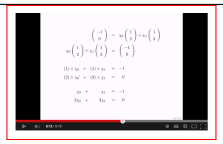
Week 6

Gaussian Elimination

6.1 Opening Remarks

6.1.1 Solving Linear Systems



☛ View at edX

6.1.2 Outline

6.1.3 What You Will Learn

Upon completion of this unit, you should be able to

- Apply Gaussian elimination to reduce a system of linear equations into an upper triangular system of equations.
- Apply back(ward) substitution to solve an upper triangular system in the form Ux = b.
- Apply forward substitution to solve a lower triangular system in the form Lz = b.
- Represent a system of equations using an appended matrix.
- Reduce a matrix to an upper triangular matrix with Gauss transforms and then apply the Gauss transforms to a right-hand side.
- Solve the system of equations in the form Ax = b using LU factorization.
- Relate LU factorization and Gaussian elimination.
- Relate solving with a unit lower triangular matrix and forward substitution.
- Relate solving with an upper triangular matrix and back substitution.
- Create code for various algorithms for Gaussian elimination, forward substitution, and back substitution.
- Determine the cost functions for LU factorization and algorithms for solving with triangular matrices.

6.2 Gaussian Elimination

6.2.1 Reducing a System of Linear Equations to an Upper Triangular System

A system of linear equations

Consider the system of linear equations

$$\begin{array}{rcrcrcr} 2x & + & 4y & - & 2z & = & -10 \\ 4x & - & 2y & + & 6z & = & 20 \\ 6x & - & 4y & + & 2z & = & 18. \end{array}$$

Notice that x, y, and z are just variables for which we can pick any symbol or letter we want. To be consistent with the notation we introduced previously for naming components of vectors, we identify them instead with χ_0, χ_1, and and χ_2, respectively:

$$\begin{array}{rcrcrcr} 2\chi_0 & + & 4\chi_1 & - & 2\chi_2 & = & -10 \\ 4\chi_0 & - & 2\chi_1 & + & 6\chi_2 & = & 20 \\ 6\chi_0 & - & 4\chi_1 & + & 2\chi_2 & = & 18. \end{array}$$

Gaussian elimination (transform linear system of equations to an upper triangular system)

Solving the above linear system relies on the fact that its solution does not change if

1. Equations are reordered (not used until next week);
2. An equation in the system is modified by subtracting a multiple of another equation in the system from it; and/or
3. Both sides of an equation in the system are scaled by a nonzero number.

These are the tools that we will employ.

The following steps are knows as (Gaussian) elimination. They transform a system of linear equations to an equivalent upper triangular system of linear equations:

- Subtract $\lambda_{1,0} = (4/2) = 2$ times the first equation from the second equation:

Before	After
$\begin{array}{rcrcrcr} 2\chi_0 & + & 4\chi_1 & - & 2\chi_2 & = & -10 \\ 4\chi_0 & - & 2\chi_1 & + & 6\chi_2 & = & 20 \\ 6\chi_0 & - & 4\chi_1 & + & 2\chi_2 & = & 18 \end{array}$	$\begin{array}{rcrcrcr} 2\chi_0 & + & 4\chi_1 & - & 2\chi_2 & = & -10 \\ & - & 10\chi_1 & + & 10\chi_2 & = & 40 \\ 6\chi_0 & - & 4\chi_1 & + & 2\chi_2 & = & 18 \end{array}$

- Subtract $\lambda_{2,0} = (6/2) = 3$ times the first equation from the third equation:

Before	After
$\begin{array}{rcrcrcr} 2\chi_0 & + & 4\chi_1 & - & 2\chi_2 & = & -10 \\ & - & 10\chi_1 & + & 10\chi_2 & = & 40 \\ 6\chi_0 & - & 4\chi_1 & + & 2\chi_2 & = & 18 \end{array}$	$\begin{array}{rcrcrcr} 2\chi_0 & + & 4\chi_1 & - & 2\chi_2 & = & -10 \\ & - & 10\chi_1 & + & 10\chi_2 & = & 40 \\ & - & 16\chi_1 & + & 8\chi_2 & = & 48 \end{array}$

- Subtract $\lambda_{2,1} = ((-16)/(-10)) = 1.6$ times the second equation from the third equation:

Before									After								
$2\chi_0$	$+$	$4\chi_1$	$-$	$2\chi_2$	$=$	-10			$2\chi_0$	$+$	$4\chi_1$	$-$	$2\chi_2$	$=$	-10		
	$-$	$10\chi_1$	$+$	$10\chi_2$	$=$	40				$-$	$10\chi_1$	$+$	$10\chi_2$	$=$	40		
	$-$	$16\chi_1$	$+$	$8\chi_2$	$=$	48						$-$	$8\chi_2$	$=$	-16		

This now leaves us with an upper triangular system of linear equations.

In the above Gaussian elimination procedure, $\lambda_{1,0}$, $\lambda_{2,0}$, and $\lambda_{2,1}$ are called the *multipliers*. Notice that their subscripts indicate the coefficient in the linear system that is being eliminated.

Back substitution (solve the upper triangular system)

The equivalent upper triangular system of equations is now solved via *back substitution*:

- Consider the last equation,

$$-8\chi_2 = -16.$$

  Scaling both sides by by $1/(-8)$ we find that

$$\chi_2 = -16/(-8) = 2.$$

- Next, consider the second equation,

$$-10\chi_1 + 10\chi_2 = 40.$$

  We know that $\chi_2 = 2$, which we plug into this equation to yield

$$-10\chi_1 + 10(2) = 40.$$

  Rearranging this we find that

$$\chi_1 = (40 - 10(2))/(-10) = -2.$$

- Finally, consider the first equation,

$$2\chi_0 + 4\chi_1 - 2\chi_2 = -10$$

  We know that $\chi_2 = 2$ and $\chi_1 = -2$, which we plug into this equation to yield

$$2\chi_0 + 4(-2) - 2(2) = -10.$$

  Rearranging this we find that

$$\chi_0 = (-10 - (4(-2) - (2)(2)))/2 = 1.$$

Thus, the solution is the vector

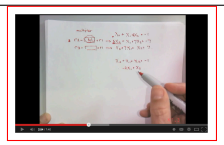
$$x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \\ 2 \end{pmatrix}.$$

Check your answer (ALWAYS!)

Check the answer (by plugging $\chi_0 = 1$, $\chi_1 = -2$, and $\chi_2 = 2$ into the original system)

$$\begin{array}{rcrcrcrl} 2(1) & + & 4(-2) & - & 2(2) & = & -10 & \checkmark \\ 4(1) & - & 2(-2) & + & 6(2) & = & 20 & \checkmark \\ 6(1) & - & 4(-2) & + & 2(2) & = & 18 & \checkmark \end{array}$$

Homework 6.2.1.1



☛ View at edX

Practice reducing a system of linear equations to an upper triangular system of linear equations by visiting the Practice with Gaussian Elimination webpage we created for you. For now, only work with the top part of that webpage.

Homework 6.2.1.2 Compute the solution of the linear system of equations given by

$$\begin{array}{rcrcrcrl} -2\chi_0 & + & \chi_1 & + & 2\chi_2 & = & 0 \\ 4\chi_0 & - & \chi_1 & - & 5\chi_2 & = & 4 \\ 2\chi_0 & - & 3\chi_1 & - & \chi_2 & = & -6 \end{array}$$

- $\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} \square \\ \square \\ \square \end{pmatrix}$

Homework 6.2.1.3 Compute the coefficients γ_0, γ_1, and γ_2 so that

$$\sum_{i=0}^{n-1} i = \gamma_0 + \gamma_1 n + \gamma_2 n^2$$

(by setting up a system of linear equations).

Homework 6.2.1.4 Compute γ_0, γ_1, γ_2, and γ_3 so that

$$\sum_{i=0}^{n-1} i^2 = \gamma_0 + \gamma_1 n + \gamma_2 n^2 + \gamma_3 n^3.$$

6.2.2 Appended Matrices

☛ View at edX

Representing the system of equations with an appended matrix

Now, in the above example, it becomes very cumbersome to always write the entire equation. The information is encoded in the coefficients in front of the χ_i variables, and the values to the right of the equal signs. Thus, we could just let

$$\left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ 4 & -2 & 6 & 20 \\ 6 & -4 & 2 & 18 \end{array}\right) \quad \text{represent} \quad \begin{array}{rcrcrcr} 2\chi_0 & + & 4\chi_1 & - & 2\chi_2 & = & -10 \\ 4\chi_0 & - & 2\chi_1 & + & 6\chi_2 & = & 20 \\ 6\chi_0 & - & 4\chi_1 & + & 2\chi_2 & = & 18. \end{array}$$

Then Gaussian elimination can simply operate on this array of numbers as illustrated next.

Gaussian elimination (transform to upper triangular system of equations)

- Subtract $\lambda_{1,0} = (4/2) = 2$ times the first row from the second row:

$$\begin{array}{c|c} \text{Before} & \text{After} \\ \left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ 4 & -2 & 6 & 20 \\ 6 & -4 & 2 & 18 \end{array}\right) & \left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ & -10 & 10 & 40 \\ 6 & -4 & 2 & 18 \end{array}\right). \end{array}$$

- Subtract $\lambda_{2,0} = (6/2) = 3$ times the first row from the third row:

$$\begin{array}{c|c} \text{Before} & \text{After} \\ \left(\begin{array}{ccc|c} 2 & 4 & 2 & -10 \\ & -10 & 10 & 40 \\ 6 & -4 & 2 & 18 \end{array}\right) & \left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ & -10 & 10 & 40 \\ & -16 & 8 & 48 \end{array}\right). \end{array}$$

- Subtract $\lambda_{2,1} = ((-16)/(-10)) = 1.6$ times the second row from the third row:

$$\begin{array}{c|c} \text{Before} & \text{After} \\ \left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ & -10 & 10 & 40 \\ & -16 & 8 & 48 \end{array}\right) & \left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ & -10 & 10 & 40 \\ & & -8 & -16 \end{array}\right). \end{array}$$

This now leaves us with an upper triangular system of linear equations.

Back substitution (solve the upper triangular system)

The equivalent upper triangular system of equations is now solved via *back substitution*:

- The final result above represents

$$\left(\begin{array}{ccc} 2 & 4 & -2 \\ & -10 & 10 \\ & & -8 \end{array}\right)\left(\begin{array}{c} \chi_0 \\ \chi_1 \\ \chi_2 \end{array}\right) = \left(\begin{array}{c} -10 \\ 40 \\ -16 \end{array}\right)$$

  or, equivalently,

$$\begin{array}{rcrcrcr} 2\chi_0 & + & 4\chi_1 & - & 2\chi_2 & = & -10 \\ & - & 10\chi_1 & + & 10\chi_2 & = & 40 \\ & & & - & 8\chi_2 & = & -16 \end{array}$$

- Consider the last equation,

$$8\chi_2 = -16.$$

  Scaling both sides by by $1/(-8)$ we find that

$$\chi_2 = -16/(-8) = 2.$$

- Next, consider the second equation,

$$-10\chi_1 + 10\chi_2 = 40.$$

  We know that $\chi_2 = 2$, which we plug into this equation to yield

$$-10\chi_1 + 10(2) = 40.$$

  Rearranging this we find that

$$\chi_1 = (40 - 10(2))/(-10) = -2.$$

- Finally, consider the first equation,

$$2\chi_0 + 4\chi_1 - 2\chi_2 = -10$$

  We know that $\chi_2 = 2$ and $\chi_1 = -2$, which we plug into this equation to yield

$$2\chi_0 + 4(-2) - 2(2) = -10.$$

  Rearranging this we find that

$$\chi_0 = (-10 - (4(-2) + (-2)(-2)))/2 = 1.$$

Thus, the solution is the vector

$$x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \\ 2 \end{pmatrix}.$$

Check your answer (ALWAYS!)

Check the answer (by plugging $\chi_0 = 1$, $\chi_1 = -2$, and $\chi_2 = 2$ into the original system)

$$\begin{array}{rcrcrcrl} 2(1) & + & 4(-2) & - & 2(2) & = & -10 & \checkmark \\ 4(1) & - & 2(-2) & + & 6(2) & = & 20 & \checkmark \\ 6(1) & - & 4(-2) & + & 2(2) & = & 18 & \checkmark \end{array}$$

Alternatively, you can check that

$$\begin{pmatrix} 2 & 4 & -2 \\ 4 & -2 & 6 \\ 6 & -4 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \\ 2 \end{pmatrix} = \begin{pmatrix} -10 \\ 20 \\ 18 \end{pmatrix} \quad \checkmark$$

Homework 6.2.2.1

☛ View at edX

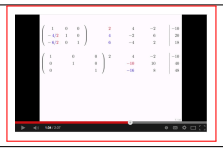
Practice reducing a system of linear equations to an upper triangular system of linear equations by visiting the Practice with Gaussian Elimination webpage we created for you. For now, only work with the top two parts of that webpage.

Homework 6.2.2.2 Compute the solution of the linear system of equations expressed as an appended matrix given by

$$\left(\begin{array}{rrr|r} -1 & 2 & -3 & 2 \\ -2 & 2 & -8 & 10 \\ 2 & -6 & 6 & -2 \end{array}\right)$$

- $\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} \square \\ \square \\ \square \end{pmatrix}$

6.2.3 Gauss Transforms



☛ View at edX

Homework 6.2.3.1
Compute ONLY the values in the boxes. A $\star$ means a value that we don't care about.

- $\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 4 & -2 & 6 \\ 6 & -4 & 2 \end{pmatrix} = \begin{pmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{pmatrix}$.
- $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 345 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 4 & -2 & 6 \\ 6 & -4 & 2 \end{pmatrix} = \begin{pmatrix} \square & \square & \square \\ \square & \square & \square \\ \star & \star & \star \end{pmatrix}$.
- $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 4 & -2 & 6 \\ 6 & -4 & 2 \end{pmatrix} = \begin{pmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{pmatrix}$.
- $\begin{pmatrix} 1 & 0 & 0 \\ \square & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 2 & -2 & 6 \\ 6 & -4 & 2 \end{pmatrix} = \begin{pmatrix} \square & \square & \square \\ 0 & \square & \square \\ \square & \square & \square \end{pmatrix}$.
- $\begin{pmatrix} 1 & 0 & 0 \\ \square & 1 & 0 \\ \square & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 2 & -2 & 6 \\ -4 & -4 & 2 \end{pmatrix} = \begin{pmatrix} \square & \square & \square \\ 0 & \square & \square \\ 0 & \square & \square \end{pmatrix}$.
- $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \square & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 0 & -10 & 10 \\ 0 & -16 & 8 \end{pmatrix} = \begin{pmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & 0 & \square \end{pmatrix}$.
- $\begin{pmatrix} 1 & 0 & \square \\ 0 & 1 & \square \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -8 \\ 1 & 1 & -4 \\ -1 & -2 & 4 \end{pmatrix} = \begin{pmatrix} \square & \square & 0 \\ \square & \square & 0 \\ \square & \square & \square \end{pmatrix}$.

Theorem 6.1 *Let $\widehat{L}_j$ be a matrix that equals the identity, except that for $i > j$ the (i, j) elements (the ones below the diagonal in the jth column) have been replaced with $-\lambda_{i,,j}$:*

$$\widehat{L}_j = \begin{pmatrix} I_j & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -\lambda_{j+1,j} & 1 & 0 & \cdots & 0 \\ 0 & -\lambda_{j+2,j} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -\lambda_{m-1,j} & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

Then $\widehat{L}_j A$ equals the matrix A except that for $i > j$ the ith row is modified by subtracting $\lambda_{i,j}$ times the jth row from it. Such a matrix $\widehat{L}_j$ is called a Gauss transform.

Proof: Let

$$\widehat{L}_j = \begin{pmatrix} I_j & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -\lambda_{j+1,j} & 1 & 0 & \cdots & 0 \\ 0 & -\lambda_{j+2,j} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -\lambda_{m-1,j} & 0 & 0 & \cdots & 1 \end{pmatrix} \quad \text{and} \quad A = \begin{pmatrix} A_{0:j-1,:} \\ \check{a}_j^T \\ \check{a}_{j+1}^T \\ \check{a}_{j+2}^T \\ \vdots \\ \check{a}_{m-1}^T \end{pmatrix},$$

where I_k equals a $k \times k$ identity matrix, $A_{s:t,:}$ equals the matrix that consists of rows s through t from matrix A, and $\check{a}_k^T$ equals the kth row of A. Then

$$\begin{aligned} \widehat{L}_j A &= \begin{pmatrix} I_j & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -\lambda_{j+1,j} & 1 & 0 & \cdots & 0 \\ 0 & -\lambda_{j+2,j} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -\lambda_{m-1,j} & 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} A_{0:j-1,:} \\ \check{a}_j^T \\ \check{a}_{j+1}^T \\ \check{a}_{j+2}^T \\ \vdots \\ \check{a}_{m-1}^T \end{pmatrix} \\ &= \begin{pmatrix} A_{0:j-1,:} \\ \check{a}_j^T \\ -\lambda_{j+1,j}\check{a}_j^T + \check{a}_{j+1}^T \\ -\lambda_{j+2,j}\check{a}_j^T + \check{a}_{j+2}^T \\ \vdots \\ -\lambda_{m-1,j}\check{a}_j^T + \check{a}_{m-1}^T \end{pmatrix} = \begin{pmatrix} A_{0:j-1,:} \\ \check{a}_j^T \\ \check{a}_{j+1}^T - \lambda_{j+1,j}\check{a}_j^T \\ \check{a}_{j+2}^T - \lambda_{j+2,j}\check{a}_j^T \\ \vdots \\ \check{a}_{m-1}^T - \lambda_{m-1,j}\check{a}_j^T \end{pmatrix}. \end{aligned}$$

Gaussian elimination (transform to upper triangular system of equations)

- Subtract $\lambda_{1,0} = (4/2) = 2$ times the first row from the second row *and* subtract $\lambda_{2,0} = (6/2) = 3$ times the first row from the third row:

Before

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & \\ -2 & 1 & 0 & \\ -3 & 0 & 1 & \end{array}\right) \left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ 4 & -2 & 6 & 20 \\ 6 & -4 & 2 & 18 \end{array}\right)$$

After

$$\left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ 0 & -10 & 10 & 40 \\ 0 & -16 & 8 & 48 \end{array}\right)$$

- Subtract $\lambda_{2,1} = ((-16)/(-10)) = 1.6$ times the second row from the third row:

Before

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 0 & -1.6 & 1 & \end{array}\right) \left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ 0 & -10 & 10 & 40 \\ 0 & -16 & 8 & 48 \end{array}\right)$$

After

$$\left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ 0 & -10 & 10 & 40 \\ 0 & 0 & -8 & -16 \end{array}\right)$$

This now leaves us with an upper triangular appended matrix.

Back substitution (solve the upper triangular system)

As before.

Check your answer (ALWAYS!)

As before.

Homework 6.2.3.2

Practice reducing an appended sytem to an upper triangular form with Gauss transforms by visiting the Practice with Gaussian Elimination webpage we created for you. For now, only work with the top three parts of that webpage.

6.2.4 Computing Separately with the Matrix and Right-Hand Side (Forward Substitution)

Transform to matrix to upper triangular matrix

- Subtract $\lambda_{1,0} = (4/2) = 2$ times the first row from the second row *and* subtract $\lambda_{2,0} = (6/2) = 3$ times the first row from the third row:

Before	After
$\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 4 & -2 & 6 \\ 6 & -4 & 2 \end{pmatrix}$	$\begin{pmatrix} 2 & 4 & -2 \\ 2 & -10 & 10 \\ 3 & -16 & 8 \end{pmatrix}$

  Notice that we are storing the multipliers over the zeroes that are introduced.

- Subtract $\lambda_{2,1} = ((-16)/(-10)) = 1.6$ times the second row from the third row:

Before	After
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1.6 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 2 & -10 & 10 \\ 3 & -16 & 8 \end{pmatrix}$	$\begin{pmatrix} 2 & 4 & -2 \\ 2 & -10 & 10 \\ 3 & 1.6 & -8 \end{pmatrix}$

  (The transformation does not affect the $(2,0)$ element that equals 3 because we are merely storing a previous multiplier there.) Again, notice that we are storing the multiplier over the zeroes that are introduced.

This now leaves us with an upper triangular matrix *and* the multipliers used to transform the matrix to the upper triangular matrix.

Forward substitution (applying the transforms to the right-hand side)

We now take the transforms (multipliers) that were computed during Gaussian Elimination (and stored over the zeroes) and apply them to the right-hand side vector.

- Subtract $\lambda_{1,0} = 2$ times the first component from the second component *and* subtract $\lambda_{2,0} = 3$ times the first component from the third component:

Before	After
$\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix} \begin{pmatrix} -10 \\ 20 \\ 18 \end{pmatrix}$	$\begin{pmatrix} -10 \\ 40 \\ 48 \end{pmatrix}$

- Subtract $\lambda_{2,1} = 1.6$ times the second component from the third component:

Before	After
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1.6 & 1 \end{pmatrix} \begin{pmatrix} -10 \\ 40 \\ 48 \end{pmatrix}$	$\begin{pmatrix} -10 \\ 40 \\ -16 \end{pmatrix}$

The important thing to realize is that this updates the right-hand side exactly as the appended column was updated in the last unit. This process is often referred to as *forward substitution*.

Back substitution (solve the upper triangular system)

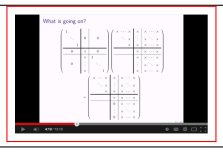
As before.

Check your answer (ALWAYS!)

As before.

Homework 6.2.4.1 No video this time! We trust that you have probably caught on to how to use the webpage. Practice reducing a matrix to an upper triangular matrix with Gauss transforms and then applying the Gauss transforms to a right-hand side by visiting the Practice with Gaussian Elimination webpage we created for you. Now you can work with all parts of the webpage. Be sure to compare and contrast!

6.2.5 Towards an Algorithm



☛ View at edX

Gaussian elimination (transform to upper triangular system of equations)

- As is shown below, compute $\begin{pmatrix} \lambda_{1,0} \\ \lambda_{2,0} \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix} / 2 = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$ and apply the Gauss transform to the matrix:

Algorithm: $A := \text{GAUSSIAN\_ELIMINATION}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

$a_{21} := a_{21}/\alpha_{11}$ $\quad (= l_{21})$

$A_{22} := A_{22} - a_{21}a_{12}^T$ $\quad (= A_{22} - l_{21}a_{12}^T)$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

endwhile

Figure 6.1: Algorithm that transforms a matrix A into an upper triangular matrix U, overwriting the uppertriangular part of A with that U. The elements of A below the diagonal are overwritten with the multipliers.

Before

$$\left(\begin{array}{c|cc} 1 & 0 & 0 \\ \hline -2 & 1 & 0 \\ -3 & 0 & 1 \end{array}\right) \left(\begin{array}{c|cc} 2 & 4 & -2 \\ \hline 4 & -2 & 6 \\ 6 & -4 & 2 \end{array}\right)$$

After

$$\left(\begin{array}{c|cc} 2 & 4 & -2 \\ \hline {\scriptstyle 2} & -10 & 10 \\ {\scriptstyle 3} & -16 & 8 \end{array}\right)$$

- As is shown below, compute $\begin{pmatrix} \lambda_{2,1} \end{pmatrix} = \begin{pmatrix} -16 \end{pmatrix} / (-10) = \begin{pmatrix} 1.6 \end{pmatrix}$ and apply the Gauss transform to the matrix:

Before

$$\left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -1.6 & 1 \end{array}\right) \left(\begin{array}{c|c|c} 2 & 4 & -2 \\ \hline {\scriptstyle 2} & -10 & 10 \\ \hline {\scriptstyle 3} & -16 & 8 \end{array}\right)$$

After

$$\left(\begin{array}{ccc} 2 & 4 & -2 \\ {\scriptstyle 2} & -10 & 10 \\ {\scriptstyle 3} & {\scriptstyle 1.6} & -8 \end{array}\right)$$

(The transformation does not affect the $(2,0)$ element that equals $_3$ because we are merely storing a previous multiplier there.)

This now leaves us with an upper triangular matrix *and* the multipliers used to transform the matrix to the upper triangular matrix.

The insights in this section are summarized in the algorithm in Figure 6.1, in which the original matrix A is overwritten with the upper triangular matrix that results from Gaussian elimination and the strictly lower triangular elements are overwritten by the multipliers.

Algorithm: $b := \text{FORWARD\_SUBSTITUTION}(A, b)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$, $b \rightarrow \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right)$

where A_{TL} is 0×0, b_T has 0 rows

while $m(A_{TL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \rightarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

$b_2 := b_2 - \beta_1 a_{21} \qquad (= b_2 - \beta_1 l_{21})$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \leftarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

endwhile

Figure 6.2: Algorithm that applies the multipliers (stored in the elements of A below the diagonal) to a right-hand side vector b.

Forward substitution (applying the transforms to the right-hand side)

We now take the transforms (multipliers) that were computed during Gaussian Elimination (and stored over the zeroes) and apply them to the right-hand side vector.

- Subtract $\lambda_{1,0} = 2$ times the first component from the second component *and* subtract $\lambda_{2,0} = 3$ times the first component from the third component:

Before	After
$\left(\begin{array}{c\|cc} 1 & 0 & 0 \\ \hline -2 & 1 & 0 \\ -3 & 0 & 1 \end{array}\right) \left(\begin{array}{c} -10 \\ \hline 20 \\ 18 \end{array}\right)$	$\left(\begin{array}{c} -10 \\ \hline 40 \\ 48 \end{array}\right)$

- Subtract $\lambda_{2,1} = 1.6$ times the second component from the third component:

Before	After
$\left(\begin{array}{c\|cc} 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -1.6 & 1 \end{array}\right) \left(\begin{array}{c} -10 \\ \hline 40 \\ \hline 48 \end{array}\right)$	$\left(\begin{array}{c} -10 \\ \hline 40 \\ \hline -16 \end{array}\right)$

The important thing to realize is that this updates the right-hand side exactly as the appended column was updated in the last unit. This process is often referred to as *forward substitution*.

The above observations motivate the algorithm for forward substitution in Figure 6.2.

Back substitution (solve the upper triangular system)

As before.

Check your answer (ALWAYS!)

As before.

Homework 6.2.5.1 Implement the algorithms in Figures 6.1 and 6.2

- `[ A_out ] = GaussianElimination( A )`
- `[ b_out ] = ForwardSubstitution( A, b )`

You can check that they compute the right answers with the following script:

- `test_GausianElimination.m`

This script exercises the functions by factoring the matrix

```
A = [
     2     0     1     2
    -2    -1     1    -1
     4    -1     5     4
    -4     1    -3    -8
]
```

by calling

```
LU = GaussianElimination( A )
```

Next, solve $Ax = b$ where

```
b = [
     2
     2
    11
    -3
]
```

by first apply forward substitution to b, using the output matrix `LU`:

```
bhat = ForwardSubstitution( LU, b )
```

extracting the upper triangular matrix U from `LU`:

```
U = triu( LU )
```

and then solving $Ux = \widehat{b}$ (which is equivalent to backward substitution) with the MATLAB intrinsic function:

```
x = U \ bhat
```

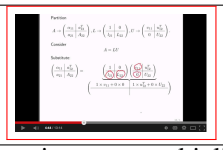
Finally, check that you got the right answer:

```
b - A * x
```

(the result should be a zero vector with four elements).

6.3 Solving $Ax = b$ via LU Factorization

6.3.1 LU factorization (Gaussian elimination)



☛ View at edX

In this unit, we will use the insights into how blocked matrix-matrix and matrix-vector multiplication works to derive and state algorithms for solving linear systems in a more concise way that translates more directly into algorithms.

The idea is that, under circumstances to be discussed later, a matrix $A \in \mathbb{R}^{n\times n}$ can be factored into the product of two matrices $L, U \in \mathbb{R}^{n\times n}$:

$$A = LU,$$

where L is unit lower triangular and U is upper triangular.

Assume $A \in \mathbb{R}^{n\times n}$ is given and that L and U are to be computed such that $A = LU$, where $L \in \mathbb{R}^{n\times n}$ is unit lower triangular and $U \in \mathbb{R}^{n\times n}$ is upper triangular. We derive an algorithm for computing this operation by partitioning

$$A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right), \quad L \rightarrow \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array}\right), \quad \text{and} \quad U \rightarrow \left(\begin{array}{c|c} \upsilon_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array}\right).$$

Now, $A = LU$ implies (using what we learned about multiplying matrices that have been partitioned into submatrices)

$$\begin{aligned} \overbrace{\left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right)}^{A} &= \overbrace{\left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array}\right)}^{L} \overbrace{\left(\begin{array}{c|c} \upsilon_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array}\right)}^{U} \\ &= \overbrace{\left(\begin{array}{c|c} 1\times\upsilon_{11} + 0\times 0 & 1\times u_{12}^T + 0\times U_{22} \\ \hline l_{21}\upsilon_{11} + L_{22}\times 0 & l_{21}u_{12}^T + L_{22}U_{22} \end{array}\right)}^{LU}. \\ &= \overbrace{\left(\begin{array}{c|c} \upsilon_{11} & u_{12}^T \\ \hline l_{21}\upsilon_{11} & l_{21}u_{12}^T + L_{22}U_{22} \end{array}\right)}^{LU}. \end{aligned}$$

For two matrices to be equal, their elements must be equal, and therefore, if they are partitioned conformally, their submatrices must be equal:

$$\left(\begin{array}{c|c} \alpha_{11} = \upsilon_{11} & a_{12}^T = u_{12}^T \\ \hline a_{21} = l_{21}\upsilon_{11} & A_{22} = l_{21}u_{12}^T + L_{22}U_{22} \end{array}\right)$$

or, rearranging,

$$\left(\begin{array}{c|c} \upsilon_{11} = \alpha_{11} & u_{12}^T = a_{12}^T \\ \hline l_{21} = a_{21}/\upsilon_{11} & L_{22}U_{22} = A_{22} - l_{21}u_{12}^T \end{array}\right).$$

This suggests the following steps for **overwriting** a matrix A with its LU factorization:

- Partition

$$A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right).$$

- Update $a_{21} = a_{21}/\alpha_{11} (= l_{21})$. (Scale a_{21} by $1/\alpha_{11}$!)

Algorithm: $A := \text{LU\_UNB\_VAR5}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

where α_{11} is 1×1

$a_{21} := a_{21}/\alpha_{11}$ $\quad (= l_{21})$

$A_{22} := A_{22} - a_{21}a_{12}^T$ $\quad (= A_{22} - l_{21}a_{12}^T)$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

endwhile

Figure 6.3: LU factorization algorithm.

- Update $A_{22} = A_{22} - a_{21}a_{12}^T (= A_{22} - l_{21}u_{12}^T)$ (Rank-1 update!).
- Overwrite A_{22} with L_{22} and U_{22} by repeating with $A = A_{22}$.

This will leave U in the upper triangular part of A and the strictly lower triangular part of L in the strictly lower triangular part of A. The diagonal elements of L need not be stored, since they are known to equal one.

The above can be summarized in Figure 6.14. If one compares this to the algorithm GAUSSIAN_ELIMINATION we arrived at in Unit 6.2.5, you find they are identical! LU factorization **is** Gaussian elimination.

We illustrate in Fgure 6.4 how LU factorization with a 3×3 matrix proceeds. Now, compare this to Gaussian elimination with an augmented system, in Figure 6.5. It should strike you that exactly the same computations are performed with the coefficient matrix to the left of the vertical bar.

Step		a_{21}/α_{11}	$A_{22} - a_{21}a_{12}^T$
	$\left(\begin{array}{c\|c\|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$		
1-2	$\left(\begin{array}{c\|cc} -2 & -1 & 1 \\ \hline 2 & -2 & -3 \\ -4 & 4 & 7 \end{array}\right)$	$\begin{pmatrix} 2 \\ -4 \end{pmatrix} /(-2) = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$	$\begin{pmatrix} -2 & -3 \\ 4 & 7 \end{pmatrix} - \begin{pmatrix} -1 \\ 2 \end{pmatrix} \begin{pmatrix} -1 & 1 \end{pmatrix} = \begin{pmatrix} -3 & -2 \\ 6 & 5 \end{pmatrix}$
3	$\left(\begin{array}{c\|c\|c} -2 & -1 & 1 \\ \hline -1 & -3 & -2 \\ \hline 2 & 6 & 5 \end{array}\right)$	$\begin{pmatrix} 6 \end{pmatrix} /(-3) = \begin{pmatrix} -2 \end{pmatrix}$	$\begin{pmatrix} 5 \end{pmatrix} - \begin{pmatrix} -2 \end{pmatrix} \begin{pmatrix} -2 \end{pmatrix} = \begin{pmatrix} 1 \end{pmatrix}$
	$\left(\begin{array}{cc\|c} -2 & -1 & 1 \\ -1 & -3 & -2 \\ \hline 2 & -2 & 1 \end{array}\right)$		

Figure 6.4: LU factorization with a 3×3 matrix

Step	Current system	Multiplier	Operation
1	$\left(\begin{array}{ccc\|c} -2 & -1 & 1 & 6 \\ 2 & -2 & -3 & 3 \\ -4 & 4 & 7 & -3 \end{array}\right)$	$\frac{2}{-2} = -1$	$\begin{array}{rrrr} 2 & -2 & -3 & 3 \\ -1\times(-2 & -1 & 1 & 6) \\ \hline 0 & -3 & -2 & 9 \end{array}$
2	$\left(\begin{array}{ccc\|c} -2 & -1 & 1 & 6 \\ 0 & -3 & -2 & 9 \\ -4 & 4 & 7 & -3 \end{array}\right)$	$\frac{-4}{-2} = 2$	$\begin{array}{rrrr} -4 & 4 & 7 & -3 \\ -(2)\times(-2 & -1 & 1 & 6) \\ \hline 0 & 6 & 5 & -15 \end{array}$
3	$\left(\begin{array}{ccc\|c} -2 & -1 & 1 & 6 \\ 0 & -3 & -2 & 9 \\ 0 & 6 & 5 & -15 \end{array}\right)$	$\frac{6}{-3} = -2$	$\begin{array}{rrrr} 0 & 6 & 5 & -15 \\ -(-2)\times(0 & -3 & -2 & 9) \\ \hline 0 & 0 & 1 & 3 \end{array}$
4	$\left(\begin{array}{ccc\|c} -2 & -1 & 1 & 6 \\ 0 & -3 & -2 & 9 \\ 0 & 0 & 1 & 3 \end{array}\right)$		

Figure 6.5: Gaussian elimination with an augmented system.

Homework 6.3.1.1 Implement the algorithm in Figures 6.4.

- `[ A_out ] = LU_unb_var5( A )`

You can check that they compute the right answers with the following script:

- test_LU_unb_var5.m

This script exercises the functions by factoring the matrix

```
A = [
      2     0     1     2
     -2    -1     1    -1
      4    -1     5     4
     -4     1    -3    -8
]
```

by calling

```
LU = LU_unb_var5( A )
```

Next, it extracts the unit lower triangular matrix and upper triangular matrix:

```
L = tril( LU, -1 ) + eye( size( A ) )

U = triu( LU )
```

and checks if the correct factors were computed:

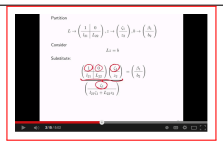
```
A - L * U
```

which should yield a 4×4 zero matrix.

Homework 6.3.1.2 Compute the LU factorization of

$$\begin{pmatrix} 1 & -2 & 2 \\ 5 & -15 & 8 \\ -2 & -11 & -11 \end{pmatrix}.$$

6.3.2 Solving $Lz = b$ (Forward substitution)



☛ View at edX

Next, we show how forward substitution is the same as solving the linear system $Lz = b$ where b is the right-hand side and L is the matrix that resulted from the LU factorization (and is thus unit lower triangular, with the multipliers from Gaussian Elimination stored below the diagonal).

Given a unit lower triangular matrix $L \in \mathbb{R}^{n \times n}$ and vectors $z, b \in \mathbb{R}^n$, consider the equation $Lz = b$ where L and b are known and z is to be computed. Partition

$$L \rightarrow \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array}\right), \quad z \rightarrow \left(\begin{array}{c} \zeta_1 \\ \hline z_2 \end{array}\right), \quad \text{and} \quad b \rightarrow \left(\begin{array}{c} \beta_1 \\ \hline b_2 \end{array}\right).$$

Algorithm: $[b] := \text{LTRSV\_UNB\_VAR1}(L, b)$

Partition $L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)$, $b \rightarrow \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right)$

where L_{TL} is 0×0, b_T has 0 rows

while $m(L_{TL}) < m(L)$ **do**

Repartition

$$\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \rightarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

where λ_{11} is 1×1, β_1 has 1 row

$b_2 := b_2 - \beta_1 l_{21}$

Continue with

$$\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \leftarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

endwhile

Figure 6.6: Algorithm for solving $Lx = b$, overwriting b with the result vector x. Here L is a lower triangular matrix.

(Recall: the horizontal line here partitions the result. It is *not* a division.) Now, $Lz = b$ implies that

$$\overbrace{\left(\begin{array}{c} \beta_1 \\ \hline b_2 \end{array}\right)}^{b} = \overbrace{\left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array}\right)}^{L} \overbrace{\left(\begin{array}{c} \zeta_1 \\ \hline z_2 \end{array}\right)}^{z}$$

$$= \overbrace{\left(\begin{array}{c} 1 \times \zeta_1 + 0 \times z_2 \\ \hline l_{21}\zeta_1 + L_{22}z_2 \end{array}\right)}^{Lz} = \overbrace{\left(\begin{array}{c} \zeta_1 \\ \hline l_{21}\zeta_1 + L_{22}z_2 \end{array}\right)}^{Lz}$$

so that

$$\frac{\beta_1 = \zeta_1}{b_2 = l_{21}\zeta_1 + L_{22}z_2} \quad \text{or, equivalently,} \quad \frac{\zeta_1 = \beta_1}{L_{22}z_2 = b_2 - \zeta_1 l_{21}}.$$

This suggests the following steps for **overwriting** the vector b with the solution vector z:

- Partition

$$L \rightarrow \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array}\right) \quad \text{and} \quad b \rightarrow \left(\begin{array}{c} \beta_1 \\ \hline b_2 \end{array}\right)$$

- Update $b_2 = b_2 - \beta_1 l_{21}$ (this is an AXPY operation!).

- Continue with $L = L_{22}$ and $b = b_2$.

This motivates the algorithm in Figure 6.15. If you compare this algorithm to FORWARD_SUBSTITUTION in Unit 6.2.5, you find them to be the same algorithm, except that matrix A has now become matrix L! So, solving $Lz = b$, overwriting b with z, **is** forward substitution when L is the unit lower triangular matrix that results from LU factorization.

We illustrate solving $Lz = b$ in Figure 6.8. Compare this to forward substitution with multipliers stored below the diagonal after Gaussian elimination, in Figure **??**.

Homework 6.3.2.1 Implement the algorithm in Figure 6.15.

- `[ b_out ] = Ltrsv_unb_var1( L, b )`

You can check that they compute the right answers with the following script:

- test_Ltrsv_unb_var1.m

This script exercises the function by setting the matrix

```
L = [
      1      0      0      0
     -1      1      0      0
      2      1      1      0
     -2     -1      1      1
]
```

and solving $Lx = b$ with the right-hand size vector

```
b = [
      2
      2
     11
     -3
]
```

by calling

```
x = Ltrsv_unb_var1( L, b )
```

Finally, it checks if `x` is indeed the answer by checking if

```
b - L * x
```

equals the zero vector.

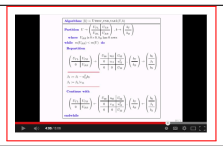
```
x = U \ z
```

We can the check if this solves $Ax = b$ by computing

```
b - A * x
```

which should yield a zero vector.

6.3.3 Solving $Ux = b$ (Back substitution)



☛ View at edX

Next, let us consider how to solve a linear system $Ux = b$. We will conclude that the algorithm we come up with is the same as backward substitution.

Step	$\left(\begin{array}{c\|c\|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right)$	$\left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$	$b_2 - l_{21}\beta_1$
1-2	$\left(\begin{array}{c\|cc} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & -2 & 1 \end{array}\right)$	$\left(\begin{array}{c} 6 \\ \hline 3 \\ -3 \end{array}\right)$	$\begin{pmatrix} 3 \\ -3 \end{pmatrix} - \begin{pmatrix} -1 \\ 2 \end{pmatrix}(6) = \begin{pmatrix} 9 \\ -15 \end{pmatrix}$
3	$\left(\begin{array}{c\|c\|c} 1 & 0 & 0 \\ \hline -1 & 1 & 0 \\ \hline 2 & -2 & 1 \end{array}\right)$	$\left(\begin{array}{c} 6 \\ \hline 9 \\ \hline -15 \end{array}\right)$	$\Big(-15\Big) - \Big(-2\Big)(9) = (3)$
	$\left(\begin{array}{cc\|c} 1 & 0 & 0 \\ -1 & 1 & 0 \\ \hline 2 & -2 & 1 \end{array}\right)$	$\left(\begin{array}{c} 6 \\ 9 \\ \hline 3 \end{array}\right)$	

Figure 6.7: Solving $Lz = b$ where L is a unit lower triangular matrix. Vector z overwrites vector b.

Step	Stored multipliers and right-hand side	Operation
1	$\left(\begin{array}{ccc\|c} - & - & - & 6 \\ -1 & - & - & 3 \\ 2 & -2 & - & -3 \end{array}\right)$	$\begin{array}{r} 3 \\ -(-1)\times(\ 6) \\ \hline 9 \end{array}$
2	$\left(\begin{array}{ccc\|c} - & - & - & 6 \\ -1 & - & - & 9 \\ 2 & -2 & - & -3 \end{array}\right)$	$\begin{array}{r} -3 \\ -(2)\times(\ 6) \\ \hline -15 \end{array}$
3	$\left(\begin{array}{ccc\|c} - & - & - & 6 \\ -1 & - & - & 9 \\ 2 & -2 & - & -15 \end{array}\right)$	$\begin{array}{r} -15 \\ -(-2)\times(\ 9) \\ \hline 3 \end{array}$
4	$\left(\begin{array}{ccc\|c} - & - & - & 6 \\ -1 & - & - & 9 \\ 2 & -2 & - & 3 \end{array}\right)$	

Figure 6.8: Forward substitutions with multipliers stored below the diagonal (e.g., as output from Gaussian Elimination).

Given upper triangular matrix $U \in \mathbb{R}^{n\times n}$ and vectors $x, b \in \mathbb{R}^n$, consider the equation $Ux = b$ where U and b are known and x is to be computed. Partition

$$U \rightarrow \left(\begin{array}{c|c} \upsilon_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array}\right), \quad x \rightarrow \left(\begin{array}{c} \chi_1 \\ \hline x_2 \end{array}\right) \quad \text{and} \quad b \rightarrow \left(\begin{array}{c} \beta_1 \\ \hline b_2 \end{array}\right).$$

Now, $Ux = b$ implies

$$\overbrace{\left(\begin{array}{c} \beta_1 \\ \hline b_2 \end{array}\right)}^{b} = \overbrace{\left(\begin{array}{c|c} \upsilon_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array}\right)}^{U} \overbrace{\left(\begin{array}{c} \chi_1 \\ \hline x_2 \end{array}\right)}^{x}$$

$$= \overbrace{\left(\begin{array}{c} \upsilon_{11}\chi_1 + u_{12}^T x_2 \\ \hline 0 \times \chi_1 + U_{22}x_2 \end{array}\right)}^{Ux} = \overbrace{\left(\begin{array}{c} \upsilon_{11}\chi_1 + u_{12}^T x_2 \\ \hline U_{22}x_2 \end{array}\right)}^{Ux}$$

so that

$$\frac{\beta_1 = \upsilon_{11}\chi_1 + u_{12}^T x_2}{b_2 = U_{22}x_2} \quad \text{or, equivalently,} \quad \frac{\chi_1 = (\beta_1 - u_{12}^T x_2)/\upsilon_{11}}{U_{22}x_2 = b_2}.$$

This suggests the following steps for overwriting the vector b with the solution vector x:

- Partition
$$U \rightarrow \left(\begin{array}{c|c} \upsilon_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array}\right), \quad \text{and} \quad b \rightarrow \left(\begin{array}{c} \beta_1 \\ \hline b_2 \end{array}\right)$$
- Solve $U_{22}x_2 = b_2$ for x_2, overwriting b_2 with the result.
- Update $\beta_1 = (\beta_1 - u_{12}^T b_2)/\upsilon_{11} (= (\beta_1 - u_{12}^T x_2)/\upsilon_{11})$.
(This requires a dot product followed by a scaling of the result by $1/\upsilon_{11}$.)

This suggests the following algorithm: Notice that the algorithm does not have "Solve $U_{22}x_2 = b_2$" as an update. The reason is that the algorithm marches through the matrix from the bottom-right to the top-left and through the vector from bottom to top. Thus, for a given iteration of the while loop, all elements in x_2 have already been computed and have overwritten b_2. Thus, the "Solve $U_{22}x_2 = b_2$" has already been accomplished by prior iterations of the loop. As a result, in this iteration, only β_1 needs to be updated with χ_1 via the indicated computations.

Homework 6.3.3.1 Side-by-side, solve the upper triangular linear system

$$\begin{array}{rrrr} -2\chi_0 - & \chi_1 + & \chi_2 = & 6 \\ -3\chi_1 - & 2\chi_2 = & & 9 \\ & \chi_2 = & & 3 \end{array}$$

via back substitution and by executing the above algorithm with

$$U = \begin{pmatrix} -2 & -1 & 1 \\ 0 & -3 & -2 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 6 \\ 9 \\ 3 \end{pmatrix}.$$

Compare and contrast!

Algorithm: $[b] := \text{UTRSV\_UNB\_VAR1}(U, b)$

Partition $U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right), b \rightarrow \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right)$

where U_{BR} is 0×0, b_B has 0 rows

while $m(U_{BR}) < m(U)$ **do**

Repartition

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & \upsilon_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \rightarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

$\beta_1 := \beta_1 - u_{12}^T b_2$

$\beta_1 := \beta_1 / \upsilon_{11}$

Continue with

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & \upsilon_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \leftarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

endwhile

Figure 6.9: Algorithm for solving $Ux = b$ where U is an uppertriangular matrix. Vector b is overwritten with the result vector x.

Homework 6.3.3.2 Implement the algorithm in Figure 6.16.

- `[ b_out ] = Utrsv_unb_var1( U, b )`

You can check that it computes the right answer with the following script:

- `test_Utrsv_unb_var1.m`

This script exercises the function by starting with matrix

```
U = [
      2      0      1      2
      0     -1      2      1
      0      0      1     -1
      0      0      0     -2
]
```

Next, it solves $Ux = b$ with the right-hand size vector

```
b = [
      2
      4
      3
      2
]
```

by calling

```
x = Utrsv_unb_var1( U, b )
```

Finally, it checks if `x` indeed solves $Ux = b$ by computing

```
b - U * x
```

which should yield a zero vector of size four.

6.3.4 Putting it all together to solve $Ax = b$

Now, the week started with the observation that we would like to solve linear systems. These could then be written more concisely as $Ax = b$, where $n \times n$ matrix A and vector b of size n are given, and we would like to solve for x, which is the vectors of unknowns. We now have algorithms for

- Factoring A into the product LU where L is unit lower triangular;
- Solving $Lz = b$; and
- Solving $Ux = b$.

We now discuss how these algorithms (and functions that implement them) can be used to solve $Ax = b$.

Start with

$$Ax = b.$$

If we have L and U so that $A = LU$, then we can replace A with LU:

$$\underbrace{(LU)}_{A} x = b.$$

Now, we can treat matrices and vectors alike as matrices, and invoke the fact that matrix-matrix multiplication is associative to place some convenient parentheses:

$$L(Ux) = b.$$

We can then recognize that Ux is a vector, which we can call z:

$$L \underbrace{(Ux)}_{z} = b$$

so that

$$Lz = b \quad \text{and} \quad Ux = z.$$

Thus, the following steps will solve $Ax = b$:

- Factor A into L and U so that $A = LU$ (LU factorization).
- Solve $Lz = b$ for z (forward substitution).
- Solve $Ux = z$ for x (back substitution).

This works if A has the right properties for the LU factorization to exist, which is what we will discuss next week...

Homework 6.3.4.1 Implement the function

- `[ A_out, b_out ] = Solve( A, b )`

that

- Computes the LU factorization of matrix A, $A = LU$, overwriting the upper triangular part of A with U and the strictly lower triangular part of A with the strictly lower triangular part of L. The result is then returned in variable `A_out`.
- Uses the factored matrix to solve $Ax = b$.

Use the routines you wrote in the previous subsections (6.3.1-6.3.3).
You can check that it computes the right answer with the following script:

- `test_Solve.m`

This script exercises the function by starting with matrix

```
A = [
      2      0      1      2
     -2     -1      1     -1
      4     -1      5      4
     -4      1     -3     -8
]
```

Next, it solves $Ax = b$ with

```
b = [
      2
      2
     11
     -3
]
```

by calling

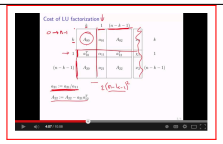
```
x = Solve( A, b )
```

Finally, it checks if `x` indeed solves $Ax = b$ by computing

```
b - A * x
```

which should yield a zero vector of size four.

6.3.5 Cost



☛ View at edX

LU factorization

Let's look at how many floating point computations are needed to compute the LU factorization of an $n \times n$ matrix A. Let's focus on the algorithm:

Assume that during the kth iteration A_{TL} is $k \times k$. Then

- A_{00} is a $k \times k$ matrix.

Algorithm: $A := \text{LU\_UNB\_VAR5}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

where α_{11} is 1×1

$a_{21} := a_{21}/\alpha_{11}$ $\quad (= l_{21})$

$A_{22} := A_{22} - a_{21}a_{12}^T$ $\quad (= A_{22} - l_{21}a_{12}^T)$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

endwhile

Figure 6.10: LU factorization algorithm.

- a_{21} is a column vector of size $n-k-1$.
- a_{12}^T is a row vector of size $n-k-1$.
- A_{22} is a $(n-k-1) \times (n-k-1)$ matrix.

Now,

- a_{21}/α_{11} is typically implemented as $(1/\alpha_{11}) \times a_{21}$ so that only one division is performed (divisions are EXPENSIVE) and $(n-k-1)$ multiplications are performed.
- $A_{22} := A_{22} - a_{21}a_{12}^T$ is a rank-1 update. In a rank-1 update, for each element in the matrix one multiply and one add (well, subtract in this case) is performed, for a total of $2(n-k-1)^2$ floating point operations.

Now, we need to sum this over all iterations $k = 0, \ldots, (n-1)$:

$$\sum_{k=0}^{n-1} \left((n-k-1) + 2(n-k-1)^2\right) \text{ floating point operations.}$$

Here we ignore the divisions. Clearly, there will only be n of those (one per iteration of the algorithm).

Let us compute how many flops this equals.

$$
\begin{array}{l}
\sum_{k=0}^{n-1} \left((n-k-1) + 2(n-k-1)^2 \right) \\
\quad = \quad < \text{Change of variable: } p = n-k-1 \text{ so that } p = 0 \text{ when } k = n-1 \text{ and} \\
\qquad p = n-1 \text{ when } k = 0 > \\
\sum_{p=n-1}^{0} \left(p + 2p^2 \right) \\
\quad = \quad < \text{Sum in reverse order} > \\
\sum_{p=0}^{n-1} \left(p + 2p^2 \right) \\
\quad = \quad < \text{Split into two sums} >
\end{array}
$$

$$
\begin{array}{l}
\sum_{p=0}^{n-1} p + \sum_{p=0}^{n-1} (2p^2) \\
\quad = \quad < \text{Results from Week 2!} > \\
\frac{(n-1)n}{2} + 2\frac{(n-1)n(2n-1)}{6} \\
\quad = \quad < \text{Algebra} > \\
\frac{3(n-1)n}{6} + 2\frac{(n-1)n(2n-1)}{6} \\
\quad = \quad < \text{Algebra} > \\
\frac{(n-1)n(4n+1)}{6}
\end{array}
$$

Now, when n is large $n-1$ and $4n+1$ equal, approximately, n and $4n$, respectively, so that the cost of LU factorization equals, approximately,

$$\frac{2}{3}n^3 \text{ flops.}$$

Forward substitution

Next, let us look at how many flops are needed to solve $Lx = b$. Again, focus on the algorithm: Assume that during the kth iteration L_{TL} is $k \times k$. Then

- L_{00} is a $k \times k$ matrix.
- l_{21} is a column vector of size $n-k-1$.
- b_2 is a column vector of size $n-k-1$.

Now,

- The `axpy` operation $b_2 := b_2 - \beta_1 l_{21}$ requires $2(n-k-1)$ flops since the vectors are of size $n-k-1$.

We need to sum this over all iterations $k = 0, \ldots, (n-1)$:

$$\sum_{k=0}^{n-1} (n-k-1) \text{ flops.}$$

Let us compute how many flops this equals:

$$
\begin{array}{l}
\sum_{k=0}^{n-1} 2(n-k-1) \\
\quad = \quad < \text{Factor out } 2 > \\
2\sum_{k=0}^{n-1} (n-k-1) \\
\quad = \quad < \text{Change of variable: } p = n-k-1 \text{ so that } p = 0 \text{ when } k = n-1 \text{ and} \\
\qquad p = n-1 \text{ when } k = 0 > \\
2\sum_{p=n-1}^{0} 2p \\
\quad = \quad < \text{Sum in reverse order} >
\end{array}
$$

Algorithm: $[b] := \text{LTRSV\_UNB\_VAR1}(L, b)$

Partition $L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)$, $b \rightarrow \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right)$

where L_{TL} is 0×0, b_T has 0 rows

while $m(L_{TL}) < m(L)$ **do**

Repartition

$$\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \rightarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

where λ_{11} is 1×1 , β_1 has 1 row

$b_2 := b_2 - \beta_1 l_{21}$

Continue with

$$\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \leftarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

endwhile

Figure 6.11: Algorithm for solving $Lx = b$, overwriting b with the result vector x. Here L is a lower triangular matrix.

$$\begin{array}{ll} & 2\sum_{p=0}^{n-1} p \\ = & \quad < \text{Results from Week 2! } > \\ & 2\frac{(n-1)n}{2} \\ = & \quad < \text{Algebra } > \\ & (n-1)n. \end{array}$$

Now, when n is large $n-1$ equals, approximately, n so that the cost for the forward substitution equals, approximately,

$$n^2 \text{ flops.}$$

Back substitution

Finally, let us look at how many flops are needed to solve $Ux = b$. Focus on the algorithm:

Algorithm: $[b] := \text{UTRSV\_UNB\_VAR1}(U, b)$

Partition $U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right)$, $b \rightarrow \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right)$

where U_{BR} is 0×0, b_B has 0 rows

while $m(U_{BR}) < m(U)$ **do**

Repartition

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & \upsilon_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \rightarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

$\beta_1 := \beta_1 - u_{12}^T b_2$

$\beta_1 := \beta_1 / \upsilon_{11}$

Continue with

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & \upsilon_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \leftarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

endwhile

Figure 6.12: Algorithm for solving $Ux = b$ where U is an uppertriangular matrix. Vector b is overwritten with the result vector x.

Homework 6.3.5.1 Assume that during the kth iteration U_{BR} is $k \times k$. (Notice we are purposely saying that U_{BR} is $k \times k$ because this algorithm moves in the opposite direction!)
Then answer the following questions:

- U_{22} is a ?×? matrix.
- u_{12}^T is a column/row vector of size ????.
- b_2 is a column vector of size ???.

Now,

- The `axpy`/`dot` operation $\beta_1 := \beta_1 - u_{12}^T b_2$ requires ??? flops since the vectors are of size ????.

We need to sum this over all iterations $k = 0, \ldots, (n-1)$ (You may ignore the divisions):

????? flops.

Compute how many floating point operations this equal. Then, approximate the result.

Total cost

The total cost of first factoring A and then performing forward and back substitution is, approximately,

$$\frac{2}{3}n^3 + n^2 + n^2 = \frac{2}{3}n^3 + 2n^2 \text{ flops.}$$

When n is large n^2 is very small relative to n^3 and hence the total cost is typically given as

$$\frac{2}{3}n^3 \text{ flops.}$$

Notice that this explains why we prefer to do the LU factorization separate from the forward and back substitutions. If we solve $Ax = b$ via these three steps, and afterwards a new right-hand side b comes along with which we wish to solve, then we need not refactor A since we already have L and U (overwritten in A). But it is the factorization of A where most of the expense is, so solving with this new right-hand side is almost free.

6.4 Enrichment

6.4.1 Blocked LU Factorization

What you saw in Week 5, Units 5.4.1 and 5.4.2, was that by carefully implementing matrix-matrix multiplication, the performance of this operation could be improved from a few percent of the peak of a processor to better than 90%. This came at a price: clearly the implementation was not nearly as "clean" and easy to understand as the routines that you wrote so far in this course.

Imagine implementing all the operations you have encountered so far in this way. When a new architecture comes along, you will have to reimplement to optimize for that architecture. While this guarantees job security for those with the skill and patience to do this, it quickly becomes a distraction from more important work.

So, how to get around this problem? Widely used linear algebra libraries like LAPACK (written in Fortran)

E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen.
LAPACK Users' guide (third ed.).
SIAM, 1999.

and the `libflame` library (developed as part of our FLAME project and written in C)

F. G. Van Zee, E. Chan, R. A. van de Geijn, E. S. Quintana-Orti, G. Quintana-Orti.
The libflame Library for Dense Matrix Computations.
IEEE Computing in Science and Engineering, Vol. 11, No 6, 2009.

F. G. Van Zee.
libflame: The Complete Reference.
www.lulu.com , 2009

implement many linear algebra operations so that most computation is performed by a call to a matrix-matrix multiplication routine. The `libflame` library is coded with an API that is very similar to the FLAME@lab API that you have been using for your routines.

More generally, in the scientific computing community there is a set of operations with a standardized interface known as the Basic Linear Algebra Subprograms (BLAS) in terms of which applications are written.

C. L. Lawson, R. J. Hanson, D. R. Kincaid, F. T. Krogh.
Basic Linear Algebra Subprograms for Fortran Usage.
ACM Transactions on Mathematical Software, 1979.

J. J. Dongarra, J. Du Croz, S. Hammarling, R. J. Hanson.
An Extended Set of FORTRAN Basic Linear Algebra Subprograms.
ACM Transactions on Mathematical Software, 1988.

J. J. Dongarra, J. Du Croz, S. Hammarling, I. Duff.
A Set of Level 3 Basic Linear Algebra Subprograms.
ACM Transactions on Mathematical Software, 1990.

F. G. Van Zee, R. A. van de Geijn.
BLIS: A Framework for Rapid Instantiation of BLAS Functionality.
ACM Transactions on Mathematical Software, to appear.

It is then expected that *someone* optimizes these routines. When they are highly optimized, any applications and libraries written in terms of these routines also achieve high performance.

In this enrichment, we show how to cast LU factorization so that most computation is performed by a matrix-matrix multiplication. Algorithms that do this are called *blocked* algorithms.

Blocked LU factorization

It is difficult to describe how to attain a blocked LU factorization algorithm by starting with Gaussian elimination as we did in Section 6.2, but it is easy to do so by starting with $A = LU$ and following the techniques exposed in Unit 6.3.1.

We start again by assuming that matrix $A \in \mathbb{R}^{n\times n}$ can be factored into the product of two matrices $L, U \in \mathbb{R}^{n\times n}$, $A = LU$, where L is unit lower triangular and U is upper triangular. Matrix $A \in \mathbb{R}^{n\times n}$ is given and that L and U are to be computed such that $A = LU$, where $L \in \mathbb{R}^{n\times n}$ is unit lower triangular and $U \in \mathbb{R}^{n\times n}$ is upper triangular.

We derive a blocked algorithm for computing this operation by partitioning

$$A \rightarrow \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right), \quad L \rightarrow \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array}\right), \quad \text{and} \quad U \rightarrow \left(\begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & U_{22} \end{array}\right),$$

where $A_{11}, L_{11}, U_{11} \in \mathbb{R}^{b\times b}$. The integer b is the block size for the algorithm. In Unit 6.3.1, $b = 1$ so that $A_{11} = \alpha_{11}$, $L_{11} = 1$, and so forth. Here, we typically choose $b > 1$ so that L_{11} is a unit lower triangular matrix and U_{11} is an upper triangular matrix. How to choose b is closely related to how to optimize matrix-matrix multiplication (Units 5.4.1 and 5.4.2).

Now, $A = LU$ implies (using what we learned about multiplying matrices that have been partitioned into submatrices)

$$\begin{aligned} \overbrace{\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right)}^{A} &= \overbrace{\left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array}\right)}^{L} \overbrace{\left(\begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & U_{22} \end{array}\right)}^{U} \\ &= \overbrace{\left(\begin{array}{c|c} L_{11}\times U_{11} + 0\times 0 & L_{11}\times U_{12} + 0\times U_{22} \\ \hline L_{21}\times U_{11} + L_{22}\times 0 & L_{21}\times U_{12} + L_{22}\times U_{22} \end{array}\right)}^{LU}. \\ &= \overbrace{\left(\begin{array}{c|c} L_{11}U_{11} & L_{11}U_{12} \\ \hline L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{array}\right)}^{LU}. \end{aligned}$$

For two matrices to be equal, their elements must be equal and therefore, if they are partitioned conformally, their submatrices must be equal:

$$\left(\begin{array}{c|c} A_{11} = L_{11}U_{11} & A_{12} = L_{11}U_{12} \\ \hline A_{21} = L_{21}U_{11} & A_{22} = L_{21}U_{12} + L_{22}U_{22} \end{array}\right)$$

or, rearranging,

$$\left(\begin{array}{c|c} A_{11} = L_{11}U_{11} & A_{12} = L_{11}U_{12} \\ \hline A_{21} = L_{21}U_{11} & A_{22} - L_{21}U_{12} = L_{22}U_{22} \end{array}\right)$$

This suggests the following steps for **overwriting** a matrix A with its LU factorization:

- Partition
$$A \rightarrow \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right).$$
- Compute the LU factorization of A_{11}: $A_{11} \rightarrow L_{11}U_{11}$. Overwrite A_{11} with this factorization.
  Note: one can use an unblocked algorithm for this.

- Now that we know L_{11}, we can solve $L_{11}U_{12} = A_{12}$, where L_{11} and A_{12} are given. U_{12} overwrites A_{12}.
  This is known as an triangular solve with multiple right-hand sides. More on this later in this unit.

- Now that we know U_{11} (still from the first step), we can solve $L_{21}U_{11} = A_{21}$, where U_{11} and A_{21} are given. L_{21} overwrites A_{21}.
  This is also known as a triangular solve with multiple right-hand sides. More on this also later in this unit.

- Update $A_{22} = A_{22} - A_{21}A_{12} (= A_{22} - L_{21}U_{12})$.
  This is a matrix-matrix multiplication and is where, if b is small relative to n, most of the computation is performed.

- Overwrite A_{22} with L_{22} and U_{22} by repeating the above steps with $A = A_{22}$.

This will leave U in the upper triangular part of A and the strictly lower triangular part of L in the strictly lower triangular part of A. The diagonal elements of L need not be stored, since they are known to equal one.

The above is summarized in Figure 6.13. In that figure, the derivation of the unblocked algorithm from Unit 6.3.1 is given on the left and the above derivation of the blocked algorithm is given on the right, for easy comparing and contrasting. The resulting algorithms, in FLAME notation, are given as well. It is important to note that the algorithm now progresses b rows and b columns at a time, since A_{11} is a $b \times b$ block.

Triangular solve with multiple right-hand sides

In the above algorithm, we needed to perform two subproblems:

- $L_{11}U_{12} = A_{12}$ where unit lower triangular matrix L_{11} and (general) matrix A_{12} are known and (general) matrix U_{12} is to be computed; and

- $L_{21}U_{11} = A_{21}$ where upper triangular matrix U_{11} and (general) matrix A_{21} are known and (general) matrix L_{21} is to be computed.

These operations are known as special cases of the "triangular solve with multiple right-hand side" operation.

Let's simplify the discussion to solving $LX = B$ where L is unit lower triangular and X and B are general matrices. Here L and B are known and X is to be computed. We slice and dice B and X into columns to observe that

$$L \left(\begin{array}{c|c|c} x_0 & x_1 & \cdots \end{array} \right) = \left(\begin{array}{c|c|c} b_0 & b_1 & \cdots \end{array} \right)$$

and hence

$$\left(\begin{array}{c|c|c} Lx_0 & Lx_1 & \cdots \end{array} \right) = \left(\begin{array}{c|c|c} b_0 & b_1 & \cdots \end{array} \right).$$

We therefore conclude that $Lx_j = b_j$ for all pairs of columns x_j and b_j. But that means that to compute x_j from L and b_j we need to solve with a unit lower triangular matrix L. Thus the name "triangular solve with multiple right-hand sides". The multiple right-hand sides are the columns b_j.

Now let us consider solving $XU = B$, where U is upper triangular. If we transpose both sides, we get that $(XU)^T = B^T$ or $U^T X^T = B^T$. If we partition X and B by columns so that

$$X = \left(\begin{array}{c} \widetilde{x}_0^T \\ \hline \widetilde{x}_1^T \\ \hline \vdots \end{array} \right) \quad \text{and} \quad B = \left(\begin{array}{c} \widetilde{b}_0^T \\ \hline \widetilde{b}_1^T \\ \hline \vdots \end{array} \right),$$

then

$$U^T \left(\begin{array}{c|c|c} \widetilde{x}_0 & \widetilde{x}_1 & \cdots \end{array} \right) = \left(\begin{array}{c|c|c} U^T\widetilde{x}_0 & U^T\widetilde{x}_1 & \cdots \end{array} \right) = \left(\begin{array}{c|c|c} \widetilde{b}_0 & \widetilde{b}_1 & \cdots \end{array} \right).$$

We notice that this, again, is a matter of solving multiple right-hand sides (now rows of B that have been transposed) with a lower triangular matrix (U^T). In practice, none of the matrices are transposed.

<table>
<tr><th>Unblocked algorithm</th><th>Blocked algorithm</th></tr>
<tr><td>$A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right), L \rightarrow \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array}\right), U \rightarrow \left(\begin{array}{c|c} \upsilon_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array}\right)$</td><td>$A \rightarrow \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right), L \rightarrow \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array}\right), U \rightarrow \left(\begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & U_{22} \end{array}\right)$</td></tr>
<tr><td>$\left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right) = \underbrace{\left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array}\right)\left(\begin{array}{c|c} \upsilon_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array}\right)}_{\left(\begin{array}{c|c} \upsilon_{11} & u_{12}^T \\ \hline l_{21}\upsilon_{11} & l_{21}u_{12}^T + L_{22}U_{22} \end{array}\right)}$</td><td>$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) = \underbrace{\left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array}\right)\left(\begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & U_{22} \end{array}\right)}_{\left(\begin{array}{c|c} L_{11}U_{11} & L_{11}U_{12} \\ \hline L_{21}U_{11} & L_{21}U_{12}^T + L_{22}U_{22} \end{array}\right)}$</td></tr>
<tr><td>$\begin{array}{c|c} \alpha_{11} = \upsilon_{11} & a_{12}^T = u_{12}^T \\ \hline a_{21} = l_{21}\upsilon_{11} & A_{22} = l_{21}u_{12}^T + L_{22}U_{22} \end{array}$</td><td>$\begin{array}{c|c} A_{11} = L_{11}U_{11} & A_{12} = L_{11}U_{12} \\ \hline A_{21} = L_{21}U_{11} & A_{22} = L_{21}U_{12} + L_{22}U_{22} \end{array}$</td></tr>
<tr><td>$\alpha_{11} := \alpha_{11}$
$a_{12}^T := a_{12}^T$
$a_{21} := a_{21}/\alpha_{11}$
$A_{22} := A_{22} - a_{21}a_{12}^T$</td><td>$A_{11} \rightarrow L_{11}U_{11}$ (overwriting A_{11} with L_{11} and U_{11})
Solve $L_{11}U_{12} := A_{12}$ (overwiting A_{12} with U_{12})
Solve $U_{21}U_{11} := A_{21}$ (overwiting A_{21} with U_{21})
$A_{22} := A_{22} - A_{21}A_{12}$</td></tr>
<tr><td>Algorithm: $[A] := \text{LU\_UNB\_VAR5}(A)$
Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$
where A_{TL} is 0×0
while $m(A_{TL}) < m(A)$ do
Repartition
$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$
$a_{21} := a_{21}/\alpha_{11}$
$A_{22} := A_{22} - a_{21}a_{12}^T$
Continue with
$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$
endwhile</td><td>Algorithm: $[A] := \text{LU\_BLK\_VAR5}(A)$
Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$
where A_{TL} is 0×0
while $m(A_{TL}) < m(A)$ do
Repartition
$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$
Factor $A_{11} \rightarrow L_{11}U_{11}$ (Overwrite A_{11})
Solve $L_{11}U_{12} = A_{12}$ (Overwrite A_{12})
Solve $L_{21}U_{11} = A_{21}$ (Overwrite A_{21})
$A_{22} := A_{22} - A_{21}A_{12}$
Continue with
$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$
endwhile</td></tr>
</table>

Figure 6.13: Side-by-side derivation of the unblocked and blocked algorithms.

Cost analysis

Let us analyze where computation is spent in just the first step of a blocked LU factorization. We will assume that A is $n \times n$ and that a block size of b is used:

- Partition
$$A \rightarrow \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \right).$$
This carries no substantial cost, since it just partitions the matrix.
- Compute the LU factorization of A_{11}: $A_{11} \rightarrow L_{11}U_{11}$. Overwrite A_{11} with this factorization.
One can use an unblocked algorithm for this and we saw that the cost of that algorithm, for a $b \times b$ matrix, is approximately $\frac{2}{3}b^3$.
- Now that we know L_{11}, we can solve $L_{11}U_{12} = A_{12}$, where L_{11} and A_{12} are given. U_{12} overwrites A_{12}.
This is a triangular solve with multiple right-hand sides with a matrix A_{12} that is $b \times (n-b)$. Now, each triangular solve with each column of A_{12} costs, approximately, b^2 flops for a total of $b^2(n-b)$ flops.
- Now that we know U_{11}, we can solve $L_{21}U_{11} = A_{21}$, where U_{11} and A_{21} are given. L_{21} overwrites A_{21}.
This is a triangular solve with multiple right-hand sides with a matrix A_{21} that is $(n-b) \times b$. Now, each triangular solve with each row of A_{21} costs, approximately, b^2 flops for a total of $b^2(n-b)$ flops.
- Update $A_{22} = A_{22} - A_{21}A_{12} (= A_{22} - L_{21}U_{12})$.
This is a matrix-matrix multiplication that multiplies $(n-b) \times b$ matrix A_{21} times $b \times (n-b)$ matrix A_{12} to update A_{22}. This requires $b(n-b)^2$ flops.
- Overwrite A_{22} with L_{22} and U_{22} by repeating with $A = A_{22}$, in future iterations. We don't count that here, since we said we were only going to analyze the first iteration of the blocked LU factorization.

Now, if n is much larger than b, $\frac{2}{3}b^3$ is small compared to $b^2(n-b)$ which is itself small relative to $2b(n-b)^2$. Thus, if n is much larger than b, most computational time is spent in the matrix-matrix multiplication $A_{22} := A_{22} - A_{21}A_{12}$. Since we saw in the enrichment of Week 5 that such a matrix-matrix multiplication can achieve extremely high performance, the blocked LU factorization can achieve extremely high performance (if n is large).

It is important to note that the blocked LU factorization algorithm executes exactly the same number of floating point operations as does the unblocked algorithm. It just does so in a different order so that matrix-matrix multiplication can be utilized.

More

A large number of algorithms, both unblocked and blocked, that are expressed with our FLAME notation can be found in the following technical report:

> P. Bientinesi and R. van de Geijn.
> Representing Dense Linear Algebra Algorithms: A Farewell to Indices.
> FLAME Working Note #17. The University of Texas at Austin, Department of Computer Sciences. Technical Report TR-2006-10, 2006.

It is available from the FLAME Publications webpage.

6.4.2 How Ordinary Elimination Became Gaussian Elimination

Read

> Joseph F. Grcar.
> How Ordinary Elimination Became Gaussian Elimination.

Cite as

> Joseph F. Grcar.
> How ordinary elimination became Gaussian elimination.
> Historia Math, 2011.

6.5 Wrap Up

6.5.1 Homework

There is no additional graded homework. However, we have an additional version of the "Gaussian Elimination" webpage:

- Practice with four equations in four unknowns.

Now, we always joke that in a standard course on matrix computations the class is asked to solve systems with three equations with pencil and paper. What defines an honor version of the course is that the class is asked to solve systems with four equations with pencil and paper...

Of course, there is little insight gained from the considerable extra work. However, here we have webpages that automate most of the rote work, and hence it IS worthwhile to at least observe how the methodology extends to larger systems. DO NOT DO THE WORK BY HAND. Let the webpage do the work and focus on the insights that you can gain from this.

6.5.2 Summary

Linear systems of equations

A linear system of equations with m equations in n unknowns is given by

$$\begin{array}{ccccccccc} \alpha_{0,0}\chi_0 & + & \alpha_{0,1}\chi_1 & + & \cdots & + & \alpha_{0,n-1}\chi_{n-1} & = & \beta_0 \\ \alpha_{1,0}\chi_0 & + & \alpha_{1,1}\chi_1 & + & \cdots & + & \alpha_{1,n-1}\chi_{n-1} & = & \beta_1 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ \alpha_{m-1,0}\chi_0 & + & \alpha_{m-1,1}\chi_1 & + & \cdots & + & \alpha_{m-1,n-1}\chi_{n-1} & = & \beta_{m-1} \end{array}$$

Variables $\chi_0, \chi_1, \ldots, \chi_{n-1}$ are the unknowns.

This Week, we only considered the case where $m = n$:

$$\begin{array}{ccccccccc} \alpha_{0,0}\chi_0 & + & \alpha_{0,1}\chi_1 & + & \cdots & + & \alpha_{0,n-1}\chi_{n-1} & = & \beta_0 \\ \alpha_{1,0}\chi_0 & + & \alpha_{1,1}\chi_1 & + & \cdots & + & \alpha_{1,n-1}\chi_{n-1} & = & \beta_1 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ \alpha_{n-1,0}\chi_0 & + & \alpha_{n-1,1}\chi_1 & + & \cdots & + & \alpha_{n-1,n-1}\chi_{n-1} & = & \beta_{n-1} \end{array}$$

Here the $\alpha_{i,j}$s are the coefficients in the linear system. The β_is are the right-hand side values.

Basic tools

Solving the above linear system relies on the fact that its solution does not change if

1. Equations are reordered (not used until next week);
2. An equation in the system is modified by subtracting a multiple of another equation in the system from it; and/or
3. Both sides of an equation in the system are scaled by a nonzero.

Gaussian elimination is a method for solving systems of linear equations that employs these three basic rules in an effort to reduce the system to an upper triangular system, which is easier to solve.

Appended matrices

The above system of n linear equations in n unknowns is more concisely represented as an appended matrix:

$$\left(\begin{array}{cccc|c} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} & \beta_0 \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} & \beta_1 \\ \vdots & \vdots & & \vdots & \vdots \\ \alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-1} & \beta_{n-1} \end{array}\right)$$

This representation allows one to just work with the coefficients and right-hand side values of the system.

Matrix-vector notation

The linear system can also be represented as $Ax = b$ where

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-1} \end{pmatrix}, \quad x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix}, \quad \text{and} \quad \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{n-1} \end{pmatrix}.$$

Here, A is the matrix of coefficients from the linear system, x is the solution vector, and b is the right-hand side vector.

Gauss transforms

A Gauss transform is a matrix of the form

$$L_j = \begin{pmatrix} I_j & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -\lambda_{j+1,j} & 1 & 0 & \cdots & 0 \\ 0 & -\lambda_{j+2,j} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -\lambda_{n-1,j} & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

When applied to a matrix (or a vector, which we think of as a special case of a matrix), it subtracts $\lambda_{i,j}$ times the jth row from the ith row, for $i = j+1, \ldots, n-1$. Gauss transforms can be used to express the operations that are inherently performed as part of Gaussian elimination with an appended system of equations.

The action of a Gauss transform on a matrix, $A := L_j A$ can be described as follows:

$$\begin{pmatrix} I_j & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -\lambda_{j+1,j} & 1 & 0 & \cdots & 0 \\ 0 & -\lambda_{j+2,j} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -\lambda_{n-1,j} & 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} A_0 \\ \widetilde{a}_j^T \\ \widetilde{a}_{j+1}^T \\ \vdots \\ \widetilde{a}_{n-1}^T \end{pmatrix} = \begin{pmatrix} A_0 \\ \widetilde{a}_j^T \\ \widetilde{a}_{j+1}^T - \lambda_{j+1,j}\widetilde{a}_j^T \\ \vdots \\ \widetilde{a}_{n-1}^T - \lambda_{n-1,j}\widetilde{a}_j^T \end{pmatrix}.$$

An important observation that was NOT made clear enough this week is that the rows of A are updates with an AXPY**! A multiple of the jth row is subtracted from the ith row.**

A more concise way to describe a Gauss transforms is

$$\widetilde{L} = \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array}\right).$$

Now, applying to a matrix A, $\widetilde{L}A$ yields

$$\left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array}\right) \left(\begin{array}{c} A_0 \\ \hline a_1^T \\ \hline A_2 \end{array}\right) = \left(\begin{array}{c} A_0 \\ \hline a_1^T \\ \hline A_2 - l_{21}a_1^T \end{array}\right).$$

In other words, A_2 is updated with a rank-1 update. **An important observation that was NOT made clear enough this week is that a rank-1 update can be used to simultaneously subtract multiples of a row of A from other rows of A.**

Algorithm: $A := \text{LU\_UNB\_VAR5}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

where α_{11} is 1×1

$a_{21} := a_{21}/\alpha_{11}$ $\quad (= l_{21})$

$A_{22} := A_{22} - a_{21}a_{12}^T$ $\quad (= A_{22} - l_{21}a_{12}^T)$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

endwhile

Figure 6.14: LU factorization algorithm.

Forward substitution

Forward substitution applies the same transformations that were applied to the matrix to a right-hand side vector.

Back(ward) substitution

Backward substitution solves the upper triangular system

$$\begin{array}{ccccccccc} \alpha_{0,0}\chi_0 & + & \alpha_{0,1}\chi_1 & + & \cdots & + & \alpha_{0,n-1}\chi_{n-1} & = & \beta_0 \\ & & \alpha_{1,1}\chi_1 & + & \cdots & + & \alpha_{1,n-1}\chi_{n-1} & = & \beta_1 \\ & & & \ddots & & & \vdots & \vdots & \vdots \\ & & & & & & \alpha_{n-1,n-1}\chi_{n-1} & = & \beta_{n-1} \end{array}$$

This algorithm overwrites b with the solution x.

LU factorization

The LU factorization factorization of a square matrix A is given by $A = LU$, where L is a unit lower triangular matrix and U is an upper triangular matrix. An algorithm for computing the LU factorization is given by

This algorithm overwrites A with the matrices L and U. Since L is unit lower triangular, its diagonal needs not be stored.

The operations that compute an LU factorization are the same as the operations that are performed when reducing a system of linear equations to an upper triangular system of equations.

Algorithm: $[b] := \text{LTRSV\_UNB\_VAR1}(L, b)$

Partition $L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)$, $b \rightarrow \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right)$

where L_{TL} is 0×0, b_T has 0 rows

while $m(L_{TL}) < m(L)$ **do**

Repartition

$$\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \rightarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

where λ_{11} is 1×1 , β_1 has 1 row

$b_2 := b_2 - \beta_1 l_{21}$

Continue with

$$\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \leftarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

endwhile

Figure 6.15: Algorithm for solving $Lx = b$, overwriting b with the result vector x. Here L is a lower triangular matrix.

Solving $Lz = b$

Forward substitution is equivalent to solving a unit lower triangular system $Lz = b$. An algorithm for this is given by This algorithm overwrites b with the solution z.

Solving $Ux = b$

Back(ward) substitution is equivalent to solving an upper triangular system $Ux = b$. An algorithm for this is given by This algorithm overwrites b with the solution x.

Solving $Ax = b$

If LU factorization completes with an upper triangular matrix U that does not have zeroes on its diagonal, then the following three steps can be used to solve $Ax = b$:

- Factor $A = LU$.
- Solve $Lz = b$.
- Solve $Ux = z$.

Cost

- Factoring $A = LU$ requires, approximately, $\frac{2}{3}n^3$ floating point operations.
- Solve $Lz = b$ requires, approximately, n^2 floating point operations.

Algorithm: $[b] := \text{UTRSV\_UNB\_VAR1}(U, b)$

Partition $U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right)$, $b \rightarrow \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right)$

where U_{BR} is 0×0, b_B has 0 rows

while $m(U_{BR}) < m(U)$ **do**

Repartition

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & \upsilon_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \rightarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

$\beta_1 := \beta_1 - u_{12}^T b_2$

$\beta_1 := \beta_1 / \upsilon_{11}$

Continue with

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & \upsilon_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right) \leftarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

endwhile

Figure 6.16: Algorithm for solving $Ux = b$ where U is an uppertriangular matrix. Vector b is overwritten with the result vector x.

- Solve $Ux = z$ requires, approximately, n^2 floating point operations.