MATH 150-03 / COMP 150-07 Homework #1

September 24, 2016

1 Overview

The homework is due by 11:59 PM on Sunday, September 25th. All code is to be uploaded to the Tufts cluster. Find your directory in /cluster/tufts/train/math150. Make a directory called submit. Inside the submit directory make a directory called hw1. Please put all submission files there by the due date. Please scan your responses to the non-programming parts (or type them up), and put in the folder with the name hw1_written.pdf. Detailed instructions on how to access and use the cluster will be provided next week.

With the exception of the last problem you are not using any libraries, just plain C code. All code can be compiled on any Linux based system, using the GCC compiler. You do also need Matlab to test the Matlab routines. I suggest you try one of the popular Linux distributions (Ubuntu, OpenSUSE, Fedora, Mint) which should run without problem on a USB stick. There is a lot of documentation on this online and I will present in detail how to do this in Monday's workshop. Another option is to install VirtualBox (a virtual machine) in Windows or on the Mac and to install Linux as a guest operating system inside VirtualBox. This will allow you to use Linux inside a window on Windows or Mac. I will go into more details on this on Monday also. (Note, you may need to enable virtualization option in your computer's bios for VirtualBox to run properly, if this is what you choose to use). At the end of your development, you should upload and test all your code on the Tufts cluster. For Matlab files, use the cluster module matlab/2015a (details to be provided). For the equation parser library use https://www.gnu.org/software/libmatheval/. We will review it's use with GCC in class.

In this homework, you will explore classical algorithms for solving linear systems of equations $Ax = b$, where $A \in \mathbb{R}^{n\times n}$ is a symmetric positive definite matrix. You will implement steepest descent and conjugate gradient algorithms and compare their results. As I mentioned in class, these are very powerful algorithms used widely in a variety of applications and not just for SPD matrices. When the general system $My = g$ is not under-determined one can use these algorithms by working with the normal equation system $M^T My = M^T g$.

Consider first the general descent method:

```
function x = general_descent_method(A,b,x0,TOL,maxiters)
    xn = x0;
    rn = b - A*x0;
    for i=1:maxiters
        dn = [search direction vector];
        alpha = [arg min_(alpha) { J(xn + alpha*dn) }];
        xn = xn + alpha*dn;
        rn = b - A*xn;
        if norm(rn) < TOL
            break;
        end
    end
    x = xn;
```

```
end
```

For steepest descent, one uses $J(x) = \frac{1}{2}x^T Ax - x^T b$ and descent direction $d^n = -\nabla J(x^n)$. Closely related to steepest descent is the conjugate gradient (CG) scheme. Please see chapter 21 of the NLA book by Ford for a description of both methods.
Finally, you will implement a numerical quadrature scheme: the composite trapezoidal rule. See details on https://en.wikipedia.org/wiki/Numerical_integration. The code is similar to the composite Simpson's rule we have discussed.

2 Assignment details

A. (2 pts) State what it means for a matrix $A \in \mathbb{R}^{n\times n}$ to be symmetric and positive definite (SPD). What can be said about the eigenvalues of A?

B. (6 pts) Prove that for an SPD matrix A and vectors x and b satisfying $Ax = b$, the solution x is the unique global minimizer of the functional $J(x)$.

C. (6 pts) Prove that $\alpha_n = \arg\min_\alpha J(x^n + \alpha d^n)$ is given by $\alpha_n = \frac{r^n \cdot d^n}{d^n \cdot Ad^n}$. (Consider the expression $\frac{d}{d\alpha}J(x + \alpha d)$).

D. (6 pts) Prove that $\nabla_x J(x) = -r(x)$ where $r(x) = b - Ax$. Justify the steps in your calculation.

E. (5 pts) Implement the steepest descent algorithm in Matlab. Please supply the following function for me to test:

```
function x = steepest_descent(A,b,x0,TOL,maxiters)
```

You can use a test script I supply to test the function.

F. (30 pts) Implement the steepest descent algorithm in C. The program should compile into an executable called *steepdc*. The program takes 1 command line argument which is the name of a text file containing options. The text file, on a single line contains in comma separated format: the filename for matrix A, the filename for vector b, the filename for vector x0, a floating point parameter TOL, an integer maxiters, and finally, the filename for output vector x. These are all to be supplied on a single line of the text file. There can be multiple spaces between the inputs following the comma. Example:
A.bin,   b.bin, x0.bin,1e-5,   100, x.bin
Scientific notation will be used for the tolerance parameter, as above. See the directory week3/hw1_test_files for examples. Note that for the matrices and vectors, they will be written by me in the following binary file formats. For matrices, we will use the following column major format:

```
num_rows (int)
num_columns (int)
nnz (double)
...
nnz (double)
```

For (column) vectors, the format is as follows:

```
num_rows (int)
nnz (double)
...
nnz (double)
```

You will write the solution computed by your routine in the same format. You can use a test script I supply (test_sd1.m in hw1_test_files folder) to test the function. Notice that the exact way I will test is subject to change, but is roughly as in the script. I also supply an example of how I write the matrices and vectors to disc. Notice that the routine will read the matrix, rhs vectors, and initial guess from disk, and write the resulting solution vector to disk, all in binary format, to the locations specified in the input file.

G. (20 pts) Implement the conjugate gradient (CG) scheme, with the same argument sequence as for steepest descent above. Base your code of the implementation given by NLALIB cg.m function of Ford's Numerical Linear Algebra book. You can get the Matlab code for the algorithms in the book from the website `http://www.ford-book.info/`. Notice that once you have done part (F), this should be not too difficult as you should have all the necessary C functions in place. Your program should compile into an executable *conjgradc* and take the same command line argument as steepest descent. As with steepdc, the routine will read the matrix, rhs vectors, and initial guess from disk, and write the resulting solution vector to disk, all in binary format, to the locations specified in the input file.

H. (10 points) Write a program which runs the Matlab codes (the steepest descent code you wrote above, the CG code from Ford's book, and the pcg function native to Matlab). Compare the Matlab runtimes with the C codes for systems $Ax = b$ (with SPD A) of different sizes from $n = 200$ to $n = 2000$ using maxiters = 100 (or your preferred number of iterations). Supply a plot comparing the runtimes. Also, compare the convergence speeds of the steepest descent and CG routines for a few different test matrices. (See the script make_system1.m in hw1_test_files folder on the cluster. You can control the decay of singular values in the matrices you generate. Faster decay will result in slower residual decrease). You can simply compare the rate of decrease of the residual vector $\|r^n\|$ with each routine, versus n. Supply one or more plots and summarize your findings in a few sentences. You don't need to supply any code you used here.

I. (15 points) Implement the composite Trapezoidal rule for numerical integration (see `https://en.wikipedia.org/wiki/Numerical_integration`), but make one of the inputs a string function to integrate. E.g. 'x*exp(-x*x)'. Make use of the equation parser from `https://www.gnu.org/software/libmatheval/`.

The resulting program should compile into a command called *trapzc* and take one command line argument the name of a text file with options, as above. This text file will contain on one line: The mathematical expression to integrate, lower bound floating point, upper bound floating point, and number of intervals to use. For example:
x*exp(-x*x), 0, 100, 1000
The output should be a string with the integration result, such as:
The integral of x*exp(-x*x) from x=0 to x=100 is approximately 0.5.