Documentation for RnLU.h and RnLU.c

Steven Andrews, © 2003
See the document "LibDoc" for general information about this and other libraries.

```
float LUdecomp(float *a,int n,int **indx);
void LUsolveV(float *a,int *indx,float *b,int n);
void LUsolveM(float *a,int *indx,float *b,int n,int nb);
float LUlndetM(float *a,int *indx,int n,int *sgn);
float LUimprV(float *a,float *alu,int *indx,float *x,float *b,int n);
float LUimprM(float *a,float *alu,int *indx,float *x,float *b,int n,int
      nb);
```

Requires: `<stdlib.h>,<math.h>,"math2.h","Rn.h"`
Example program: `LibTest.c, SpectFit.c`

Written 9/13/98. Works with Metrowerks C. The first four routines seem to work well. Other routines haven't been tested yet. Proofread and cleaned up slightly 1/31/02.

These routines are copied almost verbatim from *Numerical Recipes in C* and are intended to give moderately high performance solutions to linear equations, by using the LU decomposition method. In brief, a matrix can be decomposed into the product of a lower triangular matrix (L) and an upper triangular matrix (U), where the diagonal elements of L are each equal to 1. As the two matrices are each zero where the other is non-zero, except along the diagonal, where L is always equal to 1, the two matrices may be stored in an overlapped fashion, which is called the LU decomposition.

`LUdecomp` forms the LU decomposition of an `nxn` matrix `a`, returning the result back in `a` (thus, `a` needs to be copied beforehand if the original is of interest). As the result is sensitive to pivot choices, rows are swapped as need be (this uses implicit partial pivoting), and the new row indexing is returned in `indxptr`. `indxptr` should be sent to the routine as the address of an uninitiallized pointer to an integer (e.g. `&indx`), and is returned as a size `n+1` vector of integers. The final value of `indx`, `indx[n]`, is either 1 or -1, depending on whether an even or odd number of row swaps was performed. `LUdecomp` returns the determinant of the matrix; if it returns 0, then the contents of `a` is junk and *`indxptr` is `NULL`. When it is no longer needed, `indx` should be freed with the `<stdlib.h>` routine `free()`.

`LUsolveV` solves vector linear equations, using the LU decomposed matrix. The equation solved is **Ax=b**, where the `a` input is the LU decomposed matrix, along with its row swapping index `indx` and `b` is the known vector. The result, **x**, is sent back as `b`, so, again, `b` needs to be copied beforehand if it is of continuing interest.

`LUsolveM` solves matrix linear equations, using the LU decomposed matrix. The equation solved is **AX=B**, where the `a` input is the LU decomposed matrix, along with its row swapping index `indx` and `b` is the known matrix, with dimensions `nxnb`. The result, **X**, is sent back as `b`. The best way to do matrix inversion is to LU decompose the matrix and to make the matrix `b` the identity matrix. Then do `LUsolveM` and `b` comes back as the inverse.

`LUlndetM` returns the natural log of the absolute value of the determinant of an LU decomposed matrix, as well as the sign of the determinant. This is here primarily for matrices whose determinants overflow or underflow the normal float range. Send in `sgn` as the address of an integer variable where the sign should be returned, or `NULL` if the sign isn't desired.

`LUimprV` uses the above routines to improve a linear solution. `a` is the original matrix, `alu` is its LU decomposed form, `indx` is the corresponding row swap index, `x` is the best solution available, and `b` is the vector of known values. The routine returns the improved `x` vector in `x`. It also returns the fractional error that was calculated, which is the correction vector length, divided by the total `x` vector length. A return value of zero implies either that the input vector was exact or that memory could not be allocated for a work vector. This routine is rarely required for small or well behaved matrices, and may actually make the result slightly worse due to the fact that it does several calculations, each of which has some round-off error. However, it is worthwhile for large or poorly behaved matrices.

`LUimprM` is just like `LUimprV` except it is for matrices. Its returned value is the fractional rms error of the matrix components (the error is calculated exactly as in the previous routine, but this time treating the matrix as a vector).