User's guide for the Multilinear Engine program "ME2" for fitting multilinear and quasi-multilinear models.

Pentti Paatero
University of Helsinki, Dept. of Physics
BOX 9, FIN-00014 HELSINKI/UNIVERSITY, FINLAND

February 10, 2000

Abstract

The Multilinear Engine (ME) is a new concept for solving all kinds of multilinear and quasi-multilinear problems. The multilinear least squares model to be fitted is specified by enumerating all details of all equations that together make up the model. The end user performs this specification by using a script language built into the program ME-2. When the model has been specified, the "Engine" part of ME-2 fits the model by using a modified form ot the preconditioned Conjugate Gradient algorithm.

1. User's guide for the program ME-2 for fitting multilinear and quasi-multilinear models: introduction

This is a preliminary version of the ME-2 User's Guide. Please report all unclear or erroneous passages to the author. Also, please report about such details that should be covered in more detail!

The "Multilinear Engine" concept has been published in (Paatero 1999, due to appear in Dec. 1999 [3]). This JCGS paper should be read in parallel with the present User's Guide. Much material in the paper is not repeated in this guide. On the other hand, the paper is already obsolete in some aspects because getting the paper through the refereeing-editing-waiting process has taken almost two years. In particular, the paper is built on the basic assumption that two programs are used together for solving the multilinear problem: the program for specifying the structure of the model ("The Model Maker") and the program for fitting the model ("The Multilinear Engine"). However, the current program ME-2 contains both functions in itself so that no other programs are needed for the solution of the multilinear problem. Also, ME-2 contains possibilities for performing all such post-processing operations that might be needed for representing the results in a form suitable for human inspection.

The program ME-2 has been written for solving multilinear and quasi-multilinear models. It solves models where the data values are fitted by sums of products of unknown (and known) factor elements. The name "Engine" refers to the organization of the program: In the first part of the program, the model is set up according to instructions written by the user. The first part generates a large table that specifies the structure of the model. The second part is the "engine": it reads the structure table and works according to the instructions stored in the table.

In so doing the engine program computes the solution to the problem specifed by the user. The user need not be aware of the details of the table. The engine program does not "know" which unknown factor elements belong to which factor matrices. The table acts as a separating wall that separates the worlds of the user and the engine program from each other.

There will be two different kinds of users of ME-2: (1) *specialist users* who set up new kinds of models by writing new ME-2 scripts, and (2) *end users* who receive ready-made and well-tested scripts from specialist users. The end users will only make minor changes to the scripts, such as: changing the dimensions of data arrays and factor matrices, changing output formats and headings written by the script, and perhaps changing convergence criteria and some a-priori information such as equations representing target shapes of factors. The end users do not need any prior programming experience.

The specialist users will create entirely new scripts. They should have some prior experience with some programming language, e.g. BASIC or Matlab. They may access all features of ME-2. The two groups have different needs of documentation. At a later date, this document will be organized so that both groups may more easily find what they need. Right now, this document is mostly oriented towards the needs of the specialist users.—Usually the specialist users will also write some instructions for using the scripts they have authored. Thus the end user will get some information with the program ME-2 and some with the scripts (s)he is using. It is also possible to include necessary instructions as comment lines in the script. Experience will show if this is or is not practical.

The program ME-2 is controlled by an initialization file or .ini file, written in a special script language. This language is described in a separate document file ME2scrip.txt. Read the language document in parallel with this users guide. For the time being, ME2scrip.txt is oriented towards specialist users. We hope to clarify the structure of this document so that end users may more easily find what they need.—Additional documentation is contained in the form of demonstration .ini files, such as *parafac.ini* and *parasimp.ini*. Probably the fastest route to using ME-2 is by studying these files and referring to the other documents only where needed. The file parasimp.ini solves the PARAFAC problem in the simplest possible way, without any complications. In contrast, parafac.ini contains several alternative solutions that are chosen by setting certain control variables equal to zero or one. In this way, parafac.ini demonstrates how these different techniques are used. For this reason, parafac.ini is more complicated than any practical script files: it would not make sense to include so many different alternative solutions in any single practical script.

There is no graphical user interface in the ME-2. The .ini file is the main interface: the user codes all necessary commands in the .ini file and then ME-2 executes these commands automatically. However, for exceptional situations, there is also the possibility of interaction between the user and the script: the script may write messages to the user by using the Fortran output unit number 6 (the *standard output*) and read user responses through input unit 5 (the *standard input*).

1.1 Different notations describing the same multilinear problem

The three-way factor analytic Candecomp-PARAFAC system of equations

$$x_{ijk} = \sum_{p=1}^{P} a_{ip} b_{jp} c_{kp} + e_{ijk} = y_{ijk} + e_{ijk} \qquad (i = 1, \ldots, I,\ j = 1, \ldots, J,\ k = 1, \ldots, K) \tag{1}$$

illustrates the *customary problem-oriented notation*. The three-way array $\mathbf{X}$ contains the data to be fitted. The matrices $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$ in equation (1) may be called component matrices,

factor loading matrices, factor weight matrices, or simply factor matrices. Their elements, the *factor elements*, are the unknowns, to be determined by the program ME-2.

The *one-dimensional notation* arranges all data values as one long vector called $\mathbf{x}$. This vector will contain all elements of the matrix or array $\mathbf{X}$, and also elements of any other main or auxiliary data that is included in the problem specification. Similarly, the one-dimensional notation arranges all factor elements as one long vector called $\mathbf{f}$. Thus all elements of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are mapped onto $\mathbf{f}$. This mapping happens in the order of the definitions of the factor arrays in the script. If $\mathbf{C}$ is defined first, say, then its elements appear first in $\mathbf{f}$. For the most basic use of ME-2, the user need not be aware of the one-dimensional notation that is used inside the engine program. However, there are situations where the one-dimensional notation is the only possibility: when discussing the uniqueness questions and the uncertainties of the computed results, the one-dimensional notation must be used in order to avoid intolerable complications of notation, see [2]. The Jacobian matrix and the Hessian matrix are indexed according to the one-dimensional notation. Also, the block-diagonal preconditioning is best understood in the one-dimensional notation.

The third notation is the *ME-2 script problem-oriented notation* used within the script program. This notation mimics the customary mathematical notation of equation (1) as well as possible within the constraints imposed by the script language. It is very desirable to obey a uniform naming standard in the scripts, it will help in understanding and debugging scripts written by others. The suggested standard may be studied in sample scripts. The standard is also demonstrated in this text. The arrays and matrices are denoted by two or more upper-case letters. XX denotes the main data array, while AA, BB, and CC denote the three factor matrices. Scalar variables are denoted by two lower-case letters or one lower-case letter followed by one digit. The dimensions of XX (customarily I, J, and K) are denoted by n1, n2, and n3. The indexes i, j, and k of equation (1) are denoted by j1, j2, and j3. Thus j1 runs from 1 to n1, etc. The dimensions of factor matrices AA, BB, and CC are n1×np, n2×np, and n3×np, where np is "the number of factors". In order to enable efficient (multiple-term) encoding of the model and efficient preconditioning (see below), it is essential that the number of factors is the second dimension of factor matrices or factor arrays.—In mapping to the one-dimensional arrangement, the second dimension is the most rapidly varying one.

In the script, equations cannot be identified as such. The equations only exist by reference to the left side (data) elements. Thus the main equations (1) are referenced as XX[1,1,1] to XX[n1,n2,n3]. Sometimes one needs to include in the model such equations as AA[1,1]=0 or AA[2,2]=AA[2,3]. These *auxiliary equations* need to be transformed into a form where their left sides are elements of auxiliary data matrices. The equations could be written as AANULL[1,1]=AA[1,1] and AANULL[2,2]=AA[2,3]–AA[2,2], where AANULL is *an auxiliary data matrix*, to be defined in the script. Elements of the matrix AANULL must be set to zero by the script.—See the demonstration script parasimp.ini for examples of specifying the main and auxiliary equations in the script.

1.2 Installation of ME-2

1.2.1 Dos-Windows versions

There is no special installation tool for the program. The user should create a suitable directory, e.g. C:\PMF or D:\PMF, and copy all .exe, .key, and auxiliary files into this directory. The directory should preferably be included in DOS/Windows path. For the time being, the programs are Windows 32-bit applications and must be run under a DOS window in Windows 95, Windows 98, or Windows NT. A Linux version has also been compiled. (The author may

also produce extended-DOS versions, if needed).

Currently the Windows version of the program ME-2 for 486-Pentium platforms is based either on the Fortran 90 compiler LF90 by Lahey Computer Systems, or the Lahey-Fujitsu compiler LF95 for Fortran95. The testing-level versions are compiled with LF90, the fully optimized versions with LF95. The present instructions are for LF90.

The .EXE file(s) for ME-2 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 ME2OPT.EXE or ME2TST.EXE. If you wish, you may rename the files to any other names when copying to your fixed disk. It is recommended that you should mainly use those .exe files whose names contain the letters "tst" or the letter "t". These files are safe to use because they have been compiled so that certain error detecting mechanisms are included. Newest versions are only available as "tst" files. The files whose names contain the letters "opt" or the letter "o" are slightly faster, because some error detection mechanisms have been omitted. However, they should only be used in situations where repeated high-volume analyses are to be made, and only if achieving the fastest possible running times is critical. Make sure that you do not inadvertently use an obsolete version when using an "opt" file!

In the Dos window, ME-2 is started by a command line such as: *me2wtst parafac* . The first item is the name of the .exe file, the second is the main part of the name of the .ini file (the extension .ini is assumed by default). It is also possible to specify the full name of the .ini file.

The file lf90.eer enables the LF90 system to write plain-language error messages. Also there is the LF90 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 written by LF90.

You will need an authorization file or license file or "key file", usually it will be emailed to you. The default key file name is me2key.key. The key file should preferably be copied to a directory called C:\PMF or D:\PMF. One key file may contain licenses for additional programs, e.g. for PMF2 and/or for PMF3. A file containing multiple licenses may need to be copied to the directory multiple times, under different names. The license file name and path are included in the first line of a ME-2 script file. This enables that the license may be placed in arbitrary directory on your disk, if needed. The key file contains licensing information, e.g. information about the licensee and about the licensing period. This information is sealed with a numerical check code. Do not change the contents of the key file when copying! Note that there should be End-Of-Line character(s) at the end of the .key file!

If you would need to run ME-2 in Windows 3.11 Dos box, you might perhaps set up the Dos startup command DOSPRMPT.PIF so that ME-2 continues to run 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 ME-2 even while you are working elsewhere! But be careful not to work on those files which ME-2 is either reading or writing to! In the file readme2.txt you may find additional information, e.g. about how to configure the dos box of Windows95.

See the disclaimer at the end of this User's Guide for important information!

1.2.2 Linux versions

Recently, a Linux version has been compiled by using the Absoft compiler. It appears that this compiler has a lot of problems. Thus some experimentation may be needed before a useful

script can be written.

Make sure that all files, including the .key file, are in Linux format when using the Linux version! The ME-2 demonstration script files stored in the ME-2 FTP site are in Dos format and need to be converted to Linux/Unix format before they are used in a Linux environment.

See the preceding section about suggestions for handling the .key files. Note that there should be End-Of-Line character(s) at the end of the .key file!

See the disclaimer at the end of this User's Guide for important information!

1.2.3 The following items are needed when using the program ME-2:

- The Multilinear Engine program ME-2 for specifying the model and for fitting it. The program is distributed as ready-to-run .exe files, named e.g. me2wtst.exe, me2wopt.exe.
- A programming-oriented text editor for editing the .ini files and the data files. In Windows, Notepad is just barely useable. Much better is Programmers File Editor (PFE) which is available at no cost e.g. from http://www.lancs.ac.uk/people/cpaap/pfe/ . *If you are running a Dos/Windows version of ME-2, make sure that your text editor saves the files in Dos/Windows form (not in Unix/Linux form).*
- Data file(s)
- An .ini file for controlling the program ME-2. For typical uses, ready-made .ini files are distributed together with ME-2.
- A license file or ".key file" which enables the use of the program ME-2
- Auxiliary files, possibly required by the Fortran run-time system (this depends on the Fortran system that was used for compiling ME-2). None are needed for LF95.

The paper "The Multilinear Engine — a Table-driven Least Squares Program for Solving Multilinear Problems, Including the n-way Parallel Factor Analysis Model" [3] is necessary reading, information from that paper is not repeated in this document. The file me2scrip.txt contains a description of the script language. The file readme2.txt contains last-minute information regarding the program ME-2.

The program ME-2 writes a number of remarks about the task in the file ME2.log. It is good practice to look through these remarks after the run. Setting up a run for ME-2 is complicated and errors tend to happen. The remarks in the log file may help one in noticing and correcting some of these errors.

In the program documentation, error messages, etc. the expressions "data point, data value" and "equation" are sometimes used interchangeably.

2. Specifying the equations of the model by using the script language

It is advisable to use best possible layout and many comments in that part of script that creates the equations. In this way finding the errors in the script is as easy as possible. Also, future modifications are possible in this way.

Each main or auxiliary equation is defined by executing the command *equ* in the script. In very large models, if memory space is a problem, the command *xequ* (for extrapolated equations) may be used in place of *equ*. Equations defining subexpression (SE) factors are created by executing the command *seequ*. Details of these commands are specified in me2scrip.txt. Use of SE equations makes many of the more complicated models run much faster than without. The only disadvantage with SE equations is that the .ini file is somewhat complicated.

There must be one data array element corresponding to each non-SE equation to be defined. The opposite is not true: it is permissible to omit equations corresponding to such data values which convey no useful information. If there is a value without a corresponding equation, then such a value is omitted from the analysis although it is present in the array of measured values or auxiliary data values. It is also permissible to omit defining equations of such SE factors which are not referenced by other equations.

The command defining a main or auxiliary equation must also specify the *data value* corresponding to this equation. Sometimes this may be inconvenient. Then one may just specify the value zero when specifying the equations. The data values may be loaded later to the array connected with equations. Usually, the specifications of the equations also specify the *coefficients C1 and C3*, needed for evaluating the standard deviations connected with data values, and the *errormodel codes*. All these values may also be specified later, if desired. For main equations, the standard deviations should be related to the accuracy of measurements and to the expected size of modelling errors. For auxiliary equations, the standard deviations reflect the desire of the user, how much weight she assigns to each auxiliary equation.

3. Details of the .ini file

The .ini file contains a number of initial settings that are needed for each run. It is best to take a copy of a good .ini file as the starting point for new developments. In this way, no essential initial settings are forgotten. Good templates are e.g. the files parafac.ini, parasimp.ini and GE-demo.ini. The .ini file is written in a special script language. This language is described in a separate document file ME2scrip.txt. Read this document in parallel with the users guide. The .ini file should be written by using a ”text file editor”, e.g. Notepad or PFE. It is essential that the .ini file is a so-called ascii file or ”flat file”, containing no formatting codes.

Licence: path/file_name This item (on the very first line of the .ini file) indicates where the license file is stored. Change this according to your installation. The file name may be different from the default, e.g. D:\pmf\pmf2key.key. Note that almost all details in the .ini file are case sensitive. Thus you need to write *Licence*, not *licence*. The names of the *special variables* are written in lower-case letters.

version This *version stamp* indicates that the .ini file was written according to the rules of the indicated version number of ME-2. The rules of the script language, the names of variables, or other details may be changed in future versions of ME-2. Thus the version stamp indicates the form of the .ini file and prevents old .ini files from being misunderstood by a newer version of ME-2. Usually each version of ME-2 will understand at least one older generation of .ini files.

3.1 Special variables that control the iteration

The .ini file contains settings of many special variables that control how the fitting task is performed. A few of these variables are described well enough in the demonstration files, e.g. in “parafac.ini”. A number of the special variables are described in the following, roughly in the order of appearance.

monitor This variable controls the amount of monitoring output generated by ME-2. With the default value monitor=–3, every third step is reported and auxiliary monitoring information is written (to .log file and also to screen). If Monitor=M$>$1 then the program writes (to screen and to the .log file) only on every Mth step, and auxiliary monitoring information is not written.

In routine work, one could perhaps have M=9 or even M=99. With new applications, and when trying to find an error, M=–3 (or M=–1) is recommended. Other negative values of M may produce special massive diagnostic files, intended for finding internal errors in ME-2. The general principle is that more positive values produce less output. The values M=–3 and M=13 write 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. It might be used for comparing speeds of different computers and/or operating systems.

robust This is an example of a "*truth-valued*" parameter. The possible values are 1 meaning "True" and 0 meaning "False". In this case, 1=True means "Robust mode", 0=False means ordinary non-robust weighting.—Use the robust mode unless you know positively that the errors in your data are approximately normally distributed and that there are **no** outliers and **no** non-representative values in your data.

posoutdist, negoutdist These values affect handling of outliers with positive (*pos*-) and negative (*neg*-) residuals. In robust mode, such data values are treated as outliers (=get a decreased weight in the fit) whose scaled residuals exceed the corresponding *posoutdist* or *negoutdist* parameter value.

missdatlim All main data values which are below the value *missdatlim* or *Missing-Data-Limit* are interpreted as missing data. The corresponding equations are entirely eliminated from the analysis, thus e.g. the std-dev coefficients and errormodel codes for such equations are of no significance. Auxiliary equations are not controlled by this option at all. If missing values are not encoded as negative numbers, then *missdatlim* should have a large negative value.

bdlneg The setting *bdlneg=1* specifies that negative main data values (provided that they are less negative than the value of *missdatlim*) represent Detection Limit values of Below-Detection-Limit observations. For details, see below.

convtests (This is a command, not a special variable). After the word convtests, there is a three-row table containing numerical values that specify convergence tests for the iteration. The iteration proceeds in three stages or *levels*, and each row corresponds to one such level. The idea behind the stages is the following: the first stage finds the correct region in space, the second stage converges quite near to the final solution, and the third stage homes in to the best possible values. The first column *deltaQ_test* specifies how small change of the Q value (sum-of-squares value, χ^2 value) is needed for assuming convergence. The second column, *consecut_steps*, defines that this small change of Q must occur on each of *consecut_steps* consecutive steps in order that convergence be assumed on the level in question. Note that this is just a test specified by the user: there is no guarantee that a true convergence has been realized when any given convergence test is met. It is essential that the user explores the convergence properties of his/her problem and uses such values in *convtests* that a true convergence is guaranteed whenever the conditions are met. A good sign of convergence is if the g^2 values (gradient length, squared) decrease by several orders of magnitude. In mid-iteration, the g^2 values typically fluctuate up and down so that there may be a very slow trend downward. A steep decrease after such plateau usually seems to indicate convergence.

In large tasks, where all Q values are large, *deltaQ_test* may need to be (much) larger than the values shown in the demonstration scripts, e.g. in *parafac.ini*.

The third column of the table contains maximum cumulative step counts that are allowed for each level. If the convergence tests are not met when the number of steps exceeds *max_cumul_count*, then the level in question is terminated anyhow although no convergence may be assumed. In this way, excessively long computations may be prevented.

cgresets i1, i2, k1, k2, k3, k4; (This is a command with six parameters). This command determines how often the Conjugate Gradient algorithm is restarted. The program contains a heuristic technique that attempts to restart the CG algorithm whenever the progress seems to be getting worse. However, it may be useful to influence this heuristic. The first two parameters *i1* and *i2* of *cgresets* specify the smallest and largest number of steps that are allowed in a CG sequence. Typical values are 8 and 80 for *i1* and *i2*. If the upper limit *i2* is too small, then exceptionally difficult tasks may not converge properly. The remaining four parameters modify the allowed lengths cyclically. The coefficients k1 to k4 multiply the allowed lengths, so that the first multiplier k1 is used on CG sequences 1, 5, 9, ..., the second (k2) on sequences 2, 6, 10, ... and so on. The default is k1=k2=k4=1, k3=2. In this way, every fourth CG sequence is allowed to be longer than the others. This appears to be useful for some problems.

rewgrate In rare cases, the iteration does not converge at all if reweighting is performed all at once, i.e. each new weight is applied immediately to the data in question. In such cases one may decrease the reweighting rate *rewgrate* to 0.5 or 0.3, say. This means that each change of weight is applied gradually. The convergence is slower but safer.

goodstart Normally ME-2 performs the first few steps cautiously, avoiding extreme weightings. If the run is continuing from a previously computed reasonably good solution, such steps may be harmful because they essentially reject the previous solution. Setting *goodstart=1;* causes that even the first steps are performed similarly as the other steps.

numtasks ME-2 is able to compute several tasks in one run, provided that they are controlled by a single .ini file. The variable *numtasks* defines how many tasks are to be computed. In order to make sure that the global minimum (globally best solution) be found, one should compute with 20 (say) different pseudorandom starting values. Then one sets *numtasks=20;* and at the end of the third part of the script, loads a fresh set of pseudorandom values to the free factor matrices. Default is (*numtasks=1;*), compute a single case.

selfcancel (The variable *selfcancel* is set by ME-2, not by the user). *selfcancel* is a measure of the degeneracy of the solution, defined as

$$selfcancel = 1 - \sqrt{\sum_{m=1}^{M_1} w_m \left(\sum_{k=1}^{K_m} t_{mk}\right)^2 \Big/ \sum_{m=1}^{M_1} w_m \left(\sum_{k=1}^{K_m} |t_{mk}|\right)^2} \quad (2)$$

$$= 1 - \sqrt{\sum_{m=1}^{M_1} w_m y_m^2 \Big/ \sum_{m=1}^{M_1} w_m \left(\sum_{k=1}^{K_m} |t_{mk}|\right)^2}. \quad (3)$$

Here, t_{mk} represents the value of the k'th term in equation m. The summation extends over all main equations but not over auxiliary equations. If there are no negative terms, then *selfcancel* is zero. Conversely, in degenerate cases, where some terms approach plus and minus infinity, *selfcancel* approaches unity. Implementation problems have caused the following restriction in the definition of the self-cancel index: If common subexpression (SE) factors are used, then *selfcancel*, as computed by ME-2, does not reflect such cancellation that occurs within any single SE factor.

The degeneracy is a well-known problem for 3-way problems but does not occur for the standard 2-way factor analytic model. Usually solutions with self-cancel above 0.8, say, are useless. They are mathematical artefacts that cannot be interpreted in terms of the physical problem. For this reason, one often wishes to interrupt the iteration if the self-cancel index exceeds a set limit. One may inspect the variable *selfcancel* in the fourth part of the script and execute the command *finish* if *selfcancel* exceeds a specified limit. By using the variable

degenlimit one may cause an automatic termination of the iteration: if the *selfcancel* value exceeds the value stored in the variable *degenlimit*, the current task is automatically terminated.

unimodtoler The unimodality or monotonicity (u/m) constraints in ME-2 are approximate. The variable *unimodtoler* specifies the order of magnitude of the allowed violation of u/m constraints (relative to the max value of the factor in question). Requiring very strict unimodality or monotonicity makes the convergence slow. If full convergence has not been achieved yet, one may allow that the interim result may contain more severe violations of u/m than allowable for the final result. Reasonable values are between 0.05 and 0.005, say.

4. Input information and computed results

4.1 The order of operations when running ME-2.

When ME-2 is started, it first reads the .ini file whose name was given on the command line. Then it compiles and executes the .ini file by parts. First, the first part of the script is compiled and executed. Then, all other parts of the script are compiled. A copy of the .ini file is written to the file ME2.log (see command $listing). This copy is provided with line numbers so that if an error message later reports a line number, one may go back to the .log file and see which line is causing the error. If there are syntactical errors, they will be found by now (except for errors in format codes). The first part of the script sets dimensions of the arrays and matrices and declares those arrays and matrices.

When the .ini file has been compiled, the second part of the script is executed. This is the most important part of the script. It contains definitions of all equations that together define the mathematical-statistical model. Also, all data that is needed during the iteration are read in the second part. After the second part, the iterative computation (the least-squares fitting of the model) is performed and the resulting values appear as factor matrix elements.

The fourth part of the script (*the callback part*) is called repeatedly during the iteration. The *callback part* may be empty, or it may contain commands that adjust the operations according to some special rules.

The third part of the script is executed after the iteration has finished. First, the third part must write the results that were computed. As the very minimum, one should write the values of factor matrices and the scaled residuals. Sometimes it may be useful to write SubExpression factor values or gradient values. When the results have been written, the third part may prepare for another computation. It may read in new data values, or it may load new pseudorandom values to factor matrices. The computations continue until the number of tasks (variable *numtasks*) have been computed. After each new iteration, the third part of the script is executed again. During each new iteration, the fourth part of the script is called repeatedly.

4.2 What information is computed by ME-2

The main result are the values of computed free factor elements f_i, which usually means the matrices AA, BB, etc. The gradient components (e.g. AA.fgrad[0,0]) give information about the quality of the solution: for all unconstrained free factor elements, the gradient component should be ”small”. For actively constrained free factor elements the gradient components may be significantly non-zero. For the usual case of a positively constrained factor element being equal to zero, the gradient component may be significantly positive but it should never be a large negative value.

The residuals (measured value minus the fitted value) may be output in two different scalings. Usually the standardized residuals are the most useful: in a good model, they are mostly between plus and minus two. Outliers are easily spotted by looking at the standardized residuals. A data analysis cannot be regarded as complete unless the residuals are inspected. (In a series of similar recurrent tasks, it may suffice to inspect only a few of the tasks.)

Two output items are useful for estimating the significance of the results: the Hessian matrix and the Jacobian. By default, the Hessian is written as a symmetric matrix. The Jacobian is written as a matlab-style sparse matrix so that it may be easily loaded into matlab. The usage of the Hessian and the Jacobian is discussed in [3]. So far, there is not much practical experience in estimating the confidence intervals of results by using the Hessian. It is safe to assume that the true confidence intervals are *not smaller* than predicted by the Hessian. However, they may be considerable larger in some cases. The Hessian and the Jacobian become extremely large if the number of free factor elements and the number of data values are large. Trying to compute them may easily use up all memory in your computer.

In PARAFAC-like tasks, it is possible to decrease the size of the Hessian matrix in the file by writing the Hessian in a packed arrangement. This arrangement takes advantage of the fact that the three diagonal "super-blocks" of the Hessian are in fact block-diagonal matrices in a pure PARAFAC model. In the packed arrangemet, the zeros of one such block-diagonal matrix are not written to the file at all. More details will be available later. See the example hessdemo.ini.

4.3 Initial values for factor elements

If a known starting point is available for the iteration, then the user should load these values into the free factor matrices AA, BB, etc. Otherwise, pseudorandom values must be loaded to these matrices. The program does not randomize factor elements by default.— The SE factors are never input, as their values depend on the free factor elements. However, it may be useful to write the final values of the subexpression (SE) factor elements to a file after the iteration has converged. Sometimes they are useful "derived quantities".—If matrices containing constant factors have been defined, then of course the script must load their values. Either, the values may be read from a file, or in some cases they may be computed by the script. The fact that constant factors are encoded with negative indices in the one-dimensional notation inside ME-2 does not concern the user of ME-2.

The key values corresponding to factor elements must also be specified in the script. These values are set by using the *dot notation*, e.g. AA.fkey[0,0]=0; specifies that all elements of the factor matrix AA are defined as constrained factor elements. The key values may be set by using special constants or by using numerical codes. The special constants are preferred because they will minimize human error. The different key value meanings are:

- $KEY_i = 1$ =nolimits. f_i is not constrained, it may have arbitrary positive and/or negative values.
- $KEY_i = 0$ =lolimit. $flow_i \leq f_i$
- $KEY_i = -1$ =lohilimits. $flow_i \leq f_i \leq fhigh_i$
- $KEY_i = -5$ =locked. f_i does not change at all from its original value
- $KEY_i = -6$ =masked. f_i is forced to be equal to zero

In previous versions of ME-2, the key value *masked* implied special optimization: terms containing masked factors were automatically removed from the equations. This made the model table smaller and the run faster. However, it has turned out that such optimization is harmful because it prevents the use of other optimization techniques. From version 0.99 on, masking does not imply automatic removal of terms from the model. Of course, the user may

omit terms from equations if it appears useful.

The lower limits $flow_i$ are by default zero but may be changed by the user. The upper limits $fhigh_i$ are by default $= +\infty$. They must be set by the user to meaningful values. Example: BB.flow[0,0]=–1; BB.fhigh[0,1]=10; BB.fhigh[0,2]=15;

When forming the Jacobian or Hessian matrices for error analysis, the locked and masked factor elements should not enter the matrix at all because they are in effect constants, not variables. This removal is not done by ME-2. Before analyzing J or H, the user should remove those columns of J, and those columns and rows of H, that correspond to locked or masked factor elements.

4.4 Techniques for specifying std-dev values for data and auxiliary data

Each equation needs four parameters (C_1, C_2, C_3, and *errormodel*) for determining its weight in the fit. The weight may also depend on the measured and the fitted values. The controlling parameter is the "errormodel code" or EM code. An EM code is needed for each equation (=for each data value). The code is usually one of the values –10 to –16 for the main data values. For auxiliary data, also many other values between –1 and –1000 are often used as EM codes.

Proper use of EM codes for the main equations contributes in making ME-2 an "expert system", i.e. a system which combines expert knowledge with automatic computations. The EM codes and the std-dev values are the means how the end user may communicate her expert knowledge to the program.

4.4.1 Meaning of EM codes

The values EM=–10 to EM=–16 have the same meanings as in PMF3, see PMF Users Guide. The values –5 and –6 are special to ME-2. The different EM values define the following computations of std-dev values. The symbols x_m and y_m denote the data value and the fitted value whose std-dev σ_m is to be computed.

–5 The std-dev σ_m is computed by the expression $C1 + C2\sqrt{sumabs} + C3\ sumabs$, where the value $sumabs$ is computed by summing the absolute values of terms contributing to x_i,

$$sumabs = \sum_{k=1}^{K_i} \left| \prod_{j \in \mathcal{J}_{ik}} f_j \right| \tag{4}$$

–6 This std-dev value is computed as per EM=–12

–10 Compute $\sigma_m = \sqrt{C1^2 + 0.5C3^2\ |y_m|\ |x_m + y_m|}$. Corresponds to lognormal distribution of errors.

–11 Compute $\sigma_m = \sqrt{C1 + \max(y_m, 0.0001)}$. This corresponds to Poisson-distributed data.

–12 Take $s = |x_m|$. Compute $\sigma_m = C1 + C2\sqrt{s} + C3\ s$.

–13 Take $s = |y_m|$. Compute $\sigma_m = C1 + C2\sqrt{s} + C3\ s$.

–14 Take $s = \max(|x_m|\ , |y_m|)$. Compute $\sigma_m = C1 + C2\sqrt{s} + C3\ s$.

–15 The EM code –15 indicates a missing value.

–16 The EM code –16 has an almost similar effect as a BDL value (see below). If the fitted value is above the data value (i.e. if the residual is negative), then the data value attempts to pull the fit down towards itself, as if EM=–12. If, however, the fitted value is below the data value, then this data point gets zero weight, i.e. the data value does not pull the fitted value up towards itself.

–17 The EM code –17 has an opposite effect than code –16: If the fitted value is above the

data value, then there is no down pull, the weight is zero. If, however, the fitted value is below the data value, then the data value pulls the fitted value up towards itself as if the errormodel code were –12. With codes EM=–16 and EM=–17, negative data values are used as given, without taking absolute values first. This is a difference in comparison to the BDL scheme.

–100<EM<–20 These EM values are used for specifying unimodality equations, see below. For an example, see the GEdemo.ini demonstration example.

–200<EM<–100 These EM codes are used with monotonically increasing factors. The usage is same as for unimodality EM values.

–300<EM<–200 These EM codes are used with monotonically decreasing factors. The usage is same as for unimodality EM values.

–1000<EM<–990 These EM values are used for specifying anti-dip equations. Details will be written later.

The special EM values for unimodality or anti-dip equations may only be used for auxiliary equations. Note that robust processing only influences main equations. The auxiliary equations are always processed in non-robust way.

4.4.2 Discussion of ErrorModel codes

The option EM=–14 is a good choice for environmental data. 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 last alternative EM=–14 should never be a bad choice, except that the option EM=–12 runs a little faster than the other ones.

Simulation experiments have shown that the lognormal code (EM=–10) is practically equivalent to the code EM=–14, except for extreme situations where the width of the lognormal distribution of errors exceeds 50% of the data values. Thus one seldom needs to use the lognormal code.

In using the lognormal code EM=–10, one assumes 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) = C3. It is further assumed that there is "measurement error" having std-dev=C1 in each measured value x_{ij}. Under these assumptions, the factors are determined so that the fitted values y_{ij} maximize the likelihood of x_{ij}. The coefficient C1 is intended for representing typical measurement error which may cause zero or negative data values to appear although the genuine lognormal distribution is always strictly positive. If there is no such "lab error" in the data, then C1 is not needed and one should set C1=0.

The codes EM=–16 and EM=–17 can be used for implementing auxiliary inequalities in an approximate way. As an example, assume that the sum of factor elements f_a and f_b should be non-negative. Then the equation $0 = f_a + f_b$ with EM code –17 causes that if the sum is negative, it is pulled up towards zero with the std-dev value assigned to the equation. Thus the std-dev value determines the tolerance of the equation: how negative the sum may become in the solution.

The term *Target Transformation Factor Analysis* (TTFA) has been used to denote such factor

analysis where preferred shapes or *target shapes* for some factors are *a priori* known. The code EM=–5 has been formulated especially for implementing target shapes in ME-2. The target shapes are defined as approximately known ratios of factor elements. (Example: for a marine aerosol factor, the ratios of concentrations of other elements vs. concentrations of Na might be specified.) Assume that it is known that the ratio of factor elements a and b would be $f_a : f_b = 10 : 4$. This could be expressed as $0 = f_c f_a - f_b$, where f_c is a constant factor, $f_c = 0.4$. Normally, one would like to express the uncertainty of such target information as a relative or percent value, e.g. as 10% or 1%. In this case, the value to be fitted is zero, thus it cannot act as a basis for relative uncertainty if using the standard EM codes –12 to –14. With C1=0 and C2=0, the EM code –5 gives the following std-dev: $\sigma = C3\ (|f_c f_a| + |-f_b|)$. Taking into account that $f_c f_a \approx f_b$, this code creates approximately the equation $0 = f_c f_a - f_b \pm 2\,C3\,f_b$. Thus the value $C3 = 0.05$ together with EM=–5 coresponds to 10% uncertainty in target shape equations.—Sometimes only the ratios of main element concentrations are known for a factor. Then one only specifies target shape equations for those main elements. The minor concentrations of the factor are free to assume any values that best fit the data.

4.4.3 Choosing std-dev for main equations.

The significance of the array $\boldsymbol{\sigma}$ (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 ME-2. Generally one should attempt to specify the std-dev values so that they indicate the agreement which is expected between the (bilinear, trilinear, etc.) model and the data array. A few unrealistically small std-dev values (e.g. smaller than the data values by a factor of 1000) 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 do not give satisfactory estimates for $\boldsymbol{\sigma}$. 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 the data in question. How is one to use ME-2 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}. 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).

4.4.4 Choosing std-dev for auxiliary equations.

In principle, there are two kinds of auxiliary equations: (1) those that are intended to be obeyed strictly, and (2) those that should influence the solution in a favourable way although they should not be obeyed exactly. For the first kind, one might like to specify extremely small std-dev. This has a numerical disadvantage, however. A small sigma creates a large singular value in the Jacobian. On the other hand, the convergence rate of the C-G algorithm depends on the condition number of the matrix. Thus creating large singular values is harmful for the convergence. It may be good to define a ”reasonable” std-dev for the first iteration stage(s), and let it decrease later by respecifying its std-dev coefficients in the fourth part of the script.

If too large values are specified as std-dev for the auxiliary equations of the second kind, then the equations have no influence. On the other hand, too small values create too much of influence. E.g., if the equations are intended to ”smooth” a vector (specifying that the differences of consecutive vector elements should be zero), then too small std-dev would delete all change in the vector and create a vector of a repeated constant value. This problem is familiar in the literature of deconvolution, there the problem is to determine the strength of ”regularization”. There are different recipes, but it appears that the question must be solved mostly by common sense.

Sometimes certain auxiliary equations are only needed during the first phases of the iteration, for guiding the algorithm to a correct path. The std-dev of such equations should be increased in the fourth part of the script after a suitable number of steps have been performed. In this way the influence of such equations is made weak or negligible in the final solution.

5. Techniques for improving the speed of ME-2

With large tasks, a ME-2 run may become so slow that one wishes to pay attention to improving the speed of computations. There are several possibilities for improving the speed of the iteration steps. Also, it is possible to enhance the convergence rate by using the block-diagonal preconditioning scheme, as described in next section.

The use of subexpression (SE) factors is demonstrated in the script *parafac.ini*. Using SE factors is recommended because it has two merits: it makes the computations faster and also, it decreases the size of the model table, thus enabling one to run larger tasks in a given computer memory size. The dimensions of a SE factor matrix should be arranged as shown in newer versions of parafac.ini: the second dimension should be the *number of factors*, usually denoted by *np* in the script. In order to minimize the risk of errors in the model, one should preferably verify the arrangement of SE factors by writing the script so that it contains both an SE and a non-SE arrangement (cf. parafac.ini). Both versions should converge similarly during the first 30 to 50 steps of the iteration. After that, numerical effects (rounding errors) usually cause that the iterations drift away from each other. The SE and non-SE runs may even converge to different local minima. (Running some tests in two different ways is also good for finding possible hidden errors in ME-2!) —In principle, using SE factors does not influence the rate of convergence: the number of iteration steps is approximately the same in SE and non-SE runs.

5.1 Coding with multiple term notation

In the basic .ini file arrangement, each term of an equation is defined separately. Within the innermost program loop, the terms are defined one by one. In an equation of the PARAFAC model, the first term is defined as $a_{i1}b_{j1}c_{k1}$, the next as $a_{i2}b_{j2}c_{k2}$, and so on until the last term that is defined as $a_{ip}b_{jp}c_{kp}$. Each definition is stored separately in the model table.

The sequence of PARAFAC terms, above, is far from arbitrary: in consecutive terms, the *second indices* of each factor element in the term are incremented by one unit from one term to the next. This is a regular pattern that occurs in many multilinear models. The multiple term notation implements such a regular sequence of terms: only one term is defined in the script, the first term of the sequence. Within this first term, there should be the command *”multiterm p;”* where p stands for the number of terms in the sequence. In a PARAFAC model, the number is usually denoted by *np*.

Only one term is coded in the model when the multiterm notation is used. Thus the size of model table becomes significantly smaller. Also, the program ME-2 executes a multiterm

faster than the equivalent sequence of ordinary terms. Thus using multiterms makes the run faster. The gain is typically between 10 and 40 percent.

So far, multiterm notation has not been implemented for SE equations. Also, models often contain equations where the first index of the factor elements are incremented from term to term. Normalization equations are the most common example. Such equations cannot be coded by using multiple term notation.

Sometimes factor elements are represented as a one-dimensional vector. Then multiterm notation applies to the *only* index of such vector. The rule is that the *only* index of a vector is equivalent to the *second* index of a matrix or an array.

One should preferably test a script that uses multiple term notation so that parallel runs are performed with and without multiterms. See the discussion about testing SE equations, above .

5.2 Coding with extrapolated equation notation

Usually, a multilinear model consists of regular sequences of equations. The *extrapolated equation* notation offers a compact coding of such a sequence. As shown in *me2scrip.txt* and in *parafac.ini*, the two first *parent equations* of the sequence are specified in full detail in the normal way. After that, the remaining *child equations* of the sequence are defined one by one in a compact notation: the parent equations are indicated, and the sequence number of the child equation in question is specified. The sequence number 3 corresponds to the first child equation that immediately follows the second parent equation in the regular sequence.

The child equations take significantly less memory space that the normally coded equations. Thus the reason for using extrapolated equations is that the model takes less memory space. If the model fits in available memory space without recourse to the extrapolated notation, then one should not use the extrapolated equations, because the decoding of the extrapolated notation takes extra time. The run is somewhat slower (by 10 to 30 percent, say) when the extrapolated notation is used.

One should preferably test a script that uses extrapolated equations so that parallel runs are performed with and without them. Exactly the same convergence should occur in both ways: there are no numerical differences between these two alternatives.

6. Miscellaneous details

6.1 Preconditioning

In the JCGS paper [3], only one preconditioning arrangement is described: the diagonal preconditioning that basically only takes care of different scalings of different factor elements. In the present version of ME-2 (beginning Oct. 1999), there are other variants, too.

There are four preconditioning modes, selected by the special variable *precmode*. If *precmode* $= -1$, the original diagonal preconditioning scheme is used, as described in the JCGS paper. This option may well be used for small runs. If *precmode* $= -2$, then a different diagonal scheme is used. May be slightly better than the original scheme. If *precmode* $= -3$, then no preconditioning is performed. Only useful for comparison experiments and testing.

The most efficient preconditioning is obtained by setting *precmode=k*, where k is an integer, k$>$1, typically k=5 to k=50. These codes select the block-diagonal preconditioning and limit the maximal effect of preconditioning to be less or equal to k.—The best value of k depends on such details as the rank of factor matrices. The best value for any specific task may only be

found by experiments.

The block-diagonal preconditioning is controlled by special *preconditioning codes*. Corresponding to each factor matrix, such as $AA[n1, np]$, there is a matrix of preconditioning codes that are accessible as $AA.fprecc[n1, np]$. For understanding the block-diagonal preconditioning, one has to consider the one-dimensional arrangement of factor elements as the vector $\mathbf{f}$. Denote the corresponding vector of preconditioning codes by the vector $\mathbf{p}$. The value of the code $p_n = \varphi > 1$ establishes a diagonal preconditioning block of size φ: The factor elements $f_n, f_{n+1}, \ldots, f_{n+\varphi-1}$ form a block so that preconditioning takes into account the mutual interactions of factor elements in this block. Different block sizes may be useful. The basic setup is so that the factor elements on one row of a factor matrix form one block. These elements are in consecutive locations in the vector $\mathbf{f}$. Thus, by setting

$$AA.fprecc[j1, 1] = np \qquad (j1 = 1, \ldots, n1), \tag{5}$$

one specifies that each one of the rows of matrix AA forms a separate preconditioning block (the dimensions of AA are n1×np).

Larger preconditioning blocks lead to better convergence. One may form a preconditioning block that encompasses an entire factor matrix. By setting $AA.fprecc[1, 1] = (n1 * np)$ one specifies that the whole of AA is one block. However, large blocks lead to slow computations. Thus the overall running time may increase although the number of iteration steps decreases when one specifies large preconditioning blocks. The ultimate arrangement is to specify a very large preconditioning code for the first element of the first-defined factor matrix. This would cause that all factor elements of the model form one single preconditioning block. Then the iteration would essentially proceed similarly as in the Gauss-Newton algorithm that is used by the program PMF3. With fewer than a few hundred factor elements, a reasonable convergence may be achieved. However, with thousands of factor elements, the run would be brought to a halt or perhaps lack of memory would prevent the iteration from even getting started.

In a Tucker3 model, one would probably have all elements of the core array form one single preconditioning block.

6.2 Unimodal and monotonous factors

In Chemometrics, unimodal and/or monotonous factors are quite usual. These apriori conditions may be implemented in a ME-2 script by using special ErrorModel codes that create dynamical weighting (see above). The script must contain a large set of auxiliary equations, each one equating two elements of a factor by setting e.g. $0 = AA[i-1, j] - AA[i, j]$ or $0 = AA[i-5, j] - AA[i, j]$. Usually it is good to have at least two kinds of equations, those that equate consecutive elements, and those that equate elements that are a little apart of each other. The EM codes for such equations should be chosen depending on the kind of effect desired: unimodality or increasing or decreasing monotonicity. For all u/m equations affecting one factor (= one column of a factor matrix), one should use one EM code number. For different factors, different code numbers must be used.

Two parameters control the unimodality/monotonicity (u/m) equations. (1) The special variable *unimodtol* is common to all equations and all factors. The program attempts to satisfy all u/m equations so that the proportional violation is at most equal to *unimodtol*. The proportionality is defined with respect to the largest element of the factor in question. Small values of *unimodtol* may lead to convergence problems. (2) In each u/m equation, the std-dev parameter C1 controls the strength of the equation. It appears that the convergence is more reliable if the C1 value is relatively large, perhaps $10 < C1 < 100$, meaning a weak equation. Only when attempting a final high-quality convergence, a strong equation with $C1 = 1$, say, seems

in place.

Examples of unimodality equations may be studied in GE_demo.ini and in ME2libr.txt. The original unimodality definition is not quite satisfactory: local optima are almost always encountered in more difficult problems, so that the user must help the program around the obstacles. A new variant of unimodality equations is available for testing by interested users. The new variant will not be caught in local optima as easily as the original one.

6.3 Robust analysis

If robust analysis is selected by the command *robust=1;* in the script, then large residuals cause that the corresponding equations are "weighted down", see [1]. For auxiliary equations, robust processing is not performed.

6.4 Missing and BDL (=ND) data

Missing data means such data points where no data value is available because of some mishap, such as bad weather, broken equipment, operator error, and so on. In contrast, below detection limit (BDL) data means values that cannot be reported because the concentration was too small for a reliable measurement to be performed. Alternative notation for such data is non-detected (ND). In this presentation, only the notation BDL will be used. For BDL values, a limit is (or should be) always reported with the understanding that it is believed that the true value is below this limit.

Techniques for missing and BDL data have been enhanced for ME-2 (and for PMFx). The current situation is simple: in the .ini file, a lower limit may be specified as the variable *missdatlim*. All data values that are more negative than *missdatlim* are interpreted as missing. The setting *bdlneg=1* specifies that negative main data values (provided that they are less negative than the value of *missdatlim*) represent Detection Limit values of Below-Detection-Limit observations (the absolute value of the main data value is used as the Detection Limit).

The special EM codes connected with missing and BDL values have been explained in the list of all EM codes.

6.5 Avoiding degenerate factorizations

It is well known that 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.

A warning of degeneracy may be obtained by inspecting the value of the special variable *selfcancel*. The values of *selfcancel* approach unity if the solution becomes degenerate. For a positively constrained model, *selfcancel* =0.

With ME-2, degenerate factorizations may be avoided by introducing regularization equations that try to minimize the sum-of-squares of the factor elements. Experiment with the std-dev values of such equations, in order to find the largest value (=weakest equation) that still prevents degeneracy. Use this largest value in order to minimize the distortion that is unavoidably caused by the regularization.

6.6 A summary of useful formatting codes

Some ME-2 users may not have easy access to a Fortran language handbook. For the bene-

fit of them, the most important formatting codes (*formats*) are explained below. The use of formats in ME-2 is confused by the fact that each input/output (I/O) list element is processed quite separately from the other elements. Thus each element starts to use the format from the beginning. In the language Fortran, consecutive I/O list elements will use consecutive elements of the format string. In ME-2, a many-element format is only meaningful if an array segment is (read or) written. Then consecutive array elements may use consecutive elements of the format. This will be needed only in very special situations because usually one wishes to write all array segment elements in the same format.

The *edit descriptors* contained between parentheses in a format are interpreted sequentially from left to right. Usually the descriptors are separated by commas. The edit descriptors fall in three classes: data, control, and character-string.

An example of a data edit descriptor: (F12.3) . This descriptor means that a decimal value is to be written in next 12 locations of the current ouput line so that exactly 3 digits are written after the period. Thus there is space for at most 8 digits, or 7 digits and a minus sign, in front of the period. The data edit descriptors may be preceded by a repeat count. The notation (3F12.3) is equivalent to (F12.3,F12.3,F12.3) .

Other data edit descriptors:

- I6 means that an integer value is to be written in 6 locations.

- E14.5 means that a decimal value is to be written in exponential notation, with 5 digits after the period in the mantissa, and using 14 locations for the whole presentation.
- The variant E14.5E1 further specifies that the exponent should be written as one digit only.

- The notations G14.5 and G14.5E1 may be used for any type of data value. They write the exponent if necessary, otherwise they create the same result as the corresponding F descriptors.
- A15 is an example of a descriptor for text data. This means that the 15 first characters of the text in question are written.

Character string edit descriptors contain (within quotes or double quotes) some text that is to be written as such. Example: the format string (”Final value:”,F10.5) will first write the text *Final value:* and then the value with 5 digits after the decimal point. In the ME-2 script, single quotes are used around the whole format string. Thus it is necessary to use double quotes around the character strings within the format strings (the mechanism for representing single quotes within single quotes has not been implemented in the script language, although it is used in the reading of data files).

Control edit descriptors. The descriptor 5X means that 5 blank locations are to be written. The descriptor / (slash) starts a new line (a new record). The effect of / comes in addition to the automatic new-line mechanism that causes skipping to next line whenever a format is begun, *except* when the variable *advance* has value 0. The descriptor : (Colon) terminates the interpretation of the format string if there are no more remaining values to be written from the current output item.

Special ME-2 edit descriptor TAB (the upper-case letters T, A, and B): This descriptor causes that the character ”horizontal tabulator” is written. This may perhaps be useful when transporting ME-2 results to a spreadsheet.

The example scripts will illustrate the use of formats.

7. ME-2 User's Checklist

This list contains reminders for using the program ME-2. The items are organized approximately in decreasing priority. Those questions are discussed first that are most basic and should be checked first, before considering the more difficult and rarely needed details. Before regarding your results ready for publication, you should check all questions in the checklist and have good answers to all of them. A student should discuss all the questions with his/her supervisor. Discussing the questions together is probably useful for both of them! In that way, the supervisor is able to better understand what the student is doing when performing a multilinear analysis.

7.1 Did you inspect all the "alert messages"?

The program writes in the file me2.log important remarks and warnings as "alert messages", framed by rows of asterisks, "**********". The alert messages contain a serial number. The final output of the program mentions how many alert messages have been generated. It is good practice to look at all of the alert messages. If you do not immediately understand some of them, try to find explanations in guide books or from other users! In most cases you should change your run so that the alert messages disappear. There are, however, some cases where an alert message cannot be avoided in a correct and reasonable run.

7.2 Did the run converge?

The .ini file contains convergence criteria for the three iteration levels. The third level is crucial for obtaining correct results. The first two levels mainly influence the rate of convergence, i.e. how many iteration steps are needed in order to achieve final convergence. When the program says "convergence achieved" it only means that the convergence criterion defined by you has been met. If this criterion was too loose, it means that there only was an apparent convergence, not a real one. It is sometimes necessary to experiment with different convergence criteria. If the results do not change significantly although many more steps are performed, then a true convergence may be assumed. If the run is interrupted because the maximum iteration count is exceeded, then there obviously is no guarantee of convergence.

For extremely large models, comprising thousands of data points, the default convergence criteria tend to be (much) too tight. Then you lose time if you do not increase the convergence limits. Whereas something like 0.01 may be a good convergence limit for small models, the largest runs might converge well with limits in the range between 10.0 and 1.0, say.

Make some experiments with different convergence limits. If the result does not change although you force the program to run several hundred extra steps, then probably a good convergence has been achieved. For easy problems, ME-2 typically needs a few hundred steps. For difficult ones, up to 2000 steps may be needed.

7.3 What about outliers in your data?

The final output of the program tells how many residuals exceeded the outlier limits (=4.0 by default). If there are such residuals, it may be better to use the robust mode (it is selected by setting *robust* =1). Write the matrices of scaled residuals to a file and try to understand why some residuals are so large. However, if the residuals are large because there are too few factors, or because the model is not correct, then it would not be right to try to correct the situation by choosing the robust mode.

7.4 Have you inspected the residuals?

In a thorough data analysis, one should inspect the residuals for different anomalies. Ideally, *the scaled residuals should appear random to the eye*. There should be a random pattern of positive and negative values, most of them between -2.0 and 2.0. Groups of mostly positive residuals, or mostly negative, or mostly very small (smaller than 0.3, say) should ideally not occur. If such features do occur, one should try to understand why. One possible reason is that too few factors have been used. Another is that the true situation does not fully obey the assumed mathematical model.

7.5 Is the computed Q value (the Chi-2 value) reasonable? Is there need to reconsider the std-dev values for X?

1. If the std-dev values for the data array are theoretically known, then it is possible to judge the quality of the fit based on the Q value. The value of Q should be approximately equal to the number of points in the data matrix minus the total number of elements in the factor matrices.

2. More often, the standard deviations for data points are not well known. Then the obtained Q value depends on the assumed std-dev values, as well as on the quality of the fit. It is reasonable to adjust the std-dev so that a correct value is obtained for Q. However, then one must not say that the fit is good because a correct Q was obtained! That would be circular reasoning!

3. If the scaled residuals are especially *large* for certain variables, then one may consider that perhaps the std-dev values for these variables have been specified too small. Then one may increase std-dev for these variables.

4. If the scaled residuals are especially *small* for one variable, then two explanations are possible:

4a. Too large std-dev have been specified. You may decrease them.

4b. The variable in question is explained by a unique factor, i.e. one single factor explains practically all variation of this variable. This situation may occur ”naturally”, so that it is not an error at all, if one source emits practically only one compound. However, this situation may also appear as the result of specifying too small std-dev values for a noisy variable. Then it is essential that the std-dev values are increased to a realistic level, see Paatero and Tapper, Environmetrics (1994).

7.6 Have you tested different numbers of factors?

It appears that no mathematical criteria are able to predict the ”correct” number of factors. In environmental work, there may be no ”correct” number at all, because nature is never fully described with any number of factors. Rather, the question is which number of factors gives the most useful solution. In this connection, one has to consider rotations, too. With certain numbers of factors, there may be more need of rotation before the result may be interpreted as source signatures, say.

7.7 Have you computed multiple results, starting from different pseudorandom starting points?

This question is crucial for three-way models, but also important for two ways. The factor analytic least squares models often contain local minima in addition to the global minimum. The program is not guaranteed to find the lowest minimum. Once it gets attracted by one (local) minimum, it cannot proceed to another one. By running several times, starting from different

starting points (different seed values), one may explore the different minima. It is recommended that you look at the computed factors corresponding to a few of the deepest local minima, in addition to the global minimum.

– If you have found out that there are no local minima when solving your problem with 4 factors, say, you should not automatically assume that the same be true for other numbers of factors, such as 3 or 5!

7.8 When computing multiple starts, did you remember to load new pseudorandom starting values to factor matrices?

If new pseudorandom values are not loaded into factor matrices, then the follow-up runs start from the good solution computed during the previous task. This situation will be visible so that all tasks except for the first one converge very quickly. Also, the results will be identical and the iteration will need the same number of steps for all tasks following the first two.—With random starts, the number of iteration steps will vary greatly from task to task and usually the Q values of different solutions will fluctuate a little or be entirely different.

The loading of new pseudorandom values is performed as the last task in part three of the script.

7.9 If negative values have been used for encoding missing data in the data matrix, did you also remember to set a suitable value to the special variable *missdatlim*?

If you did not, then the negative values will be used "at face value". This will result in large residuals, large Q, and generally distorted and useless results.

7.10 If negative values have been used for encoding BDL values in the data matrix, did you also remember to set *bdlneg=1*?

If you did not, then the negative values will be used "at face value". This will result in large residuals, large Q, and possibly distorted and useless results. If different negative values are used both for encoding missing values and for encoding Below-Detection-Limit values, then you should check that the value of *missdatlim* is suitable so that all missing values are more negative than *missdatlim*, while all BDL values must be less negative than *missdatlim*.

7.11 Is there rotational freedom in the results?

In 2-way models, there is always some rotational freedom *unless* non-negativity constraints and/or target shapes restrict the freedom. If all factor values are well above zero, then all constraints are inactive and non-negativity constraints do not reduce the rotational freedom. Explore the rotational situation by trying to pull down suitable elements in different factors! Observe the increase of Q when different factor elements are pulled down with different strengths.

Consider using target shape equations for removing rotational ambiguity. The errormodel code EM=–5 is especially suitable for such equations, see above. When publishing you results, it is essential that you report if some form of target shapes was used.

The Users Guide for PMF (Part 1) discusses different methods of dealing with rotational ambiguity. This discussion is useful background for working with ME-2, too. The technique suggested for target shape equations with PMF is unnecessarily complicated for ME-2, cf. the discussion of errormodel codes in this handbook. The technique of pulling down chosen factor elements is also more straightforward in ME-2 than in PMFx. The parameter *fpeak* for creating

a uniform rotational pressure is unique for PMF2. In ME-2, the same effect is achieved by the special down-pulling approach, as described in [1]. The file GE_demo.ini demonstrates how this technique is used with ME-2.

In 3-way models, rotational freedom is only present in special cases. The most usual reason for rotational freedom is that in one mode, two factors are (almost) proportional to each other. Example: assume that the two factor vectors C1 and C2 (= the first and second columns of the factor matrix C) are almost identical. Then the model logically reduces to a 2-way model for the factors represented by (A1, B1) and (A2, B2). Between these factors, there will be full rotational freedom unless non-negativity prevents one or both of the possible two rotations. If you cannot exclude the possibility of rotational freedom, you should carefully report your results so that the reader is not mislead to believing that the results are unique if they are not.

7.12 Did you obtain a ”degenerate” three-way (PARAFAC) solution?

This question is only relevant when non-negativity is not requested for two (or three) modes of a three-way (PARAFAC) model.

It sometimes happens that the solution of PARAFAC diverges towards large positive and negative values. The contributions of different factors cancel each other to a large extent. Such so called ”Degenerate solutions” are a property of the 3-way model itself, they are not caused by the special properties of a specific program. A degenerate solution is mathematically correct but not useful for the practical problem. Using increased regularization prevents degenerate solutions at the expense of slightly distorting the ”legal” solutions.

7.13 Are the computed factors written in a suitable format?

In environmental studies, concentrations of trace elements are very small. Using a fixed-point format, e.g. F10.4, may cause that such concentrations appear as zeros although the values are significantly non-zero. The safest solution is to use an exponential format, such as (G13.5E2).

7.14 Have you avoided extremely small and extremely large numerical values?

In the script, small values may be interpreted as zero unless you make the tolerance parameter *eps* smaller than its default value. In the multilinear engine, very small values are treated as zero. The limit is approximately 10^{-10}. On the other hand, large values may cause overflow in computations. For these reasons, different variables in the data matrix should be scaled so that their significant values are between 10^{-7} and 10^{7}, say (if practical, between 10^{-6} and 10^{6}). In practice, one should use micrograms or nanograms for trace element concentrations, and so on.

7.15 Did you activate preconditioning in ME-2?

Slow runs may be made significantly faster by activating preconditioning. The default recommendation is as follows: (1) for each factor matrix AA etc, set AA.fprecc[0,1]=np; where np is the number of factors = the number of elements in one row of AA. (2) set the special variable *precmode*=5;. Observe the change of convergence rate. Also try with larger values of *precmode*. Also try with different std-dev in normalization equations: too strong normalization equations tend to slow down the iteration and prevent the action of preconditioning. On the other hand, very weak normalization will also cause extremely slow convergence, although no normalization at all may sometimes converge very fast.

7.16 Do you feel that ME-2 is too slow? Converges too slowly?

Many details influence the convergence rate and the running speed of ME-2. This is a preliminary list that will be enhanced later on.

Convergence rate is influenced by the following: Values of "cgresets": the longest allowed interval between resets must be long enough if the problem is close to singular.

Preconditioning. Normalization equations influence convergence rate in different ways. Experiment with std-dev values in normalization equations and with the precmode variable. Sometimes dramatic improvements are possible so that all small factor matrices are included in one single preconditioning block.

Robust mode. If a large fraction of all measurements have so large residuals that they are handled as outliers then the convergence becomes slow. Usually, the reason is that all std-dev values have been specified too small.

For very large data sets: the convergence parameters should not be too tight. Experiment to see how much of change in factor elements corresponds to the change of 0.1 units in Q, say.

Computations may be made faster by the following: Use subexpressions. This is almost always useful!

Use multiterms whenever possible. (Requires that all factor matrices are arranged so that different columns represent different factors).

Use the optimized version of ME-2 (not available at times of program development).

Close unnecessary open windows if running under Microsoft Windows.

7.17 Are you using an obsolete version of ME-2?

From time to time, do check the FTP site ftp://rock.helsinki.fi/pub/misc/pmf/me2/ for newer versions of the programs. If downloading a new version, make sure you do not destroy the previous version before checking that the new one works with your data. Always first use the new "tst" version, which is more safe, although also more slow. After using the "tst" version for some time, you may download the corresponding "opt" version (if it exists), in order to gain a little in program speed (the "opt" version may need that you have at least Pentium Pro level of the Intel-style processor). The handbooks and guides keep changing, too. Read the file readme2.txt frequently, it tells what features have been changed in the programs.

8. Availability of the programs

The program ME-2 is available to interested colleagues. The .exe file of the program and documentation files can be downloaded from the FTP server *rock.helsinki.fi*, directory */pub/misc/pmf/me2*. Through www, the address is *ftp://rock.helsinki.fi/pub/misc/pmf/me2/*.

Running the program requires a personal license key. Free licenses for an evaluating period of six months are available from the author.

Several different script files are distributed with the program. It is not practical to keep different up-to-date versions of scripts for very many different multilinear models posted on the FTP computer. If you cannot find a script that you would need, contact the author before writing your own version. It is possible that a useful version already exists although none is present at the FTP site.

A library of useful subroutines is also distributed with the name ME2libr.txt. New additions to this library are welcome! Inspect the library, it probably already contains useful bits and pieces, to be used when building your own scripts.

9. Disclaimer

The program ME-2 and the auxiliary files and documents 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 program ME-2 and the scripts and other auxiliary files are licensed on the condition that the end user in all circumstances bears all risk of all possible damages and losses influencing his/her computer equipment, data files, loss of work, loss of business, etc., related to the use or attempted use of the program and the auxiliary files.

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!

References

P. Paatero, Least squares formulation of robust non-negative factor analysis, *Chemometrics Intell. Lab. Syst.* **37** (1997) 23-35.

P. Paatero, P. Paatero, A weighted non-negative least squares algorithm for three-way 'PARAFAC' factor analysis. *Chemometrics Intell. Lab. Syst.* **38** (1997) 223-242.

P. 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, pp 854-888.