# Programming Assignment 1 Foundations of Computational Math 1 Fall 2024

**The solutions are due on Canvas by 11:59 PM on Monday, September 16, 2024**

# Programming Problems

# General Task

This assignment concerns the implementation and testing of structured matrix-vector products and a structured matrix-matrix product. These will be adapted to be used in the implementation and testing of the $LU$ factorization in a later programming assignment. The structure in this assignment is triangular matrices and multiple data structure restrictions.

To provide some assistance in getting used to implementing the types of experiments to be performed in FCM 1 and FCM 2, an example MATLAB code for a driver routine that specifies the experiments to be performed, performs the experiments, collects and displays the results in a user-specified format for the matrix-matrix products and matrix-vector routines. Additionally, routines are provided for

- computing the product $w \leftarrow Uv$ given the vector $v$ and the upper triangular matrix $U$ using
  - row-oriented inner product form
  - column-oriented vector triad form .

These codes use a standard 2-D array to store the associated matrices. Other data structures will be specifed for testing for your assigned codes.

You are encouraged to consult with Yue and/or me early in the development and testing of these routines.

Your tasks will unit lower triangular matrices in matrix-vector products as well as the product with an upper triangular matrix. The driver includes testing of routines for a unit lower triangular matrix times a vector.

**Note that you need not use a this driver or a single driver for your testing. In fact, initially you should develop a simpler one that contains only the pieces of specific concern for particular tests. You can then gradually build up the complexity of the tester. The example provided gives a form of driver that is not unusual when testing a set of routines that require the generation of similar test data and presents typical ways of organizing the results to support the contention that your codes work. It also can be modified to be a tester for later assignments.**

# Definitions

The matrix $L \in \mathbb{R}^{n\times n}$ is a lower triangular matrix if

$$\lambda_{ij} = e_i^T L e_j$$

$\forall i = 1, \ldots, n, \;\; j > i, \quad \lambda_{ij} = e_i^T L e_j = 0, \quad$ i.e., elements strictly above the main diagonal are 0

$$\forall i = 1, \ldots, n, \;\; j \leq i, \quad \lambda_{ij} = e_i^T L e_j, \quad \text{may or may not be 0.}$$

and the matrix $L \in \mathbb{R}^{n\times n}$ is a unit lower triangular matrix if additionally

$$\forall i = 1, \ldots, n, \quad \lambda_{ii} = e_i^T L e_i = 1, \quad \text{i.e., elements on the main diagonal are 1}$$

In other words all nonzeros other must occur strictly on or below the main diagonal. and for a unit lower triangular matrix 1's are required on the main diagonal

The matrix $U \in \mathbb{R}^{n\times n}$ is an upper triangular matrix, i.e.,

$\mu_{ij} = e_i^T U e_j \forall i = 1, \ldots, n, \;\; j < i, \quad \mu_{ij} = e_i^T U e_j = 0, \quad$ elements strictly below the main diagonal are 0

$$\forall i = 1, \ldots, n, \;\; j \geq i, \quad \mu_{ij} = e_i^T U e_j, \quad \text{may or may not be 0.}$$

In other words all nonzeros must occur on or strictly above the main diagonal.

The matrix product $M = LU \in \mathbb{R}^{n\times n}$ is, in general, a dense square matrix, Given vectors $v \in \mathbb{R}^n$ and $s \in \mathbb{R}^n$, the products $w = Lv$ and $z = Us$ are vectors in $\mathbb{R}^n$. In some important cases, the vectors $v$ may have structure, e.g., 0 values in specific positions. This occurs for example when considering the middle product implmentation of the matrix-matrix product $M = LU$ since $Me_j = L(Ue_j)$ and $Ue_j$, being the $j$-th column of an upper triangular matrix is guaranteed to have 0 values for its elements in position $j + 1$ to $n$. A similar observation can be made for inner products of rows and columns of triangular matrices, 0's in either mean some operations that are in the general dense matrix vector product can be removed. These facts must be exploited in any matrix-vector product involving such structured vectors.

# The Codes

The following subroutine/data structure combinations must be implemented and tested.

1. A subroutine that computes the product $w \leftarrow Lv$ given the vector $v$ and matrix $L$ where $L$ is a unit lower triangular matrix stored in a 2-D array.

2. A subroutine that computes the product $w \leftarrow Lv$ given the vector $v$ and matrix $L$ where $L$ is a unit lower triangular matrix stored using compressed columns or rows depending on your choice of algorithm.

3. A subroutine that computes the product $w \leftarrow Lv$ given the vector $v$ and matrix $L$ where $L$ is a banded unit lower triangular matrix where the nonzeros in $L$ are restricted to the 1's on the main diagonal (which should not be stored explicitly) and the two subdiagonals $\lambda_{2,1}, \ldots, \lambda_{i,i-1}, \ldots, \lambda_{n,n-1}$ and $\lambda_{3,1}, \ldots, \lambda_{i,i-2}, \ldots, \lambda_{n,n-2}$. The two diagonals are the only elements of $L$ to be stored and this should be done in no more than $2n$ storage locations.

4. A subroutine that computes the product $M = LU$ given the matrices $L$ and $U$ where $L$ is a unit lower triangular matrix and $U$ is an upper triagular matrix that are stored together in a single 2-D array. Note that combined $L$ and $U$ have exactly $n^2$ potentially nonzero elements that must be specified other than the known elements that must be 0 and 1 to enforce the assumed structure of the matrices.

For these routines you may use any of the approaches described in the class notes, i.e., inner product, middle product, outer product etc. You may implement these codes using a compiled typed language such as C, Fortran, C++ or in a coding/problem solving environment such as Matlab or Python. However, your coding style must be "basic", i.e., you must write code that exposes clearly the loop or vector control constructs and the singly and doubly indexed data structures required.

For the banded unit lower triangular matrix-vector product, your routine should be organized in such a way that vectors as long as possible are used. See the solution in the study questions (Problem 1.6) that explains the implementation of a matrix-vector product $Tv \rightarrow w$ where $T \in \mathbb{R}^{n \times n}$ tridiagonal matrix.

Your subroutines must be callable from a driver code that implements your testing and validation approaches in a manner similar to the one provided.

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 not use library routines inside your assigned subroutines**.

# Routine Development

As with our discussions in class concerning the basic primitives for matrices and vectors and as will be discussed for triangular systems solving next week, you are encouraged to start with the definition of the dense matrix-vector product $y \leftarrow Ax$ using the various mathematical

specifications of the operation

$$A \in \mathbb{R}^{n\times n},\ \ e_i^T A e_j = \alpha_{ij},\ x \in \mathbb{R}^n,\ e_i^T x = \xi_i,\ y \in \mathbb{R}^n,\ e_i^T y = \eta_i$$

$$\eta_i = \sum_{j=1}^{n} \alpha_{ij}\xi_j = \left(e_i^T A\right) x,\ \ 1 \le i \le n$$

$$\begin{aligned} y &= (Ae_1)\xi_1 + (Ae_2)\xi_2 + \cdots + (Ae_n)\xi_n \\ &= \begin{pmatrix} \alpha_{11} \\ \alpha_{21} \\ \vdots \\ \alpha_{n1} \end{pmatrix} \xi_1 + \begin{pmatrix} \alpha_{12} \\ \alpha_{22} \\ \vdots \\ \alpha_{n2} \end{pmatrix} \xi_2 + \cdots + \begin{pmatrix} \alpha_{1n} \\ \alpha_{2n} \\ \vdots \\ \alpha_{nn} \end{pmatrix} \xi_n \end{aligned}$$

$$y = \begin{pmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_n \end{pmatrix} = \begin{pmatrix} e_1^T Ax \\ e_2^T Ax \\ \vdots \\ e_n^T Ax \end{pmatrix}$$

and then imposing the structural requirments to simplify, first the mathematical description and then mathematical pseudocode. The latter might need to be rearranged to expose the desired long vector operations as was done in the solution for Study Question 1.6 for a tridiagonal matrix. Then the last step is to map the mathematical objects to the assumed data structure locations and minimizing additional work storage for intermediate results.

A similar approach should be taken for the matrix-matrix product. This routine will use all of the structure exploitation seen in the routines for $w \leftarrow Uv$ provided and the $w \leftarrow Lv$ you design and implement. You need only implement one approach out of the three: inner, middle and outer product, but you are encouraged to look at all three versions when solutions are provided.

# Data Structures

The key issue for exploiting structure such as triangular, unit triangular and banded is to only store values that are not known due to the structure assumed. These values have particular positions in the matrices and may take on any value including 0 but since they may be nonzero a storage location must be provided. Any element in a position required to be 0 or 1 should not be stored and the influence of the restriction on its value on the associated operation must be exploited, e.g., a 0 never generates a multiplication or addition; and and 1 never generates a multiplication.

**2-D Array:** $L$ and $U$ can be stored individually or simultaneously in a doubly indexed array, e.g., ARRAY(1:RDIM,1:CDIM), where RDIM and CDIM are dimensions that are greater than or equal to $n$ for the problems in a set you are testing and the colon notation indicates a range of index values (like Matlab or any vector supporting language). Note that this means in general that RDIM and CDIM will not be the same as $n$ for most problems. This is encouraged since it is not unusual in practice.

When storing $U$ the 0's below the main diagonal are not stored in ARRAY, i.e., ARRAY(I,J) $= \mu_{ij}$, with $I = i$, $J = j$ and $i \leq j$. Other elements of ARRAY must be viewed as undefined when computing with $U$.

Similarly, when storing $L$ the 0's above the main diagonal and the 1's on the main diagonal are not stored in ARRAY, i.e., ARRAY(I,J) $= \lambda_{ij}$, with $I = i$, $J = j$ and $j < i$. Other elements of ARRAY must be viewed as undefined when computing with $L$.

When testing your matrix-matrix product routine to compute $M = LU$, $L$ and $U$ should both be stored in ARRAY and $M$ should be placed in a second doubly indexed array MARRAY(1:MRDIM,1:MCDIM) that need not be the same size as ARRAY but of course must have row and column dimensions large than $n$.

Input and output vectors for the routines should be stored in appropiate singly index arrays.

**Compressed Column and Row Triangular Storage:** If a triangular or unit lower triangular (similarly for upper) is not involved in computations with another matrix that can be complementarily stored in a 2-D array then fewer than $n^2$ locations are required. Two standard methods to map a lower triangular matrix to a singly indexed 1-D array ARRAY[I], $I = 1, \ldots, S$ where $S = n(n+1)/2$ for a lower triangular matrix and $n$ fewer for a unit lower triangular matrix is to store columns or rows togther and sequentially in the array. For example, if $n = 4$ then

$$L = \begin{pmatrix} \lambda_{11} & 0 & 0 & 0 \\ \lambda_{21} & \lambda_{22} & 0 & 0 \\ \lambda_{31} & \lambda_{32} & \lambda_{33} & 0 \\ \lambda_{41} & \lambda_{42} & \lambda_{43} & \lambda_{44} \end{pmatrix}$$

and $S = 10$ and the dimension of ARRAY must be at least $S$ with the elements of $L$ mapped as

$$ARRAY[1:S] = [\lambda_{11}, \lambda_{21}, \lambda_{31}, \lambda_{41}, \lambda_{22}, \lambda_{32}, \lambda_{42}, \lambda_{33}, \lambda_{43}, \lambda_{44}]$$

$$ARRAY[1:S] = [\lambda_{11}, \lambda_{21}, \lambda_{22}, \lambda_{31}, \lambda_{32}, \lambda_{33}, \lambda_{41}, \lambda_{42}, \lambda 43, \lambda_{44}]$$

for compressed column and row respectively. These typically are used with the middle and inner product approaches respectively. If $L$ is a unit lower triangular matrix then the $\lambda_{ii}$ elements are not stored.

**Banded Lower Triangular Storage:** If $L$ is a unit lower triangular matrix with its other potentially nonzero locations restricted to its two subdiagonals $\lambda_{2,1}, \ldots, \lambda_{i,i-1}, \ldots, \lambda_{n,n-1}$ and $\lambda_{3,1}, \ldots, \lambda_{i,i-2}, \ldots, \lambda_{n,n-2}$ then the subdiagonals are best stored in a 2-D array ARRAY[NDIM,2] with $NDIM \geq n$ and the assignment, for example for $n = 5$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ \lambda_{21} & 1 & 0 & 0 & 0 \\ \lambda_{31} & \lambda_{32} & 1 & 0 & 0 \\ 0 & \lambda_{42} & \lambda_{43} & 1 & 0 \\ 0 & 0 & \lambda_{53} & \lambda_{54} & 1 \end{pmatrix}$$

and

$$ARRAY[1:4,1] = [--, \lambda_{3,1}, \lambda_{4,2}, \lambda_{5,3}]$$
$$ARRAY[1:4,2] = [\lambda_{2,1}, \lambda_{3,2}, \lambda_{4,3}, \lambda_{5,4}]$$

where the first location in the first column of ARRAY is not used but the location is included to use a simple 2-D ARRAY with 2 columns.

# Tests

Your testing must demonstrate the correctness of your codes. This does not mean you run them on small number of problems and verify manually the results. While such small simple cases are a useful part of the validation procedure you should also:

1. Run a range of problem sizes from $n = 10$ to $n$ in the hundreds.
2. Generate problems for which you know the result analytically, i.e., not as the result of running a library code. For example, a matrix with integer entries times a vector of all 1s must give row sums.
3. Generate problems for which you know the result and compare them to a library code, e.g., Matlab library routines.
4. Generate problems for which you know the result using your routines.For example, once you have validated your matrix vector product routines you can generate a matrix $M$ given $L$ and $U$ and exploiting the 0s in the column or row vectors.
5. You may use library codes to generate matrices and vectors, e.g., random matrix and vector generators in Matlab, and then to check the correctness of your routines.
6. When running a large number of problems over a range of $n$ values and $w$ and $v$ vectors, you should not simply display the result of each subroutine. That is not acceptable and not useful. You should present your results in a summarizing fashion. For example,

when comparing your computed results, $w_{comp}$, to true results, $w_{true}$, (however they are known).

When doing so, it is useful to examine the absolute and relative errors

$$\epsilon_{abs} = \|w_{comp} - w_{true}\| \quad \text{and} \quad \epsilon_{rel} = \epsilon_{abs}/\|w_{true}\|,$$

where $\|z\|$, is a chosen vector norm, and organizing their values over various sets, e.g., fixed $L$ or $U$ and many input vectors for matrix-vector products. (Of course, for the relative error the norm of the true result should be kept away from a small number.)

Form comparing $M_{comp}$ to $M_{true} = LU$ similar expressions should be used. The matrix norms $\|M\|_1$, $\|M\|_\infty$ and $\|M\|_F$ are all finite in complexity and easily computed. The matrix 2-norm, $\|M\|_2$, should be computed using a library routine, e.g., in Matlab, if you decided to use it.

Histograms, means and variances of the error are useful. Selected plotting of trends can also be convincing. Note that this is one of the main lessons of the assignment – thinking about how to present in a coherent and convincing manner the correctness of your codes to a reader. This computation, organization and presentation of evidence of correctness is where a computing enviorment such as Matlab is valuable. Examples of this are given in the testing driver provided.

# Submission of Results

All solutions must be submitted through Canvas. As noted in the requirements for reporting your programming assignments given on the class webpage, you should submit a document that includes the following:

1. A section that describes you choices of algorithm and the organization of your testing routines and subroutines.

2. A section describing your validation strategies. This should include:

   - how your problems are generated;
   - the range of sizes and characteristics you are using in your problems;
   - how the correctness is being evaluated;
   - what information is displayed in your evaluation section and why it is appropriate for assessing the correctness of your codes.

3. One or more sections presenting your evidence of correctness of your subroutines and your arguments supporting the assertion of correctness given the evidence.

You should also submit your codes. You may be called upon to demonstrate their usage and to reproduce evidence contained in your document.