Appendix A

Programming utilities

A.1 RatFor == RATional FORtran

Up to now **Fortran** is our most universal computer language for computational physics. For general programming, however, it has been surpassed by **C**. "**Ratfor**" is Fortran with C-like syntax and combines uncluttered, free-form programming with the power of clearly displaying mathematical formulas and algorithms. Ratfor accepts constructs that are possible in any of the Fortran dialects (Fortran IV, 66, 77, 90 and HPF). As illustrated in chapter **??**, I used Ratfor90 to implement an entire tool box of wave equation modules. My choice of Ratfor90 is dictated by the need for computational efficiency of parallel computers (thus Fortran90 or HPF) and the wish to actually include the code of algorithms in this thesis. The Ratfor preprocessor is now, usually included in the Fortran compiler environment, but a public domain version is also available. [1]

A.1.1 General ratfor language features

The following paragraphs are reprinted from Jon Claerbout's Ratfor feature summary, published in *Earth Sounding Analysis: Processing versus Inversion* (**?**) and *Applications of Three Dimensional Filtering* (**?**):

> Ratfor was invented by the people who invented C. Ratfor programs are converted to Fortran with the Ratfor **preprocessor**. Since the preprocessor is publicly available, Ratfor is practically as universal as Fortran. [Ratfor was invented at AT&T (**?**), which makes it available directly or through many computer vendors. The original Ratfor transforms Ratfor code to **Fortran** 66. See `http://sepwww.stanford.edu/sep/prof` for a general public-domain Ratfor translator.]

[1] `available from http://sepwww.stanford.edu/sep/prof`

> You will not really need the Ratfor preprocessor or any precise definitions if you already know Fortran or almost any other computer language, because then the Ratfor language will be easy to understand. Statements on a line may be separated by ";". Statements may be grouped together with braces { }. `Do` loops do not require statement numbers because { } defines the range. Given that `if( )` is true, the statements in the following { } are done. `else`{ } does what you expect. We may *not* contract `else if` to `elseif`. We may always omit the braces { } when they contain only one statement. `break` will cause premature termination of the enclosing { }. `break 2` escapes from {{ }}. `while( ) { }` repeats the statements in { } while the condition ( ) is true. `repeat { ... } until( )` is a loop that tests at the bottom. A looping statement more general than `do` is `for`(initialize; condition; reinitialize) { }. An example of one equivalent to `do i=0,n-1` is the looping statement `for(i=0;i<n;i=i+i)`. The statement `next` causes skipping to the end of any loop and a retrial of the test condition. `next` is rarely used, but when it is, we must beware of an inconsistency between Fortran and C-language. Where Ratfor uses `next`, the C-language uses `continue` (which in Ratfor and Fortran is merely a place holder for labels).
>
> The Fortran relational operators `.gt.`, `.ge.`, `.ne.`, etc. may be written >, >=, !=, etc. The logical operators `.and.` and `.or.` may be written & and |. Anything from a # to the end of the line is a comment. Anything that does not make sense to the Ratfor preprocessor, such as Fortran input-output, is passed through without change.

Ratfor has an `include` statement, which includes the specified filename. In the source code abbreviations are possible by using the `define` statement. As Claerbout describes:

> Indentation in `Ratfor` is used for readability. It is not part of the `Ratfor` language. One may choose one's own style.

A.2 SPP – A general CPP like preprocessor

CPP is the standard preprocessor that accompanies any C compiler and is designed to work on directives in the C language. It recognizes the symbols `/* */` as comments and the symbol # as preprocessor directive. It is a powerful tool in writing clean and general code.

Similar preprocessors exist for other languages, such as Fortran, but I found many vendor dependent inconsistencies. Since I wanted to design a portable code tool box, I decided to write my own preprocessor **SPP** whose features include those of CPP, and which would perform identically on any machine. I implemented it in the "Perl" language, which is a superset of `awk, grep, sed and C` and a powerful tool. Perl is public domain and available on most computers.

The basic feature of SPP [1] is the specification of an arbitrary comment symbol and directive symbol. Its syntax and features also include those of CPP. Following is the manual page that accompanies SPP.

```
CPP(1)                   USER COMMANDS                    CPP(1)
NAME
      spp - SEP's ANY language preprocessor

 SYNOPSIS
      spp [ -Ccommentsymbols ] [ -strict ] [  -f77  ]  [  -H  ]  [ -undef ]
           [ -Dname ] [ -Dname=def ] [ -Idirectory ] [ -Uname ]
           [ -Ydirectory ] [ <input-file [ >output-file ] ]

 DESCRIPTION
      spp is the ANY language language preprocessor.
      SPP is a preprocessor that differs from CPP only in that
      specficiation of arbitrary comment and directive symbols
      are possible (e.g., %#$define, instead of #define).
      So it can be used for many different programming languages.
      Although spp can be used as a macro processor, this  is  not
      normally  recommended. For general-purpose macro-processing,
      see m4(1V), and the chapter on m4 in  Programming  Utilities
      and Libraries.
      spp  redirects  standard  in  and  out.   <input-file   and
      >output-file


 OPTIONS
      -C   Define what the comment symbol is (# for ratfor, c  for
           f77  etc)  that  contains  spp directive lines. In this
           manual page  the  the  usual  #  directive  symbols  is
           assumed   to  epxlain  the  syntax  and  features.   By
           default, spp leaves in all commment lines.
      -strict
           Forces comment symbol and processor  directives  to  be
           adjacent (like #ifdef), then all lines like (#   ifdef)
           are treated as comments.
      -f77 Run the preprocessor in Fortran 77 mode,  it  correctly
           handles lines longer than 72 columns....
```

[1] SPP is available from ftp://sepftp.stanford.edu/pub/SPP