}}$ cannot be used. Instead, the equation must be formulated as $0 = f_{\text{pj}} - f_{\text{pk}}$.

Pulling is often used when one wishes to see if a certain assumption is compatible with measured data. Then one wishes to limit the increase of Q^{main} small enough so that the pulled (distorted) model can be considered acceptable whenever Q^{main} is not larger than the preset limit. Again, there is a problem in setting the limits for increase of Q^{main}. Statistical reasoning does not tell us the limits. Increase of 10 is certainly acceptable for one pulling equation, while thousands are certainly doubtful. Increases of hundreds are probably acceptable when several pulling equations are used. Experience regarding successful and unsuccessful use of pulling should be published!

In ME-2 version 1.203 (Feb. 2007) an automatic mechanism for pulling has been introduced. The user of this mechanism defines, for each pulling equation, an upper limit for the increase of Q^{main} that may be caused by the equation in question. This limit is theoretically justified but conservative: in practice (=almost always), the increase of Q^{main} will be at most 50% of the specified limit, and usually between 15% and 50% of the limit whenever the system uses maximal pulling. If maximal pulling is not needed, then the increase will be even less. When using the system, one may request "maximal pull" up (or down): the pulled quantity will obtain the largest (or smallest) value that is compatible with the chosen Q limit. Alternatively, one may request a pull towards a specified target value. If this target value is compatible with the limit, then the pulled quantity will approximate the target value. If the target is too far to be accessible, then the result of targeted pulling is identical with the result of maximal pulling.

When using the automatic pulling equations, the following quantities are needed:

- dQ for each equation. These values will typically range from 10 to 100 per equation. Different equations may have different dQ values. The larger values should only be considered when only a few pulling equations are needed.
- Expected change (absolute value of change) $f^{\text{expected\_step}} > 0$ for the pulled quantity, or the target value f^{target} of the pulled quantity. The target value must not be the same as f^{init}. When pulling down with the intention to reach zero if possible, the expected step or target value may be set to a "reasonable" negative value, e.g. to $-0.1\sum_p f_{pj}$ when pulling down a value f_{qj}. In this way, the equation will not fail even if the specified initial value happens to be zero already.

The user must decide about these values, dQ and the target or expected value.

Even when a ready-made script is used, the End User must decide what values to use for the limit *dQ* of allowed Q^{main} growth. After the fit has been computed, one may check the size of the actual increase. This size may be much smaller than the theoretical limit, the sum of specified limits for individual equations. If the actual growth was small, then it may be meaningful to redo the computations with higher limits, if it appears that desired results can be obtained in that way. If it is important to achieve a maximal pull so that the allowed increase of *Q* is maximally used, then it may be advisable to arrange the pulling computations in two stages: in the first stage, apply only a fraction of the intended *dQ*. Then perform a second computation, starting from the results of the first one. In the second computation, use the remaining amount of intended *dQ*.

Pulling individual factor elements up and down can be used for error estimation. This method accounts both for random (noise-induced) errors and for rotational ambiguity. On the other hand, this method is dependent on correct error estimates of data values. If too small std-dev have been specified for data values, then this method will compute too narrow error estimates for the factors. This method has worked well with simulated data sets. So far, there is not enough experience with real data sets. If testing this approach, one could begin by setting the growth limit of Q^{main} to 20 or to 40. Please report about your results if you try this approach.

Normalization

Normalization equations are a special case of pulling. All ME-2 scripts must contain some form of normalization. In older scripts, normalization equations used EM code -12. In these equations, a C1 or std-dev value had to be defined. In up-to-date scripts, normalization equations use EM code -23. Then the program controls the C1 values dynamically, and the control value `normc1` is not used. Thus discussion about pros and cons of different values of `normc1` is obsolete and should be disregarded.

Other remarks

Occasionally, PMF2 users have included in their reports a statement saying that PMF2 cannot accept negative values, or that PMF2 cannot accept zero values. Both of these statements are based on misunderstandings, there is no truth in them. Small data values, even zero values and negative values, should preferably be entered as such in the data matrix, both for PMF2 and for ME-2 based models. There is *absolutely no reason* to convert small values to BDL indicators. So doing will incur a bias in the results, and there is *no reliable way* to remove such bias afterwards. The use of BDL indicators is well motivated when individual data values are reported to non-scientists who do not understand what an error estimate means. However, when a multivariate data set is analyzed by statistical methods, BDL indicators serve no useful purpose at all. They are still often used, mainly because of a century-old tradition; try to convince your laboratory that they should give you the uncensored values if such values have been measured.

In ME-2, the use of errormodel codes (`em`) and std-dev coefficients (`C1` and `C3`) is similar to their use in PMF2. Regarding these details, please refer to PMF2 handbook for the time being. In contrast to PMF2, ME-2 allows that different data values have different errormodel codes.

In the script language, a shorthand notation is used when dealing with matrices or vectors, as follows: `XX[0,0]` means “all elements of matrix XX”. The elements are accessed in row-wise order: first all elements of the first row, then of the second row, and so on. Similarly, the notation `XX[j1,0]` means “all elements on row j1 of matrix XX”. Also, `XX[0,j2]` means “all elements on column j2 of matrix XX”. These shorthand notations apply for all matrices and vectors, not just matrix XX. These notations are often used when reading or writing matrices.

Modifying the contents and format of ME-2 files

Files read by ME-2

In practice, the data tables come in different formats. Often, the data form one table and the error estimates (C1 coefficients) form another table. The two tables can be in one file, or in two different files. Occasionally, the data and their errors have been merged in one single table, so that odd-numbered columns contain the data values while the errors are in even-numbered columns. It is not practical to distribute separate script files for dealing with all such formats. Instead, the End User should be prepared to adjust read commands according to the format of the data.

In the following examples, it is assumed that the main data file is file # 30, and a possible secondary file is #31. In examples 1 to 4, both data and errors are read from file 30, where they appear as two separate tables. Different alternatives are shown in order to illustrate the flexibility of the script language. Example 5 reads data and errors from files 30 and 31, respectively. (In order that file 31 can be used, a command "`openfile 31, ...;`" must be inserted for opening it, cf. the file opening command for file 30.) Example 6 reads a table where data and errors are in alternating columns. Example 7 shows how the data matrix can be read when sample identifier words appear in the first column of the spreadsheet, i.e. as first items on each line of file. The example stores these words in the text table AAHEAD.

(1)
```
  read 30, ' ', XX[0,0];
  read 30, ' ', XX.C1[0,0];
```
(2)
```
  read 30, ' ', XX[0,0], XX.C1[0,0];
```
(3)
```
  for> j1=1:1:n1;
    read 30, ' ', XX[j1,0];
  for!;
  for> j1=1:1:n1;
    read 30, ' ', XX.C1[j1,0];
  for!;
```

(4)
```
  for> j1=1:1:n1;
    for> j2=1:1:n2;
      read 30, ' ', XX[j1,j2];
    for!;
  for!;
  for> j1=1:1:n1;
    for> j2=1:1:n2;
      read 30, ' ', XX.C1[j1,j2];
    for!;
  for!;
```

(5)
```
  read 30, ' ', XX[0,0];
  read 31, ' ', XX.C1[0,0];
```

(6)
```
  for> j1=1:1:n1;
    for> j2=1:1:n2;
      read 30, ' ', XX[j1,j2], XX.C1[j1,j2];
    for!;
  for!;
```

(7)
```
  for> j1=1:1:n1;
```

```
  read 30, ' ', AAHEAD[j1], XX[j1,0];
for!;
```

Repeat notation, null values, and end-of-line comments in input files

Data and control files read by ME-2 may contain certain advanced features. These features are similar to those defined for the "list-directed" input in Fortran 90 and newer Fortran versions.

A repeat notation represents several occurrences of one numerical value in a shorthand notation. The following file segment `"15, 5, 3*2, 2*-1, 7"` is equivalent to `"15, 5, 2, 2, 2, -1, -1, 7"`. Repeat notations are particularly useful when control matrices are to be filled with large numbers of repeated values. Repeat notation can be extended over one or several rows of a matrix, or over one or several columns if the matrix is read columnwise. It could even be extended over several matrices but that is not recommended because of the risk of errors.

A "null value" is an empty place between two comma characters or between a newline and a comma (not between a comma and end-of-line). When the reading process encounters a null value in a file, then the value of the variable in question remains unchanged, i.e. retains its previous value. A null value is also present in a repeat notation if the comma follows immediately after the asterisk.

Data and control files can contain end-of-line comments. They begin with the character slash `"/"`. After the slash, any text may occur on the same line. When reading, ME-2 ignores completely the slash and the text after the slash. However, the slash must not follow immediately after a numerical value. At least one space is needed between the value and the slash. End-of-line comments have proven extremely useful in making control files more easy to understand. Their use is strongly encouraged!

Files written by ME-2

Many scripts contain text tables named `AAHEAD`, `BBHEAD`, and `FACTHEAD`. These are intended for meaningful annotation of factor matrices. There is in `AAHEAD` one word for each row of the AA factor matrix, i.e. for each sample. If possible, try to arrange data input so that sample headers are read into words of `AAHEAD`. Then they can be written together with the numerical values. Similarly, chemical names of variables should be input into words of BBHEAD, so that they can be written together with the source composition factor matrix BB.

When the run is started from random values, then factors appear in random order in factor matrices. Hence, meaningful headers cannot be written for their columns (=factors). However, when a run is to be started from a known previous result, then factors can be identified and named. Those names can be entered into the table `FACTHEAD` so that they can be written as column headers in tables of factors. Adjusting the writing commands requires that the user acquires an understanding of how writing is arranged in the scripts. Some of the publicly available scripts (e.g. the standard 2-way script) contain a small subroutine for writing the headers of factor columns. This subroutine begins with the line

```
subroutine> Fhead{fi}{};
```

near the end of the first script section. This subroutine needs to be fine tuned in order to obtain a nice layout of results.

The order of magnitude of BB factor elements is quite variable. The scripts may contain fixed-format specifications for BB writing, such as "`(F8.4)`". The largest values that can be written in this format are 999.9999 and –99.9999. (If larger values occur, they will be written as `"********"`). On the other hand, the smallest non-zero value is 0.0001 (the digit 4 in the format means that 4 decimal places will be written). If the values to be written exceed one of

these limits, then the format has to be adjusted in order to accommodate the magnitudes of the data. An example of a BB writing command is the following:

```
callsubr Fwrite2{40,'(30(1X,F8.5))', '(1X,A10)'}{BB,BBHEAD};
```

This command writes the BBHEAD texts on the left side of the BB table, and uses 5 decimal places for the BB values. This command is near the end of the script, in section "`postproc`".

Using "formats" when writing to result files by ME-2

ME-2 uses format specifications in the same way as the Fortran language. If you have access to Fortran textbooks, please refer to them for details. A short summary of formats is given in the file "me2guid.pdf".

When specifying formats, the following three conflicting aspects must be taken into account:

1. Visual clarity: the exponential notation is not easy to read, although it can represent all values (large and small) with desired precision.
2. Must be able to represent even the largest values. E.g. if three digits are available for representing ones, tens, and hundreds, then anything above 999 is too large for output and leads to the output of `"*******"` instead of the desired value.
3. Must be able to represent even the smallest values with adequate precision. E.g. in aerosol science, concentrations of trace elements are often quite small, so that a precision of 0.0001, say, may not be sufficient for the source profile matrix.

Because of these aspects, ready-made scripts may not always have the best formats for every use. Then one needs to modify the format. If you obtain the error indicator of `"*****"`, then you might modify the number of decimal digits in the output. If the original format was `'(1X, F8.5)'` you might try `'(1X, F8.4)'`, giving only four decimals in a field of width 8, instead of five decimals. If you obtain the output of `0.0000` for a value that should not be zero, try the opposite modification of the format, i.e. increase the number of decimal digits. In both cases, it is also possible to adopt the exponential notation, such as `'(1X, G12.4)'`or `'(1X, E12.4)'`. However, then the output field becomes wider which is often not convenient (12 vs. 8 in this example).

References

Pentti Paatero, The Multilinear Engine - a Table-driven Least Squares Program for Solving Multilinear Problems, Including the n-way Parallel Factor Analysis Model. *Journal of Computational and Graphical Statistics* (1999), Vol **8**, Number 4, 854-888.

Pentti Paatero, Least squares formulation of robust non-negative factor analysis, Chemometrics and Intelligent Laboratory Systems **37** (1997) 23-35.

Pentti Paatero, Rotational tools for factor analytic models implemented by using the Multilinear Engine (February 15, 2007) (To be submitted)