Chapter 1: Introduction

Hello, and welcome to "Everything I wanted to know about Fortran, but was afraid to ask." This seminar operates as an introductory, crash course in Fortran for applied mathematicians. Though you may encounter that Fortran is seen as antiquated by those in industry, know that Fortran is a fast and efficient language largely used in the applied math and the scientific computing community. The advantages of Fortran are especially prevalent in vector operations.

The name Fortran comes from FORmula TRANslation. That is, the language was originally developed for easy implementation of mathematical formulas, vector and matrix operations in particular. The flexibility of Fortran to natively handle arrays makes your life much easier when coding basic routines, like matrix vector products, to more advanced routines like linear solvers or conjugate gradient.

This introduction serves as an outline for the different topics that are to be covered in this seminar. Know that this seminar will not include all aspects of Fortran, but will be enough to get you up and running, solving problems, and coding with organization. The seminar is structured to learn Fortran through example, with commentary along the way. We will color code the discussion of the examples in the following way:

- Program structure components are red.
- Variable declarations are green.
- Intrinsic Fortran functions (like sine) are light blue.
- Comments are gray.
- Numbers, written output, and formatting are pink.

We also use a set of style conventions to make example codes in the notes easier to read. ***This does not affect whether the code compiles, but will make the code more readable and assist in debugging***. Loops, if statements, and nested control sequences are indented with three spaces. The indentations help identify when each event opens and closes. Also, we will capitalize Fortran control structures and variable declarations. We write variable names in (mostly) lower case and, since variable names in Fortran 90 can be as much as 32 characters in length, will make the names of variable as descriptive as possible to make the code easy to read/debug. It is important to always remember: ***Fortran is not case sensitive, so the variables with the name*** a ***and*** A ***are the same, as far as the compiler is concerned***.

In Chap. 2, we begin with a couple introductory examples of Fortran programs. This chapter also introduces how the code examples in this class are set-up. For the most part, examples are first introduced as psuedocode which is then translated into Fortran code that can be compiled. The ability to inspect pseudocode and translate it into working Fortran code is one of the main goals of this seminar.

Chap 3. will outline in depth the components of a Fortran program. This includes the data types available to a program, the control sequences we often use (like loops), and how to read and write information from the terminal or a file. We will then provide an example of a quadrature routine to use some of the new program components at our disposal.

Next, in Chap. 4 we examine how to organize a Fortran program. To keep programs from becoming bloated and hard to debug we split the code into functions and subroutines. This includes introducing some of the intrinsic functions available in Fortran as well as external functions, i.e., functions that are written by the user. We then show how to pass an external function to another function or program. Next, we introduce modules, an important concept that allows one to collect similar functions and subroutines in a single source file. Finally, we outline a few options of how to compile these multiple source files, as the organization techniques will produce multiple Fortran source files.

In Chap. 5 we demonstrate the ease with which Fortran handles arrays. This includes how to manipulate data stored in arrays, intrinsic functions, printing, and how to pass arrays to functions or subroutines.

Chap. 6 introduces object oriented programming in Fortran using modules. To do so we introduce derived types and improve the usability code contained in modules using interfaces.

In Chap. 7 we cover some of the more advanced features of Fortran. This includes overloading operators (using interfaces), dynamic arrays, optional input arguments in functions and subroutines, formatted file I/O, and recursive functions.

Chap. 8 introduces various data structures, with the emphasis on understanding the strengths and weaknesses of each type to choose the best one for the problem at hand.

Finally, in Chap. 9 we introduce source code management using `git`.

1.1 Getting Started

To get up and running to code in Fortran we'll need two things

1. A Fortran compiler.
2. An environment to edit source files.

You can use any compiler you want, but in class I will use the gfortran compiler. The gfortran compiler is a free open source compiler than can be found on pretty much all versions of Linux, Unix, and in cygwin for Windows. The binaries are available at http://gcc.gnu.org/wiki/GFortranBinaries along with installation instructions. If you're using Linux you can open a terminal and type the command

```
sudo apt-get install gfortran
```

to install the binaries for you. For Mac this website https://sites.google.com/site/dwhipp/tutorials provides an in depth tutorial of installing the gfortran compiler.

Now that we have a compiler installed we need a way to edit source code. I don't mind what program or tool you use, whatever your preference. There are developer tools available (like XCode on Mac) or the more barebones vim and emacs. However, any text editor works just as well (like gedit on Linux, TextEdit on Mac, or notepad on Windows).

With a compiler and your preferred method for source code creation and editing we are ready to learn Fortran!