

Hints for Homework 5

1. (Improved triangular substitutions; adapted from **FNC** 2.3.5)

(a) The function `backsub` handles the case in which A is upper triangular; call it U :

$$U \underbrace{\begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_p \end{bmatrix}}_{=X} = \underbrace{\begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_p \end{bmatrix}}_{=B}.$$

For any $j \in \mathbb{N}[1, p]$, the solution of $U\mathbf{x}_j = \mathbf{b}_j$ is obtained by applying the backward substitution formula given in lecture:

$$\begin{cases} x_{n,j} = \frac{b_{n,j}}{u_{n,n}} & \text{and} \\ x_{i,j} = \frac{1}{u_{i,i}} \left(b_{i,j} - \sum_{k=i+1}^n u_{i,k} x_{k,j} \right) & \text{for } i = n-1, n-2, \dots, 1. \end{cases}$$

Encode these formulas using nested for-loops in your program:

- outer loop: iterate over columns ($j = 1, 2, \dots, p$)
- inner loop: implement the backward substitution formula ($i = n, n-1, \dots, 1$).

You may use the following template to begin your program `backsub.m`¹ by

```
function X = backsub(U,B)
% BACKSUB X = backsub(U,B)
% Solve multiple upper triangular linear systems.
% Input:
%   U   upper triangular square matrix (n-by-n)
%   B   right-hand side vectors concatenated into an (n-by-p) matrix
% Output:
%   X   solution of UX = B (n-by-p)
%   [n,p] = size(B);           % grab dimensions
%   X = zeros(n,p);           % preallocate output
%   %% Implement back. subs. formula (nested for-loop)

end
```

¹You do not need to write external function m-files. Simply write them at the end of the document as instructed in the problem.

The function `forelim` can be written in a very analogous manner.

Note. If you want to include a check for triangularity of the coefficient matrix, use `istriu` (upper triangular) or `istril` (lower triangular), e.g.,

```
if ~istriu(U)
    error('The matrix U must be upper triangular.');
```

The inclusion of such a check will not be part of the grading rubrics.

- (b) One way to test your code using the given examples is to compute the norm of the difference of the analytical inverse and the numerically calculated inverse. For example,

```
L1 = .....;
Llinv = .....;           % analytical inverse given
LlinvNum = ltinverse(L1); % numerically calculated inverse
norm(Llinv - LlinvNum)    % norm of the difference
```

We want to see a small norm, if not zero.

2. (Triangular substitution and stability; **FNC 2.3.6**)

- (a) In this part, you show that $\mathbf{x} = (1, 1, 1, 1, 1)^T$ solves the system for any values of α and β ; this is an analytical result.
- (b) In this second part, however, you check whether the computer indeed produces the same answer for changing values of β while α is fixed. When the problem says “Using MATLAB, solve the system ...”, it simply means that you use the backslash `\` to solve the system $A\mathbf{x} = \mathbf{b}$.

3. (Vectorizing `mylu.m`; **FNC 2.4.7**)

- (a) For this part, follow the direction given in the problem:
- delete the keyword `for` in the inner loop; (just `for`, not the rest)
 - delete the matching `end` of the inner loop;
 - put a semicolon at the end of `i = j+1:n`.

This defines `i` to be a vector of indices $(j + 1, j + 2, \dots, n)$. Passing `i` into `L` or `A` in the next two lines, the code effectively accesses the $(j + 1)^{\text{st}}$ through n^{th} rows of the respective matrices; see the slides titled "Using Vectors as Indices" in Lecture 7 on arrays.

- (b) Let $\mathbf{x} \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$, not necessarily the ones appearing in the problem. The ordinary elementwise vector/matrix notation refers to the convention of denoting the elements of a vector or a matrix using subscripts such as

- x_i : the i^{th} element of vector \mathbf{x} ;
- $A_{i,j}$: the element of A on the i^{th} row and j^{th} column.

In MATLAB, they are expressed as `x(i)` and `A(i,j)`, respectively. Now, let's concentrate on the matrix case:

$$A(\text{< rowindex >}, \text{< columnindex >})$$

When one of the two indices is a vector, as in the case of the problem, it represents the *slice* of the matrix on a specified row or column. For example,

- A row vector consisting of 5th through 8th elements on the 3rd row of A :

$$A(3, 5 : 8) \longrightarrow \begin{bmatrix} A_{3,5} & A_{3,6} & A_{3,7} & A_{3,8} \end{bmatrix}$$

- A column vector consisting of 5th through 8th elements on the 3rd column of A :

$$A(5 : 8, 3) \longrightarrow \begin{bmatrix} A_{5,3} \\ A_{6,3} \\ A_{7,3} \\ A_{8,3} \end{bmatrix}$$

When both indices are vectors, it represents the *submatrix* on the specified rows and columns. For example,

- The submatrix of A between 2nd and 4th rows and between 1st and 5th columns:

$$A(2 : 4, 1 : 5) \longrightarrow \begin{bmatrix} A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} & A_{2,5} \\ A_{3,1} & A_{3,2} & A_{3,3} & A_{3,4} & A_{3,5} \\ A_{4,1} & A_{4,2} & A_{4,3} & A_{4,4} & A_{4,5} \end{bmatrix}$$

You are to translate the MATLAB statements in your vectorized `mylu` into mathematical notations as shown above.

4. (Application of LU factorization: **FNC 2.4.6**)

- (a) Recall the following properties of the determinant from linear algebra:

- $\det(AB) = \det(A) \det(B)$.
- $\det(T) = t_{11}t_{22} \cdots t_{nn} = \prod_{i=1}^n t_{ii}$, where $T \in \mathbb{R}^{n \times n}$ is a triangular matrix.

- (b) Try to vectorize your code. The function can be written in only couple lines (excluding the function header and the `end` statement) without using any loop. It would be also useful to recall that if you only want to generate the second output of a certain function, you put `~` in place of the first output argument as a placeholder. For instance, if you only need `U` from `mylu(A)`, instead of calling it with `[L,U] = mylu(A)`, use

```
[~, U] = mylu(A);
```

Note. Use the vectorized version of `mylu` which you already wrote for Question 3.

5. (Proper usage of `lu`; **FNC 2.6.1**)

The expression `U \ L \ b` is equivalent to `(U \ L) \ b`.

6. (FLOP Counting)

- (a) Consider the following example question.

Question. How would you code $\mathbf{x} = A\mathbf{B}\mathbf{b}$ most efficiently in MATLAB, and how many *flops* are needed?

Answer. There are only two ways² to code it: `x=A*(B*b)` or `x=(A*B)*b`. If the former is used, MATLAB

- i. carries out `B*b`: matrix-by-vector multiplication ($\sim 2n^2$ *flops*)

²This is the same as `x = A*B*b`.

- ii. left-multiplies the previous result by A: another matrix-by-vector multiplication ($\sim 2n^2 \text{ flops}$)

In total, it takes $\sim 4n^2 \text{ flops}$. On the other hand, the latter version requires the matrix-by-matrix multiplication $A \star B$, which already takes $\sim 2n^3 \text{ flops}$. So the former is the efficient one.

Lesson. Avoid matrix-by-matrix multiplication as much as possible!

Also remember to use the backslash operator if matrix inversion is required, instead of `inv`. When `\` is invoked, MATLAB in general does a pivoted Gaussian elimination, which takes $\sim (2/3)n^3 \text{ flops}$.

7. (Matrix norms; Sp20 midterm)

Hints are found in the lecture for 10/01/21 (F).