Millepede-II

V04-01-01

Generated by Doxygen 1.8.5

Fri Dec 13 2013 11:07:37

Contents

Chapter 1

Overview

1.1 Introduction

In certain least squares fit problems with a very large number of parameters the set of parameters can be divided into two classes, global and local parameters. Local parameters are those parameters which are present only in subsets of the data. Detector alignment and calibration based on track fits is one of the problems, where the interest is only in optimal values of the global parameters, the alignment parameters. The method, called Millepede, to solve the linear least squares problem with a simultaneous fit of all global and local parameters, irrespectively of the number of local parameters, is described in the draft manual.

The Millepede method and the initial implementation has been developed by `V. Blobel` from he University of Hamburg. Meanwhile the code is maintained at DESY by the statistics tools group of the analysis center of the Helmholtz Terascale alliance (`www.terascale.de`).

1.2 Installation

To install **Millepede** (on a linux system):

1. Download the software package from the DESY `svn` server to *target* directory, e.g.:

   ```
   svn checkout http://svnsrv.desy.de/public/MillepedeII/tags/V04-01-01 target
   ```

2. Create **Pede** executable (in *target* directory):

   ```
   make pede
   ```

3. Optionally check the installation by running the simple test case:

   ```
   ./pede -t
   ```

   This will create (and use) the necessary text and binary files.

1.3 News

- 131008: New solution method MINRES-QLP [ref 9] implemented.

1.4 Tools

The subdirectory `tools` contains some useful scripts:

- `readMilleBinary.py`: Python script to read binary files and print records in text form.
- `readPedeHists.C`: ROOT script to read and convert the **Millepede** histogram file `millepede.his`.

1.5 Details

Detailed information is available at:

Millepede II - Draft Manual

Major changes

List of options and commands

List of exit codes

1.6 Contact

For information exchange the **Millepede** mailing list `anacentre-millepede2@desy.de` should be used.

1.7 References

1. A New Method for the High-Precision Alignment of Track Detectors, Volker Blobel and Claus Kleinwort, Proceedings of the Conference on Adcanced Statistical Techniques in Particle Physics, Durham, 18 - 22 March 2002, Report DESY 02-077 (June 2002) and `hep-ex/0208021`
2. Alignment Algorithms, V. Blobel, `Proceedings` of the LHC Detector Alignment Workshop, September 4 - 6 2006, CERN
3. Software alignment for Tracking Detectors, V. Blobel, NIM A, 566 (2006), pp. 5-13, `doi:10.1016/j.nima.2006.05.157`
4. A new fast track-fit algorithm based on broken lines, V. Blobel, NIM A, 566 (2006), pp. 14-17, `doi:10.1016/j.nima.2006.05.156`
5. Millepede 2009, V. Blobel, `Contribution` to the 3rd LHC Detector Alignment Workshop, June 15 - 16 2009, CERN
6. General Broken Lines as advanced track fitting method, C. Kleinwort, NIM A, 673 (2012), pp. 107-110, `doi:10.1016/j.nima.2012.01.024`
7. Volker Blobel und Erich Lohrmann, Statistische und numerische Methoden der Datenanalyse, Teubner Studienbücher, B.G. Teubner, Stuttgart, 1998. `Online-Ausgabe`.
8. `Systems Optimization Laboratory`, Stanford University;

   C. C. Paige and M. A. Saunders (1975), Solution of sparse indefinite systems of linear equations, SIAM J. Numer. Anal. 12(4), pp. 617-629.
9. `Systems Optimization Laboratory`, Stanford University;

   Sou-Cheng Choi, Christopher Paige, and Michael Saunders, MINRES-QLP: A Krylov subspace method for indefinite or singular symmetric systems, SIAM Journal of Scientific Computing 33:4, 1810-1836, 2011, `doi:10.1137/100787921`

Chapter 2

Millepede II - Draft Manual

Linear Least Squares Fits with a Large Number of Parameters

Author

V. Blobel, University Hamburg, 2007

Remarks

Adapated to formatting with Doxygen (C. Kleinwort, DESY, 2012). The abstract has been moved to section Introduction. Some clarifications as result of user feedback included (C. Kleinwort, DESY, 2013).

2.1 Preface

Linear least squares problems. The most important general method for the determination of parameters from measured data is the linear least squares method. It is usually stable and accurate, requires no initial values of parameters and is able to take into account all the correlations between different parameters, even if there are many parameters.

Global and local parameters. The parameters of a least squares problem can sometimes be distinguished as *global* and *local* parameters. The mathematical model underlying the measurement depends on both types of parameters. The interest is in the determination of the global parameters, which appear in all the measurements. There are many sets of local parameters, where each set appears only in one, sometimes small, subset of measurements.

Alignment of large detectors in particle physics. Alignment problems for large detectors in particle physics often require the determination of a large number of alignment parameters, typically of the order of 100 to $1\,000$, but sometimes above $10\,000$. Alignment parameters for example define the accurate space coordinates and orientation of detector components. In the alignment usually special alignment measurements are combined with data of particle reaction, typically tracks from physics interactions and from cosmics. In this paper the alignment parameters are called *global* parameters. Parameters of a single track like track slopes and curvatures are called *local* parameters.

One approximate alignment method is to perform least squares fits on the data e.g. of single tracks assuming fixed alignment parameters. The *residuals*, the *deviations* between the fitted and measured data, are then used to estimate the alignment parameters afterwards. Single fits depend only on the small number of local parameters like slope or curvature of the track, and are easy to solve. This approximate method however is not a correct method, because the (local) fits depend on a wrong model, they ignore the global parameters, and the result are biased fits. The adjustment of parameters based on the observed (biased) residuals will then result in biased alignment parameters. If these alignment parameters are applied as corrections in repeated fits, the remaining residuals will be reduced, as desired. However, the fitted parameters will still be biased. In practice this procedure is often applied iteratively; it is however not clear wether the procedure is converging.

A more efficient and faster method is an overall least squares fit, with all the global parameters and local parameters, perhaps from thousands or millions of events, determined simultaneously. The **Millepede** algorithm makes use

of the special structure of the least squares matrices in such a simultaneous fit: the global parameters can be determined from a matrix equation, where the matrix dimension is given by the number of global parameters only, irrespective of the total number of local parameters, and without any approximation. If n is the number of global parameters, then an equation with a symmetric n-by- n matrix has to be solved. The **Millepede** programm has been used for up to $n \approx 5000$ parameters in the past. Solution of the matrix equation was done with a fast program for the solution of matrix equations with inversion of the symmetric matrix, which on a standard PC takes a time $t \approx 2 \times 10^{-8}\,\mathrm{sec} \times n^3$. Even for $n = 5000$ the computing time is, with $t = 2 \times 10^{-8}\,\mathrm{sec} \times 5000^3 = 40$ minutes, still acceptable.

The next-generation detectors in particle physics however require up to about 100000 global parameters and with the previous solution method the space- and time-consumption is too large for a standard PC. Memory space is needed for the full symmetric matrix, corresponding to $1/2n^2 \times 8$ bytes for double precision, if the symmetry of the matrix is observed, and if solution is done *in-space*. This means memory space of 400 Mbyte and 40 Gbyte for $n = 10\,000$ and $n = 100\,000$, and computing times of about 6 hours and almost a year, respectively.

Millepede I and II. The second version of the **Millepede** programm, described here, allows to use several different methods for the solution of the matrix equation for global parameters, while keeping the algorithm, which decouples the determination of the local parameters, i.e. still taking into account all correlations between all global parameters. The matrix can be stored as a sparse matrix, if a large fraction of off-diagonal elements has zero content. A complete matrix inversion of a large sparse matrix would result in a full matrix, and would take an unacceptable time. Different methods can be used in **Millepede II** for a sparse matrix. The symmetric matrix can be accumulated in a special sparse storage scheme. Special solution methods can be used, which require only products of the symmetric sparse matrix with different vectors. Depending on the fraction of vanishing matrix elements, solutions for up to a number of 100000 global parameters should be possible on a standard PC.

The structure of **Millepede II** is different from the **Millepede I** version. The accumulation of data and the solution are splitted, with the solution in a stand-alone program. Furthermore the global parameters are now characterized by a label (any positive integer) instead of a (continuous) index. Those features should simplify the application of the program for alignment problems even for large number of global parameters in the range of 10^5.

2.2 Mathematical Methods

2.2.1 Large problems with global and local parameters

The solution of optimization problems requires the determination of certain parameters. In certain optimization problems the parameters can be classified as either *global* or *local*. This classification can be used, when the input data of the optimization problem consist out of a potentially large group of data sets, where the description of each single set of data requires certain *local* parameters, which appear only in the description of one data set. The *global* parameter may appear in all data sets, and the interest is in the determination of these global parameters. The local parameters can be called nuisance parameters.

An example for an optimization problem with global and local parameters from experimental high energy physics is the alignment of track detectors using a large number of track data. The track data are position measurements of a charged particle track, which can be fitted for example by a helix with five parameters, if the track detector is in a homogeneous magnetic field. The position measurement, from several detector planes of the track detector, depend on the position and orientation of certain sensors, and the corresponding coordinates and angles are global parameters. In an alignment the values of the global parameters are improved by minimizing the deviations between measurement and parametrization of a large number of tracks. Numbers of tracks in the order of one Million, with of the order of 10 data poinst per track, and of 1000 to $100,000$ global parameters are typical for modern track detectors in high energy physics.

Optimal values of the global parameters require a simultaneous fit of all parameters with all data sets. A straightforward ansatz for such a fit seems to require to fit Millions of parameters, which is impossible. An alternative is to perform each local fit separately, fitting the local parameters only. From the residuals of the local fits one can then try to fit the values of the global parameters. This approach neglects the correlations between the global and local parameters. Nevertheless the method can be applied iteratively with the hope, that the convergence is not to slow and does not require too many iterations.

The optimization problem is complicated by the fact, that often not all global parameters are defined by the ansatz. In the alignment example the degrees of freedom describing a translation and rotation of the whole detector are

not fixed. The solution of the optimization problem requires therefore certain equality constraints, for example zero overall translation and rotation of the detector; these constraints are described by a linear combination of global parameters. Such an equality constraint can only be satisfied in an overall fit, not with separated local and global fits.

Note

Constraints in optimization are either equality or inequality constraints, which have exactly to be taken into account in the solution. The term *constraint* is often used in a loose sense, like: "the parameters are constrained by our measurement".

Global parameters are denoted by the vector $\boldsymbol{p}$ and local parameters for the data set j by the vector $\boldsymbol{q}_j$. The objective function to be minimized in the global parameter optimization is the sum of the local objective Function, depending on the global parameters $\boldsymbol{p}$ and the vectors $\boldsymbol{q}_j$:

$$F(\boldsymbol{p},\boldsymbol{q}) = \sum_j F_j(\boldsymbol{p},\boldsymbol{q}_j)$$

In the least squares method the objective function to be minimized is a sum of the squares of residuals z_i between the measured values and the parametrization, weighted by the inverse variance of the measured value:

$$F_j(\boldsymbol{p},\boldsymbol{q}_j) = \tfrac{1}{2}\sum_i \frac{z_i^2}{\sigma_i^2}$$

with the residual $z_i = y_i - f_i(\boldsymbol{p},\boldsymbol{q}_j)$, where y_i is the measured value and $f_i(\boldsymbol{p},\boldsymbol{q}_j)$ is the corresponding parametrization. This form of the objective function assumes independent measurements, with a single variance value σ_i assigned.

Note

In experimental high energy physics the objective function is usually called a χ^2-function. In statistics there is a χ^2-distribution with a well-defined meaning. The minimum value $\times 2$ of the objective function follows, under certain conditions, the χ^2-distribution.

2.2.2 Optimization without constraints

2.2.2.1 The quadratic model

The standard optimization method for smooth objective functions is the **Newton** method. The objective function $F(\boldsymbol{p})$, depending on a parameter vector $\boldsymbol{p}$, is approximated by a quadratic model $\tilde{F}(\boldsymbol{p})$

$$\tilde{F}_k(\boldsymbol{p}_k + \boldsymbol{d}) = F_k + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{d} + \tfrac{1}{2}\boldsymbol{d}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{d}$$

where $\boldsymbol{p}_k$ is the vector $\boldsymbol{p}$ in the k-th iteration and where the vector $\boldsymbol{g}$ is the gradient of the objective function; the matrix $\boldsymbol{C}$ is the Hessian (second derivative matrix) of the objective function $F(\boldsymbol{p})$ or an approximation of the Hessian. The minimum of the quadratic approximation requires the gradient to be equal to zero. A step $\boldsymbol{d}$ in parameter space is calculated by the solution of the matrix equation, obtained from the derivative of the quadratic model:

(1)

$$\boldsymbol{C}\,\boldsymbol{d} = -\boldsymbol{g}\,.$$

With the correction vector $\boldsymbol{d}$ the new value $\boldsymbol{p}_{k+1} = \boldsymbol{p}_k + \boldsymbol{d}$ for the next iteration is obtained. The matrix $\boldsymbol{C}$ is a constant in a *linear least squares* problem and the minimum is determined in a single step (no iteration necessary).

2.2.2.2 Partitioning of matrices

The special structure of the matrix $\boldsymbol{C}$ in a matrix equation $\boldsymbol{C}\boldsymbol{d} = -\boldsymbol{g}$ may allow a significant simplification of the solution. Below the symmetric matrix $\boldsymbol{C}$ is partitioned into submatrices, and the vectors $\boldsymbol{d}$ and $\boldsymbol{g}$ are partitioned into

two subvectors; then the matrix equation can be written in the form

$$\left(\begin{array}{c|c} \boldsymbol{C_{11}} & \boldsymbol{C_{21}^{\mathrm{T}}} \\ \hline \boldsymbol{C_{21}} & \boldsymbol{C_{22}} \end{array}\right) \left(\begin{array}{c} \boldsymbol{d_1} \\ \hline \boldsymbol{d_2} \end{array}\right) = - \left(\begin{array}{c} \boldsymbol{g_1} \\ \hline \boldsymbol{g_2} \end{array}\right) ,$$

where the submatrix $\boldsymbol{C_{11}}$ is a p-by- p square matrix and the submatrix $\boldsymbol{C_{22}}$ is a q-by- q square matrix, with $p+q=n$, and $\boldsymbol{C_{21}}$ is a q-by- p matrix. Now it is assumed that the inverse of the q-by- q sub-matrix $\boldsymbol{C_{22}}$ is available. In certain problems this may be easily calculated, for example if $\boldsymbol{C_{22}}$ is diagonal.

If the sub-vector $\boldsymbol{d_1}$ would not exist, the solution for the sub-vector $\boldsymbol{d_2}$ would be defined by the matrix equation $\boldsymbol{C_{22}}\,\boldsymbol{d_2^*} = -\boldsymbol{g_2}$, where the star indicates the special character of this solution, which is

(2)
$$\boldsymbol{d_2}^* = -\boldsymbol{C_{22}^{-1}}\,\boldsymbol{g_2} .$$

Now, having the inverse sub-matrix $\boldsymbol{C_{22}^{-1}}$, the submatrix of the complete inverse matrix $\boldsymbol{C}$ correponding to the upper left part $\boldsymbol{C_{11}}$ is the inverse of the symmetric p-by- p matrix

(3)
$$\boldsymbol{S} = \boldsymbol{C_{11}} - \boldsymbol{C_{21}^{\mathrm{T}}}\boldsymbol{C_{22}^{-1}}\boldsymbol{C_{21}} ,$$

the so-called *Schur complement*. With this matrix $\boldsymbol{S}$ the solution of the whole matrix equation can be written in the form

$$\left(\begin{array}{c} \boldsymbol{d_1} \\ \hline \boldsymbol{d_2} \end{array}\right) = - \left(\begin{array}{c|c} \boldsymbol{S^{-1}} & \boldsymbol{-S^{-1}C_{21}^{\mathrm{T}}C_{22}^{-1}} \\ \hline \boldsymbol{-C_{22}^{-1}C_{21}S^{-1}} & \boldsymbol{C_{22}^{-1}-C_{22}^{-1}C_{21}S^{-1}C_{21}^{\mathrm{T}}C_{22}^{-1}} \end{array}\right) \left(\begin{array}{c} \boldsymbol{g_1} \\ \hline \boldsymbol{g_2} \end{array}\right) .$$

The sub-vector $\boldsymbol{d_1}$ can be obtained from the solution of the matrix equation

(4)
$$\left(\boldsymbol{C_{11}} - \boldsymbol{C_{21}^{\mathrm{T}}}\boldsymbol{C_{22}^{-1}}\boldsymbol{C_{21}} \right) \left(\boldsymbol{d_1} \right) = - \left(\boldsymbol{g_1} - \boldsymbol{C_{21}^{\mathrm{T}}}\boldsymbol{C_{22}^{-1}}\boldsymbol{g_2} \right) = - \left(\boldsymbol{g_1} - \boldsymbol{C_{21}^{\mathrm{T}}}\boldsymbol{d_2}^* \right)$$

using the known right-hand-side of the equation with the special solution $\boldsymbol{d_2}^*$.

In a similar way the vector $\boldsymbol{d_2}$ could be calculated. However, if the interest is the determination of this sub-vector $\boldsymbol{d_1}$ only, while the sub-vector $\boldsymbol{d_2}$ is not needed, then only the equation (4) has to be solved after calculation of the special solution $\boldsymbol{d_2}^*$ (equation (2)) and the Schur complement $\boldsymbol{S}$ (equation (3)). Some computer time can be saved by this method, especially if the matrix $\boldsymbol{C_{22}^{-1}}$ is easily calculated or already known before; note that the matrix $\boldsymbol{C_{22}}$ does not appear directly in the solution, only the inverse $\boldsymbol{C_{22}^{-1}}$.

This method of removing unnecessary parameters was already known in the nineteenth century. The method can be applied repeatedly, and therefore may simplify the solution of problems with a large number of parameters. The method is not an approximation, but is exact and it takes into account all the correlations introduced by the removed parameters.

Note

One example is: Schreiber, O. (1877): Rechnungsvorschriften für die trigonometrische Abteilung der Landesaufnahme, Ausgleichung und Berechnung der Triangulation zweiter Ordnung. Handwritten notes. Mentioned in W. Jordan (1910): Handbuch der Vermessungskunde, Sechste erw. Auflage, Band I, Paragraph III: 429-433. J.B.Metzler, Stuttgart.

2.2.2.3 Local parameters

A set of local measured data y_i is considered. The local data y_i are assumed to be described by a linear or non-linear function $f(x_i, \boldsymbol{q})$, depending on a (small) number of local parameters $\boldsymbol{q}$.

$$y_i = f(x_i, \boldsymbol{q}) + \varepsilon_i .$$

The parameters $\boldsymbol{q}$ are called the local parameters, valid for the specific group of measurements (local-fit object). The quantity ε is the measurement error, with standard deviation σ_i. The quantity x_i is assumed to be the coordinate of the measured value y_i, and is one argument of the function $f(x_i, \boldsymbol{q})$.

The local parameters are determined in a least squares fit. If the function $f(x_i, \boldsymbol{q})$ depends *non-linearly* on the local parameters $\boldsymbol{q}$, an iterative procedure is used, where the function is linearized, i.e. the *first* derivatives of the function $f(x_i, \boldsymbol{q})$ with respect to the local parameters $\boldsymbol{q}$ are calculated. The function is thus expressed as a linear function of local parameter corrections $\boldsymbol{\Delta q}$ at some reference value $\boldsymbol{q}_k$:

(5)
$$f(x_i, \boldsymbol{q}_k + \boldsymbol{\Delta q}) = f(x_i, \boldsymbol{q}_k) + \frac{\partial f}{\partial q_1}\Delta q_1 + \frac{\partial f}{\partial q_2}\Delta q_2 + \dots ,$$

where the derivatives are calculated for $\boldsymbol{q} \equiv \boldsymbol{q}_k$. For each single measured value, the residual measurement z_i

$$z_i \equiv y_i - f(x_i, \boldsymbol{q}_k)$$

is calculated. For each iteration a linear system of equations (normal equations of least squares) has to be solved for the parameter corrections $\boldsymbol{\Delta q}$ with a matrix $\boldsymbol{\Gamma}$ and a gradient vector $\boldsymbol{g}$ with elements

(6)
$$\Gamma_{jk} = \sum_i \left(\frac{\partial f_i}{\partial q_j}\right)\left(\frac{\partial f_i}{\partial q_k}\right)\frac{1}{\sigma_i^2} \qquad \beta_j = \sum_i \left(\frac{\partial f_i}{\partial q_j}\right)\frac{z_i}{\sigma_i^2} ,$$

where the sum is over all measurements y_i of the local-fit object. Corrections $\boldsymbol{\Delta q}$ are determined by the solution of the matrix equation

$$\boldsymbol{\Gamma \Delta q} = -\boldsymbol{\beta} ,$$

and a new reference value is obtained by

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \boldsymbol{\Delta q}$$

and then, with k increased by 1, this is repeated until convergence is reached.

2.2.2.4 Global parameters

Now global parameters are considered, which contribute to all the measurements. The expectation function (5) is extended to include corrections for global parameters. Usually only few of the global parameters influence a local-fit object. A global parameter is identified by a label ℓ; assuming that labels ℓ from a set Ω contribute to a single measurement the extended equation becomes

$$z_i = y_i - f(x_i, \boldsymbol{q}, \boldsymbol{p}) = \sum_{j=1}^{\nu}\left(\frac{\partial f}{\partial q_j}\right)\Delta q_j + \sum_{\ell\in\Omega}\left(\frac{\partial f}{\partial p_\ell}\right)\Delta p_\ell .$$

2.2.2.5 The simultaneous fit of global and local parameters

In the following it is assumed that there is a set of N local measurements. Each local measurement, with index j, depends on ν local parameters $\boldsymbol{q}_j$, and all of them depend on the global parameters. In a simultaneous fit of all global parameters plus local parameters from N subsets of the data there are in total $(n + N\cdot\nu)$ parameters, and the standard solution requires the solution of $(n + N\cdot\nu)$ equations with a computation proportional to $(n + N\cdot\nu)^3$. In the next chapter it is shown, that the problem can be reduced to a system of n equations, for the global parameters only.

For a set of N local measurements one obtains a system of least squares normal equations with large dimensions, as is shown in equation (7) The matrix on the left side of equation (7) has, from each local measurement, three types of contributions. The first part is a contribution of a symmetric matrix $\boldsymbol{C_1}_j$, of dimension n (number of global parameters), and is calculated from the (global) derivatives $\partial f/\partial p_\ell$. All the matrices $\boldsymbol{C_1}_j$ are added up in the upper left corner of the big matrix of the normal equations. The second contribution is the symmetric matrix $\boldsymbol{\Gamma}_j$ (compare equation (6)), which gives a contribution to the big matrix on the diagonal and is depending only on the j-th local measurement and the (local) derivatives $\partial f/\partial q_j$. The third (mixed) contribution is a rectangular matrix $\boldsymbol{G}_j$, with a

row number of n (global) and a column number of ν (local). There are two contributions to the vector of the normal equations (gradient), $\boldsymbol{g_{1}}_j$ for the global and $\boldsymbol{\beta}_j$ for the local parameters. The complete matrix equation is given by

(7)

$$\left(\begin{array}{c|c||c|c} \sum \boldsymbol{C_{1}}_j & \cdots & \boldsymbol{G}_j & \cdots \\ \hline\hline \vdots & \ddots & 0 & 0 \\ \hline \boldsymbol{G}_j^{\mathrm{T}} & 0 & \boldsymbol{\Gamma}_j & 0 \\ \hline \vdots & 0 & 0 & \ddots \end{array}\right) \cdot \left(\begin{array}{c} \boldsymbol{d} \\ \hline\hline \vdots \\ \hline \boldsymbol{\Delta q}_j \\ \hline \vdots \end{array}\right) = - \left(\begin{array}{c} \sum \boldsymbol{g_{1}}_j \\ \hline\hline \vdots \\ \hline \boldsymbol{\beta}_j \\ \hline \vdots \end{array}\right)$$

In this matrix equation the matrices $\boldsymbol{C_{1}}_j$, $\boldsymbol{\Gamma}_j$, $\boldsymbol{G}_j$ and the vectors $\boldsymbol{g_{1}}_j$ and $\boldsymbol{\beta}_j$ contain contributions from the j-th local measurement. Ignoring the global parameters (i.e. keeping them constant) one could solve the normal equations $\boldsymbol{\Gamma}_j \boldsymbol{\Delta q}_j^* = -\boldsymbol{\beta}_j$ for each local measurement separately by

$$\boldsymbol{\Delta q}_j^* = -\boldsymbol{\Gamma}_j^{-1} \boldsymbol{\beta}_j \,.$$

The complete system of normal equations has a special structure, with many vanishing sub-matrices. The only connection between the local parameters of different partial measurements is given by the sub-matrices $\boldsymbol{G}_j$ und $\boldsymbol{C_{1}}_j$,

2.2.2.6 Reduction of matrix size

The aim of the fit is solely to determine the global parameters; final best parameters of the local parameters are not needed. The matrix of equation (7) is written in a partitioned form. The general solution can also be written in partitioned form. Many of the sub-matrices of the huge matrix in equation (7) are zero and this has the effect, that the formulas for the sub-matrices of the inverse matrix are very simple.

By this procedure the n normal equations

(8)

$$\left(\begin{array}{c} \\ \boldsymbol{C} \\ \\ \end{array}\right) \left(\begin{array}{c} \\ \boldsymbol{d} \\ \\ \end{array}\right) = - \left(\begin{array}{c} \\ \boldsymbol{g} \\ \\ \end{array}\right) \,,$$

are obtained, which only contain the global parameters, with a modified matrix $\boldsymbol{C}$ and a modified vector $\boldsymbol{g}$,

$$\boldsymbol{C} = \sum_j \boldsymbol{C_{1}}_j + \sum_j \boldsymbol{C_{2}}_j \qquad \boldsymbol{g} = \sum_j \boldsymbol{g_{1}}_j + \sum_j \boldsymbol{g_{2}}_j$$

with the following local contributions to $\boldsymbol{C}$ and $\boldsymbol{g}$ from the j-th local fit:

(9)

$$\boldsymbol{C_{2}}_j = -\boldsymbol{G}_j \boldsymbol{\Gamma}_j^{-1} \boldsymbol{G}_j^{\mathrm{T}} \qquad \boldsymbol{g_{2}}_j = -\boldsymbol{G}_j \left(\boldsymbol{\Gamma}_j^{-1} \boldsymbol{\beta}_j\right) = -\boldsymbol{G}_j \boldsymbol{\Delta q}_j^* \,.$$

The set of normal equations (8) contains explicitly only the global parameters; implicitly it contains, through the correction matrices, the complete information from the local parameters, influencing the fit of the global parameters. The parentheses in equation (9) represents the solution for the local parameters, ignoring the global parameters. The solution

$$\boldsymbol{d} = -\boldsymbol{C}^{-1} \boldsymbol{g}$$

represents the solution vector $\boldsymbol{d}$ with covariance matrix $\boldsymbol{C}^{-1}$. The solution is direct, no iterations or approximations are required. The dimension of the matrix to compute $\boldsymbol{d}$ from equation (1) is reduced from $(n + N \cdot \nu)$ to n. The vector $\boldsymbol{d}$ is the correction for the global parameter vector $\boldsymbol{p}$. Iterations may be necessary for other reasons, namely

- the equations depend *non-linearly* on the global parameters; the equations have to be linearized;
- the data contain outlier, which have to be removed in a sequence of cuts, becoming narrower during the iteration, or which have to be down-weighted;
- the accuracy of the data is not known before, and has to be determined from the data (after the alignment).

For iterations the vector $\boldsymbol{d}$ is the correction for the global parameter vector $\boldsymbol{p}_k$ in iteration k to obtain the global parameter vector $\boldsymbol{p}_{k+1} = \boldsymbol{p}_k + \boldsymbol{d}$ for the next iteration.

2.2.2.7 Nonlinear least squares

A method for *linear* least squares fits with a large number of parameters, perhaps with *linear* constraints, is discussed in this paper. Sometime of course the model is *nonlinear* and also constraints may be nonlinear. The standard method to treat these problems is linearization: the nonlinear equation is replaced by a linear equation for the correction of a parameter (Taylor expansion); this requires a good approximate value of the parameter. In principle this method requires an iterative improvement of the parameters, but sometimes even one iteration may be sufficient.

The treatment of nonlinear equations is not directly supported by the program package, but it will in general not be too difficult to organize a program application with nonlinear equations.

2.2.2.8 Outliers

Cases with very large residuals within the data can distort the result of the method of least squares. In the method of M-estimates, the least squares method is modified to Maximum likelihood method. Basis is the residual between data and function value (expected data value), normalized by the standard deviation:

$$\zeta = \frac{y - f(x)}{\sigma}$$

In the method of M-estimates the objective function $F(.)$ to be minimized is defined in terms of a probability density function of the normalized residual

$$F(.) = \sum_i \rho(\zeta_i) \qquad \rho(\zeta) = \ln \mathrm{pdf}(\zeta)$$

From the probability density function $\mathrm{pdf}(\zeta)$ a *influence function* $\psi(\zeta)$ is defined

$$\text{influence function } \psi(\zeta) = \mathrm{d}\rho(\zeta)/\mathrm{d}\zeta \qquad \text{additional weight factor } \omega(\zeta) = \psi(\zeta)/\zeta$$

and a weight in addition to the normal least squares weight $w_i = 1/\sigma_i^2$ can be derived from the influence function for the calculation of the (modified) normal equations of least squares.

For the standard least squares method the function $\rho(\zeta)$ is simply $\rho(\zeta) = \zeta^2/2$ and it follows, that the influence function is $\psi(\zeta) = \zeta$ and the weight factor is $\omega(\zeta) = 1$ (i.e. no extra weight). The influence function value increases with the value of the normalized residual without limits, and thus outliers have a large and unlimited influence. In order to reduce the influence of outliers, the probability density function $\mathrm{pdf}(\zeta)$ has to be modified for large values of $|\zeta|$ to avoid the unlimited increase of the influence. Several functions $\mathrm{pdf}(\zeta)$ are proposed, for example the Huber function and the Cauchy function.

Huber function: A simple function is the Huber function, which is quadratic and thus identical to least squares for small $|\zeta|$, but linear for larger $|\zeta|$, where the influence function becomes a constant $C_\mathrm{H} \cdot \mathrm{sign}(\zeta)$:

$$\text{Huber function: pdf} \quad \rho(\zeta) = \begin{cases} \zeta^2/2 \\ C_\mathrm{H}\,(|\zeta| - C_\mathrm{H}/2) \end{cases} \qquad \text{factor} \quad \omega(\zeta) = \begin{cases} 1 & \text{if } |\zeta| \le C_\mathrm{H} \\ C_\mathrm{H}/|\zeta| & \text{if } |\zeta| > C_\mathrm{H} \end{cases}$$

The extra weight is $\omega(\zeta) = C_\mathrm{H}/|\zeta|$ for large $|\zeta|$. A standard value for C_H is $C_\mathrm{H} = 1.345$; for this value the efficiency for Gaussian data without outliers is still 95 %. For very large deviations the additional weight factor decreases with $1/|\zeta|$.

Cauchy function: For small deviation the Cauchy function is close to the least squares expression, but for large deviations it increases only logarithmically. For very large deviations the additional weight factor decreases with $1/\zeta^2$:

$$\text{Cauchy function: pdf} \quad \rho(\zeta) = \tfrac{1}{2} C_\text{c} \ln\left(1+(\zeta/C_\text{c})^2\right) \qquad \text{factor} \quad \omega = 1/\left(1+(\zeta/C_\text{c})^2\right)$$

A standard value is $C_\text{c} = 2.3849$; for this value the efficiency for Gaussian data without outliers is still 95 %.

2.2.3 Optimization with linear constraints

The minimization of an objective function $F(\boldsymbol{p})$ is often not sufficient. Several degrees of freedom may be undefined and require additional conditions, which can be expressed as equality constraints. For m linear equality constraints the problem is the minimization of a non-linear function $F(\boldsymbol{p})$ subject to a set of linear constraints:

$$\min F(\boldsymbol{p}) \qquad \text{subject to } \boldsymbol{A}\boldsymbol{p} = \boldsymbol{c}\,,$$

where $\boldsymbol{A}$ is a m-by- n matrix and $\boldsymbol{c}$ is a m-vector with $m \leq n$. In iterative methods the parameter vector $\boldsymbol{p}$ is expressed by $\boldsymbol{p} = \boldsymbol{p}_k + \boldsymbol{d}$ with the correction $\boldsymbol{d}$ to $\boldsymbol{p}_k$ in the k-th iteration, satisfying the equation

$$\boldsymbol{A}(\boldsymbol{p}_k + \boldsymbol{d}) = \boldsymbol{c}$$

with $\boldsymbol{p}_{k+1} = \boldsymbol{p}_k + \boldsymbol{d}$.

There are two methods for linear constrainst:

- in the *Lagrange multiplier method* additional m parameters are introduced and the linear system of $n+m$ unknowns has to be solved;
- by *elimination* the minimization problem with constraints is transformed to an unconstrained problem with $n-m$ unknowns. However the sparsity of a matrix may be destroyed by the elimination.

The Lagrange method is used in **Millepede**.

2.2.3.1 The Lagrange multiplier method

In the Lagrange multiplier method one additional parameter λ is introduced for each single constraint, resulting in an m-vector $\boldsymbol{\lambda}$ of Lagrange multiplier. A term depending on $\boldsymbol{\lambda}$ and the constraints is added to the function $F(\boldsymbol{p})$, resulting in the Lagrange function

$$\mathscr{L}(\boldsymbol{p}, \boldsymbol{\lambda}) = F(\boldsymbol{p}) + \boldsymbol{\lambda}(\boldsymbol{A}\boldsymbol{p} - \boldsymbol{c})$$

Using as before a quadratic model for the function $F(\boldsymbol{p})$ and taking derivatives w.r.t. the parameters $\boldsymbol{p}$ and the Lagrange multipliers $\boldsymbol{\lambda}$, the two equations

$$\begin{aligned} \boldsymbol{C}\boldsymbol{d} + \boldsymbol{A}^\text{T}\boldsymbol{\lambda} &= -\boldsymbol{g} \\ \boldsymbol{A}\boldsymbol{d} &= \boldsymbol{c} - \boldsymbol{A}\boldsymbol{p}_k \end{aligned}$$

are obtained; the second of these equations is the constraint equation. This system of two equations can be combined into one matrix equation

(10)

$$\left(\begin{array}{c|c} \boldsymbol{C} & \boldsymbol{A}^\text{T} \\ \hline \boldsymbol{A} & \boldsymbol{0} \end{array}\right) \left(\begin{array}{c} \boldsymbol{d} \\ \hline \boldsymbol{\lambda} \end{array}\right) = \left(\begin{array}{c} -\boldsymbol{g} \\ \hline \boldsymbol{c} - \boldsymbol{A}\boldsymbol{p}_k \end{array}\right) .$$

The matrix on the left hand side is still symmetric. *Linear* least squares problems with *linear* constraints can be solved directly, without iterations and without the need for initial values of the parameters.

The matrix in equation (10) is indefinite, with positive and negative eigenvalues. A solution can be found even if the submatrix $\boldsymbol{S}$ is singular, if the matrix $\boldsymbol{A}$ of the constraints supplies sufficient information. Because of the different signs of the eigenvalues the stationary solution is not a minimum of the function $\mathscr{L}(\boldsymbol{p}, \boldsymbol{\lambda})$.

2.2.3.2 Feasible parameters

Feasible parameters. A particular value for the correction $\boldsymbol{d}$ can be calculated by

(11)
$$\boldsymbol{d} = \boldsymbol{A}^{\mathrm{T}} \left(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\right)^{-1} (\boldsymbol{c} - \boldsymbol{A}\boldsymbol{p}_k) \ ,$$

which is the *minimum-norm solution* of the constraint equation, that is, the solution of

$$\min \ \|\boldsymbol{A}\,(\boldsymbol{p}_k + \Delta\boldsymbol{p}) - \boldsymbol{c}\|_2 \ ,$$

which is zero here. The matrix $\boldsymbol{A}$ is a m-by- n matrix for m constraints and the product $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ is a square m-by- m matrix, which has to be inverted in equation (11), which allows to obtain a correction such that the linear constraints are satisfied. Parameter vectors $\boldsymbol{p}$, satisfying the linear constraint equations $\boldsymbol{A}\boldsymbol{p} = \boldsymbol{c}$, are called *feasible*. If the vector $\boldsymbol{p}_k$ in the k-th iteration already satisfies the linear constraint equations, then the correction $\boldsymbol{d}$ has to have the property $\boldsymbol{A}\,\boldsymbol{d} = \boldsymbol{0}$.

2.3 The Manual

2.3.1 The programm package Millepede II

The second version of the program package with the name **Millepede** (german: *Tausendfüssler*) is based on the same mathematical principle as the first version; global parameters are determined in a simultaneous fit of global and local parameters. In the first version the solution of the matrix equation for the global parameter corrections was done by matrix inversion. This method is adequate with respect to memory space and execution time for a number of global parameters of the order of 1000. In actual applications e.g. for the alignment of track detectors at the LHC storage ring at CERN however the number of global parameters is much larger and may be above 10^5. Different solution methods are available in the **Millepede** II version, which should allow the solution of problems with a large number of global parameters with the memory space, available in standard PCs, and with an execution time of hours. The solution methods differ in the requirements of memory space and execution time.

The structure of the second version **Millepede II** can be visualized as the *decay* of the single program **Millepede** into two parts, a part **Mille** and a part **Pede** (Figure 1):

$$\text{MILLEPEDE} \Rightarrow \text{MILLE} + \text{PEDE} \ .$$

The first part, **Mille**, is a short subroutine, which is called in user programs to write data files for **Millepede II**. The second part, **Pede**, is a stand-alone program, which requires data files and text files for the steering of the solution. The result is written to text files.

Figure 1

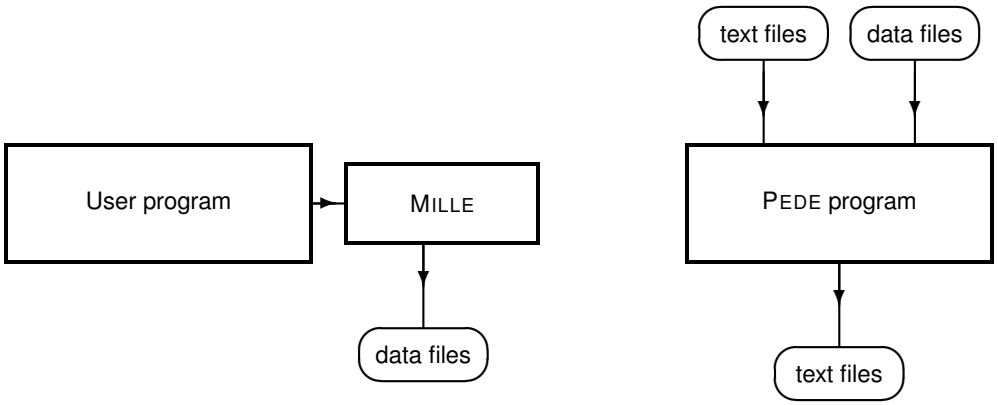

Figure 1: The subprogram MILLE (left), called inside the user program, and the stand-alone program PEDE (right), with the data flow from text and data files to text files.

2.3.1.1 Program code and makefile

The complete program is on a (`tar`)-file `Mptwo.tgz`, which is expanded by the command

```
tar -xzf Mptwo.tgz
```

into the actual directory. A `Makefile` for the program *Pede* is included; it is invoked by the

```
make
```

command. There is a test mode **Pede** (see section Solution with the stand-alone program Pede), selected by

```
./pede -t
```

which can be used to test the program installation.

The computation of the solution for a large number of parameter requires large vector and matrix arrays. Most of the memory space is required in nearly all solution methods for a single matrix. The solution of the corresponding matrix equation is done *in-space* for almost all methods (no second matrix or matrix copy is needed). The total space of all data arrays, used in **Pede**, is defined as a dimension parameter within the include file `dynal.inc` by the statement

```
PARAMETER       (MEGA=100 000 000) ! 100 mio words}
```

corresponding to the memory space allowed by a 512 Mbyte memory. Larger memories allow an increase of the dimension parameter in this statement in file `dynal.inc`.

Memory space above 2 Gbyte (?) can not be used with 32-bit systems, but on 64-bit systems; a small change in the makefile to allow linking with a big static array (see file `dynal.inc`) may be necessary (see comment in makefile).

2.3.1.2 Data collection in the user program with subroutine Mille

Data files are written within the user program by the subroutine `MILLE`, which is available in Fortran and in C. Data on the measurement and on derivatives with respect to local and global parameters are written to a binary file. The file or several files are the input to the stand-alone program **Pede**, which performs the fits and determines the global parameters. The data required for **Millepede** and the data collection in the user program are discussed in detail in section Measurements and parameters.

2.3.1.3 Solution with the stand-alone program Pede

The second part, **Pede**, is a *stand-alone program*, which performs the fits and determines the global parameters. It is written in Fortran. Input to **Pede** are the binary (unformatted) data files, written using subroutine **Mille**, and text (formatted) files, which supply steering information and, optionally, data about initial values and status of global parameters. Different binary and text files can be combined.

Synopsis:

```
pede [options] [main steering text file name]
```

The following options are implemented:

- `-i` interactive mode
- `-t` test mode
- `-s` subito option: stop after one data loop

Option i. The interactive mode allows certain interactive steering, depending on the selected method.

Option t. In the test mode no user input files are required. This mode is recommended to learn about the properties of **Millepede**. Data files are generated by Monte Carlo simulation for a simple 200-parameter problem, which

is subsequently solved. The file generated are: `mp2str.txt` (steering file), `mp2con.txt` (constraints) and `mp2tst.bin` (datafile). The latter file is always (re-)created, the two text files are created, if they do not exit. Thus one can edit these files after a first test job in order to test different methods.

Option s. In the *subito* mode, the **Pede** program stops after the first data loop, irrespective of the options selected in the steering text files.

Text files. Text files should have either the characters `xt` or `tx` in the 3-character filename-extension. At least one text file is necessary (the default name `steer.txt` is assumed, if no filename is given in the command line), which specifies at least the names of data files. The text files are described in detail in section Text files.

Pede generates, depending on the selected method, several output text files:

```
millepede.log !  log-file for pede execution
mpgparm.txt   !  global parameter results
mpdebug.txt   !  debug output for selected events
mpeigen.txt   !  selected eigenvectors
```

Existing files of the given names are renamed with a $\sim$ extension, existing files with the $\sim$ extension are removed.

2.3.2 Measurements and parameters

2.3.2.1 Measurements and local parameters

Basic data elements are single measurements y_i; several single measurements belong to a group of measurements, which can be called a local-fit object. For example in a track-based alignment based on tracks a track is a local-fit object (see section Application: Alignment of tracking detectors for a detailed discussion on the data for tracks in a track-based alignment). A local-fit object is described by a linear or non-linear function $f(x_i, \boldsymbol{q}, \boldsymbol{p})$, depending on a (small) number of local parameters $\boldsymbol{q}$ and in addition on global parameters $\boldsymbol{p}$:

(12)
$$y_i = f(x_i, \boldsymbol{q}, \boldsymbol{p}) + \varepsilon_i \; .$$

The parameters $\boldsymbol{q}$ are called the local parameters, valid for the specific group of measurements (local-fit object). The quantity ε is the measurement error, expected to have (for ideal parameter values) mean zero (i.e. the measurement is unbiased) and standard deviation σ_i; often ε will follow at least The quantity x_i is assumed to be the coordinate of the measured value y_i, and is one of the arguments of the function $f(x_i, \boldsymbol{q}, \boldsymbol{p})$.

The global parameters needed to compute the expectation $f(x_i, \boldsymbol{q}, \boldsymbol{p})$ are usually already quite accurate and only small corrections habe to be determined. The convention is often to define the vector $\boldsymbol{p}$ to be a correction with initial values zero. In this case the final values of the global parameters will be small.

The fit of a local-fit object. The local parameters are usually determined in a least squares fit within the users code, assuming fixed global parameter values $\boldsymbol{p}$. If the function $f(x_i, \boldsymbol{q}, \boldsymbol{p})$ depends *non-linearly* on the local parameters $\boldsymbol{q}$, an iterative procedure is used, where the function is linearized, i.e. the *first* derivatives of the function $f(x_i, \boldsymbol{q}, \boldsymbol{p})$ with respect to the local parameters $\boldsymbol{q}$ are calculated. Then the function is expressed as a linear function of local parameter corrections $\boldsymbol{\Delta q}$ at some reference value $\boldsymbol{q}_k$:

$$f(x_i, \boldsymbol{q}_k + \boldsymbol{\Delta q}, \boldsymbol{p}) = f(x_i, \boldsymbol{q}_k, \boldsymbol{p}) + \frac{\partial f}{\partial q_1}\Delta q_1 + \frac{\partial f}{\partial q_2}\Delta q_2 + \dots \; ,$$

where the derivatives are calculated for $\boldsymbol{q} \equiv \boldsymbol{q}_k$. The corrections are determined by the linear least squares method, a new reference value is obtained by

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \boldsymbol{\Delta q}$$

and then, with k increased by 1, this is repeated until convergence is reached, i.e. until the corrections become essentially zero. For each iteration a linear system of equations (normal equations of least squares) has to be solved for the parameter corrections $\boldsymbol{\Delta q}$ with a right-hand side vector $\boldsymbol{b}$ with components

$$b_j = \sum_i \left(\frac{\partial f_i}{\partial q_j} \right) (y_i - f(x_i, \boldsymbol{q}_k, \boldsymbol{p}_k)) \, \frac{1}{\sigma_i^2} \; ,$$

where the sum is over all measurements y_i of the local-fit object. All components b_j become essential zero after convergence.

After convergence the equation (12) can be expressed with the fitted parameters $\boldsymbol{q}$ in the form

$$y_i = f(x_i, \boldsymbol{q}, \boldsymbol{p}) + \frac{\partial f}{\partial q_1}\Delta q_1 + \frac{\partial f}{\partial q_2}\Delta q_2 + \ldots + \varepsilon_i \; .$$

The difference $z_i = y_i - f(x_i, \boldsymbol{q}, \boldsymbol{p})$ is called the residual and the equation can be expressed in terms of the residual measurement z_i as

(13)

$$z_i \equiv y_i - f(x_i, \boldsymbol{q}, \boldsymbol{p}) = \left(\frac{\partial f}{\partial q_1}\right)\Delta q_1 + \left(\frac{\partial f}{\partial q_2}\right)\Delta q_2 + \ldots + \varepsilon_i \; .$$

After convergence of the local fit all corrections $\Delta q_1, \Delta q_2, \ldots$ are zero, and the residuals z_i are small and can be called the *measurement error*. As mentioned above the model function $f(x_i, \boldsymbol{q}, \boldsymbol{p})$ may be a linear or a non-linear function of the local parameters. The fit procedure in the users code may be a more advanced method than least squares. In track fits often Kalman filter algorithms are applied to treat effects like multiple scattering into account. But for any fit procedure the final result will be *small* residuals z_i, and it should also be possible to define the derivatives $\partial f/\partial q_j$ and $\partial f/\partial p_\ell$; these derivatives should express the *change* of the residual z_i, if the local parameter q_j or the global parameter p_ℓ is changed by Δq_j or Δp_ℓ. In a track fit with strong multiple scattering the derivatives will become rather small with increasing track length, because the information content is reduced due to the multiple scattering.

Local fits are later (in **Pede**) repeated in a simplified form; these fits need as information the difference z_i and the derivatives, and correspond to the *last iteration* of the local fit, because only small parameter changes ares involved. Modified values of global parameters $\boldsymbol{p}$ during the global fit result in a change of the value of $f(x_i, \boldsymbol{q}, \boldsymbol{p})$ and therefore of z_i, which can be calculated from the derivatives. The derivatives with respect to the local parameters allow to repeat the local fit and to calculate corrections for the local parameters $\boldsymbol{q}$.

2.3.2.2 Global parameters

Now the global parameters $\boldsymbol{p}$ are considered. The calculation of the residuals $z_i = y_i - f(x_i, \boldsymbol{q}, \boldsymbol{p})$ depends on the global parameters $\boldsymbol{p}$ (valid for *all* local-fit objects). This dependence can be considered in two ways. As expressed above, the *parametrization* $f(x_i, \boldsymbol{q}, \boldsymbol{p})$ depends on the global parameters $\boldsymbol{p}$ and on corrections $\boldsymbol{\Delta p}$ of the global parameters (this case is assumed below in the sign of the derivatives with respect to the global parameters). Technically equivalent is the case, where the *measured* value y_i is calculated from a raw measured value, using global parameter values; in this case the residual could be written in the form $z_i = y_i(\boldsymbol{p}) - f(x_i, \boldsymbol{q})$. It is assumed that reasonable values are already assigned to the global parameters $\boldsymbol{p}$ and the task is to find (small) corrections $\boldsymbol{\Delta p}$ to these initial values.

Equation (13) is extended to include corrections for global parameters. Usually only few of the global parameters influence a local-fit object. A global parameter carries a label ℓ, which is used to identify the global parameter uniquely, and is an arbitrary positive integer. Assuming that labels ℓ from a set Ω contribute to a single measurement the extended equation becomes

(14)

$$z_i = y_i - f(x_i, \boldsymbol{q}, \boldsymbol{p}) = \sum_{j=1}^{v}\left(\frac{\partial f}{\partial q_j}\right)\Delta q_j + \sum_{\ell\in\Omega}\left(\frac{\partial f}{\partial p_\ell}\right)\Delta p_\ell \; .$$

Millepede essentially performs a fit to all single measurements for all groups of measurements simultaneously for all sets of local parameters and for all global parameters. The result of this simultaneous fit are optimal corrections $\boldsymbol{\Delta p}$ for all global parameters; there are no limitations in the number of local parameters sets. Within this global fit the local fits (or better: the last iteration of the local fit) has to be repeated for each local-fit object.

Global parameter labels. The label ℓ, carried by a global parameter, is an arbitrary positive (31-bit) integer (most-significant bit is zero), and is used to identify the global parameter uniquely. Arbitrary gaps between the numerical values of labels values are allowed. It is recommended to design the label in a way which allows to reconstruct the meaning of the global parameter from the numerical value of the label ℓ.

2.3.2.3 Writing data files with Mille

The user has to provide the following data for each single measurement:

$$n_{lc} = \text{number of local parameters} \qquad \text{array} : \left(\frac{\partial f}{\partial q_j}\right)$$

$$n_{gl} = \text{number of global parameters} \qquad \text{array} : \left(\frac{\partial f}{\partial p_\ell}\right);\ \text{label-array} \quad \ell$$

$$z = \text{residual} \quad (\equiv y_i - f(x_i, \boldsymbol{q}, \boldsymbol{p})) \qquad \sigma = \text{standard deviation of the measurement}$$

These data are sufficient to compute the least squares normal equations for the local and global fits. Using calls of `MILLE` the measured data and the derivatives with respect to the local and global parameters are specified; they are collected in a buffer and written as one record when one local-fit object is finished in the user reconstruction code.

Calls for Fortran version: Data files should be written with the Fortran MILLE on the system used for **Pede**; otherwise the internal file format could be different.

```
CALL MILLE(NLC,DERLC,NGL,DERGL,LABEL,RMEAS,SIGMA)
```

where

- `NLC` = number of local parameters `DERLC`
- `DERLC` = array `DERLC(NLC)` of derivatives
- `NGL` = number of global derivatives in this call
- `DERGL` = array `DERLG(NGL)` of derivatives
- `LABEL` = array `LABEL(NGL)` of labels
- `RMEAS` = measured value
- `SIGMA` = error of measured value (standard deviation)

After transmitting the data for all measured points the record containing the data for one local-fit object is written. following call

```
CALL ENDLE
```

The buffer content is written to a file with the record.

Alternatively the collected data for the local-fit object have to be discarded, if some reason for *not using* this local-fit object is found.

```
CALL KILLE
```

The content of the buffer is reset, i.e. the data from preceeding `MILLE` calls are removed.

Additional floating-point and integer data (special data) can be added to a local fit object by the call

```
CALL MILLSP(NSPEC,FSPEC,ISPEC)
```

where

- `NSPEC` = number of special data `DERLC`
- `FSPEC` = array `FSPEC(NSPEC)` of floating point data
- `ISPEC` = array `ISPEC(NSPEC)` of integer data

The floating-point and integer arrays have the same length. These special data are not yet used, but may be used in future options.

Calls for C version: The C++-class Mille can be used to write C-binary files. The constructor

```
Mille(const char *outFileName, bool asBinary = true, bool writeZero = false);
```

takes at least one argument, defining the name of the output file. For debugging purposes it is possible to give two further arguments: If `asBinary` is false, a text file (not readable by pede) is written instead of a binary output. If `writeZero` is true, derivatives that are zero are not suppressed in the output as usual.

The member functions

```
void mille(int NLC, const float *derLc, int NGL, const float *derGl,
           const int *label, float rMeas, float sigma);
void special();
void end();
void kill();
```

have to be called equivalently like the Fortran subroutines `MILLE`, `MILLSP`, `ENDLE}` and `KILLE`. To properly close the output file, the `Mille` object should be deleted (or go out of scope) after the last processed record.

Root version: A `root` version is perhaps added in the future.

2.3.2.4 Non-linearities

Experience has shown that non-linearities in the function $f(x_i, \boldsymbol{q}, \boldsymbol{p})$ are usually small or at least not-dominating. As mentioned before, the convention is often to define the start values of the global parameters as zero and to expect only *small* final values. Derivatives with respect to the local and global parameters can be assumed to be constants within the range of corrections in the determination of the global parameter corrections. Then a single pass through the data, writing data files and calculating global parameter corrections with **Millepede** is sufficient. If however corrections are large, they may require to re-calculate the derivatives and another pass or several passes through the data with re-writing data files may be necessary. As a check of the whole procedure a second pass is recommended.

2.3.2.5 Application: Alignment of tracking detectors

In the alignment of tracking detectors the group of measurement, the local-fit object, may be the set of hits belonging to a single track. Two to five parameters $\boldsymbol{q}$ may be needed to describe the dependence of the measured values on the coordinate x_i.

In a real detector the trajectory of a charged particle does not follow a simple parametrization because of effects like multiple scattering due to the detector material. In practice often a Kalman filter method is used without an explicit parametrization; instead the calculation proceeds along the track taking into account e.g. multiple scattering effects. Derivatives with respect to the (local) track parameters are not directly available. Nevertheless derivatives as required in equation (14) are still well-defined and can be calculated. Assuming that the (local) track parameters refer to the starting point of the track, for each point along the track the derivative $(\partial f/\partial q_j)$ has to be equal to $\Delta z_i/\Delta q_j$, if the local parameter q_j if changed by Δq_j, and the corresponding change of the residual z is Δz_i. This derivative could be calculated numerically.

For low-momentum tracks the multiple-scattering effects are large. Thus the derivative above will tend to small values for measured points with a lot of material between the measured point and the starting point of the track. This shows that the contribution of low-momentum tracks with large multiple-scattering effects to the precision of an alignment is small.

2.3.3 Text files

In text files the steering information for the program execution and further information on parameters and constraints is given. The main text file is mandatory. Its default name is `steer.txt`; if the name is different it has to be given in the command line of **Pede**. In the main text file the file names of data files and optionally of further text files are

given. Text files should have an extension which contains the character pairs `xt` or `tx`. In this section all options for the global fits and their keywords are explained. The computation of the global fit is described in detail in the next section Computation of the global fit; it may be necessary to read this section Computation of the global fit to get an understanding of the various options.

2.3.3.1 General data format of text files

Text files contain file names, numerical data, keywords (text) and comment in free format, but there are certain rules.

File names for all text and steering file should be the leading information in the main steering file, with one file name per line, with correct characters, since the file names are used to open the files.

Numerical can be given with or without decimal point, optionally with exponent field. The numerical data

```
13234       13234.0      13.234E+3
```

are all identical.

Comments can be given in every line, preceeded by the ! sign; the text after the ! is ignored. Lines with $*$ or ! in the first columm are considered as comment lines. Blank lines are ignored.

Keywords are necessary for certain data; they can be given in upper or lower case characters. Keywords with few typo errors may also be recognized correctly. The program will stop if a keyword is not recognized.

Below is an example for a steering file:

```
Fortranfiles
!/home/albert/filealign/lhcrun1.11    ! data from first test run
 /home/albert/filealign/lhcrun2.11    ! data from second run
 /home/albert/filealign/cosmics.bin   ! cosmics
 /home/albert/detalign/mydetector.txt ! steering file from previous log file
 /home/albert/detalign/myconstr.txt   ! test constraints

constraint 0.14       ! numerical value of r
713 1.0               ! pair of parameter label and numerical factor
719 0.5  720 0.5      ! two pairs

CONSTRAINTS 1.2
 112 1.0
 113 -1.0
 114 0.5
 116 -0.5

Parameter
201 0.0   -1.0
202 1.0   -1.0
204 1.23 0.020

method inversion 5 0.1
end
```

2.3.3.2 File information

Names of data and text files are given in single text lines. The file-name extension is used to distinguish text files (extension containing `tx` or `xt`) and binary data files. Data files may be Fortran or C files; by default C files are assumed. Fortran files have to be preceded by the textline

```
Fortranfiles
```

and C files may be preceded by the textline

```
Cfiles
```

Mixing of C- and Fortran-files is allowed.

2.3.3.3 Parameter information

By default all global parameters appearing in data files with their labels are assumed to be variable parameters with initial value zero, and used in the fit. Initial values different from zero and the so-called presigma (see below) may be assigned to global parameters, identified by the parameter label. The optional parameter information has to be provided in the form below, starting with a line with the keyword `Parameter`:

```
Parameter
label   initial_value   presigma
 ...
label   initial_value   presigma
```

The three numbers *label*, *initial_value* and *presigma* in a textline may be followed by further numbers, which are ignored.

Note

Note that the result file has the same format and may be used as input file, if a program execution should be continued.

All parameters not given in this file, but present in the data files are assumed to be variable, with initial value of zero.

Pre-sigma. The pre-sigma s_ℓ defines the status of the global parameter:

$s_\ell > 0$: The parameter is *variable* with the given initial value. A term $1/s_\ell^2$ is added to the diagonal matrix element of the global parameter to stabilize a perhaps poorly defined parameter. This addition should *not* bias the fitted parameter value, if a sufficient number of iterations is performed; it may however bias the calculated error of the parameter (this is available only for the matrix inversion method).

$s_\ell = 0$: The pre-sigma is zero; the parameter is *variable* with the given initial value.

$s_\ell < 0$: The pre-sigma is negative. The parameter is defined as **fixed**; the initial value of the parameter is used in the fits.

The lines below show examples, with the numerical value of the label, the initial value, and the pre-sigma:

```
11 0.01232  0.0    ! parameter variable, non-zero initial value
12 0.0      0.0    ! parameter variable, initial value zero
20 0.00232  0.0300 ! parameter variable with initial value 0.00232
30 0.00111  -1.0   ! parameter fixed, but non-zero parameter value
```

Result file. At the end of the **Pede** program a result file `millepede.res` with the result is written. In the first three columns it carries the label, the fitted parameter value (initial value + correction) and the pre-sigma (default is zero). This file can, perhaps after editing, be used for a repeated **Pede** job. Column 4 contains the correction and in the case of solution by inversion or diagonalization column 5 its error and optionally column 6 the global correlation.

2.3.3.4 Constraint information

Equality constraints allow to fix certain undefined or weakly defined linear combinations of global parameters by the addition of equations of the form

$$c = \sum_{\ell \in \Omega} f_\ell \cdot p_\ell$$

The constraint value c is usually zero. The format is:

```
Constraint   value
label        factor
 ...
label        factor
```

where *value*, *label* and *factor* are numerical values. Note that no error of the value is given. The equality constraints are, within the numerical accuracy, exactly observed in a global fit. Each constraint adds another Lagrange multiplier to the problem, and needs the corresponding memory space in the matrix.

Instead of the keyword `Constraint` the keyword `Wconstraint` can be given (`this option is not yet implemented!`). In this case the factor for each global parameter given in the text file is, in addition, multiplied by the weight W_ℓ of the corresponding global parameter in the complete fit:

$$c = \sum_{\ell \in \Omega} f_\ell \cdot W_\ell \cdot p_\ell$$

Thus global parameters with a large weight in the fit get also a large weight in the constraint.

Mathematically the effect of a constraint does not change, if the constraint equation is multiplied by an arbitrary number. Because of round-off errors however it is recommended to have the constraint equation on a equal accuracy level, perhaps by a scale factor for constraint equations.

2.3.3.5 Global parameter measurements

Measurements with measurement *value* and measurement *error* for linear combinations of global parameters in the form

$$y = \sum_{\ell \in \Omega} f_\ell \cdot p_\ell + \varepsilon$$

can be given, where ε is the measurement error, assumed to follow a distribution $N(0, \sigma^2)$, i.e. zero bias and standard deviation σ. The format is:

```
Measurement    value    sigma
label    factor
 ...
label    factor
```

where *value*, *sigma*, *label* and *factor* are numerical values. Each measurement *value* $\pm$ *sigma* (standard deviation) given in the text files contributes to the overall objective function in the global fit. At present no outlier tests are made.

The global parameter *measurement* and the *constraint* information from the previous section differ, in the definition, only in the additional information on the standard deviation given for the measurement. They differ however in the mathematical method: constraints add another parameter (Lagrange multiplier); measurements contribute to the parameter part of the matrix, and may require no extra space; however a sparse matrix may become less *sparse*, if matrix elements are added, which are empty otherwise.

2.3.3.6 Solution method selection

Methods. One out of several solution methods can be selected. The methods have different execution time and memory space requirement, and may also have a slighly different accuracy. The statement to select a method are:

```
method inversion         number1 number2
method diagonalization   number1 number2
method fullGMRES         number1 number2
method sparseGMRES       number1 number2
method cholesky          number1 number2
method bandcholesky      number1 number2
method HIP               number1 number2
```

The two numbers are:

- *number1* = number of iterations
- *number2* = limit for ΔF (convergence recognition).

For preconditioning in the GMRES methods a band matrix is used. The width of the variable-band matrix is defined by

```
bandwidth     number
```

method	memory space requirement
inversion, fullGMRES, cholesky	$(n^2+n)/2+nn_C+(n_C^2+n_C)/2$
diagonalization	$n+n(n-1)/2+n^2$
sparseGMRES	$n+qn(n-1)/2n_{cf}$
bandcholesky, preconditioning	$nm+nn_C+(n_C^2+n_C)/2$
HIP (no constraints)	nm

Table 1: Matrix space requirements for the different methods, with: $q =$ fraction of non-zero off-diagonal elements, $m =$ bandwidth of variable band matrix, $n_C =$ number of constraints, $n_{cf} =$ number of constraint-factors.

where *number* is the numerical value of the semi-bandwidth of a band matrix. The method called `bandcholesky` uses only the band matrix (reduced memory space requirement), and needs more iterations. The different methods are described below.

Iterations and convergence. The mathematical method in principle allows to solve the problem in a single step. However due to potential inaccuracies in the solution of the large linear system and due to a required outlier treatment certain internal iterations may be necessary. Thus the methods work in iterations, and each iteration requires one or several loops with evaluation of the objective function and the first-derivative vector of the objective function (gradient), which requires reading all data and local fits of the data. The first loop is called iteration 0, and (only) in this loop in addition the second-derivative matrix is evaluated, which takes more cpu time. This first loop usually brings a large reduction of the value of the objective function, and all following loops will bring small additional improvements. In all following loops the same second-derivative matrix is used (in order to reduce cpu time), which should be a good approximation.

A single loop may be sufficient, if there are no outlier problems. Using the command line option $-\mathbf{s}$ (*subito*) the program executes only a single loop, irrespective of the options required in the text files. The treatment of outliers turns a linear problem into a non-linear problem and this requires iterations. Also precision problems, with rounding errors in the case of a large number of global parameters, may require iterations. Each iteration $1, 2, \ldots$ is a line search along the calculated search direction, using the *strong Wolfe conditions*, which force a *sufficient* decrease of the function value and the gradient. Often an iteration requires only one loop. If, in the outlier treatment, a cut is changed, or if the precision of the constraints is insufficient, one additional loop may be necessary.

At present the iterations end, when the number specified with the method is reached. For each loop, the expected objective-function decrease and the actual decrease are compared. If the two values are below the limit for ΔF specified with the method, the program will end earlier.

Memory space requirements. The most important consumer of space is the symmetric matrix of the normal equations. This matrix has a parameter part, which requires for example for full storage $n(n+1)/2$ words, and for sparse storage $n+qn(n-1)/2$ words, for $n =$ number of global parameters and $q =$ fraction of non-zero off-diagonal elements. In addition the matrix has a constraint part, corresponding to the Lagrange multipliers. Formulae to calculate the number of (double precision) elements of the matrix are given in table 1.

The GMRES method can be used with a full or with a sparse matrix. Without preconditioning the GMRES method may be slow or may even fail to get an acceptable solution. Preconditioning is recommended and usually speeds up the GMRES method considerably. The band matrix required for preconditioning also takes memory space. The parameter part requires (only) $n \cdot m$ words, where $m =$ semibandwidth specified with the method. A small value like $m = 6$ is usually sufficient. The constraint part of the band matrix however requires up to $n \cdot n_C + n_C(n_C+1)$ words, and this can be a large number for large number n_C of constraint equations.

Another matrix with $n_C(n_C+1)/2$ words is required for the constraints, and this matrix is used to improve the precision of the constraints. Several vectors of length n and n_C are used in addition, but this space is only proportional to n and n_c.

The diagonalization method requires more space: in addition to the symmetric matrix another matrix with $(n+n_C)^2$ words is used for the eigenvectors; since diagonalization is also slower, the method can only be used for smaller problems.

Table 1

Comparison of the methods. The methods have a different computing-time dependence on the number of parameters n. The methods `diagonalization`, `inversion` and `cholesky` have a computing time proportional

to the third power of n, and can therefore be used only up to $n = 1000$ or perhaps up to $n = 5000$. Especially the diagonalization is slow, but provides a detailed information on weakly- and well-defined linear combinations of global parameters.

The GMRES methods has a weaker dependence on n, and allows much larger n-values; larger values of n of course require a lot of space and the sparse matrix mode should be used. The bandcholesky method can be used for preconditioning the GMRES method (recommended), but also as a stand-alone method; its cpu time depends only linearly on n, but the matrix is only an approximation and usually many iterations are required. For all methods the constraints increase the time and space consumption; this remains modest if the number of constraints n_C is modest, for example $n_C \leq 100$, but may be large for $n_C =$ several thousand.

2.3.4 Solution methods

Inversion. The computing time for inversion is roughly $2 \times 10^{-8} \cdot n^3$ seconds (on a standard PC). For $n \approx 1000$ the inversion time of ≈ 20 seconds is still acceptable, but for $n \approx 10000$ the inversion time would be already more than 5 hours. The advantage of the inversion is the availability of *all* parameter errors and global correlations.

Cholesky decomposition. The Cholesky decomposition (under test) is an alternative to inversion; eventually this method is faster and/or numerically more stable than inversion. At present parameter errors and global correlations are not available.

Diagonalization. The computing time for diagonalization is roughly a factor 10 larger than for inversion. Space for the square (non-symmetric) transformation matrix of eigenvectors is necessary. Parameter errors and global correlations are calculated for all parameters. The main advantage of the diagonalization method is the availability of the eigenvalues. Small positive eigenvalues of the matrix correspond to linear combinations of parameters which are only weakly defined and it will become possible to interactively remove the weakly defined linear combinations. This removal could be an alternative to certain constraints. Constraints are also possible and they should correspond to *negative* eigenvalues.

Generalized minimization of residuals (GMRES). The GMRES method is a fast iterative solution method for full and sparse matrices. Especially for large dimensions with $n \gg 1000$ it should be much faster than inversion, but of similar accuracy. Although the solution is fast, the building of the sparse matrix may require a larger cpu time. At present there is a limit for the number of internal GMRES-iterations of 2000. For a matrix with a bad condition number this number of 2000 internal GMRES-iterations will often be reached and this is often also an indication of a bad and probably inaccurate solution. An improvement is possible by preconditioning, selected if GMRES is selected together with a bandwidth parameter for a band matrix; an approximate solution is determined by solving the matrix equation with a band matrix within the GMRES method, which improves the eigenvalue spectrum of the matrix and often speeds up the method considerably. Experience shows that a small bandwidth of e.g. 6 is sufficient. In interactive mode parameter errors for selected parameters can be determined.

Variable band matrix. Systems of equations with a band matrix can be solved by the Cholesky decomposition. Often the matrix element around the diagonal are the essential matrix elements and in these cases the matrix can be approximated by a band matrix of small width, with a small memory requirement. The computing time is also small; it is linear with the number of parameters. However because the band matrix is only the approximate matrix often many iterations are necessary to get a good solution. The definition of the band width is necessary. The band width is variable and thus the constraints equations can be treated completely. This means that the constraints are observed even in this approximate solution.

2.3.4.1 Outlier treatment and debugging

Cases with very large residuals within the data can distort the result of the least squares fit. Reason for outliers may be selection mistakes or statistical fluctuations. A good outlier treatment may be neccessary to get accurate results. The problem of outlier treatment is complicated by the fact that initially the global parameters may be far from optimal and therefore large deviation may occur even for correct data before the global parameter determination. There are options for the complete removal of bad cases and for the down-weighting of bad data. Cases with a **huge χ^2 are automatically removed** in every iteration. The options to remove large χ^2 cases and down-weighting are *not* done in the first iteration. The options are:

```
chisqcut              number1 number2
outlierdownweighting  number
dwfractioncut         number
```

```
printrecord          number1 number2
```

Chisquare cut. With the keyword `chisqcut` two numbers can be given. Basis of χ^2 rejection is the χ^2-value corresponding to 3 standard deviations (and to a probability of 0.27%). or one degree of freedom this χ^2-value is 9.0, and for 10 degrees of freedom the χ^2-value is 26.9. The first number given with the keyword `chisqcut` is a factor for the χ^2-cut value, to be used in the first iteration, and the second number is the factor for the second iteration. In subsequent iterations the factor is reduced by the square root, with values below 1.5 replaced by 1. For example the statement

```
chisqcut 5.0 2.5
```

means: the cut factor is $5 \times \chi^2$-value corresponding to three standard deviations, $2.5 \times \chi^2$-value for the second iteration, $1.58 \times \chi^2$-value for the third iteration and $1 \times \chi^2$-value for subsequent iterations. Thus in the first iteration the cut is $5 \times 26.9 = 134.5$ for 10 degrees of freedom, and 26.9 after the third iteration.

Outlier downweighting. Outlier down-weighting for single data values requires repeated local fits with > 1 iterations in the local fit. The number given with the keyword `outlierdownweighting` is the number of iterations. In down-weighting the weight of the data point, which by default is $1/\sigma^2$, is reduced by an extra factor $\omega_i < 1$ according to the residual (see below). For n data points the total sum $S_f = \sum_i \omega_i$ of the extra factors is $\leq n$. The ratio $(n - S_f)/n$ is called the down-weight fraction, and a histogram of this ratio is accumulated in the second function evaluation. The first iteration of the local fit is done *without* down-weighting, iterations 2 and 3 use the M-estimation method with the Huber function. Subsequent iterations, if requested, use the M-estimation method with the Cauchy function for down-weighting, where the influence of very large deviations is further reduced. For example the statement

```
outlierdownweighting 4
```

means: iteration 1 of the local fit without down-weighting, iterations 2 and 3 with Huber function down-weighting and iteration 4 with Cauchy function down-weighting.

Downweighting fraction cut. Cases with a very large fraction of down-weighted measurements are very likely wrong data and should be rejected. The cut value should be determined from the histogram showing the down-weight fraction. Typical values are 0.1 for weights from the Huber function (`outlierdownweighting` ≤ 3) and 0.2 for weights from the Cauchy function (`outlierdownweighting` > 3). For example the statement

```
dwfractioncut 0.2
```

means to reject all cases with a down-weight fraction ≥ 0.2.

Record printout. For debugging many details of single records and the local fits are printed. Two record number *number1 number2* can be given with the keyword `printrecord`; printout is done in iteration 1 and 3. For numbers given as negative, the records with extreme deviations are selected in iteration 2, and printed in iteration 3. If the first number is given as -1, the record with the largest single residual is selected. If the second number is given as -1, the record with the largest value of χ^2/N_{df} is selected. For example the statement

```
 printrecord 4321 -1
```

means: the record 4321 is printed in iterations 1 and 3, and the record with the largest value of χ^2/N_{df} is selected in iteration 2 and printed in iteration 3.

2.3.4.2 Further options

There are miscellaneous further options which can be selected within the text files:

```
subito
entries           number
nofeasiblestart
wolfe             C1 C2
histprint
end
```

The option `subito` is also selected by **-s** in the command line. The program will end regularly, with normal output of the result, already after the first function evaluation (iteration 0).

The number, given with the keyword `entries`, is the minimum number of data for a global parameter. Global parameters which have a number of entries $=$ number of measured points connected to it in the local fit-objects,

smaller then the given number are ignored. This option is used to suppress global parameters with only few data, which are therefore rather inaccurate, and would spoil the condition of the linear system.

By default the initial global parameter values are corrected to follow all the constraints. A check for the validity of the constraints is repeated in every iteration, and eventually another correction is made at the end of an iteration. With the keyword `nofeasiblestart` the parameters are *not* made feasible (respecting the constraints) at the start.

The constants *C1* and *C2* are the line search constants for the strong Wolfe conditions; default values are the standard values $C_1 = 10^{-4}$ and $C_2 = 0.9$.

Histograms accumulated during the program are written to the textfile `millepede.his` and can be read after the job execution. If selected by the keyword `histprint` the histograms accumulated during the program are also printed.

The reading of a textfile ends, when the keyword `end` is encountered. Textlines after the line with `end` are not read.

2.3.5 Computation of the global fit

2.3.5.1 Memory managament

The solution of the optimization problem for a large number of parameters requires one large memory with many arrays, required to store vectors and matrices of large size corresponding to the number of parameters and constraints. **Millepede II** uses a large array in a common. This large array is dynamically divided into so-called subarrays, which are created, enlarged and moved, and removed according to the requirements. The space limitation is thus given for almost all subproblems only by the total space requirement. The largest space is required for the symmetric matrix of the normal least squares equations. This matrix can be a full matrix or for certain problems a sparse matrix. As an approximation a band matrix with a small band width can be used, if there is not enough space for the full or sparse matrix.

2.3.5.2 Initialization

After initialization of the memory management the command line options are read and analysed. Then the main text file is analysed, the names of other text files and the data files are recognized and stored in a table. Default value are assigned to the various variables; then all text files are read and the requested options are interpreted. The data for the definition for parameters, constraints and measurement are read and stored in subarrays.

2.3.5.3 First data loop

In subroutine `LOOP1` tables for the variable and fixed global parameters and translation tables are defined. The table of global parameter labels is first filled with the global parameters, appearing in the text files. All data files are read and all global parameter labels from the records are included in the list.

There are three integers to characterize a global parameter.

Global parameter label = `ITGBL`: this is a positive integer, from the full range of 32-bit integers. $1 \ldots 2147483647 = 2^{31} - 1$.

Global parameter index = `ITGBL`: A translation table is constructed to translate the global parameter label `ITGBL` to a global parameter index `ITGBI`, with a range $1 \ldots$ `NTGB`, where `NTGB` is the total number of global parameters (variable and fixed parameters).

Variable-parameter index = `IVGBI`: the global parameters which are *variable* carry a variable-parameter index, with a range $1 \ldots$ `NVGB`, where `NVGB` is the number of variable global parameters. Parameter vectors in the mathematical computation are vectors containing only the variable parameters.

Function calls are used to translate the global parameter label `ITGBL`:

$$\text{global parameter index} \leftarrow \text{global parameter label} \qquad \texttt{ITGBI} = \texttt{INONE(ITGBL)}$$
$$\text{variable parameter index} \leftarrow \text{global parameter label} \qquad \texttt{IVGBI} = \texttt{INSEC(ITGBL)}\ .$$

The translation is done with a hash-index table. The translation from one parameter index to another one and back to the parameter label is done by the following statement functions:

$$\begin{aligned} \text{global parameter label} \leftarrow \text{global parameter index} \qquad & \texttt{ITGBL} = \texttt{JTGBL(ITGBI)} \\ \text{variable-parameter index} \leftarrow \text{global parameter index} \qquad & \texttt{IVGBI} = \texttt{JVGBI(ITGBI)} \\ \text{global parameter index} \leftarrow \text{variable-parameter index} \qquad & \texttt{ITGBI} = \texttt{JTGBI(IVGBI)}\ , \end{aligned}$$

which use simple fixed-length tables.

2.3.5.4 Second data loop

In subroutine `LOOP2` the subarray dimensions of the various vectors and matrices are determined. All data files are read and for each local-fit object the number of local and of global parameters is determined; the maximum values of the numbers are used to define the subarray dimensions for the arrays to be used in local fits, and for the contributions to the global fit.

The sparse matrix storage requires to establish the pointer structure of the sparse matrix. During reading the data files the list of global parameters used in each local-fit object is determined; an entry is made in a table for each pair of global parameters, which later corresponds to an off-diagonal matrix element. A hash-index table is used for the search operation. For sparse and full matrix storage the requirements of the constraint equations and Lagrange multipliers has to be included. For the sparse matrix a pointer structure is build, which allows a fast *matrix* $\times *$vector$*$ product. At the end of the subroutine all subarrays required for the determination of the solution are prepared.

2.3.5.5 Solution method overview

The section gives an short overview over the solution method; more detailed explanation is given in the subsequent sections.

In principle the solution can be determined in a single step; this mode can be selected by the keyword `subito` or option **-s**. There are however two reasons to perform iterations, which improve the result:

- Due to the large size or the method the one-step solution may be affected by rounding errors and may not be precise. Experience shows that the overall value objective-function value can be reduce using more than one step, although the decrease is sometimes rather small;

- Outliers may be important; they add a $*$non-linear component to the otherwise linear problem and this requires iterations and repeated evaluation of the objective function; each function evaluation requires to read again all data files and to repeat the local fits. In a comparison of Millepede I and II results initially certain differences were observed; only after a careful outlier treatment these differences became small or disappeared.

The solution determined in iterations is explained. The starting iteration, with iteration number 0, is the most important and time-consuming one. The data files are read, and for each case a local fit is made. The matrix of the normal equations is formed only in this starting iteration. Depending on the selected method the matrix is inverted, diagonalized or decomposed, and the resulting matrix and decomposition is used in all later data-file loops, which take less time compared to the first data-file loop. Thus the data-file loop in iteration number 0 may be rather time consuming, especially for the case of a sparse matrix, where the index determination takes some time. The right-hand-side vector is formed in each data-file loop. A correction step in global parameter space is calculated at the end of the data-file loop in iteration number 0. Often the precision is already sufficient and no further data-file loops are necessary. In the subito mode (keyword `subito` or option **-s**) the correction is added to the global parameter values and the program stops.

Iterations with more data-file loops are recommended, to check the result, eventually to improve the result and to treat outliers. In each iterations a so-called line search is done, where the overall value of the objective functions is optimized along the correction-step direction *sufficiently*, using the strong Wolfe criterion. Often a single step is already sufficient for an iteration. The sample of local-fit objects may change during the iterations becuse of changing cut values; this may require extra function and derivative calculations.

2.3.5.6 Data file loops during iterations

In subroutine `LOOPN` all data files are read. For each local-fit object the local fit is performed, based on the actual global parameter corrections, and eventually including downweighting of outliers. Using the results of the local fit the value of the objective function F, the vector of first derivatives (gradient) and, in the first data file loop, the matrix of second derivatives is calculated. For a linear fit the matrix of second derivatives will not change during the iterations; the outlier treatment will change the matrix, but usually the changes are small und the matrix collected in the first data loop is at least a good and sufficiently accurate approximation. Thus data loops after the first one are faster; depending on the method the solution of the matrix equation will be faster after the first data loop.

2.3.5.6.1 The local fit

The number of local parameters of a local-fit object is usually small; typical values are between 2 and 5. Since the problem is linear, a single step is sufficient in a local fit with minimization of the sum

$$\frac{1}{2}\sum_i \left(\frac{y_i - f(x_i, \boldsymbol{q}, \boldsymbol{p})}{\sigma_i}\right)^2 ,$$

unless outlier downweighting is required, which may require a few iterations. In a loop over the local measurements the matrix and the right-hand side of the least squares normal equations are summed. For each single measured value, the residual measurement z_i

$$z_i \equiv y_i - f(x_i, \boldsymbol{q}, \boldsymbol{p})$$

(see equation (13)) has to be corrected for the actual global parameters corrections $\boldsymbol{\Delta p}$ using the first global parameter derivatives (see equation (14)). The corrected residual z_i' is then used in the accumulation of the matrix and vector:

$$\Gamma_{jk} = \sum_i \left(\frac{\partial f_i}{\partial q_j}\right)\left(\frac{\partial f_i}{\partial q_k}\right)\frac{1}{\sigma_i^2} \qquad \beta_j = \sum_i \left(\frac{\partial f_i}{\partial q_j}\right)\frac{z_i'}{\sigma_i^2} .$$

Corrections $\boldsymbol{\Delta q}$ are determined by the solution of the matrix equation

$$\boldsymbol{\Gamma \Delta q} = -\boldsymbol{\beta} ,$$

which is determined using matrix inversion $\boldsymbol{\Delta q} = -\boldsymbol{\Gamma}^{-1}\boldsymbol{b}$, because the inverse matrix $\boldsymbol{\Gamma}^{-1}$ (the covariance matrix) is necessary for the contribution to the matrix of the global normal equations. The residual z_i' are then corrected for the local parameter corrections:

$$z_i'' = z_i' - \sum_j \frac{\partial f_i}{\partial q_j}\Delta q_j$$

and the new residuals z_i'' are used to calculate the χ^2 value S of the local fit

$$S = \sum_i \left(\frac{z_i''}{\sigma_i}\right)^2 ,$$

which should follow a χ^2. The number of local parameters of a local-fit object is usually small; typical values are between 2 and 5. Since the problem is linear, a single step is sufficient in a local fit with minimization of the sum

$$\frac{1}{2}\sum_i \left(\frac{y_i - f(x_i, \boldsymbol{q}, \boldsymbol{p})}{\sigma_i}\right)^2 ,$$

unless outlier downweighting is required, which may require a few iterations. In a loop over the local measurements the matrix and the right-hand side of the least squares normal equations are summed. For each single measured distribution with the given number of degrees of freedom n_{df} = number of measurements minus number of parameters. A fraction of 0.27 % or one out of 370 of the *correct* cases should have a deviation which corresponds to 3 standard devations in the $n_{\mathrm{df}} = 1$ case.

Very badly fitting cases should be rejected before using them for the global parameter fit. Basis of the rejection is the comparison of the sum S and the *0.27 % value* χ^2_{cut} of the $\chi^2_{n_{\mathrm{df}}}$ distribution. If the value S exceeds χ^2_{cut} by more than a factor of 50, then the sum is called *huge* and the local-fit object is rejected. Cases with $n_{\mathrm{df}} = 0$ are rejected too, because they do no allow any check. These rejections are always made, even if the user has not requested any outlier rejection.

With keyword `chisqcut` the user can require a further rejection. The first number given with the keyword is used during iteration 0; the local-fit object is rejected, if the value S is larger than this number times χ^2_{cut}. Often this number has to rather high, e.g. 12.0; the initial value of S may be rather high before the first determination of global parameters, even for correct data, and, if possible, no correct data should be rejected by the cut. The second number given with the keyword is used during iteration 1 and can be smaller than the first number, because the first improvement of the local parameters is large. In further iterations the number is reduced by the sqrt-function. Values below 1.5 are replaced by 1, and thus the rejection is finally be done with a 0.27 %-rejection probability for correct data.

All not rejected local-fit objects contribute to the global fit. The function value F is the sum of the S-values for the local-fit objects:

$$F = \sum_l S_l \qquad \text{sum with index } l \text{ over all local fit-objects .}$$

Here even S-values from local-fit objects, which are rejected by cuts are included, with $S_l =$ cut value, in order to keep the sum F meaningful (otherwise the rejection of many local-fit object could simulate an improvement – which is not the case).

2.3.5.6.2 Contributions to the global parameter fit

After an accepted local fit the contributions to the normal least squares equations of the global fit have to be calculated. The first contribution is from the first order derivatives with respect to the global parameters:

(15)

$$(\boldsymbol{\Delta C}_1)_{jk} = \sum_i \left(\frac{\partial f_i}{\partial p_j}\right)\left(\frac{\partial f_i}{\partial p_k}\right)\frac{1}{\sigma_i^2} \qquad \Delta g_j = \sum_i \left(\frac{\partial f_i}{\partial p_j}\right)\frac{z_i''}{\sigma_i^2} .$$

Note that in the g_j-term the residual is already corrected for the local fit result. The vector $\boldsymbol{g}$ is the gradient of the global objective function.

The second contribution is, according to the **Millepede** principle, a mixed contribution from first order global and local derivatives. After the local fit the matrix $\boldsymbol{G}$ is formed, which has a number of columns equal to the number of local parameters, and a number of rows equal to the number of global parameters. The elements are

$$(\boldsymbol{G})_{jk} = \sum_i \left(\frac{\partial f_i}{\partial p_j}\right)\left(\frac{\partial f_i}{\partial q_k}\right)\frac{1}{\sigma_i^2} .$$

The second contribution to the global matrix $\boldsymbol{C}$ is then

(16)

$$\boldsymbol{\Delta C_2} = -\boldsymbol{G\Gamma}^{-1}\boldsymbol{G}^{\mathrm{T}}$$

with the matrices $\boldsymbol{G}$ and $\boldsymbol{\Gamma}$ from a local fit. Because of the dimension of matrices $\boldsymbol{G}$ and $\boldsymbol{\Delta C}$ the calculation would require a large number of operations. However only a small number of rows of matrix $\boldsymbol{G}$ is not equal to zero and the calculation can be restricted, with the help of pointers, to the non-zero part and in addition one can use the fact that the result is a symmetric matrix. With this code, which may appear somewhat complicated in comparison to the standard matrix multiplication, the computation time is small. Note that no extra contribution to the gradient vector $\boldsymbol{g}$ is necessary because the residual z_i'' used above already includes the local fit result. The sum $\boldsymbol{C}$ of the two contributions, $\boldsymbol{\Delta C}_1$ from equation (15), and $\boldsymbol{\Delta C}_2$ from equation (16), summed over all local fit-objects, is the final matrix of the least squares normal equations. Element $(\boldsymbol{C})_{jk}$ is related to the global parameters with indices j and k. The matrix $\boldsymbol{C}$ corresponds to a simultaneous fit of the global *and* local parameters of all local-fit objects. The first term (15) alone corresponds to the fit, when the local parameters are assumed to be fixed; only those elements $(\boldsymbol{C})_{jk}$ of the matrix $\boldsymbol{C}$ are non-zero, where the two global parameters with indices j and k appear in the same measurement y_i. In contrast, in the second term (16) those elements $(\boldsymbol{C})_{jk}$ of the matrix $\boldsymbol{C}$ are non-zero, where the two global parameters with indices j and k appear in the same local fit.

2.3.5.6.3 Outlier downweighting in local fits

The influence of a single measurement y_i resp. z_i is proportional to the absolute value of the normalised residual $\zeta_i = z_i/\sigma_i$ in the pure least squares fit of local-fit objects. Thus measurements with a large deviation will in general distort the resulting fit. This effect can be avoided by downweighting those measurements. Measurements with large residuals are included in the method of M-estimates by assigning to them, instead of the ideal Gaussian distribution of least squares, a different distribution with larger tails. Technically this is done by multiplying the standard weight

$1/\sigma_i^2$ of least squares by another factor, which depends on the actual value of $\zeta_i = z_i/\sigma_i$. This method requires an initial fit without downweighting, followed by iterative downweighting.

In **Millepede II** the local fits are improved, if necessary, by downweighting *after* the first iteration, with a number of iterations specified by the user with keyword `outlierdownweighting` {it numberofiterations}. In iterations 2 and 3 the downweighting is done with the Huber function and the Cauchy function is used, if more iterations are requested (see section Outlier treatment and debugging for an explanation of the two functions).

2.3.5.7 Global fit

2.3.5.7.1 Global fit without constraints

The aim is to find a step vector $\boldsymbol{\Delta p}$, which minimizes the objective function: $F(\boldsymbol{p}+\boldsymbol{\Delta p}) =$ minimum. In the k-th iteration the approximate solution is $\boldsymbol{p}_k$. A quadrative model function $\widetilde{F}(\boldsymbol{p}+\boldsymbol{d})$

(17)
$$\widetilde{F}(\boldsymbol{p}_k+\boldsymbol{d}) = F_k + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{d} + \tfrac{1}{2}\boldsymbol{d}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{d}$$

is defined with a vector $\boldsymbol{d}$. The step vector $\boldsymbol{d}$ is determined as solution of the linear system

$$\boldsymbol{C}\boldsymbol{d} = -\boldsymbol{g}\ .$$

The step $\boldsymbol{d}$ will minimize the quadratic function $\widetilde{F}(\boldsymbol{p}+\boldsymbol{d})$ of equation (17), unless there is some inaccuracy due to rounding errors and other approximations of the matrix or the solution method. In order to get a *sufficient* decrease of the objective function $F(\boldsymbol{p}+\boldsymbol{\Delta p})$ a line search algorithm is used.

2.3.5.7.2 Line search

In the line search algorithm a search is made for the minimum of the function $F(\boldsymbol{p}+\boldsymbol{\Delta p})$ along a line $\boldsymbol{p}+\alpha\cdot\boldsymbol{d}$ in the parameter space, where α is a factor to be determined. A function $\Phi(\alpha)$ of this factor,

$$\Phi(\alpha) \equiv F(\boldsymbol{p}+\alpha\cdot\boldsymbol{d})$$

is considered. Each function evaluation at a certain value of α requires one loop through the data with all local fits. The function value for $\alpha = 0$ is already calculated before (start value); the first value of α to be considered is $\alpha = 1$ and this value is often already a sufficiently accurate minimum. In the line search algorithm the so-called strong Wolfe conditions are used; these are

$$F(\boldsymbol{p}+\alpha\cdot\boldsymbol{d}) \le F(\boldsymbol{p}) + C_1\alpha\nabla F^{\mathrm{T}}\boldsymbol{d}$$
$$\left|\nabla F(\boldsymbol{p}+\alpha\cdot\boldsymbol{d})^{\mathrm{T}}\boldsymbol{d}\right| \le C_2\left|\nabla F^{\mathrm{T}}\boldsymbol{d}\right|$$

with $0 < C_1 < C_2 < 1$. The first condition requires a sufficient decrease of the function. The second condition, called curvature condition, ensures a slope reduction, and rules out unacceptable short steps. In practice the constant C_1 is chosen to small, for example $C_1 = 10^{-4}$. Typical values of the constant C_2 are 0.9 for a search vector by a Newton method and 0.1 for a search vector by the conjugate gradient method. Default values in **Millepede II** are $C_1 = 10^{-4}, C_2 = 0.9$. Often only one value of α, sometimes two to three values are evaluated. The last function value should never be higher than the start value.

2.3.5.7.3 Iterations

The global fit can be performed with a result $\boldsymbol{\Delta p}$ in one step, if the matrix equation is accurately solved and if there are no outliers. The result $\boldsymbol{\Delta p}$ calculated in the first data loop is usually already close to the final solution; it gives the largest reduction of the objective function. Outliers introduce some non-linearity into the problem and require iterations with repeated data loops. Another argument for repeating the steps is the non-neglectigible inaccuracy of the numerical solution of the step calculation due to rounding errors for the large number of parameters.

Iteration 0. The first data loop is called iteration 0. The function value, the first derivative vector $\boldsymbol{g}$ and the second derivative matrix $\boldsymbol{C}$ or an approximation to it are calculated. They allow to calculate the first step $\boldsymbol{\Delta p} = \boldsymbol{d}$.

Iteration ≥ 1. In all further iterations a line search is performed. Starting value is the point in parameter space, obtained in the previous iteration. Each function evaluation requires a data loop with local fits. The expected decrease ΔF in an iteration can be estimated by

$$\Delta F_{\mathrm{estimate}} = -\boldsymbol{g}^{\mathrm{T}}\boldsymbol{d}\ .$$

This can be compared with the actual decrease

$$\Delta F_{\text{actual}} = F(\boldsymbol{p}) - F(\boldsymbol{p} + \alpha \boldsymbol{d})$$

at the end of the iteration. Both values should become smaller during the iterations; convergence can be assumed if both are of the order of ≈ 1.

2.3.5.7.4 Global fit with constraints

In the Lagrange multiplier method one additional parameter λ is introduced for each single constraint, resulting in an m-vector $\boldsymbol{\lambda}$ of Lagrange multiplier. A term depending on $\boldsymbol{\lambda}$ and the constraints is added to the objective function, resulting in the Lagrange function

$$\mathscr{L}(\boldsymbol{p}, \boldsymbol{\lambda}) = F_k + \boldsymbol{g}^{\mathrm{T}} \boldsymbol{d} + \tfrac{1}{2} \boldsymbol{d}^{\mathrm{T}} \boldsymbol{C} \boldsymbol{d} + \boldsymbol{\lambda} (\boldsymbol{A} (\boldsymbol{p}_k + \boldsymbol{d}) - \boldsymbol{c})$$

Using as before a quadratic model for the function $F(\boldsymbol{p})$ and taking derivatives w.r.t. the parameters $\boldsymbol{p}$ and the Lagrange multipliers $\boldsymbol{\lambda}$, the two equations

$$\begin{aligned} \boldsymbol{C}\, \boldsymbol{d} + \boldsymbol{A}^{\mathrm{T}} \boldsymbol{\lambda} &= -\boldsymbol{g} \\ \boldsymbol{A}\, \boldsymbol{d} &= \boldsymbol{c} - \boldsymbol{A} \boldsymbol{p}_k \end{aligned}$$

are obtained; the second of these equations is the constraint equation. This system of two equations can be combined into one matrix equation

$$\left(\begin{array}{c|c} \boldsymbol{C} & \boldsymbol{A}^{\mathrm{T}} \\ \hline \boldsymbol{A} & \boldsymbol{0} \end{array} \right) \left(\begin{array}{c} \boldsymbol{d} \\ \hline \boldsymbol{\lambda} \end{array} \right) = \left(\begin{array}{c} -\boldsymbol{g} \\ \hline \boldsymbol{c} - \boldsymbol{A} \boldsymbol{p}_k \end{array} \right) .$$

The matrix on the left hand side is still symmetric. *Linear* least squares problems with *linear* constraints can be solved directly, without iterations and without the need for initial values of the parameters. Insufficient precision of the equality constraints is corrected by the method discussed in section Feasible parameters.

2.3.5.8 Output text files

Several output text file are generated in **Pede**. All files have a fixed name. If files with this name are existing at the **Pede** start, they are renamed by appending a $\sim$ to the file name; existing files with file name already carrying the $\sim$ are removed!

Log-file for Pede. The file `millepede.log` summarizes the job execution. Especially information on the iterations is given.

Result file. The file `millepede.res` contains one line per (variable or fixed) global parameter; the first three numbers have the same meaning as the lines following the `Parameter` keyword in the input text files. The first value is the label, the second is the fitted parameter value, and the third is the presigma, either copied from the input text file or zero (default value).

Debug file. The file `mpdebug.txt`, if requested by the `printrecord` keyword, contains detailed information of all data and fits for local-fit objects.

Eigenvector file. The file `millepede.eve` is written for the `method diagonalization`. It contains selected eigenvectors and their eigenvalues, and can be used e.g. for an analysis of weak modes.

2.3.6 Using Millepede II for track based alignment

2.3.6.1 Global parameters

Global parameters to be determined in a track-based alignment are mostly geometrical corrections like translation shifts and rotation angles. Usually only small corrections to the default position and angle information are necessary, and the assumption of *constant* first derivatives is sufficient (otherwise an iteration with creation of new data files

would be necessary). In general an alignment should be done simultaneously of *all* relevant detectors, perhaps using structural constraints (see below).

In addition to the geometrical corrections there are several additional parameters, which determine track-hit positions like Lorentz-angle and T_0-values, drift-velocities etc. These parameters should be included in the alignment process, since, due to the correlation between all pairs of global parameters, a separate determination of those parameters cannot be optimal. Also beam parameters like vertex position and beam direction angles should be included in the alignment, if those value are used with the tracks. Incorrectness of the model used e.g. to calculate the expected track hit positions, for example a wrong value of the Lorentz-angle, can introduce distortions in other parameters.

2.3.6.2 Constraints

Equality constraints are, in the mathematical sense, equations between global parameters, which have to be exact (within the computing accuracy). Only linear constraints are supported. Below two areas of constraints are discussed.

2.3.6.2.1 Removal of undefined degrees of freedom

A general linear transformation has 12 parameters. Three parameters of the translation and additional three parameters of a rotation are undefined, if only track residuals are minimized. At least these six parameters have to be fixed by constraining to zero the overall translation and rotation.

2.3.6.2.2 Structural constraints

A large track detector has usually a sub-detector structure: the track detector is built from smaller units, which are built by smaller sub-units etc. down to the level of a single sensor. The different levels can be visualized by a tree structure. Figure 2 shows the tree structure of the cms detector.

The tree structure can be reflected in the label structure and constraints. One set of global parameters is assigned to a unit. To each of the subunits another set of global parameters; the sum of the transformations of the subunits is forced to zero by sum constraints, which remove the otherwise undefined degrees of freedom. For a unit with N subunits, with for example 6 degrees of freedom, the total number of degrees of freedom becomes

$$[6+6\cdot N]\ \text{(parameters)} - 6\ \text{(constraints)} = 6\cdot N\ .$$

The parameters of the unit are more accurately defined than the parameters of the sub-units. Additional external measurement information from e.g. a survey can be assigned to the parameters of the unit. If all sub-unit parameters are fixed after an initial alignment, the parameters of the unit can still be variable and be improved in an alignment with a reduced total number of parameters.

Figure 2

2.3.6.3 Linear transformations in 3D

Nominal transformation. For track reconstruction the local coordinate system and the global xyz coordinate system are used. The local system with coordinate $\boldsymbol{q}$ and components u, v and w is defined w.r.t. a sensor with origin at the center of the sensor; the u-axis is along the precise and the v-axis is along the coarse coordinate, and the w-axis is normal to the sensor. The transformation from the global to the local system is given by

$$\boldsymbol{q} = \boldsymbol{R}(\boldsymbol{r} - \boldsymbol{r}_0)$$

with the definitions

$$\boldsymbol{q} = \begin{pmatrix} u \\ v \\ w \end{pmatrix} \qquad \boldsymbol{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \qquad \boldsymbol{r}_0 = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}\ .$$

The nominal position $\boldsymbol{r}_0$ and the rotation matrix $\boldsymbol{R}$ are determined by detector assembly and survey information. The convention is to use the three Euler angles ϑ, ψ and φ to determine the nominal rotation matrix $\boldsymbol{R}$:

$$\boldsymbol{R} = \begin{pmatrix} \cos\psi\cos\varphi - \cos\vartheta\sin\psi\sin\varphi & -\cos\psi\sin\varphi - \cos\vartheta\sin\psi\cos\varphi & \sin\vartheta\sin\psi \\ \sin\psi\cos\varphi + \cos\vartheta\cos\psi\sin\varphi & -\sin\psi\sin\varphi + \cos\vartheta\cos\psi\cos\varphi & -\sin\vartheta\cos\psi \\ \sin\vartheta\sin\varphi & \sin\vartheta\cos\varphi & \cos\vartheta \end{pmatrix}$$

or

$$\boldsymbol{R} = \begin{pmatrix} \cos\psi\cos\varphi - \cos\vartheta\sin\psi\sin\varphi & \sin\psi\cos\varphi + \cos\vartheta\cos\psi\sin\varphi & \sin\vartheta\sin\varphi \\ -\cos\psi\sin\varphi - \cos\vartheta\sin\psi\cos\varphi & -\sin\psi\sin\varphi + \cos\vartheta\cos\psi\cos\varphi & \sin\vartheta\cos\varphi \\ \sin\vartheta\sin\psi & -\sin\vartheta\cos\psi & \cos\vartheta \end{pmatrix}$$

The ranges of the angles are: $0 \leq \vartheta < \pi$, $0 \leq \psi < 2\pi$, and $0 \leq \phi < 2\pi$.

A different convention is to parametrize the rotation by Euler angles φ, ϑ and ψ with the rotation $\boldsymbol{R} = \boldsymbol{R}_3(\varphi)\boldsymbol{R}_2(\vartheta)\boldsymbol{R}_3(\psi)$, where $R_i(\delta)$ is a rotation by an angle δ about the axis $\boldsymbol{n}_i$. The rotation is

$$\boldsymbol{R} = \begin{pmatrix} \cos\varphi\cos\vartheta\cos\psi - \sin\varphi\sin\psi & -\sin\varphi\cos\psi - \cos\varphi\cos\vartheta\sin\psi & \cos\varphi\sin\vartheta \\ \cos\varphi\sin\psi + \sin\varphi\cos\vartheta\cos\psi & \cos\varphi\cos\psi - \sin\varphi\cos\vartheta\sin\psi & \sin\varphi\sin\vartheta \\ -\sin\vartheta\cos\psi & \sin\vartheta\sin\psi & \cos\vartheta \end{pmatrix}$$

The ranges of the angles are: $0 \leq \phi < 2\pi$, $0 \leq \vartheta < \pi$ and $0 \leq \psi < 2\pi$.

Alignment correction to the transformation. The alignment procedure determines a correction to the nominal transformation by an incremental rotation $\Delta\boldsymbol{R}$ and a translation $\Delta\boldsymbol{r}_0$. The combined translation and rotation becomes

$$\begin{aligned} \boldsymbol{r}_0 &\rightarrow \boldsymbol{r}_0 + \Delta\boldsymbol{r} \\ \boldsymbol{R} &\rightarrow \Delta\boldsymbol{R}\boldsymbol{R}\,. \end{aligned}$$

The correction matrix $\Delta\boldsymbol{R}$ is given by small rotations by $\Delta\alpha$, \, $\Delta\beta$ and $\Delta\gamma$ around the u-axis, the (new) v-axis and the (new) w-axis:

$$\Delta\boldsymbol{R} = \boldsymbol{R}_\alpha\boldsymbol{R}_\beta\boldsymbol{R}_\gamma \approx \begin{pmatrix} 1 & \Delta\gamma & -\Delta\beta \\ -\Delta\gamma & 1 & \Delta\alpha \\ \Delta\beta & -\Delta\alpha & 1 \end{pmatrix},$$

where the approximation has sufficient accuracy for small angles $\Delta\alpha$, \, $\Delta\beta$ and $\Delta\gamma$. The position correction $\Delta\boldsymbol{r}$ transforms to the local system as

$$\Delta\boldsymbol{q} = \Delta\boldsymbol{R}\,\boldsymbol{R}\Delta\boldsymbol{r} \qquad \text{with} \quad \Delta\boldsymbol{q} = \begin{pmatrix} \Delta u \\ \Delta v \\ \Delta w \end{pmatrix} \qquad \Delta\boldsymbol{r} = \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix}$$

These corrections define the corrected (aligned) transformation from global to local coordinates:

$$\boldsymbol{q}^{\text{aligned}} = \Delta\boldsymbol{R}\,\boldsymbol{R}(\boldsymbol{r} - \boldsymbol{r}_0) - \Delta\boldsymbol{q}\,.$$

The task of the alignment by tracks is to determine the correction angles $\Delta\alpha$, $\Delta\beta$ and $\Delta\gamma$ (and thus the rotation matrix $\Delta\boldsymbol{R}$) and the translation vector $\Delta\boldsymbol{r}$ or vector $\Delta\boldsymbol{q}$ for each individual detector element.

2.3.7 Acknowledgement

Markus Stoye has worked with different versions of **Millepede II** during the development phase in the last two years in alignment studies for the inner tracker of the cms experiment. I would like to thank him for the close collaboration, which was essential for the development of **Millepede II**. I would like to thank also Gero Flucke, who contributed the C $^{++}$ code, and Claus Kleinwort for the **Millepede I/II** comparison using data from the H1 experiment.

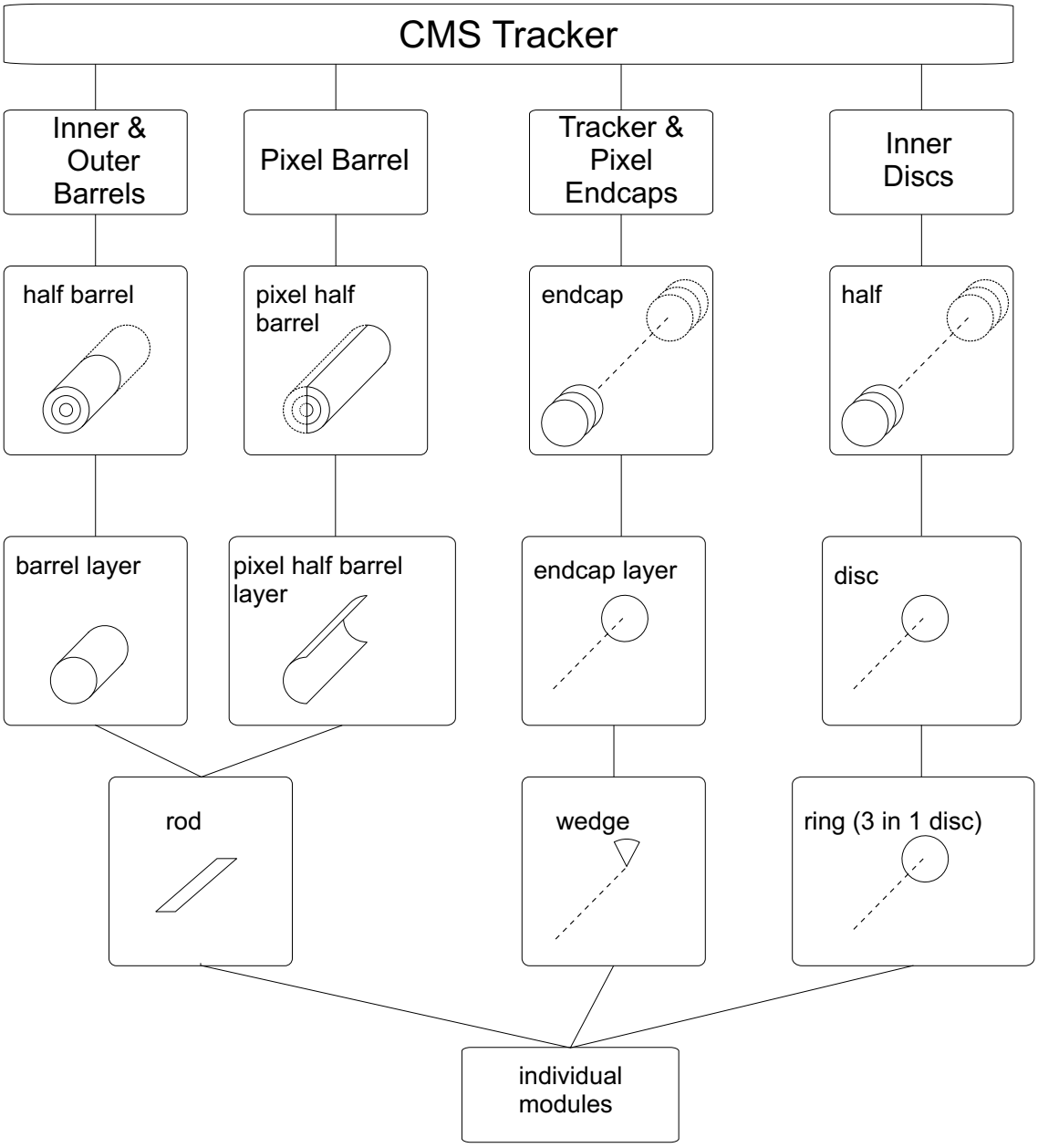

Figure 2: The tree of components of the cms inner track detectors.

Chapter 3

Major changes

Major changes with respect to the draft manual.

3.1 Solution methods

The following methods to obtain the solution $\boldsymbol{x}$ from a linear equation system $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$ are implemented:

3.1.1 Inversion

The solution and the covariance matrix $\boldsymbol{A}^{-1}$ are obtained by inversion of $\boldsymbol{A}$. Available are the value, error and global correlation for all global parameters. The matrix inversion routine has been parallelized and can be used for up to several 10000 parameters.

3.1.2 Diagonalization

The solution and the covariance matrix $\boldsymbol{A}^{-1}$ are obtained by diagonalization of $\boldsymbol{A}$. Available are the value, error, global correlation and eigenvalue (and eigenvector) for all global parameters.

3.1.3 Minimal Residual Method (MINRES)

The solution is obtained by minimizing $||\boldsymbol{A} \cdot \boldsymbol{x} - \boldsymbol{b}||_2$ iteratively. MINRES [ref 8] is a special case of the generalized minimal residual method (GMRES) for symmetric matrices. Preconditioning with a band matrix of zero or finite bandwidth is possible. Individual columns $\boldsymbol{c_i}$ of the covariance matrix can be calculated by solving $\boldsymbol{A} \cdot \boldsymbol{c}_i = \boldsymbol{1}_i$ where $\boldsymbol{1}_i$ is the i-th column on the unit matrix. The most time consuming part (product matrix times vector per iteration) has been parallelized. Available are the value for all (and optionally error, global correlation for few) global parameters.

3.1.4 Advanced Minimal Residual Method (MINRES-QLP)

The MINRES-QLP implementation [ref 9] is a MINRES evolution with improved norm estimates and stopping conditions (leading potentially to different numbers of internal iterations). Internally it uses QLP instead of the QR factorization in MINRES which should be numerically superior and allows to find for singular systems the minimal length (pseudo-inverse) solution.

The default behavior is to start (the internal iterations) with QR factorization and to switch to QLP if the (estimated) matrix condition exceeds mrtcnd. Pure QR or QLP factorization can be enforced by mrmode.

3.2 Regularization

Optionally a term $\tau \cdot ||\boldsymbol{x}||$ can be added to the objective function (to be minimized) where $\boldsymbol{x}$ is the vector of global parameters weighted with the inverse of their individual pre-sigma values.

3.3 Local fit

In case the local fit is a track fit with proper description of multiple scattering in the detector material additional local parameters have to be introduced for each scatterer and solution by *inversion* can get time consuming ($\sim n_{lp}^3$ for n_{lp} local parameters). For trajectories based on **broken lines** [ref 4,6] the corresponding matrix $\boldsymbol{\Gamma}$ has a bordered band structure ($\Gamma_{ij} = 0$ for $\min(i,j) > b$ (border size) and $|i-j| > m$ (bandwidth)). With *root-free Cholesky decomposition* the time for the solution is linear and for the calculation of $\boldsymbol{\Gamma}^{-1}$ (needed for the construction of the global matrix) quadratic in n_{lp}. For each local fit the structure of $\boldsymbol{\Gamma}$ is checked and the faster solution method selected automatically.

3.4 Parallelization

The code has been largely parallelized using OpenMP™. This includes the reading of binary files, the local fits, the construction of the sparsity structure and filling of the global matrix and the global fit (except by diagonalization). The number of threads is set by the command threads.

Caching. The records are read in blocks into a *read cache* and processed from there in parallel, each record by a single thread. For the filling of the global matrix the (zero-compressed) update matrices ($\boldsymbol{\Delta C}_1 + \boldsymbol{\Delta C}_2$ from equations (15), (16)) produced by each local fit are collected in a *write cache*. After processing the block of records this is used to update the global matrix in parallel, each row by a single thread. The total cache size can be changed by the command cache.

3.5 Compressed sparse matrix

In sparse storage mode for each row the list of column indices (and values) for the non-zero elements are stored. With compression regions of continous column indices are represented by the first index and their number (packed into a single 32bit integer). Compression is selected by the command compress. In addition rare elements can be neglected (,histogrammed) or stored in single instead of double precision according to the pairentries command.

3.6 Gzipped C binary files

The `zlib` can be used to directly read *gzipped* C binary files. In this case reading with multiple threads (each file by single thread) can speed up the decompression.

3.7 Transformation from FORTRAN77 to Fortran90

The **Millepede** source code has been formally transformed from *fixed form* FORTRAN77 to *free form* Fortran90 (using TO_F90 by Alan Miller) and (most parts) modernized:

- `IMPLICIT NONE` everywhere. Unused variables removed.
- `COMMON` blocks replaced by `MODULEs`.
- Backward `GOTOs` replaced by proper `DO` loops.
- `INTENT` (input/output) of arguments described.

- Code documented with doxygen.

Unused parts of the code (like the interactive mode) have been removed. The reference compiler for the Fortran90 version is gcc-4.6.2 (gcc-4.4.4 works too).

3.8 Memory management

The memory management for dynamic data structures (matrices, vectors, ..) has been changed from a subdivided *static* `COMMON` block to *dynamic* (`ALLOCATABLE`) Fortran90 arrays. One **Pede** executable is now sufficient for all application sizes.

3.9 Read buffer size

In the first loop over all binary files a preset read buffer size is used. Too large records are skipped, but the maximal record length is still being updated. If any records had to be skipped the read buffer size is afterwards adjusted according to the maximal record length and the first loop is repeated.

3.10 Number of binary files

The number of binary files has no hard-coded limit anymore, but is calculated from the steering file and resources (file names, descriptors, ..) are allocated dynamically. Some resources may be limited by the system.

Chapter 4

List of options and commands

4.1 Command line options:

4.1.1 -t

Create text and binary files for wire chamber test case, set ictest to 1.

4.1.2 -t=track-model

Create text and binary files for silicon strip tracker test case using *track-models* with different accounting for multiple scattering, set ictest to 2..6.

4.1.3 -s

Solution is not iterated. Automatically switched on in case of rank deficits for constraints.

4.1.4 -f

Force iterating of solution (in case of rank deficits for constraints).

4.2 Steering file commands:

In general the commands are defined by a single line:

```
keyword   number1  number2  ...
```

For those specifying properties of the global parameters (*keyword* = `parameter`, `constraint` or `measurement`) for each involved global parameter (identified by a label) one additional line follows:

```
label     number1  number2  ...
```

Default values for the numerical arguments are shown in the command descriptions in '[]'. Missing arguments without default values have no effect.

4.2.1 bandwidth

Set band width mbandw for MINRES preconditioner to *number1*.

4.2.2 cache

Set (read+write) cache size ncache to *number1*. Define cache size and average fill level.

4.2.3 Cfiles

Following binaries are C files.

4.2.4 chisqcut

For local fit setChi^2 cut chicut to *number1* [1.], chirem to *number2* [1.].

4.2.5 compress

Set compression flag mcmprs for sparse storage to 1 (true) (and msngpe to 1).

4.2.6 constraint

Define constraints for global parameters.

4.2.7 debug

Set number of records with debug printout mdebug to *number1* [3], number of measurements with printout mdebg2 to *number2*.

4.2.8 dwfractioncut

Set down-weighting fraction cut dwcut to *number1* (max. 0.5).

4.2.9 entries

Set entries cut for variable global parameter mreqen to *number1* [0].

4.2.10 errlabels

Define (up to 100 in total) global labels *number1* .. *numberN* for which the parameter errors are calculated for method MINRES too (by solving $\boldsymbol{C} \cdot \boldsymbol{x}_i = \boldsymbol{b}^i, b^i_j = \delta_{ij}$).

4.2.11 force

Set force (iterations) flag iforce to 1 (true). Same as -f.

4.2.12 fortranfiles

Following binaries are Fortran files.

4.2.13 globalcorr

Set flag igcorr for output of global correlations to 1 (true).

4.2.14 histprint

Set flag nhistp for histogram printout to 1 (true).

4.2.15 hugecut

For local fit set Chi^2 cut chhuge for unreasonable data to *number1* [1.].

4.2.16 linesearch

The mode lsearch of the line search to improve the solution is set to *number1*.

4.2.17 localfit

For local fit set number of iterations lfitnp with calculation of pulls to *number1*, flag lfitbb for auto-detection of bordered band matrices to *number2*.

4.2.18 matiter

Set number of iterations matrit with (re)calcuation of global matrix to *number1*.

4.2.19 matmoni

Set record interval matmon for monitoring of (sparse) matrix construction to *number1*.

4.2.20 maxrecord

Set record limit mxrec to *number1*.

4.2.21 measurement

Define (additional) measurements for global parameters.

4.2.22 memorydebug

Set debug flag memdbg for memory management to *number1* [1].

4.2.23 method

Has special format:

```
method   name     number1  number2
```

Set solution method metsol and storage mode matsto according to *name*, (`inversion` : (1,1), `diagonalization` : (2,1), `fullMINRES` : (3,1) or `sparseMINRES` : (3,2), `fullMINRES-QLP` : (4,1) or `sparseMINRES-QLP` : (4,2)), (minimum) number of iterations mitera to *number1*, convergence limit dflim to *number2*.

4.2.24 mresmode

Set MINRES-QLP factorization mode mrmode to *number1*.

4.2.25 mrestranscond

Set MINRES-QLP transition (matrix) condition mrtcnd to *number1*.

4.2.26 mrestol

Set tolerance criterion mrestl for MINRES to *number1* (10^{-10} .. 10^{-4}).

4.2.27 nofeasiblestart

Set flag nofeas for skipping making parameters feasible to *number1* [1].

4.2.28 outlierdownweighting

For local fit set number of outlier down-weighting iterations lhuber to *number1*.

4.2.29 pairentries

Set entries cut for variable global parameter pairs mreqpe to *number1*, histogram upper bound mhispe for pairs to *number2* ($<$1: no histogramming), upper bound msngpe for pair entries with single precision storage to *number3*.

4.2.30 parameter

Define initial value, pre-sigma for global parameters.

4.2.31 presigma

Set default pre-sigma regpre to *number1* [1].

4.2.32 print

Set print level mprint to *number1* [1].

4.2.33 printrecord

Record numbers with printout.

4.2.34 pullrange

Set (symmetric) range prange for histograms of pulls, normalized residuals to *number1* (=0: auto-ranging).

4.2.35 regularisation

Set flag nregul for regularization to 1 (true), regularization parameter regula to *number2*, default pre-sigma regpre to *number3*.

4.2.36 regularization

Set flag nregul for regularization to 1 (true), regularization parameter regula to *number2*, default pre-sigma regpre to *number3*.

4.2.37 subito

Set subito (no iterations) flag isubit to 1 (true). Same as -s.

4.2.38 threads

Set number mthrd of OpenMP™ threads for processing to *number1*, number mthrdr of threads for reading binary files to *number2* [*number1*].

4.2.39 wconstraint

Define weighted constraints for global parameters.

4.2.40 wolfe

For strong Wolfe condition in line search set parameter wolfc1 to *number1*, wolfc2 to *number2*.

Chapter 5

List of exit codes

The exit code and message of the **Pede** executable can be found in the file `millepede.end`:

- **-1** Still running or crashed
- **00** Ended normally
- **01** Ended with warnings (bad measurements)
- **02** Ended with severe warnings (bad global matrix)
- **10** Aborted, no steering file
- **11** Aborted, open error for steering file
- **12** Aborted, second text file in command line
- **13** Aborted, unknown keywords in steering file
- **14** Aborted, no binary files
- **15** Aborted, open error(s) for binary files
- **16** Aborted, open error(s) for text files
- **17** Aborted, file name too long
- **20** Aborted, bad binary records
- **21** Aborted, no labels/parameters defined
- **22** Aborted, no variable global parameters
- **23** Aborted, bad matrix index
- **24** Aborted, vector/matrix size mismatch
- **25** Aborted, result vector contains NaNs
- **26** Aborted, too many rejects
- **30** Aborted, memory allocation failed
- **31** Aborted, memory deallocation failed
- **32** Aborted, iteration limit reached in diagonalization
- **33** Aborted, stack overflow in quicksort
- **34** Aborted, pattern string too long

Chapter 6

Data Type Index

6.1 Data Types List

Here are the data types with brief descriptions:

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Chapter 8

Data Type Documentation

8.1 linesrch Module Reference

Line search data.

Public Attributes

- integer(mpi), parameter msfd =20
- integer(mpi) nsfd

  number of function calls
- integer(mpi) idgl

  index of smallest negative slope
- integer(mpi) idgr

  index of smallest positive slope
- integer(mpi) idgm

  index of minimal slope
- integer(mpi) minf =1

  min. number of function calls
- integer(mpi) maxf =5

  max. number of function calls
- integer(mpi) lsinfo

  (status) information
- real(mpd), dimension(4, msfd) sfd

  abscissa; function value; slope; predicted zero
- real(mpd) stmx =0.9

  maximum slope ratio
- real(mpd) gtol

  slope ratio

8.1.1 Detailed Description

Line search data.

Definition at line 33 of file linesrch.f90.

8.1.2 Member Data Documentation

8.1.2.1 real(mpd) linesrch::gtol

slope ratio

Definition at line 48 of file linesrch.f90.

8.1.2.2 integer(mpi) linesrch::idgl

index of smallest negative slope

Definition at line 40 of file linesrch.f90.

8.1.2.3 integer(mpi) linesrch::idgm

index of minimal slope

Definition at line 42 of file linesrch.f90.

8.1.2.4 integer(mpi) linesrch::idgr

index of smallest positive slope

Definition at line 41 of file linesrch.f90.

8.1.2.5 integer(mpi) linesrch::lsinfo

(status) information

Definition at line 45 of file linesrch.f90.

8.1.2.6 integer(mpi) linesrch::maxf =5

max. number of function calls

Definition at line 44 of file linesrch.f90.

8.1.2.7 integer(mpi) linesrch::minf =1

min. number of function calls

Definition at line 43 of file linesrch.f90.

8.1.2.8 integer(mpi), parameter linesrch::msfd =20

Definition at line 38 of file linesrch.f90.

8.1.2.9 integer(mpi) linesrch::nsfd

number of function calls

Definition at line 39 of file linesrch.f90.

8.1.2.10 real(mpd), dimension(4,msfd) linesrch::sfd

abscissa; function value; slope; predicted zero

Definition at line 46 of file linesrch.f90.

8.1.2.11 real(mpd) linesrch::stmx =0.9

maximum slope ratio

Definition at line 47 of file linesrch.f90.

The documentation for this module was generated from the following file:

- linesrch.f90

8.2 mpdef::listitem Type Reference

list items from steering file

Public Attributes

- integer(mpi) label
- real(mps) value

8.2.1 Detailed Description

list items from steering file

Definition at line 17 of file mpdef.f90.

8.2.2 Member Data Documentation

8.2.2.1 integer(mpi) mpdef::listitem::label

Definition at line 18 of file mpdef.f90.

8.2.2.2 real(mps) mpdef::listitem::value

Definition at line 19 of file mpdef.f90.

The documentation for this type was generated from the following file:

- mpdef.f90

8.3 Mille Class Reference

Class to write C binary file.

`#include <Mille.h>`

Public Member Functions

- Mille (const char *outFileName, bool asBinary=true, bool writeZero=false)

  Opens outFileName (by default as binary file).
- ~Mille ()

  Closes file.
- void mille (int NLC, const float *derLc, int NGL, const float *derGl, const int *label, float rMeas, float sigma)

  Add measurement to buffer.
- void special (int nSpecial, const float *floatings, const int *integers)

  Add special data to buffer.
- void kill ()

  Reset buffers, i.e. kill derivatives accumulated for current set.
- void end ()

  Write buffer (set of derivatives with same local parameters) to file.

Private Types

- enum { myBufferSize = 5000 }

  buffer size for ints and floats
- enum { myMaxLabel = (0xFFFFFFFF - (1 << 31)) }

  largest label allowed: 2^31 - 1

Private Member Functions

- void newSet ()

  Initialize for new set of locals, e.g. new track.
- bool checkBufferSize (int nLocal, int nGlobal)

  Enough space for next nLocal + nGlobal derivatives incl. measurement?

Private Attributes

- std::ofstream myOutFile

  C-binary for output.
- bool myAsBinary

  if false output as text
- bool myWriteZero
- int myBufferInt [myBufferSize]

  to collect labels etc.
- float myBufferFloat [myBufferSize]

  to collect derivatives etc.
- int myBufferPos

  position in buffer
- bool myHasSpecial

8.3.1 Detailed Description

Class to write C binary file.

Class to write a C binary (cf. below) file of a given name and to fill it with information used as input to **pede**. Use its member functions `mille()`, `special()`, `kill()` and `end()` as you would use the fortran MILLE and its entry points `MILLSP`, `KILLE` and `ENDLE`.

For debugging purposes constructor flags enable switching to text output and/or to write also derivatives and labels which are ==0. But note that **pede** will not be able to read text output and has not been tested with derivatives/labels ==0.

Author

: Gero Flucke date : October 2006

Revision:

1.3

Date:

2007/04/16 17:47:38

(last update by

Author:

flucke

)

Definition at line 28 of file Mille.h.

8.3.2 Member Enumeration Documentation

8.3.2.1 anonymous enum [private]

buffer size for ints and floats

Enumerator

myBufferSize

Definition at line 48 of file Mille.h.

8.3.2.2 anonymous enum [private]

largest label allowed: 2^31 - 1

Enumerator

myMaxLabel

Definition at line 54 of file Mille.h.

8.3.3 Constructor & Destructor Documentation

8.3.3.1 Mille::Mille (const char * *outFileName*, bool *asBinary* = `true`, bool *writeZero* = `false`)

Opens outFileName (by default as binary file).

Parameters

in	*outFileName*	file name
in	*asBinary*	flag for binary
in	*writeZero*	flag for keeping of zeros

Definition at line 25 of file Mille.cc.

References myBufferFloat, myBufferInt, and myOutFile.

8.3.3.2 Mille::~Mille ( )

Closes file.

Definition at line 42 of file Mille.cc.

References myOutFile.

8.3.4 Member Function Documentation

8.3.4.1 bool Mille::checkBufferSize (int *nLocal*, int *nGlobal*) [private]

Enough space for next nLocal + nGlobal derivatives incl. measurement?

Parameters

in	*nLocal*	number of local derivatives
in	*nGlobal*	number of global derivatives

Returns

true if sufficient space available (else false)

Definition at line 194 of file Mille.cc.

References myBufferInt, myBufferPos, and myBufferSize.

Referenced by mille(), and special().

8.3.4.2 void Mille::end ( )

Write buffer (set of derivatives with same local parameters) to file.

Definition at line 149 of file Mille.cc.

References myAsBinary, myBufferFloat, myBufferInt, myBufferPos, and myOutFile.

8.3.4.3 void Mille::kill ( )

Reset buffers, i.e. kill derivatives accumulated for current set.

Definition at line 142 of file Mille.cc.

References myBufferPos.

8.3.4.4 void Mille::mille (int *NLC*, const float * *derLc*, int *NGL*, const float * *derGl*, const int * *label*, float *rMeas*, float *sigma*)

Add measurement to buffer.

Parameters

in	*NLC*	number of local derivatives
in	*derLc*	local derivatives
in	*NGL*	number of global derivatives
in	*derGl*	global derivatives
in	*label*	global labels
in	*rMeas*	measurement (residuum)
in	*sigma*	error

Definition at line 58 of file Mille.cc.

References checkBufferSize(), myBufferFloat, myBufferInt, myBufferPos, myMaxLabel, myWriteZero, and newSet().

8.3.4.5 void Mille::newSet ( ) [private]

Initialize for new set of locals, e.g. new track.

Definition at line 179 of file Mille.cc.

References myBufferFloat, myBufferInt, myBufferPos, and myHasSpecial.

Referenced by mille(), and special().

8.3.4.6 void Mille::special (int *nSpecial,* const float * *floatings,* const int * *integers*)

Add special data to buffer.

Parameters

in	*nSpecial*	number of floats/ints
in	*floatings*	floats
in	*integers*	ints

Definition at line 107 of file Mille.cc.

References checkBufferSize(), myBufferFloat, myBufferInt, myBufferPos, myHasSpecial, and newSet().

8.3.5 Member Data Documentation

8.3.5.1 bool Mille::myAsBinary [private]

if false output as text

Definition at line 45 of file Mille.h.

Referenced by end().

8.3.5.2 float Mille::myBufferFloat[myBufferSize] [private]

to collect derivatives etc.

Definition at line 50 of file Mille.h.

Referenced by end(), Mille(), mille(), newSet(), and special().

8.3.5.3 int Mille::myBufferInt[myBufferSize] [private]

to collect labels etc.

Definition at line 49 of file Mille.h.

Referenced by checkBufferSize(), end(), Mille(), mille(), newSet(), and special().

8.3.5.4 int Mille::myBufferPos [private]

position in buffer

Definition at line 51 of file Mille.h.

Referenced by checkBufferSize(), end(), kill(), mille(), newSet(), and special().

8.3.5.5 bool Mille::myHasSpecial [private]

if true, special(..) already called for this record

Definition at line 52 of file Mille.h.

Referenced by newSet(), and special().

8.3.5.6 std::ofstream Mille::myOutFile [private]

C-binary for output.

Definition at line 44 of file Mille.h.

Referenced by end(), Mille(), and ~Mille().

8.3.5.7 bool Mille::myWriteZero [private]

if true also write out derivatives/labels ==0

Definition at line 46 of file Mille.h.

Referenced by mille().

The documentation for this class was generated from the following files:

- Mille.h
- Mille.cc

8.4 minresdatamodule Module Reference

Defines real(kind=dp) and a few constants for use in other modules.

Public Attributes

- integer, parameter, public dp = mpd
- real(kind=dp), parameter, public zero = 0.0_dp
- real(kind=dp), parameter, public one = 1.0_dp

8.4.1 Detailed Description

Defines real(kind=dp) and a few constants for use in other modules.

Definition at line 19 of file minresDataModule.f90.

8.4.2 Member Data Documentation

8.4.2.1 integer, parameter, public minresdatamodule::dp = mpd

Definition at line 25 of file minresDataModule.f90.

8.4.2.2 real(kind=dp), parameter, public minresdatamodule::one = 1.0_dp

Definition at line 26 of file minresDataModule.f90.

8.4.2.3 real(kind=dp), parameter, public minresdatamodule::zero = 0.0_dp

Definition at line 26 of file minresDataModule.f90.

The documentation for this module was generated from the following file:

- minresDataModule.f90

8.5 minresmodule Module Reference

MINRES solves symmetric systems Ax = b or min ||Ax - b||_2, where the matrix A may be indefinite and/or singular.

Public Member Functions

- subroutine, public minres (n, Aprod, Msolve, b, shift, checkA, precon, x, itnlim, nout, rtol, istop, itn, Anorm, Acond, rnorm, Arnorm, ynorm)

  Solution of linear equation system.

8.5.1 Detailed Description

MINRES solves symmetric systems Ax = b or min ||Ax - b||_2, where the matrix A may be indefinite and/or singular.

```
The software for MINRES (f90 version) is provided by SOL, Stanford University
under the terms of the OSI Common Public License (CPL):
http://www.opensource.org/licenses/cpl1.0.php

Contributors:
    Chris Paige <chris@cs.mcgill.ca>
    Sou-Cheng Choi <scchoi@stanford.edu>

    Michael Saunders <saunders@stanford.edu>
    Systems Optimization Laboratory (SOL)
    Stanford University
    Stanford, CA 94305-4026, USA
    (650)723-1875

09 Oct 2007: F90 version constructed from the F77 version.
             Initially used compiler option -r8, but this is nonstandard.
15 Oct 2007: Test on Arnorm = ||Ar|| added to recognize singular systems.
15 Oct 2007: Temporarily used real(8) everywhere.
16 Oct 2007: Use minresDataModule to define dp = selected_real_kind(15).
             We need "use minresDataModule"
             at the beginning of modules AND inside interfaces.

             g95 compiles successfully with the following options:
  g95 -c -g -O0 -pedantic -Wall -Wextra -fbounds-check -ftrace=full minresModule.f90

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Definition at line 39 of file minresModule.f90.

8.5.2 Member Function/Subroutine Documentation

8.5.2.1 subroutine, public minresmodule::minres (integer, intent(in) *n,* *Aprod,* *Msolve,* real(dp), dimension(n), intent(in) *b,* real(dp), intent(in) *shift,* logical, intent(in) *checkA,* logical, intent(in) *precon,* real(dp), dimension(n), intent(out) *x,* integer, intent(in) *itnlim,* integer, intent(in) *nout,* real(dp), intent(in) *rtol,* integer, intent(out) *istop,* integer, intent(out) *itn,* real(dp), intent(out) *Anorm,* real(dp), intent(out) *Acond,* real(dp), intent(out) *rnorm,* real(dp), intent(out) *Arnorm,* real(dp), intent(out) *ynorm*)

Solution of linear equation system.

```
-----------------------------------------------------------------

 MINRES  is designed to solve the system of linear equations

    Ax = b

 or the least-squares problem

    min ||Ax - b||_2,

 where A is an n by n symmetric matrix and b is a given vector.
 The matrix A may be indefinite and/or singular.

 1. If A is known to be positive definite, the Conjugate Gradient
 Method might be preferred, since it requires the same number
 of iterations as MINRES but less work per iteration.

 2. If A is indefinite but Ax = b is known to have a solution
 (e.g. if A is nonsingular), SYMMLQ might be preferred,
 since it requires the same number of iterations as MINRES
 but slightly less work per iteration.

 The matrix A is intended to be large and sparse.  It is accessed
 by means of a subroutine call of the form
 SYMMLQ development:

    call Aprod ( n, x, y )

 which must return the product y = Ax for any given vector x.


 More generally, MINRES is designed to solve the system

    (A - shift*I) x = b
 or
    min ||(A - shift*I) x - b||_2,

 where  shift  is a specified scalar value.  Again, the matrix
 (A - shift*I) may be indefinite and/or singular.
 The work per iteration is very slightly less if  shift = 0.

 Note: If  shift  is an approximate eigenvalue of  A
 and  b  is an approximate eigenvector,  x  might prove to be
 a better approximate eigenvector, as in the methods of
 inverse iteration and/or Rayleigh-quotient iteration.
 However, we're not yet sure on that -- it may be better to use SYMMLQ.

 A further option is that of preconditioning, which may reduce
 the number of iterations required.  If M = C C' is a positive
 definite matrix that is known to approximate  (A - shift*I)
 in some sense, and if systems of the form  My = x  can be
 solved efficiently, the parameters precon and Msolve may be
 used (see below).  When  precon = .true., MINRES will
 implicitly solve the system of equations

    P (A - shift*I) P' xbar  =  P b,

 i.e.             Abar xbar  =  bbar
 where               P  =  C**(-1),
                  Abar  =  P (A - shift*I) P',
                  bbar  =  P b,
```

```
and return the solution       x  =  P' xbar.
The associated residual is rbar  =  bbar - Abar xbar
                                 =  P (b - (A - shift*I)x)
                                 =  P r.

In the discussion below, eps refers to the machine precision.

Parameters
----------

n       input      The dimension of the matrix A.
b(n)    input      The rhs vector b.
x(n)    output     Returns the computed solution x.

Aprod   external   A subroutine defining the matrix A.
                      call Aprod ( n, x, y )
                   must return the product y = Ax
                   without altering the vector x.

Msolve  external   An optional subroutine defining a
                   preconditioning matrix M, which should
                   approximate (A - shift*I) in some sense.
                   M must be positive definite.

                      call Msolve( n, x, y )

                   must solve the linear system My = x
                   without altering the vector x.

                   In general, M should be chosen so that Abar has
                   clustered eigenvalues.  For example,
                   if A is positive definite, Abar would ideally
                   be close to a multiple of I.
                   If A or A - shift*I is indefinite, Abar might
                   be close to a multiple of diag( I  -I ).

checkA  input      If checkA = .true., an extra call of Aprod will
                   be used to check if A is symmetric.  Also,
                   if precon = .true., an extra call of Msolve
                   will be used to check if M is symmetric.

precon  input      If precon = .true., preconditioning will
                   be invoked.  Otherwise, subroutine Msolve
                   will not be referenced; in this case the
                   actual parameter corresponding to Msolve may
                   be the same as that corresponding to Aprod.

shift   input      Should be zero if the system Ax = b is to be
                   solved.  Otherwise, it could be an
                   approximation to an eigenvalue of A, such as
                   the Rayleigh quotient b'Ab / (b'b)
                   corresponding to the vector b.
                   If b is sufficiently like an eigenvector
                   corresponding to an eigenvalue near shift,
                   then the computed x may have very large
                   components.  When normalized, x may be
                   closer to an eigenvector than b.

nout    input      A file number.
                   If nout > 0, a summary of the iterations
                   will be printed on unit nout.

itnlim  input      An upper limit on the number of iterations.

rtol    input      A user-specified tolerance.  MINRES terminates
                   if it appears that norm(rbar) is smaller than
                      rtol * norm(Abar) * norm(xbar),
                   where rbar is the transformed residual vector,
                      rbar = bbar - Abar xbar.

                   If shift = 0 and precon = .false., MINRES
                   terminates if norm(b - A*x) is smaller than
```

```
                    rtol * norm(A) * norm(x).

istop   output      An integer giving the reason for termination...

         -1         beta2 = 0 in the Lanczos iteration; i.e. the
                    second Lanczos vector is zero.  This means the
                    rhs is very special.
                    If there is no preconditioner, b is an
                    eigenvector of A.
                    Otherwise (if precon is true), let My = b.
                    If shift is zero, y is a solution of the
                    generalized eigenvalue problem Ay = lambda My,
                    with lambda = alpha1 from the Lanczos vectors.

                    In general, (A - shift*I)x = b
                    has the solution         x = (1/alpha1) y
                    where My = b.

          0         b = 0, so the exact solution is x = 0.
                    No iterations were performed.

          1         Norm(rbar) appears to be less than
                    the value  rtol * norm(Abar) * norm(xbar).
                    The solution in  x  should be acceptable.

          2         Norm(rbar) appears to be less than
                    the value  eps * norm(Abar) * norm(xbar).
                    This means that the residual is as small as
                    seems reasonable on this machine.

          3         Norm(Abar) * norm(xbar) exceeds norm(b)/eps,
                    which should indicate that x has essentially
                    converged to an eigenvector of A
                    corresponding to the eigenvalue shift.

          4         Acond (see below) has exceeded 0.1/eps, so
                    the matrix Abar must be very ill-conditioned.
                    x may not contain an acceptable solution.

          5         The iteration limit was reached before any of
                    the previous criteria were satisfied.

          6         The matrix defined by Aprod does not appear
                    to be symmetric.
                    For certain vectors y = Av and r = Ay, the
                    products y'y and r'v differ significantly.

          7         The matrix defined by Msolve does not appear
                    to be symmetric.
                    For vectors satisfying My = v and Mr = y, the
                    products y'y and r'v differ significantly.

          8         An inner product of the form  x' M**(-1) x
                    was not positive, so the preconditioning matrix
                    M does not appear to be positive definite.

                    If istop >= 5, the final x may not be an
                    acceptable solution.

itn     output      The number of iterations performed.

Anorm   output      An estimate of the norm of the matrix operator
                    Abar = P (A - shift*I) P',   where P = C**(-1).

Acond   output      An estimate of the condition of Abar above.
                    This will usually be a substantial
                    under-estimate of the true condition.

rnorm   output      An estimate of the norm of the final
                    transformed residual vector,
                       P (b  -  (A - shift*I) x).

ynorm   output      An estimate of the norm of xbar.
```

```
                  This is sqrt( x'Mx ).  If precon is false,
                  ynorm is an estimate of norm(x).
-----------------------------------------------------------------
 MINRES is an implementation of the algorithm described in
 the following reference:

 C. C. Paige and M. A. Saunders (1975),
 Solution of sparse indefinite systems of linear equations,
 SIAM J. Numer. Anal. 12(4), pp. 617-629.
-----------------------------------------------------------------


 MINRES development:
    1972: First version, similar to original SYMMLQ.
          Later lost @#%*!!
    Oct 1995: Tried to reconstruct MINRES from
              1995 version of SYMMLQ.
 30 May 1999: Need to make it more like LSQR.
              In middle of major overhaul.
 19 Jul 2003: Next attempt to reconstruct MINRES.
              Seems to need two vectors more than SYMMLQ.  (w1, w2)
              Lanczos is now at the top of the loop,
              so the operator Aprod is called in just one place
              (not counting the initial check for symmetry).
 22 Jul 2003: Success at last.  Preconditioning also works.
              minres.f added to http://www.stanford.edu/group/SOL/.

 16 Oct 2007: Added a stopping rule for singular systems,
              as derived in Sou-Cheng Choi's PhD thesis.
              Note that ||Ar|| small => r is a null vector for A.
              Subroutine minrestest2 in minresTestModule.f90
              tests this option.  (NB: Not yet working.)
-----------------------------------------------------------------
```

Definition at line 294 of file minresModule.f90.

References precon().

Referenced by mminrs(), and solglo().

The documentation for this module was generated from the following file:

- minresModule.f90

8.6 minresqlpblasmodule Module Reference

Public Member Functions

- real(dp) function, public ddot (n, dx, incx, dy, incy)

  DDOT forms the dot product of two vectors.
- real(dp) function, public dnrm2 (n, x, incx)

  DNRM2 returns the euclidean norm of a vector.

8.6.1 Detailed Description

Definition at line 34 of file minresqlpBlasModule.f90.

8.6.2 Member Function/Subroutine Documentation

8.6.2.1 real(dp) function, public minresqlpblasmodule::ddot (integer(ip), intent(in) *n,* real(dp), dimension(∗), intent(in) *dx,* integer(ip), intent(in) *incx,* real(dp), dimension(∗), intent(in) *dy,* integer(ip), intent(in) *incy*)

DDOT forms the dot product of two vectors.

Definition at line 87 of file minresqlpBlasModule.f90.

8.6.2.2 real(dp) function, public minresqlpblasmodule::dnrm2 (integer(ip), intent(in) *n,* real(dp), dimension(∗), intent(in) *x,* integer(ip), intent(in) *incx*)

DNRM2 returns the euclidean norm of a vector.

Definition at line 184 of file minresqlpBlasModule.f90.

Referenced by minresqlpmodule::minresqlp().

The documentation for this module was generated from the following file:

- minresqlpBlasModule.f90

8.7 minresqlpdatamodule Module Reference

Defines precision and range in real(kind=dp) and integer(kind=ip) for portability and a few constants for use in other modules.

Public Attributes

- integer, parameter, public dp = mpd
- integer, parameter, public sp = mps
- integer, parameter, public ip = mpi
- real(dp), parameter, public zero = 0.0_dp
- real(dp), parameter, public one = 1.0_dp
- real(dp), parameter, public eps = epsilon(zero)
- real(dp), parameter, public realmin = tiny(one)
- real(dp), parameter, public realmax = huge(one)
- integer, parameter, public prcsn = precision(zero)
- logical, public debug = .false.
- logical, public testsymortho = .true.
- logical, public testmtx = .true.

8.7.1 Detailed Description

Defines precision and range in real(kind=dp) and integer(kind=ip) for portability and a few constants for use in other modules.

Definition at line 35 of file minresqlpDataModule.f90.

8.7.2 Member Data Documentation

8.7.2.1 logical, public minresqlpdatamodule::debug = .false.

Definition at line 55 of file minresqlpDataModule.f90.

8.7.2.2 integer, parameter, public minresqlpdatamodule::dp = mpd

Definition at line 43 of file minresqlpDataModule.f90.

8.7.2.3 real(dp), parameter, public minresqlpdatamodule::eps = epsilon(zero)

Definition at line 49 of file minresqlpDataModule.f90.

8.7.2.4 integer, parameter, public minresqlpdatamodule::ip = mpi

Definition at line 47 of file minresqlpDataModule.f90.

8.7.2.5 real(dp), parameter, public minresqlpdatamodule::one = 1.0_dp

Definition at line 49 of file minresqlpDataModule.f90.

8.7.2.6 integer, parameter, public minresqlpdatamodule::prcsn = precision(zero)

Definition at line 52 of file minresqlpDataModule.f90.

8.7.2.7 real(dp), parameter, public minresqlpdatamodule::realmax = huge(one)

Definition at line 50 of file minresqlpDataModule.f90.

8.7.2.8 real(dp), parameter, public minresqlpdatamodule::realmin = tiny(one)

Definition at line 50 of file minresqlpDataModule.f90.

8.7.2.9 integer, parameter, public minresqlpdatamodule::sp = mps

Definition at line 44 of file minresqlpDataModule.f90.

8.7.2.10 logical, public minresqlpdatamodule::testmtx = .true.

Definition at line 56 of file minresqlpDataModule.f90.

8.7.2.11 logical, public minresqlpdatamodule::testsymortho = .true.

Definition at line 56 of file minresqlpDataModule.f90.

8.7.2.12 real(dp), parameter, public minresqlpdatamodule::zero = 0.0_dp

Definition at line 49 of file minresqlpDataModule.f90.

The documentation for this module was generated from the following file:

- minresqlpDataModule.f90

8.8 minresqlpmodule Module Reference

MINRESQLP solves symmetric systems Ax = b or min ||Ax - b||_2, where the matrix A may be indefinite and/or singular. "A" is really (A - shift∗I), where shift is an input real scalar.

Public Member Functions

- subroutine, public minresqlp (n, Aprod, b, shift, Msolve, disable, nout, itnlim, rtol, maxxnorm, trancond, Acondlim, x, istop, itn, rnorm, Arnorm, xnorm, Anorm, Acond)

  Solution of linear equation system or least squares problem.
- subroutine, public symortho (a, b, c, s, r)

  SymOrtho: Stable Householder reflection.

8.8.1 Detailed Description

MINRESQLP solves symmetric systems Ax = b or min ||Ax - b||_2, where the matrix A may be indefinite and/or singular. "A" is really (A - shift∗I), where shift is an input real scalar.

```
 09 Sep 2013: Version 27
-----------------------------------------------------------------

 The software for MINRES-QLP is provided by SOL, Stanford University
 under the terms of the OSI Common Public License (CPL)
    http://www.opensource.org/licenses/cpl1.0.php
 or the BSD License
    http://www.opensource.org/licenses/bsd-license.php

-----------------------------------------------------------------

 Authors:
     Sou-Cheng Choi <sctchoi@uchicago.edu>
     Computation Institute (CI)
     University of Chicago
     Chicago, IL 60637, USA

     Michael Saunders <saunders@stanford.edu>
     Systems Optimization Laboratory (SOL)
     Stanford University
     Stanford, CA 94305-4026, USA

 Contributor:
     Christopher Paige <paige@cs.mcgill.ca>

   See also: Makefile
```

Definition at line 42 of file minresqlpModule.f90.

8.8.2 Member Function/Subroutine Documentation

8.8.2.1 subroutine, public minresqlpmodule::minresqlp (integer(ip), intent(in) *n,* *Aprod,* real(dp), dimension(n), intent(in) *b,* real(dp), intent(in), optional *shift,* optional *Msolve,* logical, intent(in), optional *disable,* integer(ip), intent(in), optional *nout,* integer(ip), intent(in), optional *itnlim,* real(dp), intent(in), optional *rtol,* real(dp), intent(in), optional *maxxnorm,* real(dp), intent(in), optional *trancond,* real(dp), intent(in), optional *Acondlim,* real(dp), dimension(n), intent(out) *x,* integer(ip), intent(out), optional *istop,* integer(ip), intent(out), optional *itn,* real(dp), intent(out), optional *rnorm,* real(dp), intent(out), optional *Arnorm,* real(dp), intent(out), optional *xnorm,* real(dp), intent(out), optional *Anorm,* real(dp), intent(out), optional *Acond*)

Solution of linear equation system or least squares problem.

```
-----------------------------------------------------------------

 MINRESQLP  is designed to solve the system of linear equations

                Ax = b

     or the least-squares problem
```

```
    min || Ax - b ||_2,

where A is an n by n symmetric matrix and b is a given vector.
The matrix A may be indefinite and/or singular.

1. If A is known to be positive definite, the Conjugate Gradient
   Method might be preferred, since it requires roughly the same
   number of iterations as MINRESQLP but less work per iteration.
   But if a low-accuracy solution is adequate, MINRESQLP will
   terminate sooner.

2. If A is indefinite but Ax = b is known to have a solution
   (e.g. if A is nonsingular), SYMMLQ might be preferred,
   since it requires roughly the same number of iterations as
   MINRESQLP but slightly less work per iteration.

3. If A is indefinite and well-conditioned, and Ax = b has a
   solution, i.e., it is not a least-squares problem, MINRES might
   be preferred as it requires the same number of iterations as
   MINRESQLP but slightly less work per iteration.

The matrix A is intended to be large and sparse.  It is accessed
by means of a subroutine call of the form

           call Aprod ( n, x, y )

which must return the product y = Ax for any given vector x.


More generally, MINRESQLP is designed to solve the system

           (A - shift*I) x = b
or
    min || (A - shift*I) x - b ||_2,

where shift is a specified real scalar.  Again, the matrix
(A - shift*I) may be indefinite and/or singular.
The work per iteration is very slightly less if shift = 0.

Note: If  shift  is an approximate eigenvalue of  A
and  b  is an approximate eigenvector,  x  might prove to be
a better approximate eigenvector, as in the methods of
inverse iteration and/or Rayleigh-quotient iteration.
However, we are not yet sure on that -- it may be better
to use SYMMLQ.

In this documentation, ' denotes the transpose of
a vector or a matrix.

A further option is that of preconditioning, which may reduce
the number of iterations required.  If M = C C' is a positive
definite matrix that is known to approximate  (A - shift*I)
in some sense, and if systems of the form  My = x  can be
solved efficiently, the parameter Msolve may be used (see below).
When an external procedure Msolve is supplied, MINRESQLP will
implicitly solve the system of equations

        P (A - shift*I) P' xbar  =  P b,

i.e.                 Abar xbar  =  bbar
where                       P   =  C**(-1),
                         Abar   =  P (A - shift*I) P',
                         bbar   =  P b,

and return the solution       x  =  P' xbar.
The associated residual is rbar  =  bbar - Abar xbar
                                 =  P (b - (A - shift*I)x)
                                 =  P r.

In the discussion below, eps refers to the machine precision.
eps is computed by MINRESQLP.  A typical value is eps = 2.22d-16
for IEEE double-precision arithmetic.
```

```
Parameters
----------
Some inputs are optional, with default values described below.
Mandatory inputs are n, Aprod, and b.
All outputs other than x are optional.

n       input      The dimension of the matrix or operator A.

b(n)    input      The rhs vector b.

x(n)    output     Returns the computed solution  x.

Aprod   external   A subroutine defining the matrix A.
                   For a given vector x, the statement

                         call Aprod ( n, x, y )

                   must return the product y = Ax
                   without altering the vector x.
                   An extra call of Aprod is
                   used to check if A is symmetric.

Msolve  external   An optional subroutine defining a
                   preconditioning matrix M, which should
                   approximate (A - shift*I) in some sense.
                   M must be positive definite.
                   For a given vector x, the statement

                         call Msolve( n, x, y )

                   must solve the linear system My = x
                   without altering the vector x.

                   In general, M should be chosen so that Abar has
                   clustered eigenvalues.  For example,
                   if A is positive definite, Abar would ideally
                   be close to a multiple of I.
                   If A or A - shift*I is indefinite, Abar might
                   be close to a multiple of diag( I  -I ).

shift   input      Should be zero if the system Ax = b is to be
                   solved.  Otherwise, it could be an
                   approximation to an eigenvalue of A, such as
                   the Rayleigh quotient b'Ab / (b'b)
                   corresponding to the vector b.
                   If b is sufficiently like an eigenvector
                   corresponding to an eigenvalue near shift,
                   then the computed x may have very large
                   components.  When normalized, x may be
                   closer to an eigenvector than b. Default to 0.

nout    input      A file number. The calling program must open a file
                   for output using for example:
                     open(nout, file='MINRESQLP.txt', status='unknown')
                   If nout > 0, a summary of the iterations
                   will be printed on unit nout. If nout is absent or
                   the file associated with nout is not opened properly,
                   results will be written to 'MINRESQLP_tmp.txt'.
                   (Avoid 0, 5, 6 because by convention stderr=0,
                   stdin=5, stdout=6.)

itnlim  input      An upper limit on the number of iterations. Default to 4n.

rtol    input      A user-specified tolerance.  MINRESQLP terminates
                   if it appears that norm(rbar) is smaller than
                         rtol*[norm(Abar)*norm(xbar) + norm(b)],
                   where rbar = bbar - Abar xbar,
                   or that norm(Abar*rbar) is smaller than
                         rtol*norm(Abar)*norm(rbar).

                   If shift = 0 and Msolve is absent, MINRESQLP
                   terminates if norm(r) is smaller than
                         rtol*[norm(A)*norm(x) + norm(b)],
```

```
                    where r = b - Ax,
                    or if norm(A*r) is smaller than
                          rtol*norm(A)*norm(r).

                    Default to machine precision.

 istop   output     An integer giving the reason for termination...
           0        Initial value of istop.

           1        beta_{k+1} < eps.
                    Iteration k is the final Lanczos step.

           2        beta2 = 0 in the Lanczos iteration; i.e. the
                    second Lanczos vector is zero.  This means the
                    rhs is very special.

                    If there is no preconditioner, b is an
                    eigenvector of Abar. Also, x = (1/alpha1) b
                    is a solution of Abar x = b.

                    Otherwise (if Msolve is present), let My = b.
                    If shift is zero, y is a solution of the
                    generalized eigenvalue problem Ay = lambda My,
                    with lambda = alpha1 from the Lanczos vectors.

                    In general, (A - shift*I)x = b
                    has the solution x = (1/alpha1) y
                    where My = b.

           3        b = 0, so the exact solution is x = 0.
                    No iterations were performed.

           4        Norm(rbar) appears to be less than
                    the value  rtol * [ norm(Abar) * norm(xbar) + norm(b) ].
                    The solution in  x  should be an acceptable
                    solution of Abar x = b.

           5        Norm(rbar) appears to be less than
                    the value  eps * norm(Abar) * norm(xbar).
                    This means that the solution is as accurate as
                    seems reasonable on this machine.

           6        Norm(Abar rbar) appears to be less than
                    the value  rtol * norm(Abar) * norm(rbar).
                    The solution in x should be an acceptable
                    least-squares solution.

           7        Norm(Abar rbar) appears to be less than
                    the value  eps * norm(Abar) * norm(rbar).
                    This means that the least-squares solution is as
                    accurate as seems reasonable on this machine.

           8        The iteration limit was reached before convergence.

           9        The matrix defined by Aprod does not appear
                    to be symmetric.
                    For certain vectors y = Av and r = Ay, the
                    products y'y and r'v differ significantly.

           10       The matrix defined by Msolve does not appear
                    to be symmetric.
                    For vectors satisfying My = v and Mr = y, the
                    products y'y and r'v differ significantly.

           11       An inner product of the form  x' M**(-1) x
                    was not positive, so the preconditioning matrix
                    M does not appear to be positive definite.

           12       xnorm has exceeded maxxnorm or will exceed it
                    next iteration.

           13       Acond (see below) has exceeded Acondlim or 0.1/eps,
                    so the matrix Abar must be very ill-conditioned.
```

```
                14       | gamma_k | < eps.
                         This is very likely a least-squares problem but
                         x may not contain an acceptable solution yet.

                15       norm(Abar x) < rtol * norm(Abar) * norm(x).
                         If disable = .true., then a null vector will be
                         obtained, given rtol.

                         If istop >= 7, the final x may not be an
                         acceptable solution.

     itn     output      The number of iterations performed.

     Anorm   output      An estimate of the norm of the matrix operator
                         Abar = P (A - shift*I) P',   where P = C**(-1).

     Acond   output      An estimate of the condition of Abar above.
                         This will usually be a substantial
                         under-estimate of the true condition.

     rnorm   output      An estimate of the norm of the final
                         transformed residual vector,
                            P (b  -  (A - shift*I) x).

     xnorm   output      An estimate of the norm of xbar.
                         This is sqrt( x'Mx ).  If Msolve is absent,
                         xnorm is an estimate of norm(x).

   maxxnorm  input       An upper bound on norm(x). Default value is 1e7.

   trancond  input       If trancond > 1, a switch is made from MINRES
                         iterations to MINRES-QLP iterations when
                         Acond > trancond.
                         If trancond = 1, all iterations are MINRES-QLP
                         iterations.
                         If trancond = Acondlim, all iterations are
                         conventional MINRES iterations (which are
                         slightly cheaper).
                         Default to 1e7.

   Acondlim  input       An upper bound on Acond. Default value is 1e15.

   disable   input       All stopping conditions are disabled except
                         norm(Ax) / norm(x) < tol. Default to .false..

-----------------------------------------------------------------------

     MINRESQLP is an implementation of the algorithm described in
     the following references:

     Sou-Cheng Choi,
     Iterative Methods for Singular Linear Equations and Least-
     Squares Problems, PhD dissertation, ICME, Stanford University,
     2006.

     Sou-Cheng Choi, Christopher Paige, and Michael Saunders,
     MINRES-QLP: A Krylov subspace method for indefinite or
     singular symmetric systems, SIAM Journal of Scientific
     Computing 33:4 (2011) 1810-1836.

     Sou-Cheng Choi and Michael Saunders,
     ALGORITHM & DOCUMENTATION: MINRES-QLP for singular Symmetric and Hermitian
     linear equations and least-squares problems, Technical Report,
     ANL/MCS-P3027-0812, Computation Institute,
     University of Chicago/Argonne National Laboratory, 2012.

     Sou-Cheng Choi and Michael Saunders,
     ALGORITHM xxx: MINRES-QLP for singular Symmetric and Hermitian
     linear equations and least-squares problems,
     ACM Transactions on Mathematical Software, to appear, 2013.

     FORTRAN 90 and MATLAB implementations are
```

```
     downloadable from
            http://www.stanford.edu/group/SOL/software.html
            http://home.uchicago.edu/sctchoi/

-----------------------------------------------------------------

     MINRESQLP development:
     14 Dec 2006: Sou-Cheng's thesis completed.
                  MINRESQLP includes a stopping rule for singular
                  systems (using an estimate of ||Ar||) and very many
                  other things(!).
                  Note that ||Ar|| small => r is a null vector for A.
     09 Oct 2007: F90 version constructed from the F77 version.
                  Initially used compiler option -r8, but this is
                  nonstandard.
     15 Oct 2007: Test on Arnorm = ||Ar|| added to recognize
                  singular systems.
     15 Oct 2007: Temporarily used real(8) everywhere.
     16 Oct 2007: Use minresqlpDataModule to define
                  dp = selected_real_kind(15).
                  We need "use minresqlpDataModule" at the
                  beginning of modules AND inside interfaces.
     06 Jun 2010: Added comments.
     12 Jul 2011: Created complex version zminresqlpModule.f90
                  from real version minresqlpModule.f90.
     23 Aug 2011: (1) Tim Hopkins ran version 17 on the NAG Fortran compiler
                  We removed half a dozen unused variables in MINRESQLP
                  and also local var sgn_a and sgn_b in SMMORTHO,
                  as they result in division by zero for inputs a=b=0.
                  (2) Version 18 was submitted to ACM TOMS for review.
     20 Aug 2012: Version 19:
                  (1) Added optional inputs and outputs, and
                  default values for optional inputs.
                  (2) Removed inputs 'checkA' and 'precon'.
                  (3) Changed slightly the order of parameters in the
                  MINRESQLP API.
                  (4) Updated documentation.
                  (5) Fixed a minor bug in printing x(1) in iteration
                      log during MINRES mode.
                  (6) Made sure MINRESQLP is portable in both single
                      and double precison.
                  (7) Fixed a bug to ensure the 2x2 Hermitian reflectors
                      are orthonormal. Make output c real.
     24 Apr 2013: istop = 12 now means xnorm just exceeded maxxnorm.
     28 Jun 2013: likeLS introduced to terminate with big xnorm
                  only if the problem seems to be singular and inconsistent.
     08 Jul 2013: (1) dot_product replaces ddotc.
     04 Aug 2013: If present(maxxnorm), use maxxnorm_ = min(maxxnorm, one/eps).
     09 Sep 2013: Initialize relresl and relAresl to zero.
-----------------------------------------------------------------
```

Definition at line 411 of file minresqlpModule.f90.

References minresqlpblasmodule::dnrm2(), and symortho().

Referenced by mminrsqlp(), and solgloqlp().

8.8.2.2 subroutine, public minresqlpmodule::symortho (real(dp), intent(in) *a,* real(dp), intent(in) *b,* real(dp), intent(out) *c,* real(dp), intent(out) *s,* real(dp), intent(out) *r*)

SymOrtho: Stable Householder reflection.

```
  USAGE:
     SymOrtho(a, b, c, s, r)

  INPUTS:
    a      first element of a two-vector  [a; b]
    b      second element of a two-vector [a; b]

  OUTPUTS:
```

```
   c      cosine(theta), where theta is the implicit angle of reflection
   s      sine(theta)
   r      two-norm of [a; b]

 DESCRIPTION:
    Stable Householder reflection that gives c and s such that
       [ c  s ][a] = [r],
       [ s -c ][b]   [0]
    where r = two norm of vector [a, b],
       c = a / sqrt(a**2 + b**2) = a / r,
       s = b / sqrt(a**2 + b**2) = b / r.
    The implementation guards against overlow in computing sqrt (a**2 + b**2).


 REFERENCES:
   Algorithm 4.9, stable unsymmetric Givens rotations in
    Golub and van Loan's book Matrix Computations, 3rd edition.

 MODIFICATION HISTORY:
   20/08/2012: Fixed a bug to ensure the 2x2 Hermitian reflectors
               are orthonormal.
   05/27/2011: Created this file from Matlab SymGivens2.m

 KNOWN BUGS:
    MM/DD/2004: description

 AUTHORS: Sou-Cheng Choi, CI, University of Chicago
          Michael Saunders, MS&E, Stanford University

 CREATION DATE: 05/27/2011
```

Definition at line 1334 of file minresqlpModule.f90.

Referenced by minresqlp().

The documentation for this module was generated from the following file:

- minresqlpModule.f90

8.9 mpdalc::mpalloc Interface Reference

allocate array

Public Member Functions

- subroutine mpallocdvec (array, length, text)

  allocate (1D) double precision array
- subroutine mpallocfvec (array, length, text)

  allocate (1D) single precision array
- subroutine mpallocivec (array, length, text)

  allocate (1D) integer array
- subroutine mpallocfarr (array, rows, cols, text)

  allocate (2D) single precision array
- subroutine mpallociarr (array, rows, cols, text)

  allocate (2D) INTEGER(mpi) array
- subroutine mpalloclarr (array, rows, cols, text)

  allocate (2D) large integer array
- subroutine mpalloclist (array, length, text)

  allocate (1D) list item array
- subroutine mpalloccvec (array, length, text)

  allocate (1D) character array

8.9.1 Detailed Description

allocate array

Definition at line 19 of file mpdalc.f90.

8.9.2 Member Function/Subroutine Documentation

8.9.2.1 subroutine mpdalc::mpalloc::mpalloccvec (character, dimension(:), intent(inout), allocatable *array,* integer(mpl), intent(in) *length,* character (len=∗), intent(in) *text*)

allocate (1D) character array

Definition at line 112 of file mpdalc.f90.

8.9.2.2 subroutine mpdalc::mpalloc::mpallocdvec (real(mpd), dimension(:), intent(inout), allocatable *array,* integer(mpl), intent(in) *length,* character (len=∗), intent(in) *text*)

allocate (1D) double precision array

Definition at line 32 of file mpdalc.f90.

8.9.2.3 subroutine mpdalc::mpalloc::mpallocfarr (real(mps), dimension(:,:), intent(inout), allocatable *array,* integer(mpl), intent(in) *rows,* integer(mpl), intent(in) *cols,* character (len=∗), intent(in) *text*)

allocate (2D) single precision array

Definition at line 65 of file mpdalc.f90.

8.9.2.4 subroutine mpdalc::mpalloc::mpallocfvec (real(mps), dimension(:), intent(inout), allocatable *array,* integer(mpl), intent(in) *length,* character (len=∗), intent(in) *text*)

allocate (1D) single precision array

Definition at line 43 of file mpdalc.f90.

8.9.2.5 subroutine mpdalc::mpalloc::mpallociarr (integer(mpi), dimension(:,:), intent(inout), allocatable *array,* integer(mpl), intent(in) *rows,* integer(mpl), intent(in) *cols,* character (len=∗), intent(in) *text*)

allocate (2D) INTEGER(mpi) array

Definition at line 77 of file mpdalc.f90.

8.9.2.6 subroutine mpdalc::mpalloc::mpallocivec (integer(mpi), dimension(:), intent(inout), allocatable *array,* integer(mpl), intent(in) *length,* character (len=∗), intent(in) *text*)

allocate (1D) integer array

Definition at line 54 of file mpdalc.f90.

8.9.2.7 subroutine mpdalc::mpalloc::mpalloclarr (integer(mpl), dimension(:,:), intent(inout), allocatable *array,* integer(mpl), intent(in) *rows,* integer(mpl), intent(in) *cols,* character (len=∗), intent(in) *text*)

allocate (2D) large integer array

Definition at line 89 of file mpdalc.f90.

8.9.2.8 subroutine mpdalc::mpalloc::mpalloclist (type(listitem), dimension(:), intent(inout), allocatable *array,* integer(mpl), intent(in) *length,* character (len=*), intent(in) *text*)

allocate (1D) list item array

Definition at line 101 of file mpdalc.f90.

The documentation for this interface was generated from the following file:

- mpdalc.f90

8.10 mpbits Module Reference

Bit field data.

Public Attributes

- integer(mpl) ndimb

  dimension for bit (field) array
- integer(mpi) n

  matrix size
- integer(mpi) ibfw

  bit field width
- integer(mpi) mxcnt

  max value for bit field counters
- integer(mpi) nencdm

  max value for column counter
- integer(mpi) nencdb

  number of bits for encoding column counter
- integer(mpi) nthrd

  number of threads
- integer(mpi), dimension(:), allocatable bitfieldcounters

  fit field counters for global parameters pairs
- integer(mpi), parameter bs = BIT_SIZE(1_mpi)

  number of bits in INTEGER(mpi)

8.10.1 Detailed Description

Bit field data.

Definition at line 14 of file mpbits.f90.

8.10.2 Member Data Documentation

8.10.2.1 integer(mpi), dimension(:), allocatable mpbits::bitfieldcounters

fit field counters for global parameters pairs

Definition at line 25 of file mpbits.f90.

8.10.2.2 integer(mpi), parameter mpbits::bs = BIT_SIZE(1_mpi)

number of bits in INTEGER(mpi)

Definition at line 26 of file mpbits.f90.

8.10.2.3 integer(mpi) mpbits::ibfw

bit field width

Definition at line 20 of file mpbits.f90.

8.10.2.4 integer(mpi) mpbits::mxcnt

max value for bit field counters

Definition at line 21 of file mpbits.f90.

8.10.2.5 integer(mpi) mpbits::n

matrix size

Definition at line 19 of file mpbits.f90.

8.10.2.6 integer(mpl) mpbits::ndimb

dimension for bit (field) array

Definition at line 18 of file mpbits.f90.

8.10.2.7 integer(mpi) mpbits::nencdb

number of bits for encoding column counter

Definition at line 23 of file mpbits.f90.

8.10.2.8 integer(mpi) mpbits::nencdm

max value for column counter

Definition at line 22 of file mpbits.f90.

8.10.2.9 integer(mpi) mpbits::nthrd

number of threads

Definition at line 24 of file mpbits.f90.

The documentation for this module was generated from the following file:

- mpbits.f90

8.11 mpdalc Module Reference

(De)Allocate vectors and arrays.

Data Types

- interface mpalloc

  allocate array
- interface mpdealloc

  deallocate array

Public Member Functions

- subroutine mpallocdvec (array, length, text)

  allocate (1D) double precision array
- subroutine mpallocfvec (array, length, text)

  allocate (1D) single precision array
- subroutine mpallocivec (array, length, text)

  allocate (1D) integer array
- subroutine mpallocfarr (array, rows, cols, text)

  allocate (2D) single precision array
- subroutine mpallociarr (array, rows, cols, text)

  allocate (2D) INTEGER(mpi) array
- subroutine mpalloclarr (array, rows, cols, text)

  allocate (2D) large integer array
- subroutine mpalloclist (array, length, text)

  allocate (1D) list item array
- subroutine mpalloccvec (array, length, text)

  allocate (1D) character array
- subroutine mpalloccheck (ifail, numwords, text)

  check allocation
- subroutine mpdeallocdvec (array)

  deallocate (1D) double precision array
- subroutine mpdeallocfvec (array)

  deallocate (1D) single precision array
- subroutine mpdeallocivec (array)

  deallocate (1D) integer array
- subroutine mpdeallocfarr (array)

  allocate (2D) single precision array
- subroutine mpdeallociarr (array)

  allocate (2D) integer array
- subroutine mpdealloclarr (array)

  deallocate (2D) large integer array
- subroutine mpdealloclist (array)

  deallocate (1D) list item array
- subroutine mpdealloccvec (array)

  deallocate (1D) character array
- subroutine mpdeallocchеck (ifail, numwords)

  check deallocation

Public Attributes

- integer(mpl) numwordsalloc = 0
  current dynamic memory allocation (words)
- integer(mpl) maxwordsalloc = 0
  peak dynamic memory allocation (words)
- integer(mpi) nummpalloc = 0
  number of dynamic allocations
- integer(mpi) nummpdealloc = 0
  number of dynamic deallocations
- integer(mpi) printflagalloc = 0
  print flag for dynamic allocations

8.11.1 Detailed Description

(De)Allocate vectors and arrays.

Definition at line 7 of file mpdalc.f90.

8.11.2 Member Function/Subroutine Documentation

8.11.2.1 subroutine mpdalc::mpalloccheck (integer(mpi), intent(in) *ifail,* integer(mpl), intent(in) *numwords,* character (len=∗), intent(in) *text*)

check allocation

Definition at line 123 of file mpdalc.f90.

References peend().

Referenced by mpalloccvec(), mpallocdvec(), mpallocfarr(), mpallocfvec(), mpallociarr(), mpallocivec(), mpalloclarr(), and mpalloclist().

8.11.2.2 subroutine mpdalc::mpalloccvec (character, dimension(:), intent(inout), allocatable *array,* integer(mpl), intent(in) *length,* character (len=∗), intent(in) *text*)

allocate (1D) character array

Definition at line 112 of file mpdalc.f90.

References mpalloccheck().

8.11.2.3 subroutine mpdalc::mpallocdvec (real(mpd), dimension(:), intent(inout), allocatable *array,* integer(mpl), intent(in) *length,* character (len=∗), intent(in) *text*)

allocate (1D) double precision array

Definition at line 32 of file mpdalc.f90.

References mpalloccheck().

8.11.2.4 subroutine mpdalc::mpallocfarr (real(mps), dimension(:,:), intent(inout), allocatable *array,* integer(mpl), intent(in) *rows,* integer(mpl), intent(in) *cols,* character (len=∗), intent(in) *text*)

allocate (2D) single precision array

Definition at line 65 of file mpdalc.f90.

References mpalloccheck().

8.11.2.5 subroutine mpdalc::mpallocfvec (real(mps), dimension(:), intent(inout), allocatable *array,* integer(mpl), intent(in) *length,* character (len=∗), intent(in) *text*)

allocate (1D) single precision array

Definition at line 43 of file mpdalc.f90.

References mpalloccheck().

8.11.2.6 subroutine mpdalc::mpallociarr (integer(mpi), dimension(:,:), intent(inout), allocatable *array,* integer(mpl), intent(in) *rows,* integer(mpl), intent(in) *cols,* character (len=∗), intent(in) *text*)

allocate (2D) INTEGER(mpi) array

Definition at line 77 of file mpdalc.f90.

References mpalloccheck().

8.11.2.7 subroutine mpdalc::mpallocivec (integer(mpi), dimension(:), intent(inout), allocatable *array,* integer(mpl), intent(in) *length,* character (len=∗), intent(in) *text*)

allocate (1D) integer array

Definition at line 54 of file mpdalc.f90.

References mpalloccheck().

8.11.2.8 subroutine mpdalc::mpalloclarr (integer(mpl), dimension(:,:), intent(inout), allocatable *array,* integer(mpl), intent(in) *rows,* integer(mpl), intent(in) *cols,* character (len=∗), intent(in) *text*)

allocate (2D) large integer array

Definition at line 89 of file mpdalc.f90.

References mpalloccheck().

8.11.2.9 subroutine mpdalc::mpalloclist (type(listitem), dimension(:), intent(inout), allocatable *array,* integer(mpl), intent(in) *length,* character (len=∗), intent(in) *text*)

allocate (1D) list item array

Definition at line 101 of file mpdalc.f90.

References mpalloccheck().

8.11.2.10 subroutine mpdalc::mpdeallocchеck (integer(mpi), intent(in) *ifail,* integer(mpl), intent(in) *numwords*)

check deallocation

Definition at line 233 of file mpdalc.f90.

References peend().

Referenced by mpdealloccvec(), mpdeallocdvec(), mpdeallocfarr(), mpdeallocfvec(), mpdeallociarr(), mpdeallocivec(), mpdealloclarr(), and mpdealloclist().

8.11.2.11 subroutine mpdalc::mpdealloccvec (character, dimension(:), intent(inout), allocatable *array*)

deallocate (1D) character array

Definition at line 222 of file mpdalc.f90.

References mpdeallocchеck().

8.11.2.12 subroutine mpdalc::mpdeallocdvec (real(mpd), dimension(:), intent(inout), allocatable *array*)

deallocate (1D) double precision array

Definition at line 145 of file mpdalc.f90.

References mpdeallocchеck().

8.11.2.13 subroutine mpdalc::mpdeallocfarr (real(mps), dimension(:,:), intent(inout), allocatable *array*)

allocate (2D) single precision array

Definition at line 178 of file mpdalc.f90.

References mpdeallocchеck().

8.11.2.14 subroutine mpdalc::mpdeallocfvec (real(mps), dimension(:), intent(inout), allocatable *array*)

deallocate (1D) single precision array

Definition at line 156 of file mpdalc.f90.

References mpdeallocchеck().

8.11.2.15 subroutine mpdalc::mpdeallociarr (integer(mpi), dimension(:,:), intent(inout), allocatable *array*)

allocate (2D) integer array

Definition at line 189 of file mpdalc.f90.

References mpdeallocchеck().

8.11.2.16 subroutine mpdalc::mpdeallocivec (integer(mpi), dimension(:), intent(inout), allocatable *array*)

deallocate (1D) integer array

Definition at line 167 of file mpdalc.f90.

References mpdeallocchеck().

8.11.2.17 subroutine mpdalc::mpdealloclarr (integer(mpl), dimension(:,:), intent(inout), allocatable *array*)

deallocate (2D) large integer array

Definition at line 200 of file mpdalc.f90.

References mpdeallocchеck().

8.11.2.18 subroutine mpdalc::mpdealloclist (type(listitem), dimension(:), intent(inout), allocatable *array*)

deallocate (1D) list item array

Definition at line 211 of file mpdalc.f90.

References mpdeallocchеck().

8.11.3 Member Data Documentation

8.11.3.1 integer(mpl) mpdalc::maxwordsalloc = 0

peak dynamic memory allocation (words)

Definition at line 13 of file mpdalc.f90.

8.11.3.2 integer(mpi) mpdalc::nummpalloc = 0

number of dynamic allocations

Definition at line 14 of file mpdalc.f90.

8.11.3.3 integer(mpi) mpdalc::nummpdealloc = 0

number of dynamic deallocations

Definition at line 15 of file mpdalc.f90.

8.11.3.4 integer(mpl) mpdalc::numwordsalloc = 0

current dynamic memory allocation (words)

Definition at line 12 of file mpdalc.f90.

8.11.3.5 integer(mpi) mpdalc::printflagalloc = 0

print flag for dynamic allocations

Definition at line 16 of file mpdalc.f90.

The documentation for this module was generated from the following file:

- mpdalc.f90

8.12 mpdalc::mpdealloc Interface Reference

deallocate array

Public Member Functions

- subroutine mpdeallocdvec (array)
  deallocate (1D) double precision array
- subroutine mpdeallocfvec (array)
  deallocate (1D) single precision array
- subroutine mpdeallocivec (array)
  deallocate (1D) integer array
- subroutine mpdeallocfarr (array)
  allocate (2D) single precision array
- subroutine mpdeallociarr (array)
  allocate (2D) integer array
- subroutine mpdealloclarr (array)
  deallocate (2D) large integer array

- subroutine mpdealloclist (array)

  deallocate (1D) list item array
- subroutine mpdealloccvec (array)

  deallocate (1D) character array

8.12.1 Detailed Description

deallocate array

Definition at line 24 of file mpdalc.f90.

8.12.2 Member Function/Subroutine Documentation

8.12.2.1 subroutine mpdalc::mpdealloc::mpdealloccvec (character, dimension(:), intent(inout), allocatable *array*)

deallocate (1D) character array

Definition at line 222 of file mpdalc.f90.

8.12.2.2 subroutine mpdalc::mpdealloc::mpdeallocdvec (real(mpd), dimension(:), intent(inout), allocatable *array*)

deallocate (1D) double precision array

Definition at line 145 of file mpdalc.f90.

8.12.2.3 subroutine mpdalc::mpdealloc::mpdeallocfarr (real(mps), dimension(:,:), intent(inout), allocatable *array*)

allocate (2D) single precision array

Definition at line 178 of file mpdalc.f90.

8.12.2.4 subroutine mpdalc::mpdealloc::mpdeallocfvec (real(mps), dimension(:), intent(inout), allocatable *array*)

deallocate (1D) single precision array

Definition at line 156 of file mpdalc.f90.

8.12.2.5 subroutine mpdalc::mpdealloc::mpdeallociarr (integer(mpi), dimension(:,:), intent(inout), allocatable *array*)

allocate (2D) integer array

Definition at line 189 of file mpdalc.f90.

8.12.2.6 subroutine mpdalc::mpdealloc::mpdeallocivec (integer(mpi), dimension(:), intent(inout), allocatable *array*)

deallocate (1D) integer array

Definition at line 167 of file mpdalc.f90.

8.12.2.7 subroutine mpdalc::mpdealloc::mpdealloclarr (integer(mpl), dimension(:,:), intent(inout), allocatable *array*)

deallocate (2D) large integer array

Definition at line 200 of file mpdalc.f90.

8.12.2.8 subroutine mpdalc::mpdealloc::mpdealloclist (type(listitem), dimension(:), intent(inout), allocatable *array*)

deallocate (1D) list item array

Definition at line 211 of file mpdalc.f90.

The documentation for this interface was generated from the following file:

- mpdalc.f90

8.13 mpdef Module Reference

Definition of constants.

Data Types

- type listitem

  list items from steering file

Public Attributes

- integer, parameter mpi = selected_int_kind(9)
- integer, parameter byte
- integer, parameter integer
- integer, parameter mpl = selected_int_kind(18)
- integer, parameter mps = selected_real_kind(6, 37)
- integer, parameter float
- integer, parameter mpd = selected_real_kind(15, 307)
- integer, parameter mpq = selected_real_kind(33, 4931)
- integer, parameter gcc
- integer, parameter needs
- integer, parameter libquadmath

8.13.1 Detailed Description

Definition of constants.

Definition at line 5 of file mpdef.f90.

8.13.2 Member Data Documentation

8.13.2.1 integer parameter mpdef::byte

Definition at line 11 of file mpdef.f90.

8.13.2.2 integer parameter mpdef::float

Definition at line 13 of file mpdef.f90.

8.13.2.3 integer, parameter mpdef::gcc

Definition at line 15 of file mpdef.f90.

8.13.2.4 integer parameter mpdef::integer

Definition at line 11 of file mpdef.f90.

8.13.2.5 integer, parameter mpdef::libquadmath

Definition at line 15 of file mpdef.f90.

8.13.2.6 integer, parameter mpdef::mpd = selected_real_kind(15, 307)

Definition at line 14 of file mpdef.f90.

8.13.2.7 integer, parameter mpdef::mpi = selected_int_kind(9)

Definition at line 11 of file mpdef.f90.

8.13.2.8 integer, parameter mpdef::mpl = selected_int_kind(18)

Definition at line 12 of file mpdef.f90.

8.13.2.9 integer, parameter mpdef::mpq = selected_real_kind(33, 4931)

Definition at line 15 of file mpdef.f90.

8.13.2.10 integer, parameter mpdef::mps = selected_real_kind(6, 37)

Definition at line 13 of file mpdef.f90.

8.13.2.11 integer, parameter mpdef::needs

Definition at line 15 of file mpdef.f90.

The documentation for this module was generated from the following file:

- mpdef.f90

8.14 mpmod Module Reference

Parameters, variables, dynamic arrays.

Public Attributes

- integer(mpi) ictest =0

  test mode '-t'
- integer(mpi) metsol =0

  solution method (1: inversion, 2: diagonalization, 3: MINRES-QLP)
- integer(mpi) matsto =2

  (global) matrix storage mode (1: full, 2: sparse)
- integer(mpi) mprint =1

print flag (0: minimal, 1: normal, >1: more)

- integer(mpi) mdebug =0

  debug flag (number of records to print)
- integer(mpi) mdebg2 =10

  number of measurements for record debug printout
- integer(mpi) mreqen =10

  required number of entries (for variable global parameter)
- integer(mpi) mitera =1

  number of iterations
- integer(mpi) nloopn =0

  number of data reading, fitting loops
- integer(mpi) mbandw =0

  band width of preconditioner matrix
- integer(mpi) lunkno =0

  flag for unkown keywords
- integer(mpi) lhuber =0

  Huber down-weighting flag.
- real(mps) chicut =0.0

  cut in terms of 3-sigma cut, first iteration
- real(mps) chirem =0.0

  cut in terms of 3-sigma cut, other iterations, approaching 1.
- real(mps) chhuge =50.0

  cut in terms of 3-sigma for unreasonable data, all iterations
- integer(mpi) nrecpr =0

  record number with printout
- integer(mpi) nrecp2 =0

  record number with printout
- integer(mpi) nrec1 =0

  record number with largest residual
- integer(mpi) nrec2 =0

  record number with largest chi^2/Ndf
- real(mps) value1 =0.0

  largest residual
- real(mps) value2 =0.0

  largest chi^2/Ndf
- real(mps) dwcut =0.0

  down-weight fraction cut
- integer(mpi) isubit =0

  subito flag '-s'
- real(mps) wolfc1 =0.0

  C_1 of strong Wolfe condition.
- real(mps) wolfc2 =0.0

  C_2 of strong Wolfe condition.
- real(mpd) mrestl =1.0E-06

  tolerance criterion for MINRES-QLP
- real(mpd) mrtcnd =1.0E+07

  transition (QR -> QLP) (matrix) condition for MINRES-QLP
- integer(mpi) mrmode =0

  MINRES-QLP mode (0: QR+QLP, 1: only QR, 2: only QLP factorization)
- integer(mpi) nofeas =0

  flag for skipping making parameters feasible

- integer(mpi) nhistp =0

  flag for histogram printout
- real(mps) delfun =0.0

  expected function change
- real(mps) actfun =0.0

  actual function change
- real(mps) angras =0.0

  angle between gradient and search direction
- integer(mpi) iterat =0

  iterations in solution
- integer(mpi) nregul =0

  regularization flag
- real(mps) regula =1.0

  regularization parameter, add regula $*$ norm(global par.) to objective function
- real(mps) regpre =0.0

  default presigma
- integer(mpi) matrit =0

  matrix calculation up to iteration MATRIT
- integer(mpi) icalcm =0

  calculation mode (for XLOOPN) , $>$0: calculate matrix
- integer(mpi) numbit =1

  number of bits for pair counters
- integer(mpi) nbndr =0

  number of records with bordered band matrix for local fit
- integer(mpi) nbdrx =0

  max border size for local fit
- integer(mpi) nbndx =0

  max band width for local fit
- integer(mpi) nrecer =0

  record with error (rank deficit or Not-a-Number) for printout
- integer(mpi) nrec3 = huge(nrec3)

  (1.) record number with error
- integer(mpi) mreqpe =1

  min number of pair entries
- integer(mpi) mhispe =0

  upper bound for pair entry histogrammimg
- integer(mpi) msngpe =0

  upper bound for pair entry single precision storage
- integer(mpi) mcmprs =0

  compression flag for sparsity (column indices)
- integer(mpi) mthrd =1

  number of (OpenMP) threads
- integer(mpi) mxrec =0

  max number of records
- integer(mpi) matmon =0

  record interval for monitoring of (sparse) matrix construction
- integer(mpi) lfitnp =huge(lfitnp)

  local fit: number of iteration to calculate pulls
- integer(mpi) lfitbb =1

  local fit: check for bordered band matrix (if $>$0)
- integer(mpi) mnrsel =0

number of MINRES error labels in LBMNRS (calc err, corr with SOLGLO)

- integer(mpi) ncache =-1

  buffer size for caching (default 100MB per thread)
- real(mps), dimension(3) fcache = (/ 0.8, 0., 0. /)

  read cache, average fill level; write cache; dynamic size
- integer(mpi) mthrdr =1

  number of threads for reading binary files
- integer(mpi) mnrsit =0

  total number of MINRES internal iterations
- integer(mpi) iforce =0

  switch to SUBITO for (global) rank defects if zero
- integer(mpi) igcorr =0

  flag for output of global correlations for inversion, =0: none
- integer(mpi) memdbg =0

  debug flag for memory management
- real(mps) prange =0.0

  range (-PRANGE..PRANGE) for histograms of pulls, norm. residuals
- integer(mpi) lsearch =2

  iterations (solutions) with line search: >2: all, =2: all with (next) Chi2 cut scaling factor =1., =1: last, <1: none
- integer(mpi) lunlog

  unit for logfile
- integer(mpi) lvllog

  log level
- integer(mpi) ntgb

  total number of global parameters
- integer(mpi) nvgb

  number of variable global parameters
- integer(mpi) nagb

  number of fit parameters (global par. + Lagrange mult.)
- integer(mpi) ncgb

  number of constraints
- integer(mpi) nagbn

  max number of global paramters per record
- integer(mpi) nalcn

  max number of local paramters per record
- integer(mpi) naeqn

  max number of equations (measurements) per record
- integer(mpi) nrec

  (current) record number
- real(mps) dflim

  convergence limit
- integer(mpi), dimension(0:3) nrejec

  rejected events
- real(mps), dimension(0:8) times

  cpu time counters
- real(mps) stepl

  step length (line search)
- character(len=74) textl

  name of current MP 'module' (step)
- logical newite

  flag for new iteration

- integer(mpi) ndfsum
  sum(ndf)
- integer(mpi) iitera
  MINRES iterations.
- integer(mpi) istopa
  MINRES istop (convergence)
- integer(mpi) lsinfo
  line search: returned information
- real rstart
  cpu start time for solution iterations
- real(mps) deltim
  cpu time difference
- integer(mpi) npresg
  number of pre-sigmas
- integer(mpi) nrecal
  number of records
- integer(mpi) ndefec =0
  rank deficit for global matrix (from inversion)
- integer(mpi) nmiss1 =0
  rank deficit for constraints
- integer(mpi) lcalcm
  last calclation mode
- integer(mpi) nspc
  number of precision for sparse global matrix (1=D, 2=D+F)
- integer(mpi) nencdb
  encoding info (number bits for column counter)
- integer(mpi), dimension(100) lbmnrs
  MINRES error labels.
- real(mpd) fvalue
  function value (chi2 sum) solution
- real(mpd) flines
  function value line search
- real(mpd) sumndf
  weighted sum(ndf)
- integer(mpi) numreadbuffer
  number of buffers (records) in (read) block
- integer(mpi) numblocks
  number of (read) blocks
- integer(mpi) sumrecords
  sum of records
- integer(mpi) skippedrecords
  number of skipped records (buffer too small)
- integer(mpi) minrecordsinblock
  min. records in block
- integer(mpi) maxrecordsinblock
  max. records in block
- integer(mpi), parameter nexp20 =1048576
- real(mpd) accuratedsum =0.0_mpd
  fractional part of sum
- integer(mpi) accuratensum =0
  *sum mod 2**20*

- integer(mpi) accuratenexp =0

  *sum / 2**20*
- integer(mpi) lenglobalvec

  *length of global vector 'b' (A*x=b)*
- real(mpd), dimension(:), allocatable globalparameter

  global parameters (start values + sum(x_i))
- real(mpd), dimension(:), allocatable globalparcopy

  copy of global parameters
- real(mpd), dimension(:), allocatable globalcorrections

  *correction x_i (from A*x_i=b_i in iteration i)*
- real(mps), dimension(:), allocatable globalparstart

  start value for global parameters
- real(mps), dimension(:), allocatable globalparpresigma

  pre-sigma for global parameters
- real(mps), dimension(:), allocatable globalparpreweight

  weight from pre-sigma
- real(mpd), dimension(:), allocatable globalmatd

  global matrix 'A' (double, full or sparse)
- real(mps), dimension(:), allocatable globalmatf

  global matrix 'A' (float part for compressed sparse)
- real(mpd), dimension(:), allocatable globalvector

  *global vector 'x' (in A*x=b)*
- real(mpd), dimension(:), allocatable matprecond

  preconditioner (band) matrix
- integer(mpi), dimension(:), allocatable indprecond

  preconditioner pointer array
- real(mpd), dimension(:), allocatable workspaced

  (general) workspace (D)
- real(mpd), dimension(:), allocatable workspacelinesearch

  workspace line search
- real(mpd), dimension(:), allocatable workspacediagonalization

  workspace diag.
- real(mpd), dimension(:), allocatable workspaceeigenvalues

  workspace eigen values
- real(mpd), dimension(:), allocatable workspaceeigenvectors

  workspace eigen vectors

- integer(mpi), dimension(:), allocatable workspacei
  (general) workspace (I)
- real(mpd), dimension(:), allocatable matconsproduct
  product matrix of constraints
- real(mpd), dimension(:), allocatable vecconsresiduals
  residuals of constraints
- integer(mpi), dimension(:,:), allocatable globalparlabelindex
  global parameters label, total -> var. index
- integer(mpi), dimension(:), allocatable globalparhashtable
  global parameters hash table
- integer(mpi), dimension(:), allocatable globalparvartototal
  global parameters variable -> total index
- integer(mpi), dimension(-7:0) globalparheader = 0
  global parameters (mapping) header
- integer(mpi), dimension(:), allocatable sparsematrixcompression
  compression info (per row)
- integer(mpi), dimension(:), allocatable sparsematrixcolumns
  (compressed) list of columns for sparse matrix
- integer(mpl), dimension(:,:), allocatable sparsematrixoffsets
  row offsets for column list, sparse matrix elements
- integer(mpi), dimension(:,:), allocatable readbufferinfo
  buffer management (per thread)
- integer(mpi), dimension(:), allocatable readbufferpointer
  pointer to used buffers
- integer(mpi), dimension(:), allocatable readbufferdatai
  integer data
- real(mps), dimension(:), allocatable readbufferdataf
  float data
- integer(mpi), dimension(:), allocatable globalindexusage
  indices of global par in record
- integer(mpi), dimension(:), allocatable backindexusage
  list of global par in record
- real(mpd), dimension(:), allocatable blvec
  local fit vector 'b' (in A∗x=b), replaced by 'x'
- real(mpd), dimension(:), allocatable clmat
  local fit matrix 'A' (in A∗x=b)

- integer(mpi), dimension(:), allocatable ibandh

  local fit 'band width histogram' (band size autodetection)
- real(mpd), dimension(:), allocatable vbnd

  local fit band part of 'A'
- real(mpd), dimension(:), allocatable vbdr

  local fit border part of 'A'
- real(mpd), dimension(:), allocatable aux

  local fit 'solutions for border rows'
- real(mpd), dimension(:), allocatable vbk

  local fit 'matrix for border solution'
- real(mpd), dimension(:), allocatable vzru

  local fit 'border solution'
- real(mpd), dimension(:), allocatable scdiag

  local fit workspace (D)
- integer(mpi), dimension(:), allocatable scflag

  local fit workspace (I)
- real(mps), dimension(:), allocatable localcorrections

  local fit corrections (to residuals)
- real(mpd), dimension(:), allocatable localglobalmatrix

  matrix correlating local and global par
- integer(mpi), dimension(:,:), allocatable writebufferinfo

  write buffer management (per thread)
- real(mps), dimension(:,:), allocatable writebufferdata

  write buffer data (largest residual, Chi2/ndf, per thread)
- integer(mpi), dimension(:), allocatable writebufferindices

  write buffer for indices
- real(mpd), dimension(:), allocatable writebufferupdates

  write buffer for update matrices
- integer(mpi), dimension(-6:6) writebufferheader = 0

  write buffer header (-6..-1: updates, 1..6: indices)
- integer(mpi) lenparameters =0

  list items from steering file
- type(listitem), dimension(:), allocatable listparameters

  list of parameters from steering file
- integer(mpi) lenpresigmas =0

  length of list of pre-sigmas from steering file
- type(listitem), dimension(:), allocatable listpresigmas

list of pre-sgmas from steering file

- integer(mpi) lenconstraints =0

  length of list of constraints from steering file

- type(listitem), dimension(:), allocatable listconstraints

  list of constraints from steering file

- integer(mpi) lenmeasurements =0

  length of list of measurements from steering file

- type(listitem), dimension(:), allocatable listmeasurements

  list of measurements from steering file

- integer(mpi), dimension(:), allocatable mfd

  file mode: cbinary =1, text =2, fbinary=3

- integer(mpi), dimension(:), allocatable lfd

  length of file name

- integer(mpi), dimension(:), allocatable nfd

  index (line) in (steering) file

- integer(mpi), dimension(:,:), allocatable kfd

  (1,.)= number of records in file, (2,..)= file order

- integer(mpi), dimension(:), allocatable ifd

  file: integrated record numbers (=offset)

- integer(mpi), dimension(:), allocatable jfd

  file: number of accepted records

- integer(mpi), dimension(:), allocatable dfd

  file: ndf sum

- integer(mpi), dimension(:), allocatable xfd

  file: max. record size

- real(mps), dimension(:), allocatable cfd

  file: chi2 sum

- real(mps), dimension(:), allocatable ofd

  file: option

- real(mps), dimension(:), allocatable wfd

  file: weight

- character(len=1024) filnam

  name of steering file

- integer(mpi) nfnam

  length of sterring file name

- character, dimension(:), allocatable tfd

  file names (concatenation)

- integer(mpi) ifile

current file (index)

- integer(mpi) nfiles

  number of files
- integer(mpi) nfilb

  number of binary files
- integer(mpi) nfilf

  number of Fortran binary files
- integer(mpi) nfilc

  number of C binary files
- integer(mpi) nfilw

  number of weighted binary files
- integer(mpi) ndimbuf =10000

  default read buffer size (I/F words, half record length)

8.14.1 Detailed Description

Parameters, variables, dynamic arrays.

For parameters which can be set from command line or steering files more details are available in: List of options and commands.

Definition at line 9 of file mpmod.f90.

8.14.2 Member Data Documentation

8.14.2.1 real(mpd) mpmod::accuratedsum =0.0_mpd

fractional part of sum

Definition at line 122 of file mpmod.f90.

8.14.2.2 integer(mpi) mpmod::accuratenexp =0

sum / 2∗∗20

Definition at line 124 of file mpmod.f90.

8.14.2.3 integer(mpi) mpmod::accuratensum =0

sum mod 2∗∗20

Definition at line 123 of file mpmod.f90.

8.14.2.4 real(mps) mpmod::actfun =0.0

actual function change

Definition at line 45 of file mpmod.f90.

8.14.2.5 real(mps) mpmod::angras =0.0

angle between gradient and search direction

Definition at line 46 of file mpmod.f90.

8.14.2.6 real(mpd), dimension(:), allocatable mpmod::aux

local fit 'solutions for border rows'

Definition at line 186 of file mpmod.f90.

8.14.2.7 integer(mpi), dimension(:), allocatable mpmod::backindexusage

list of global par in record

Definition at line 178 of file mpmod.f90.

8.14.2.8 real(mpd), dimension(:), allocatable mpmod::blvec

local fit vector 'b' (in A∗x=b), replaced by 'x'

Definition at line 180 of file mpmod.f90.

8.14.2.9 real(mps), dimension(:), allocatable mpmod::cfd

file: chi2 sum

Definition at line 225 of file mpmod.f90.

8.14.2.10 real(mps) mpmod::chhuge =50.0

cut in terms of 3-sigma for unreasonable data, all iterations

Definition at line 28 of file mpmod.f90.

8.14.2.11 real(mps) mpmod::chicut =0.0

cut in terms of 3-sigma cut, first iteration

Definition at line 26 of file mpmod.f90.

8.14.2.12 real(mps) mpmod::chirem =0.0

cut in terms of 3-sigma cut, other iterations, approaching 1.

Definition at line 27 of file mpmod.f90.

8.14.2.13 real(mpd), dimension(:), allocatable mpmod::clmat

local fit matrix 'A' (in A∗x=b)

Definition at line 181 of file mpmod.f90.

8.14.2.14 real(mps) mpmod::delfun =0.0

expected function change

Definition at line 44 of file mpmod.f90.

8.14.2.15 real(mps) mpmod::deltim

cpu time difference

Definition at line 101 of file mpmod.f90.

8.14.2.16 integer(mpi), dimension(:), allocatable mpmod::dfd

file: ndf sum

Definition at line 223 of file mpmod.f90.

8.14.2.17 real(mps) mpmod::dflim

convergence limit

Definition at line 90 of file mpmod.f90.

8.14.2.18 real(mps) mpmod::dwcut =0.0

down-weight fraction cut

Definition at line 35 of file mpmod.f90.

8.14.2.19 real(mps), dimension(3) mpmod::fcache = (/ 0.8, 0., 0. /)

read cache, average fill level; write cache; dynamic size

Definition at line 70 of file mpmod.f90.

8.14.2.20 character (len=1024) mpmod::filnam

name of steering file

Definition at line 228 of file mpmod.f90.

8.14.2.21 real(mpd) mpmod::flines

function value line search

Definition at line 111 of file mpmod.f90.

8.14.2.22 real(mpd) mpmod::fvalue

function value (chi2 sum) solution

Definition at line 110 of file mpmod.f90.

8.14.2.23 real(mpd), dimension(:), allocatable mpmod::globalcorrections

correction x_i (from A∗x_i=b_i in iteration i)

Definition at line 131 of file mpmod.f90.

8.14.2.24 integer(mpi), dimension(:), allocatable mpmod::globalindexusage

indices of global par in record

Definition at line 177 of file mpmod.f90.

8.14.2.25 real(mpd), dimension(:), allocatable mpmod::globalmatd

global matrix 'A' (double, full or sparse)

Definition at line 136 of file mpmod.f90.

8.14.2.26 real(mps), dimension(:), allocatable mpmod::globalmatf

global matrix 'A' (float part for compressed sparse)

Definition at line 137 of file mpmod.f90.

8.14.2.27 real(mpd), dimension(:), allocatable mpmod::globalparameter

global parameters (start values + sum(x_i))

Definition at line 129 of file mpmod.f90.

8.14.2.28 real(mpd), dimension(:), allocatable mpmod::globalparcopy

copy of global parameters

Definition at line 130 of file mpmod.f90.

8.14.2.29 integer(mpi), dimension(:), allocatable mpmod::globalparhashtable

global parameters hash table

Definition at line 154 of file mpmod.f90.

8.14.2.30 integer(mpi), dimension(-7:0) mpmod::globalparheader = 0

global parameters (mapping) header

0: length of labels/indices;

-1: number of stored items;

-2: =0 during build-up;

-3: next number;

-4: (largest) prime number ($<$ length);

-5: number of overflows;

-6: nr of variable parameters;

-7: call counter for build-up;

Definition at line 156 of file mpmod.f90.

8.14.2.31 integer(mpi), dimension(:,:), allocatable mpmod::globalparlabelindex

global parameters label, total -$>$ var. index

Definition at line 153 of file mpmod.f90.

8.14.2.32 real(mps), dimension(:), allocatable mpmod::globalparpresigma

pre-sigma for global parameters

Definition at line 133 of file mpmod.f90.

8.14.2.33 real(mps), dimension(:), allocatable mpmod::globalparpreweight

weight from pre-sigma

Definition at line 134 of file mpmod.f90.

8.14.2.34 real(mps), dimension(:), allocatable mpmod::globalparstart

start value for global parameters

Definition at line 132 of file mpmod.f90.

8.14.2.35 integer(mpi), dimension(:), allocatable mpmod::globalparvartototal

global parameters variable -> total index

Definition at line 155 of file mpmod.f90.

8.14.2.36 real(mpd), dimension(:), allocatable mpmod::globalvector

global vector 'x' (in A∗x=b)

Definition at line 138 of file mpmod.f90.

8.14.2.37 integer(mpi), dimension(:), allocatable mpmod::ibandh

local fit 'band width histogram' (band size autodetection)

Definition at line 182 of file mpmod.f90.

8.14.2.38 integer(mpi) mpmod::icalcm =0

calculation mode (for XLOOPN) , >0: calculate matrix

Definition at line 52 of file mpmod.f90.

8.14.2.39 integer(mpi) mpmod::ictest =0

test mode '-t'

Definition at line 14 of file mpmod.f90.

8.14.2.40 integer(mpi), dimension(:), allocatable mpmod::ifd

file: integrated record numbers (=offset)

Definition at line 221 of file mpmod.f90.

8.14.2.41 integer(mpi) mpmod::ifile

current file (index)

Definition at line 231 of file mpmod.f90.

8.14.2.42 integer(mpi) mpmod::iforce =0

switch to SUBITO for (global) rank defects if zero

Definition at line 73 of file mpmod.f90.

8.14.2.43 integer(mpi) mpmod::igcorr =0

flag for output of global correlations for inversion, =0: none

Definition at line 74 of file mpmod.f90.

8.14.2.44 integer(mpi) mpmod::iitera

MINRES iterations.

Definition at line 97 of file mpmod.f90.

8.14.2.45 integer(mpi), dimension(:), allocatable mpmod::indprecond

preconditioner pointer array

Definition at line 141 of file mpmod.f90.

8.14.2.46 integer(mpi) mpmod::istopa

MINRES istop (convergence)

Definition at line 98 of file mpmod.f90.

8.14.2.47 integer(mpi) mpmod::isubit =0

subito flag '-s'

Definition at line 36 of file mpmod.f90.

8.14.2.48 integer(mpi) mpmod::iterat =0

iterations in solution

Definition at line 47 of file mpmod.f90.

8.14.2.49 integer(mpi), dimension(:), allocatable mpmod::jfd

file: number of accepted records

Definition at line 222 of file mpmod.f90.

8.14.2.50 integer(mpi), dimension(:,:), allocatable mpmod::kfd

(1,.)= number of records in file, (2,..)= file order

Definition at line 220 of file mpmod.f90.

8.14.2.51 integer(mpi), dimension(100) mpmod::lbmnrs

MINRES error labels.

Definition at line 109 of file mpmod.f90.

8.14.2.52 integer(mpi) mpmod::lcalcm

last calclation mode

Definition at line 106 of file mpmod.f90.

8.14.2.53 integer(mpi) mpmod::lenconstraints =0

length of list of constraints from steering file

Definition at line 211 of file mpmod.f90.

8.14.2.54 integer(mpi) mpmod::lenglobalvec

length of global vector 'b' (A∗x=b)

Definition at line 125 of file mpmod.f90.

8.14.2.55 integer(mpi) mpmod::lenmeasurements =0

length of list of measurements from steering file

Definition at line 213 of file mpmod.f90.

8.14.2.56 integer(mpi) mpmod::lenparameters =0

list items from steering file

length of list of parameters from steering file

Definition at line 207 of file mpmod.f90.

8.14.2.57 integer(mpi) mpmod::lenpresigmas =0

length of list of pre-sigmas from steering file

Definition at line 209 of file mpmod.f90.

8.14.2.58 integer(mpi), dimension(:), allocatable mpmod::lfd

length of file name

Definition at line 218 of file mpmod.f90.

8.14.2.59 integer(mpi) mpmod::lfitbb =1

local fit: check for bordered band matrix (if >0)

Definition at line 67 of file mpmod.f90.

8.14.2.60 integer(mpi) mpmod::lfitnp =huge(lfitnp)

local fit: number of iteration to calculate pulls

Definition at line 66 of file mpmod.f90.

8.14.2.61 integer(mpi) mpmod::lhuber =0

Huber down-weighting flag.

Definition at line 25 of file mpmod.f90.

8.14.2.62 type(listitem), dimension(:), allocatable mpmod::listconstraints

list of constraints from steering file

Definition at line 212 of file mpmod.f90.

8.14.2.63 type(listitem), dimension(:), allocatable mpmod::listmeasurements

list of measurements from steering file

Definition at line 214 of file mpmod.f90.

8.14.2.64 type(listitem), dimension(:), allocatable mpmod::listparameters

list of parameters from steering file

Definition at line 208 of file mpmod.f90.

8.14.2.65 type(listitem), dimension(:), allocatable mpmod::listpresigmas

list of pre-sgmas from steering file

Definition at line 210 of file mpmod.f90.

8.14.2.66 real(mps), dimension(:), allocatable mpmod::localcorrections

local fit corrections (to residuals)

Definition at line 191 of file mpmod.f90.

8.14.2.67 real(mpd), dimension(:), allocatable mpmod::localglobalmatrix

matrix correlating local and global par

Definition at line 192 of file mpmod.f90.

8.14.2.68 integer(mpi) mpmod::lsearch =2

iterations (solutions) with line search: >2: all, =2: all with (next) Chi2 cut scaling factor =1., =1: last, <1: none
Definition at line 77 of file mpmod.f90.

8.14.2.69 integer(mpi) mpmod::lsinfo

line search: returned information
Definition at line 99 of file mpmod.f90.

8.14.2.70 integer(mpi) mpmod::lunkno =0

flag for unkown keywords
Definition at line 24 of file mpmod.f90.

8.14.2.71 integer(mpi) mpmod::lunlog

unit for logfile
Definition at line 80 of file mpmod.f90.

8.14.2.72 integer(mpi) mpmod::lvllog

log level
Definition at line 81 of file mpmod.f90.

8.14.2.73 real(mpd), dimension(:), allocatable mpmod::matconsproduct

product matrix of constraints
Definition at line 150 of file mpmod.f90.

8.14.2.74 integer(mpi) mpmod::matmon =0

record interval for monitoring of (sparse) matrix construction
Definition at line 65 of file mpmod.f90.

8.14.2.75 real(mpd), dimension(:), allocatable mpmod::matprecond

preconditioner (band) matrix
Definition at line 140 of file mpmod.f90.

8.14.2.76 integer(mpi) mpmod::matrit =0

matrix calculation up to iteration MATRIT
Definition at line 51 of file mpmod.f90.

8.14.2.77 integer(mpi) mpmod::matsto =2

(global) matrix storage mode (1: full, 2: sparse)

Definition at line 16 of file mpmod.f90.

8.14.2.78 integer(mpi) mpmod::maxrecordsinblock

max. records in block

Definition at line 119 of file mpmod.f90.

8.14.2.79 integer(mpi) mpmod::mbandw =0

band width of preconditioner matrix

Definition at line 23 of file mpmod.f90.

8.14.2.80 integer(mpi) mpmod::mcmprs =0

compression flag for sparsity (column indices)

Definition at line 62 of file mpmod.f90.

8.14.2.81 integer(mpi) mpmod::mdebg2 =10

number of measurements for record debug printout

Definition at line 19 of file mpmod.f90.

8.14.2.82 integer(mpi) mpmod::mdebug =0

debug flag (number of records to print)

Definition at line 18 of file mpmod.f90.

8.14.2.83 integer(mpi) mpmod::memdbg =0

debug flag for memory management

Definition at line 75 of file mpmod.f90.

8.14.2.84 integer(mpi) mpmod::metsol =0

solution method (1: inversion, 2: diagonalization, 3: MINRES-QLP)

Definition at line 15 of file mpmod.f90.

8.14.2.85 integer(mpi), dimension(:), allocatable mpmod::mfd

file mode: cbinary =1, text =2, fbinary=3

Definition at line 217 of file mpmod.f90.

8.14.2.86 integer(mpi) mpmod::mhispe =0

upper bound for pair entry histogrammimg

Definition at line 60 of file mpmod.f90.

8.14.2.87 integer(mpi) mpmod::minrecordsinblock

min. records in block

Definition at line 118 of file mpmod.f90.

8.14.2.88 integer(mpi) mpmod::mitera =1

number of iterations

Definition at line 21 of file mpmod.f90.

8.14.2.89 integer(mpi) mpmod::mnrsel =0

number of MINRES error labels in LBMNRS (calc err, corr with SOLGLO)

Definition at line 68 of file mpmod.f90.

8.14.2.90 integer(mpi) mpmod::mnrsit =0

total number of MINRES internal iterations

Definition at line 72 of file mpmod.f90.

8.14.2.91 integer(mpi) mpmod::mprint =1

print flag (0: minimal, 1: normal, $>$1: more)

Definition at line 17 of file mpmod.f90.

8.14.2.92 integer(mpi) mpmod::mreqen =10

required number of entries (for variable global parameter)

Definition at line 20 of file mpmod.f90.

8.14.2.93 integer(mpi) mpmod::mreqpe =1

min number of pair entries

Definition at line 59 of file mpmod.f90.

8.14.2.94 real(mpd) mpmod::mrestl =1.0E-06

tolerance criterion for MINRES-QLP

Definition at line 39 of file mpmod.f90.

8.14.2.95 integer(mpi) mpmod::mrmode =0

MINRES-QLP mode (0: QR+QLP, 1: only QR, 2: only QLP factorization)

Definition at line 41 of file mpmod.f90.

8.14.2.96 real(mpd) mpmod::mrtcnd =1.0E+07

transition (QR -> QLP) (matrix) condition for MINRES-QLP

Definition at line 40 of file mpmod.f90.

8.14.2.97 integer(mpi) mpmod::msngpe =0

upper bound for pair entry single precision storage

Definition at line 61 of file mpmod.f90.

8.14.2.98 integer(mpi) mpmod::mthrd =1

number of (OpenMP) threads

Definition at line 63 of file mpmod.f90.

8.14.2.99 integer(mpi) mpmod::mthrdr =1

number of threads for reading binary files

Definition at line 71 of file mpmod.f90.

8.14.2.100 integer(mpi) mpmod::mxrec =0

max number of records

Definition at line 64 of file mpmod.f90.

8.14.2.101 integer(mpi) mpmod::naeqn

max number of equations (measurements) per record

Definition at line 88 of file mpmod.f90.

8.14.2.102 integer(mpi) mpmod::nagb

number of fit parameters (global par. + Lagrange mult.)

Definition at line 84 of file mpmod.f90.

8.14.2.103 integer(mpi) mpmod::nagbn

max number of global paramters per record

Definition at line 86 of file mpmod.f90.

8.14.2.104 integer(mpi) mpmod::nalcn

max number of local paramters per record

Definition at line 87 of file mpmod.f90.

8.14.2.105 integer(mpi) mpmod::nbdrx =0

max border size for local fit

Definition at line 55 of file mpmod.f90.

8.14.2.106 integer(mpi) mpmod::nbndr =0

number of records with bordered band matrix for local fit

Definition at line 54 of file mpmod.f90.

8.14.2.107 integer(mpi) mpmod::nbndx =0

max band width for local fit

Definition at line 56 of file mpmod.f90.

8.14.2.108 integer(mpi) mpmod::ncache =-1

buffer size for caching (default 100MB per thread)

Definition at line 69 of file mpmod.f90.

8.14.2.109 integer(mpi) mpmod::ncgb

number of constraints

Definition at line 85 of file mpmod.f90.

8.14.2.110 integer(mpi) mpmod::ndefec =0

rank deficit for global matrix (from inversion)

Definition at line 104 of file mpmod.f90.

8.14.2.111 integer(mpi) mpmod::ndfsum

sum(ndf)

Definition at line 96 of file mpmod.f90.

8.14.2.112 integer(mpi) mpmod::ndimbuf =10000

default read buffer size (I/F words, half record length)

Definition at line 237 of file mpmod.f90.

8.14.2.113 integer(mpi) mpmod::nencdb

encoding info (number bits for column counter)

Definition at line 108 of file mpmod.f90.

8.14.2.114 logical mpmod::newite

flag for new iteration

Definition at line 95 of file mpmod.f90.

8.14.2.115 integer(mpi), parameter mpmod::nexp20 =1048576

Definition at line 121 of file mpmod.f90.

8.14.2.116 integer(mpi), dimension(:), allocatable mpmod::nfd

index (line) in (steering) file

Definition at line 219 of file mpmod.f90.

8.14.2.117 integer(mpi) mpmod::nfilb

number of binary files

Definition at line 233 of file mpmod.f90.

8.14.2.118 integer(mpi) mpmod::nfilc

number of C binary files

Definition at line 235 of file mpmod.f90.

8.14.2.119 integer(mpi) mpmod::nfiles

number of files

Definition at line 232 of file mpmod.f90.

8.14.2.120 integer(mpi) mpmod::nfilf

number of Fortran binary files

Definition at line 234 of file mpmod.f90.

8.14.2.121 integer(mpi) mpmod::nfilw

number of weighted binary files

Definition at line 236 of file mpmod.f90.

8.14.2.122 integer(mpi) mpmod::nfnam

length of sterring file name

Definition at line 229 of file mpmod.f90.

8.14.2.123 integer(mpi) mpmod::nhistp =0

flag for histogram printout

Definition at line 43 of file mpmod.f90.

8.14.2.124 integer(mpi) mpmod::nloopn =0

number of data reading, fitting loops

Definition at line 22 of file mpmod.f90.

8.14.2.125 integer(mpi) mpmod::nmiss1 =0

rank deficit for constraints

Definition at line 105 of file mpmod.f90.

8.14.2.126 integer(mpi) mpmod::nofeas =0

flag for skipping making parameters feasible

Definition at line 42 of file mpmod.f90.

8.14.2.127 integer(mpi) mpmod::npresg

number of pre-sigmas

Definition at line 102 of file mpmod.f90.

8.14.2.128 integer(mpi) mpmod::nrec

(current) record number

Definition at line 89 of file mpmod.f90.

8.14.2.129 integer(mpi) mpmod::nrec1 =0

record number with largest residual

Definition at line 31 of file mpmod.f90.

8.14.2.130 integer(mpi) mpmod::nrec2 =0

record number with largest chi^2/Ndf

Definition at line 32 of file mpmod.f90.

8.14.2.131 integer(mpi) mpmod::nrec3 = huge(nrec3)

(1.) record number with error

Definition at line 58 of file mpmod.f90.

8.14.2.132 integer(mpi) mpmod::nrecal

number of records

Definition at line 103 of file mpmod.f90.

8.14.2.133 integer(mpi) mpmod::nrecer =0

record with error (rank deficit or Not-a-Number) for printout

Definition at line 57 of file mpmod.f90.

8.14.2.134 integer(mpi) mpmod::nrecp2 =0

record number with printout

Definition at line 30 of file mpmod.f90.

8.14.2.135 integer(mpi) mpmod::nrecpr =0

record number with printout

Definition at line 29 of file mpmod.f90.

8.14.2.136 integer(mpi) mpmod::nregul =0

regularization flag

Definition at line 48 of file mpmod.f90.

8.14.2.137 integer(mpi), dimension(0:3) mpmod::nrejec

rejected events

Definition at line 91 of file mpmod.f90.

8.14.2.138 integer(mpi) mpmod::nspc

number of precision for sparse global matrix (1=D, 2=D+F)

Definition at line 107 of file mpmod.f90.

8.14.2.139 integer(mpi) mpmod::ntgb

total number of global parameters

Definition at line 82 of file mpmod.f90.

8.14.2.140 integer(mpi) mpmod::numbit =1

number of bits for pair counters

Definition at line 53 of file mpmod.f90.

8.14.2.141 integer(mpi) mpmod::numblocks

number of (read) blocks

Definition at line 115 of file mpmod.f90.

8.14.2.142 integer(mpi) mpmod::numreadbuffer

number of buffers (records) in (read) block

Definition at line 114 of file mpmod.f90.

8.14.2.143 integer(mpi) mpmod::nvgb

number of variable global parameters

Definition at line 83 of file mpmod.f90.

8.14.2.144 real(mps), dimension(:), allocatable mpmod::ofd

file: option

Definition at line 226 of file mpmod.f90.

8.14.2.145 real(mps) mpmod::prange =0.0

range (-PRANGE..PRANGE) for histograms of pulls, norm. residuals

Definition at line 76 of file mpmod.f90.

8.14.2.146 real(mps), dimension(:), allocatable mpmod::readbufferdataf

float data

Definition at line 175 of file mpmod.f90.

8.14.2.147 integer(mpi), dimension(:), allocatable mpmod::readbufferdatai

integer data

Definition at line 174 of file mpmod.f90.

8.14.2.148 integer(mpi), dimension(:,:), allocatable mpmod::readbufferinfo

buffer management (per thread)

Definition at line 172 of file mpmod.f90.

8.14.2.149 integer(mpi), dimension(:), allocatable mpmod::readbufferpointer

pointer to used buffers

Definition at line 173 of file mpmod.f90.

8.14.2.150 real(mps) mpmod::regpre =0.0

default presigma

Definition at line 50 of file mpmod.f90.

8.14.2.151 real(mps) mpmod::regula =1.0

regularization parameter, add regula $*$ norm(global par.) to objective function

Definition at line 49 of file mpmod.f90.

8.14.2.152 real mpmod::rstart

cpu start time for solution iterations

Definition at line 100 of file mpmod.f90.

8.14.2.153 real(mpd), dimension(:), allocatable mpmod::scdiag

local fit workspace (D)

Definition at line 189 of file mpmod.f90.

8.14.2.154 integer(mpi), dimension(:), allocatable mpmod::scflag

local fit workspace (I)

Definition at line 190 of file mpmod.f90.

8.14.2.155 integer(mpi) mpmod::skippedrecords

number of skipped records (buffer too small)

Definition at line 117 of file mpmod.f90.

8.14.2.156 integer(mpi), dimension(:), allocatable mpmod::sparsematrixcolumns

(compressed) list of columns for sparse matrix

Definition at line 169 of file mpmod.f90.

8.14.2.157 integer(mpi), dimension(:), allocatable mpmod::sparsematrixcompression

compression info (per row)

Definition at line 168 of file mpmod.f90.

8.14.2.158 integer(mpl), dimension(:,:), allocatable mpmod::sparsematrixoffsets

row offsets for column list, sparse matrix elements

Definition at line 170 of file mpmod.f90.

8.14.2.159 real(mps) mpmod::stepl

step length (line search)

Definition at line 93 of file mpmod.f90.

8.14.2.160 real(mpd) mpmod::sumndf

weighted sum(ndf)

Definition at line 112 of file mpmod.f90.

8.14.2.161 integer(mpi) mpmod::sumrecords

sum of records

Definition at line 116 of file mpmod.f90.

8.14.2.162 character (len=74) mpmod::textl

name of current MP 'module' (step)

Definition at line 94 of file mpmod.f90.

8.14.2.163 character, dimension(:), allocatable mpmod::tfd

file names (concatenation)

Definition at line 230 of file mpmod.f90.

8.14.2.164 real(mps), dimension(0:8) mpmod::times

cpu time counters

Definition at line 92 of file mpmod.f90.

8.14.2.165 real(mps) mpmod::value1 =0.0

largest residual

Definition at line 33 of file mpmod.f90.

8.14.2.166 real(mps) mpmod::value2 =0.0

largest chi^2/Ndf

Definition at line 34 of file mpmod.f90.

8.14.2.167 real(mpd), dimension(:), allocatable mpmod::vbdr

local fit border part of 'A'

Definition at line 185 of file mpmod.f90.

8.14.2.168 real(mpd), dimension(:), allocatable mpmod::vbk

local fit 'matrix for border solution'

Definition at line 187 of file mpmod.f90.

8.14.2.169 real(mpd), dimension(:), allocatable mpmod::vbnd

local fit band part of 'A'

Definition at line 184 of file mpmod.f90.

8.14.2.170 real(mpd), dimension(:), allocatable mpmod::vecconsresiduals

residuals of constraints

Definition at line 151 of file mpmod.f90.

8.14.2.171 real(mpd), dimension(:), allocatable mpmod::vzru

local fit 'border solution'

Definition at line 188 of file mpmod.f90.

8.14.2.172 real(mps), dimension(:), allocatable mpmod::wfd

file: weight

Definition at line 227 of file mpmod.f90.

8.14.2.173 real(mps) mpmod::wolfc1 =0.0

C_1 of strong Wolfe condition.

Definition at line 37 of file mpmod.f90.

8.14.2.174 real(mps) mpmod::wolfc2 =0.0

C_2 of strong Wolfe condition.

Definition at line 38 of file mpmod.f90.

8.14.2.175 real(mpd), dimension(:), allocatable mpmod::workspaced

(general) workspace (D)

Definition at line 143 of file mpmod.f90.

8.14.2.176 real(mpd), dimension(:), allocatable mpmod::workspacediagonalization

workspace diag.

Definition at line 145 of file mpmod.f90.

8.14.2.177 real(mpd), dimension(:), allocatable mpmod::workspaceeigenvalues

workspace eigen values

Definition at line 146 of file mpmod.f90.

8.14.2.178 real(mpd), dimension(:), allocatable mpmod::workspaceeigenvectors

workspace eigen vectors

Definition at line 147 of file mpmod.f90.

8.14.2.179 integer(mpi), dimension(:), allocatable mpmod::workspacei

(general) workspace (I)

Definition at line 148 of file mpmod.f90.

8.14.2.180 real(mpd), dimension(:), allocatable mpmod::workspacelinesearch

workspace line search

Definition at line 144 of file mpmod.f90.

8.14.2.181 real(mps), dimension(:,:), allocatable mpmod::writebufferdata

write buffer data (largest residual, Chi2/ndf, per thread)

Definition at line 195 of file mpmod.f90.

8.14.2.182 integer(mpi), dimension(-6:6) mpmod::writebufferheader = 0

write buffer header (-6..-1: updates, 1..6: indices)

+/-1: buffer size (words) per thread;

+/-2: min number of free words;

+/-3: number of buffer flushes;

+/-4: number of buffer overruns;

+/-5: average fill level;

+/-6: peak fill level;

Definition at line 198 of file mpmod.f90.

8.14.2.183 integer(mpi), dimension(:), allocatable mpmod::writebufferindices

write buffer for indices

Definition at line 196 of file mpmod.f90.

8.14.2.184 integer(mpi), dimension(:,:), allocatable mpmod::writebufferinfo

write buffer management (per thread)

Definition at line 194 of file mpmod.f90.

8.14.2.185 real(mpd), dimension(:), allocatable mpmod::writebufferupdates

write buffer for update matrices

Definition at line 197 of file mpmod.f90.

8.14.2.186 integer(mpi), dimension(:), allocatable mpmod::xfd

file: max. record size

Definition at line 224 of file mpmod.f90.

The documentation for this module was generated from the following file:

- mpmod.f90

8.15 mptest1 Module Reference

Parameters and data.

Public Attributes

- integer(mpi), parameter nplan =100
- real(mps), parameter detx = 10.0
  x-value of first plane
- real(mps), parameter disx = 10.0
  distance between planes
- real(mps), parameter thck = 2.0
  thickness of plane
- real(mps), parameter heit =100.0
  height of detector plane
- real(mps), parameter effp =0.90
  plane efficiency
- real(mps), parameter sgmp =0.0150
  measurement sigma
- real(mps), dimension(nplan) del
  shift (position deviation) (alignment parameter)
- real(mps), dimension(nplan) dvd
  rel. drift velocity deviation (calibration parameter)
- real(mps) ynull
  track position at vertex
- real(mps) slope
  track slope
- integer(mpi) nhits
  number of hits
- integer(mpi), dimension(nplan) ihits

plane numbers (planes with hits)

- real(mps), dimension(nplan) eff

  plane efficiency

- real(mps), dimension(nplan) sgm

  measurement sigma (plane)

- real(mps), dimension(nplan) ydrft

  signed drift length

- real(mps), dimension(nplan) xhits

  position perp. to plane (hit)

- real(mps), dimension(nplan) yhits

  measured position in plane (hit)

- real(mps), dimension(nplan) sigma

  measurement sigma (hit)

8.15.1 Detailed Description

Parameters and data.

Definition at line 16 of file mptest1.f90.

8.15.2 Member Data Documentation

8.15.2.1 real(mps), dimension(nplan) mptest1::del

shift (position deviation) (alignment parameter)

Definition at line 33 of file mptest1.f90.

8.15.2.2 real(mps), parameter mptest1::detx = 10.0

x-value of first plane

Definition at line 25 of file mptest1.f90.

8.15.2.3 real(mps), parameter mptest1::disx = 10.0

distance between planes

Definition at line 26 of file mptest1.f90.

8.15.2.4 real(mps), dimension(nplan) mptest1::dvd

rel. drift velocity deviation (calibration parameter)

Definition at line 34 of file mptest1.f90.

8.15.2.5 real(mps), dimension(nplan) mptest1::eff

plane efficiency

Definition at line 41 of file mptest1.f90.

8.15.2.6 real(mps), parameter mptest1::effp =0.90

plane efficiency

Definition at line 29 of file mptest1.f90.

8.15.2.7 real(mps), parameter mptest1::heit =100.0

height of detector plane

Definition at line 28 of file mptest1.f90.

8.15.2.8 integer(mpi), dimension(nplan) mptest1::ihits

plane numbers (planes with hits)

Definition at line 40 of file mptest1.f90.

8.15.2.9 integer(mpi) mptest1::nhits

number of hits

Definition at line 39 of file mptest1.f90.

8.15.2.10 integer(mpi), parameter mptest1::nplan =100

Definition at line 22 of file mptest1.f90.

8.15.2.11 real(mps), dimension(nplan) mptest1::sgm

measurement sigma (plane)

Definition at line 42 of file mptest1.f90.

8.15.2.12 real(mps), parameter mptest1::sgmp =0.0150

measurement sigma

Definition at line 30 of file mptest1.f90.

8.15.2.13 real(mps), dimension(nplan) mptest1::sigma

measurement sigma (hit)

Definition at line 46 of file mptest1.f90.

8.15.2.14 real(mps) mptest1::slope

track slope

Definition at line 37 of file mptest1.f90.

8.15.2.15 real(mps), parameter mptest1::thck = 2.0

thickness of plane

Definition at line 27 of file mptest1.f90.

8.15.2.16 real(mps), dimension(nplan) mptest1::xhits

position perp. to plane (hit)

Definition at line 44 of file mptest1.f90.

8.15.2.17 real(mps), dimension(nplan) mptest1::ydrft

signed drift length

Definition at line 43 of file mptest1.f90.

8.15.2.18 real(mps), dimension(nplan) mptest1::yhits

measured position in plane (hit)

Definition at line 45 of file mptest1.f90.

8.15.2.19 real(mps) mptest1::ynull

track position at vertex

Definition at line 36 of file mptest1.f90.

The documentation for this module was generated from the following file:

- mptest1.f90

8.16 mptest2 Module Reference

Parameters and data.

Public Attributes

- integer(mpi), parameter nlyr =10
  number of detector layers
- integer(mpi), parameter nmlyr =14
  number of measurement layers
- integer(mpi), parameter nmx =10
  number of modules in x direction
- integer(mpi), parameter nmy =5
  number of modules in y direction
- integer(mpi), parameter ntot =nlyr∗nmx∗nmy
  total number of modules
- real(mps), parameter dets = 10.0
  arclength of first plane
- real(mps), parameter diss = 10.0

distance between planes

- real(mps), parameter thck = 0.02

  thickness of plane (X0)
- real(mps), parameter offs = 0.5

  offset of stereo modules
- real(mps), parameter stereo =0.08727

  stereo angle
- real(mps), parameter sizel = 20.0

  size of layers
- real(mps), parameter sigl =0.002
- integer(mpi) nhits

  number of hits
- real(mps) the0

  multiple scattering error
- integer(mpi), dimension(nmlyr) islyr

  (detector) layer
- integer(mpi), dimension(nmlyr) ihits

  module number
- real(mps), dimension(ntot) sdevx

  shift in x (alignment parameter)
- real(mps), dimension(ntot) sdevy

  shift in y (alignment parameter)
- real(mps), dimension(nmlyr) sarc

  arc length
- real(mps), dimension(nmlyr) ssig

  resolution
- real(mps), dimension(2, nmlyr) spro

  projection of measurent direction in (XY)
- real(mps), dimension(nmlyr) xhits

  position perp. to plane (hit)
- real(mps), dimension(nmlyr) yhits

  measured position in plane (hit)
- real(mps), dimension(nmlyr) sigma

  measurement sigma (hit)

8.16.1 Detailed Description

Parameters and data.

Definition at line 39 of file mptest2.f90.

8.16.2 Member Data Documentation

8.16.2.1 real(mps), parameter mptest2::dets = 10.0

arclength of first plane

Definition at line 51 of file mptest2.f90.

8.16.2.2 real(mps), parameter mptest2::diss = 10.0

distance between planes

Definition at line 52 of file mptest2.f90.

8.16.2.3 integer(mpi), dimension(nmlyr) mptest2::ihits

module number

Definition at line 62 of file mptest2.f90.

8.16.2.4 integer(mpi), dimension(nmlyr) mptest2::islyr

(detector) layer

Definition at line 61 of file mptest2.f90.

8.16.2.5 integer(mpi) mptest2::nhits

number of hits

Definition at line 59 of file mptest2.f90.

8.16.2.6 integer(mpi), parameter mptest2::nlyr =10

number of detector layers

Definition at line 45 of file mptest2.f90.

8.16.2.7 integer(mpi), parameter mptest2::nmlyr =14

number of measurement layers

Definition at line 46 of file mptest2.f90.

8.16.2.8 integer(mpi), parameter mptest2::nmx =10

number of modules in x direction

Definition at line 47 of file mptest2.f90.

8.16.2.9 integer(mpi), parameter mptest2::nmy =5

number of modules in y direction

Definition at line 48 of file mptest2.f90.

8.16.2.10 integer(mpi), parameter mptest2::ntot =nlyr∗nmx∗nmy

total number of modules

Definition at line 49 of file mptest2.f90.

8.16.2.11 real(mps), parameter mptest2::offs = 0.5

offset of stereo modules

Definition at line 54 of file mptest2.f90.

8.16.2.12 real(mps), dimension(nmlyr) mptest2::sarc

arc length

Definition at line 65 of file mptest2.f90.

8.16.2.13 real(mps), dimension(ntot) mptest2::sdevx

shift in x (alignment parameter)

Definition at line 63 of file mptest2.f90.

8.16.2.14 real(mps), dimension(ntot) mptest2::sdevy

shift in y (alignment parameter)

Definition at line 64 of file mptest2.f90.

8.16.2.15 real(mps), parameter mptest2::sigl =0.002

Definition at line 57 of file mptest2.f90.

8.16.2.16 real(mps), dimension(nmlyr) mptest2::sigma

measurement sigma (hit)

Definition at line 70 of file mptest2.f90.

8.16.2.17 real(mps), parameter mptest2::sizel = 20.0

size of layers

Definition at line 56 of file mptest2.f90.

8.16.2.18 real(mps), dimension(2,nmlyr) mptest2::spro

projection of measurent direction in (XY)

Definition at line 67 of file mptest2.f90.

8.16.2.19 real(mps), dimension(nmlyr) mptest2::ssig

resolution

Definition at line 66 of file mptest2.f90.

8.16.2.20 real(mps), parameter mptest2::stereo =0.08727

stereo angle

Definition at line 55 of file mptest2.f90.

8.16.2.21 real(mps), parameter mptest2::thck = 0.02

thickness of plane (X0)

Definition at line 53 of file mptest2.f90.

8.16.2.22 real(mps) mptest2::the0

multiple scattering error

Definition at line 60 of file mptest2.f90.

8.16.2.23 real(mps), dimension(nmlyr) mptest2::xhits

position perp. to plane (hit)

Definition at line 68 of file mptest2.f90.

8.16.2.24 real(mps), dimension(nmlyr) mptest2::yhits

measured position in plane (hit)

Definition at line 69 of file mptest2.f90.

The documentation for this module was generated from the following file:

- mptest2.f90

8.17 mptext Module Reference

Keyword position.

Public Attributes

- integer(mpi) keya
  start (position) of keyword
- integer(mpi) keyb
  end (position) of keyword

8.17.1 Detailed Description

Keyword position.

Definition at line 9 of file mptext.f90.

8.17.2 Member Data Documentation

8.17.2.1 integer(mpi) mptext::keya

start (position) of keyword

Definition at line 14 of file mptext.f90.

8.17.2.2 integer(mpi) mptext::keyb

end (position) of keyword

Definition at line 15 of file mptext.f90.

The documentation for this module was generated from the following file:

- mptext.f90

Chapter 9

File Documentation

9.1 Dbandmatrix.f90 File Reference

Symmetric (band) matrix routines.

Functions/Subroutines

- subroutine dchdec (w, n, aux)

  Decomposition of symmetric matrix.
- REAL(mps) function condes (w, n, aux)

  Etimate condition.
- subroutine dbcdec (w, mp1, n, aux)

  Decomposition of symmetric band matrix.
- subroutine dbcprv (w, mp1, n, b)

  Print corr band matrix and pars.
- subroutine dbcprb (w, mp1, n)

  Print band matrix.
- subroutine db2dec (w, n, aux)

  Decomposition (M=1).
- subroutine db3dec (w, n, aux)

  Decomposition (M=2).
- subroutine dcfdec (w, n)

  Decomposition of symmetric matrix.
- subroutine dbfdec (w, mp1, n)

  Decomposition of symmetric band matrix.

9.1.1 Detailed Description

Symmetric (band) matrix routines. For the original broken lines implementation by V. Blobel (University Hamburg).

```
************************************************************
*                                                          *
*   Subroutines for symmetric and symmetric band matrices, *
*   based on the (square root free) Cholesky decomposition.*
*                                                          *
************************************************************

All floating point arguments are in DOUBLE PRECISION (and all
entry names start with a D).
```

```
The Cholesky decomposition transforms a symmetric matrix W
e.g. the matrix from normal equation of least squares,
according to
                      W = L D L^         (L^ means L transposed)
where D is a diagonal matrix and L is a unit triangular matrix
(diagonal elements all ones, all elements above diagonal zero).

The above decomposition allows to solve a matrix equation
                    W x = b
in two steps, using an auxiliary vector y:

              solve  L y = b  for y, and

            solve D L^ x = y  for x.

The inverse matrix of W can be calculated from the decomposition.

In least-squares normal equations the inverse matrix is equal to
the covariance matrix of the fitted parameters. All diagonal elements
of the inverse matrix, the parameter variances, are positive, and
the matrix is positive-definite (all eigenvalues > 0).

The Cholesky algorithm is stable for a positive-definite matrix.
The standard form of the Cholesky algorithm includes n square roots
for a n-by-n matrix, and is possible only for positive-definite
matrices. The version used here is squareroot-free; this algorithm
does not necessarily break down in the indefinite case, although
it is potentially unstable in this case. All decomposition routines
include a check for singularity, and this check needs an auxiliary
array AUX of dimension n.

Method: The Cholesky algorithm for symmetric matrix decomposition
makes use of the symmetry. The operation count (leading term)
is n**3/6 (compared to n**3/3 for normal Gaussian elimination).
The solution of the two triangular systems involves operations
proportional to n**2.

The real advantage of the Cholesky algorithm is for band matrices,
where all matrix elements outside of a band with total width
(2m+1) around the diagonal are zero. The band structure is kept
in the decomposition, and allows a fast solution of matrix equations.
The operation count (leading term) is proportional to m**2*n
and thus (for fixed m) linear in n. Thus for n=100 and m=2 the
Cholesky algorithm for the band matrix is 1000 times faster than
the standard solution method.

The inverse of a band matrix is a full matrix. It is not necessary
to calculate the inverse, if only the solution for a matrix equation
is needed. However the inverse is often needed, because the elements
of the inverse are the variances and covariances of parameters in
a least-squares fit. The inverse can be calculated afterwards from
the decomposition. Since the inverse matrix is a full matrix, this
has of course an operation count proportional to n**3.

Usually only the elements of the inverse in and around the diagonal
are really needed, and this subset of inverse elements, corresponding
to the original band, can be calculated from the decomposition with
an operation count, which is linear in n. Thus all variances (the
diagonal elements) and covariances between neighbour parameters
are calculated in a short time even for large matrices.

Matrix storage: the mathematical indexing of matrix elements follows
the scheme:

                 (  W11   W12   W13 ... W1n  )
                 (  W21   W22   W23 ... W2n  )
            W =  (  ...   ...   ...     ...  )
                 (  ...   ...   ...     ...  )
                 (  Wn1   Wn2   Wn3 ... Wnn  )

and a storage in an array would require n**2 words, although the
symmetric matrix has only (n**2+n)/2 different elements, and a band
matrix has less than (m+1)*n different elements. Therefore the
```

following storage schemes are used.

Symmetric matrix: the elements are in the order
W11 W12 W22 W13 W23 W33 W14 ... Wnn
with total (n**2+n)/2 array elements.

Band matrix: a band matrix of bandwidth m is stored in an array
of dimension W(m+1,n), according to

W(1,.)	W(2,.)	W(3,.)
W11	W12	W13
W22	W23	W24
W33	W34	W35
...		
Wnn	-	-

The example is for a bandwidth of m=2; three elements at the end
are unused. The diagonal elements are in the array elements W(1,.).

This package includes subroutines for:

(1) Symmetric matrix W: decomposition, solution, inverse

(2) Symmetric band matrix: decomposition, solution, complete
inverse and band part of the inverse

(3) Symmetric band matrix of band width m=1: decomposition,
solution, complete, inverse and band part of the inverse

(4) Symmetric band matrix of band width m=2: decomposition,
solution, complete, inverse and band part of the inverse

(5) Symmetric matrix with band structure, bordered by full row/col
(not yet included)

The subroutines for a fixed band width of m=1 and of m=2 are
faster than the general routine, because certain loops are avoided
and replaced by the direct code.

Historical remark: the square-root algorithm was invented by the
french Mathematician Andre-Louis Cholesky (1875 - 1918).
Cholesky's method of computing solutions to the normal equations was
published 1924, after the death of Cholesky, by Benoit.
The method received little attention after its publication in 1924.
In 1948 the method was analysed in a paper by Fox, Huskey and
Wilkinson, and in the same year Turing published a paper on the
stability of the method.

The fast method to calculate the band part of the inverse matrix
is usually not mentioned in the literature. An exception is:
I.S.Duff, A.M.Erisman and J.K.Reid, Direct Methods for Sparse
Matrices, Oxford Science Publications, 1986.
The following original work is quoted in this book:
K.Takahashi, J.Fagan and M.Chin, Formation of a sparse bus
impedance matrix and its application to short circuit study.
Proceedings 8th PICA Conference, Minneapolis, Minnesota, 1973
A.M.Erisman and W.F.Tinney, On computing certain elements of the
inverse of a sparse matrix, CACM 18, 177-179, 1975

symmetric	decomposit.	solution	inv-element	inverse
n x n matrix	DCHDEC	DCHSLV	-	DCHINV
band matrix m,n	DBCDEC	DBCSLV	DBCIEL/DBCINB	DBCINV
bandwidth m=1	DB2DEC	DB2SLV	DB2IEL	-
bandwidth m=2	DB3DEC	DB3SLV	DB3IEL	-

The DB2... and DB3... routines are special routines for a fixed bandwidth
of 1 and 2, they are faster versions of the general DBG... routines.
The complete inverse matrix can be obtained by DBGINV.
The routine DBGPRI can be used to print all types of band matrices.

```
The decomposition in routines ...DEC replaces (overwrites) the
original matrix (the number of elements is identical). All other
routines require W to be the already decomposed matrix.
The matrix L is a unit lower triangular matrix, with ones on the
diagonal, which have not be stored. Instead the inverse of the
diagonal elements of matrix D are stored in those places.

In the  solution routines ...SLV the array B is the right-hand matrix,
the array is the resulting solution. The same array can be used
for B and X.


W(.) and V(.) are symmetric n-by-n matrices with (N*N+N)/2 elements

SUBROUTINE DCHDEC(W,N, AUX)      ! decomposition, symmetric matrix
     ENTRY DCHSLV(W,N,B, X)      ! solution B -> X
     ENTRY DCHINV(W,N, V)        ! inversion

SUBROUTINE DCFDEC(W,N)           ! modified composition, symmetric
                                 ! alternative to DCHDEC

W(.) and V(.) are band matrices, n rows, band width m (i.e. the total
     width of the band is (2m+1).
     With MP1 = m +1, the array has dimension W(MP1,N).
     The symmetric matrix VS has (N*N+N)/2 elements

SUBROUTINE DBCDEC(W,MP1,N, AUX)  ! decomposition, bandwidth M
     ENTRY DBCSLV(W,MP1,N,B, X)  ! solution B -> X
     ENTRY DBCIEL(W,MP1,N, V)    ! V = inverse band matrix elements
     ENTRY DBCINV(W,MP1,N, VS)   ! V = inverse symmetric matrix

SUBROUTINE DBFDEC(W,MP1,N)       ! modified decomposition, bandwidth M
                                 ! alternative to DBCDEC

SUBROUTINE DBCPRB(W,MP1,N)       ! print band matrix
SUBROUTINE DBCPRV(W,MP1,N,B)     ! print corr band matrix and pars

SUBROUTINE DB2DEC(W,N, AUX)      ! decomposition (M=1)
     ENTRY DB2SLV(W,N,B, X)      ! solution B -> X
     ENTRY DB2IEL(W,N, V)        ! V = inverse band matrix elements

SUBROUTINE DB3DEC(W,N, AUX)      ! decomposition (M=2)
     ENTRY DB3SLV(W,N,B, X)      ! solution B -> X
     ENTRY DB3IEL(W,N, V)        ! V = inverse band matrix elements
```

Definition in file Dbandmatrix.f90.

9.1.2 Function/Subroutine Documentation

9.1.2.1 REAL(mps) function condes (real(mpd), dimension(∗), intent(in) *w,* integer(mpi), intent(in) *n,* real(mpd), dimension(n), intent(inout) *aux*)

Etimate condition.

Parameters

in	*W*	symmetric matrix
in	*N*	size
in	*AUX*	scratch array

Returns

condition

Definition at line 326 of file Dbandmatrix.f90.

9.1.2.2 subroutine db2dec (real(mpd), dimension(2,n), intent(inout) *w,* integer(mpi), intent(inout) *n,* real(mpd), dimension(n), intent(out) *aux*)

Decomposition (M=1).

W is a symmetrix positive definite band matrix of bandwidth 1(+1). W(1,.) are the diagonal elements, W(2,.) is the next diagonals; W(2,N) is never referenced. AUX is an auxiliary array of length N. W is decomposed to L D Lt, where D = diagonal and L unit triangular. A row is set to zero, if the diagonal element is reduced in previous steps by a word length (i.e. global correlation coefficient large). The resulting L and D replace W: the diagonal elements W(1,...) will contain the inverse of the D-elements; the diagonal elements of L are all 1 and not stored. The other elements of L are stored in the corresponding elements of W.

ENTRY DB2SLV(W,N,B, X), solution B -> X

ENTRY DB2IEL(W,N, V), V = inverse band matrix elements

Parameters

in,out	*W*	symmetric band matrix
in	*N*	size
in	*AUX*	scratch array

Definition at line 603 of file Dbandmatrix.f90.

9.1.2.3 subroutine db3dec (real(mpd), dimension(3,n), intent(inout) *w,* integer(mpi), intent(inout) *n,* real(mpd), dimension(n), intent(out) *aux*)

Decomposition (M=2).

W is a symmetrix positive definite band matrix of bandwidth 2(+1). W(1,.) are the diagonal elements, W(2,.) and W(3,.) are the next diagonals; W(3,N-1), W(2,N) and W(3,N) are never referenced. AUX is an auxiliary array of length N. W is decomposed to L D Lt, where D = diagonal and L unit triangular. A row is set to zero, if the diagonal element is reduced in previous steps by a word length (i.e. global correlation coefficient large). The resulting L and D replace W: the diagonal elements W(1,...) will contain the inverse of the D-elements; the diagonal elements of L are all 1 and not stored. The other elements of L are stored in the corresponding elements of W.

ENTRY DB3SLV(W,N,B, X), solution B -> X

ENTRY DB3IEL(W,N, V), V = inverse band matrix elements

Parameters

in,out	*W*	symmetric band matrix
in	*N*	size
in	*AUX*	scratch array

Definition at line 691 of file Dbandmatrix.f90.

9.1.2.4 subroutine dbcdec (real(mpd), dimension(mp1,n), intent(inout) *w,* integer(mpi), intent(in) *mp1,* integer(mpi), intent(in) *n,* real(mpd), dimension(n), intent(out) *aux*)

Decomposition of symmetric band matrix.

ENTRY DBCSLV(W,MP1,N,B, X) for solution B -> X

ENTRY DBCIEL(W,MP1,N, V), V = inverse band matrix elements

ENTRY DBCINB(W,MP1,N, VS), VS = band part of inverse symmetric matrix

ENTRY DBCINV(W,MP1,N, VS), V = inverse symmetric matrix

Parameters

in,out	*W*	symmetric band matrix
in	*MP1*	band width (M) + 1
in	*N*	size
in	*AUX*	scratch array

Definition at line 400 of file Dbandmatrix.f90.

Referenced by sqmibb().

9.1.2.5 subroutine dbcprb (real(mpd), dimension(mp1,n), intent(inout) *w,* integer(mpi), intent(in) *mp1,* integer(mpi), intent(in) *n*)

Print band matrix.

Parameters

in	*W*	symmetric band matrix
in	*MP1*	band width (M) + 1
in	*N*	size

Definition at line 553 of file Dbandmatrix.f90.

9.1.2.6 subroutine dbcprv (real(mpd), dimension(mp1,n), intent(inout) *w,* integer(mpi), intent(in) *mp1,* integer(mpi), intent(in) *n,* real(mpd), dimension(n), intent(out) *b*)

Print corr band matrix and pars.

Parameters

in	*W*	symmetric band matrix
in	*MP1*	band width (M) + 1
in	*N*	size
in	*B*	vector

Definition at line 507 of file Dbandmatrix.f90.

9.1.2.7 subroutine dbfdec (real(mpd), dimension(mp1,n), intent(out) *w,* integer(mpi), intent(inout) *mp1,* integer(mpi), intent(in) *n*)

Decomposition of symmetric band matrix.

Band matrix modified Cholesky decomposition, Philip E.Gill, Walter Murray and Margarete H.Wright: Practical Optimization, Academic Press, 1981

Parameters

in,out	*W*	symmetric band matrix
in	*MP1*	band width (M) + 1
in	*N*	size

Definition at line 885 of file Dbandmatrix.f90.

9.1.2.8 subroutine dcfdec (real(mpd), dimension(∗), intent(out) *w,* integer(mpi), intent(in) *n*)

Decomposition of symmetric matrix.

Modified Cholesky decomposition, Philip E.Gill, Walter Murray and Margarete H.Wright: Practical Optimization, Academic Press, 1981

Parameters

in,out	*W*	symmetirc matrix
in	*N*	size

Definition at line 833 of file Dbandmatrix.f90.

9.1.2.9 subroutine dchdec (real(mpd), dimension(∗), intent(inout) *w,* integer(mpi), intent(in) *n,* real(mpd), dimension(n), intent(out) *aux*)

Decomposition of symmetric matrix.

ENTRY DCHSLV(W,N,B, X) for solution B -> X

ENTRY DCHINV(W,N,V) for inversion

Parameters

in,out	*W*	symmetirc matrix
in	*N*	size
in	*AUX*	scratch array

Definition at line 226 of file Dbandmatrix.f90.

9.2 linesrch.f90 File Reference

Line search.

Data Types

- module linesrch

  Line search data.

Functions/Subroutines

- subroutine ptline (n, x, f, g, s, step, info)

  Perform linesearch.
- subroutine ptldef (gtole, stmax, minfe, maxfe)

  Initialize line search.
- subroutine ptlopt (nf, m, slopes, steps)

  Get details.
- subroutine ptlprt (lunp)

  Print line search data.

9.2.1 Detailed Description

Line search. Line search routine with sufficient decrease of slope.

In many minimization problems the objective function is close to quadratic, except far from the solution. Close to the minimum the behaviour may be almost quadratic or, due to round-off errors, it may have a non-smooth behaviour, which often complicates any further progress and the recognition of convergence. Round-off errors affect the function value, which may be large and small parameter changes result in small relative changes of the function value. Close to the minimum the gradient becomes small and the behaviour is not so much affected by Round-off errors.

```
      CALL PTLDEF(0.0,0.0, 0,0) ! init line search
      N=...
      X(.)=...
      D(.)=...
      ALPHA=1.0D0
10    F(X)=...
      G(X)=...
      IF(.) S(X)=..
      CALL PTLINE(N,X,F,G,D,ALPHA,INFO)
      IF(INFO.LT.0) GOTO 10
```

Definition in file linesrch.f90.

9.2.2 Function/Subroutine Documentation

9.2.2.1 subroutine ptldef (real(mps), intent(in) *gtole,* real(mps), intent(in) *stmax,* integer(mpi), intent(in) *minfe,* integer(mpi), intent(in) *maxfe*)

Initialize line search.

Parameters

in	*gtole*	slope ratio
in	*stmax*	total step limit
in	*minfe*	minimum number of evaluations

in	*maxfe*	maximum number of evaluations
		
		`                    --- range ----      default`
		`slope ratio       1.0E-4 ... 0.9        0.9`
		`min. F-calls      1 ... 2                1`
		`max. F-calls      2 ... 10               5`

Definition at line 213 of file linesrch.f90.

Referenced by xloopn().

9.2.2.2 subroutine ptline (integer(mpi), intent(in) *n,* real(mpd), dimension(n), intent(inout) *x,* real(mpd), intent(inout) *f,* real(mpd), dimension(n), intent(inout) *g,* real(mpd), dimension(n), intent(inout) *s,* real(mpd), intent(out) *step,* integer(mpi), intent(out) *info*)

Perform linesearch.

Parameters

in	*N*	dimension of problem
in,out	*X*	current iterate
in,out	*F*	associated function value
in,out	*G*	associated gradient
in,out	*S*	search vector
out	*STEP*	step factor (initially = 1.0)

out	*INFO*	information
		
		`= -1  repeat function evaluation`
		`=  0  input error (e.g. gradient not negative)`
		`=  1  convergence reached`
		`=  2  convergence assumed, but round-off errors`
		`=  3  too many function calls`
		`=  4  step factor ALPHA to small (ALPHA <= TOL)`

Definition at line 70 of file linesrch.f90.

Referenced by xloopn().

9.2.2.3 subroutine ptlopt (integer(mpi), intent(out) *nf,* integer(mpi), intent(out) *m,* real(mps), dimension(3), intent(out) *slopes,* real(mps), dimension(3), intent(out) *steps*)

Get details.

Parameters

out	*NF*	number of function values
out	*M*	index of function value with smallest slope
out	*SLOPES*	initial, current, smallest slope
out	*STEPS*	initial position, current, smallest step

Definition at line 239 of file linesrch.f90.

Referenced by ploopb(), and ploopc().

9.2.2.4 subroutine ptlprt (integer(mpi), intent(in) *lunp*)

Print line search data.

Parameters

in	*lunp*	unit number

Definition at line 275 of file linesrch.f90.

Referenced by xloopn().

9.3 Mille.cc File Reference

Write Millepede-II C-binary record.

```
#include "Mille.h"
#include <fstream>
#include <iostream>
```

9.3.1 Detailed Description

Write Millepede-II C-binary record. Create Millepede-II C-binary record.

Author

: Gero Flucke date : October 2006

Revision:

1.3

Date:

2007/04/16 17:47:38

(last update by

Author:

flucke

)

Definition in file Mille.cc.

9.4 mille.f90 File Reference

Write Millepede-II F-binary record.

Functions/Subroutines

- subroutine mille (nlc, derlc, ngl, dergl, label, rmeas, sigma)

  Add data block to record. Called from user code.

9.4.1 Detailed Description

Write Millepede-II F-binary record.

Definition in file mille.f90.

9.4.2 Function/Subroutine Documentation

9.4.2.1 subroutine mille (integer(mpi), intent(in) *nlc,* real(mps), dimension(nlc), intent(in) *derlc,* integer(mpi), intent(in) *ngl,* real(mps), dimension(ngl), intent(in) *dergl,* integer(mpi), dimension(ngl), intent(in) *label,* real(mps), intent(in) *rmeas,* real(mps), intent(in) *sigma*)

Add data block to record. Called from user code.

```
    CALL MILLE(...)       ! measured value, derivatives (one set)
    CALL ENDLE            ! complete, write record (many sets)
(or CALL KILLE            ! stop record)
```

The data transmitted by MILLE calls are collected in two arrays, a real array and an integer array, of same length. The collected data are written at the ENDLE call. The content of the arrays:

```
    real array              integer array
1   0.0                     error count (this record)
2   RMEAS, measured value   0                           JA
3   local derivative        index of local derivative
4   local derivative        index of local derivative
5    ...
6   SIGMA, error (>0)       0                           JB
    global derivative       label of global derivative
    global derivative       label of global derivative  IST
    RMEAS, measured value   0
    local derivative        index of local derivative
    local derivative        index of local derivative
    ...
    SIGMA, error            0
    global derivative       label of global derivative
    global derivative       label of global derivative
    ...
NR  global derivative       label of global derivative
```

The 0's in the integer array allow to recognize the start of a new set, the measured value and the error. The local and the global derivatives are inbetween, with a positive value in the integer array, the index of the local derivative or the label of the global derivative.

If more than one output unit is needed: duplicate this subroutine change the entry names to e.g. AMILLE, AENDLE, AKILLE and change the value of LUN and evtl. the dimension parameter in the parameter statements.

Parameters

in	*NLC*	number of local derivatives
in	*DERLC*	local derivatives
in	*NGL*	number of global derivatives
in	*DERGL*	global derivatives
in	*LABEL*	labels for global derivatives
in	*RMEAS*	measurement
in	*SIGMA*	error of measurement

Definition at line 65 of file mille.f90.

Referenced by mptest(), and mptst2().

9.5 Mille.h File Reference

Define class Mille.

```
#include <fstream>
```

Classes

- class Mille

  Class to write C binary file.

9.5.1 Detailed Description

Define class Mille.

Definition in file Mille.h.

9.6 minresDataModule.f90 File Reference

MINRES (data) definitions.

Data Types

- module minresdatamodule

  Defines real(kind=dp) and a few constants for use in other modules.

9.6.1 Detailed Description

MINRES (data) definitions.

Definition in file minresDataModule.f90.

9.7 minresModule.f90 File Reference

MINRES algorithm.

Data Types

- module minresmodule

  MINRES solves symmetric systems Ax = b or min ||Ax - b||_2, where the matrix A may be indefinite and/or singular.

9.7.1 Detailed Description

MINRES algorithm.

Definition in file minresModule.f90.

9.8 minresqlpBlasModule.f90 File Reference

MINRES-QLP BLAS subroutines.

Data Types

- module minresqlpblasmodule

9.8.1 Detailed Description

MINRES-QLP BLAS subroutines.

Definition in file minresqlpBlasModule.f90.

9.9 minresqlpDataModule.f90 File Reference

MINRES-QLP (data) definitions.

Data Types

- module minresqlpdatamodule

  Defines precision and range in real(kind=dp) and integer(kind=ip) for portability and a few constants for use in other modules.

9.9.1 Detailed Description

MINRES-QLP (data) definitions.

Definition in file minresqlpDataModule.f90.

9.10 minresqlpModule.f90 File Reference

MINRES-QLP algorithm.

Data Types

- module minresqlpmodule

  MINRESQLP solves symmetric systems Ax = b or min ||Ax - b||_2, where the matrix A may be indefinite and/or singular. "A" is really (A - shift∗I), where shift is an input real scalar.

9.10.1 Detailed Description

MINRES-QLP algorithm.

Definition in file minresqlpModule.f90.

9.11 mpbits.f90 File Reference

Bit field counters.

Data Types

- module mpbits

  Bit field data.

Functions/Subroutines

- subroutine inbits (im, jm, inc)

  Fill bit fields (counters).
- subroutine clbits (in, idimb, iencdb, jbfw)

  Calculate bit (field) array size, encoding.
- subroutine ndbits (ndims, ncmprs, nsparr, mnpair, ihst, jcmprs)

  Analyze bit fields.
- subroutine ckbits (ndims, mnpair, jcmprs)

  Check sparsity of matrix.
- subroutine spbits (nsparr, nsparc, ncmprs)

  Create sparsity information.

9.11.1 Detailed Description

Bit field counters. Count pairs of global parameters for sparse storage of global matrix, apply pair entries cut and build (compressed) sparsity structure (row offsets, column lists).

In sparse storage mode for each row the list of column indices with non zero elements (and those elements) are stored. With compression this list is represented by the first column and their number for continous regions (encoded in single INTEGER(mpi) words). Rare elements may be stored in single precision.

Definition in file mpbits.f90.

9.11.2 Function/Subroutine Documentation

9.11.2.1 subroutine ckbits (integer(mpl), dimension(4), intent(out) *ndims,* integer(mpi), intent(in) *mnpair,* integer(mpi), intent(in) *jcmprs*)

Check sparsity of matrix.

Parameters

out	*ndims*	(1): (reduced) size of bit array; (2): size of column lists; (3/4): number of (double/single precision) off diagonal elements;

in	*mnpair*	min. entries for pair
in	*jcmprs*	<>0: compress row information (column indices)

Definition at line 396 of file mpbits.f90.

Referenced by loop2().

9.11.2.2 subroutine clbits (integer(mpi), intent(in) *in,* integer(mpl), intent(out) *idimb,* integer(mpi), intent(out) *iencdb,* integer(mpi), intent(in) *jbfw*)

Calculate bit (field) array size, encoding.

Parameters

in	*in*	matrix size
out	*idimb*	dimension for bit (field) array
out	*iencdb*	number of bits for encoding column counter
in	*jbfw*	bit field width

Definition at line 113 of file mpbits.f90.

Referenced by loop2().

9.11.2.3 subroutine inbits (integer(mpi), intent(in) *im,* integer(mpi), intent(in) *jm,* integer(mpi), intent(in) *inc*)

Fill bit fields (counters).

Parameters

in	*im*	first index
in	*jm*	second index
in	*inc*	increment (usually 1)

Definition at line 36 of file mpbits.f90.

Referenced by loop2().

9.11.2.4 subroutine ndbits (integer(mpl), dimension(4), intent(out) *ndims,* integer(mpi), dimension(:), intent(out) *ncmprs,* integer(mpl), dimension(:,:), intent(out) *nsparr,* integer(mpi), intent(in) *mnpair,* integer(mpi), intent(in) *ihst,* integer(mpi), intent(in) *jcmprs*)

Analyze bit fields.

Parameters

out	*ndims*	(1): (reduced) size of bit array; (2): size of column lists; (3/4): number of (double/single precision) off diagonal elements;
out	*ncmprs*	compression info (per row)
out	*nsparr*	row offsets
in	*mnpair*	min. entries for pair
in	*ihst*	>0: histogram number
in	*jcmprs*	<>0: compress row information (column indices)

Definition at line 168 of file mpbits.f90.

Referenced by loop2().

9.11.2.5 subroutine spbits (integer(mpl), dimension(:,:), intent(in) *nsparr,* integer(mpi), dimension(:), intent(out) *nsparc,* integer(mpi), dimension(:), intent(in) *ncmprs*)

Create sparsity information.

Parameters

in	*nsparr*	row offsets
out	*nsparc*	column indices
in	*ncmprs*	compression info (per row)

Definition at line 495 of file mpbits.f90.

Referenced by loop2().

9.12 mpdalc.f90 File Reference

Dynamic memory management.

Data Types

- module mpdalc

  (De)Allocate vectors and arrays.
- interface mpdalc::mpalloc

  allocate array
- interface mpdalc::mpdealloc

  deallocate array

9.12.1 Detailed Description

Dynamic memory management.

Author

C. Kleinwort, DESY, 2012 (Claus.Kleinwort@desy.de)

Definition in file mpdalc.f90.

9.13 mpdef.f90 File Reference

Definitions.

Data Types

- module mpdef

  Definition of constants.
- type mpdef::listitem

  list items from steering file

9.13.1 Detailed Description

Definitions.

Definition in file mpdef.f90.

9.14 mphistab.f90 File Reference

Histogramming package.

Functions/Subroutines

- subroutine hmpdef (ih, xa, xb, text)
- subroutine hmpmak (inhist, fnhist, jnhist, xl, dl)
- subroutine bintab (tab, n, xa, xb)
- subroutine kprint (lun, list, n)
- subroutine gmpdef (ig, ityp, text)
- subroutine stmars
- subroutine rmesig (x, n, xloc, xsca)

9.14.1 Detailed Description

Histogramming package.

```
                    HMP... and GMP...
               Histogram and XY data in text files

  Booking:

  CALL HMPDEF(IH,XA,XB,TEXT)              CALL GMPDEF(IG,ITYP,TEXT)
  where                                   where
     IH    = 1 ... 10                        IG    = 1 ... 10
     XA,XB = left, right limit               ITYP  = 1 dots
     TEXT  = explanation                           = 2 line
                                                   = 3 dots and line
                                                   = 4 symbols
                                                   = 5 mean/sigma
                                             TEXT  = explanation

  CALL HMPLUN(LUNW)                       CALL GMPLUN(LUNW)
  unit for output                         unit for output

  CALL HMPENT(IH,X)                       CALL GMPXY(IG,X,Y)
  entry flt.pt. X                         add (X,Y) pair

                                          CALL GMPXYD(IG,X,Y,DX,DY)
                                          add (X,Y,DX,DY) ITYP=4

     new                                  CALL GMPMS(IG,X,Y)
                                          mean/sigma from x,y

  Booking log integer histogram:

  CALL HMPLDF(IH,TEXT)
  book and reset log integer histogram

  CALL HMPLNT(IH,IX)
  entry integer IX

  Printing and writing:

  CALL HMPRNT(IH)                         CALL GMPRNT(IG)
  print histogram IH or all, if 0         print data Ig or all, if 0

  CALL HMPWRT(IH)                         CALL GMPWRT(IG)
  write histogram IH or all to file       write data IG or all to file


  Storage manager for GMP...

   CALL STMARS                !! init/reset  storage manager
```

```
CALL STMAPR(JFLC,X,Y)        !! store pair (X,Y)

CALL STMADP(JFLC,FOUR)       !! store double pair

CALL STMACP(JFLC,ARRAY,N)    !! copy (cp) all pairs to array

CALL STMARM(JFLC)            !! remove (rm) stored paiirs
```

The number of histograms is limited to NUMHIS (=15), the number of XY data plots to NUMGXY (=10) and the storage of XY points to NDIM (=5000). As each XY plot can contain up to NLIMIT (=500) points (before averaging) NDIM should be NLIMIT∗NUMGXY.

Definition in file mphistab.f90.

9.14.2 Function/Subroutine Documentation

9.14.2.1 subroutine bintab (real(mps), dimension(n), intent(in) *tab,* integer(mpi), intent(in) *n,* real(mps), intent(out) *xa,* real(mps), intent(out) *xb*)

Definition at line 396 of file mphistab.f90.

References heapf().

Referenced by hmpmak().

9.14.2.2 subroutine gmpdef (integer(mpi), intent(in) *ig,* integer(mpi), intent(in) *ityp,* character (len=∗), intent(in) *text*)

Definition at line 562 of file mphistab.f90.

References rmesig(), and stmars().

Referenced by loopn(), mdiags(), and mptwo().

9.14.2.3 subroutine hmpdef (integer(mpi), intent(in) *ih,* real(mps), intent(in) *xa,* real(mps), intent(in) *xb,* character (len=∗), intent(in) *text*)

Definition at line 73 of file mphistab.f90.

References hmpmak(), kprint(), pfvert(), pivert(), and psvert().

Referenced by loop1(), loop2(), loopn(), mdiags(), and mptwo().

9.14.2.4 subroutine hmpmak (integer(mpi), dimension(120), intent(out) *inhist,* real(mps), dimension(120), intent(in) *fnhist,* integer(mpi), dimension(5), intent(inout) *jnhist,* real(mps), dimension(6), intent(inout) *xl,* real(mpd), dimension(2), intent(out) *dl*)

Definition at line 331 of file mphistab.f90.

References bintab(), and heapf().

Referenced by hmpdef().

9.14.2.5 subroutine kprint (integer(mpi), intent(inout) *lun,* integer(mpi), dimension(n), intent(in) *list,* integer(mpi), intent(in) *n*)

Definition at line 500 of file mphistab.f90.

Referenced by hmpdef().

9.14.2.6 subroutine rmesig (real(mps), dimension(n), intent(inout) *x,* integer(mpi), intent(in) *n,* real(mps), intent(out) *xloc,* real(mps), intent(out) *xsca*)

Definition at line 993 of file mphistab.f90.

References heapf().

Referenced by gmpdef().

9.14.2.7 subroutine stmars ( )

Definition at line 891 of file mphistab.f90.

Referenced by gmpdef().

9.15 mpmanvb.f90 File Reference

Draft Manual by V. Blobel.

9.15.1 Detailed Description

Draft Manual by V. Blobel.

Definition in file mpmanvb.f90.

9.16 mpmod.f90 File Reference

Data structures.

Data Types

- module mpmod

  Parameters, variables, dynamic arrays.

9.16.1 Detailed Description

Data structures.

Definition in file mpmod.f90.

9.17 mpnum.f90 File Reference

General linear algebra routines.

Functions/Subroutines

- subroutine sqminv (v, b, n, nrank, diag, next)

  Matrix inversion and solution.
- subroutine sqminl (v, b, n, nrank, diag, next)

  Matrix inversion for LARGE matrices.
- subroutine devrot (n, diag, u, v, work, iwork)

Diagonalization.

- subroutine devsig (n, diag, u, b, coef)

  Calculate significances.
- subroutine devsol (n, diag, u, b, x, work)

  Solution by diagonalization.
- subroutine devinv (n, diag, u, v)

  Inversion by diagonalization. Get inverse matrix V from DIAG and U.
- subroutine choldc (g, n)

  Cholesky decomposition.
- subroutine cholsl (g, x, n)

  Solution after decomposition.
- subroutine cholin (g, v, n)

  Inversion after decomposition.
- subroutine vabdec (n, val, ilptr)

  Variable band matrix decomposition.
- subroutine vabmmm (n, val, ilptr)

  Variable band matrix print minimum and maximum.
- subroutine vabslv (n, val, ilptr, x)

  Variable band matrix solution.
- REAL(mpd) function dbdot (n, dx, dy)

  Dot product.
- subroutine dbaxpy (n, da, dx, dy)

  Multiply, addition.
- subroutine dbsvx (v, a, b, n)

  Product symmetric matrix, vector.
- subroutine dbsvxl (v, a, b, n)

  Product LARGE symmetric matrix, vector.
- subroutine dbgax (a, x, y, m, n)

  Multiply general M-by-N matrix A and N-vector X.
- subroutine dbavat (v, a, w, n, ms)

  A V AT product (similarity).
- subroutine dbmprv (lun, x, v, n)

  Print symmetric matrix, vector.
- subroutine dbprv (lun, v, n)

  Print symmetric matrix.
- subroutine heapf (a, n)

  Heap sort direct (real).
- subroutine sort1k (a, n)

  Quick sort 1.
- subroutine sort2k (a, n)

  Quick sort 2.
- REAL(mps) function chindl (n, nd)

  Chi2/ndf cuts.
- subroutine lltdec (n, c, india, nrkd)

  LLT decomposition.
- subroutine lltfwd (n, c, india, x)

  Forward solution.
- subroutine lltbwd (n, c, india, x)

  Backward solution.
- subroutine equdec (n, m, c, india, nrkd, nrkd2)

  Decomposition of equilibrium systems.

- subroutine equslv (n, m, c, india, x)

  Solution of equilibrium systems (after decomposition).
- subroutine precon (p, n, c, cu, a, s)

  Constrained preconditioner, decomposition.
- subroutine presol (p, n, cu, a, s, x, y)

  Constrained preconditioner, solution.
- subroutine sqmibb (v, b, n, nbdr, nbnd, inv, nrank, vbnd, vbdr, aux, vbk, vzru, scdiag, scflag)

  Bordered band matrix.

9.17.1 Detailed Description

General linear algebra routines. ***** Collection of utility routines from V. Blobel *****

```
V. Blobel, Univ. Hamburg
Numerical subprograms used in MP-II: matrix equations,
   and matrix products, double precision

Solution by inversion
   SQMINV
   SQMINL  for LARGE matrix, use OpenMP (CHK)

Solution by diagonalization
   DEVROT, DEVPRT, DEFSOL,DEVINV

Solution by Cholesky decomposition of symmetric matrix
   CHOLDC

Solution by Cholesky decomposition of variable-band matrix
   VABDEC

Solution by Cholesky decomposition of bordered band matrix
   SQMIBB  (CHK)

Matrix/vector products
   DBDOT     dot vector product
   DBAXPY    multiplication and addition
   DBSVX     symmetric matrix vector
   DBSVX     LARGE symmetric matrix vector (CHK)
   DBGAX     general matrix vector
   DBAVAT    AVAT product
   DBMPRV    print parameter and matrix
   DBPRV     print matrix  (CHK)

Chi^2 cut values
   CHINDL

Accurate summation (moved to pede.f90)
   ADDSUM

Sorting
   HEAPF    heap sort reals direct
   SORT1K   sort 1-dim key-array (CHK)
   SORT2K   sort 2-dim key-array
```

Definition in file mpnum.f90.

9.17.2 Function/Subroutine Documentation

9.17.2.1 REAL(mps) function chindl (integer(mpi), intent(in) *n*, integer(mpi), intent(in) *nd*)

Chi2/ndf cuts.

Return limit in Chi^2/ndf for N sigmas (N=1, 2 or 3).

Parameters

in	*n*	number of sigmas
in	*nd*	ndf

Returns

Chi2/ndf cut value

Definition at line 1677 of file mpnum.f90.

Referenced by loop2(), and loopbf().

9.17.2.2 subroutine choldc (real(mpd), dimension(∗), intent(inout) *g,* integer(mpi), intent(in) *n*)

Cholesky decomposition.

Cholesky decomposition of the matrix G: G = L D L^T

- G = symmetric matrix, in symmetric storage mode
- L = unit triangular matrix (1's on diagonal)
- D = diagonal matrix (elements store on diagonal of L)

The sqrts of the usual Cholesky decomposition are avoided by D. Matrices L and D are stored in the place of matrix G; after the decomposition, the solution of matrix equations and the computation of the inverse of the (original) matrix G are done by CHOLSL and CHOLIN.

Parameters

in,out	*g*	symmetric matrix, replaced by D,L
in	*n*	size of matrix

Definition at line 729 of file mpnum.f90.

9.17.2.3 subroutine cholin (real(mpd), dimension(∗), intent(in) *g,* real(mpd), dimension(∗), intent(out) *v,* integer(mpi), intent(in) *n*)

Inversion after decomposition.

The inverse of the (original) matrix G is computed and stored in symmetric storage mode in matrix V. Arrays G and V must be different arrays.

Parameters

in	*g*	decomposed symmetric matrix
in,out	*v*	inverse matrix
in	*n*	size of matrix

Definition at line 821 of file mpnum.f90.

9.17.2.4 subroutine cholsl (real(mpd), dimension(∗), intent(in) *g,* real(mpd), dimension(n), intent(inout) *x,* integer(mpi), intent(in) *n*)

Solution after decomposition.

The matrix equation G X = B is solved for X, where the matrix G in the argument is already decomposed by CHOLDC. The vector B is called X in the argument and the content is replaced by the resulting vector X.

Parameters

in	*g*	decomposed symmetric matrix
in,out	*x*	r.h.s vector B, replaced by solution vector X
in	*n*	size of matrix

Definition at line 775 of file mpnum.f90.

9.17.2.5 subroutine dbavat (real(mpd), dimension(∗), intent(in) *v,* real(mpd), dimension(∗), intent(in) *a,* real(mpd), dimension(∗), intent(inout) *w,* integer(mpi), intent(in) *n,* integer(mpi), intent(in) *ms*)

A V AT product (similarity).

Multiply symmetric N-by-N matrix from the left with general M-by-N matrix and from the right with the transposed of the same general matrix to form symmetric M-by-M matrix (used for error propagation).

Parameters

in	*V*	symmetric N-by-N matrix
in	*A*	general M-by-N matrix
in,out	*W*	symmetric M-by-M matrix
in	*MS*	rows of A (-rows: don't reset W)
in	*N*	columns of A

Definition at line 1252 of file mpnum.f90.

Referenced by loopbf().

9.17.2.6 subroutine dbaxpy (integer(mpi), intent(in) *n,* real(mpd), intent(in) *da,* real(mpd), dimension(∗), intent(in) *dx,* real(mpd), dimension(∗), intent(inout) *dy*)

Multiply, addition.

Constant times vector added to a vector: DY:=DY+DA∗DX

Parameters

in	*n*	vector size
in	*dx*	vector
in,out	*dy*	vector
in	*da*	scalar

Definition at line 1097 of file mpnum.f90.

9.17.2.7 REAL(mpd) function dbdot (integer(mpi), intent(in) *n,* real(mpd), dimension(∗), intent(in) *dx,* real(mpd), dimension(∗), intent(in) *dy*)

Dot product.

Parameters

in	*n*	vector size
in	*dx*	vector
in	*dy*	vector

Returns

dot product dx∗dy

Definition at line 1066 of file mpnum.f90.

Referenced by xloopn().

9.17.2.8 subroutine dbgax (real(mpd), dimension(∗), intent(in) *a,* real(mpd), dimension(∗), intent(in) *x,* real(mpd), dimension(∗), intent(out) *y,* integer(mpi), intent(in) *m,* integer(mpi), intent(in) *n*)

Multiply general M-by-N matrix A and N-vector X.

Parameters

in	*A*	general M-by-N matrix (A11 A12 ... A1N A21 A22 ...)
in	*X*	N vector
out	*Y*	= M vector
in	*M*	rows of A
in	*N*	columns of A

Definition at line 1215 of file mpnum.f90.

9.17.2.9 subroutine dbmprv (integer(mpi), intent(in) *lun,* real(mpd), dimension(∗), intent(in) *x,* real(mpd), dimension(∗), intent(in) *v,* integer(mpi), intent(in) *n*)

Print symmetric matrix, vector.

Prints the n-vector X and the symmetric N-by-N covariance matrix V, the latter as a correlation matrix.

Parameters

in	*lun*	unit number
in	*x*	vector
in	*v*	symmetric matrix
in	*n*	size of matrix, vector

Definition at line 1324 of file mpnum.f90.

9.17.2.10 subroutine dbprv (integer(mpi), intent(in) *lun,* real(mpd), dimension(∗), intent(in) *v,* integer(mpi), intent(in) *n*)

Print symmetric matrix.

Prints the symmetric N-by-N matrix V.

Parameters

in	*lun*	unit number
in	*v*	symmetric matrix
in	*n*	size of matrix,

Definition at line 1399 of file mpnum.f90.

9.17.2.11 subroutine dbsvx (real(mpd), dimension(∗), intent(in) *v,* real(mpd), dimension(∗), intent(in) *a,* real(mpd), dimension(∗), intent(out) *b,* integer(mpi), intent(in) *n*)

Product symmetric matrix, vector.

Multiply symmetric N-by-N matrix and N-vector.

Parameters

in	*v*	symmetric matrix
in	*a*	vector
out	*b*	vector B = V ∗ A

in	*n*	size of matrix

Definition at line 1128 of file mpnum.f90.

Referenced by feasib().

9.17.2.12 subroutine dbsvxl (real(mpd), dimension(∗), intent(in) *v,* real(mpd), dimension(∗), intent(in) *a,* real(mpd), dimension(∗), intent(out) *b,* integer(mpi), intent(in) *n*)

Product LARGE symmetric matrix, vector.

Multiply LARGE symmetric N-by-N matrix and N-vector:

Parameters

in	*v*	symmetric matrix
in	*a*	vector
out	*b*	product vector B = V ∗ A
in	*n*	size of matrix

Definition at line 1172 of file mpnum.f90.

Referenced by minver().

9.17.2.13 subroutine devinv (integer(mpi), intent(in) *n,* real(mpd), dimension(n), intent(in) *diag,* real(mpd), dimension(n,n), intent(in) *u,* real(mpd), dimension(∗), intent(out) *v*)

Inversion by diagonalization. Get inverse matrix V from DIAG and U.

Parameters

in	*N*	size of matrix
in	*DIAG*	diagonal elements
in	*U*	transformation matrix
out	*V*	smmmetric matrix

Definition at line 680 of file mpnum.f90.

Referenced by zdiags().

9.17.2.14 subroutine devrot (integer(mpi), intent(in) *n,* real(mpd), dimension(n), intent(out) *diag,* real(mpd), dimension(n,n), intent(out) *u,* real(mpd), dimension(∗), intent(in) *v,* real(mpd), dimension(n), intent(out) *work,* integer(mpi), dimension(n), intent(out) *iwork*)

Diagonalization.

Determination of eigenvalues and eigenvectors of symmetric matrix V by Householder method

Parameters

in	*n*	size of matrix
out	*diag*	diagonal elements
out	*u*	transformation matrix
in	*v*	symmetric matrix, unchanged
out	*work*	work array
out	*iwork*	work array

Definition at line 353 of file mpnum.f90.

References peend().

Referenced by mdiags().

9.17.2.15 subroutine devsig (integer(mpi), intent(in) *n,* real(mpd), dimension(n), intent(in) *diag,* real(mpd), dimension(n,n), intent(in) *u,* real(mpd), dimension(n), intent(in) *b,* real(mpd), dimension(n), intent(out) *coef*)

Calculate significances.

Definition at line 595 of file mpnum.f90.

Referenced by mdiags().

9.17.2.16 subroutine devsol (integer(mpi), intent(in) *n,* real(mpd), dimension(n), intent(in) *diag,* real(mpd), dimension(n,n), intent(in) *u,* real(mpd), dimension(n), intent(in) *b,* real(mpd), dimension(n), intent(out) *x,* real(mpd), dimension(n), intent(out) *work*)

Solution by diagonalization.

Solution of matrix equation V ∗ X = B after diagonalization of V.

Parameters

in	*N*	size of matrix
in	*DIAG*	diagonal elements
in	*U*	transformation matrix
in	*B*	r.h.s. of matrix equation (unchanged)
out	*X*	solution vector
out	*WORK*	work array

Definition at line 633 of file mpnum.f90.

Referenced by mdiags().

9.17.2.17 subroutine equdec (integer(mpi), intent(in) *n,* integer(mpi), intent(in) *m,* real(mpd), dimension(∗), intent(inout) *c,* integer(mpi), dimension(n+m), intent(inout) *india,* integer(mpi), intent(out) *nrkd,* integer(mpi), intent(out) *nrkd2*)

Decomposition of equilibrium systems.

```
N x N matrix C:     starting with sym.pos.def. matrix (N)
length  of array C: INDIA(N) + N*M + (M*M+M)/2
Content of array C: band matrix, as described by INDIA(1)...INDIA(N)
       followed by: NxM elements of constraint matrix A
       followed by: (M*M+M)/2 unused elements
                    INDIA(N+1)...INDIA(N+M) defined internally
```

Parameters

in	*n*	size of symmetric matrix
in	*m*	number of constrains
in,out	*c*	combined variable-band + constraints matrix, replaced by decomposition
in,out	*india*	pointer array
out	*nrkd*	removed components
out	*nrkd2*	removed components

Definition at line 1895 of file mpnum.f90.

References lltdec(), and lltfwd().

Referenced by mminrs(), and mminrsqlp().

9.17.2.18 subroutine equslv (integer(mpi), intent(in) *n,* integer(mpi), intent(in) *m,* real(mpd), dimension(∗), intent(in) *c,* integer(mpi), dimension(n+m), intent(in) *india,* real(mpd), dimension(n+m), intent(inout) *x*)

Solution of equilibrium systems (after decomposition).

```
N x N matrix C:     starting with sym.pos.def. matrix (N)
length  of array C: INDIA(N) + N*M + (M*M+M)/2
Content of array C: band matrix, as described by INDIA(1)...INDIA(N)
       followed by: NxM elements of constraint matrix A
       followed by: (M*M+M)/2 unused elements
                    INDIA(N+1)...INDIA(N+M) defined internally
```

Parameters

in	*n*	size of symmetric matrix
in	*m*	number of constrains
in	*c*	decomposed combined variable-band + constraints matrix
in	*india*	pointer array
in,out	*x*	r.h.s vector B, replaced by solution vector X

Definition at line 1958 of file mpnum.f90.

References lltbwd(), and lltfwd().

Referenced by mvsolv().

9.17.2.19 subroutine heapf (real(mps), dimension(∗), intent(inout) *a,* integer(mpi), intent(in) *n*)

Heap sort direct (real).

Real keys A(∗), sorted at return.

Parameters

in,out	*a*	array of keys
in	*n*	number of keys

Definition at line 1443 of file mpnum.f90.

Referenced by bintab(), hmpmak(), and rmesig().

9.17.2.20 subroutine lltbwd (integer(mpi), intent(in) *n,* real(mpd), dimension(∗), intent(in) *c,* integer(mpi), dimension(n), intent(in) *india,* real(mpd), dimension(n), intent(inout) *x*)

Backward solution.

The matrix equation A X = B is solved by forward + backward solution. The matrix is assumed to decomposed before using LLTDEC. The array X(N) contains on entry the right-hand-side B(N); at return it contains the solution.

Parameters

in	*n*	size of matrix
in,out	*c*	decomposed variable-band matrix
in	*india*	pointer array
in,out	*x*	r.h.s vector B, replaced by solution vector X

Definition at line 1858 of file mpnum.f90.

Referenced by equslv().

9.17.2.21 subroutine lltdec (integer(mpi), intent(in) *n,* real(mpd), dimension(∗), intent(inout) *c,* integer(mpi), dimension(n), intent(in) *india,* integer(mpi), intent(out) *nrkd*)

LLT decomposition.

Decomposition: C = L L^T.

Variable-band matrix row-Doolittle decomposition of pos. def. matrix. A variable-band NxN symmetric matrix, is stored row by row in the array C(.). For each row all coefficients from the first non-zero element in the row to the

diagonal is stored. The pointer array INDIA(N) contains the indices in C(.) of the diagonal elements. INDIA(1) is always 1, and INDIA(N) is equal to the total number of coefficients stored, called the profile. The form of a variable-band matrix is preserved in the L D L^T decomposition. No fill-in is created ahead in any row or ahead of the first entry in any column, but existing zero-values will become non-zero. The decomposition is done "in-place".

- NRKD = 0 no component removed
- NRKD $<$ 0 1 component removed, negative index
- NRKD $>$ 1 number of

The matrix C is assumed to be positive definite, e.g. from the normal equations of least squares. The (positive) diagonal elements are reduced during decomposition. If a diagonal element is reduced by about a word length (see line "test for linear dependence"), then the pivot is assumed as zero and the entire row/column is reset to zero, removing the corresponding element from the solution.

Parameters

in	*n*	size of matrix
in,out	*c*	variable-band matrix, replaced by L
in	*india*	pointer array
out	*nrkd*	removed components

Definition at line 1750 of file mpnum.f90.

Referenced by equdec().

9.17.2.22 subroutine lltfwd (integer(mpi), intent(in) *n,* real(mpd), dimension(∗), intent(in) *c,* integer(mpi), dimension(n), intent(in) *india,* real(mpd), dimension(n), intent(inout) *x*)

Forward solution.

The matrix equation A X = B is solved by forward + backward solution. The matrix is assumed to decomposed before using LLTDEC. The array X(N) contains on entry the right-hand-side B(N); at return it contains the solution.

Parameters

in	*n*	size of matrix
in,out	*c*	decomposed variable-band matrix
in	*india*	pointer array
in,out	*x*	r.h.s vector B, replaced by solution vector X

Definition at line 1824 of file mpnum.f90.

Referenced by equdec(), and equslv().

9.17.2.23 subroutine precon (integer(mpi), intent(in) *p,* integer(mpi), intent(in) *n,* real(mpd), dimension(n), intent(in) *c,* real(mpd), dimension(n), intent(out) *cu,* real(mpd), dimension(n,p), intent(inout) *a,* real(mpd), dimension((p∗p+p)/2), intent(out) *s*)

Constrained preconditioner, decomposition.

```
Constrained preconditioner, e.g for GMRES solution:

                                                 intermediate
   (            )  (   )      (   )              (   )
   (   C    A^T )  ( x )   =  ( y )              ( u )
   (            )  (   )      (   )              (   )
   (   A     0  )  ( l )      ( d )              ( v )

input:
   C(N) is diagonal matrix and remains unchanged
        may be identical to CU(N), then it is changed
```

```
   A(N,P) is modified
   Y(N+P) is rhs vector, unchanged
        may be identical to X(N), then it is changed

result:
   CU(N) is 1/sqrt of diagonal matrix C(N)
   X(N+P) is result vector
   S((P*P+P)/2) is Cholesky decomposed symmetric (P,P) matrix
```

Parameters

in	*p*	number of constraints
in	*n*	size of diagonal matrix
in	*c*	diagonal matrix (changed if c=cu as actual parameters)
out	*cu*	1/sqrt(c)
in,out	*a*	constraint matrix (size n∗p), modified
out	*s*	Cholesky decomposed symmetric (P,P) matrix

Definition at line 2022 of file mpnum.f90.

Referenced by minresmodule::minres(), mminrs(), and mminrsqlp().

9.17.2.24 subroutine presol (integer(mpi), intent(in) *p,* integer(mpi), intent(in) *n,* real(mpd), dimension(n), intent(in) *cu,* real(mpd), dimension(n,p), intent(in) *a,* real(mpd), dimension((p∗p+p)/2), intent(in) *s,* real(mpd), dimension(n+p), intent(out) *x,* real(mpd), dimension(n+p), intent(in) *y*)

Constrained preconditioner, solution.

Parameters

in	*p*	number of constraints
in	*n*	size of diagonal matrix
in	*cu*	1/sqrt(c)
in	*a*	modified constraint matrix (size n∗p)
in	*s*	Cholesky decomposed symmetric (P,P) matrix
out	*x*	result vector
in	*y*	rhs vector (changed if x=y as actual parameters)

Definition at line 2093 of file mpnum.f90.

Referenced by mcsolv().

9.17.2.25 subroutine sort1k (integer(mpi), dimension(∗), intent(inout) *a,* integer(mpi), intent(in) *n*)

Quick sort 1.

Quick sort of A(1,N) integer.

Parameters

in,out	*a*	vector of integers, sorted at return
in	*n*	size of vector

Definition at line 1498 of file mpnum.f90.

References peend().

Referenced by loop2(), and loopbf().

9.17.2.26 subroutine sort2k (integer(mpi), dimension(2,∗), intent(inout) *a,* integer(mpi), intent(in) *n*)

Quick sort 2.

Quick sort of A(2,N) integer.

Parameters

in,out	*a*	vector (pair) of integers, sorted at return
in	*n*	size of vector

Definition at line 1583 of file mpnum.f90.

References peend().

Referenced by peread(), and upone().

9.17.2.27 subroutine sqmibb (real(mpd), dimension(∗), intent(inout) *v,* real(mpd), dimension(n), intent(out) *b,* integer(mpi), intent(in) *n,* integer(mpi), intent(in) *nbdr,* integer(mpi), intent(in) *nbnd,* integer(mpi), intent(in) *inv,* integer(mpi), intent(out) *nrank,* real(mpd), dimension(n∗(nbnd+1)), intent(out) *vbnd,* real(mpd), dimension(n∗nbdr), intent(out) *vbdr,* real(mpd), dimension(n∗nbdr), intent(out) *aux,* real(mpd), dimension((nbdr∗nbdr+nbdr)/2), intent(out) *vbk,* real(mpd), dimension(nbdr), intent(out) *vzru,* real(mpd), dimension(nbdr), intent(out) *scdiag,* integer(mpi), dimension(nbdr), intent(out) *scflag*)

Bordered band matrix.

Obtain solution of a system of linear equations with symmetric bordered band matrix (V ∗ X = B), on request inverse is calculated. For band part root-free Cholesky decomposition and forward/backward substitution is used.

Use decomposition in border and band part for block matrix algebra:

```
| A  Ct |   | x1 |   | b1 |        , A  is the border part
|       | * |    | = |    |        , Ct is the mixed part
| C  D  |   | x2 |   | b2 |        , D  is the band part
```

Explicit inversion of D is avoided by using solution X of D∗X=C (X=D^-1∗C, obtained from Cholesky decomposition and forward/backward substitution)

```
| x1 |   | E*b1 - E*Xt*b2 |        , E^-1 = A-Ct*D^-1*C = A-Ct*X
|    | = |                |
| x2 |   |  x   - X*x1    |        , x is solution of D*x=b2 (x=D^-1*b2)
```

Inverse matrix is:

```
|  E   -E*Xt          |
|                     |            , only band part of (D^-1 + X*E*Xt)
| -X*E  D^-1 + X*E*Xt |              is calculated for inv=1
```

Parameters

in,out	*v*	symmetric N-by-N matrix in symmetric storage mode (V(1) = V11, V(2) = V12, V(3) = V22, V(4) = V13, ...), replaced by inverse matrix
in,out	*b*	N-vector, replaced by solution vector
in	*n*	size of V, B
in	*nbdr*	border size
in	*nbnd*	band width
in	*inv*	=1 calculate band part of inverse (for pulls), >1 calculate complete inverse
out	*nrank*	rank of matrix V
out	*vbnd*	band part of V
out	*vbdr*	border part of V
out	*aux*	solutions for border rows
out	*vbk*	matrix for border solution
out	*vzru*	border solution
out	*scdiag*	workspace (D)
out	*scflag*	workspace (I)

Definition at line 2202 of file mpnum.f90.

References dbcdec(), and sqminv().

Referenced by loopbf().

9.17.2.28 subroutine sqminl (real(mpd), dimension(∗), intent(inout) *v,* real(mpd), dimension(n), intent(out) *b,* integer(mpi), intent(in) *n,* integer(mpi), intent(out) *nrank,* real(mpd), dimension(n), intent(out) *diag,* integer(mpi), dimension(n), intent(out) *next*)

Matrix inversion for LARGE matrices.

Like SQMINV, additional parallelization with OpenMP.

Parameters

in,out	*V*	symmetric N-by-N matrix in symmetric storage mode (V(1) = V11, V(2) = V12, V(3) = V22, V(4) = V13, ...), replaced by inverse matrix
in,out	*B*	N-vector, replaced by solution vector
in	*N*	size of V, B
out	*NRANK*	rank of matrix V
out	*DIAG*	double precision scratch array
out	*NEXT*	integer aux array

Definition at line 200 of file mpnum.f90.

Referenced by minver().

9.17.2.29 subroutine sqminv (real(mpd), dimension(∗), intent(inout) *v,* real(mpd), dimension(n), intent(out) *b,* integer(mpi), intent(in) *n,* integer(mpi), intent(out) *nrank,* real(mpd), dimension(n), intent(out) *diag,* integer(mpi), dimension(n), intent(out) *next*)

Matrix inversion and solution.

Obtain solution of a system of linear equations with symmetric matrix (V ∗ X = B) and the inverse.

Method of solution is by elimination selecting the pivot on the diagonal each stage. The rank of the matrix is returned in NRANK. For NRANK ne N, all remaining rows and cols of the resulting matrix V and the corresponding elements of B are set to zero.

Parameters

in,out	*V*	symmetric N-by-N matrix in symmetric storage mode (V(1) = V11, V(2) = V12, V(3) = V22, V(4) = V13, ...), replaced by inverse matrix
in,out	*B*	N-vector, replaced by solution vector
in	*N*	size of V, B
out	*NRANK*	rank of matrix V
out	*DIAG*	double precision scratch array
out	*NEXT*	INTEGER(mpi) aux array

Definition at line 72 of file mpnum.f90.

Referenced by feasma(), loopbf(), and sqmibb().

9.17.2.30 subroutine vabdec (integer(mpi), intent(in) *n,* real(mpd), dimension(∗), intent(inout) *val,* integer(mpi), dimension(n), intent(in) *ilptr*)

Variable band matrix decomposition.

Decomposition: A = L D L^T

Variable-band matrix row Doolittle decomposition. A variable-band NxN symmetric matrix, also called skyline, is stored row by row in the array VAL(.). For each row every coefficient between the first non-zero element in the row and the diagonal is stored. The pointer array ILPTR(N) contains the indices in VAL(.) of the diagonal elements. ILPTR(1) is always 1, and ILPTR(N) is equal to the total number of coefficients stored, called the profile. The form of a variable-band matrix is preserved in the L D L^T decomposition no fill-in is created ahead in any row or ahead of the first entry in any column, but existing zero-values will become non-zero. The decomposition is done "in-place".

Parameters

in	*n*	size of matrix
in,out	*val*	variable-band matrix, replaced by D,L
in	*ilptr*	pointer array

Definition at line 876 of file mpnum.f90.

9.17.2.31 subroutine vabmmm (integer(mpi), intent(in) *n,* real(mpd), dimension(∗), intent(inout) *val,* integer(mpi), dimension(n), intent(in) *ilptr*)

Variable band matrix print minimum and maximum.

Parameters

in	*n*	size of matrix
in,out	*val*	variable-band matrix, replaced by D,L
in	*ilptr*	pointer array

Definition at line 988 of file mpnum.f90.

9.17.2.32 subroutine vabslv (integer(mpi), intent(in) *n,* real(mpd), dimension(∗), intent(inout) *val,* integer(mpi), dimension(n), intent(in) *ilptr,* real(mpd), dimension(n), intent(inout) *x*)

Variable band matrix solution.

The matrix equation A X = B is solved. The matrix is assumed to decomposed before using VABDEC. The array X(N) contains on entry the right-hand-side B(N); at return it contains the solution.

Parameters

in	*n*	size of matrix
in,out	*val*	decomposed variable-band matrix
in	*ilptr*	pointer array
in,out	*x*	r.h.s vector B, replaced by solution vector X

Definition at line 1025 of file mpnum.f90.

9.18 mptest1.f90 File Reference

MC for simple 100 plane chamber.

Data Types

- module mptest1

  Parameters and data.

Functions/Subroutines

- subroutine mptest

  Generate test files.
- subroutine genlin (ip)

  Generate line and measurements.

9.18.1 Detailed Description

MC for simple 100 plane chamber. No B-field, straight tracks. Selected with command line option '-t'.

Global parameters:

- Position offsets in measurement direction (alignment).
- Relative drift velocity corrections (calibration).

Definition in file mptest1.f90.

9.18.2 Function/Subroutine Documentation

9.18.2.1 subroutine genlin (integer(mpi), intent(in) *ip*)

Generate line and measurements.

Parameters

in	*ip*	print flag

Definition at line 286 of file mptest1.f90.

References gran(), and uran().

Referenced by mptest().

9.18.2.2 subroutine mptest ( )

Generate test files.

Create text and binary files.

```
unit  8: textfile mp2str.txt   = steering file
unit  9: textfile mp2con.txt   = constraint file
unit 51: binary file mp2test.bin, written using CALL MILLE(.)
existing file are removed
```

Definition at line 59 of file mptest1.f90.

References genlin(), gran(), and mille().

Referenced by filetc().

9.19 mptest2.f90 File Reference

MC for simple 10 layer silicon strip tracker.

Data Types

- module mptest2

  Parameters and data.

Functions/Subroutines

- subroutine mptst2 (imodel)

  Generate test files.
- subroutine genln2 (ip)

  Generate line and measurements.

9.19.1 Detailed Description

MC for simple 10 layer silicon strip tracker. No B-field, straight tracks. Selected with command line option '-t=track-model' The *track-models* differ in the implementation of multiple scattering (errors):

- `SL0`: Ignore multiple scattering. Fit 4 track parameters.
- `SLE`: Ignore correlations due to multiple scattering, use only diagonal of m.s. covariance matrix. Fit 4 track parameters.
- `BP`: Intoduce explicit scattering angles at each scatterer. Fit 4+2∗(nmlyr-2) parameters. Matrix of corresponding linear equation system is full and solution is obtained by inversion (time ∼ parameters^3).
- `BRLF`: Use (fine) broken lines (see References). Multiple scattering kinks are described by triplets of offsets at scatterers as track parameters. Fit 4+2∗(nmlyr-2) parameters. Matrix of corresponding linear equation system has band structure and solution is obtained by root-free Cholesky decomposition (time ∼ parameters).
- `BRLC`: Use (coarse) broken lines. Similar to `BRLF`, but with stereo layers combined into single layer/scatterer. Fit 4+2∗(nlyr-2) parameters.

MC for simple silicon strip tracker:

- 10 silicon detector layers
- 50 modules per layer (1∗2cm)
- 10 cm spacing, no B-field
- layers 1,4,7,10 have additional +/-5deg stereo modules
- intrinsic resolution 20mu, 2% X0 per strip module
- uniform track offsets/slopes
- momentum: log10(p) 10..100 GeV uniform

Global parameters:

- Position offsets (2D) in measurement plane per module (alignment).

Definition in file mptest2.f90.

9.19.2 Function/Subroutine Documentation

9.19.2.1 subroutine genln2 (integer(mpi), intent(in) *ip*)

Generate line and measurements.

Parameters

in	*ip*	print flag

Definition at line 405 of file mptest2.f90.

References gran(), and uran().

Referenced by mptst2().

9.19.2.2 subroutine mptst2 (integer(mpi), intent(in) *imodel*)

Generate test files.

Create text and binary files.

```
unit  8: textfile mp2str.txt   = steering file
unit  9: textfile mp2con.txt   = constraint file
unit 51: binary file mp2test.bin, written using CALL MILLE(.)
existing file are removed
```

Parameters

in	*imodel*	track model 0: 'straight line', ignoring multiple scattering 1: 'straight line', using diagonal of m.s. error matrix 2: 'break points' 3: 'broken lines', fine 4: 'broken lines', coarse (stereo layers combined)

Definition at line 92 of file mptest2.f90.

References genln2(), gran(), mille(), and uran().

Referenced by filetc().

9.20 mptext.f90 File Reference

Analyse text string.

Data Types

- module mptext
  Keyword position.

Functions/Subroutines

- subroutine ratext (text, nums, dnum)
  Translate text.
- subroutine rltext (text, ia, ib, nab)
  Analyse text range.
- INTEGER(mpi) function matint (pat, text, npat, ntext)
  Approximate string matching.

9.20.1 Detailed Description

Analyse text string.

Definition in file mptext.f90.

9.20.2 Function/Subroutine Documentation

9.20.2.1 INTEGER(mpi) function matint (character (len=*), intent(in) *pat,* character (len=*), intent(in) *text,* integer(mpi), intent(out) *npat,* integer(mpi), intent(out) *ntext*)

Approximate string matching.

Approximate string matching - case insensitive. Return number of matches of string PAT in string TEXT, and number NPAT, NTEXT of characters of string PAT and string TEXT. Strings are considered from first to last non-blank character.

Example:

```
MATCH = MATINT(' keYs ','keyWO RD',NPAT,NTEXT)
returns MATCH=3, NPAT=4, NTEXT=8
```

Parameters

in	*pat*	pattern
in	*text*	text
out	*npat*	number of characters in pattern
out	*ntext*	number of characters in text

Returns

number of matching characters of pattern in text

Definition at line 269 of file mptext.f90.

References peend().

Referenced by filetc(), filetx(), intext(), and nufile().

9.20.2.2 subroutine ratext (character (len=∗), intent(in) *text,* integer(mpi), intent(out) *nums,* real(mpd), dimension(∗), intent(out) *dnum*)

Translate text.

Translate TEXT into arrays of double precision numbers DNUMS(NUMS). Text preceeding numbers is TEXT(KEYA:KEYB), if KEYB >= KEYA.

Parameters

in	*text*	text
out	*nums*	number of numbers found
out	*dnum*	array of numbers found

Definition at line 28 of file mptext.f90.

Referenced by filetc(), and intext().

9.20.2.3 subroutine rltext (character (len=∗), intent(in) *text,* integer(mpi), intent(out) *ia,* integer(mpi), intent(out) *ib,* integer(mpi), intent(out) *nab*)

Analyse text range.

Parameters

in	*text*	text
out	*ia*	index of first non-blank character, or =1
out	*ib*	index of last non-blank character, or =0 - comment excluded
out	*nab*	index of last non-blank character (=0 for blank text)

Definition at line 216 of file mptext.f90.

Referenced by filetc(), filetx(), and intext().

9.21 pede.f90 File Reference

Millepede II program, subroutines.

Functions/Subroutines

- program mptwo

  Millepede II main program Pede.
- subroutine solglo (ivgbi)

  Error for single global parameter from MINRES.
- subroutine solgloqlp (ivgbi)

  Error for single global parameter from MINRES-QLP.
- subroutine addcst

  Add constraint information to matrix and vector.
- subroutine feasma

  Matrix for feasible solution.
- subroutine feasib (concut, iact)

  Make parameters feasible.
- subroutine peread (more)

  Read (block of) records from binary files.
- subroutine peprep (mode)

  Prepare records.
- subroutine pechk (ibuf, nerr)

  Check Millepede record.
- subroutine isjajb (nst, is, ja, jb, jsp)

  Decode Millepede record.
- subroutine loopn

  Loop with fits and sums.
- subroutine ploopa (lunp)

  Print title for iteration.
- subroutine ploopb (lunp)

  Print iteration line.
- subroutine ploopc (lunp)

  Print sub-iteration line.
- subroutine ploopd (lunp)

  Print solution line.
- subroutine explfc (lunit)

  Print explanation of iteration table.
- subroutine mupdat (i, j, add)

  Update element of global matrix.
- subroutine loopbf (nrej, ndfs, sndf, dchi2s, numfil, naccf, chi2f, ndff)

  Loop over records in read buffer (block), fits and sums.
- subroutine prtglo

  Print final log file.
- subroutine avprod (n, x, b)

  Product symmetric matrix times vector.
- integer(mpl) function ijadd (itema, itemb)

  Index for sparse storage.
- subroutine sechms (deltat, nhour, minut, secnd)

  Time conversion.

- INTEGER(mpi) function inone (item)

  Translate labels to indices (for global parameters).
- subroutine upone

  Update, redefine hash indices.
- INTEGER(mpi) function iprime (n)

  largest prime number < N.
- subroutine loop1

  First data loop (get global labels).
- subroutine loop2

  Second data loop (number of derivatives, global label pairs).
- subroutine vmprep (msize)

  Prepare storage for vectors and matrices.
- subroutine minver

  Solution by matrix inversion.
- subroutine mdiags

  Solution by diagonalization.
- subroutine zdiags

  Covariance matrix for diagonalization.
- subroutine mminrs

  Solution with MINRES.
- subroutine mminrsqlp

  Solution with MINRES-QLP.
- subroutine mcsolv (n, x, y)

  Solution for zero band width preconditioner.
- subroutine mvsolv (n, x, y)

  Solution for finite band width preconditioner.
- subroutine xloopn

  Standard solution algorithm.
- subroutine filetc

  Interprete command line option, steering file.
- subroutine filetx

  Interprete text files.
- INTEGER(mpi) function nufile (fname)

  Inquire on file.
- subroutine intext (text, nline)

  Interprete text.
- subroutine additem (length, list, label, value)

  add item to list
- subroutine mstart (text)

  Start of 'module' printout.
- subroutine mend

  End of 'module' printout.
- subroutine mvopen (lun, fname)

  Open file.
- subroutine petime

  Print times.
- subroutine peend (icode, cmessage)

  Print exit code.
- subroutine addsum (add)

  Accurate summation.
- subroutine getsum (asum)

  Get accurate sum.

9.21.1 Detailed Description

Millepede II program, subroutines.

Definition in file pede.f90.

9.21.2 Function/Subroutine Documentation

9.21.2.1 subroutine addcst ( )

Add constraint information to matrix and vector.

Definition at line 1022 of file pede.f90.

References inone(), and mupdat().

Referenced by xloopn().

9.21.2.2 subroutine additem (integer(mpi), intent(inout) *length,* type(listitem), dimension(:), intent(inout), allocatable *list,* integer(mpi), intent(in) *label,* real(mps), intent(in) *value*)

add item to list

Parameters

in,out	*length*	length of list
in,out	*list*	list of items
in	*label*	item label
in	*value*	item value

Definition at line 7432 of file pede.f90.

Referenced by intext().

9.21.2.3 subroutine addsum (real(mpd) *add*)

Accurate summation.

Parameters

in	*add*	summand

Definition at line 7628 of file pede.f90.

Referenced by loopn().

9.21.2.4 subroutine avprod (integer(mpi), intent(in) *n,* real(mpd), dimension(n), intent(in) *x,* real(mpd), dimension(n), intent(out) *b*)

Product symmetric matrix times vector.

A(sym) $*$ X => B. Used by MINRES method (Is most CPU intensive part). The matrix A is the global matrix in full symmetric or (compressed) sparse storage.

Parameters

in	*n*	size of matrix
in	*x*	vector X
in	*b*	result vector B

Definition at line 3633 of file pede.f90.

References peend().

Referenced by mminrs(), mminrsqlp(), solglo(), and solgloqlp().

9.21.2.5 subroutine explfc (integer(mpi) *lunit*)

Print explanation of iteration table.

Definition at line 2414 of file pede.f90.

Referenced by mptwo().

9.21.2.6 subroutine feasib (real(mps), intent(in) *concut,* integer(mpi), intent(out) *iact*)

Make parameters feasible.

Correct for constraint equation discrepancies.

Parameters

in	*concut*	cut for discrepancies
out	*iact*	=1 if correction needed, else =0

Definition at line 1204 of file pede.f90.

References dbsvx(), and inone().

Referenced by xloopn().

9.21.2.7 subroutine feasma ( )

Matrix for feasible solution.

Check rank of product matrix of constraints.

Definition at line 1099 of file pede.f90.

References inone(), and sqminv().

Referenced by loop2().

9.21.2.8 subroutine filetc ( )

Interprete command line option, steering file.

Fetch and interprete command line options, if steering file specified, check file existence (calling NUFILE)

If no steering file specified, check default steering file.

Create test files for command line option '-t'.

Read steering file, print some lines, detect names of text and binary files, check file existence, store all file names.

Open all binary files.

Definition at line 6087 of file pede.f90.

References intext(), matint(), mptest(), mptst2(), mstart(), nufile(), peend(), ratext(), and rltext().

Referenced by mptwo().

9.21.2.9 subroutine filetx ( )

Interprete text files.

Reset flags and read steering and all other text files. Print some lines from each file.

Store parameter values, constraints and measurements.

Check flags METSOL (method of solution) and MATSTO (matrix storage mode). Set default values for flags, which are undefined.

Parameter values, format:

```
  1  label
  2  (initial) parameter value
  3  pre-sigma
  4  label
  5  (initial) parameter value
  6  pre-sigma
  7  label
...  ...
(number of words is multiple of 3)
```

Constraint and Wconstrained data, format:

```
  1  0                ! constraint header of four words:
  2  right-hand-side  ! 0 and -1 ...
  3  -1; -2           ! ... indicate ...
  4  sigma            ! ... header
  5  label
  6  factor
  7  label
  8  factor
  9  ...
...  ...
(number of words is multiple of 2, at least 6)
```

Measured data, format:

```
  1  0                ! constraint header of four words:
  2  right-hand-side  ! 0 and -1 ...
  3  -1               ! ... indicate ...
  4  sigma            ! ... header
  5  label
  6  factor
  7  label
  8  factor
  9  ...
...  ...
(number of words is multiple of 2, at least 6)
```

Definition at line 6514 of file pede.f90.

References intext(), matint(), mend(), peend(), and rltext().

Referenced by mptwo().

9.21.2.10 subroutine getsum (real(mpd), intent(out) *asum*)

Get accurate sum.

Parameters

out	*asum*	accurate sum

Definition at line 7653 of file pede.f90.

Referenced by loopn().

9.21.2.11 integer(mpl) function ijadd (integer(mpi), intent(in) *itema,* integer(mpi), intent(in) *itemb*)

Index for sparse storage.

In case of (compressed) sparse storage calculate index for off-diagonal matrix element.

Parameters

in	*itema*	row number
in	*itemb*	column number

Returns

index (>(<) 0: double(single) precision element, =0: not existing)

Definition at line 3780 of file pede.f90.

Referenced by mupdat(), solglo(), and solgloqlp().

9.21.2.12 INTEGER(mpi) function inone (integer(mpi), intent(in) *item*)

Translate labels to indices (for global parameters).

Functions INONE and subroutine UPONE are used to collect items, i.e. labels, and to order and translate them.

In the first phase items are collected and stored by calling `IRES=INONE(ITEM)`.

At the first entry the two sub-arrays "a" (globalParLabelIndex) and "b" (globalParHashTable) of length 2N are generated with a start length for N=128 entries. In array "a" two words are reserved for each item: (ITEM, count). The function INONE(ITEM) returns the number of the item. At each entry the argument is compared with the already stored items, new items are stored. Search for entries is done using hash-indices, stored in sub-array "b". The initial hash-index is

```
j = 1 + mod(ITEM, n_prime) + N
```

where n_prime is the largest prime number less than N. At each entry the count is increased by one. If N items are stored, the size of the sub-arrays is increased by calling `CALL UPONE`.

Parameters

in	*item*	label

Returns

index

Definition at line 3934 of file pede.f90.

References iprime(), and upone().

Referenced by addcst(), feasib(), feasma(), loop1(), loop2(), loopn(), mvsolv(), peprep(), and xloopn().

9.21.2.13 subroutine intext (character (len=∗), intent(in) *text,* integer(mpi), intent(in) *nline*)

Interprete text.

Look for keywords and argument in text line.

Parameters

in	*text*	text
in	*nline*	line number

Definition at line 6795 of file pede.f90.

References additem(), matint(), ratext(), and rltext().

Referenced by filetc(), and filetx().

9.21.2.14 INTEGER(mpi) function iprime (integer(mpi), intent(in) *n*)

largest prime number < N.

Parameters

in	*n*	N

Returns

largest prime number $<$ N

Definition at line 4061 of file pede.f90.

Referenced by inone(), and upone().

9.21.2.15 subroutine isjajb (integer(mpi), intent(in) *nst,* integer(mpi), intent(inout) *is,* integer(mpi), intent(out) *ja,* integer(mpi), intent(out) *jb,* integer(mpi), intent(out) *jsp*)

Decode Millepede record.

Get indices JA, JB, IS for next measurement within record:

- Measurement is: `GLDER(JA)`
- Local derivatives are: `(INDER(JA+J),GLDER(JA+J),J=1,JB-JA-1)` i.e. JB-JA-1 derivatives
- Standard deviation is: `GLDER(JB)`
- Global derivatives are: `(INDER(JB+J),GLDER(JB+J),J=1,IS-JB)` i.e. IST-JB derivatives

End_of_data is indicated by returned values JA=0 and JB=0 Special data are ignored. At end_of_data the info to the special data is returned: IS = pointer to special data; number of words is `NSP=-GLDER(IS)`.

Parameters

in	*nst*	index of last word of record
in,out	*is*	index of last global derivative (index of first word of record at the first call)
out	*ja*	index of measured value (=0 at end), = pointer to local derivatives
out	*jb*	index of standard deviation (=0 at end), = pointer to global derivatives
out	*jsp*	index to special data

Definition at line 1844 of file pede.f90.

Referenced by loop2(), loopbf(), and peprep().

9.21.2.16 subroutine loop1 ( )

First data loop (get global labels).

Read all data files and add all labels to global labels table, add labels from parameters, constraints and measurements (from text files).

Define variable and fixed global parameters (depending on entries and pre-sigma).

Iterate if records had been skipped due to too small read buffer size.

Definition at line 4093 of file pede.f90.

References hmpdef(), inone(), mend(), mstart(), peend(), peprep(), peread(), and upone().

Referenced by mptwo().

9.21.2.17 subroutine loop2 ( )

Second data loop (number of derivatives, global label pairs).

Calculate maximum number of local, global derivatives and equations per record.

For sparse storage count index pairs with bit (field) counters to construct sparsity structure (row offsets, (compressed) column lists).

Determine read/write cache splitting from average record values (length, global par. vector/matrix).

Check constraints for rank deficit.

Definition at line 4355 of file pede.f90.

References chindl(), ckbits(), clbits(), feasma(), hmpdef(), inbits(), inone(), isjajb(), mend(), mstart(), ndbits(), peprep(), peread(), sort1k(), spbits(), and vmprep().

Referenced by mptwo().

9.21.2.18 subroutine loopbf (integer(mpi), dimension(0:3), intent(inout) *nrej,* integer(mpi), intent(inout) *ndfs,* real(mpd), intent(inout) *sndf,* real(mpd), intent(inout) *dchi2s,* integer(mpi), intent(in) *numfil,* integer(mpi), dimension(numfil), intent(inout) *naccf,* real(mps), dimension(numfil), intent(inout) *chi2f,* integer(mpi), dimension(numfil), intent(inout) *ndff*)

Loop over records in read buffer (block), fits and sums.

Loop over records in current read buffer block (with multiple threads). Perform local fits (optionally with outlier downweigthing) to calculate Chi2, ndf and r.h.s. 'b' of linear equation system A∗x=b. In first iteration(s) fill global matrix A.

For the filling of the global matrix each thread creates from his share of local fits (small) udpdate matrices ($\boldsymbol{\Delta C}_1 + \boldsymbol{\Delta C}_2$ from equations (15), (16)) stored in a write buffer. After all events in the read buffer block have been processed the global matrix is being updated from the matrices in the write buffer in parallel (each row by different thread).

The matrices of the local fits are checked for bordered band structure. For border size b and band width m all elements (i,j) are zero for min(i,j)>b and abs(i-j)>m. For sufficient small (b,m) a solution by root free Cholesky decomposition and forward/backward substitution of the band part is much faster compared to inversion (see broken lines in references). Based on the expected computing cost the faster solution method is selected.

Parameters

in,out	*nrej*	number of rejected records
in,out	*ndfs*	sum(ndf)
in,out	*sndf*	sum(weighted ndf)
in,out	*dchi2s*	sum(weighted chi2)
in	*numfil*	number of binary files
in,out	*naccf*	number of accepted records per binary file
in,out	*chi2f*	sum(chi2/ndf) per binary file
in,out	*ndff*	sum(ndf) per binary file

Definition at line 2591 of file pede.f90.

References chindl(), dbavat(), isjajb(), mupdat(), sort1k(), sqmibb(), and sqminv().

Referenced by loopn().

9.21.2.19 subroutine loopn ( )

Loop with fits and sums.

Loop over all binary files. Perform local fits to calculate Chi2, ndf and r.h.s. 'b' of linear equation system A∗x=b. In first iteration(s) fill matrix A.

Definition at line 1897 of file pede.f90.

References addsum(), getsum(), gmpdef(), hmpdef(), inone(), loopbf(), mupdat(), peprep(), and peread().

Referenced by xloopn().

9.21.2.20 subroutine mcsolv (integer(mpi), intent(in) *n*, real(mpd), dimension(n), intent(in) *x*, real(mpd), dimension(n), intent(out) *y*)

Solution for zero band width preconditioner.

Used by MINRES.

Parameters

in	*n*	size of vectors
in	*x*	rhs vector
out	*y*	result vector

Definition at line 5393 of file pede.f90.

References presol().

Referenced by mminrs(), mminrsqlp(), solglo(), and solgloqlp().

9.21.2.21 subroutine mdiags ( )

Solution by diagonalization.

Definition at line 5127 of file pede.f90.

References devrot(), devsig(), devsol(), gmpdef(), and hmpdef().

Referenced by xloopn().

9.21.2.22 subroutine mend ( )

End of 'module' printout.

Definition at line 7504 of file pede.f90.

References petime().

Referenced by filetx(), loop1(), loop2(), and xloopn().

9.21.2.23 subroutine minver ( )

Solution by matrix inversion.

Parallelized (SQMINL).

Definition at line 5090 of file pede.f90.

References dbsvxl(), and sqminl().

Referenced by xloopn().

9.21.2.24 subroutine mminrs ( )

Solution with MINRES.

Solve A∗x=b by minimizing |A∗x-b| iteratively. Parallelized (AVPROD).

Use preconditioner with zero (precon) or finite (equdec) band width.

Definition at line 5252 of file pede.f90.

References avprod(), equdec(), mcsolv(), minresmodule::minres(), mvsolv(), and precon().

Referenced by xloopn().

9.21.2.25 subroutine mminrsqlp ( )

Solution with MINRES-QLP.

Solve A∗x=b by minimizing |A∗x-b| iteratively. Parallelized (AVPROD).

Use preconditioner with zero (precon) or finite (equdec) band width.

Definition at line 5319 of file pede.f90.

References avprod(), equdec(), mcsolv(), minresqlpmodule::minresqlp(), mvsolv(), and precon().

Referenced by xloopn().

9.21.2.26 program mptwo ( )

Millepede II main program Pede.

Definition at line 425 of file pede.f90.

References explfc(), filetc(), filetx(), gmpdef(), hmpdef(), loop1(), loop2(), mstart(), mvopen(), peend(), sechms(), and xloopn().

9.21.2.27 subroutine mstart (character (len=∗), intent(in) *text*)

Start of 'module' printout.

Definition at line 7468 of file pede.f90.

Referenced by filetc(), loop1(), loop2(), and mptwo().

9.21.2.28 subroutine mupdat (integer(mpi), intent(in) *i,* integer(mpi), intent(in) *j,* real(mpd), intent(in) *add*)

Update element of global matrix.

Add value ADD to matrix element (I,J).

Parameters

in	*i*	first index
in	*j*	second index
in	*add*	summand

Definition at line 2505 of file pede.f90.

References ijadd(), and peend().

Referenced by addcst(), loopbf(), and loopn().

9.21.2.29 subroutine mvopen (integer(mpi), intent(in) *lun,* character (len=∗), intent(in) *fname*)

Open file.

Evtl. move existing file to ∼ version.

Parameters

in	*lun*	unit number
in	*fname*	file name

Definition at line 7521 of file pede.f90.

References peend().

Referenced by mptwo(), peend(), and prtglo().

9.21.2.30 subroutine mvsolv (integer(mpi), intent(in) *n,* real(mpd), dimension(n), intent(in) *x,* real(mpd), dimension(n), intent(out) *y*)

Solution for finite band width preconditioner.

Used by MINRES.

Parameters

in	*n*	size of vectors
in	*x*	rhs vector
out	*y*	result vector

Definition at line 5413 of file pede.f90.

References equslv(), and inone().

Referenced by mminrs(), mminrsqlp(), solglo(), and solgloqlp().

9.21.2.31 INTEGER(mpi) function nufile (character (len=∗), intent(inout) *fname*)

Inquire on file.

Result = 1 for existing binary file, =2 for existing text file, else =0, $<$ 0 open error.

Text file names are recognized by the filename extension, which should contain 'xt' or 'tx'.

Parameters

in,out	*fname*	file name, optionaly strip prefix.

Definition at line 6741 of file pede.f90.

References matint().

Referenced by filetc().

9.21.2.32 subroutine pechk (integer(mpi), intent(in) *ibuf,* integer(mpi), intent(out) *nerr*)

Check Millepede record.

Check integer structure of labels and markers (zeros). Check floats for NaNs.

Parameters

in	*ibuf*	buffer number
out	*nerr*	error flags

Definition at line 1748 of file pede.f90.

Referenced by peprep().

9.21.2.33 subroutine peend (integer(mpi), intent(in) *icode,* character (len=∗), intent(in) *cmessage*)

Print exit code.

Print exit code and message.

Parameters

in	*icode*	exit code
in	*cmessage*	exit massage

Definition at line 7608 of file pede.f90.

References mvopen().

Referenced by avprod(), devrot(), filetc(), filetx(), loop1(), matint(), mpdalc::mpalloccheck(), mpdalc::mpdealloccheck(), mptwo(), mupdat(), mvopen(), peprep(), sort1k(), sort2k(), and xloopn().

9.21.2.34 subroutine peprep (integer(mpi), intent(in) *mode*)

Prepare records.

For global parameters replace label by index (`INONE`).

Parameters

in	*mode*	<=0: build index table (INONE) for global variables; >0: use index table, can be parallelized

Definition at line 1661 of file pede.f90.

References inone(), isjajb(), pechk(), and peend().

Referenced by loop1(), loop2(), and loopn().

9.21.2.35 subroutine peread (integer(mpi), intent(out) *more*)

Read (block of) records from binary files.

Optionally using several threads (each file read by single thread). Records larger than the read buffer (`ndimbuf`) are skipped. In case of skipped events in the first loop over all binary files the buffer size is adapted to the maximum record size (and the initial loop (`LOOP1`) is repeated). C binary files are handled by readc.c and may be gzipped.

Parameters

out	*more*	more records to come

The records consist of parallel integer and real arrays:

```
    real array              integer array
1   0.0                     error count (this record)
2   RMEAS, measured value   0                           JA
3   local derivative        index of local derivative
4   local derivative        index of local derivative
5    ...
6   SIGMA, error (>0)       0                           JB
    global derivative       label of global derivative
    global derivative       label of global derivative  IST
    RMEAS, measured value   0
    local derivative        index of local derivative
    local derivative        index of local derivative
    ...
    SIGMA, error            0
    global derivative       label of global derivative
    global derivative       label of global derivative
    ...
NR  global derivative       label of global derivative
```

Definition at line 1359 of file pede.f90.

References files, and sort2k().

Referenced by loop1(), loop2(), and loopn().

9.21.2.36 subroutine petime ( )

Print times.

Print the elapsed and total time.

Definition at line 7558 of file pede.f90.

Referenced by mend().

9.21.2.37 subroutine ploopa (integer(mpi), intent(in) *lunp*)

Print title for iteration.

Parameters

in	*lunp*	unit number

Definition at line 2269 of file pede.f90.

Referenced by xloopn().

9.21.2.38 subroutine ploopb (integer(mpi), intent(in) *lunp*)

Print iteration line.

Parameters

in	*lunp*	unit number

Definition at line 2290 of file pede.f90.

References ptlopt(), and sechms().

Referenced by xloopn().

9.21.2.39 subroutine ploopc (integer(mpi), intent(in) *lunp*)

Print sub-iteration line.

Parameters

in	*lunp*	unit number

Definition at line 2346 of file pede.f90.

References ptlopt(), and sechms().

Referenced by xloopn().

9.21.2.40 subroutine ploopd (integer(mpi), intent(in) *lunp*)

Print solution line.

Parameters

in	*lunp*	unit number

Definition at line 2388 of file pede.f90.

References sechms().

Referenced by xloopn().

9.21.2.41 subroutine prtglo ( )

Print final log file.

For each global parameter:

- label (I10)
- parameter value (G14.5)
- presigma (G14.5)

- difference of parameters values (G14.5)
- difference at last iteration (G14.5)
- error (standard deviation) (G14.5)
- global correlation (F8.3), on request only

Definition at line 3473 of file pede.f90.

References mvopen().

Referenced by xloopn().

9.21.2.42 subroutine sechms (real(mps), intent(in) *deltat,* integer(mpi), intent(out) *nhour,* integer(mpi), intent(out) *minut,* real(mps), intent(out) *secnd*)

Time conversion.

Convert from seconds to hours, minues, seconds

Parameters

in	*deltat*	time in seconds
out	*nhour*	hours
out	*minut*	minutes
out	*secnd*	seconds

Definition at line 3889 of file pede.f90.

Referenced by mptwo(), ploopb(), ploopc(), and ploopd().

9.21.2.43 subroutine solglo (integer(mpi), intent(in) *ivgbi*)

Error for single global parameter from MINRES.

Calculate single row 'x_i' from inverse matrix by solving A∗x_i=b with b=0 except b_i=1.

Parameters

in	*ivgbi*	index of variable parameter

Definition at line 831 of file pede.f90.

References avprod(), ijadd(), mcsolv(), minresmodule::minres(), and mvsolv().

Referenced by xloopn().

9.21.2.44 subroutine solgloqlp (integer(mpi), intent(in) *ivgbi*)

Error for single global parameter from MINRES-QLP.

Calculate single row 'x_i' from inverse matrix by solving A∗x_i=b with b=0 except b_i=1.

Parameters

in	*ivgbi*	index of variable parameter

Definition at line 927 of file pede.f90.

References avprod(), ijadd(), mcsolv(), minresqlpmodule::minresqlp(), and mvsolv().

Referenced by xloopn().

9.21.2.45 subroutine upone ( )

Update, redefine hash indices.

Definition at line 3997 of file pede.f90.

References iprime(), and sort2k().

Referenced by inone(), and loop1().

9.21.2.46 subroutine vmprep (integer(mpl), dimension(2), intent(in) *msize*)

Prepare storage for vectors and matrices.

Parameters

in	*msize*	number of words for storage of global matrix (double, single prec.)

Definition at line 5004 of file pede.f90.

Referenced by loop2().

9.21.2.47 subroutine xloopn ( )

Standard solution algorithm.
Iterative solution. In current iteration calculate:

```
ICALCM = +1   Matrix, gradient, Function value & solution
ICALCM =  0   gradient, Function value
ICALCM = -1   solution
ICALCM = -2   end
```

Solution is obtained by selected method and improved by line search.

Definition at line 5481 of file pede.f90.

References addcst(), dbdot(), feasib(), inone(), loopn(), mdiags(), mend(), minver(), mminrs(), mminrsqlp(), peend(), ploopa(), ploopb(), ploopc(), ploopd(), prtglo(), ptldef(), ptline(), ptlprt(), solglo(), solgloqlp(), and zdiags().

Referenced by mptwo().

9.21.2.48 subroutine zdiags ( )

Covariance matrix for diagonalization.

Definition at line 5237 of file pede.f90.

References devinv().

Referenced by xloopn().

9.22 randoms.f90 File Reference

Random numbers.

Functions/Subroutines

- subroutine gbrshi (n, a)

  F.Gutbrod random number generator.
- subroutine gbrtim

GBRSHI initialization using TIME().

- REAL(mps) function uran ()

  Random number U(0,1) using RANSHI.

- REAL(mps) function gran ()

  Gauss random number.

9.22.1 Detailed Description

Random numbers. Random number generators for Uniform and Normal distribution:

```
URAN() for U(0,1)
GRAN() for N(0,1)
```

Definition in file randoms.f90.

9.22.2 Function/Subroutine Documentation

9.22.2.1 subroutine gbrshi (integer(mpi), intent(in) *n*, real(mps), dimension(∗), intent(out) *a*)

F.Gutbrod random number generator.

Return N random numbers U(0,1) in array A(N). Initialization by entry GBRVIN.

Parameters

in	*n*	number of requested random number
out	*a*	array of requested random number

Definition at line 21 of file randoms.f90.

Referenced by gran(), and uran().

9.22.2.2 subroutine gbrtim ( )

GBRSHI initialization using TIME().

Definition at line 102 of file randoms.f90.

9.22.2.3 REAL(mps) function gran ( )

Gauss random number.

Returns

random number N(0,1)

Definition at line 142 of file randoms.f90.

References gbrshi().

Referenced by genlin(), genln2(), mptest(), and mptst2().

9.22.2.4 REAL(mps) function uran ( )

Random number U(0,1) using RANSHI.

Returns

random number U(0,1)

Definition at line 122 of file randoms.f90.

References gbrshi().

Referenced by genlin(), genln2(), and mptst2().

9.23 readc.c File Reference

Read C-binary files.

```
#include <stdio.h>
#include "cfortran.h"
```

Functions

- void initC (int *nFiles)

  Initialises the 'global' variables used for file handling.
- void resetC (int *nFileIn)

  Rewind file.
- void openC (const char *fileName, int *errorFlag)

  Open file.
- void readC (float *bufferFloat, int *bufferInt, int *lengthBuffers, int *nFileIn, int *errorFlag)

  Read record from file.

Variables

- FILE ** files

  pointer to list of pointers to opened binary files
- unsigned int maxNumFiles

  max number of files
- unsigned int numAllFiles

  number of opened files

9.23.1 Detailed Description

Read C-binary files. C-methods to handle input of C/C++ binary files as input for the fortran **pede** program (see peread). This includes macros utilising `cfortran.h` to allow direct callability from fortran.

`initC()` has to be called once in the beginning, followed by one or several calls to `openC()` to open one or several files. `readC()` is then called to read the records sequentially. `resetC()` allows to rewind files.

If compiled with preprocessor macro `USE_SHIFT_RFIO`, uses `libRFIO`, i.e. includes `shift.h` instead of `stdio.h`

If compiled with preprocessor macro `USE_ZLIB`, uses `libz`, enables direct reading of gzipped files.

Written by Gero Flucke (`gero.flucke@cern.ch`) in 2006/7

- update on July 14th, 2008
- update on October 29th, 2008: return for file number in `readC()`

Last major update on April 24th, 2012 by C.Kleinwort:

- skip records larger than buffer size (to determine max record length)
- dynamic allocation of file pointer list (no hard-coded max number of files)

Definition in file readc.c.

9.23.2 Function Documentation

9.23.2.1 void initC (int * *nFiles*)

Initialises the 'global' variables used for file handling.

Parameters

in	*nFiles*	Maximal number of C binary files to use.

Definition at line 61 of file readc.c.

References files, maxNumFiles, and numAllFiles.

9.23.2.2 void openC (const char * *fileName,* int * *errorFlag*)

Open file.

Parameters

in	*fileName*	File name
out	*errorFlag*	error flag: • 0: if file opened and OK, • 1: if too many files open, • 2: if file could not be opened • 3: if file opened, but with error (can that happen?)

Definition at line 113 of file readc.c.

References files, maxNumFiles, and numAllFiles.

9.23.2.3 void readC (float * *bufferFloat,* int * *bufferInt,* int * *lengthBuffers,* int * *nFileIn,* int * *errorFlag*)

Read record from file.

Parameters

out	*bufferFloat*	read buffer for floats
out	*bufferInt*	read buffer for integers
in,out	*lengthBuffers*	in: buffer length, out: number of floats/ints in records ($>$ buffer size: record skipped)
in	*nFileIn*	File number (1 .. maxNumFiles)
out	*errorFlag*	error flag: • -1: pointer to a buffer or lengthBuffers are null • -2: problem reading record length • -4: given buffers too short for record • -8: problem with stream or EOF reading floats • -16: problem with stream or EOF reading ints • =0: reached end of file (or read empty record?!) • $>$0: number of words (floats + integers) read and stored in buffers

Definition at line 170 of file readc.c.

References files.

9.23.2.4 void resetC (int * *nFileIn*)

Rewind file.

Parameters

in	*nFileIn*	File number (1 .. maxNumFiles)

Definition at line 97 of file readc.c.

References files.

9.23.3 Variable Documentation

9.23.3.1 FILE** files

pointer to list of pointers to opened binary files

Definition at line 49 of file readc.c.

Referenced by initC(), openC(), peread(), readC(), and resetC().

9.23.3.2 unsigned int maxNumFiles

max number of files

Definition at line 52 of file readc.c.

Referenced by initC(), and openC().

9.23.3.3 unsigned int numAllFiles

number of opened files

Definition at line 53 of file readc.c.

Referenced by initC(), and openC().

9.24 test.f90 File Reference

Functions/Subroutines

- program test

9.24.1 Function/Subroutine Documentation

9.24.1.1 program test ( )

Definition at line 1 of file test.f90.

9.25 vertpr.f90 File Reference

Print vertical.

Functions/Subroutines

- subroutine pzvert (n, x)

  Print vertical.
- subroutine pivert (n, list)

  Vertical print of integer data.
- subroutine pfvert (n, x)

  Vertical print of floating point data.
- subroutine psvert (xa, xb)

  Print scale.

9.25.1 Detailed Description

Print vertical.

Definition in file vertpr.f90.

9.25.2 Function/Subroutine Documentation

9.25.2.1 subroutine pfvert (integer(mpi), intent(in) *n,* integer(mpi), dimension(n), intent(in) *x*)

Vertical print of floating point data.

Print in up to 60 columns. Optionally average data.

Parameters

in	*n*	number of floats
in	*x*	array of floats

Definition at line 204 of file vertpr.f90.

References pzvert().

Referenced by hmpdef().

9.25.2.2 subroutine pivert (integer(mpi), intent(in) *n,* integer(mpi), dimension(n), intent(in) *list*)

Vertical print of integer data.

Print in up to 60 columns. Optionally average data.

Parameters

in	*n*	number of integers
in	*list*	array of integers

Definition at line 164 of file vertpr.f90.

References pzvert().

Referenced by hmpdef().

9.25.2.3 subroutine psvert (real(mps), intent(in) *xa,* real(mps), intent(in) *xb*)

Print scale.

Parameters

in	*xa*	lower bound of range
in	*xb*	upper bound of range

Definition at line 238 of file vertpr.f90.

Referenced by hmpdef().

9.25.2.4 subroutine pzvert (integer(mpi), intent(in) *n,* real(mps), dimension(n), intent(in) *x*)

Print vertical.

Print the array X of dimension N (MAX 120) in 6 lines.

Parameters

in	*n*	number of numbers
in	*x*	array of numbers

Definition at line 15 of file vertpr.f90.

Referenced by pfvert(), and pivert().