Program 3 Foundations of Computational Math 2 Spring 2019

Due date: via email by 11:59PM on Monday 4 March 2019

Programming Problem

General Task

Your general task is to implement two basic approaches to solving the linear least squares problem

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

given $b \in \mathbb{R}^n$ and the matrix $A \in \mathbb{R}^{n\times k}$ with linearly independent columns.

The first approach uses a Cholesky factorization.

1. Show that there is a linear system of equations $Mx_{min} = g$ where $M \in \mathbb{R}^{k\times k}$ is symmetric positive definite, $g \in \mathbb{R}^k$, and $x_{min} \in \mathbb{R}^k$ solves the linear least squares problem. M and g can be defined based on the formulas involving only the elements of A and b based on the geometric characterization of the solution of the linear least squares problem.

2. Implement a Cholesky factorization code that is efficient in operation and storage and apply it to solve the linear least squares problem. It should be compared to the approaches based on Householder transformations discussed below and its accuracy assessed systematically based on the design of different types of problems.

The second approach arises from considering the matrix $A \in \mathbb{R}^{n\times k}$ with linearly independent columns and the transformation/factorization

$$H_k H_{k-1} \cdots H_1 A = \begin{pmatrix} R \\ 0 \end{pmatrix}$$

where H_i, $1 \le i \le k$ are Householder reflectors and $R \in \mathbb{R}^{k\times k}$ is a nonsingular upper triangular matrix.

1. Implement a code that computes the reflector H_i that transforms a given vector v

$$H_i v = \begin{pmatrix} \nu_1 \\ \vdots \\ \nu_{i-1} \\ \pm\|\tilde{v}\|_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad v = \begin{pmatrix} \nu_1 \\ \vdots \\ \nu_n \end{pmatrix} \quad \tilde{v} = \begin{pmatrix} \nu_i \\ \vdots \\ \nu_n \end{pmatrix}$$

2. Develop, describe and execute a plan that demonstrates that this primitive works and is efficient in storage. Your plan should include things such as verifying that the primitive defines a matrix with all of the desired properties: orthogonal, isometry, correct action on v, and correct action on vectors other than v. This demonstration should include that it achieves the expected accuracy in finite precision computation, i.e., single precision. (Note that to include the assessment of single precision it will be necessary to have a working double precision version also.)

3. Implement a code that is capable of performing this transformation of A to upper trapezoidal, i.e., to compute R, and storing the information about the H_i and R efficiently in-place in a 2-dimensional $n \times k$ array and a small number of additional 1-dimensional arrays of length n or k. Also implement the necessary additional routines to solve the linear least squares problem

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

given $b \in \mathbb{R}^n$.

4. Develop, describe and execute a plan that demonstrates that this transformation/factorization routine works correctly and within the tolerances expected for finite precision computation, i.e., single precision. (Note that to include the assessment of single precision it will be necessary to have a working double precision version also.)

5. Develop, describe and execute a plan that demonstrates that the additional routines required to solve full rank least squares problems work correctly and within the tolerances expected for finite precision computation, i.e., single precision. (Note that to include the assessment of single precision it will be necessary to have a working double precision version also.)

The last demonstration in the list, solving least squares problems, must be demonstrated on multiple examples with a wide range of values of n, k, and b for three situations:

1. $n = k$, i.e., a square nonsingular matrix A where $x_{min} = A^{-1}b$.

2. $n > k$ and $Ax = b$ for $b \in \mathbb{R}^n$ and $b \in \mathcal{R}(A)$ i.e., a rectangular matrix A with full column rank and a vector b that define a consistent set of overdetermined equations.

3. $n > k$ and $b \in \mathbb{R}^n$ and $b \notin \mathcal{R}(A)$ i.e., a rectangular matrix A with full column rank and a vector b that define a linear least squares problem with a nonzero residual.

This requires careful construction of the test problems. See the discussion below.

A Particular Test Problem

We have analyzed the use of LU factorization with multiple pivoting strategies, e.g., partial and complete. The standard example of a problem that demonstrates that partial pivoting can be unstable by illustrating an exponential growth factor, e.g., for $n = 4$,

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 \end{pmatrix}$$

It is more precise to say that for this matrix partial pivoting does not stop exponential growth since pivoting is not necessary for the existence of the factorization. After a fairly small number of Gauss transforms are applied, the relative sizes of elements in the transformed matrix are problematic, i.e., $O(1)$ sized elements are insignificant compared to the elements with magnitude $O(2^k)$. Solve systems with various values of n with the pattern of values seen in this example matrix using the Householder reflectors for factorization and discuss if it is a viable and stable approach for a reasonable range of problem sizes.

Test Problems

When generating test problems, you may use library routines available in whatever language environment you are using. **For generating problems only**, you may, for example use MATLAB, or any other problem solving environment, and its capabilities associated with generating test matrices at random or with specific properties. You may also use its factorizations, e.g., to compute the QR factors of a given matrix A. These can only be used to generate tests or to assess the correctness of the results of your code. **MATLAB, any other problem solving environment, and any prohibited languages, may not be used to implement the code you submit as a solution.**

Note you can also generate the matrices of the types required for this assignment, e.g., nonsingular matrices, full rank rectangular matrices, i.e., $A \in \mathbb{R}^{n\times k}$ for $n \geq k$ with linearly independent columns, and isometries, i.e., $Q \in \mathbb{R}^{n\times k}$ for $n \geq k$ such that $Q^TQ = I_k$. The techniques are reviewed below. If there is confusion on how to generate these matrices ask in class or set up an appointment to discuss them.

A nonsingular $n \times n$ matrix, A, can easily be generated from a random $n \times n$ matrix G by adding an $n \times n$ diagonal matrix, D. (The nonzero diagonal elements of D can be positive or negative.) The magnitude of the nonzero elements of D are taken so that the matrix $G + D$ is strictly diagonally dominant. Since you may want the test matrix not to be strictly diagonally dominant for some tests, you can postprocess $G + D$ by applying random row and column permutations $A = P_{rows}(G + D)P_{cols}$. This preserves nonsingularity while generically destroying diagonal dominance.

An $n \times k$ matrix, A, with linearly independent columns can be generated using the technique described above to form an $n \times n$ nonsingular matrix. Selecting k columns of an

$n \times n$ nonsingular matrix yields an $n \times k$ matrix, A, with linearly independent columns. You may also select k rows to get a $k \times n$ matrix and then take A to be the transpose.

A technique that avoids producing an $n \times n$ matrix starts by forming a set of k linearly independent vectors by defining a lower trapezoidal matrix, $L \in \mathbb{R}^{n \times k}$ with nonzero diagonals. The columns must be linearly independent due to the locations of the 0's and the nonzeros on the diagonal. To create the matrix A that does not have the upper triangular part 0 simply postmultiply by a $k \times k$ nonsingular matrix. This can be created as above or more simply by generating a $k \times k$ upper triangular matrix with nonzeros on the diagonal. Note that this technique also generates an $n \times n$ nonsingular matrix when $k = n$. As before random permutations can also be applied to scramble the matrices.

The conditioning of the nonsingular matrices produced this way can be easily controlled for the diagonal dominant technique. Think about this using Gershgorin disks to see how D can be chosen to have the resulting $A = G + D$ be well-conditioned. If A is produced by generating a triangular L and/or U, well-conditioned matrices A can be generated by keeping the off diagonal elements of the triangular matrices reasonable in magnitude compared to the diagonal elements of the triangular matrix. For example, a lower triangular matrix with diagonal elements all $O(1)$ while all off diagonal elements smaller than 1 in magnitude is usually well-conditioned. Note in all cases you may make use of available libraries to estimate the condition numbers of the matrices generated and reject those that are unacceptable, e.g., LAPACK or MATLAB.

In addition to using built-in primitives of MATLAB or similar environments to generate orthogonal matrices it is possible to generate them via simple techniques. For example, an $n \times n$ rotation matrix can be easily defined by considering a random index pair (i, j) and random angle θ. The matrix, Z, that is the identity everywhere except positions (i, i), (j, j), (i, j), (j, i), where it is taken to be

$$\cos\theta = e_i^T Z e_i, \quad \cos\theta = e_j^T Z e_j, \sin\theta = e_i^T Z e_j, \quad -\sin\theta = e_i^T Z e_j$$

is a plane rotation and orthogonal. Selecting many, say s, random index pairs (i, j) and random angles θ and multiplying all of the rotations they define together yields an orthogonal matrix

$$Q = Z_1 Z_2 \cdots Z_s.$$

Of course, you need to select enough pairs and angles, s, so that the matrix is dense.

Similarly, one can use reflectors to generate an orthogonal matrix Q. Simply choose several random vectors, v_i, $i = 1, \ldots, s$ for a large s. Then for each v_i form an elementary reflector

$$Q_i = I - 2u_i u_i^T, \quad u_i = \frac{1}{\|v_i\|_2} v_i$$

and $Q = Q_1 Q_2 \cdots Q_s$. Taking $s >> n$ is a good way of scrambling the directions defining Q.

Once an $n \times n$ orthogonal Q is computed selecting randomly k of the n columns yields an $n \times k$ matrix $\tilde{Q}$ that has orthonormal columns, i.e., $\tilde{Q}^T \tilde{Q} = I_k$. Of course, it is not necessary to form the $n \times n$ orthogonal Q to take k selected columns. This may be done as described in the homework solutions. After selecting the indices of the columns of Q you

plan to use, then rather than computing all of Q by taking the products of the s rotations or reflectors you can simply apply each one in turn to a set of k vectors that are initialized to the standard basis elements defined by the randomly selected column indices. That is, suppose you want columns 2, 10 and 50 of Q and Q is defined as the product of s some set of reflectors. The isometry $\tilde{Q} \in \mathbb{R}^{n\times 3}$ is efficiently computed using

$$\tilde{Q} = \begin{pmatrix} e_2 & e_{10} & e_{50} \end{pmatrix}$$

$$for \quad i = 1, \ldots, s \quad \tilde{Q} \leftarrow Q_i \tilde{Q} \quad end$$

Of course, there is no reason to use only reflectors or rotations. A mix of reflectors and rotations can also be used.

When using either rotations, reflectors or a combination, it is necessary to compute the the matrix-matrix product or matrix-vector products efficiently so as not to take huge amounts of time when creating test problems since you must run many of them. Make sure you exploit all of the structure available to gain computational and storage efficiency. You should of course point out anything you do along these lines in your solution.

Finally, you should verify that the matrix computed has orthonormal columns to at least single precision accuracy. This computation of $Q^T Q$ or $\tilde{Q}^T \tilde{Q}$ and comparison to I_n or I_k should be performed in double precision.

Note that it is a good idea to do all of the computations creating the test matrix of any type, i.e., full rank or orthogonal, in double precision and verify that it satisfies all of the properties required of the matrix up to single precision levels for assessing single precision factorization and solution algorithms.

For the three situations mentioned in the General Task section, you should run problems where you have created the problem with a known solution and those for which you do not know the solution. **You must consider each of these classes of problems in your report.**

To form a consistent overdetermined set of equations given A with a known solution, z, simply set b to $b = Az$. These computations should also be done in double precision.

A linear least squares problem with known solution, x_{min}, and nonzero residual $r_{min} = b - Ax_{min} \neq 0$ can be created by modifying a consistent overdetermined system with known solution as follows:

1. Choose your desired solution x_{min}
2. Compute $b_1 \in \mathbb{R}^n$ with $b_1 \in \mathcal{R}(A)$ by $b_1 = Ax_{min}$.
3. Set b to $b = b_1 + b_2$ where b_2 is any vector that is chosen to be orthogonal to $\mathcal{R}(A)$.

The linear least squares problem

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

therefore has solution x_{min} and residual $r_{min} = b_2$.

This requires dealing with the subspace $\mathcal{R}(A)$. For these problems it is convenient therefore to generate an orthonormal basis, $Q \in \mathbb{R}^{n\times k}$, and then generate $A \in \mathbb{R}^{n\times k}$ so that $\mathcal{R}(A) = \mathcal{R}(Q)$. Given Q, A can be generated by computing $A = QM$ where $M \in \mathbb{R}^{k\times k}$ is a random nonsingular matrix.

Generating b_1 and b_2 is a bit more complicated. Given an vector $v \in \mathbb{R}^n$ we have

$$v = v_1 + v_2, \quad v_1 \in \mathcal{R}(Q) \quad v_2 \in \mathcal{R}^{\perp}(Q)$$
$$v_1 = Q(Q^T v) \quad v_2 = v - v_1$$

So one way of generating these vectors is to take a random vector v and compute v_1 and v_2 as above. Of course, a priori you will not know the relative magnitudes of v_1 and v_2. It is possible that the random vector v will be almost entirely in $\mathcal{R}(Q)$ or almost orthogonal to it. So you may have to try several random v until you get two reliable directions v_1 and v_2. In any case, you should check that $\cos\theta_{1,2} = v_1^T v_2/(|v_1|_2|v_1|_2)$ is suitably small to verify that finite precision has not caused difficulties. You should, in fact, try several such pairs of various dimensions to test your code. Additionally, to analyze your code's performance, given any such pair, you can create multiple b vectors by taking various combinations of v_1 and v_2 in a controlled manner, i.e., set

$$b = b_1 + b_2 = \alpha_1 v_1 + \alpha_2 v_2$$

keeping $\|b\|_2$ constant. When α_1 is large relative to α_2 the system is closer to consistent than when α_2 dominates.

When you generate problems for which you do not know the solution a priori, you should think about how you would determine if the solution is reasonable. For example, you should examine the residual carefully to make sure it satisfies all required conditions. Also note, since the solution is supposed to minimize the norm of the residual over all x, you can also check residuals generated for randomly selected x vectors and compare their norms to the norm of the residual generated by x_{min}. Finally, as noted above, you can compare your results to other libraries or routines available to you.

Important Point

The main part of the programming assignment is the clear design, definition and execution of a plan to generate sufficient numbers and types of test examples, compute efficiently and to convincingly write an argument that the codes are performing correctly in a single precision finite precision environment (for which you also have to compute in double precision when producing the tests and for comparison purposes). This is a key goal of FCM 1 and 2.

Do not simply implement your code or use a library code (properly cited of course) on a few problems with simplistic checks of correctness. **This will not receive a significant amount of credit. You must carefully structure what you did, why you did it and how what you observed is consistent with your goal of validating the code.**

Submission of Results

Expected results comprise:

- A document describing your solutions as prescribed in the notes on writing up a programming solution posted on the class website.
- The source code, makefiles, and instructions on how to compile and execute your code including the Math Department machine used, if applicable.
- Code documentation should be included in each routine.
- All text files that do not contain code or makefiles must be PDF files. **Do not send Microsoft word files of any type**.

These results should be submitted by 11:59 PM on the due date. Submission of results is to be done via FSU Dropbox at `https://dropbox.fsu.edu`. Drop the files off for Zhifeng Deng using his MyFSU email `zd16d@my.fsu.edu`. (Note: **do not use `zdeng@math.fsu.edu` to drop off the files**) You should login to FSU Dropbox using your MyFSU login. If for some reason you cannot use FSU Dropbox please email the files to Zhifeng at the email address above.