AMD Version 1.0 User Guide

Patrick R. Amestoy* Timothy A. Davis† Iain S. Duff‡

April 30, 2003

Abstract

AMD is a set of routines for permuting sparse matrices prior to factorization, using the approximate minimum degree ordering algorithm. It is written in both C and Fortran, with a MATLAB interface.

Technical report TR-03-011, CISE Department, University of Florida, Gainesville, FL, 2003.

Availability: http://www.cise.ufl.edu/research/sparse/amd

Acknowledgments:

This work was supported by the National Science Foundation, under grants ASC-9111263 and DMS-9223088 and DMS-0203270. The conversion to C, the addition of the elimination tree post-ordering, and the handling of dense rows and columns were done while Davis was on sabbatical at Stanford University and Lawrence Berkeley National Laboratory.

*ENSEEIHT-IRIT, 2 rue Camichel 31017 Toulouse, France. email: amestoy@enseeiht.fr. http://www.enseeiht.fr/∼amestoy.

†Dept. of Computer and Information Science and Engineering, Univ. of Florida, Gainesville, FL, USA. email: davis@cise.ufl.edu. http://www.cise.ufl.edu/∼davis. This work was supported by the National Science Foundation, under grants ASC-9111263, DMS-9223088, and DMS-0203270. Portions of the work were done while on sabbatical at Stanford University and Lawrence Berkeley National Laboratory.

‡Rutherford Appleton Laboratory, Chilton, Didcot, Oxon OX11 0QX, England. email: i.s.duff@rl.ac.uk. http://www.numerical.rl.ac.uk/people/isd/isd.html. This work was supported by the the EPSRC under grant GR/R46441.

1 Overview

AMD is a set of routines for pre-ordering a sparse matrix prior to numeric factorization. It uses the approximate minimum degree ordering algorithm to find a permutation matrix $\mathbf{P}$ such that the Cholesky factorization $\mathbf{PAP}^\mathsf{T} = \mathbf{LL}^\mathsf{T}$ has low fill-in (fill-in refers to nonzeros in $\mathbf{L}$ that did not appear in $\mathbf{A}$). If given an unsymmetric matrix, it operates on the symmetric nonzero pattern of $\mathbf{A} + \mathbf{A}^\mathsf{T}$.

The algorithm starts with an undirected graph representation of a symmetric sparse matrix $\mathbf{A}$. Node i in the graph corresponds to row and column i of the matrix, and there is an edge (i, j) in the graph if a_{ij} is nonzero. The degree of a node starts out as the number of off-diagonal nonzeros in row i, which is the size of the set of nodes adjacent to i in the graph.

The selection of a pivot a_{ii} from the diagonal of $\mathbf{A}$ and the first step of Gaussian elimination corresponds to one step of graph elimination. Numerical fill-in results in new nonzero entries in the matrix. Node i is eliminated and edges are added to its neighbors so that they form a clique (or *element*). To reduce fill-in, node i is selected as the node of least degree in the graph. This process repeats until the graph is eliminated. The reduce fill-in, node i is selected as the node of least degree.

The clique is represented implicitly. Rather than listing all the new edges in the graph, a single list of nodes is kept which represents the clique. This list corresponds to the nonzero pattern of the first column of $\mathbf{L}$. As the elimination proceeds, some of these cliques become subsets of subsequent cliques, and are removed. This graph can be stored in place, using the same amount of memory as the original graph.

The most costly part of the minimum degree algorithm is the recomputation of the degrees of nodes adjacent to the current pivot element. Rather than keep track of the exact degree, the approximate minimum degree algorithm finds an upper bound on the degree that is easier to compute. For nodes of least degree, this bound tends to be tight. Using the approximate degree instead of the exact degree leads to a substantial savings in run time, particularly for very irregularly structured matrices. It has no effect on the quality of the ordering.

In the C version of AMD, the elimination phase is followed by an elimination tree post-ordering. This has no effect on fill-in, but reorganizes the ordering so that the subsequent numeric factorization is more efficient. It also includes a pre-processing phase in which nodes of very high degree are removed (without causing fill-in), and placed last in the permutation $\mathbf{P}$. This reduces the run time substantially if the matrix has a few rows with many nonzero entries, and has little effect on the quality of the ordering.

Details of the algorithm are discussed in [1]. For a discussion of the long history of the minimum degree algorithm, see [2].

2 Availability

In addition to appearing as a Collected Algorithm of the ACM, AMD Version 1.0 is available at http://www.cise.ufl.edu/research/sparse. The Fortran version is available as the routine `MC47` in HSL (formerly the Harwell Subroutine Library) [3].

3 Using AMD in MATLAB

The easiest way to use AMD is within MATLAB. To compile the AMD mexFunction, just type `make` in the Unix system shell, while in the `AMD` directory. You can also type `amd_make` in MATLAB,

if you are in the `AMD/MATLAB` directory, or if that directory is in your MATLAB path. This works on any system with MATLAB, including Windows. See Section 8 for more details on how to install AMD.

The MATLAB statement `p = amd (A)` finds a permutation vector `p` such that the Cholesky factorization `chol (A (p,p))` is typically sparser than `chol (A)`. If `A` is not symmetric positive definite, but has substantial diagonal entries and a mostly symmetric nonzero pattern, then this ordering is also suitable for LU factorization. A partial pivoting threshold may be required to prevent pivots from being selected off the diagonal, such as the statement `[L,U,P] = lu (A (p,p), 0.1)`. Type `help lu` for more details. The statement `[L,U,P,Q] = lu (A (p,p))` in MATLAB 6.5 is not suitable, however, because it uses UMFPACK Version 4.0 and thus does not attempt to select pivots from the diagonal. UMFPACK Version 4.1 uses several strategies, including a symmetric pivoting strategy, and will give you better results if you want to factorize an unsymmetric matrix of this type. Refer to the UMFPACK User Guide for more details, at http://www.cise.ufl.edu/research/sparse/umfpack.

An optional input argument can be used to modify the control parameters for AMD, and an optional output argument provides statistics on the ordering, including an analysis of the fill-in and floating-point operation count of a subsequent factorization. AMD will print these statistics if you turn on the sparse matrix monitor flag with `spparms ('spumoni',1)`. Type `help amd` for more information.

4 Using AMD in a C program

The C-callable AMD library consists of four user-callable routines and one include file. Each of the routines come in two versions, with `int` and `long` integers. The routines with prefix `amd_l_` use `long` integer arguments; the others use `int` integer arguments. If you compile AMD in the standard ILP32 mode (32-bit `int`'s, `long`'s, and pointers) then the versions are essentially identical. You will be able to solve problems using up to 2GB of memory. If you compile AMD in the standard LP64 mode, the size of an `int` remains 32-bits, but the size of a `long` and a pointer both get promoted to 64-bits.

The following routines are fully described in Section 9:

- `amd_order` (`long` version: `amd_l_order`)

  Computes the approximate minimum degree ordering of an n-by-n matrix $\mathbf{A}$. Returns a permutation vector `P` of size `n`, where `P [k] = i` if row and column `i` are the `k`th row and column in the permuted matrix. This routine allocates its own memory, and requires $O(|\mathbf{A}|)$ space, where $|\mathbf{A}|$ is the number of nonzero entries in the matrix. It computes statistics about the matrix $\mathbf{A}$, such the symmetry of its nonzero pattern, the number of nonzeros in $\mathbf{L}$, and the floating-point operations required for Cholesky and LU factorizations. The user's input matrix is not modified.

- `amd_defaults` (`long` version: `amd_l_defaults`)

  Sets the default control parameters in the `Control` array. These can then be modified as desired before passing the array to the other AMD routines.

- `amd_control` (`long` version: `amd_l_control`)

  Prints the control parameters.

- `amd_info` (`long` version: `amd_l_info`)

  Prints the statistics computed by AMD.

The nonzero pattern of the matrix $\mathbf{A}$ is represented in compressed column form, which is identical to the sparse matrix representation used by MATLAB. It consists of two arrays, where the matrix is `n`-by-`n`, with `nz` entries. For the `int` version of AMD:

```
int Ap [n+1] ;
int Ai [nz] ;
```

For the `long` version of AMD:

```
long Ap [n+1] ;
long Ai [nz] ;
```

The row indices of entries in column `j` are stored in `Ai[Ap[j]` ... `Ap[j+1]-1]`.

No duplicate row indices may be present, and the row indices in any given column must be sorted in ascending order. The first entry `Ap[0]` must be zero. The total number of entries in the matrix is thus `nz = Ap[n]`. The matrix must be square, but it does not need to be symmetric. If it is unsymmetric, AMD computes the ordering of $\mathbf{A} + \mathbf{A}^\mathsf{T}$. The diagonal entries may be present, but are ignored. AMD checks the input matrix and returns a negative value if it is invalid.

For a more flexible method for providing an input matrix to AMD, refer to the discussion of the "triplet" data structure in the UMFPACK User Guide.

Here is a simple main program, `amd_demo.c`, that illustrates the basic usage of AMD. See Section 5 for a short description of each calling sequence.

```
#include <stdio.h>
#include "amd.h"

int n = 5 ;
int Ap [ ] = { 0,   2,       6,       10,  12, 14} ;
int Ai [ ] = { 0,1, 0,1,2,4, 1,2,3,4, 2,3, 1,4   } ;
int P [5] ;

int main (void)
{
    int k ;
    (void) amd_order (n, Ap, Ai, P, (double *) NULL, (double *) NULL) ;
    for (k = 0 ; k < n ; k++) printf ("P [%d] = %d\n", k, P [k]) ;
    return (0) ;
}
```

The `Ap` and `Ai` arrays represent the binary matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

The diagonal entries are ignored. AMD constructs the pattern of $\mathbf{A}+\mathbf{A}^\mathsf{T}$, and returns a permutation vector of $(0, 3, 2, 4, 1)$. Since the matrix is unsymmetric but with a mostly symmetric nonzero pattern, this would be a suitable permutation for an LU factorization of a matrix with this nonzero

Table 1: AMD Control parameters

MATLAB	ANSI C	default	description
`Control(1)`	`Control[AMD_DENSE]`	10	dense row/column parameter
`Control(2)`	`Control[AMD_AGGRESSIVE]`	1 (yes)	aggressive absorption

pattern and whose diagonal entries are not too small. The program uses default control settings and does not return any statistics about the ordering, factorization, or solution (`Control` and `Info` are both `(double *) NULL`). It also ignores the status value returned by `amd_order`.

The `amd_demo.c` program provides a more detailed demo of AMD. Another example is the AMD mexFunction, `amd_mex.c`.

4.1 A note about zero-sized arrays

AMD uses several user-provided arrays of size `n` or `nz`. Either `n` or `nz` can be zero. If you attempt to `malloc` an array of size zero, however, `malloc` will return a null pointer which AMD will report as invalid. If you `malloc` an array of size `n` or `nz` to pass to AMD, make sure that you handle the `n` = 0 and `nz = 0` cases correctly.

4.2 Control parameters

AMD uses an optional `double` array of size 5, `Control`, to modify its control parameters. If you pass `(double *) NULL` instead of a `Control` array, then defaults are used. NOTE: the size of this array may increase in future versions. Follow good programming practices and define your `Control` array to be of size `AMD_CONTROL`, which is currently defined as 5.

The contents of this array may be modified by the user (see `amd_defaults` or `amd_l_defaults`). Table 1 summarizes the contents of the `Control` array. Note that ANSI C uses 0-based indexing, while MATLAB uses 1-based indexing. Thus, `Control(1)` in MATLAB is the same as `Control[0]` or `Control[AMD_DENSE]` in ANSI C.

Let α =`Control [AMD_DENSE]`. A row or column of $\mathbf{A} + \mathbf{A}^\mathsf{T}$ is considered "dense" if it has more than $\max(16, \alpha\sqrt{n})$ entries. These rows/columns are placed last in AMD's output ordering. If `Control[AMD_AGGRESSIVE]` is nonzero, then AMD performs aggressive absorption. See [1] for details.

5 Synopsis of C-callable routines

The matrix $\mathbf{A}$ is `n`-by-`n` with `nz` entries.

```
#include "amd.h"
int n, status, Ap [n+1], Ai [nz], P [n] ;
double Control [AMD_CONTROL], Info [AMD_INFO] ;
amd_defaults (Control) ;
status = amd_order (n, Ap, Ai, P, Control, Info) ;
amd_control (Control) ;
amd_info (Info) ;
```

The `amd_l_*` routines are identical, except that all `int` arguments become `long`:

```
#include "amd.h"
long n, status, Ap [n+1], Ai [nz], P [n] ;
```

```
double Control [AMD_CONTROL], Info [AMD_INFO] ;
amd_l_defaults (Control) ;
status = amd_l_order (n, Ap, Ai, P, Control, Info) ;
amd_l_control (Control) ;
amd_l_info (Info) ;
```

6 Using AMD in a Fortran program

Two Fortran versions of AMD are provided. The `AMD` routine computes the approximate minimum degree ordering, using aggressive absorption. The `AMDBAR` routine is identical, except that it does not perform aggressive absorption. The `AMD` routine is essentially identical to the HSL routine `MC47B/BD`.

The Fortran versions differ from the C routines. The AMD algorithms were originally coded in Fortran and so are identical to the routines used in the experimental results in [1]. The internal routines require a symmetric nonzero pattern, with no diagonal entries present although the `MC47A/AD` wrapper allows duplicates, ignores out-of-range entries, and only uses entries from the upper triangular part of the matrix. Although we have an experimental Fortran code for treating "dense" rows, the Fortran codes in this release do not treat "dense" rows and columns of $\mathbf{A}$ differently, and thus their run time can be high if there are a few dense rows and columns in the matrix. They do not perform a post-ordering of the elimination tree, compute statistics on the ordering, or check the validity of their input arguments. These functions are provided by the HSL routines `MA57L/LD` and `MC47A/AD`, which are not part of this release. However, details on an assembly tree that respects the AMD ordering are returned from the calls to the Fortran codes. Only one `integer` version of each routine is provided. Both Fortran versions overwrite the user's input matrix, in contrast to the C version. The two Fortran versions have the same calling sequence, and only differ in the name of the routine.

The input matrix is provided to `AMD` and `AMDBAR` in three arrays, `PE`, of size `N`, `LEN`, of size `N`, and `IW`, of size `IWLEN`. The size of `IW` must be at least `NZ+N`. The recommended size is `1.2*NZ + N`. On input, the indices of nonzero entries in row `I` are stored in `IW`. `PE (I)` is the index in `IW` of the start of row `I`. `LEN (I)` is the number of entries in row `I`. Row `I` is contained in `IW (PE (I) ... PE (I) + LEN (I) - 1)`. The diagonal entries must not be present. The indices within each row must not contain any duplicates, but they need not be sorted. The rows themselves need not be in any particular order, and there may be empty space between the rows. If `LEN (I)` is zero, then there are no off-diagonal entries in row `I`, and `PE (I)` is ignored. The integer `PFREE` defines what part of `IW` contains the user's input matrix. The input matrix is in `IW (1 ... PFREE-1)`. The contents of `IW` and `LEN` are undefined on output, and `PE` is modified to contain information about the ordering.

As the algorithm proceeds, it modifies the `IW` array, placing the pattern of the partially eliminated matrix in `IW (PFREE ... IWLEN)`. If this space is exhausted, the space is compacted. The number of compactions performed on the `IW` array is returned in the scalar `NCMPA`. The value of `PFREE` on output is the size `IW` required for no compactions to occur.

The output permutation is returned in the array `LAST`, of size `N`. If `I = LAST (K)`, then `I` is the `K`th row in the permuted matrix. The inverse permutation is returned in the array `ELEN`, where `K = ELEN (I)` if `I` is the `K`th row in the permuted matrix.

On output, the `PE` and `NV` arrays hold the assembly tree, a supernodal elimination tree that represents the relationship between columns of the Cholesky factor $\mathbf{L}$. If `NV (I)` > 0, then `I` is a node in the assembly tree, and the parent of `I` is `-PE (I)`. If `I` is a root of the tree, then `PE (I)` is zero. The value of `NV (I)` is the number of entries in the corresponding column of $\mathbf{L}$, including the

diagonal. If NV (I) is zero, then I is a non-principal node that is not in the assembly tree. Node -PE (I) is the parent of node I in a subtree, the root of which is a node in the assembly tree. All nodes in one subtree belong to the same supernode in the assembly tree.

The other size N arrays (DEGREE, HEAD, NEXT, and W) are used as workspace, and are not defined on input or output.

If you want to use a simpler user-interface and compute the elimination tree post-ordering, you should be able to call the C routines amd_order or amd_l_order from a Fortran program. Just be sure to take into account the 0-based indexing in the P, Ap, and Ai arguments to amd_order and amd_l_order. A sample interface is provided in the files AMD/Demo/amd_f77cross.f and AMD/Demo/amd_f77wrapper.c. To compile the amd_f77cross program, type make cross in the AMD/Demo directory. The Fortran-to-C calling conventions are highly non-portable, so this example is not guaranteed to work with your compiler C and Fortran compilers. The output of amd_f77cross is in amd_f77cross.out.

7 Synopsis of Fortran-callable routines

```
   INTEGER N, IWLEN, PFREE, NCMPA, IW (IWLEN), PE (N), DEGREE (N), NV (N),
$          NEXT (N), LAST (N), HEAD (N), ELEN (N), W (N), LEN (N)

   CALL AMD (N, PE, IW, LEN, IWLEN, PFREE, NV, NEXT,
$          LAST, HEAD, ELEN, DEGREE, NCMPA, W)

   CALL AMDBAR (N, PE, IW, LEN, IWLEN, PFREE, NV, NEXT,
$          LAST, HEAD, ELEN, DEGREE, NCMPA, W)
```

8 Installation

The following discussion assumes you have the make program, either in Unix, or in Windows with Cygwin.

System-dependent configurations are in the AMD/Make directory. You can edit the Make.include file in that directory to customize the compilation. The default settings will work on most systems. Sample configuration files are provided for Linux, Sun Solaris, SGI IRIX, IBM AIX, and the DEC/Compaq Alpha.

To compile and install the C-callable AMD library, go to the AMD directory and type make. The library will be placed in AMD/Lib/libamd.a. Two demo programs of the AMD ordering routine will be compiled and tested in the AMD/Demo directory. The outputs of these demo programs will then be compared with output files in the distribution. The AMD mexFunction for use in MATLAB will also be compiled. If you do not have MATLAB type make lib instead.

To compile and install the Fortran-callable AMD library, go to the AMD directory and type make fortran. The library will be placed in AMD/Lib/libamdf77.a. A demo program will be compiled and tested in the AMD/Demo directory. The output will be compared with an output file in the distribution.

If you compile AMD and then later change the Make.include file or your system-specific configuration file such as Make.linux, then you should type make purge and then make to recompile.

Here are the various parameters that you can control in your Make.include file:

- CC = your C compiler, such as cc.
- RANLIB = your system's ranlib program, if needed.

- `CFLAGS =` optimization flags, such as `-O`.
- `LIB =` your libraries, such as `-lm` or `-lblas`.
- `RM =` the command to delete a file.
- `MV =` the command to rename a file.
- `MEX =` the command to compile a MATLAB mexFunction.
- `F77 =` the command to compile a Fortran program (optional).
- `F77FLAGS =` the Fortran compiler flags (optional).
- `F77LIB =` the Fortran libraries (optional).

When you compile your program that uses the C-callable AMD library, you need to add the `AMD/Lib/libamd.a` library and you need to tell your compiler to look in the `AMD/Include` directory for include files. To compile a Fortran program that calls the Fortran AMD library, you need to add the `AMD/Lib/libamdf77.a` library. See `AMD/Demo/Makefile` for an example.

If all you want to use is the AMD mexFunction in MATLAB, you can skip the use of the `make` command entirely. Simply type `amd_make` in MATLAB while in the `AMD/MATLAB` directory.

9 The AMD routines

The file `AMD/Include/amd.h` listed below describes each user-callable routine in the C version of AMD, and gives details on their use.

```
/* ========================================================================== */
/* === AMD:  approximate minimum degree ordering ============================ */
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* AMD Version 1.0 (Apr. 30, 2003), Copyright (c) 2003 by Timothy A. Davis,   */
/* Patrick R. Amestoy, and Iain S. Duff.  See ../README for License.          */
/* email: davis@cise.ufl.edu    CISE Department, Univ. of Florida.            */
/* web: http://www.cise.ufl.edu/research/sparse/amd                           */
/* -------------------------------------------------------------------------- */

/* AMD finds a symmetric ordering P of a matrix A so that the Cholesky
 * factorization of P*A*P' has fewer nonzeros and takes less work than the
 * Cholesky factorization of A.  If A is not symmetric, then it performs its
 * ordering on the matrix A+A'.  Two sets of user-callable routines are
 * provided, one for "int" integers and the other for "long" integers.
 *
 * The method is based on the approximate minimum degree algorithm, discussed in
 * Amestoy, Davis, and Duff, "An approximate degree ordering algorithm", SIAM
 * Journal of Matrix Analysis and Applications, vol. 17, no. 4, pp.  886-905,
 * 1996.  This package can perform both the AMD ordering (with aggressive
 * absorption), and the AMDBAR ordering (without aggressive absorption)
 * discussed in the above paper.  This package differs from the Fortran codes
 * discussed in the paper:
 *
 *      (1) it can ignore "dense" rows and columns, leading to faster run times,
 *      (2) it computes the ordering of A+A' if A is not symmetric,
 *      (3) it is followed by a depth-first post-ordering of the assembly tree
 *          (or supernodal elimination tree)
 *
 * For historical reasons, the Fortran versions, amd.f and amdbar.f, have
 * been left unchanged.  They compute the identical ordering as described in
 * the above paper.
 */

#ifndef AMD_H
#define AMD_H

int amd_order (             /* returns 0 if OK, negative value if error */
    int n,                  /* A is n-by-n.  n must be >= 0. */
    const int Ap [ ],       /* column pointers for A, of size n+1 */
    const int Ai [ ],       /* row indices of A, of size nz = Ap [n] */
    int P [ ],              /* output permutation, of size n */
    double Control [ ],     /* input Control settings, of size AMD_CONTROL */
    double Info [ ]         /* output Info statistics, of size AMD_INFO */
) ;

long amd_l_order (          /* see above for description of arguments */
    long n,
    const long Ap [ ],
    const long Ai [ ],
    long P [ ],
    double Control [ ],
    double Info [ ]
```

```
) ;

/* Input arguments (not modified):
 *
 *      n: the matrix A is n-by-n.
 *      Ap: an int/long array of size n+1, containing the column pointers of A.
 *      Ai: an int/long array of size nz, containing the row indices of A,
 *          where nz = Ap [n].
 *      Control:  a double array of size AMD_CONTROL, containing control
 *          parameters.  Defaults are used if Control is NULL.
 *
 * Output arguments (not defined on input):
 *
 *      P: an int/long array of size n, containing the output permutation. If
 *          row i is the kth pivot row, then P [k] = i.  In MATLAB notation,
 *          the reordered matrix is A (P,P).
 *      Info: a double array of size AMD_INFO, containing statistical
 *          information.  Ignored if Info is NULL.
 *
 * On input, the matrix A is stored in column-oriented form.  The row indices
 * of nonzero entries in column j are stored in Ai [Ap [j] ... Ap [j+1]-1].
 * The row indices must appear in ascending order in each column, and there
 * must not be any duplicate entries.  Row indices must be in the range 0 to
 * n-1.  Ap [0] must be zero, and thus nz = Ap [n] is the number of nonzeros
 * in A.  The array Ap is of size n+1, and the array Ai is of size nz = Ap [n].
 * The matrix does not need to be symmetric, and the diagonal does not need to
 * be present (if diagonal entries are present, they are ignored except for
 * the output statistic Info [AMD_NZDIAG]).  The arrays Ai and Ap are not
 * modified.  This form of the Ap and Ai arrays to represent the nonzero
 * pattern of the matrix A is the same as that used internally by MATLAB.
 * If you wish to use a more flexible input structure, please see the
 * umfpack_*_triplet_to_col routines in the UMFPACK package, at
 * http://www.cise.ufl.edu/research/sparse/umfpack.
 *
 * Restrictions:  n >= 0.  Ap [0] = 0.  Ap [j] <= Ap [j+1] for all j in the
 *      range 0 to n-1.  nz = Ap [n] >= 0.  For all j in the range 0 to n-1,
 *      and for all p in the range Ap [j] to Ap [j+1]-2, Ai [p] < Ai [p+1] must
 *      hold.  Ai [0..nz-1] must be in the range 0 to n-1.  To avoid integer
 *      overflow, (2.4*nz + 8*n) < INT_MAX / sizeof (int) for must hold for the
 *      "int" version. (2.4*nz + 8*n) < LONG_MAX / sizeof (long) must hold
 *      for the "long" version.  Finally, Ai, Ap, and P must not be NULL.  If
 *      any of these restrictions are not met, AMD returns AMD_INVALID.
 *
 * AMD returns:
 *
 *      AMD_OK if the matrix is valid and sufficient memory can be allocated to
 *          perform the ordering.
 *
 *      AMD_OUT_OF_MEMORY if not enough memory can be allocated.
 *
 *      AMD_INVALID if the input arguments n, Ap, Ai are invalid, or if P is
 *          NULL.
 *
 * The AMD routine first forms the pattern of the matrix A+A', and then computes
 * a fill-reducing ordering, P.  If P [k] = i, then row/column i of the original
 * is the kth pivotal row.  In MATLAB notation, the permuted matrix is A (P,P),
 * except that 0-based indexing is used instead of the 1-based indexing in
 * MATLAB.
 *
```

```
* The Control array is used to set various parameters for AMD.  If a NULL
* pointer is passed, default values are used.  The Control array is not
* modified.
*
*      Control [AMD_DENSE]:  controls the threshold for "dense" rows/columns.
*          A dense row/column in A+A' can cause AMD to spend a lot of time in
*          ordering the matrix.  If Control [AMD_DENSE] >= 0, rows/columns with
*          more than Control [AMD_DENSE] * sqrt (n) entries are ignored during
*          the ordering, and placed last in the output order.  The default
*          value of Control [AMD_DENSE] is 10.  If negative, no rows/columns
*          are treated as "dense".  Rows/columns with 16 or fewer off-diagonal
*          entries are never considered "dense".
*
*      Control [AMD_AGGRESSIVE]: controls whether or not to use aggressive
*          absorption, in which a prior element is absorbed into the current
*          element if is a subset of the current element, even if it is not
*          adjacent to the current pivot element (refer to Amestoy, Davis,
*          & Duff, 1996, for more details).  The default value is nonzero,
*          which means to perform aggressive absorption.  This nearly always
*          leads to a better ordering (because the approximate degrees are more
*          accurate) and a lower execution time.  There are cases where it can
*          lead to a slightly worse ordering, however.  To turn it off, set
*          Control [AMD_AGGRESSIVE] to 0.
*
*      Control [2..4] are not used in the current version, but may be used in
*          future versions.
*
* The Info array provides statistics about the ordering on output.  If it is
* not present, the statistics are not returned.  This is not an error
* condition.
*
*      Info [AMD_STATUS]:  the return value of AMD, either AMD_OK,
*          AMD_OUT_OF_MEMORY, or AMD_INVALID.
*
*      Info [AMD_N]: n, the size of the input matrix
*
*      Info [AMD_NZ]: the number of nonzeros in A, nz = Ap [n]
*
*      Info [AMD_SYMMETRY]:  the symmetry of the matrix A.  It is the number
*          of "matched" off-diagonal entries divided by the total number of
*          off-diagonal entries.  An entry A(i,j) is matched if A(j,i) is also
*          an entry, for any pair (i,j) for which i != j.  In MATLAB notation,
*              S = spones (A) ;
*              B = tril (S, -1) + triu (S, 1) ;
*              symmetry = nnz (B & B') / nnz (B) ;
*
*      Info [AMD_NZDIAG]: the number of entries on the diagonal of A.
*
*      Info [AMD_NZ_A_PLUS_AT]:  the number of nonzeros in A+A', excluding the
*          diagonal.  If A is perfectly symmetric (Info [AMD_SYMMETRY] = 1)
*          with a fully nonzero diagonal, then Info [AMD_NZ_A_PLUS_AT] = nz-n
*          (the smallest possible value).  If A is perfectly unsymmetric
*          (Info [AMD_SYMMETRY] = 0, for an upper triangular matrix, for
*          example) with no diagonal, then Info [AMD_NZ_A_PLUS_AT] = 2*nz
*          (the largest possible value).
*
*      Info [AMD_NDENSE]: the number of "dense" rows/columns of A+A' that were
*          removed from A prior to ordering.  These are placed last in the
*          output order P.
```

```
 *
 *      Info [AMD_MEMORY]: the amount of memory used by AMD, in bytes.  In the
 *          current version, this is 1.2 * Info  [AMD_NZ_A_PLUS_AT] + 9*n
 *          times the size of an integer.  This is at most 2.4nz + 9n.  This
 *          excludes the size of the input arguments Ai, Ap, and P, which have
 *          a total size of nz + 2*n + 1 integers.
 *
 *      Info [AMD_NCMPA]: the number of garbage collections performed.
 *
 *      Info [AMD_LNZ]: the number of nonzeros in L (excluding the diagonal).
 *          This is a slight upper bound because mass elimination is combined
 *          with the approximate degree update.  It is a rough upper bound if
 *          there are many "dense" rows/columns.  The rest of the statistics,
 *          below, are also slight or rough upper bounds, for the same reasons.
 *          The post-ordering of the assembly tree might also not exactly
 *          correspond to a true elimination tree postordering.
 *
 *      Info [AMD_NDIV]: the number of divide operations for a subsequent LDL'
 *          or LU factorization of the permuted matrix A (P,P).
 *
 *      Info [AMD_NMULTSUBS_LDL]:  the number of multiply-subtract pairs for a
 *          subsequent LDL' factorization of A (P,P).
 *
 *      Info [AMD_NMULTSUBS_LU]:  the number of multiply-subtract pairs for a
 *          subsequent LU factorization of A (P,P), assuming that no numerical
 *          pivoting is required.
 *
 *      Info [AMD_DMAX]:  the maximum number of nonzeros in any column of L,
 *          including the diagonal.
 *
 *      Info [14..19] are not used in the current version, but may be used in
 *          future versions.
 */

/* ---------------------------------------------------------------------- */
/* AMD Control and Info arrays */
/* ---------------------------------------------------------------------- */

/* amd_defaults:  sets the default control settings */
void amd_defaults   (double Control [ ]) ;
void amd_l_defaults (double Control [ ]) ;

/* amd_control: prints the control settings */
void amd_control    (double Control [ ]) ;
void amd_l_control  (double Control [ ]) ;

/* amd_info: prints the statistics */
void amd_info       (double Info [ ]) ;
void amd_l_info     (double Info [ ]) ;

#define AMD_CONTROL 5       /* size of Control array */
#define AMD_INFO 20         /* size of Info array */

/* contents of Control */
#define AMD_DENSE 0         /* "dense" if degree > Control [0] * sqrt (n) */
#define AMD_AGGRESSIVE 1    /* do aggressive absorption if Control [1] != 0 */

/* default Control settings */
#define AMD_DEFAULT_DENSE 10.0      /* default "dense" degree 10*sqrt(n) */
```

```
#define AMD_DEFAULT_AGGRESSIVE 1    /* do aggressive absorption by default */

/* contents of Info */
#define AMD_STATUS 0        /* return value of amd_order and amd_l_order */
#define AMD_N 1             /* A is n-by-n */
#define AMD_NZ 2            /* number of nonzeros in A */
#define AMD_SYMMETRY 3      /* symmetry of pattern (1 is sym., 0 is unsym.) */
#define AMD_NZDIAG 4        /* # of entries on diagonal */
#define AMD_NZ_A_PLUS_AT 5  /* nz in A+A' */
#define AMD_NDENSE 6        /* number of "dense" rows/columns in A */
#define AMD_MEMORY 7        /* amount of memory used by AMD */
#define AMD_NCMPA 8         /* number of garbage collections in AMD */
#define AMD_LNZ 9           /* approx. nz in L, excluding the diagonal */
#define AMD_NDIV 10         /* number of fl. point divides for LU and LDL' */
#define AMD_NMULTSUBS_LDL 11 /* number of fl. point (*,-) pairs for LDL' */
#define AMD_NMULTSUBS_LU 12  /* number of fl. point (*,-) pairs for LU */
#define AMD_DMAX 13          /* max nz. in any column of L, incl. diagonal */

/* ------------------------------------------------------------------------ */
/* return values of AMD */
/* ------------------------------------------------------------------------ */

#define AMD_OK 0                /* success */
#define AMD_OUT_OF_MEMORY -1    /* malloc failed, or 2.4*nz+9*n is too large */
#define AMD_INVALID -2          /* input arguments are not valid */

#endif
```

References

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Applic.*, 17(4):886–905, 1996.

[2] A. George and J. W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989.

[3] HSL. HSL 2002: A collection of Fortran codes for large scale scientific computation, 2002. `www.cse.clrc.ac.uk/nag/hsl`.