CS 404: Lab 1: Finding Libraries

Getting Started

1. Boot into Linux. To do this, click on "shutdown" and then select "restart." The computer will begin restarting. You will eventually have the option to select either Win2K or Linux (use up and down arrows to switch, press enter to select).

2. Open a web browser and go to the course website[1]. Download `Fpca.tar`.

3. To unpack the tar-file, type `tar xvf Fpca.tar` in a terminal window. Cd into the directory "Fpca."

BLAS and LAPACK

1. Fpca contains three FORTRAN files (main.f, subs.f, and system.f) and a Makefile. Open these files in an editor and have a look. The three code files implement a statistical technique known as "principal component analysis" (PCA). The goal of PCA (also known as "empirical orthogonal functions" analysis) is to reduce the dimensionality of the data set. Specifically, PCA determines from multiple observations of k variables how the variables are related and assigns each variable a weight. By scaling each variable by its weight and adding them together, we can produce a one-dimensional approximation of our k-dimensional data. This approximation, known as the "first principal component" or "leading mode" represents a pattern common in several of the variables, and hopefully, it explains a large percentage of the total variance in the system. Although we're typically most interested in the leading mode, it is possible to partition the variance among additional modes, each representing successively less variance.

   Although PCA sounds complicated, it is simple to compute using tools from linear algebra. Given an array of data, C with each column representing a variable and each row representing a sample, we first need to compute the covariance matrix Cov. The covariance matrix is defined

[1] www.cs.cornell.edu/Course/cs404/2002sp

using matrix multiplication:

$$Cov = \frac{1}{m-1} C^T C$$

where m is the number of observations and the superscript T indicates the transpose of the matrix. We then need to compute the eigenvectors and eigenvalues of Cov. The eigenvectors are the principal components (the weights), and the eigenvalues indicate the amount of variance explained by each component.

The PCA algorithm is implemented in the three FORTRAN files: main.f is the main program, subs.f contain subroutines for reading and writing data and producing the covariance matirx, and system.f contains some utility routines. The first thing that main does is asks you for a data file, which is then read in the ReadData routine. The data is stored as a column of numbers. The first number is the number of samples (n), the second is the number of variables (m), and the next n numbers are the first column of data (variable one), followed by the samples for the remaining $n-1$ variables. The data is stored in the array C. The covariance matrix is constructed in the routine GetCov. The covariance matrix is then saved to the file Cov.txt. The eigenvalues/vectors are computed using the LAPACK routine SSYEV (Single precision, SYmetric EigenValue). The eigenvalues and the percentage of the variance which they explain are printed to the screen, and the prinicpal components are stored in the file PComp.txt.

2. Whew! Don't worry about the details of PCA–try to focus on the main problem: producing Cov and then solving the eigenproblem. Look in GetCov (in subs.f). This routine uses the BLAS-level 3 routine SSYRK which computes

$$Cov := fac * C^T C + 0.0 * Cov$$

   Go to Netlib and find this routine in the BLAS package. Try to figure out how we are calling it (you don't need to download it, it is already installed). Do we really need to initialize Cov to zero?

3. Now, go back in main.f and look at the call to SSYEV. The easiest way to look up a LAPACK routine is with the LAPACK search engine

(go to the LAPACK package in Netlib and click on the search engine link). We're interested in driver routines, so click on that link (on the left). Then click on the "Symmetric Eigenproblems" link. This will bring up a Java applet with several menus. If you select "Real, Single" as the precision, "Simple" as the driver, "With Dependencies," and "Symmetric/Hermitian" then SSYEV will be listed in the box on the upper right. Try changing the precision, does the name change? Restore the settings. LAPACK's search engine is handy for looking up the subroutine to call for a specific problem, but to figure out how to call it, we need to see the code. Click on the "see code" button to view the code for SSYEV. The subroutine call and the explanation of the needed variables is found at the top of all LAPACK routines. What is the strange array WORK? How big should it be?

4. Now you know how the code works, lets get it to compile. Try typing `make` in the command prompt. What happens? The problem is that our source code contains no information on either SSYEV or SSYRK–these are found in the LAPACK and BLAS libraries. As I explained, I've built the ATLAS version of BLAS as well as LAPACK and placed them in my directory (/home/ajp9/cs404/ATLASLinux_PIIISSE1256). UNIX libraries are stored in "archive" files which end with the .a suffix and typically begin with "lib." Each archive is actually a collection of related object-code files (.o files–obtained by compiling with the -c flag), much as a .tar file can contain several files. What archives are available in the ATLAS directory? To get our code to run, we need to link to some of these libraries. This is done using the -l compiler flag: -lFoo would link to libFoo.a. Look at the Makefile. Everything is set, except for the macro LIBS (-L*path* as defined for LIBPTH tells the compiler where to look for libraries). We need to link to the LAPACK library (for SSYEV) and the f77 BLAS library (for SSYRK). This suggest that we should put "-llapack and -lf77blas" on the LIBS line. However, this is not quite complete. The routines in libf77blas call routines in the atlas library, so we need to link to that library as well. Note: the order of the libraries is important. If library A calls routines in library B, the you must link to B after linking to A. Finish setting up the Makefile and build Fpca.

5. Run the program. Use the 5-by-3 sample problem in data.txt.