Chapter 7

Installation and copyright

7.1 Installation

There are several different forms in which you might receive Meschach. To provide a shorthand for describing collections of files, the Unix convention of putting alternative letters in `[...]` will be used. (So, `fred[123]` means the collection `fred1`, `fred2` and `fred3`.) Meschach is available over Internet/AARnet via netlib, or at the anonymous ftp site `thrain.anu.edu.au` in the directory `pub/meschach`. There are five `.shar` files: `meschach[01234].shar` (which contain the library itself), of which `meschach0.shar` contains basic documentation and machine dependent files for a number of machines. Of the `meschach[1234].shar` files, only `meschach[12].shar` are needed for the basic Meschach library; the third `.shar` file contains the sparse matrix routines, and the the fourth contains the routines for complex numbers, vectors and matrices. There is also this `README` file that you should get directly, or extract it from `meschach0.shar`.

If you need the old iterative routines, the file `oldmeschach.shar` contains the files `conjgrad.c`, `arnoldi.c` and `lanczos.c`.

To get the library from netlib,

```
mail netlib@research.att.com
send all from c/meschach
```

There are a number of other netlib sites which mirror the main netlib sites. These include `netlib@ornl.gov` (Oak Ridge, TN, USA), `netlib@nac.no` (Oslo, Norway), `ftp.cs.uow.edu.au` (Wollongong, Australia; ftp only), `netlib@nchc.edu.tw` (Taiwan), `elib.zib-berlin.de` (Berlin, Germany; ftp only). (For anonymous ftp sites the directory containing the Meschach `.shar` files is `pub/netlib/c/meschach` or similar, possibly depending on the site.)

Meschach is available in other forms on `thrain.anu.edu.au` by ftp in the directory `pub/meschach`. It is available as a `.tar` file (`mesch12a.tar` for version 1.2a), or as a collection of `.shar` files, or as a `.zip` file. The `.tar` and `.zip` versions each contain the entire contents of the Meschach library.

To extract the files from the `.shar` files, put them all into a suitable directory and use

```
sh meschach0.shar
sh meschach1.shar
sh meschach2.shar
sh meschach3.shar
sh meschach4.shar
sh meschach5.shar
```

to expand the files. (Use one `sh` command per file; `sh *.shar` will not work in general.)

For the `.tar` file, use

```
tar xvf mesch12a.tar
```

and for the `.zip` file use

```
unzip mesch12a.zip
```

(Or use `pkunzip mesch12a.zip` if you have `pkunzip`.)

On a Unix system you can use the `configure` script to set up the machine-dependent files. The script takes a number of options which are used for installing different subsets of the full Meschach. For the basic system, which requires only `meschach[012].shar`, use

```
configure
make basic
make clean
```

For including sparse operations, which requires `meschach[0123].shar`, use

```
configure --with-sparse
make sparse
make clean
```

For including complex operations, which requires `meschach[0124].shar`, use

```
configure --with-complex
make complex
make clean
```

For including everything, which requires `meschach[01234].shar`, use

```
configure --with-all
make all
make clean
```

To compile the library in single precision, add the `--with-float` option to `configure` (with `Real` equivalent to `float`); e.g. use

```
configure --with-all --with-float
make all
make clean
```

Some Unix-like systems may have some problems with this due to bugs or incompatibilities in various parts of the system. To check this use `make torture` and run `torture`. In this case use the machine-dependent files from the `machines` directory. (This is the case for RS/6000 machines, the `-O` switch results in failure of a routine in `schur.c`. Compiling without the `-O` switch results in correct results.)

If you want to use the GNU `gcc` compiler, use the `configgnu` configuration script. This works just like the `configure` script, except that it will use `gcc` in preference to other compilers.

If you have problems using `configure`, or you use a non-Unix system, check the `MACHINES` directory (generated by `meschach0.shar`) for your machine, operating system and/or compiler. Save the machine dependent files `makefile`, `machine.c` and `machine.h`. Copy those files from the directory for your machine to the directory where the source code is.

To link into a program `prog.c`, compile it using

```
cc -o prog_name prog.c ...(source files)... meschach.a -lm
```

This code has been mostly developed on the University of Queensland, Australia's Pyramid 9810 running BSD4.3. Initial development was on a Zilog Zeus Z8000 machine running Zeus, a Unix workalike operating system. Versions have also been successfully used on various Unix machines including Sun 3's, IBM RT's, SPARC's and an IBM RS/6000 running AIX. It has also been compiled on an IBM AT clone using Quick C. It has been designed to compile under either Kernighan and Richie, (Edition 1) C and under ANSI C. (And, indeed, it has been compiled in both ANSI C and non-ANSI C environments.)

7.1.1 Installation on non-Unix systems

First look in the `machines` directory for your system type. If it is there, then copy the machine dependent files `machine.h`, `makefile` (and possibly `machine.c`) to the Meschach directory.

If your machine type is not there, then you will need to either compile "by hand", or construct your own makefile and possibly `machine.h` as well. The machine-dependent files for various systems should be used as a starting point, and the "vanilla" version of `machine.h` should be used. Information on the machine-dependent files follows in the next three subsections.

On an IBM PC clone, the source code would be on a floppy disk. Use

```
xcopy a:* meschach
```

to copy it to the `meschach` directory. Then `cd meschach`, and then compile the source code. Different compilers on MSDOS machines will require different installation procedures. Check the directory `meschach\machines` for the appropriate "makefile" for your compiler. If your compiler is not listed, then you should try compiling it "by hand", modifying the machine-dependent files as necessary.

7.1.2 makefile

This is setup by using the `configure` script on a Unix system, based on the `makefile.in` file. However, if you want to modify how the library is compiled, you are free to change the `makefile`.

The most likely change that you would want to make to this file is to change the line

```
CFLAGS = -O
```

to suit your particular compiler.

The code is intended to be compilable by both ANSI and non-ANSI compilers. To achieve this portability without sacrificing the ANSI function prototypes (which are very useful for avoiding problems with passing parameters) there is a token `ANSI_C` which must be `#define`'d in order to take full advantage of ANSI C. To do this you should do all compilations with

```
#define ANSI_C 1
```

This can also be done at the compilation stage with a `-DANSI_C` flag. Again, you will have to use the `-DANSI_C` flag or its equivalent whenever you compile, or insert the line

```
#define ANSI_C 1
```

in `machine.h`, to make full use of ANSI C with this matrix library.

7.1.3 machine.h

Like `makefile` this is normally set up by the `configure` script on Unix machines. However, for non-Unix systems, or if you need to set some things "by hand", change `machine.h`.

There are a few quantities in here that should be modified to suit your particular compiler. Firstly, the macros `MEM_COPY()` and `MEM_ZERO()` need to be correctly defined here. The original library was compiled on BSD systems, and so it originally relied on `bcopy()` and `bzero()`.

In `machine.h` you will find the definitions for using the standard ANSI C library routines:

```
/*--------------------ANSI C--------------------*/
#include        <stddef.h>
#include        <string.h>
#define MEM_COPY(from,to,size)   memmove((to),(from),(size))
#define MEM_ZERO(where,size)     memset((where),'\0',(size))
```

Delete or comment out the alternative definitions and it should compile correctly. The source files containing `memmove()` and/or `memset()` are available by anonymous ftp from some ftp sites (try archie to discover them). The files are usually called `memmove.c` or `memset.c`. Some ftp sites which currently (Jan '94) have a version of these files are `munnari.oz.au` (in Australia), `ftp.uu.net`, `gatekeeper.dec.com` (USA), and `unix.hensa.ac.uk` (in the UK). The directory in which you will find `memmove.c` and `memset.c` typically looks like `.../bsd-sources/lib/libc/...`

There are two further machine-dependent quantities that should be set. These are *machine epsilon* or the *unit roundoff* for double precision arithmetic, and the maximum value produced by the `rand()` routine, which is used in `rand_vec()` and `rand_mat()`. The current definitions of these are

```
#define MACHEPS 2.2e-16
#define MAX_RAND 2.147483648e9
```

The value of `MACHEPS` should be correct for all IEEE standard double precision arithmetic.

However, ANSI C's `<float.h>` contains `#define`'d quantities `DBL_EPSILON` and `RAND_MAX`, so if you have an ANSI C compiler and headers, replace the above two lines of `machine.h` with

```
#include <float.h>
/* for Real == float */
#define MACHEPS DBL_EPSILON
#define MAX_RAND RAND_MAX
```

The default value given for `MAX_RAND` is 2^{31}, as the Pyramid 9810 and the SPARC 2's both have 32 bit words. There is a program `macheps.c` which is included in your source files which computes and prints out the value of `MACHEPS` for your machine.

Some other macros control some aspects of Meschach. One of these is `SEGMENTED` which should be `#define`'d if you are working with a machine or compiler that does not allow large arrays to be allocated. For example, the most common memory models for MS-DOS compilers do not allow more than 64Kbyte to be allocated in one block. This limits square matrices to be no more than 90×90. Inserting `#define SEGMENTED 1` into `machine.h` will mean that matrices are allocated a row at a time.

7.1.4 machine.c

The core routines in `machine.c` as they presently are, are adequate on scalar processors. However, they are not designed to make best use of the recent super–scalar processors, or of vector processors. If you wish to make best use of these features of your machine in using the matrix library, then you should re-write these appropriately, possibly in assembly language. This has already been done to some extent, using "loop-unrolling":

```
sum0 = sum1 = sum2 = sum3 = 0.0;

len4 = len / 4;
len  = len % 4;

for ( i = 0; i < len4; i++ )
{
        sum0 += dp1[4*i]*dp2[4*i];
        sum1 += dp1[4*i+1]*dp2[4*i+1];
        sum2 += dp1[4*i+2]*dp2[4*i+2];
        sum3 += dp1[4*i+3]*dp2[4*i+3];
}
sum = sum0 + sum1 + sum2 + sum3;
dp1 += 4*len4;          dp2 += 4*len4;

for ( i = 0; i < len; i++ )
        sum += dp1[i]*dp2[i];
```

It may seem odd to use `dp1[i]*dp2[i]` instead `(*dp1++)*(*dp2++)` in the quest for speed, but optimising compilers cannot be trusted to do what you intend. The expression `dp1[i]*dp2[i]` was recognised for what it is, but `(*dp1++)*(*dp2++)` was not, by the RS/6000 optimising compiler. This may be a matter of taste by the compiler writers, so check it out on your own system before making any terminal decisions about what is fastest on your machine.

Also note that the `__zero__()` routine is defined from `machine.c`. This uses the `MEM_ZERO()` macro in `machine.h` in the standard release. However, if the double precision zero is **not** represented by a bitstring of zeros, the body of this routine would need to be replaced by

```
for ( i = 0; i < len; i++ )
    dp[i] = 0.0;
```

These are the only routines that need be modified, as essentially all other routines rely on these routines and on the `MEM_COPY()` macro, to provide adequate speed.

Such a re-writing effort may be worthwhile on, say, the i860 processor, where the speed of computing inner products in assembly (using special pipeline instructions) is an order of magnitude faster than general arithmetic operations. (See "Personal supercomputing with the Intel i860" by Stephen S. Fried, *Byte*, **16**, no. 1, Jan 1991, pp. 347–358 for an indication of possible performance.) Better use of the IBM RS/6000 super-scalar architecture has been obtained by re-writing some of the routines in `machine.c`. The speed of the core inner product routine on a 20MHz RS/6000 320 went from near the LINPACK rating of 7Mflops to about 20Mflops, half the theoretical peak speed of 40Mflops for a multiply and add each clock cycle.

7.2 Backward compatibility

As with any piece of software that is being modified, there is the problem of being able to use programs written for older versions of the library. This is especially important with Meschach 1.2 as the naming scheme has been made much more uniform and self-consistent. Names such as `get_vec()` (allocate vector) and `cp_vec()` (copy vector) have been changed to `v_get()` and `v_copy()` to be more consistent with `v_add()` (add vectors) and `m_mlt()` (multiply matrices).

The cost of this consistency is inconsistency with programs written for the older versions of Meschach. To deal with this, there is included in Meschach 1.2 a "compatibility" header file `oldnames.h`. Add the line

```
#include "oldnames.h"
```

at the beginning of files using pre-version 1.2 names. This header file consists of a collection of `#define`'s such as

```
#define get_vec    v_get
#define freevec    V_FREE
#define cp_vec     v_copy
```

The old iterative routines are still included in release 1.2a of Meschach (`pccg()`, `sp_pccg()`, `cgs()`, `sp_cgs()`, `lsqr()`, `sp_lsqr()`, `lanczos()`, `sp_lanczos()`, `lanczos2()`, `sp_lanczos2()`, `arnoldi()` and `sp_arnoldi()`). However, because of the new data structure for iterative methods, these are being phased out and can be replaced by the newer routines `iter_cg()`, `iter_spcg()` etc. The old iterative routines will not be supported in future.

7.3 Copyright

The copyright provisions for Meschach are intended to follow the lead of the Free Software Foundation in ensuring that the rights of people using and modifying the library cannot take away rights from others, while still enabling commercial use of the library. In that sense Meschach is not entirely "in the public domain". Notice that there is no intention to restrict the possible uses to which Meschach and parts of it are put, or to impede the work of programmers. The intent is only to make sure that users of any derivatives or modified versions of Meschach can still obtain access to the original code, and also to protect the reputations of ourselves and other programmers who modify or use Meschach.

Copyright subsists on the documentation and on the matrix library and source code for same and is held by David Edward Stewart and Zbigniew Leyk. It may be used free of charge provided the following rules are followed:

For legal purposes, in this section "the matrix library" shall refer to the "Meschach matrix library" as copyrighted by David Edward Stewart and Zbigniew Leyk.

1. Anyone to whom software is sold containing part or all of the matrix library in any form, whether modified or not, must have the matrix library source code made available to them in machine readable form at nominal cost.

2. Anyone distributing the library must ensure that copyright notices "Copyright (C) David E. Stewart and Zbigniew Leyk, 1986–1993" are published prominently along with the distribution in whatever form.

3. Anyone making changes to the library must prominently display this fact on any documentation relating to any use of the library (whether the use involves source or compiled code). Also, any such modification must be reflected in the routine `m_version()`, which prints out the current list of modifications to `stdout`.

4. Any code sold in object code form must include `m_version()` so that if the user so desires, he/she can determine what modifications and/or extensions to the original library have been made and who by.

Item (4) is deemed to be satisfied if there is a "version" command which executes the `m_version()` routine.

Finally, there is the usual statement about legal rights if something goes wrong in using the software. Trying to frame conditions under which Meschach can be guaranteed to work is unlikely to be a rewarding task for anyone to undertake, especially with the wide range of software and hardware systems it could work under. This is further complicated by the usual problems of numerical analysis where "proof of correctness" is not a realistic possibility and round-off errors are always present. Finally, due to the non-commercial nature of Meschach, there is unlikely to be any value to persons attempting to sue me for failure of the library in any situation.

Meschach IS PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, THE AUTHOR DOES NOT MAKE ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.