A practical guide to computer simulation II

Alexander K. Hartmann, University of Göttingen

June 11, 2003

6 Libraries

Library = collection of data types and subroutines.

Many types, e.g.

- Numerical
- Data storing/access (data bases)
- Enhanced data types
- Graph algorithms
- Graphics

Idea: Don't try to write everything yourself. Build on the things others have done. Saves much time and effort.

6.1 Numerical recipes

Literature: [1]. Source code + short explenation how algorithms work.
Subroutines for:

- solving linear equations
- performing interpolations
- evaluation and integration of functions
- solving nonlinear equations
- minimizing functions
- diagonalization of matrices
- Fourier transform
- solving ordinary and partial differential equations.

Similar libraries: *Numerical Algorithms Group* [3] and *Maple* software package [4].
Example: Calculate eigenvalues of symmetric matrix. Method: Housholder transformation to tridiagonal form. The use **QL** decomposition (**Q**=orthogonal matrix, **L**= lower left triangular matrix). Attention: Vectors in NR start at index 1!

```
#include <stdio.h>
#include <stdlib.h>
#include "nrutil.h"
#include "nr.h"

int main(int argc, char *argv[])
{
  float **m, *d, *e;                        /* matrix, two vectors */
  long n =  10;                                  /* size of matrix */
  int i, j;                                        /* loop counter */

  m = matrix(1, n, 1, n);                        /* allocate matrix */
  for(i=1; i<=n; i++)                /* initialize matrix randomly */
    for(j=i; j<=n; j++)
    {
        m[i][j] = drand48();
        m[j][i] = m[i][j];       /* matrix must be symmetric here */
    }

  d = vector(1,n);          /* contains diagonal elements          */
  e = vector(1,n);          /* contains off diagonal elements      */
  tred2(m, n, d, e);        /* convert symmetric m. -> tridiagonal */
  tqli(d, e, n, m);         /* calculate eigenvalues               */
  for(j=1; j<=n; j++)       /* print result stored now in array 'd'*/
    printf("ev %d = %f\n", j, d[j]);

  free_vector(e, 1, n);                         /* give memory back */
  free_vector(d, 1, n);
  free_matrix(m, 1, n, 1, n);
  return(0);
}
```

The library is distributed over many source code files. Which ones you need, you can see from the appendix of [1] Here, compile with:
`cc -o ev_test ev_test.c tqli.c tred2.c nrutil.c pythag.c -lm`
Sometimes the library is locally available:
`cc -o ev_test ev_test.c -lrecipes_c -lm`
Remark: also functions for eigenvalues of general matrices are included in NR.

6.2 LEDA

Library of Efficient Data types and Algorithms (LEDA) [2]: C++, can be use from C as well. Contains data types:

- strings
- numbers of arbitrary precision
- one- and two-dimensional arrays
- lists and similar objects like stacks or queues
- sets

- trees
- graphs (directed and undirected, also labeled)
- dictionaries, there you can store objects with arbitrary key words as indices
- data types for two and three dimensional geometries, like points, segments or spheres

Templates: object of arbitrary basic type (list of arbitrary data types) possible.

Different implementations of data types for special requirements. Highly efficient implementations.

All opeerations are included, e.g. lists: creating, appending, splitting, printing and deleting lists as well as inserting, searching, sorting and deleting elements in a list, also iterating over all elements of a list.

Major part: graphs and related algorithms. E.g. calculate strongly connected components, shortest paths, maximum flows, minimum cost flows and minimum matchings.
Example: List of objects. Remark: Input and output operators must be provied to make program compile.

```
#include <iostream.h>
#include <LEDA/list.h>

class Mydatatype                // self defined example class
{
 public:
    int       info;                              // user data 1
    short int flag;                              // user data 2
    Mydatatype() {info=0; flag=0;};              // constructor
    ~Mydatatype() {};                            // destructor
    friend ostream& operator<<(ostream& O, const Mydatatype& dt)
        { O << "info: " << dt.info << " flag: " << dt.flag << "\n";
          return(O);};                           // output operator
    friend istream& operator>>(istream &I, Mydatatype& dt)
        {return(I);};                            // dummy
};

int main(int argc, char *argv[])
{
  list<Mydatatype> l;       // list with elements of 'Mydatatype'
  Mydatatype element;
  int t;

  for(t=0; t<10; t++)       // create list
  {
    element.info = t;
    element.flag = t%2;
    l.append(element);
```

```
  }
  forall(element, l)        // iterate over all elements
    if(element.flag)        // print only 'even' elements
      cout << element;
  return(0);
}
```

Compile:

```
g++ -I$LEDAROOT/incl -L$LEDAROOT -o leda_test leda_test.cc -lG -lL
```

LEDA is not free, only recently became commercial, cheapest: student licence = 49 Euro. Installed in theoretical physics and GWDG. Cip pool: ask system administrator to install (including old g++ 2.95.2).

6.3 Creating your own Libraries

During the years: build upon your own work, include in other programs.

Input: object file e.g. `tasks.o`, and a header file `tasks.h`.
Create Library:

```
ar r libmy.a tasks.o
```

"`r`": replace object file (or add with not included). More options `man ar`.
Then: update internal object table of library:

```
ar s libmy.a
```

Compiling e.g. `prog.c`:

```
cc -o prog prog.c libmy.a
```

More comfortable: create cirectories ∼`/lib` and ∼`/include`. Then compiling:

```
cc -o prog prog.c -I$HOME/include -L$HOME/lib -lmy
```

If others want to use library: write `man` page (`man man`).

References

[1] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C* (Cambridge University Press, Cambridge 1995);
see also `www.nr.com`

[2] K. Mehlhorn and St. Näher, *The LEDA Platform of Combinatorial and Geometric Computing* (Cambridge University Press, Cambridge 1999);
see also `www.algorithmic-solutions.com`

[3] J. Phillips, *The Nag Library: A Beginner's Guide* (Oxford University Press, Oxford 1987);
see also `www.nag.com`

[4] A. Heck, *Introduction to Maple*, (Springer-Verlag, New York 1996);
see also `www.maplesoft.com`