1 LIBRARIES

1.1 Why use libraries?

In the most simple case, libraries are made up of general code that may be referenced/used by many unrelated programs. This way, you can remove the need to duplicate code by maintaining a single, multipurpose library. Other, more specialized, uses exist as well.

1.2 Types of libraries

There are two types of libraries I will be focusing on:

1. Static libraries (.a [Linux/OSX], .lib [Windows]) : All of the library code is included in the compiled program. Programs do not require the library to exist in the same location at all times.

2. Shared libraries (.so [Linux], .dll [Windows], .dylib [OSX]) : Library code is referenced from the compiled program at runtime. Programs required that a library exist in the same location at all times, but that library may change.

1.3 Creating a library

I will be focusing on compiling libraries using FORTRAN and gfortran on Linux. These examples may be generalized to other languages and gnu compilers (e.g., gcc). Look up documentation for any other compilers/languages.

To generate a static library

1. Compile all files using the -c flag (producing $(FILE).o)

2. Use the command ar with options crs (create, insert/replace, index) to generate the static library.

or, in code

```
gfortran -c $(FILES).f
ar crs lib$(LIBRARY).a $(FILES).o

gcc -c $(FILES).c
ar crs lib$(LIBRARY).a $(FILES).o
```

this will produce the file lib$(LIBRARY).a to be linked later. To generate a shared library you only need to call gfortran using the -shared flag. In code,

```
gfortran -shared $(FILES).f -o lib$(LIBRARY).so
gcc -shared $(FILES).c -o lib$(LIBRARY).so
```

noting that library names start with lib by convention. Of course, any other options may be included in these calls.

1.4 Linking libraries

If you are using a static library it is possible to simple include it as a file when you compile your code

```
gfortran -o $(OUTPUT) $(CODE) $(LIBRARY)
gcc -o $(OUTPUT) $(CODE) $(LIBRARY)
```

however, the more conventional way to include libraries is to link them. This is done through the -L, which points to the directory containing the libraries, and -l flags, which has the library names.

```
gfortran -o $(OUTPUT) $(CODE) -L/path/to/library -l$(LIBRARY)
gcc -o $(OUTPUT) $(CODE) -L/path/to/library -l$(LIBRARY)
```

where the "lib" portion of the library name may be dropped. However, in order to run code using shared libraries you must ensure that the directory is included in the dynamic linker path. This is done by including it in LD_LIBRARY_PATH as so

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/library
```

* In C you can also use the #include keyword to header files. Here, the notation ¡...¿ will look in a predefined path and "..." will look in the same directory as the source file.

2 LAPACK

A widely-used and maintained example of a library is LAPACK (Linear Algebra PACKage) which provides routines to perform common linear algebra problems. The user guide can be found here:

   NetLib: LAPACK User Guide

2.1 Installing LAPACK

2.1.1 From scratch

Like most libraries, LAPACK's user guide describes how to install the library. To install LAPACK use the commands

```
wget "http://www.netlib.org/lapack/lapack-3.8.0.tar.gz"
tar -xvzf lapack-3.8.0.tar.gz
cd lapack-3.8.0
cp make.inc.example make.inc
make
```

and the provided Makefile will handle the rest.

- For specific architectures or requirements make.inc may be edited to suit your needs.

- An error may occur during the testing portion of the Makefile, this will not affect your use of the package. If you want to run all tests without this error you may run the command "ulimit -s unlimited" and the error should no longer appear.

2.1.2 Ubuntu

Alternatively, LAPACK can be added using the package handler (apt/apt-get for Ubuntu) through

```
sudo apt-get install libblas-dev liblapack-dev
```

which will install the libraries to the usual place.

2.1.3 MacOS

If you are running MacOS these libraries are provided already and can be included by compiling with the flag "-framework Accelerate". E.g.,

```
gfortran program.f90 -framework Accelerate
```

However, if needed, you can explicitly install the libraries with Homebrew using the following

```
brew install openblas
brew install lapack
```

2.2 LAPACK Naming Conventions

The LAPACK routine names may seem cryptic at first glance; however, they follow a naming scheme which may be split into 3 parts:

- Part X : Data type

  S :   Real (Single precision)
  D :   Double precision
  C :   Complex
  Z :   Complex*16

- Part Y : Matrix type. Many types are supported, allowing for efficient calculations. Ensure you use the correct form for your problem

  GE :   General form
  GB :   General band
  BD :   Bidiagonal
  DI :   Diagonal
  HE :   Hermitian
  Etc.

- Part Z : Computation to perform. There are a large number of these, so the user guide should be used as reference. Here are a couple useful examples

  EV :   Calculates eigenvalues and, optionally, either left or right eigenvectors of a matrix.
  ES :   Calculates eigenvalues and, optionally, the Schur form of a matrix.
  SV :   Computers the solution to a system of linear equations $A * X = B$
  Etc.

These names are then constructed by concatenating each part, XYZ. E.g., CGEEV calculates the eigenvalues/eigenvectors (EV) for complex (C), general form (GE) matrices. A complete list can be found online at the LAPACK Index

2.3 Using LAPACK routines

While some routines have similar arguments, I would suggest that you use the LAPACK Index entry for your specific routine to make sure you use it correctly.

2.4 Compiling with LAPACK

As with other libraries, LAPACK can be included in one of two ways:

1. By embedding the static library file when compiling

   ```
   gfortran -o $(NAME) -f $(FILE) $(ARCHIVE)
   gcc -o $(NAME) -f $(FILE) $(ARCHIVE)
   ```

2. By linking the library when compiling

   ```
   gfortran -o $(NAME) -f $(FILE) -L/path/to/library -llapack -lrefblas
   gcc -o $(NAME) -f $(FILE) $(ARCHIVE)
   ```

A few Fortran programs have been included to show how this is compiled and used:

- testSGESV.f90 solves a system of equations of a real-valued, general, matrix

- poisson-student.f solves a differential equation using the centred difference scheme for the derivative.

- testQM.f90 takes in some magnetic field and determines the eigenvalues and eigenvectors of a simple Hamiltonian