

Week 7 Notes

Adapted from the CAAM 420 Fall 2010 Notes of **Professor Symes**

September 26, 2011

Contents

1 Remark on GSL in CAAM 420	2
2 Document Typesetting with \LaTeX	2
3 Fortran	4
4 Old Class Assignment	6

1 Remark on GSL in CAAM 420

Old course comments in this section

May not have been clear from last week: I have installed GSL in my home dir on CLEAR. You may use the include (header files) and lib (library) directories from this installation - you do not need to download and install GSL yourself to do the work in this course, so long as you do your work in CLEAR.

The makefile mex4 shows exactly how to do this, and you can modify and use it for the next homework, or whenever you want to work with GSL on CLEAR (at least during this semester!).

That said, I do not want to discourage you from installing GSL either - it's not that big, and its install procedure is perfectly typical of well-designed Un*x installs. If you want to do your own install, don't hesitate to ask either me or the TAs for help. Of course, if you want to work with GSL on your laptop, for example, you'll need to install it yourself.

2 Document Typesetting with L^AT_EX

And now for something completely different...

LaTeX is a typesetting system for scientific publication. Based on Donald Knuth's TeX specification. It's different from word processing tools such as Microsoft Word in that

- (1) it does not "what you see is what you get" - you enter the source text using any editor, and the LaTeX compiler turns it into a portable graphical format of some kind, such as pdf;
- (2) it does a respectable job of mathematical formulae, and in general a provides far better typography than Word. While many scientific papers - and even books - are set in Word, TeX variants (LaTeX, Plain Tex, AMSTeX) see increasing use in many fields - in mathematics, TeX is used almost exclusively since the early 90's.

Besides the two fine reference books listed in the course bibliography, there are a number of good online tutorials. I'll use Getting Started with LaTeX by David R. Wilkins in class.

A couple of things that these resources don't cover:

- how do you actually turn the nice source file into a viewable document: there are a number of options, depending on platform. On CLEAR and similar Linux platforms, you are likely to find the pdflatex command, which takes your filename as argument. I've provided a simple tex file which illustrates many of the points presented in Wilkin's notes:

```
lex1.tex


---


1 \documentclass[12pt]{article}
  \usepackage{fullpage}
3
  \begin{document}
5
  % this is generic stuff - particular journals will have their
7 % own style files for titles, addresses in footnotes, etc.
9 \title{A LaTeX Document}
  \author{William W. Symes}
11
  \maketitle
13
  \begin{abstract}
15 This is an abstract, in fact a very abstract abstract. Such exercises
```

```

17 in abstraction are extremely important, as most of your readers will
read the abstract first (if indeed they read anything at all!).
\end{abstract}
19
\section{Introduction}
21 The introduction introduces.

\section{Another Section}
23 It should contain some material, for example a formula:
25 \begin{equation}
\label{ha}
27 x=y
\end{equation}
29
\begin{equation}
31 \label{my-eqn}
f(x,k,\alpha) = \int_{-\infty}^{\infty} g(x)e^{ikx-\alpha x^2} dx,
33 \end{equation}
and possibly some inline discussion of math, like  $f(x,k,\alpha)=0$ .
35
\section{Yet Another Section}
37 The discussion may refer to previous material, such as equation
(\ref{my-eqn}), and possibly even some computer code:
39 \begin{verbatim}
int quad(double x, double alpha, double k, double (*g)(double)) {
41     int i;
double dx;
43     int nx;
...
45 }
\end{verbatim}
47
\section{Conclusion}
49 A bibliography should follow, but that's a topic for another day.

51 \end{document}

```

- You can compile it as follows:

```
$ pdflatex lex1.tex
```

- The result is lex1.pdf, which you can view with Acrobat Reader in any of its forms.
- A number of integrated text-processing environments exist, and many people are fond of them. I use TexShop on Macs (get it through Fink), and similar products (mostly free) exist for Linux and Windows. However the basic combination of (1) an editor, (2) a latex or pdflatex command, and (3) some kind of viewer for the graphical output files, is still a very viable way to go.
- Several useful abbreviations exist, for example: ... for ..., ... for
- Debugging a LaTeX document can be as tedious as other forms of debugging - I will cause a couple of LaTeX busts in class, of the easier variety.

This week's homework includes a very simple exercise to get your LaTeX feet wet (does that qualify as rubber boots?).

3 Fortran

This course will do less with Fortran than preceding editions, continuing a long-running trend. However I do want you to be aware of Fortran, and of the large fund of high-quality numerical software written in that language. The course goal is that you are able to write code to organize your problems, get data from files or the terminal or wherever it needs to be got, call the necessary functions (usually library code written by others, but, failing that, your own) to process this data as needed, then get the output into usable form - that is, so that you can carry out common scientific computing tasks. The contemporary lingua franca in high-performance computing is C/C++, so we are concentrating on that language family. However, access to code in other language families can be very useful, and Fortran is Exhibit A.

Fortran is the oldest computer language in widespread use to express scientific programs. The first standard for the language appeared in the 1950's, and it continues to evolve today. The vast preponderance of high-quality scientific library software is written in Fortran. Even though an increasing number of libraries are written in other languages, principally C and C++, the versatile early-21st-century scientific programmer must still be comfortable with Fortran.

The two versions of the language in widespread use today are Fortran 77 and Fortran 90 (and its minor update, Fortran 95). Most compilers available today handle source from both languages (Fortran 77 being a subset of Fortran 95, this is entirely feasible). The Gnu project has produced a freely-available compiler, gfortran, which is available on CLEAR (and which you can download and install yourself, if need be). Please note that many other vendors provide Fortran compilers (Sun, Intel, HP, IBM,...) and that for historical reasons these tend to exhibit more incompatibilities than do C compilers - standards exist for both f77 and f9x, but f77 developed and was widely used before the adoption of an official standard, so it's occasionally violated or (more often) extended in significant ways.

A newer standard (Fortran 2003) exists, which specifies many modern language features (object orientation, code templates,...). However compilers are hard to come by at the moment, and no library code to speak of exists.

In fact, most numerical software written in Fortran is written in f77. For this reason, I concentrate entirely on the f77 subset in these notes, as does the recommended text by Etter. Many books cover f90/f95; most that I have examined are reasonable sources of information on the larger language.

Functionally, Fortran is a subset of C. Most common numerical tasks are as easily accomplished in Fortran as in C, and some are more easily accomplished: Fortran's syntax for multidimensional arrays is much more natural and easier to use than C's, and complex numbers are an intrinsic type with efficiently compiled instructions - to name two prominent examples. On the other hand system level tasks are often much harder or impossible. The principal reason is the Fortran does not permit direct manipulation of pointers (i.e. variables holding addresses), an extremely powerful (and dangerous!) feature of C.

This section of notes is mostly concentrated in the comment sections of several examples:

- Example 1: overviews the basics of f77 syntax, and shows how to produce screen and file output, also how to pass data via disk files and system calls to external programs.
- Example 2: shows how to call a Fortran subroutine from a C driver, which will be your usual interaction with Fortran. Duplicates functionality of Example 17 - defines a matrix and a vector (as simple arrays, the matrix in column-major), then passes them to a Fortran subroutine matvec.f which multiplies them and prints the output. Shows how to pass basic variables (ints, floats,...), arrays, and multi-dimensional arrays to Fortran. The basic principle is: everything is a pointer - Fortran implements pass-by-reference. Arrays are stored column-major (or, for higher-dimensional arrays, with the fastest index being the first), so that's how you must arrange them (NOT as multi-d arrays in C!!!).

The driver code for this example is here:

```
1 /*
   * Author: WWS
3  * Purpose: illustrate conventions for accessing Fortran subroutines
   * from C.
5 */
```

```

7 #include <stdio.h>

9 /* declaration of the Fortran matvec subroutine – note that all
11 arguments are pointers, and the name is “mangled” by appending
an underscore – this is what the compiler produces in its output
object code, and what the linker expects (see nm output).
13 */
void matmult_(int*, int*, float*, float*, float*);
15
int main() {
17
18     const int ncols = 5;    /* number of columns */
19     const int nrows = 5;   /* number of rows */
20     int nr, nc;           /* nonconst versions */
21     int i, j;             /* loop counters */
22     float a[nrows*ncols];  /* col-major storage */
23     float b[nrows];       /* output vector */
24     float x[ncols];       /* input vector */
25
26     /* assign matrix entries – loop over rows, cols
27 * with loop scopes explicitly indicated by braces
28 * NOTE: this matrix is the famous HILBERT MATRIX,
29 * 5x5 instance
30 */
31     for (i=0; i<nrows; i++)
32         for (j=0; j<ncols; j++)
33             a[i+j*nrows]=1.0f/((float)(i+j+1));
34
35     /* print matrix */
36
37     for (i=0; i<nrows; i++) {
38         for (j=0; j<ncols; j++) printf(“a(%d,%d)=%12.4e “, i, j, a[i+j*nrows]);
39         printf(“\n”);
40     }
41     printf(“\n”);
42
43     /* define x (input vector), print */
44     for (j=0; j<ncols; j++) {
45         x[j]=j+1;
46         printf(“x(%d)=%12.4e\n”, j, x[j]);
47     }
48     printf(“\n”);
49
50     /* multiply x by a, store in b, print result
51 * Three things to notice:
52 * (1) everything argument is passes as a pointer – a, x, and b
53 * are already pointers, so you don’t have to do anything, but nrows
54 * and ncols are not
55 * (2) the matrix is simply passed by pointer – Fortran expects
56 * column-major storage
57 * (3) the subroutine name must have an underscore appended to be
correctly interpreted by the C compiler

```

```

59  *
60  * Note that passing the const ints generates a compiler warning -
61  * to get rid of it, use non-const variables with same values - this
62  * happens because Fortran does not understand const args.
63  */
64  // matmult_(&nrows,&ncols,a,x,b);
65  // /*
66  nr=nrows;
67  nc=ncols;
68  matmult_(&nr,&nc,a,x,b);
69  // */

71  /* print out the result */
72  for (i=0;i<nrows;i++) {
73      printf( 'b(%d)=%12.4e\n', i, b[i] );
74  }
75
76  return 0;
77 }

```

This example shows how Fortran subroutine (and function) names must be handled from C: the C linker sees the symbols in the symbol table part of the .o file directly, rather than through a translation step. On most systems that you will encounter, Fortran compilers add an underscore to subroutine and function names when generating the symbol table. Thus the C source must refer to the Fortran subroutine foo as “void foo_(...)”.

4 Old Class Assignment

Problem 2 this week asks you to set up and solve a system of linear equations, using GSL. The relevant sections in the manual is BLAS Support - stands for “Basic Linear Algebra Subroutines”, which were originally written in Fortran but are now widely available in C implementation, also in vendor-optimized libraries for various platforms - and Linear Algebra. We won’t worry about optimization at the moment, and will use the CBLAS implementation packaged with GSL.

First we will review the solution of last week’s class exercise. Then we’ll create a program which

- Reads a matrix A from the file data structure created by last week’s exercise,
- Creates a GSL vector x
- Multiplies the matrix by the vector to create a “right hand side” vector b, using gsl blas_dgemv
- Solves the system $Ax=b$ by calling LU decomposition and fwd/backsolve from the GSL Linear Algebra Package