Graded Homework 3 Foundations of Computational Math 1 Fall 2021

The solutions must be submitted by 11:59PM on November 8, 2021 using the Canvas course page.

Written Exercises

There are no written exercises for this assignment.

Programming Exercise

General Task

The Codes

1. Give your code that computes the LU factorization of a matrix $P_r A P_c$ without pivoting, using partial pivoting or using complete pivoting from the previous programming assignment, implement the appropriate triangular system solutions algorithms that allow the use of your factorization to solve $Ax = b$ where $A \in \mathbb{R}^{n\times n}$ is nonsingular.

2. Implement the codes necessary to solve the least squares problem defined by a vector $b \in \mathbb{R}^n$ and the matrix $A \in \mathbb{R}^{n\times k}$ with linearly independent columns using 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.

3. You will also need various test routines designed to evaluate and validate the correctness of the code and accomplish the tasks described below. You may code in any compiled and typed language you wish although C, C++, Julia, and Fortran are preferred. In all cases, however, you may not use standard libraries such as LAPACK or built-in matrix routines for pieces of your routines implementing the computations described above.

Suggested Subroutines for the factorization:

1. Initialization: This routine generates the matrix to be factored and places it in the array to be used during the factorization as well as another array used to preserve the matrix for use when assessing the accuracy of the factorization and the solutions of the least squares problems and system solves.

2. FormHouse**:** This routine examines the appropriate part of the active part of the matrix and determines the parameters determining H_i. You may find it convenient to return the parameters in a work vector and then place the values in the appropriate positions of the array in which A is being transformed.

3. ApplyHouse**:** This routine applies H_i to the active part of the matrix or to a vector. The former is needed during the factorization and the latter when transforming the vector b. You will also use this routine to assess the quality of the factorization of A by computing
$$\|A - H_1 H_2 \dots H_k \begin{pmatrix} R \\ 0 \end{pmatrix}\|.$$

4. The triangular system $Rx = c$ should be solved by an appropriate upper triangular solve routine. You should be able to exploit all or part of the code from your $Ux = y$ code from the LU factorization set of codes.

5. You will also need to develop support routines needed to generate A, b etc. that define linear least squares problems and linear systems to solve. Some suggestions for this are given below and were given in the LU programming assignment.

Library Codes

You may use libaries and external routines in your test routines to generate solutions for comparisons, to generate historgrams, graphs and any other useful summary display mechanisms. You may also compare your solutions against least squares library routines and LU factorization.

Metrics

There are several important metrics to use when assessing the code's correctness. These metrics should be computed in double precision espeicially if you have run your routines in single precision. Some suggestions follow:

1. When comparing matrices use more than one matrix norm, e.g., the finitely computable ones, $\|M\|_1$, $\|M\|_\infty$ and $\|M\|_F$.

2. When comparing vectors use more than one norm, e.g., $\|v\|_1$, $\|v\|_\infty$ and $\|v\|_2$.

3. Check the QR factorization accuracy (as you did for the LU in the previous programming assignment)
$$\frac{\|A - H_1 H_2 \dots H_k \begin{pmatrix} R \\ 0 \end{pmatrix}\|}{\|A\|}$$
where $\|A\| \geq 1$, i.e., relative error for large A.

4. If the solution is known by design of the problem check

$$\frac{\|x - \tilde{x}\|}{\|x\|}$$

where $\tilde{x}$ is the computed solution and $\|x\| \geq 1$.

5. You should check the accuracy via the residual $b - A\tilde{x}$ and

$$\frac{\|b - A\tilde{x}\|}{\|b\|}$$

whether or not you know the true solution and for least squares compare to the size of b_2, the part of b not in $\mathcal{R}(A)$ when it is known.

6. You should compute the growth factor

$$\gamma_\epsilon = \frac{\| \ |L_\epsilon||U_\epsilon| \ \|}{\|A\|}$$

where $L_\epsilon U_\epsilon = P_r A P_c$ is the **computed** factorization of $P_r A P_c$ using the selected pivoting strategy. This will be important for the next assignment that will assess the numerical stability of solving systems but it is also useful for checking the correctness of structured problems such as the one in the study questions with the large growth of elements. This can be compared to the growth observed when using the Householder approach for square nonsingular systems.

Empirical Tasks Set 1 : Testing the Correctness of the Householder Factorization Codes

- 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.)
- 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.)

Empirical Tasks Set 2 : Solving Square Systems

Compare the solutions of $Ax = b$ generated by the LU factorization you have already implemented and the Householder approach. You can use the systems discussed in the LU programming assignment and any others you think appropriate. For each problem size and class of problem, generate many example problems and evaluate/compare the quality of the factorizations, the growth factor, the quality of the residual, and, if the true solution is known, the relative error in the computed solution $\tilde{x}$, You should present your results in a form appropriate to characterize these metrics over a large data set, i.e., too large to look at each problem individually. This can be done, for example, by graphs and histograms. The latter is particularly useful for detecting outliers in the performance such as large factorization error. These outliers can be discussed in more detail and explained if you wish.

Empirical Tasks Set 3 : Linear Least Squares Problems

- 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 two situations:

  1. $n > k$ 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.

2. $n > k$ 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.

Generation of Test Problems

The previous programming assignment discussed the generation of square nonsingular linear systems to solve using LU factorization and using double precision to represent "exact" computations. Those systems can also be solved using the Householder approach when comparing its accuracy and stability to LU factorization. Below are some further suggestions that include some related to the least squares problems. A key consideration in this assignment is the generation of test problems. They must have full rank, i.e., linearly independent columns.

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.

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.

Given the ability to generate orthogonal matrices it is then possible to generate matrices with a specified condition number $\kappa = \|A\|_2 \|A^{-1}\|_2$ for nonsingular $A \in \mathbb{R}^{n \times n}$ and $\kappa = \|A\|_2 \|A^\dagger\|_2$ for full rank $A \in \mathbb{R}^{n \times k}$.

The technique is described here for $A \in \mathbb{R}^{n \times n}$ but it generalizes to rectangular matrices easily. Any matrix $A \in \mathbb{R}^{n \times n}$ has a factorization $A = U \Sigma V^T$ where $U \in \mathbb{R}^{n \times n}$, $V \in \mathbb{R}^{n \times n}$, and $\Sigma \in \mathbb{R}^{n \times n}$ where U and V are orthogonal matrices, i.e., $U^T U = U U^T = I$ and $V^T V = V V^T = I$, and Σ is a nonnegative diagonal matrix, i.e., it has positive diagonal elements $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$ and 0 in all off-diagonal positions. For example, for $n = 5$, the matrix has the form

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 \\ 0 & 0 & 0 & \sigma_4 & 0 \\ 0 & 0 & 0 & 0 & \sigma_5 \end{pmatrix}.$$

A is nonsingular if and only if all $\sigma_j > 0$. The condition number is defined by these positive values by

$$\|A\|_2 = \sigma_1, \quad \|A^{-1}\|_2 = \frac{1}{\sigma_n}$$

$$\kappa = \|A\|_2 \|A^{-1}\|_2 = \frac{\sigma_1}{\sigma_n}.$$

The σ_j are called the singular values of A. So after generating the orthogonal matrices U and V you can select the singular values, σ_j, and define an A with a specific condition number.

For a rectangular matrix $A \in \mathbb{R}^{n\times k}$, $U \in \mathbb{R}^{n\times n}$, $V \in \mathbb{R}^{k\times k}$, and $\Sigma \in \mathbb{R}^{n\times k}$ where U and V are orthogonal and Σ is a diagonal rectangular matrix, e.g., for $n = 10$ and $k = 3$,

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

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.