Approximating Solutions Using Linear Least Squares, Program 3

Colton Piper

March 2nd, 2018

1 Executive Summary

In this program we will introduce a new method that can solve a linear system $Ax = b$ where A is an nxn matrix. This new method though can also be drawn further to other applications like finding the best approximation in respect to the 2-norm which we call the linear least squares. Thus for a system $Ax = b$ where A is an nxk matrix where $k \leq n$ we can find the best approximate solution x that minimizes the residual $Ax - b$. We will implement code to solve for this and also many test cases in which to test our code.

2 Statement of the Problem

This program is almost a more generalization of solving the $Ax = b$ even when a solution does not exist. Even if we cannot solve the system we can get a best approximation in respect to the 2-norm for a vector. First in this program we will need to implement code that computes the reflector H_i that transforms a given vector v. Next we need to know if our implemented code works and we can do this by showing that H_i has the properties of a householder reflector, being orthogonal and being an isometry. We can also show that H_i operates on v correctly and any other vectors other than the one used to create H_i.

Next we need to implement a code that solves the linear least squares problem which is finding the x that gives the minimum of $||b - Ax||_2$. First we need the code that transforms

A to upper trapezoidal and store R and all the H_i efficiently. Then implement code that solves the least squares problem after you solve for R. Then we need to do many tests on our code that shows that this factorization works and is numerically acceptable under finite precision of the machine being used.

3 Description of the Mathematics

The mathematics behind the creation of this routine is not the most difficult in the world. We are essentially using the fact that orthogonal matrices are an isometry and preserve norms when used as an operator, i.e. $||v|| = ||Qv||$ where Q is an orthogonal matrix. The problem we are trying to solve is

$$\min_{x \in \mathbb{R}^k} ||b - Ax||_2.$$

Now the way we are going to solve this is by finding an orthogonal matrix that transforms A into an upper trapezoidal matrix. Thus we would have

$$\min_{x \in \mathbb{R}^k} ||Q(b - Ax)||_2$$
$$\min_{x \in \mathbb{R}^k} ||Qb - QAx||_2.$$

Now how can we construct a matrix such that it is orthogonal and when applied to $QA = (R^T \ 0^T)^T$. The creative part comes into play when we will not try to find Q all at once but instead find pieces of it almost as we did with permutations and L matrices in the LU factorization program. Thus we will have Q be a combination of many orthogonal matrices.

Consider the matrix $H = I + axx^T$, where $a = -2/||x||_2^2$, $x = v + \gamma e_1$, $\gamma = \pm||v||_2$, and v

is any vector, but we will be taking it as part of a column of A. Then we can see that

$$\begin{aligned}
H * v &= (I - \frac{2}{||x||_2^2} x x^T) * v \\
&= v - \frac{2}{||x||_2^2} x x^T v \\
&= v - \left(\frac{2}{||x||_2^2} x^T v \right) x \\
&= v - \frac{2(v_1^2 \pm v_1 ||v||_2 + v_2^2 + v_3^2 + \cdots + v_n^2)}{v_1^2 \pm 2 v_1 ||v||_2 + ||v||_2^2 + v_2^2 + v_3^2 + \cdots + v_n^2} x \\
&= v - x = -(\pm ||v||_2) e_1.
\end{aligned}$$

Thus multiplying what we will call a householder reflector by the vector that was used to create the reflector is $-\gamma e_1$. Thus we can apply many of these reflectors to the matrix A because they are orthogonal as well. Now we want to make our matrix into an upper trapezoidal, so we will go column by column and the vector v that we use to create H_i will only be below the diagonal of our soon to be kxk triangle matrix. But to keep the rest of the matrix in upper trapezoidal form, our H_i.

$$H_i = \begin{pmatrix} I_{i-1} & 0 \\ 0 & H \end{pmatrix}$$

$$H_i v = \begin{pmatrix} I_{i-1} & 0 \\ 0 & H \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} v_1 \\ -\gamma e_1 \end{pmatrix}$$

Thus we will have $H_k H_{k-1} \cdots H_2 H_1 A = QA$. Thus we have our orthogonal matrix Q. Then we have

$$||Qb - QAx||_2 = || \begin{pmatrix} c \\ d \end{pmatrix} - \begin{pmatrix} R \\ 0 \end{pmatrix} x ||_2 = ||c - Rx||_2^2 + ||d||_2^2.$$

Thus we need to just solve for x that gives the norm of $||c - Rx||_2^2 = 0$. That is how we solve our linear least squares.

4 Description of the Algorithms and Implementation

4.1 Creating and Applying Householder Matrices

The implementation of this code was more difficult then I had initially thought. I expected it to be very similar to the LU code implemented in program 2. I have three functions in my "main.cpp" for this program. The first is tailored toward the first task which is creating H_i based on a given vector v and applying it to an arbitrary vector u. Then the second and third are functions are tailored towards the second task which is actually solving the linear least squares problem.

The first function takes in as input a vector v which is to be used to create H_i, a vector u which the reflector matrix will be applied to and an integer i, which represents the i^{th} stage and for which H we are creating. If the vector u is v then we get the following

$$H_i v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{i-1} \\ \pm||\tilde{v}||_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \text{ where } v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}, \text{ and } \tilde{v} = \begin{pmatrix} v_i \\ v_{i+1} \\ \vdots \\ v_n \end{pmatrix}$$

Then for any other vector u we have $H_i u = (I + axx^T)u = u + (ax^T u)x$. First we compute the $||v||$ norm as $x = v \pm ||v||_2 e_1$. That is a simple for loop requiring n time. Then we have no need for v thus we change $v \to x$ by adding γ to the first element of v. Next we solve for $||x||_2^2$, but we do not need to explicitly solve it as we can see that

$$\begin{aligned} ||x||_2^2 &= v_i^2 \pm 2v_i||v||_2 + ||v||^2 + v_{i+1}^2 + v_{i+2}^3 + \cdots + v_n^2 \\ &= 2(v_i^2 + v_{i+1}^2 + \cdots + v_n^2) \pm 2v_i||v||_2 \\ &= 2||v||_2^2 \pm 2v_i||v||_2 \\ &= 2||v||_2(||v||_2 \pm v_i). \end{aligned}$$

Thus we can solve for what we need with only 3 flops. Then as our matrix-vector product is $u + (ax^Tu)x$, we still need to compute the scalar x^Tu which is another for loop. Then finally we can update u with a for loop because we have $a = -\frac{2}{||x||_2^2}$, $x^T * u$ and x. Thus because of the structure of this matrix vector product it only takes linear time as we only have three non nested for loops in the function. The storage is constant as we only declared 3 variables in the code. One for the size of n, one for the norms $||v||_2$ and $||x||_2^2$, and one for the scalar $x^T * u$.

4.2 Solving Linear Least Squares Problem

The next code to implement was transforming A into an upper trapezoidal matrix, then after storing R and all the H_i used efficiently we were to actually solve the linear least squares problem with a given b. The first function only takes in the matrix A where A is an n by k and returns an $n+1$ by k matrix. First Notice that the householder reflector matrix, $H_i = I + ax_ix_i^T$, only depends x_i. Thus to know what H_i is we only need to store x_i. Now because we are transforming A into an upper trapezoidal matrix we can use the empty zero spaces. Each H_i makes elements under the i^{th} element of the i^{th} column zero we can store x_i there. Unfortunately that only has $n-i-1$ spots open where x_i is $n-i$. Thus we will need an vector of k length to hold one of the elements of each x_i. Now because $A[i][i] = \mp||v_i||_2$ and $x_i[0] = v_i[0] \pm ||v_i||_2$, we can just store $v_i[0]$ in that k length vector, call it $v0$ and if we need $x_i[0]$ we can subtract $v_i[i] - A[i][i]$.

This will have a large for loop outside of everything which will iterate through the columns and are for each H_i which will be computed inside the loop. Inside the for loop is similar to above as we will first calculate both the $||v_i||_2$ and then $||x_i||_2^2$ norms, then we will have a for loop which iterates through the columns right of the i^{th} column and applies H_i to them. We only need to change the diagonal element $A[i][i]$ in the i^{th} column because everything under it we fill with x_i which is the same as v_i, thus we do not need to touch it. We do this k times and that will yield the upper trapezoidal matrix where all the zero parts of the columns are filled in with x_i. Then we can add $v0$ vector to the bottom of A yielding the $n+1$ by k matrix. This only requires k storage with some variables needed for loop iterations like n and k. The time complexity though is of order of nk^2 thus if $k = n$ it will be n^3 just like the LU factorization of a square matrix.

Then our last function is for solving the linear least squares problem. This function takes in both the transformed A, which holds R and all our x_i which are used to create the householder reflectors, and b. Recall from the description of mathematics section that we need to apply H_i to both b and A because we want $||b - Ax||_2$ to be minimized. Thus we first will need to apply all our reflectors to b. We can do that quite simply because we have everything to that we need to apply $H_i = I + ax_ix_i^T$ to b. So we will have a for loop that iterates k amount of times corresponding to each reflector being applied to b. Then inside that loop we will need to compute $||x_i||_2^2$ and x_i^Tb, then can update each element under the $(i-1)^{th}$ element of b. Once we have done that we only need to solve $||c - Rx||_2^2 = 0$. Thus solve $Rx = c$. This is trivial as R is an upper triangular k by k matrix. Thus we can solve by each row and back substitute. This function only takes order $kn + k^2$ time as we have two for loops with for loops inside them. The storage is constant as we only declare 5 variables throughout the function. Again though if we have a square matrix this solver goes to order n^2 for time complexity just like the LU factorization did.

5 Description of the Experimental Design and Results

The easiest thing to test our method is first solving a linear system that is square, i.e. the system $Ax = b$ where A is an n by n matrix. The good news for this is that we already have code from the last program that we can check this program with. So for the first test let us solve the system

$$\begin{pmatrix} 5 & 1 & 3 & 1 \\ 10 & 5 & 12 & 3 \\ 5 & 10 & 23 & 5 \\ 15 & 6 & 19 & 7 \end{pmatrix} \vec{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

We get the solution in both cases to be $\vec{x} = (0.1 \quad -4 \quad 2.5 \quad -3)^T$. To show an example when $k < n$ and $b \in R(A)$, thus a unique solution does exist for $Ax = b$, we can choose x before hand then get be from Ax. Then run the code to ensure that we get the solution we originally chose.

$$\begin{pmatrix} 5 & 1 & -3 & 1 \\ 10 & 5 & 12 & -3 \\ 5 & -10 & 23 & 5 \\ 15 & -6 & 19 & 7 \\ 8 & -6 & -5 & 3 \end{pmatrix} \vec{x} = \begin{pmatrix} 16 \\ -48 \\ -24 \\ -2 \\ 47 \end{pmatrix}$$

I chose the solution $\vec{x}$ to be $(1 \ \ -2 \ \ -3 \ \ 4)^T$ and the code does return the correct solution.

To check the code for the least squares solution is a little bit more difficult as you need to find a residual vector that is orthogonal to all of the columns of the given matrix A. First I checked my code with an example used from a khan academy least squares video. There problem was

$$\begin{pmatrix} 2 & -1 \\ 1 & 2 \\ 1 & 1 \end{pmatrix} \vec{x} = \begin{pmatrix} 2 \\ 1 \\ 4 \end{pmatrix}, \qquad \vec{x}_{min} = \begin{pmatrix} 1.42857 \\ 0.428571 \end{pmatrix}.$$

My program does come to the same conclusion of 10/7 and 3/7.

6 Conclusions

Unfortunately I ran out of time to do a rigorous testing of the code for accuracy and did not get to test it extensively. It worked for a simple examples I showed above an a few other simple examples from the internet I found. I believe the code works and we can see that it is of order of time complexity nk^2. Thus it does get to cubic time if we have a square matrix just like the LU factorization did.