Contents

effi: environment for fitting

H Spiering, Institut für Anorganische Chemie und Analytische Chemie,
Johannes Gutenberg-Universität Mainz

November 16, 2010

1 Introduction

The structure of the theory which can be handled by **effi** is demonstrated by the structure of a complicated Mossbauer experiment, which may look as in Fig.1
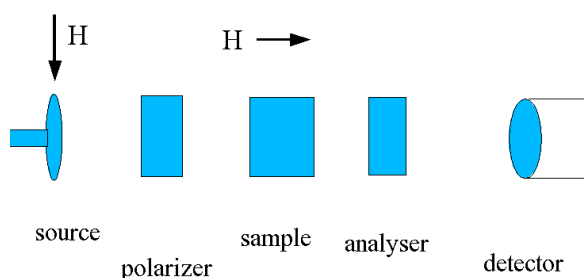

Figure 1: Not a typical Mossbauer experimental setup.

Each part may require different theories. The source in an emission experiment has different charge states (2+,3+,even 1+) of the Fe atom and different spin states which may change during the lifetime of the nucleus. Each situation uses a different theory for the calculation of the emitted radiation. The polarizer, sample, and analyzer have iron sites of different type and require also different theories for the calculation of their scattering amplitudes. Even the detector window typically contains a small amount of iron which in case of low iron concentration in the sample produces non negligible effects in the Mossbauer spectrum. Not shown in Fig.1 are apertures collimating the γ-beam from source to detector. The geometry, distances and apertures, effects the baseline and also line shapes of the Mossbauer spectrum.

A simple program would handle such a problem with an large input file (template) containing a large number of logical flags and positions for all constants and parameters for the most general problem to be calculated. A simultaneous fit of several measurements on the same sample under different conditions (experimental setups) would require large template files difficult to handle and may become complex. Simultaneous fitting means to correlate parameters of the different measurement such that an extra template somehow connecting the templates of the different measurements complicates the set up problem even more.
effi aims at an environment for fitting which simplifies these problems and allows not only for handling different measurements of the same type or method but also for simultaneously fitting

measurements requiring theories even of different structure like Mossbauer, synchrotron and neutron scattering, magnetic susceptibility data, perturbed angular correlation, NMR, etc.

The development of **effi** [1] started more than 20 years ago from the Mossbauer program called MOSFUN [2] which already used ideas realized in the Mossbauer program SIRIUS [3]. The structure was changed step by step in order to get the different parts and routines of the code more and more independent. To that time memory and frequency of the computers (even main frames) imposed many restrictions on the code. The programming technique, developed to that time, determined the code even later on when memory and computing time allowed for a lot of 'freedom' in writing code. The program was used in all stages of development mainly for fitting Mossbauer spectra and later synchrotron nuclear scattering. On one hand the usage of the program provided continuous testing on the other hand radical changes were not well accepted. This way the structure has developed as a compromise between technical requirements and those of users insisting on structures and patterns of data and parameter files. A change of the outcome,the present **effi**, which can be considered as a dead-end street of an evolution process, would be a task beyond the authors capabilities and the time put at his disposal. Therefore it was decided to build a new and 'modern' **effi** from scratch in the frame of the DYNASYNC project at the research institute KFKI in Budapest.. Modern also means an up-to-date user interface avoiding typed commands but clicking the mouse. The outcome after 3 years work of Szilard Sajti is now renamed to FitSuite. The theories implemented in **effi** were taken over to FitSuite. However, it does not yet have all features of the original **effi**. The 'to do list' of the user manual of FitSuite contains many features which are realized in **effi**. But FitSuite follows a more general concept without restrictions into which **effi** developed by its history. Whether these restrictions are of any concern for future implementations of theories cannot be estimated.

The aim of FitSuite/**effi** is also to involve a community into the project by contributing theories. Therefore in case of **effi** the distribution follows a procedure which leaves to the user to compile the source code of the theories. The first contact to **effi** is the source code written mainly in fortran77 and the plot routines for X-Window in C language. Only the **effi** part is hidden by the binary code and collected in an archive (static library named areffi.a). At right time in the future (to the authors decision) this source code will be also available.

In order to be able to contribute the interface to 'effi' has to be described. For long such a description had not been attempted because requirements of theories like synchrotron stroboscopic scattering on multilayers lead again and again to changes and extensions. The following description of the interface was written down in order to give Szilard Sajti an insight of **effi** when he started. FitSuite is written in C++, which educates the programmer to think of quite different structures than realized in **effi**. 'Old' programmer grown up with Fortran will have problems to get familiar with FitSuite while young researchers educated with C++ will prefer it.

The interface has now been described to some extent: Reading matter for developers 8.

2 General ideas

2.1 Theories and sub-theories

The theory for a measurement has a general part and calls subroutines which may be also called for other type of experiments. In the example of a Mossbauer experiment of Fig.1 a global (frame) theory T handles the "properties" of source, ..., detector calculated in 'sub-theories'.

The frame theory specifies already constants and parameters especially if the geometry of the experiment is taken into account. Sub-theories (subroutines) $\mathbf{T}_i$ are called for each nuclear site.

Source: Choose the number of sites n_s

In a loop over n_s sites one of the sub-theories

$\mathbf{T}_i$ offered by **effi** have to be chosen.

The polarizer...detector play similar roles, they are absorbers. The number of absorbers are chosen as a layer number and for each layer the procedure is the same as for the source.

These sub-theories may call further subroutines which are selected when defining the sub-theory $\mathbf{T}_i$. A typical example is the calculation of the absorption cross section of the nucleus by the nuclear Spin-Hamiltonian. The nuclear Spin-Hamiltonian (theory $\mathbf{T}_i$) depends on hyperfine fields V_{ik}, H_i in the sample.

The following list gives three different choices for the parameters to be fitted:

- $\mathbf{T}_i$ = nuclear Spin-Hamiltonian (V_{ik}, H_i)
  V_{ik}, H_i are parameters inserted in the parameter list.

- $\mathbf{T}_{i1}$=electronik Spin-Hamiltonian (parameter D, E) from which the magnetic hyperfine field H_i is calculated.
  Theory $\mathbf{T}_i$ calls theory $\mathbf{T}_{i1}$ and D, E, H_{ext}, V_{ik} are inserted in the parameter list.

- $\mathbf{T}_{i2}$=ligand field Hamiltonian (parameter E_i) calculates H_i and V_{ik}.
  Theory $\mathbf{T}_{i1}$ calls theory $\mathbf{T}_{i2}$ and E_i, H_{ext} are inserted in the parameter list.

This way any theory calling $\mathbf{T}_i$ automatically adopts all sub-theories $\mathbf{T}_{i1}$, $\mathbf{T}_{i2}$, ... (this example is not implemented yet)

2.2   Constants

Constants cannot be fitted but their values can be changed. **effi** lists the constants and allows to alter their values. However, there are constants which describe properties of the data sets, like the channel number or the transducer mode (sinus/linear in case of a Mossbauer experiment), which are typically never changed. Those constants should not lengthen the list. Therefore two types of constants are handled:

Constants which value are read from the header of the data file and later on are not directly accessible in the program (missing in the list of the constants) and constants in the constant list. The developer decides for the type of the constant. **effi** leaves the option to the user to read from the data file and not to include in the list of constants.

Since the theory can be simulated without having data, constants expected from the header of the data file get default values defined by the developer. **effi** provides to alter them when calling the theory.

2.3   Parameters

Parameters are fitted that means changed by **effi** at each iteration step. They can be excluded/included from a fit (fix/free) by the user. There are also parameters which are never changed if they have been precisely measured. Typical example is the background fraction of a Mossbauer spectrum or the Lamb Mossbauer factor of the source. Therefore there are also two types of parameters. The developer can provide the option " to be read from the header of the data file" in order to shorten the list of parameters.

The developer may give another option for parameters, such that the user has the choice to include them in the list or strike them off completely. An example is a Hamiltonian for hyperfine

interaction providing parameters of the electric field gradient and magnetic field. If one of the fields is zero all the time it is useful to ban it from the parameter list. In that case the parameters are put to zero by **effi** independent of their default value.

2.4 Transformation matrices and variables

The transformation matrices already introduced 30 years ago in the Mossbauer fit program SIRIUS are the essential part of **effi**. These matrices enable to correlate fit parameters by the user (and not by programming), a necessary condition for arranging simultaneous fits of several data sets.

If for example data sets fitted together measured with different external conditions (like magnetic fields), the parameters, which are independent of the external field can be correlated, this way reducing the number of independent fit parameters.

It is useful to differentiate between the quantities seen by the fit routines, which are parameters, and the quantities the theories depend on, here called variables and constants.

The parameters or a subset of them are fitted. They are mapped onto variables by a transformation matrix. Three simple examples are shown in the tables 1,2, and 3.

Table 1: Three parameters p_i are mapped onto three variables v_i by a unit matrix.

	p_i	p_{i+1}	p_{i+2}
v_i	1	0	0
v_{i+1}	0	1	0
v_{i+2}	0	0	1

Table 2: Two parameters, the quadrupole splitting ΔEq and the Isomer-shift IS, are mapped onto the line positions $position_{1,2}$.

	ΔEq	IS
$position_1$	$-\frac{1}{2}$	1
$position_2$	$\frac{1}{2}$	1

Table 3: Two parameters p_i are mapped onto three variables v_i by a 3x2 matrix.

	p_i	p_{i+1}
v_i	1	0
v_{i+1}	0	1
v_{i+2}	0	1

The developer of the theory defines the name of transformation matrices and parameters and variables belonging to. **effi** allows to redefine the whole set of transformation matrices of all theories called for a simultaneous fit. An essential task when programming **effi** was the handling of the transformation matrices according to the following commands:

- cp :correlate two parameters (pi,pk)

- dp :de-correlate a parameter pi → pi,pi+1

- ip :insert product of two parameters pi*pk

- cm :combine two matrices i,k

- sm :split of a matrix in two matrices

- ch :change of names and values

- pv :permute sequence of parameters and variables

- pm :permute the order of matrices in the list

- rm :remove matrices of zero dimensions

The **product of two parameters** $p_i * p_k$ can also be added to the variables, which means an extension of the linear form to second order terms. This way ratios of parameters can be fitted.
Identifying p_2 with y and p_1/p_2 with x in table 4, then again we have $v_i = p_i$, but p_2 and p_1/p_2

Table 4: Two parameters x,y and their product x*y mapped onto two variables

	x	y	$x * y$
v_1	0	0	1
v_2	0	1	0

as fit parameters.

Transformation matrices of equal name are automatically merged to a larger matrix when calling the same theory several times or even different theories fitting different type of data sets.

Here and everywhere the aim has been perused to have the parameter and constant list as short as possible. Reading two theories with constants or parameters of the same name, they can be marked by a # and then will be correlated automatically and appear only once in the constant/parameter list. The transformation matrices of the parameters are changed correspondingly.

3   Header of the data file

effi allows for 6 lines of information about the data. The first line is just an arbitrary comment. The second line starts with the fortran format of the data followed by letters/abbreviations which tell about the meaning of the columns of the data (x,y,errx,erry - z,errz not yet implemented). Standard Mossbauer spectra have only y-values (x is the channel number). If there is only one type (say y), then the data can be written in several columns defined by the Fortran format.
The third line needs the number of data to read. The name of this number is defined by the theory, in case of the Mossbauer routine the name is nu_channels, typically nu_channels=1024. **effi** does not look for the EOF. This line is inspected for other names defining values. All names are defined by the theory for which data are read in by **effi**. Again in case of Mossbauer with standard geometry **effi** looks for the following names:
channels_vmax, drive_mode, f_source, bg_fraction, geo, v_max, channel_v0, vmax2, vphase2, vmax3,

vphase3
If cosine smearing is taken into account further constants/parameters can be read from the data file:
N_om_samp, sc-aperture, Dabsorber,sc-absorber

The meaning of these constants and parameters are described in the help files. Actually it is the users decision, which of this values are expected from **effi** to be found on the data file, with one exception, namely the number of data, here nu_channels. If a value is found, which was not expected to be there, **effi** writes a warning:
*warning** v_max not accepted
*warning** channel_v0 not accepted
Up to 4 lines are analyzed for definitions of values.

The interpretation of the header requires a theory. The information contained in the header is typical knowledge about experimental conditions (like the bg_fraction of the Mossbauer gamma spectrum) which belong to each data set and this way is automatically taken into account by the theory.
If no data are read and only theoretical curves are simulated, the default vales from the frstr_...f files are taken. They are shown in the printed list calling the command {te} where the values can be interactively altered.

4   Parameter file .par

The parameter file .par stores all information of a project, the theories of all data sets, the variables and constants including their positions in the common blocks, a list of matrices, the transformation matrices, and the parameters including their boundaries and step-width for numerical differentiation.
The name .par is misleading as the list of parameters is the smaller part of this file. To the time of MOSFUN this name was justified. Now .project would be more adequate, but a change to another extension would require a change of the default .par everywhere in **effi**.
When reading a .par file **effi** looks into the object code of the Fortran files frstr...f and thstr...f for further information of the announced theories. As the theories are recognized by name, the complete string of the names have to match. An extra blank or letter destroys the .par file and leads to error messages for which typically no information has been written in the help file.
Most of the content of the .par file is self explaining. Few remarks shall help to get an idea of the structure.
The difference between variables and parameters is obvious. There is a parameter list at the end of the file and their names appear only there. The name variables appear two times, first together with the theory name they belong to and the second time in the left column of the transformation matrices. Variables have no values assigned to (**effi** does this by the transformation matrix and the parameter values), the number in brackets are the position of the variable in the common block. The type of common block is given by the level the theory is called from. The levels are indicated by horizontally broken lines in the parameter file:

---

level 0 *global*
/common/avstrf/

     device(i)=k ————————— ——————————————

level 1 *device*
/common/avfr1trf/

            module=lm ——————————————

level 2 *module*
/common/avfr2trf/

                component=jn —————————

level 3 *component*
/common/avsstrf/

The constants are printed also with their position in the common block (same structure but instead of av.. they are named ac..) in brackets and in addition their actual value. Constants do not have a transformation matrix, a fact which lead to another solution for setting up correlations. Inspection of one of the examples (effi/project/classic_Mossbauer/biotit/blaum19/blaum19.par) the sign '#' as the first character of the names of constants is found many times. In this example only 20 constants are listed by the command {co a} instead of more then 100.
effi correlates all constants of the same name with a # as first letter. Constants from the handbook automatically have got the #. The other # are added by the user.

Automatic correlation of parameters is also possible if reading several sets by the command {rs}. The first character of the parameter has been changed in the .par file by an editor to #. Reading several times the same set - the case of data sets of the same type of experiment but different conditions (temperature, external field, ...) - the parameters are correlated and the transformation matrices are build. If .par files of different type of experiments are used, then the user has to comply with the conditions of equal name of the parameters and the transformation matrix they belong to (see section **Building correlations** 6).

5   Names

Names are not only names but carry also some information. The information is separated from the freely chosen part by underscores. The number of underscores varies, it is adjusted such a way that a list of names is well aligned. The length of the string of names is 24 characters. Compared with length of names of MOSFUN with 6 characters for the LSI version, 12 character for the TOS operating system (ATARI) and VMS system (DEC alpha) the length of 24 characters looked very comfortable. Now there is the wish to double the length, but it means a pretty effort with the chance of many mistakes in the code. Since the names have these first characters for structural in formation followed by underscores there are effectively only 16 characters left for the name. That is the length for the names of transformation matrices which originally had no prefix.

        1st___name,..., nnst___name,...,32st___name

The name starts with one or two digits, the number of the set. The next two characters are defined by the theory: (st=sp) for spectrum (Mossbauer), (st=sc) for scatterer (synchrotron radiation, neutron), ... At the device level, two letters naming the device are offered by the structure routines

frstr␣...f: (sc) for source and (ab) for absorber (Mossbauer). The dimensions of the arrays allow up to maxs=32 sets.

nst1␣name,   nsti␣name

If theories are called by subroutine get␣theory() on the set level (0), the digit following (st) count the depths of recursive calls of get␣theory() such that $i \leq 8$.

nstsm␣name

If a device has only one module a two characters name of the module is not introduced, so that the following two characters stand for the component (site) number. (s) for site and one letter for the number i=1,...,9,A,...,Z,a,...,z,...

nstLYi␣name,   nstLYsm␣name

If a device has more than one module **effi** offers names defined in the structure routines frstr␣...f: (po) for polarizer, (an) for analyzer (Mossbauer theory) or just (mi, i=1,..). As above on level (0) constants/parameters introduced by subroutine get␣theory() are counted with $i \leq 8$. Components of the module (si) are the next two characters.

The components are continuously counted (not beginning with 1,.. for each module).
Names of constants taken from the handbook start with the # (see section 4): #hdbik␣name

Names of transformation matrices have got a two characters prefix and an underscore. The two characters have been introduced to suppress combinations of transformation matrices of equal name which is automatically done by **effi**. There is, for example, the matrix sc␣isomer␣shift and ab␣isomer␣shift for source and absorber. Without the prefix the parameters belonging to the two matrices isomer␣shift (the same name for absorber and source given by the subroutine thstr␣...f) are in the one matrix: isomer␣shift. With the prefix 2+1 characters, only 15 characters are left for the name (one of the bottlenecks of **effi**).

6   Building correlations

Starting with one set correlations are build up by the command {tr} and the commands therein (see 2.4). Transformation matrices as introduced by the set up of a theory model (command {th}) belong to a module/component. For example, the choice of Euler angles introduces a 3x3 unit matrix mapping the parameter (phi,theta,psi) on to the variables (usually having the same name). Correlations inside such a matrix will never rise problems, which will be described below.
By the command {cb} matrices belonging to different modules/components can be build and this way correlations between variables of different modules/components introduced by correlating the parameters.
A complex correlation scheme involving many variables of modules and components may be the requirement of the theory model. Such a single set cannot be easily extended to a number of sets for a simultaneous fit of several measurements.

What does **effi** offer in such a situation?

If the number of transformation matrices is less than 1/2 of the maximal number maxco (64 at present) the command {un} could be used to unify two .par files (the original one and a copied one with another name). Afterwards the identical matrices can be combined. The command { tr cb} offers the further command to combine all matrices of the same name. In the case of unification of two .par files all matrices have the same name such that this command halves the number of matrices. Having the parameters of different sets in the same transformation matrix further correlations between the sets can be introduced. To do this a command is offered calling {tr cp}, which correlates the parameters of all matrices if they are marked by a '#' instead of the number of the set (see 4). A '#' can be edited directly in the second .par file called by the unification command {un}. This way the .par file may be added several times (for several sets modeling the same experiment with minor changes of external conditions).

Another procedure (realized in **effi** from the beginning) starts with the first set in the original set up (matrices are unit matrices or of some other shapes, which **effi** can also handle- see below). The .par file of this theory model can be read in several times by the command {rs }. Correlations are automatically introduced for parameters with the first character # for the names of constants and parameters (see 4). Afterwards the correlations between variables of different moduless or components inside the sets (and again between the sets) may be introduced.

If a new measurement has been made so that a new set has to be added, this can be done with the original .par file (of course also with the command {un}). Still parameters are automatically correlated if the names of the parameters (with #) and the transformation matrix they belong to are the same. The correlations which are already there will not be destroyed.

A set with correlations can also be added this way if the transformation matrices are of special type:
There are only correlations between variables belonging to the same component. Correlations between variables of different components are allowed if the matrix has a shape shown in the following table 5.

Table 5: Two parameters p_i are mapped onto three variables v_i^k belonging to different components k and k+1.

	p_i	p_{i+1}
v_i^k	a	0
v_{i+1}^{k+1}	b	0
v_{i+2}^{k+2}	0	1

A matrix of the type of table 6 is misunderstood - **effi** introduces additional parameters and other nonsense. There is a specialty for matrices of more parameter than variables which are build by the command {ip} introducing a product of two parameters (see section 2 the matrix 4). This matrix in this shape will not be properly build in. The two zero in the first column cause the next two 'parameter' y and $x * y$ to be omitted.
The matrix will be accepted if it has the following shape:

Table 6: Two parameters p_i are mapped onto three variables v_i^k belonging to different components k and k+1.

	p_i	p_{i+1}
v_i^k	a	0
v_{i+1}^{k+1}	b	c
v_{i+2}^{k+2}	0	1

Table 7: Two parameters x,y and their product x*y mapped onto two variables

	x	y	$x * y$
v_1	0	0	1
v_2	0	1	0

There are two values different from zero in the last row. If c=0 the product $x * y$ as 'parameter' will be missing in the transformation matrix and the parameters listing as well. After building up the total number of sets the transformation matrix elements are conveniently changed directly in the .par file by an editor.

7 Helpfiles

Any time **effi** is waiting for input the user may type {he} and should get a help. If there is no helpfile found by **effi** the user get the information of the name of the helpfile expected. The name is constructed from the name of the theory assigned to the string 'property'. If the file is created but still empty **effi** tells the user which entry is expected for information. This entry may be copied to the file. All lines following this entry are written to the terminal by the {he} command. Inside the list of constants/parameters help is invoked by {he pi},{he ci}.

effi carries an integer 'info' to the output of subroutines. If info > 0 a message of maximal 6 lines is printed. typically it is an error message inserted in the program by the developer to control if constants or parameter have been chosen by the user out of some range. The first string points to an entry in an user defined help file. The next line contains text which is printed on the output screen and the last must be empty (recognition of the end of infotext strings).
info=1
infotext(1)='theory/mossbauer/topic/subtopic'
infotext(2)='message line1'
infotext(3)='message line2'
...

Table 8: Two parameters x,y and their product x*y mapped onto two variables. This shape is correctly read in by the command {rs}

	x	y	$x * y$
v_1	a	0	0
v_2	0	b	c

infotext(8)=' '

The number info is also printed. The developer may make use of this integer number (print some value like channel number etc.).

8   Reading matter for the developer

8.1   Routines defining the structure of theory trees

The structure of a theory tree is defined by one subroutine
>   *frstr_name(l_nfrtheo,lstruct1,lstruct2,cmch,nhelpentry)*

and an in effect unlimited number of subroutines
>   *thstr_name(l_jth,cmch,nhelpentry).*

The subroutines named *thstr_name(l_jth,cmch,nhelpentry)* are collected into files named *str_method.f* where str stands for structure and method for mossbauer, neutron, stroboscopy, etc.
The integer l_nfrtheo is an input parameter which is defined by the selection of a theory from a list defined in a *thstr_name(l_jth,cmch,nhelpentry)* with a length of l_jth (output parameter). The loop numbers lstruct1 and lstruct2 and the string 'cmch' are input parameters.The string 'nhelpentry' is an output parameter.

The routine *frstr_name()* defines a type of theory like Mossbauer, Neutron scattering, etc and may define branches to theories like Mossbauer transmission, Mossbauer reemission, etc, which in turn may have again branches defined by routines *thstr_name()*.

The number lstruct1 counts the device (maximum number of types is 3 - Mossbauer:source, scatterer, detector) and lstruct2 up to 192 modules (in case of scatterer the modules are typically layers) of the device.
By the input string cmch='name', 'hdbk' **effi** looks for the name of the theory calling the subroutine *getfrstring()*. The output string 'nhelpentry' is constructed by **effi** and used to trace the theory tree in order to find the helpfile for the theory with its constants and variables.
At each level defined by lstruct1 (devices) and lstruct2 (modules) constants (also from the handbook: cmch='hdbk') parameters and variables are defined. The menu of **effi** for the selection of theories is patterned according to the specialties of the theory in question. The text string needed for the menue are also defined for each level.

To each subroutine *frstr_name()* corresponds a frame theory
>   *subroutine fr_name(cmch,iss,info)* 9.2

which may cover several cases of similar structures (Mossbauer transmission, Mossbauer reemission, etc). As the different cases may require different constants, variables, parameters and text strings the number l_nfrtheo counts the cases. A name for each case is defined in *fr_name(cmch,iss,info)*. So the fork of cases for the two routines *fr_name()* and *frstr_name()* correspond to each other.

8.1.1   The routine frstr_name

The structure of the code is exemplified for the Mossbauer routine with only a single case: l_nfrtheo=1.

```
subroutine frstr_name(l_nfrtheo,lstruct1, lstruct2,cmch,nhelpentry)
include 'comglb.for'
include 'comstr.for'
include 'comcvm.for'
```

Names are taken over by **effi**. The helpfile Mossbauer_spectrum.help is expected by **effi**. The name of this theory and three device names are defined.

```
if(cmch.eq.'name') then
 nhelpentry ='Mossbauer spectrum'
 ntheory(1)='Mossbauer radioactive source'

 n_device=3
 ndevice(1)='source'
 ndevice(2)='absorber/scatterer'
 ndevice(3)='detector'
 return
 end if
```

The input string 'cmch' has two further values: hdbk and empt. In both cases the 8 characters of chth are defined

```
chth='fr'//nhelpentry(1:6)
```

which are added to the strings defining the constants, variables, correlation matrices and parameters. The information contained in chth is needed for the entry points in helpfiles.

lstruct1 runs from 0 to mstruct1 defined by the user (e.g. answering 'number of incoherent parts of the mosaic absorber'). The entry number_device(lstruct1=0) jumps to the section of global definitions. The entries number_device(lstruct1=1,2,..) to the sections for the devices.

The standard case assumes that each device is called only once. There may be the necessity to call a device several times (e.g. incoherent parts of the absorber/scatterer). The default values of the strings 'strcmt()' shown in Tab. 8.1.4 in section 8.1.4 prompts **effi** to ask for 'mdevf' (device factor) the number of devices of number 'm_device' (device number 'm_device' > 0 is a multiple device), defined by the string 'strcmt()'.

```
if(lstruct1.eq.0)then
```

The common blocks common/acstrf/ and common/avstrf/ for constants and variables are defined.

effi first asks for handbook constants. If no handbook constants are included at this point the string 'nstruct' has to be filled by blanks (nstruct=' '), otherwise the string contains the name of the handbook, the list entries (list=1,3,...) taken from the handbook and the string chth. In case of nucleus.hdb the entry 1 looks as

list1:thickness weight (transmission)
uthick_w *f_factor* mg(Fe)/cm**2 =teff (nat. abundance of Fe)
#sigma_0=2.56*10**(-18)cm**2, abundance=0.0214,atomic_weight=55.85
Fe_uthick_w = 0.589 1 (name, value, position in the common block common/acstrf/)

```
if(cmch.eq.'hdbk') then
 nstruct='handbook=nucleus.hdb    list=1    chth='//chth
```

```
  return
  end if

  call conmatpar('coll',jc,jm,jp,jv,chth)
```

Between the routines *conmatpar()* the strings

strcmt()	containing comments to the structure and default values of the number of devices and names of the set, modules and components 8.1.4.
thco()	name of constants, position in the common block and default value 8.1.5
ntrmt()	name and dimension of a transformation matrix 8.1.6
ctrmt()	description of the transformation matrix 8.1.6
thpa()	name of parameter and its default value 8.1.6
thvr()	name of variables, position in the common block and default value 8.1.6

are defined. A detailed description of the strings are given later.

```
  call conmatpar('aaaa',jc,jm,jp,jv,chth)   (aaaa.ne.coll but arbitrary
                                              aaaa=dev0 tells where we are inside the code)
  return
  end if      ! of lstruct1.eq.0

  goto(100,200,300)number_device(lstruct1)
```
The integer function *number_device(lstruct1)* calculates the device number (1, 2, 3). if no device is called more than once, the device number is just lstruct1.

The structure for the 3 branch addresses (100,200,300) of the devices (here source, absorber, detector) is the same.

```
100 if(lstruct2.gt.0) goto 110
  common/avfr1trf/ and common/avfr1trf/

  if(cmch.eq.'hdbk') then
   nstruct=' '
   return
  end if
  call conmatpar('coll',jc,jm,jp,jv,chth)
  ...
  call conmatpar('dev1',jc,jm,jp,jv,chth)
  return

110 continue
  modules (layers) of device source common/acfr2trf/ and common/avfr2trf/

  if(cmch.eq.'hdbk') then
   nstruct=' '
   return
  end if
  call conmatpar('coll',jc,jm,jp,jv,chth)
  ...
  call conmatpar('layn',jc,jm,jp,jv,chth)
  return
```

```
200 if(lstruct2.gt.0) goto 310
   ...
   return
300 if(lstruct2.gt.0) goto 310
   ...
   return
```

At each level (100-300) and also at the global level (:lstruct1=0) **effi** can call for *thstr_name()* routine, which itself introduces further constants and variables at that level and can call for another *thstr_name()* routine down to 8 levels.

8.1.2   The routine thstr_name (distribute)

The present structure of the routine *thstr_mossbauer()* makes 6 different child routines (or branch addresses) available for the routine named 'Mossbauer radioactive source'. This name was defined above for *frstr_mossbauer()*, the parent routine. **effi** looks according to the distribution list 9.1 into all *thstr_name()* for the selected parent theory (name) by the function *name_eq_ntheo()* (if(name_eq_ntheo(mthparent))l_jth=mthchild).

```
   subroutine thstr_mossbauer(l_jth,cmch,nhelpentry)
   include 'comglb.for'
   include 'comstr.for'
   include 'comcvm.for'

   if(cmch.eq.'name')then
    nhelpentry='Mossbauer radioactive source(distribute)'
    nameth(1) ='Mossbauer transmission_vA'
    nameth(2) ='Mossbauer transmission_vB'
    nameth(5) ='Mossbauer reemission_vA'
    nameth(6) ='Mossbauer reemission_vB'
    nameth(3) ='Mossbauer transmission_vC'
    nameth(4) ='Mossbauer transmission_vD'
    return
   else if(cmch.eq.'theo') then
    ntheory(1)='Mossbauer radioactive source'
    if(name_eq_ntheo(1))l_jth=6 ! mthparent=1,mthchild=6
    return
   end if

   chth='th'//nhelpentry(1:6)
```

The definition of constants, variables, parameters and matrices follows the same structure as in the *frstr_name()* routines. With the number l_jth of the child routines branch addresses are defined for individual definitions. Care must be taken for the position of the constants and variables in the common blocks.

```
   if(cmch.eq.'hdbk') then
```

```
  nstruct=' '
  return
end if
call conmatpar('coll',jc,jm,jp,jv,chth)
...
call conmatpar('layn',jc,jm,jp,jv,chth)
return
```

8.1.3   The routine **thstr_name**

The present structure of the routine *thstr_22indexfmoss()* makes 1 child routine nameth(1) available for 3 parent routines ntheory(1-3). **effi** looks according to the distribution list 9.1 into all *thstr_name()* for the selected parent theory (name) by the function *name_eq_ntheo()* (if(name_eq_ntheo(m

```
  subroutine thstr_22indexfmoss(l_jth,cmch,nhelpentry)
  if(cmch.eq.'name')then
  nhelpentry='2x2 refraction index_f (sad)'
c names for child theories , mthchild=1 ¡= max_nth
  nameth(1)='2x2 refraction index (static,f_anisotrop,sad)'
  namepg(1)='2x2_ref_index_static_sad plrs,1 '
  return
 else if(cmch.eq.'theo') then
c look for ntheory,mthparent=3 ¡=max_nth
 ntheory(1)='Mossbauer transmission_vB_3b'
 ntheory(2)='Mossbauer reemission_vB_3'
 ntheory(3)='Mossbauer transmission_vC_3b'

 if(name_eq_ntheo(3))l_jth=1 ! mthparent=3,mthchild=1
  return
  end if
      chth='th'//nhelpentry(1:6)
      if(cmch.eq.'hdbk') then
  nstruct='handbook=nucleus.hdb list=3 chth='//chth
  return
  end if
        chth='th'//nhelpentry(1:6)
```

The definition of constants, variables, parameters and matrices follows the same structure as in the *frstr_name()* routines. With the number l_jth of the child routines branch addresses are defined.

```
  if(cmch.eq.'hdbk') then
   nstruct=' '
  return
  end if
  call conmatpar('coll',jc,jm,jp,jv,chth)
  ...
  call conmatpar('layn',jc,jm,jp,jv,chth)
  return
```

The string namepg(i) comprises three informations. the first substring '2x2_ref_index_static_sad' is the name of the theory to be called. The theory is recognised by this name declared in the subroutine by the string variable:theoryname (*64 of 64 byte). The second string of 4 characters is passed to the theories by the common block 'common/struct/' (see sections 9.2, 9.3.1 and 9.5). Here it says 'plrs' - a theory including polarisation. The 3rd string is a number which differentiates between cases (here case=1) of the theory.

The parant routines are extended by two numbers which indicate the level (device, module, component) and the device number the theory can be called from. Here the theory '2x2_ref_index_static_sad' can be called from level 3= component (site theory) and from all devices of the parent theory 'Mossbauer reemission_vB', from devices 1 (source) and 2(absorber/scatterer) of the two transmission theories. b stand for 1 or 2. The definitions are:

nameth_as_ij	i=level, j=device
	i=1 device-, =2 module-, =3 component-level
	j=1,2,3 (maximum 3 different devices)
ntheory(k) selects sub_theories by name	
if ntheory(k)=name	nameth_as for all i,j
ntheory(k)=name_l	nameth_as level i=l of all devices
ntheory(k)=name_lm	nameth_as for i=l j=m
ntheory(k)=name_XY	X,Y=a,b,c,d
	a=all, b=1 and 2, c=1 and 3 , d=2 and 3
ntheory(k=name)_aa	=:ntheory(k)
ntheory(k)=name_ia	=:ntheory(k)_i
ntheory(k)=name_bd	theory nameth_as is offered for
	level 1 and 2 of device 2 and 3

Table 9: The definitions of the two characters added to the name of the thories

8.1.4 Definition of text strcmt()

The global structure comment has 5 lines (strcmt(i), i=1,2,...,5). The first one (strcmt(1)) has 6 possibilities shown in the table 8.1.4. The comment of ≤ 24 characters is used as a header in the set up procedure. n_set=ab or n_set=[ab] provides the name of the spectrum/measurement. If it is enclosed in brackets, the **effi** offers an alteration of the two letters.

The number is the number of devices used in the theory. If the number is bracketed the theory is written such a way, that device type number $1 \leq m\_device \leq 3$ can be multiply included in the theory or if m_device=0 the device number+1 are added (answer [y]) or even number+1 and number+2 are added (answer [Y] -capital Y). Since there are only 3 device types available, this last possibility can only be used for theories starting with 1 device.

Multiple devices are needed if the scatterer/absorber has different independent parts and the γ-beam is incoherently scattered.

The case of adding 1 device in some circumstances is programmed for the standard Mossbauer theory. The 3rd detector device is added for detector windows containing resonant nuclei, which have zero relative velocity with respect to the source.

For each device modules(layers) are defined (lstruct1=1,2,3 and lstruct2=0), the comment may designate the type of device, 2 characters for the name of the device, a string for the construction

$$
\begin{array}{llllll}
\text{strcmt(1)=} & \text{'comment} & \text{n\_set=ab} & \text{number} & \text{0'} \\
& \text{'comment} & \text{n\_set=ab} & \text{[number]} & \text{0 '} \\
& \text{'comment} & \text{n\_set=ab} & \text{[number]} & \text{m\_device'} \\
& \text{'comment} & \text{n\_set=[ab]} & \text{number} & \text{0 '} \\
& \text{'comment} & \text{n\_set=[ab]} & \text{[number]} & \text{0 '} \\
& \text{'comment} & \text{n\_set=[ab]} & \text{[number]} & \text{m\_device'} \\
\end{array}
$$

strcmt(2)= 'name for the set'
strcmt(3)= 'text dependent on [number] m_device'
strcmt(4)= '**effi**/name/set'
strcmt(5)= 'theory/'//chth//'/helpfile_entry'

Table 10: The strings strcmt() inside the section of global definitions

of default names for the modules, the number (default number) of modules and its lower limit (ll ≥ 1).

$$
\begin{array}{llllll}
\text{strcmt(1)=} & \text{'comment} & \text{n\_device=ab} & & \text{[number]} & \text{ll'} \\
& \text{'comment} & \text{n\_device=[ab]} & \text{li} & \text{[number]} & \text{ll'} \\
& \text{'comment} & \text{n\_device=[ab]} & \text{mij=xx;yy} & \text{[number]} & \text{ll'} \\
& \text{'comment} & \text{n\_device=[ab]} & \text{mijk=xx;yy;zz} & \text{[number]} & \text{ll'} \\
\end{array}
$$

strcmt(2)= 'name for the module'
strcmt(3)= 'number of modules of the device'
strcmt(4)= 'distribution of parameters at components of the module'
strcmt(5)= 'theory/'//chth//'/helpfile_entry'

Table 11: The strings strcmt() for the definitions of the device name and its structure.

If the device has more than 1 module a name of the module (2 characters) is added to the name of the parameters and constants. Column 4 The rules for the construction of default names for the modules (column 4) are as follows. If it is empty, the default is also empty. mi: The names are m1,m2,...; mij=xx;yy: xx, m2,m3,...,yy; mijk=xx;yy;zz: xx,y1,y2,..,zz or for 3 modules xx, yy, zz.

For each module of each device components are defined (lstruct1=1,2,3 and lstruct2 >0), the comment may designate the type of device (according to lstruct1), n_cmpt defines the one character name of the component of the actual module (lstruct2>0 counts the modules) and actual device, [number] the number of components, ll the lower limit of that number. ll=0 is also allowed (e.g. a layer may be present only by its electronic susceptibility without nuclear scattering centers.)

The queries for distribution of parameters belonging to a site are on two levels. On the device level for each layer a query (y/n): distribution of parameters at sites of the layer, on the module (layer) level site for site a y/n query have to be answered. In a last step (fixed text-string) the number of distributed parameter sets is requested.

strcmt(1)=	'comment	n_cmpta	[number]	ll'
	'comment	n_cmpt=[a]	[number]	ll'

strcmt(2)=	'one letter for name of the site'
strcmt(3)=	'number of inequivalent sites of the layer'
strcmt(4)=	'distribution of parameters of site number'
strcmt(5)=	'theory/'//chth//'/helpfile_entry'

Table 12: The strings strcmt() for the definitions of the name of the sites (components) and the number of sites inside a layer (module). Each site may consist of a distribution of sites with up to 3 parameters independently distributed.

8.1.5 Definition of constants thco()

The subroutine *conmatpar('coll',jc,jm,jp,jv,chth)* provides the index jc. The next constants to be defined is jc+1, jc+2,... At the end jc has to be incremented to the present value.
The string contains the name(≤ 16 characters), the relative position added to the positions in the common block, an option how to handle this constant, and the default value x.

thco(jc+1)=	'name	i_pos=m	include=[y]	x'
	'name	i_pos=m	include=[n]	x'
	'name	i_pos=m	include=y	x'
	'name	i_pos=m	include=n	x'
	'name	i_pos=m	from_data=[y]	x'
	'name	i_pos=m	from_data=[n]	x'
	'name	i_pos=m	from_data=y	x'
	'name	i_pos=m	from_data=n	x'

Table 13: The string thco() for the definition of constants

The options in brackets appear in the set up of the theory. The bracketed character is the default one.
The option 'include' may always be answered =y. In order to have not constants in the list, which value is zero and which are never altered for the special problem, the answer =n puts the constant to zero and excludes it from any printout list. There appear to be many constants of that type when selecting specialized theories. The option 'from_data=n' is equivalent to 'include=y'.

The option 'from_data' is useful for constants which are defined by the data set, e.g. the channel number= number of data points, or other characteristics like the drive mode. If no data file is read or the constant is not defined in the header of data file the default value is taken. In case of no data file the constant appears in the list shown by the command test:{te} otherwise the command read data:{rd} leads to the subcommand {header of data} which shows the list.

8.1.6 Transformation matrices ntrmt(), variables thvr(),and parameters thpa()

As for the constants 8.1.5 the subroutine *conmatpar('coll',jc,jm,jp,jv,chth)* provides the indices jm,jp,jv for the matrices, parameters and variables. The indices have to be incremented to the present value.

The string ntrmt for the matrix contains the name($\leq$ 18 characters), the dimension of the matrix and an option which has the same definition as above for the constants 8.1.5,

$$
\begin{array}{llll}
\text{ntrmt(jm+1)=} & \text{'name} & \text{matrix=(m,n)} & \text{include=[y]'} \\
 & \text{'name} & \text{matrix=(m,n)} & \text{include=[n] '} \\
 & \text{'name} & \text{matrix=(m,n)} & \text{include=y '} \\
 & \text{'name} & \text{matrix=(m,n)} & \text{include=n '} \\
 & & & \\
 & \text{'name} & \text{matrix=(m,n)} & \text{from\_data=[y] '} \\
 & \text{'name} & \text{matrix=(m,n)} & \text{from\_data=[n] '} \\
 & \text{'name} & \text{matrix=(m,n)} & \text{from\_data=y '} \\
 & \text{'name} & \text{matrix=(m,n)} & \text{from\_data=n '} \\
 & \text{'name} & \text{intensity '} & \\
\end{array}
$$

Table 14: The string ntrmt() for the definition of a transformation matrix. The option 'from_data=n' is equivalent to 'include=y'and 'intensity' stands for 'matrix=(1,1) include=y'.

The string ctrmt contains a description of the matrix, the parameters which is printed in the selection procedure of **effi**. The parameters are transformed to the variables by the matrix as shown in table 3. The parameter and variable names are $\leq$ 16 characters. The parameters have a default value x, a lower and upper limit ll,ul and a step width step for numerical differentiation. The string 'thvr()' of the variables contains the relative position i_pos and the row elements of the matrix.

$$
\begin{array}{lll}
\text{ntrmt(jm+1)=} & \text{'name} & \text{matrix=(3,2)} \quad \text{include=[y]'} \\
\text{ctrmt(jm+1)=} & \text{'description of the matrix type'} \\
\text{thpa(jp+1)=} & \text{'parameter1} & \text{x1 ll1 ul1 step1'} \\
\text{thpa(jp+2)=} & \text{'parameter2} & \text{x2 ll2 ul2 step2'} \\
\text{thvr(jv+1)=} & \text{'variable1} & \text{i\_pos1} \quad \text{1.0 0.0 '} \\
\text{thvr(jv+2)=} & \text{'variable2} & \text{i\_pos2} \quad \text{0.5 0.5 '} \\
\text{thvr(jv+3)=} & \text{'variable3} & \text{i\_pos3} \quad \text{0.0 1.0 '} \\
\end{array}
$$

Table 15: The complete definition of a transformation matrix with its parameters and variables.

9 The fork of subroutines

The subroutines named *thstr_name(l_jth,cmch,nhelpentry)* are collected into files named *str_method.f* where sttr stands for structure and method for mossbauer, neutron, stroboscopy, etc. the strings 'cmch' and 'nhelpentry' play the same role as for the *frstr_nmae()* routine described in sec. 8.1.1.

The output integer l_jth specifies the number of prongs of the fork.

```
subroutine thstr_mossbauer(l_jth,cmch,nhelpentry)

if(cmch.eq.'name')then
nhelpentry='Mossbauer radioactive source (distribute)'
c names for child theories, mthchild=6
nameth(1) ='Mossbauer transmission_vA'
nameth(2) ='Mossbauer transmission_vB'
nameth(5) ='Mossbauer reemission_vA'
nameth(6) ='Mossbauer reemission_vB'
nameth(3) ='Mossbauer transmission_vC'
nameth(4) ='Mossbauer transmission_vD'
return
else if(cmch.eq.'theo') then
ntheory(1)='Mossbauer radioactive source'
if(name_eq_ntheo(1))l_jth=6 ! mthparent=1,mthchild=6
return
end if

chth='th'//nhelpentry(1:6)

if(cmch.eq.'hdbk') then
nstruct=' '
return
end if

call conmatpar('coll',jc,jm,jp,jv,chth)
......
call conmatpar('layn',jc,jm,jp,jv,chth)
return
end
```

9.1 Distribution list of theories

A theory which shall be added to **effi** becomes known by **effi** by 4 subroutines. The first two of them assign numbers to the name of the subroutines called by the frame theory and all other theories.

effi runs through all numbers nfrtheo (fr_routine number),l_nfrtheo (cases handled in one fr_routine) and collects all names, which are listed on screen. The user selects a theory from the list. **effi** takes out further information from the selected theory by the strings cmch*4 and property*132. The two numbers (nfrtheo,l_nfrtheoare) are communicated to **effi** as the first 4 characters of the string property (here symbolically written as a function characterof()). These 4 characters are attached to the constants and parameters defined in that subroutine, in order to find them when calling the help function.

9.1.1 Structure selecting routines

```
subroutine getfrstring(nfrtheo,l_nfrtheo,lstruct1,lstruct2,
                       cmch,property)
property=characterof(nfrtheo)//characterof(l_nfrtheo)
if(nfrtheo.eq.1) then
  call frstr_stroboscopy(l_nfrtheo,lstruct1,lstruct2,
                       cmch,property)
else if(nfrtheo.eq.2) then
  call frstr_synchrotron(l_nfrtheo,lstruct1,
                       lstruct2,cmch,property)
else if(nfrtheo.eq.3) then
  call frstr_mossbauer(l_nfrtheo,lstruct1,lstruct2,
                       cmch,property)
  ...
else if(nfrtheo.eq.nn) then
  nfrtheo=nn (number of entries)
end if
return
end

subroutine getthstring(jth,l_jth,cmch,property)
property=characterof(nfrtheo)//characterof(l_nfrtheo)
if(jth.eq.1) then
  call thstr_11indexmoss(l_jth,cmch,property)
else if(jth.eq.2) then
... else if(jth.eq.18) then
  call thstr_syntransmission(l_jth,cmch,property)
else if(jth.eq.19) then
... else if(jth.eq.30) then
  call thstr_aperture(l_jth,cmch,property)
end if
  jth=30
return
end
```

9.1.2 Assignment of subroutines

The distribthfr(distribute theories for frame) and distribth(distribute theories) routines call the routines by the number which was before found out by **effi** calling all routines by name. The string 'nath' (name of theory has been introduce for the status command {st theory}, which enables a write command for the sequence of theories called.

```
subroutine distribthfr(iframe,cmch,iss,info)
if(iframe.eq.1) then
  call fr_stroboscopy(cmch,iss,info)
else if(iframe.eq.2) then
  call fr_synchrotron(cmch,iss,info)
```

```
...
else if(iframe.eq.4) then
  call fr_neutron(cmch,iss,info)
end if
  iframe=4
return
end

subroutine distribth(isubthe_n,ntheo,icase,iput,iget,nath)
if(ntheo.eq.1) then
  call th_22index(isubthe_n,icase,iput,iget)
  nath='th_22index'
else if(ntheo.eq.2) then
  call th_hdeq(isubthe_n,icase,iput,iget)
  nath='th_hdeq'
else if(ntheo.eq.3) then
  ...
else if(ntheo.eq.17) then
  call th_relaxtwostatesad(isubthe_n,icase,iput,iget)
  nath='th_relaxtwostatesad'
else
  isubthe_n=-isubthe_n
end if
  ntheo=17
return
end
```

9.2   Structure of the frame subroutine

A theory called by **effi** is called by the frame theory: fr_routine. There is a first sequence of calls for some information from the theory which is then stored on a separated part of array_fr. These are pointers and constants and strings which typically do not change when the program is repeatedly called by a fit routine. The routine stores this information on the common/struct/ block the content of which is transferred to array_fr and later on read from array_fr by **effi** before entering the fr_routine.

The description of the 'first call' is along the code of the *subroutine fr_mossbauer(cmch,iss,info)* of the fortran file *pg_mossbauer.f*. The body of the *fr_routine()* is also outlined on the basis of Mossbauer theory.

To the frame of a type of theory (like Mossbauer) belong further subroutines called at different labels of the *fr_routine()*. They are all independent of the tree of routines starting with *th_mainroutine()* (see 9.3.1)

9.2.1   The first call

The *fr_routine()* has three arguments: the string cmch*4, the counter iss, and an integer info. The integer iss also function later on as read or write channel opened in **effi**. Both cmch and iss are used for calls for information. The string cmch has 3 values: 'arr0', 'arri' with iss > 0 and 'arre'.

- arr0: Global constants are defined, which are written to the common block 'common/struct/', and a size of a segment of the large complex*4 array named array_fr() by the integer 'ifrsto'. If experimental data are read in by an user written routine (here rdpu2d.f and rdpu3d.f, read and punch of2 and 3 dimensional data) the data are stored on the first segment from 1-nuofdatas (nuofdatas has to be defined in the read data routine). Further segments are reserved for some constants (5 of them) theoretical results and information about subspectra. Two pointers are defined, that is nstorTH for the result of the theory and nstorTHyy for the theoretical values at each experimental data point (by interpolation) for calculation of chi**2. The relative adresses are obvious. The theory at nstorTH+1, the base has its value at nstorTH+nstorTH2+1, and the theoretical values for chi**2 at nstorTHyy+1.

- arri: The structure of any theory is defined by a nested loop lstruct1 counting devices, lstruct2 counting the 'modules of devices, and lstruct3 the components of a module. **effi** provides a projective map between the integers 'iss' and the triplets lstructi by the subroutine *l_structure(iss,lstruct1,lstruct2,lstruct3)*. Theoretical data of each calculation within this nested loop (that means for each iss value) may be stored in the array_fr(). The required size is again written on ifrsto.
  Besides these storage space information from the subroutines *str_routine()* is transferred by the 4 character strings 'ci_prp()', which is used to set constants, which are added to the common/struct/ block to be taken by the corresponding sub-theories.

- arre: If the nested loop ends (decided inside **effi**), there is a last possibility to reserve storage space. **effi** enters the fr_routine with 'cmch=arre'. Here storage space is reserved for all absorber layers together depending on the polarization. It is accessible at the index l1_put(mstruct1+1)+1 (see pointer structure 9.5).

After collection of the information about the theory the string cmch is used for the different tasks of the theory (Mossbauer: 'test', 'dfdp','subs', 'subt', 'plex', 'dmod', 'plot') which are assigned by a distrution list (goto statement from label 100 to 500) (see "Body of the fr_routine").

The space belonging to nuofdatas of experimental data to be fitted is the only one on the array 'array_fr, which is required by **effi**. The other storage space is organized by the programmer (HS) in such a way, that repeated calculation of parts of the theory cam be avoided.
E.g. the subroutine which calculates the Lorentz curve with nstorLG values is stored and calculated only once if the values for the linewidth and maximum velocity remains the same for different sites in the absorber.

effi carries an integer 'info' to the main output (see section "Help file" 7).

9.2.2 Body of the fr_routine

subroutine fr_mossbauer(cmch,iss,info)
include 'comglb.for'
include 'comdat.for'
include 'complt.for'
include 'comthe.for'

```fortran
      common/struct/mstruct1,
     &       istruct1(0:maxfr1),istruct2(0:maxfr2),
     &       l1_get(0:maxfr1),l2_get(maxfr2),l3_get(maxss),
     &       l1_put(0:maxfr1),l2_put(maxfr2),l3_put(maxss),
     &       c1_prp(0:maxfr1),c2_prp(maxfr2),c3_prp(maxss),
     &       nstorTH,nstorTHyy,iaddress(6),
     &       nstorTH1,nstorTH2,nstorLG,nAS,nSC,nLG,ak8,
     &       sk8,srw8,lstr1case,jpolsource,jpolabsorber,cfill(12)

      common/acstrf/acsp,ajg,aje,amueg,amuee,QgdQe,r_E2M1,
     & cTms,gamnat,uthick_w,uthick_A,wavelength,
     & relchi,flag_SAMP,flag_Voigt,channels,vmsplit,aib,
     & drive_mode,channels_vmax,....

      common/avstrf/avsp,base,f_factor,bg_fraction,geo,v_max,
     & channel_v0,v_max2,vphase2,v_max3,vphase3,...

      parameter(km=256)
c**********************************************
      nchansp=channels
      if(cmch.eq.'arr0') then
        ...
        calculations of the channel number of the source nSC
        and the absorber nAS in view the convolution. They
        dependent on the variable v_max and the constant
        vmsplit, the splitting of the source.

        nLG=4*(nAS+nSC) Lorentzs curve, the size
        is needed for convolution of source and absorber

        nstorTH1=1            base (one real number)
        nstorTH2=km+1         theory spectrum

        nstorLG=(nLG+1)+(nLG+2*mGa+1)+mGa+1
        storage for Lorentz-Gauss curves

        ifrsto=nstorLG+nstorTH1+nstorTH2+nchansp/2
        total storage taken up by effi

        nstorTH=nstorLG        pointer for the result
        of the theory

        nstorTHyy=nstorTH+nstorTH1+nstorTH2    pointer
        for the result of the theory transformed to nchansp
        experimental data points (real)

        jpolsource=0
        jpolabsorber=0        1,0= polarized or not
        return
      else if(cmch.eq.'arri') then
        call l_structure(iss,lstruct1,lstruct2,lstruct3)
```

```fortran
  ifrsto=0
  if(lstruct1.eq.2.and.lstruct3.gt.0.and.
  c3_prp(lstruct3).eq.'plrs') jpolabsorber=1

  if(lstruct1.eq.2) then
    if(lstruct3.eq.0) jpollay=0
    if(istruct2(lstruct2)-istruct2(lstruct2-1).gt.1) then
      if(c3_prp(lstruct3).eq.'plrs') jpollay=1
      if(lstruct3.eq.istruct2(lstruct2)) then
        if(jpollay.eq.1) then
          ifrsto=4*(nAS+1)
        else
          ifrsto=nAS+1
        end if
      end if
    end if
  end if
  return
else if(cmch.eq.'arre') then
  storage for absorber (sum over all layers)
  ifrsto=nAS+1
  if(jpolabsorber.eq.1) ifrsto=4*(nAS+1)
  return
end if ! arr0,i,e

srw8=abs(v_max)/float(km)/8.
this stepwidth is calculated here and transferred to the subroutines with the common/struct/. The
transfer of v_max and the parameter km (see declaration) is avoided.
mi=istruct2(istruct1(mstruct1))
is used for the calculation of the subspectra (developers solution)
*********************************************
if(cmch.eq.'data') then
  goto 100
else if(cmch.eq.'test') then
  goto 200
else if(cmch.eq.'dfdp') then
  goto 300
else if(cmch.eq.'subs'.or.cmch.eq.'subt') then
  goto 400
else if(cmch.eq.'dmod'.or.cmch.eq.'plot') then
  goto 500
end if
*********************************************
100 continue
  i=int(mode_drive)
  if(i.le.2) data_str='y,dy'
  if(i.ge.3) data_str='x,y,dy'
  return
*********************************************
200 continue
```

```
k1=1
call spec2exp_moss(k1,info)
return
```
...

effi does not accept data without a theory defined. The idea behind is that data can only be plotted with some pre-knowledge about their structure. Before reading the data the jump to label 100 in the fr_routine defines the expected format depending on some constants like the drive mode of the transducer of the Mossbauer set up.

The command test leads to the subroutine call of spec2exp_moss(k1,info). spec2exp_moss organizes the connection between experimental data and theoretical values. The subroutine calls the main subroutine th_moosbauer and expects the theoretical values at nstorTHyy. The theoretical values are calculated on a linear scale, in case of the Mossbauer spectra with a fixed channel number 512. According to the x-scale of the experimental data the theoretical values are interpolated and stored in an array of a common block which is accessible for the fit routines.

9.2.3 Connection to the experimental data

When the fr_routine is called by **effi** (after the first call) it branches (in the Mossbauer case) to 'subroutine spec2exp_moss()' for the simulation of the theory (command {te}), calculation the derivative for a parameter (command {ft}), and plots of sets and subsets (command {pl}). By the integer number ifrcase the th_mainroutine is selected, here 'mossbauer_refl(b_total)' or 'mossbauer_trm(b_total)', and comes back with the logical 'b_total'. If b_total=true, the theory was recalculated and the values stored in array_fr have been changed, otherwise not. The developer may use this knowledge in order to save computing time.

The task of this subroutine is the interpolation of the theoretical values calculated at a certain number of channels to the scale of the experimental data. Thereby, the number of theoretical channels may be larger or smaller than the number of measured data. This interpolation is also done for the derivatives and for the plot especially also for the plots of the subsets.

The interpolated theoretical values are stored on 'array_fr(nstorTHyy+j)', j=1,..., number of experimental data. The derivatives are stored in a common block 'commonscrt' which is used by fr_routine for other purpose too.

9.2.4 Connection to the plot routine

The plot routines are specific for the type of the theory. For Mossbauer the name is 'subroutine plotfold(cmch)'. (The attribute 'fold' points at the folding procedure of Mossbauer spectra -history). The string cmch (4 characters) is transferred by the calling fr_routine. If cmch='dmod' (display mode) **effi** asked the theory of the set for its display modi, which later is stored in the number ndisplay_mode. In the Mossbauer case there are 4 display modi. When printing the experimental and theoretical curves to a file **effi** takes from here also short names for the different data arrays used in the plot of the screen and a string for a default printout: plotstrdef='(xd,yd,erry) (xt,yt)'. For example, (xd,yd,erry) means three columns with xddata (may be velocity), ydata (counts or counts normalized to 1), erry (error values of ydata. The next bracket defining theory data is appended in the file (see {he wd} (write display data). The ndisplay_mode number allows **effi** also to obtain from this routine the Fortran format of the values to be printed and the default

names of the axes.

The plot routine does not plot but filles the common block 'common/plot/' which contains all information for the routine which does the plot on the screen. Here the C-program xmosplo.c and further subroutines are used. This routine is a derivative of the code programmed by A. Vef for MOSFUN and other routines like MODEL.

effi first calls for all the information necessary and then calls for the plot routine executing the plot in fr_routine label 500. Here is it xmosplo(). The developer could also offer other plot routines like xmgrace by extending the list of ndisply_mode. The common block 'common/plot/' is transferred to xmosplo by a external structure plot{}.

The complex code of this routine for Mossbauer is necessary for the folded spectrum. **effi** fits the unfolded spectrum and only for the plot it is folded. That means experimental data from the first half and second half spectrum have to be added, which needs interpolations of the data too, as the folding channel is not necessarily an integer number.

9.3 Structure of the subroutines of the files th_routine.f

The structure of the main subroutine (th_main-routine) of a user written theory (example here th_mossbauer()) and the subroutines (th_routine, examples are th_11index(), th_11indexsad(), th_22indexfsad(), th_hdeqf(),...) are fixed by the structure of **effi**. The main subroutine does not really take care about the experimental data. In the Mossbauer example the number of channels (=512, fixed) calculated is independent of the number of experimental data. The main subroutine th_main-routine() calls recursively by the subroutine get_theory() the subroutines th_routine().

9.3.1 Body of the th_main-routine

The developer finds 2 include files 17 common blocks, 4 calls to subroutines followed by do loops which take values from array_fr to a local array of interim results, here called res_interim.

The *include file* comglb.for contains the dimensions of the various arrays, here maxfr1,maxfr1,and maxss. The file comthe.for contains few important addresses for **effi**, two indices for case selections of the user built theory (see later) and a large complex array array_fr(marray_sp=1024*786 at present)(single precision real*4 values) which should be used to store interim results and the final result of the theory at pointer positions provided by **effi**. This is necessary because the theory can be called on different parameter sets by **effi**, so that the user will overwrite the results on any locally defined storage area.

The four groups of *common blocks* contain the variables and constants entering the theory with names as chosen by the developer. There are common blocks for constant values and variables (which are 'connected' to the parameters by the transformation matrices) as indicated by the second letter of the common block name (c, v). The first letter is an a for real values or a b for logical values. Each constant and variable on the common block have a logical value at the corresponding place on the b-block. The value is .true. if the real value has been changed since the last entry of the routine and otherwise .false.. Here at this place shall be mentioned that the first logical of the constants has the value b_c****=b_cont(2).or.b_cont(3).or...., such that the value is .true. if at least one constant has changed otherwise .false. (see section "Common blocks"9.4). The developer can use this information to avoid recalculations and instead read the interim result

from array_fr(pointer1...pointer2). How to handle the pointers in interaction with **effi** is described in section "Pointers on array_fr" 9.5.

The common blocks with the endings of the block name $strf$ (set-transfer) are reserved for global constants and variables, which are the same for all cases of the theory. For a Mossbauer theory these are channel number, constants describing geometry, drive mode etc and parameters like base, background, etc.

When entering the subroutine mossbauer(b_total) the values of the common blocks are not yet assigned. The *subroutine mapsub(iss)* maps the values from an internal large array to this local common block. With the counter value iss=0 the $strf$ common blocks for global values are filled. The further 3 groups of common blocks are assigned inside the 3 nested loops. The counter iss which in each loop is incremented by 1 tells the routine mapsub(iss) to assign the proper group of common blocks (the position after mapsub to increment iss must not be changed).

The *subroutine get_theory(issthe_n)* is called next. It has a counter issthe_n which is increases inside the subroutine. For simply constructed programs this call will not be used. Instead the developer will call directly a subroutine which calculates some output with the common block as input or he writes directly down a code at this place. The get_theory(issthe_n) subroutine (belonging to **effi**) is supposed to be used if there is a decision branch in the user's set up in order to take the subroutine which fits to the present data evaluation. An example implemented are the calculation of Fourier coefficient of time windows for stroboscopic Mossbauer measurements. In the set up the type of time windows are selected. **effi** will call the proper subroutine, and stores the result in array_fr. The pointer l1_get(0) to the coefficients are contained in the first common block common/struct/ which will be described later (section "Pointers on array_fr"9.5).

```
subroutine mossbauer_tmr(b_total,info)
include 'comglb.for'
include 'comcvm.for'
include 'comthe.for'
common/struct/mstruct1,istruct1(0:maxfr1),istruct2(0:maxfr2),
&          l1_get(0:maxfr1),l2_get(maxfr2),l3_get(maxss),
&          l1_put(0:maxfr1),l2_put(maxfr2),l3_put(maxss),
&          c1_prp(0:maxfr1),c2_prp(maxfr2),c3_prp(maxss),
&          frthname(2),nstorTH,nstorTH1,nstorTHyy,iaddress(5),
&          jpolsource,jpolabsorber,ifrth(6),
&          iflag_basis,l_Poincare,nLG,nAS,nSC,idum2,ak8,sk8,srw8,idex(7)

common/acstrf/acs,constant(icstrf-1)
common/avstrf/avs,variable(ivstrf-1)
common/bvstrf/b_vs,b_variable(ivstrf-1)

common/acfr1trf/acfr1,constant(icfr1trf-1)
common/avfr1trf/avfr1,variable(ivfr1trf-1)
common/bvfr1trf/b_vfr1,b_variable(ivfr1trf-1)

common/acfr2trf/acfr2,constant(icfr2trf-1)
common/avfr2trf/avfr2,variable(ivfr2trf-1)
common/bvfr2trf/b_vfr2,b_variable(ivfr2trf-1)
```

```
common/acsstrf/acss,constant(icsstrf-1)
common/avsstrf/avss,variable(ivsstrf-1)
common/bvsstrf/b_vss,b_variable(ivsstrf-1)

call mapsub(0,0,0)
issthe_n=1
call get_theory(issthe_n)
if(issthe_n.lt.0) no parameter/constant change (in the whole tree of subroutines called)
if(issthe_n.eq.0) no subroutine called
do k=1,l
  res_interim(k)=array_fr(l1_get(0)+k)
end do


do lstruct1=1,mstruct1
  call mapsub(lstruct1,0,0)
  issthe_n=1
  call get_theory(issthe_n)
  see above for issthe_n ≥ 0
  do k=1,l
    res_interim(k)=array_fr(l1_get(lstruct1)+k)
  end do

  do lstruct2=istruct1(lstruct1-1)+1,istruct1(lstruct1)
    call mapsub(lstruct1,lstruct2,0)
    issthe_n=1
    call get_theory(issthe_n)
     see above for issthe_n ≥ 0
    do k=1,l
      res_interim(k)=array_fr(l2_get(lstruct2)+k)
    end do

    do lstruct3=istruct2(lstruct2-1)+1,istruct2(lstruct2)
      call mapsub(lstruct1,lstruct2,lstruct3)
      issthe_n=0      ! here 0 instead of 1
      call get_theory(issthe_n)
      see above for issthe_n ≥ 0
      do k=1,l
        res_interim(k)=array_fr(l3_get(lstruct3)+k)
      end do

    end do            ! lstruct3
  end do            ! lstruct2
end do            ! lstruct1


do k=1,n/2
array_fr(nstorTH+k)=cmplx(result(k),result(k+n/2))
end do
return
end
```

effi allows the subroutines of the file 'th_routine.f' called by get_theory(issthe_n) also to call

get_theory(issthe_n) up to a depth of 8 calls (see section "Subroutines called by get_theory"9.3.2).

The result of the subroutine th_mossbauer() is at the end stored in *array_fr*. As the theory function is real it is here by one half stored in the real and the other half in the imaginary part of the complex array. The array has been chosen of complex type because almost all interim results to be stored are complex numbers. The array can also be handled as a real array *real_fr()* defined by the command: equivalence (array_fr(1),real_fr(1)) to found in the include file *comthe.for*.
The pointer nstorTH is user defined with the aid of **effi** (see section "Pointers on array_fr"9.5) If the theory function has not changed at all (the logical values of all parameters and constants are .false.) the developer can communicate this fact to the calling routine by the logical b_total.

Three nested loops are supported by **effi**, that means that the common block belonging to a loop are assigned by the call of mapsub(lstruct1,lstruct2,lstruct3). **fr1trf belongs to the outer loop lstruct1, **fr2trf to the loop lstruct2, and **sstrf to lstruct3. The loops are organized in such a way that the counters lstruct2 and lstruct3 are continuously increasing as does lstruct1. At each level (device, module, component) of the loop a subroutine can be called by get_theory and the interim result res_interim(...) calculated by these theories is taken from a array_fr(li_get(lstructi)+...).
For the Mossbauer theory the loops have the following meaning: lstruct1 counts devices. 3 types are defined, 1:source, 2:absorber, and 3:detector. The two further loops (lstruct2) are used for layers (the modules of the theory) and (lstruct3) for nuclear sites (the components of the the modules) inside the layers. The source and the thin detector window are considered as one layer, so that this loop is used only once for them. The absorber my consist of several layers (see figure 1) the number of which can be chosen by the user. The variables and constants belonging to source,absorber, detector of the outer loop 1, belonging to each layer in loop 2 and to each nuclear site in loop 3 have the same names (the names in the common block) but will have different values, which are mapped by mapsub onto the common block.
The same names of variables in the common block for all loops can only be used in a meaningful way if the theories called for source,..., detector, each layer and each site are the same or at least very similar. This disadvantage has been overcome by introducing a 5th group of common blocks in the subroutines called by get_theory (see section "Subroutines called by get_theory"9.3.2).
Simpler structured theories like X-ray scattering on multilayers (no nuclear scattering) lstruct1 is just one and lstruct3 is not executed. For even simpler structured theories only the common block of the outer loop is used and only once.

9.3.2 Body of the th_routine called by get_theory

The structure of a user written subroutine called by get_theory -here called th_abc(issthe,icase,iput,iget)- is also fixed by the structure of **effi**. There are 4 common blocks, a single one common/struct/, which contains some pointers and constants defined in the fr_routine (see section "Structure of the fr_routine"9.2), and a group of 4 which are almost of the same shape as the group containing the global constants and variables (see previous section "Structure of the user written fr_routine"9.2). A further call to get_theory(issthe_n) is optional (mark that the counter issthe_n is untouched here). The calculated values for res_interim are obtained from array_fr at the address *iget* provided by **effi**. The result of this subroutine (next do loop of the body) is written on array_fr(*iput* +...).
The branch given by *icase* is defined by the developer and is selected by the user.

subroutine th_abc(issthe,icase,iput,iget)

```
      include 'comglb.for'
      include 'comthe.for'

(character*4 c1_prp,c2_prp,c3_prp,frthname*64, the strings take the space of 3*maxss+32 real*4
variables of dum.)
      parameter(idum=6+4*(maxfr1+maxfr2)+3*maxss+32+8)
      common/struct/dum(idum),
     &         jpolsource,jpolabsorber,ifrth(6),
     &         iflag_basis,l_Poincare,nLG,nAS,nSC,idum2,ak8,sk8,srw8,idex(7)

      common/acmaps/c1,const(cstrf-17),theoryname
      common/bcmaps/b_c1,b_const(cstrf-1)
      common/avmaps/variable(cstrf)
      common/bvmaps/b_variable(cstrf)

      if(issthe.lt.0) then
        theoryname='refraction_index'
        if(icase.eq.1) then
         c1=constant(i)
        else if(icase.eq.2)
         c1=4*constant(i)
        end if
        return
      end if
         .
      call get_theory(issthe_n)
      see above for issthe_n ≥ 0

      do k=1,l
        res_interim(k)=array_fr(iget+k)
      end do
         .
      m=constant(i) or 4*constant(i)
      do k=1,m
        array_fr(iput+k)= result(k)
      end do
      return
      end
```

When the subroutine ist called the first time by **effi** the value of issthe is -lstruct1. **effi** reads
the name of the theory (here refraction_index) from the common block acmaps (theoryname*64),
which is used to recognize the subroutine in the assignment list (see section "Assignment list"9.1.2),
and requests for storage size on array_fr. This request may depend on the case (icase=1 or 2).
For example the refraction index may be a scalar or a 2x2 matrix, which requires 4 times more
storage space for the result of the theory. The requested storage size is written on c1. The stor-
age requirement typically depends on global constants like channel numbers, the accuracy of the
calculation etc. In the fr_routine (section "Structure of the fr_routine"9.2) several constants are
derived from the global constants when it is called first. Such derived constants are transported to

the subroutines which need them by common/struct/. The developer is of course free to declare also the global common/acstrf/ here and take the information.

9.4   Common blocks of the 3 levels

The group of common blocks which belong to the three levels (device, layer, site) are of type' real*4' and 'logical'. with dimensions icXtrf/ivXtrf (X=fr1, fr2, ss). The first constant/variable is the number of constants/variables of the common block. These numbers are added by **effi**.

The second place of the common/avXtrf/ of the variables is expected to be an intensity variable by subroutines which calculate subsets (subspectra). The first logical has the value b_vX = b_variable(1) .or. ... .or. b_variable(avXtrf-1)

common/acXtrf/acX,constant(icXtrf-1)
common/avXtrf/avX,aint,variable(ivXtrf-2)
common/bvXtrf/b_vX,b_int,b_variable(ivXtrf-2)

In the group of constants/variables of a common block, common/acmaps/ and common/avmaps called by get_theory, the first place, both for constants and variables, counts the number of constants/variables as in the acXtrf and avXtrf common blocks. **effi** actually takes the values from the acXtrf and avXtrff common blocks. For example if a1,...,al are the variables defined in the main subroutine and b1,...,bm in the subroutine beta called by get_theory in main and g1,...,gn in subroutine gamma called by get_theory in subroutine beta then the common block is assigned as follows:

common/avXtrf/l+m+n,a1,...,al,b1,...,bm,g1,...,gn,
                variable(ivXtrf-(1+l+m+n))
This means, that values for bi are already available in the fr_routine. In the subroutine the variables have the names g1,...,gn, in the fr_routine the values are a member of the array avXtrf with indices from l+m+1 to l+m+n. The same holds for the constants.

9.5   Pointers on array_fr

The common block *common/store/* defines a large complex array_fr(maxn*768) which stores all the information belonging to a number of sets. If the storage is not sufficient for the 32 sets allowed, the information of sets is externalized on disk with file name storage.am. This file is removed if **effi** has been left by a command ({qu} or({ex}. For each set the first 2200 complex numbers contain information evaluated by **effi**. The block '*common/struct/*' is stored in this first section of the part on the array_fr belonging to the actual set. With the first call of the set the content of '*common/struct/*' is transferred to array_fr and for each later call transferred back to the common block (one has to keep in mind that different sets may use the same theory differently structured).

common/struct/mstruct1,
&   istruct1(0:maxfr1),istruct2(0:maxfr2),
&   l1_get(0:maxfr1),l2_get(maxfr2),l3_get(maxss),
&   l1_put(0:maxfr1),l2_put(maxfr2),l3_put(maxss),

```
&        c1_prp(0:maxfr1),c2_prp(maxfr2),c3_prp(maxss),
&        frthname(2),nstorTH,nstorTHyy,iaddress(6),
&        ifrth(8),a1,n2,n3,...,a10,nfill(6)
```

The block contains the definition of the nested loops (the boundaries) and the pointer li_get where the results of the of the subroutines called by get_theory 9.3.1 are read from. l1_get(0) (instead of l0_get) is the pointer for the results outside the loop in the beginning of the subroutine. Starting with the pointer li_put results are written on array_fr. The pointer list can be printed in a structured layout using the command "{st pointer}" (st for status). The command "{st pointer}" sets a logical to a write command inside **effi**. With the command "te" the pointers are printed on the output window together with the lstructi values, the iss counters and the requested storage in each called subroutines. This way one can check for the pointers in the program. (see {he st} in **effi**). The space required by the subroutines called by get_theory()9.3.1 is communicated to **effi** with the first call of the set.

The fr_theory 9.2.2 has three entries: *arr0*, *arri*, *arre*. The string 'cmch' (command character) has the value *arr0* for global definitions. The integer ifrsto is used to communicate the storage requirement to **effi**. These requirements depend on the code of th_mainroutine 9.3 (i.e th_mossbauer()). To the first call belong the jump through all the subroutine tree (the nested loop , see 9.3) to read the storage requirements. At each loop index iss (*arri*) fr_theory can again ask for storage by the number ifrsto. The strings 'ci_prp(..)' (character i - property) of the common block '*common/struct/*' carries information from the subroutines. There is a subroutine
call l_structure(iss,lstruct1,lstruct2,lstruct3)
which gives back the loop indices: lstruct1,lstruct2,lstruct3 for each counter index iss by which the fr_theory is called. This information may be used to allocate on array_fr appropriate space for interim results.

The pointers li_get/put are absolute addresses on array_fr. The relative pointer addresses declared by fr_theory in a first call are changed to absolute addresses if they are stored of one of the 8 positions behind theoryname(2) on the block 'common/struct/'. Results of the theory used for the plot and the fit routine need two pointers: nstorTH,nstorTHyy. They are defined in fr_theory and stored at one of the 8 positions. The last 24 positions are used for everything, what is calculated only once and communicated to other subroutines for later calls. This group has been divided into two groups (above), 8 for communication fr constants, (ifrth(8), and further 16 constants. All constants can be declared as integer and real (indicated by a1,n2,n3,...,a10) on all 24 positions.

References

[1] H. Spiering, L. Deák and L. Bottyán, Hyperfine Interactions **125**, 197–204 (2000).

[2] E.W. Müller, MOSFUN, Internal report, Anorganische Chemie und Analytische Chemie, Johannes Gutenberg-Universität, Mainz, 1982.

[3] K. Kulcsár, D.L. Nagy and L. Pócs, A complete package of programs for the evaluation of Mssbauer and gamma spectra, In *Proc. Conf. Mössbauer Spectrometry, Dresden*, p. 594, 1971.