

1 Writing C/C++/Fortran programs on SUN workstations

Added Spring 2007:

If you have any programming experience, you can safely skip this section, otherwise read on. Solution of every problem in the homework will follow the same routine:

The C++ version of Numerical Recipes is now installed on the departmental Linux machines (e.g. argon and krypton - please don't use concorde for heavy computing) in subdirectory /usr/site/numerical-recipes (and note that it's a dash between the numerical and recipes, not an underscore). Also, the C and FORTRAN versions are 2.11, the latest

1. Planning the overall structure of the program
2. Creating a text file with the C/C++/Fortran code using a text editor
3. Compilation
4. Presentation of the results

1.1 Editing

There is large number of text editors available on SUN stations, and you can use any of them: emacs/xer all of them can be customized to make coding easier by syntax highlighting, automatic indenting etc. In order for all these language-specific features to work, the editor must "know" the language of the file you are typing. It means that the features will work after you save the file with appropriate extension (filename.c, filename.cpp, filename.f etc).

- **emacs:** To turn on the syntax highlighting and experiment with other useful options, go to the Options in the Menu Bar, and check appropriate boxes. You can make your preferences permanent by selecting *Options* → *Save Options*. Pressing TAB will justify the current line in accordance with default indentation style, which is more than adequate.
- **nedit:** This is an editor that by default will be easiest to use for MS Windows users. To permanently turn syntax highlighting on, check

Preferences → *Default Settings* → *Highlight Syntax*.

- **vi/vim/vgim** If you are using vi, you probably don't need any instructions. A better version of vi called vim (Vi IMproved) is available at

/home/ashot/bin

The details are available at www.vim.org.

- **jed/wjed** My favorite. If you are used to emacs, you might want to try this editor. It can emulate several editors such as emacs, vi, and others. The distinct advantage of the JED is that it is very compact and fast. In addition, it comes with several syntax highlighting modes with well chosen color schemes. As with emacs, you can adjust current line by typing TAB.

- **MS Windows editors** There is a large number of editors having syntax highlighting, auto-indenting, parentheses checking, etc, for MS Windows. Any of them is just fine, but **please make sure** that if you are using text *processor* (rather than *editor*), such as MS Word, you save your file as *Text only*. The second point of concern arises only when you transfer files back and forth between Unix and Windows. The two operating systems use different conventions on how each line of a file is terminated. Unix uses a carriage return \r character, while Windows uses both carriage return \r and a newline characters \n. As a result, you have to remove or add newline character after downloading the files by executing

```
dos2unix filename or unix2dos filename
```

after you copy the file from Windows to Unix, or from Unix to Windows respectively.

2 Compilation

After you created a text file containing the source code written in C/C++/Fortran, you need to translate it into machine code (also called executable code) which can be executed by a computer. The process, called compilation is performed by using a special program, called compiler. There are several compilers available on SUN workstations: the SUN's native compiler called CC, and a family of GNU compilers: gcc (for C code), g++ (for C++), and g77 (for Fortran). The basic usage is always the same. To compile a file called problem1.c you simply execute

```
gcc problem1.c
```

The result is a file called a.out. You can rename the file if you don't like the name. You can also specify the output file name using -o option:

```
gcc problem1.c -o problem1
```

will produce an executable file named `problem1`. The above will work for self-contained programs that do not use functions or sub-routines from external libraries. You will notice that bare C (and to a lesser extent Fortran) have a small number of commands. For example, C has no idea how to calculate $\sin(x)$ or how to read data from a file. You access math, input-output, string manipulation and many other functions using numerous libraries of functions. There are several *standard* libraries, which come with any compiler, and compiler knows where to look for them. The library of functions for Numerical Recipes is a non-standard library.

When using library functions, you need two types of files. The header files, which have `.h` ending, contain only declarations of functions. They specify the type of the function, the number and the types of its arguments, etc. For instance, if you use $\sin(x)$, declared in `math.h` you simply include the following line in the beginning of your source file:

```
#include <math.h>
```

It is important to realize that the header files contain only declarations of functions, but not the actual code implementing the functions. For instance the header file `math.h` contains the declaration

```
double sin(double x);
```

but information of how to compute $\sin(x)$. The header files are needed for the compiler to check the consistency of your code. For example if by mistake you tried to use `sin(2.3, 4.5)`, the compiler would generate an error message, because it knows that $\sin(x)$ must have a single argument.

The actual code used to compute $\sin(x)$ or other functions is located in a precompiled “library” files. According to the naming convention in Unix, the library files usually have names starting with `lib` and ending with `.a`. For example, the frequently used library containing simple **math** functions, which are defined in `math.h`, is called `libm.a`. The C and Fortran versions of numerical recipes library are called `librecipes_c.a` and `librecipes_f.a` respectively.

When using a standard library (such as math library), all you need to do is to add `-lm` switch in the compilation line:

```
gcc problem1.c -o problem1 -lm
```

Here is how `-lm` switch is formed: if the library filename is `libNAME.a`, then `lib` is abbreviated to `-l`, and `.a` is omitted. Thus, `libm.a` becomes `-lm`. Similarly, `librecipes_c.a` becomes `-lrecipes_c`.

When using a non-standard library installed on your computer, compiler has no idea where the library files and the header files are, unless you explicitly specify the location of the library and the header files during compilation. This is how you do it in general:

```
gcc myfilename.c -o myoutputfilename -I IncludeDir -L LibDir -llibRARYname
```

 where `IncludeDir` is the directory con-

taining the header files, and `LibDir` is the directory containing the libraries.

Compilation of program (let us assume you named it `test.c`) using Numerical recipes is done by

```
gcc test.c -o test -I/usr/site/numerical_recipes/recipes_c-ansi/include -L/usr/site/numerical_recipes/lib-lrecipes_c -lm
```

As an exercise, look into directories in the line above. What are the files you see? Which file `-lrecipes_c` refers to?

3 Making life easier

3.1 Quick solution

Needless to say, entering the long compilation command above can be frustrating if you need to do it 100 times. Fortunately, there are many ways to simplify the process. The quick and dirty solution is to create an alias. For instance, if you use `csh` or `tcsh` as your command shell, you can insert the following line

```
alias nrcc 'gcc \!* -I/usr/site/numerical\_recipes/recipes\_c-ansi/include -L/usr/site/numerical\_recipes/lib -lrecipes\_c -lm'
```

anywhere in your `.cshrc` file. The text *must* be entered as a single line, otherwise you will receive error messages.

Next time you start your shell (or after you type `source ~/.cshrc` to reread your initialization file `.cshrc`), you will be able to compile `test.c` by doing

```
nrcc test.c
```

3.2 make command

A better solution is to use `make` command and Makefiles. You would have to invest a little time to understand how they work, but it's worth the effort. This method will be briefly explained in the class next time.

4 Presenting your results.

Your homework solution for each problem should generally have the following structure:

1. One line indicating the location of your source files. The privileges should allow me to read the files. Although this is not required, I suggest the following directory tree for your NR homeworks:

```
/home/username/  
/home/username/mathmethods/  
/home/username/mathmethods/hw1  
/home/username/mathmethods/hw2  
/home/username/mathmethods/hw3  
...
```

Please, do NOT include printouts of your source files.

2. A *brief* (1-2 paragraphs) description of *how* you solved the problem. For a typical problem. I will be able to read your code if I have additional questions.
3. The most important part: your results and comments. Here, you can use a large selection of software available to present the results in the most clear and crisp form. If you decide to make a plot to display a large array of data, please include the location of the data file. The best program to produce publication quality plots, in my opinion, is `xmgrace`. The documentation and tutorials are available at

<http://plasma-gate.weizmann.ac.il/Grace>

A similar freeware program for MS Windows, with somewhat reduced functionality is called `prestoplot` and can be downloaded at

<http://www.mbg.cornell.edu/shalloway/jason/presto.html>

Of course, Mathematica, IDL, and many other packages can be used to present your results. The use of Mathematica is allowed and encouraged for these purposes. However, using Mathematica functions such as `NSolve[...]` to solve a homework problem is not allowed, since use of this and similar functions defeats the purpose of the course. You should be able to understand and *write* yourself functions similar to `NSolve`. This being said, it is perfectly fine to use Mathematica or other software packages, books, tables, etc. at the intermediate stages of your solution to check yourself and verify that your code is correct.