Alecsis Manual

Version 2.4

Release Note 1

software version 2.4.1

june 1998

Laboratory for Electronic Design Automation
University of Niš, Faculty of Electronic Engineering
Beogradska 14, 18000 Niš, Yugoslavia

2/1998

Alecsis Manual
Version 2.4
Release Note 1

software version 2.4.1
june 1998

Vladimir Risojević
Željko Mr~arica
Michel Lenczner*

Laboratory for Electronic Design Automation
University of Ni{, Faculty of Electronic Engineering
Beogradska 14, 18000 Niš, Yugoslavia
http://leda.elfak.ni.ac.yu/

**Laboratoire de Calcul Scientifique*
Groupe Materiaux Intelligents
16 Route de Gray, 25000 Besançon

Note:

Introduction of new system of equations solver is explained in this release note. This solver is intended for very large matrices. It enables solution of the matrix block by block, where block which are not currently used are temporarily stored on disk. The matrix must be organized in block for this method to be effective. This is actually intended for simulation of micromechanical problems.

Three sections of the original Manual are given here. Section 5.6.3.3. replaces old version,; section A1.2.1. replaces part of the old version (section is long, so unchaned parts are not repeated here); and section A1.3. replaces old version.

5.6.3.3. Control of system of equations solver

(*new version of the section*)

Table 5.4. Control of sparse matrix solver.

Name	Default value	Meaning
`renum`	`Best` (`2`)	Sparse matrix renumeration algorithm. It can be `None` (`0`), `Fast` (`1`), `Best` (`2`), or `Frontal` (`3`).
`pivot_threshold`	`0`	Used if `renum` equals `Frontal`. This parameter is used by the frontal LU solver during the pivoting. It can be between 0 and 1.
`buffersize`	`0`	Used if `renum` equals `Frontal`. Size of the buffers where coefficients and its indices obtained during the frontal LU decomposition are stored before they are written to temporary files.

Sparse matrix solver can be controlled by means of the option `renum`. This option offers the possibility of choosing between standard, column-oriented, sparse matrix storing scheme (when `renum` is either `None`, `Fast`, or `Best`), and frontal scheme (`renum` is `Frontal`) which is appropriate for matrices arising from finite element method applications, and very closely related to the LU decomposition itself. Furthermore, by its value an algorithm for reordering of matrix rows and columns is specified. In detail, the number of nonzero elements in the system matrix generated during LU decomposition, and consequently memory space needed for storing calculated coefficients, depends on the ordering of matrix rows and columns. This choice can also affect the CPU time necessary for simulation. When frontal LU decomposition is chosen the parameters `pivot_threshold` and `buffersize` are used, otherwise their value is ignored.

In the case of choosing column-oriented sparse matrix storing scheme reordering is performed only once, at the beginning of simulation. If you chose option `Best`, a variant of Berry's algorithm is used, when very detailed (and slow) reordering is performed. This is the default value, as reordering is performed only once, and good reordering guaranties fast simulation. With option `Fast`, a variant of Markowitz's algorithm is used, when reordering is performed much faster, with somewhat slower simulation in time domain afterwards. This option should be chosen for very large matrices (several hundreds of equations or more), since with Berry's algorithm, reordering can take more CPU time than time-domain simulation. When option `None` is chosen, no reordering is performed. This is implemented for comparison only, it has no practical effect, since simulation can take too much time.

If option `Frontal` is chosen, a system matrix is decomposed using the frontal technique, first devised by Irons. This technique is originally developed for solving large positive-definite, symmetric sparse systems of equations such as occur in finite element method applicatons. Duff

extended frontal technique for solving arbitrary large sparse systems of equations, and, a variant of Duff's algorithm is implemented here. Option `Frontal` decomposes matrix in blocks, and can swap currently unused blocks to disk. This enables very large matrices to be solved, that cannot be held in computer memory as a whole. However, if the matrix cannot be organized in blocks, i.e. if there are many nonzero entries far away from the main diagonal, this method cannot be efficient, as blocks cannot be identified.

The basic idea of this algorithm is the fact that a system matrix is formed by performing subsequent assemblies of elemental matrices (stamps). It is obvious that there is an elemental matrix after whose assembly there are no further contributions to some row and column. We say that the variable corresponding to that row and column is fully summed, and can be eliminated if some numerical stability criterion is satisfied. After the elimination is done that row and column are removed from the matrix and the obtained factors are written to disk. In this way, the complete system matrix is never held in the main memory. Instead, all operations are performed in a matrix, called frontal matrix, whose rows and columns correspond to variables that have not yet been eliminated but occur in at least one of the elements that have been assembled. Here, the process of LU decomposition is interleaved with the assembly of system matrix, so this technique permits solving of large sparse systems of equations.

The above mentioned numerical stability criterion depends on the value of the parameter `pivot_threshold`. If its value is 0 (default) then the only constraint on the pivot value is that it must not be zero. This helps in avoid zero pivots that can arise from electrical elements like ideal voltage generators and inductors. However, it is usually important to have larger pivots, as the numerical error is smaller in such case.If the value of `pivot_threshold` is different from zero then the pivot is choosen using the following threshold pivoting criterion: a_{lk} is good pivot if

$$|a_{lk}| \geq \texttt{pivot\_threshold} \cdot \max_i |a_{ik}| \tag{5.14}$$

is satisfied. Note that `pivot_threshold=1.0` corresponds to partial pivoting.

The factors obtained during the elimination are not immediately written to disk. They are first put to buffers in the main memory and only when some buffer is full, its contents is flushed to disk. There are three buffers, two for factors (L and U matrices), and one for their row and column indices. Obviously, if the buffers are large enough it is possible to avoid the usage of disk, or at least reduce it, because it slows down the simulation. The size of the buffers can be specified by setting the value of parameter `buffersize`. It is the size of each of the three buffers. Default value is zero meaning that there is no buffering.

***Note*:** The order of assembly of the elemental matrices determines the order of elimination, and therefore the memory space needed for LU decomposition, so as the simulation time. One may conclude that frontal method is not appropriate for arbitrary, large sparse matrices, but only for those with particular sparsity patterns. Block diagonal matrices represent a class of matrices on which frontal method is applicable, and they often occur in finite element problems. In general, it is desirable that the order of the frontal matrix is as small as possible. This can be achieved by means of the element reordering. Finite element codes usually yield the sequence of elements which meets this requirement.

Note: Values of parameter `renum` - `None`, `Fast`, `Best` and Frontal are actually integer values, defined in standard Alecsis header file `alec.h`. In that file, it is defined:

```
#define None      0
#define Fast      1
#define Best      2
#define Frontal   3
```

Therefore, to use textual values of parameter `renum`, you should have file `alec.h` file included before your `root module` definition, using command:

```
# include <alec.h>
```

A1.2.1. Program call from the command line -- command options

(*section update*)

`-v`*verbose_level* Gives more information about the simulation run. There are following options:

- `-v1` tracks symbol table activity;
- `-v2` tracks intermediate code generation (operand types etc.);
- `-v3` all LEX tokens are printed out as they arrive;
- `-v4` follows voltage generator/inductor loops detection;
- `-v5` prints instructions as they are flushed;
- `-v6` tracks overloading and prototype mangling;
- `-v7` prints list of nodes;
- `-v8` follows the use of weights if option `dcon` is used;
- `-v9` follows the process of static/global initialization;
- `-v10` follows library management;
- `-v11` follows function declaration;
- `-v12` clear global symbol table before simulation;
- `-v13` tracks function prototype existence;
- `-v15` tracks class member access control;
- `-v16` follows function inline expansion;
- `-v31` prints system matrix and right-hand side vector in every iteration, as without reordering. It has no effect if option `renum` equals `Frontal`;
- `-v32` prints system matrix and rhs vector in every iteration as reordered. It has no effect if option `renum` equals `Frontal`;
- `-v33` prints **both** non-reordered and reordered system matrix and rhs vector, respectively, in every iteration. It has no effect if option `renum` equals `Frontal`;
- `-v34` prints statistics if option `renum` equals `Frontal`, otherwise has no effect, viz.
  1. system size,
  2. frontal matrix size,
  3. fully summed variables block size,
  4. number of non-zero entries in the original matrix,
  5. number of factors in both L and U matrices,
  6. number of indices stored for bookkeeping purposes.
- `-v55` turns on full logic initialization
- `-v99` changes all calls of exit() with abort() to dump core file

Note: Verbose level `55` (full logic initialization) is rather a simulation option than a verbose level, and it will be organized as such in following versions of Alecsis.

Most of these verbose levels are of interest only for us that created Alecsis, for our debugging purposes. However, there are some of them that can be very useful for Alecsis users. For instance, **verbose level `8` follows use of weight when option `dcon` for difficult convergence problems is used**. This can be very useful for setting correct values for options `max_weight`, `min_weight`, `p`, `q`, and `maxdcon`, if you are not satisfied with their default values (see Chapter 5, section on simulation `options` for details).

Verbose level `31` prints out system matrix, which can sometimes be helpful if you have problems with zero pivot (singular matrix). This is, however, useful only for small matrices, as it is very difficult to analyze large matrices.

Note: If more than one *verbose_level* is given, only the last one will take effect. For example:

```
alec -v1 -v2 file_name
```

has the same effect as:

```
alec -v2 file_name
```

A1.3. Overview of Alecsis versions

(*new version of the section*)

We use notation of Alecsis versions with tree numbers. First number denotes crucial change of Alecsis/AleC++ functionality. The second one denotes change of functionality (new feature) from the user point of view. The last number is denotes improvement (usually debugging) of existing functions.

Alecsis 1.x	input language based on C, no object-orientation;
Alecsis 2.1.1. - 2.1.50	object-oriented input language AleC++ is introduced;
Alecsis 2.2.1. - 2.2.33.	operator `d2dt2` is introduced;
Alecsis 2.3.1. - 2.3.38	through and across `eqn` statements are introduced;
Alecsis 2.4.1. - 2.4.x	new frontal method for LU decomposition of large sparse matrices is introduced.