Week 8

More on Matrix Inversion

8.1 Opening Remarks

8.1.1 When LU Factorization with Row Pivoting Fails

☛ View at edX

The following statements are equivalent statements about $A \in \mathbb{R}^{n \times n}$:

- A is nonsingular.
- A is invertible.
- A^{-1} exists.
- $AA^{-1} = A^{-1}A = I$.
- A represents a linear transformation that is a bijection.
- $Ax = b$ has a unique solution for all $b \in \mathbb{R}^n$.
- $Ax = 0$ implies that $x = 0$.
- $Ax = e_j$ has a solution for all $j \in \{0, \ldots, n-1\}$.

Homework 8.1.1.1 Assume that $A, B, C \in \mathbb{R}^{n \times n}$, let $BA = C$, and B be nonsingular.

A is nonsingular if and only if C is nonsingular.

True/False

The reason the above result is important is that we have seen that LU factorization computes a sequence of pivot matrices and Gauss transforms in an effort to transform the matrix into an upper triangular matrix. We know that the permutation matrices and Gauss transforms are all nonsingular since we saw last week that inverses could be constructed. If we now look at under what circumstance LU factorization with row pivoting breaks down, we will see that with the help of the above result we can conclude that the matrix is singular (does not have an inverse).

Let us assume that a number of pivot matrices and Gauss transforms have been successfully computed by LU factorization

with partial pivoting:

$$\widetilde{L}_{k-1}P_{k-1}\cdots\widetilde{L}_0P_0\widehat{A} = \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array}\right)$$

where $\widehat{A}$ equals the original matrix with which the LU factorization with row pivoting started and the values on the right of $=$ indicate what is currently in matrix A, which has been overwritten. The following picture captures when LU factorization breaks down, for $k = 2$:

$$\overbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & -\times & 1 & 0 & 0 \\ 0 & -\times & 0 & 1 & 0 \\ 0 & -\times & 0 & 0 & 1 \end{pmatrix}}^{\widetilde{L}_1} P_1 \overbrace{\begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix}}^{\widetilde{L}_0P_0A} = \overbrace{\left(\begin{array}{cc|c|cc} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ \hline 0 & 0 & 0 & \times & \times \\ \hline 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{array}\right)}^{\left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array}\right)}.$$

Here the $\times$s are "representative" elements in the matrix. In other words, if in the current step $\alpha_{11} = 0$ and $a_{21} = 0$ (the zero vector), then no row can be found with which to pivot so that $\alpha_{11} \neq 0$, and the algorithm fails.

Now, repeated application of the insight in the homework tells us that matrix A is nonsingular if and only if the matrix to the right is nonsingular. We recall our list of equivalent conditions:

The following statements are equivalent statements about $A \in \mathbb{R}^{n\times n}$:

- A is nonsingular.
- A is invertible.
- A^{-1} exists.
- $AA^{-1} = A^{-1}A = I$.
- A represents a linear transformation that is a bijection.
- $Ax = b$ has a unique solution for all $b \in \mathbb{R}^n$.
- $Ax = 0$ implies that $x = 0$.
- $Ax = e_j$ has a solution for all $j \in \{0,\ldots,n-1\}$.
- The determinant of A is nonzero: $\det(A) \neq 0$.

It is the condition "$Ax = 0$ implies that $x = 0$" that we will use. We show that if LU factorization with partial pivoting breaks down, then there is a vector $x \neq 0$ such that $Ax = 0$ for the current (updated) matrix A:

$$\left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & 0 & a_{12}^T \\ \hline 0 & 0 & A_{22} \end{array}\right) \overbrace{\begin{pmatrix} -U_{00}^{-1}u_{01} \\ \hline 1 \\ \hline 0 \end{pmatrix}}^{x} = \begin{pmatrix} -U_{00}U_{00}^{-1}u_{01} + u_{01} \\ \hline 0 \\ \hline 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \hline 0 \\ \hline 0 \end{pmatrix}$$

We conclude that if LU factorization with partial pivoting breaks down, then the original matrix A is not nonsingular. (In other words, it is singular.)

This allows us to add another condition to the list of equivalent conditions:

The following statements are equivalent statements about $A \in \mathbb{R}^{n \times n}$:

- A is nonsingular.
- A is invertible.
- A^{-1} exists.
- $AA^{-1} = A^{-1}A = I$.
- A represents a linear transformation that is a bijection.
- $Ax = b$ has a unique solution for all $b \in \mathbb{R}^n$.
- $Ax = 0$ implies that $x = 0$.
- $Ax = e_j$ has a solution for all $j \in \{0, \ldots, n-1\}$.
- The determinant of A is nonzero: $\det(A) \neq 0$.
- LU with partial pivoting does not break down.

8.1.2 Outline

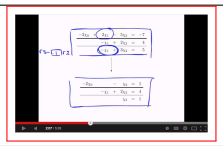
8.1.3 What You Will Learn

Upon completion of this unit, you should be able to

- Determine with Gaussian elimination (LU factorization) when a system of linear equations with n equations in n unknowns does not have a unique solution.
- Understand and apply Gauss Jordan elimination to solve linear systems with one or more right-hand sides and to find the inverse of a matrix.
- Identify properties that indicate a linear transformation has an inverse.
- Identify properties that indicate a matrix has an inverse.
- Create an algorithm to implement Gauss-Jordan elimination and determine the cost function.
- Recognize and understand that inverting a matrix is not the method of choice for solving a linear system.
- Identify specialized factorizations of matrices with special structure and/or properties and create algorithms that take advantage of this (enrichment).

8.2 Gauss-Jordan Elimination

8.2.1 Solving $Ax = b$ via Gauss-Jordan Elimination



☛ View at edX

In this unit, we discuss a variant of Gaussian elimination that is often referred to as Gauss-Jordan elimination.

Homework 8.2.1.1 Perform the following steps

- To transform the system on the left to the one on the right:

$$\begin{array}{rcrcrcr} -2\chi_0 & + & 2\chi_1 & - & 5\chi_2 & = & -7 \\ \hline 2\chi_0 & - & 3\chi_1 & + & 7\chi_2 & = & 11 \\ -4\chi_0 & + & 3\chi_1 & - & 7\chi_2 & = & -9 \end{array} \quad \longrightarrow \quad \begin{array}{rcrcrcr} -2\chi_0 & + & 2\chi_1 & - & 5\chi_2 & = & -7 \\ \hline & & -\chi_1 & + & 2\chi_2 & = & 4 \\ & & -\chi_1 & + & 3\chi_2 & = & 5 \end{array}$$

  one must subtract $\lambda_{1,0} =$ ☐ times the first row from the second row and subtract $\lambda_{2,0} =$ ☐ times the first row from the third row.

- To transform the system on the left to the one on the right:

$$\begin{array}{rcrcrcr} -2\chi_0 & + & 2\chi_1 & - & 5\chi_2 & = & -7 \\ \hline & & -\chi_1 & + & 2\chi_2 & = & 4 \\ & & -\chi_1 & + & 3\chi_2 & = & 5 \end{array} \quad \longrightarrow \quad \begin{array}{rcrcrcr} -2\chi_0 & & & - & \chi_2 & = & 1 \\ \hline & & -\chi_1 & + & 2\chi_2 & = & 4 \\ & & & & \chi_2 & = & 1 \end{array}$$

  one must subtract $\upsilon_{0,1} =$ ☐ times the second row from the first row and subtract $\lambda_{2,1} =$ ☐ times the second row from the third row.

- To transform the system on the left to the one on the right:

$$\begin{array}{rcrcrcr} -2\chi_0 & & & - & \chi_2 & = & 1 \\ & & -\chi_1 & + & 2\chi_2 & = & 4 \\ \hline & & & & \chi_2 & = & 1 \end{array} \quad \longrightarrow \quad \begin{array}{rcrcrcr} -2\chi_0 & & & & & = & 2 \\ & & -\chi_1 & & & = & 2 \\ \hline & & & & \chi_2 & = & 1 \end{array}$$

  one must subtract $\upsilon_{0,2} =$ ☐ times the third row from the first row and subtract $\upsilon_{1,2} =$ ☐ times the third row from the first row.

- To transform the system on the left to the one on the right:

$$\begin{array}{rcrcrcr} -2\chi_0 & & & & & = & 2 \\ & & -\chi_1 & & & = & 2 \\ & & & & \chi_2 & = & 1 \end{array} \quad \longrightarrow \quad \begin{array}{rcrcrcr} \chi_0 & & & & & = & -1 \\ & & \chi_1 & & & = & -2 \\ & & & & \chi_2 & = & 1 \end{array}$$

  one must multiply the first row by $\delta_{0,0} =$ ☐, the second row by $\delta_{1,1} =$ ☐, and the third row by $\delta_{2,2} =$ ☐.

- Use the above exercises to compute the vector x that solves

$$\begin{array}{rcrcrcr} -2\chi_0 & + & 2\chi_1 & - & 5\chi_2 & = & -7 \\ 2\chi_0 & - & 3\chi_1 & + & 7\chi_2 & = & 11 \\ -4\chi_0 & + & 3\chi_1 & - & 7\chi_2 & = & -9 \end{array}$$

Be sure to compare and contrast the above order of eliminating elements in the matrix to what you do with Gaussian elimination.

Homework 8.2.1.2 Perform the process illustrated in the last exercise to solve the systems of linear equations

- $$\begin{pmatrix} 3 & 2 & 10 \\ -3 & -3 & -14 \\ 3 & 1 & 3 \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} -7 \\ 9 \\ -5 \end{pmatrix}$$

- $$\begin{pmatrix} 2 & -3 & 4 \\ 2 & -2 & 3 \\ 6 & -7 & 9 \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} -8 \\ -5 \\ -17 \end{pmatrix}$$

8.2.2 Solving $Ax = b$ via Gauss-Jordan Elimination: Gauss Transforms

☛ View at edX

We again discuss Gauss-Jordan elimination, but now with an appended system.

Homework 8.2.2.1 Evaluate

- $$\left(\begin{array}{c|cc} 1 & 0 & 0 \\ \hline 1 & 1 & 0 \\ -2 & 0 & 1 \end{array}\right) \left(\begin{array}{c|cc||c} -2 & 2 & -5 & -7 \\ \hline 2 & -3 & 7 & 11 \\ -4 & 3 & -7 & -9 \end{array}\right) =$$
- $$\left(\begin{array}{c|c|c} 1 & 2 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -1 & 1 \end{array}\right) \left(\begin{array}{c|c|c||c} -2 & 2 & -5 & -7 \\ \hline 0 & -1 & 2 & 4 \\ \hline 0 & -1 & 3 & 5 \end{array}\right) =$$
- $$\left(\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & -2 \\ \hline 0 & 0 & 1 \end{array}\right) \left(\begin{array}{cc|c||c} -2 & 0 & -1 & 1 \\ 0 & -1 & 2 & 4 \\ \hline 0 & 0 & 1 & 1 \end{array}\right) =$$
- $$\left(\begin{array}{ccc} -\frac{1}{2} & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{array}\right) \left(\begin{array}{ccc||c} -2 & 0 & 0 & 2 \\ 0 & -1 & 0 & 2 \\ 0 & 0 & 1 & 1 \end{array}\right) =$$
- Use the above exercises to compute $x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix}$ that solves

$$\begin{array}{rcrcrcr} -2\chi_0 & + & 2\chi_1 & - & 5\chi_2 & = & -7 \\ 2\chi_0 & - & 3\chi_1 & + & 7\chi_2 & = & 11 \\ -4\chi_0 & + & 3\chi_1 & - & 7\chi_2 & = & -9 \end{array}$$

Homework 8.2.2.2 This exercise shows you how to use MATLAB to do the heavy lifting for Homework 8.2.2.1.
Again solve

$$\begin{array}{rcrcrcr} -2\chi_0 & + & 2\chi_1 & - & 5\chi_2 & = & -7 \\ 2\chi_0 & - & 3\chi_1 & + & 7\chi_2 & = & 11 \\ -4\chi_0 & + & 3\chi_1 & - & 7\chi_2 & = & -9 \end{array}$$

via Gauss-Jordan elimination. This time we set this up as an appended matrix:

$$\left(\begin{array}{ccc|c} -2 & 2 & -5 & -7 \\ 2 & -3 & 7 & 11 \\ -4 & 3 & -7 & -9 \end{array}\right).$$

We can enter this into MATLAB as

```
A = [
-2  2 -5  ??
 2 -3  7  ??
-4  3 -7  ??
]
```

(You enter `??`.) Create the Gauss transform, G_0, that zeroes the entries in the first column below the diagonal:

```
G0 = [
 1  0  0
??  1  0
??  0  1
]
```

(You fill in the `??`). Now apply the Gauss transform to the appended system:

```
A0 = G0 * A
```

Similarly create G_1,

```
G1 = [
 1 ??  0
 0  1  0
 0 ??  1
]
```

A_1, G_2, and A_2, where A_2 equals the appended system that has been transformed into a diagonal system. Finally, let D equal to a diagonal matrix so that $A_3 = D * A2$ has the identity for the first three columns.
You can then find the solution to the linear system in the last column.

Homework 8.2.2.3 Assume below that all matrices and vectors are partitioned "conformally" so that the operations make sense.

$$\left(\begin{array}{c|c|c} I & -u_{01} & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array}\right) \left(\begin{array}{c|c|c||c} D_{00} & a_{01} & A_{02} & b_0 \\ \hline 0 & \alpha_{11} & a_{12}^T & \beta_1 \\ \hline 0 & a_{21} & A_{22} & b_2 \end{array}\right) = \left(\begin{array}{c|c|c||c} D_{00} & a_{01} - \alpha_{11}u_{01} & A_{02} - u_{01}a_{12}^T & b_0 - \beta_1 u_{01} \\ \hline 0 & \alpha_{11} & a_{12}^T & \beta_1 \\ \hline 0 & a_{21} - \alpha_{11}l_{21} & A_{22} - l_{21}a_{12}^T & b_2 - \beta_1 l_{21} \end{array}\right)$$

Always/Sometimes/Never

Homework 8.2.2.4 Assume below that all matrices and vectors are partitioned "conformally" so that the operations make sense. Choose

- $u_{01} := a_{01}/\alpha_{11}$; and
- $l_{21} := a_{21}/\alpha_{11}$.

Consider the following expression:

$$\left(\begin{array}{c|c|c} I & -u_{01} & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array}\right) \left(\begin{array}{c|c|c||c} D_{00} & a_{01} & A_{02} & b_0 \\ \hline 0 & \alpha_{11} & a_{12}^T & \beta_1 \\ \hline 0 & a_{21} & A_{22} & b_2 \end{array}\right) = \left(\begin{array}{c|c|c||c} D_{00} & 0 & A_{02} - u_{01}a_{12}^T & b_0 - \beta_1 u_{01} \\ \hline 0 & \alpha_{11} & a_{12}^T & \beta_1 \\ \hline 0 & 0 & A_{22} - l_{21}a_{12}^T & b_2 - \beta_1 l_{21} \end{array}\right)$$

Always/Sometimes/Never

The above exercises showcase a variant on Gauss transforms that not only take multiples of a current row and add or subtract these from the rows below the current row, but also take multiples of the current row and add or subtract these from the rows above the current row:

$$\left(\begin{array}{c|c|c} I & -u_{01} & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array}\right) \left(\begin{array}{c} A_0 \\ \hline a_1^T \\ \hline A_2 \end{array}\right) = \left(\begin{array}{c} A_0 - u_{01}a_1^T \\ \hline a_1^T \\ \hline A_2 - l_{21}a_1^T \end{array}\right) \begin{array}{l} \longleftarrow \text{ Subtract multiples of } a_1^T \text{ from the rows above } a_1^T \\ \leftarrow \text{ Leave } a_1^T \text{ alone} \\ \longleftarrow \text{ Subtract multiples of } a_1^T \text{ from the rows below } a_1^T \end{array}$$

The discussion in this unit motivates the algorithm GAUSSJORDAN_PART1 in Figure 8.1, which transforms A to a diagonal matrix and updates the right-hand side accordingly, and GAUSSJORDAN_PART2 in Figure 8.2, which transforms the diagonal matrix A to an identity matrix and updates the right-hand side accordingly. The two algorithms together leave A overwritten with the identity and the vector to the right of the double lines with the solution to $Ax = b$.

The reason why we split the process into two parts is that it is easy to create problems for which only integers are encountered during the first part (while matrix A is being transformed into a diagonal). This will make things easier for us when we extend this process so that it computes the inverse of matrix A: fractions only come into play during the second, much simpler, part.

Algorithm: $[A,b] := \text{GAUSSJORDAN\_PART1}(A,b)$

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

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

$a_{01} := a_{01}/\alpha_{11}$ $(= u_{01})$

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

$A_{02} := A_{02} - a_{01}a_{12}^T$ $(= A_{02} - u_{01}a_{12}^T)$

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

$b_0 := b_0 - \beta_1 a_{01}$ $(= b_2 - \beta_1 u_{01})$

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

$a_{01} := 0$ (zero vector)

$a_{21} := 0$ (zero vector)

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 8.1: Algorithm that transforms matrix A to a diagonal matrix and updates the right-hand side accordingly.

Algorithm: $[A,b] := \text{GAUSSJORDAN\_PART2}(A,b)$

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

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

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

$\alpha_{11} := 1$

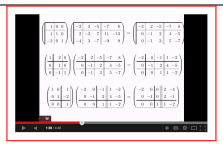
Continue with

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

endwhile

Figure 8.2: Algorithm that transforms diagonal matrix A to an identity matrix and updates the right-hand side accordingly.

8.2.3 Solving $Ax = b$ via Gauss-Jordan Elimination: Multiple Right-Hand Sides



☛ View at edX

Homework 8.2.3.1 Evaluate

- $$\left(\begin{array}{c|cc} 1 & 0 & 0 \\ \hline 1 & 1 & 0 \\ -2 & 0 & 1 \end{array}\right)\left(\begin{array}{c|cc||c|c} -2 & 2 & -5 & -7 & 8 \\ \hline 2 & -3 & 7 & 11 & -13 \\ -4 & 3 & -7 & -9 & 9 \end{array}\right) = \left(\begin{array}{c|cc||c|c} -2 & 2 & -5 & -7 & \square \\ \hline 0 & -1 & 2 & 4 & \square \\ 0 & -1 & 3 & 5 & \square \end{array}\right)$$
- $$\left(\begin{array}{c|c|c} 1 & 2 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -1 & 1 \end{array}\right)\left(\begin{array}{c|c|c||c|c} -2 & 2 & -5 & -7 & 8 \\ \hline 0 & -1 & 2 & 4 & -5 \\ \hline 0 & -1 & 3 & 5 & -7 \end{array}\right) = \left(\begin{array}{c|c|c||c|c} -2 & 0 & -1 & 1 & \square \\ \hline 0 & -1 & 2 & 4 & \square \\ \hline 0 & 0 & 1 & 1 & \square \end{array}\right)$$
- $$\left(\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & -2 \\ \hline 0 & 0 & 1 \end{array}\right)\left(\begin{array}{cc|c||c|c} -2 & 0 & -1 & 1 & -2 \\ 0 & -1 & 2 & 4 & -5 \\ \hline 0 & 0 & 1 & 1 & -2 \end{array}\right) = \left(\begin{array}{cc|c||c|c} -2 & 0 & 0 & 2 & \square \\ 0 & -1 & 0 & 2 & \square \\ \hline 0 & 0 & 1 & 1 & \square \end{array}\right)$$
- $$\left(\begin{array}{ccc} -\frac{1}{2} & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{array}\right)\left(\begin{array}{ccc||c|c} -2 & 0 & 0 & 2 & -4 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & 1 & 1 & -2 \end{array}\right) = \left(\begin{array}{ccc||c|c} 1 & 0 & 0 & -1 & \square \\ 0 & 1 & 0 & -2 & \square \\ 0 & 0 & 1 & 1 & \square \end{array}\right)$$
- Use the above exercises to compute $x_0 = \begin{pmatrix} \chi_{00} \\ \chi_{10} \\ \chi_{20} \end{pmatrix}$ and $x_1 = \begin{pmatrix} \chi_{01} \\ \chi_{11} \\ \chi_{21} \end{pmatrix}$ that solve

$$\begin{array}{rcrcrcr} -2\chi_{00} & + & 2\chi_{10} & - & 5\chi_{20} & = & -7 \\ 2\chi_{00} & - & 3\chi_{10} & + & 7\chi_{20} & = & 11 \\ -4\chi_{00} & + & 3\chi_{10} & - & 7\chi_{20} & = & -9 \end{array} \quad \text{and} \quad \begin{array}{rcrcrcr} -2\chi_{01} & + & 2\chi_{11} & - & 5\chi_{21} & = & 8 \\ 2\chi_{01} & - & 3\chi_{11} & + & 7\chi_{21} & = & -13 \\ -4\chi_{01} & + & 3\chi_{11} & - & 7\chi_{21} & = & 9 \end{array}$$

Homework 8.2.3.2 This exercise shows you how to use MATLAB to do the heavy lifting for Homework 8.2.3.1.
Start with the appended system:

$$\left(\begin{array}{ccc|cc} -2 & 2 & -5 & -7 & 8 \\ 2 & -3 & 7 & 11 & -13 \\ -4 & 3 & -7 & -9 & 9 \end{array}\right)$$

Enter this into MATLAB as

```
A = [
-2  2 -5  ?? ??
 2 -3  7  ?? ??
-4  3 -7  ?? ??
]
```

(You enter ??.) Create the Gauss transform, G_0, that zeroes the entries in the first column below the diagonal:

```
G0 = [
 1  0  0
??  1  0
??  0  1
]
```

(You fill in the ??). Now apply the Gauss transform to the appended system:

```
A0 = G0 * A
```

Similarly create G_1,

```
G1 = [
 1 ??  0
 0  1  0
 0 ??  1
]
```

A_1, G_2, and A_2, where A_2 equals the appended system that has been transformed into a diagonal system. Finally, let D equal to a diagonal matrix so that $A_3 = D * A2$ has the identity for the first three columns.
You can then find the solutions to the linear systems in the last column.

Homework 8.2.3.3 Evaluate

- $$\left(\begin{array}{c|cc} \square & 1 & 0 \\ \hline \square & 1 & 0 \\ \square & 0 & 1 \end{array}\right)\left(\begin{array}{c|cc||cc} 3 & 2 & 10 & -7 & 16 \\ \hline -3 & -3 & -14 & 9 & -25 \\ 3 & 1 & 4 & -5 & 3 \end{array}\right) = \left(\begin{array}{c|cc||cc} 3 & 2 & 10 & \square & \square \\ \hline 0 & -1 & -4 & \square & \square \\ 0 & -1 & -6 & \square & \square \end{array}\right)$$
- $$\left(\begin{array}{c|c|c} 1 & \square & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & \square & 1 \end{array}\right)\left(\begin{array}{c|cc||cc} 3 & 2 & 10 & -7 & 16 \\ \hline 0 & -1 & -4 & 2 & -9 \\ \hline 0 & -1 & -6 & 2 & -13 \end{array}\right) = \left(\begin{array}{c|cc||cc} 3 & 0 & 2 & \square & \square \\ \hline 0 & -1 & -4 & \square & \square \\ \hline 0 & 0 & -2 & \square & \square \end{array}\right)$$
- $$\left(\begin{array}{cc|c} 1 & 0 & \square \\ 0 & 1 & \square \\ \hline 0 & 0 & 1 \end{array}\right)\left(\begin{array}{cc|c||cc} 3 & 0 & 2 & -3 & -2 \\ 0 & -1 & -4 & 2 & -9 \\ \hline 0 & 0 & -2 & 0 & -4 \end{array}\right) = \left(\begin{array}{cc|c||cc} 3 & 0 & 0 & \square & \square \\ 0 & -1 & 0 & \square & \square \\ \hline 0 & 0 & -2 & \square & \square \end{array}\right)$$
- $$\left(\begin{array}{ccc} \square & 0 & 0 \\ 0 & \square & 0 \\ 0 & 0 & \square \end{array}\right)\left(\begin{array}{ccc||cc} 3 & 0 & 0 & -3 & -6 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & -2 & 0 & -4 \end{array}\right) = \left(\begin{array}{ccc||cc} 1 & 0 & 0 & \square & \square \\ 0 & 1 & 0 & \square & \square \\ 0 & 0 & 1 & \square & \square \end{array}\right)$$

Use the above exercises to compute $x_0 = \begin{pmatrix} \chi_{0,0} \\ \chi_{1,0} \\ \chi_{2,0} \end{pmatrix}$ and $x_1 = \begin{pmatrix} \chi_{0,1} \\ \chi_{1,1} \\ \chi_{2,1} \end{pmatrix}$ that solve

$$\begin{array}{rcrcrcr} 3\chi_{0,0} & + & 2\chi_{1,0} & + & 10\chi_{2,0} & = & -7 \\ -3\chi_{0,0} & - & 3\chi_{1,0} & - & 14\chi_{2,0} & = & 9 \\ 3\chi_{0,0} & + & 1\chi_{1,0} & + & 4\chi_{2,0} & = & -5 \end{array} \quad \text{and} \quad \begin{array}{rcrcrcr} 3\chi_{0,0} & + & 2\chi_{1,0} & + & 10\chi_{2,0} & = & 16 \\ -3\chi_{0,0} & - & 3\chi_{1,0} & - & 14\chi_{2,0} & = & -25 \\ 3\chi_{0,0} & + & 1\chi_{1,0} & + & 4\chi_{2,0} & = & 3 \end{array}$$

(You could use MATLAB to do the heavy lifting, like in the last homework...)

Homework 8.2.3.4 Assume below that all matrices and vectors are partitioned "conformally" so that the operations make sense.

$$\left(\begin{array}{c|c|c} I & -u_{01} & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array}\right)\left(\begin{array}{c|c|c||c} D_{00} & a_{01} & A_{02} & B_0 \\ \hline 0 & \alpha_{11} & a_{12}^T & b_1^T \\ \hline 0 & a_{21} & A_{22} & B_2 \end{array}\right) = \left(\begin{array}{c|c|c||c} D_{00} & a_{01}-\alpha_{11}u_{01} & A_{02}-u_{01}a_{12}^T & B_0-u_{01}b_1^T \\ \hline 0 & \alpha_{11} & a_{12}^T & b_1^T \\ \hline 0 & a_{21}-\alpha_{11}l_{21} & A_{22}-l_{21}a_{12}^T & B_2-l_{21}b_1^T \end{array}\right)$$

Always/Sometimes/Never

Algorithm: $[A,B] := \text{GAUSSJORDAN\_MRHS\_PART1}(A,B)$

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

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right) \rightarrow \left(\begin{array}{c} B_0 \\ \hline b_1^T \\ \hline B_2 \end{array}\right)$$

$a_{01} := a_{01}/\alpha_{11}$ $(= u_{01})$

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

$A_{02} := A_{02} - a_{01}a_{12}^T$ $(= A_{02} - u_{01}a_{12}^T)$

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

$B_0 := B_0 - a_{01}b_1^T$ $(= B_0 - u_{01}b_1^T)$

$B_2 := B_2 - a_{21}b_1^T$ $(= B_2 - l_{21}b_1^T)$

$a_{01} := 0$ (zero vector)

$a_{21} := 0$ (zero vector)

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right) \leftarrow \left(\begin{array}{c} B_0 \\ \hline b_1^T \\ \hline B_2 \end{array}\right)$$

endwhile

Figure 8.3: Algorithm that transforms diagonal matrix A to an identity matrix and updates a matrix B with multiple right-hand sides accordingly.

Homework 8.2.3.5 Assume below that all matrices and vectors are partitioned "conformally" so that the operations make sense. Choose

- $u_{01} := a_{01}/\alpha_{11}$; and
- $l_{21} := a_{21}/\alpha_{11}$.

The following expression holds:

$$\left(\begin{array}{c|c|c} I & -u_{01} & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array}\right) \left(\begin{array}{c|c|c||c} D_{00} & a_{01} & A_{02} & b_0 \\ \hline 0 & \alpha_{11} & a_{12}^T & \beta_1 \\ \hline 0 & a_{21} & A_{22} & b_2 \end{array}\right) = \left(\begin{array}{c|c|c||c} D_{00} & 0 & A_{02} - u_{01}a_{12}^T & B_0 - u_{01}b_1^T \\ \hline 0 & \alpha_{11} & a_{12}^T & b_1^T \\ \hline 0 & 0 & A_{22} - l_{21}a_{12}^T & B_2 - l_{21}b_1^T \end{array}\right)$$

Always/Sometimes/Never

The above observations justify the two algorithms in Figures 8.3 and 8.4 for "Gauss-Jordan elimination" that work with "multiple right-hand sides" (viewed as the columns of matrix B).

Algorithm: $[A,B] := \text{GAUSSJORDAN\_MRHS\_PART2}(A,B)$

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

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right) \rightarrow \left(\begin{array}{c} B_0 \\ \hline b_1^T \\ \hline B_2 \end{array}\right)$$

$b_1^T := (1/\alpha_{11}) b_1^T$

$\alpha_{11} := 1$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right) \leftarrow \left(\begin{array}{c} B_0 \\ \hline b_1^T \\ \hline B_2 \end{array}\right)$$

endwhile

Figure 8.4: Algorithm that transforms diagonal matrix A to an identity matrix and updates a matrix B with multiple right-hand sides accordingly.

8.2.4 Computing A^{-1} via Gauss-Jordan Elimination

Recall the following observation about the inverse of matrix A. If we let X equal the inverse of A, then

$$AX = I$$

or

$$A \left(\begin{array}{c|c|c|c} x_0 & x_1 & \cdots & x_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} e_0 & e_1 & \cdots & e_{n-1} \end{array} \right),$$

so that $Ax_j = e_j$. In other words, the jth column of $X = A^{-1}$ can be computed by solving $Ax = e_j$. Clearly, we can use the routine that performs Gauss-Jordan with the appended system $\left(\begin{array}{c||c} A & B \end{array} \right)$ to compute A^{-1} by feeding it $B = I$!

Homework 8.2.4.1 Evaluate

- $$\left(\begin{array}{c|cc} 1 & 0 & 0 \\ \hline 1 & 1 & 0 \\ -2 & 0 & 1 \end{array}\right) \left(\begin{array}{c|cc||c|cc} -2 & 2 & -5 & 1 & 0 & 0 \\ \hline 2 & -3 & 7 & 0 & 1 & 0 \\ -4 & 3 & -7 & 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{c|cc||c|cc} -2 & 2 & -5 & \square & \square & \square \\ \hline 0 & -1 & 2 & \square & \square & \square \\ 0 & -1 & 3 & \square & \square & \square \end{array}\right)$$
- $$\left(\begin{array}{c|c|c} 1 & 2 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -1 & 1 \end{array}\right) \left(\begin{array}{c|c|c||c|c|c} -2 & 2 & -5 & 1 & 0 & 0 \\ \hline 0 & -1 & 2 & 1 & 1 & 0 \\ \hline 0 & -1 & 3 & -2 & 0 & 1 \end{array}\right) = \left(\begin{array}{c|c|c||c|c|c} -2 & 0 & -1 & \square & \square & \square \\ \hline 0 & -1 & 2 & \square & \square & \square \\ \hline 0 & 0 & 1 & \square & \square & \square \end{array}\right)$$
- $$\left(\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & -2 \\ \hline 0 & 0 & 1 \end{array}\right) \left(\begin{array}{cc|c||cc|c} -2 & 0 & -1 & 3 & 2 & 0 \\ 0 & -1 & 2 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & -3 & -1 & 1 \end{array}\right) = \left(\begin{array}{cc|c||cc|c} -2 & 0 & 0 & \square & \square & \square \\ 0 & -1 & 0 & \square & \square & \square \\ \hline 0 & 0 & 1 & \square & \square & \square \end{array}\right)$$
- $$\left(\begin{array}{ccc} -\frac{1}{2} & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{array}\right) \left(\begin{array}{ccc||ccc} -2 & 0 & 0 & 0 & 1 & 1 \\ 0 & -1 & 0 & 7 & 3 & -2 \\ 0 & 0 & 1 & -3 & -1 & 1 \end{array}\right) = \left(\begin{array}{ccc||ccc} 1 & 0 & 0 & \square & \square & \square \\ 0 & 1 & 0 & \square & \square & \square \\ 0 & 0 & 1 & \square & \square & \square \end{array}\right)$$
- $$\left(\begin{array}{ccc} -2 & 2 & -5 \\ 2 & -3 & 7 \\ -4 & 3 & -7 \end{array}\right) \left(\begin{array}{ccc} 0 & -\frac{1}{2} & -\frac{1}{2} \\ -7 & -3 & 2 \\ -3 & -1 & 1 \end{array}\right) =$$

Homework 8.2.4.2 In this exercise, you will use MATLAB to compute the inverse of a matrix using the techniques discussed in this unit.

Initialize	`A = [ -2  2 -5`
	`  2 -3  7`
	`-4  3 -7 ]`

Create an appended matrix by appending the identity	`A_appended = [ A  eye(size(A)) ]`

Create the first Gauss transform to introduce zeros in the first column (fill in the ?s).	`G0 = [ 1  0  0`
	`  ?  1  0`
	`  ?  0  1 ]`

Apply the Gauss transform to the appended system	`A0 = G0 * A_appended`

Create the second Gauss transform to introduce zeros in the second column	`G1 = [ 1  ?  0`
	`  0  1  0`
	`  0  ?  1 ]`

Apply the Gauss transform to the appended system	`A1 = G1 * A0`

Create the third Gauss transform to introduce zeros in the third column	`G2 = [ 1  0  ?`
	`  0  1  ?`
	`  0  0  1 ]`

Apply the Gauss transform to the appended system	`A2 = G2 * A1`

Create a diagonal matrix to set the diagonal elements to one	`D3 = [ -1/2  0  0`
	`  0  -1  0`
	`  0  0  1 ]`

Apply the diagonal matrix to the appended system	`A3 = D3 * A2`
Extract the (updated) appended columns	`Ainv = A3( :, 4:6 )`
Check that the inverse was computed	`A * Ainv`

The result should be a 3×3 identity matrix.

Homework 8.2.4.3 Compute

- $\begin{pmatrix} 3 & 2 & 9 \\ -3 & -3 & -14 \\ 3 & 1 & 3 \end{pmatrix}^{-1} =$
- $\begin{pmatrix} 2 & -3 & 4 \\ 2 & -2 & 3 \\ 6 & -7 & 9 \end{pmatrix}^{-1} =$

Homework 8.2.4.4 Assume below that all matrices and vectors are partitioned "conformally" so that the operations make sense.

$$\left(\begin{array}{c|c|c} I & -u_{01} & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array}\right)\left(\begin{array}{c|c|c||c|c|c} D_{00} & a_{01} & A_{02} & B_{00} & 0 & 0 \\ \hline 0 & \alpha_{11} & a_{12}^T & b_{10}^T & 1 & 0 \\ \hline 0 & a_{21} & A_{22} & B_{20} & 0 & I \end{array}\right)$$

$$= \left(\begin{array}{c|c|c||c|c|c} D_{00} & a_{01}-\alpha_{11}u_{01} & A_{02}-u_{01}a_{12}^T & B_{00}-u_{01}b_{10}^T & -u_{01} & 0 \\ \hline 0 & \alpha_{11} & a_{12}^T & b_{10}^T & 1 & 0 \\ \hline 0 & a_{21}-\alpha_{11}l_{21} & A_{22}-l_{21}a_{12}^T & B_{20}-l_{21}b_{10}^T & -l_{21} & I \end{array}\right)$$

Always/Sometimes/Never

Homework 8.2.4.5 Assume below that all matrices and vectors are partitioned "conformally" so that the operations make sense. Choose

- $u_{01} := a_{01}/\alpha_{11}$; and
- $l_{21} := a_{21}/\alpha_{11}$.

Consider the following expression:

$$\left(\begin{array}{c|c|c} I & -u_{01} & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array}\right)\left(\begin{array}{c|c|c||c|c|c} D_{00} & a_{01} & A_{02} & B_{00} & 0 & 0 \\ \hline 0 & \alpha_{11} & a_{12}^T & b_{10}^T & 1 & 0 \\ \hline 0 & a_{21} & A_{22} & B_{20} & 0 & I \end{array}\right)$$

$$= \left(\begin{array}{c|c|c||c|c|c} D_{00} & 0 & A_{02}-u_{01}a_{12}^T & B_{00}-u_{01}b_{10}^T & -u_{01} & 0 \\ \hline 0 & \alpha_{11} & a_{12}^T & b_{10}^T & 1 & 0 \\ \hline 0 & 0 & A_{22}-l_{21}a_{12}^T & B_{20}-l_{21}b_{10}^T & -l_{21} & I \end{array}\right)$$

Always/Sometimes/Never

The above observations justify the two algorithms in Figures 8.5 and 8.6 for "Gauss-Jordan elimination" for inverting a matrix.

Algorithm: $[A,B] := \text{GJ\_INVERSE\_PART1}(A,B)$

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

where A_{TL} is 0×0, B_{TL} is 0×0

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

Repartition

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

where α_{11} is 1×1, β_{11} is 1×1

$$\begin{array}{c|c} & a_{01} := a_{01}/\alpha_{11} \\ \hline & \\ \hline & a_{21} := a_{21}/\alpha_{11} \end{array} \quad \begin{array}{c|c} A_{02} := A_{02} - a_{01}a_{12}^T & \\ & \\ \hline A_{22} := A_{22} - a_{21}a_{12}^T & \end{array} \qquad \begin{array}{c|c} B_{00} := B_{00} - a_{01}b_{10}^T & b_{01} := -a_{01} \\ \hline & \\ B_{20} := B_{20} - a_{21}b_{10}^T & b_{21} := -a_{21} \end{array}$$

(Note: a_{01} and a_{21} on the left need to be updated first.)

$a_{01} := 0$ (zero vector)

$a_{21} := 0$ (zero vector)

Continue with

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

endwhile

Figure 8.5: Algorithm that transforms diagonal matrix A to an identity matrix and updates an identity matrix stored in B accordingly.

Algorithm: $[A,B] := \text{GJ\_INVERSE\_PART2}(A,B)$

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

where A_{TL} is 0×0, B_{TL} is 0×0

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

Repartition

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

where α_{11} is 1×1, β_{11} is 1×1

$b_{10}^T := b_{10}^T / \alpha_{11}$

$\beta_{11} := \beta_{11} / \alpha_{11}$

$b_{12}^T := b_{12}^T / \alpha_{11}$

$\alpha_{11} := 1$

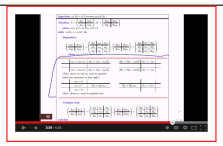
Continue with

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

endwhile

Figure 8.6: Algorithm that transforms diagonal matrix A to an identity matrix and updates an identity matrix stored in B accordingly.

8.2.5 Computing A^{-1} via Gauss-Jordan Elimination, Alternative



☛ View at edX

We now motivate a slight alternative to the Gauss Jordan method, which is easiest to program.

Homework 8.2.5.1

- Determine $\delta_{0,0}$, $\lambda_{1,0}$, $\lambda_{2,0}$ so that

$$\left(\begin{array}{c|cc} \delta_{0,0} & 0 & 0 \\ \hline \lambda_{1,0} & 1 & 0 \\ \lambda_{2,0} & 0 & 1 \end{array}\right)\left(\begin{array}{ccc||ccc} -1 & -4 & -2 & 1 & 0 & 0 \\ 2 & 6 & 2 & 0 & 1 & 0 \\ -1 & 0 & 3 & 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{ccc||ccc} 1 & 4 & 2 & -1 & 0 & 0 \\ 0 & -2 & -2 & 2 & 1 & 0 \\ 0 & 4 & 5 & -1 & 0 & 1 \end{array}\right)$$

- Determine $\upsilon_{0,1}$, $\delta_{1,1}$, and $\lambda_{2,1}$ so that

$$\left(\begin{array}{c|c|c} 1 & \upsilon_{0,1} & 0 \\ \hline 0 & \delta_{1,1} & 0 \\ \hline 0 & \lambda_{2,1} & 1 \end{array}\right)\left(\begin{array}{c|ccc||c|c|c} 1 & 4 & 2 & -1 & 0 & 0 \\ \hline 0 & -2 & -2 & 2 & 1 & 0 \\ \hline 0 & 4 & 5 & -1 & 0 & 1 \end{array}\right) = \left(\begin{array}{c|c|c||c|c|c} 1 & 0 & -2 & 3 & 2 & 0 \\ \hline 0 & 1 & 1 & -1 & -\frac{1}{2} & 0 \\ \hline 0 & 0 & 1 & 3 & 2 & 1 \end{array}\right)$$

- Determine $\upsilon_{0,2}$, $\upsilon_{0,2}$, and $\delta_{2,2}$ so that

$$\left(\begin{array}{cc|c} 1 & 0 & \upsilon_{0,2} \\ 0 & 1 & \upsilon_{1,2} \\ \hline 0 & 0 & \delta_{2,2} \end{array}\right)\left(\begin{array}{cc|c||cc|c} 1 & 0 & -2 & 3 & 2 & 0 \\ 0 & 1 & 1 & -1 & -\frac{1}{2} & 0 \\ \hline 0 & 0 & 1 & 3 & 2 & 1 \end{array}\right) = \left(\begin{array}{cc|c||cc|c} 1 & 0 & 0 & 9 & 6 & 2 \\ 0 & 1 & 0 & -4 & -\frac{5}{2} & -1 \\ \hline 0 & 0 & 1 & 3 & 2 & 1 \end{array}\right)$$

- Evaluate

$$\left(\begin{array}{ccc} -1 & -4 & -2 \\ 2 & 6 & 2 \\ -1 & 0 & 3 \end{array}\right)\left(\begin{array}{ccc} 9 & 6 & 2 \\ -4 & -\frac{5}{2} & -1 \\ 3 & 2 & 1 \end{array}\right) =$$

Homework 8.2.5.2 Assume below that all matrices and vectors are partitioned "conformally" so that the operations make sense.

$$\left(\begin{array}{c|c|c} I & -u_{01} & 0 \\ \hline 0 & \delta_{11} & 0 \\ \hline 0 & -l_{21} & I \end{array}\right)\left(\begin{array}{c|c|c||c|c|c} I & a_{01} & A_{02} & B_{00} & 0 & 0 \\ \hline 0 & \alpha_{11} & a_{12}^T & b_{10}^T & 1 & 0 \\ \hline 0 & a_{21} & A_{22} & B_{20} & 0 & I \end{array}\right)$$

$$= \left(\begin{array}{c|c|c||c|c|c} I & a_{01}-\alpha_{11}u_{01} & A_{02}-u_{01}a_{12}^T & B_{00}-u_{01}b_{10}^T & -u_{01} & 0 \\ \hline 0 & \delta_{11}\alpha_{11} & \delta_{11}a_{12}^T & \delta_{11}b_{10}^T & \delta_{11} & 0 \\ \hline 0 & a_{21}-\alpha_{11}l_{21} & A_{22}-l_{21}a_{12}^T & B_{20}-l_{21}b_{10}^T & -l_{21} & I \end{array}\right)$$

Always/Sometimes/Never

Homework 8.2.5.3 Assume below that all matrices and vectors are partitioned "conformally" so that the operations make sense. Choose

- $u_{01} := a_{01}/\alpha_{11}$;
- $l_{21} := a_{21}/\alpha_{11}$; and
- $\delta_{11} := 1/\alpha_{11}$.

$$\left(\begin{array}{c|c|c} I & -u_{01} & 0 \\ \hline 0 & \delta_{11} & 0 \\ \hline 0 & -l_{21} & I \end{array}\right) \left(\begin{array}{c|c|c||c|c|c} I & a_{01} & A_{02} & B_{00} & 0 & 0 \\ \hline 0 & \alpha_{11} & a_{12}^T & b_{10}^T & 1 & 0 \\ \hline 0 & a_{21} & A_{22} & B_{20} & 0 & I \end{array}\right)$$

$$= \left(\begin{array}{c|c|c||c|c|c} I & 0 & A_{02} - u_{01}a_{12}^T & B_{00} - u_{01}b_{10}^T & -u_{01} & 0 \\ \hline 0 & 1 & a_{12}^T/\alpha_{11} & b_{10}^T/\alpha_{11} & 1/\alpha_{11} & 0 \\ \hline 0 & 0 & A_{22} - l_{21}a_{12}^T & B_{20} - l_{21}b_{10}^T & -l_{21} & I \end{array}\right)$$

Always/Sometimes/Never

The last homework motivates the algorithm in Figure 8.7

Homework 8.2.5.4 Implement the algorithm in Figure 8.7 yielding the function

- `[ A_out ] = GJ_Inverse_alt_unb( A, B )`. Assume that it is called as

```
Ainv = GJ_Inverse_alt_unb( A, B )
```

  Matrices A and B must be square and of the same size.

Check that it computes correctly with the script

- `test_GJ_Inverse_alt_unb.m`.

Homework 8.2.5.5 If you are very careful, you can overwrite matrix A with its inverse without requiring the matrix B.
Modify the algorithm in Figure 8.7 so that it overwrites A with its inverse without the use of matrix B yielding the function

- `[ A_out ] = GJ_Inverse_inplace_unb( A )`.

Check that it computes correctly with the script

- `test_GJ_Inverse_inplace_unb.m`.

Algorithm: $[B] := \text{GJ\_INVERSE\_ALT}(A, B)$

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

where A_{TL} is 0×0, B_{TL} is 0×0

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

Repartition

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

where α_{11} is 1×1, β_{11} is 1×1

$$\begin{array}{|c|c|} \hline & a_{01} := a_{01}/\alpha_{11} \\ \hline & \\ \hline & a_{21} := a_{21}/\alpha_{11} \\ \hline \end{array} \quad \begin{array}{|c|} \hline A_{02} := A_{02} - a_{01}a_{12}^T \\ \hline \\ \hline A_{22} := A_{22} - a_{21}a_{12}^T \\ \hline \end{array} \quad \begin{array}{|c|c|} \hline B_{00} := B_{00} - a_{01}b_{10}^T & b_{01} := -a_{01} \\ \hline & \\ \hline B_{20} := B_{20} - a_{21}b_{10}^T & b_{21} := -a_{21} \\ \hline \end{array}$$

(Note: above a_{01} and a_{21} must be updated before the operations to their right.)

$$\begin{array}{|c|c|} \hline & a_{01} := 0 \\ \hline & \alpha_{11} := 1 \\ \hline & a_{21} := 0 \\ \hline \end{array} \quad \begin{array}{|c|} \hline \\ \hline a_{12}^T := a_{12}^T/\alpha_{11} \\ \hline \\ \hline \end{array} \quad \begin{array}{|c|c|} \hline & \\ \hline b_{10}^T := b_{10}^T/\alpha_{11} & \beta_{11} = 1/\alpha_{11} \\ \hline & \\ \hline \end{array}$$

(Note: above α_{11} must be updated last.)

Continue with

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

endwhile

Figure 8.7: Algorithm that simultaneously transforms matrix A to an identity and matrix B from the identity to A^{-1}.

8.2.6 Pivoting

☛ View at edX

Adding pivoting to any of the discussed Gauss-Jordan methods is straight forward. It is a matter of recognizing that if a zero is found on the diagonal during the process at a point where a divide by zero will happen, one will need to swap the current row with another row below it to overcome this. If such a row cannot be found, then the matrix does not have an inverse.

We do not further discuss this in this course.

8.2.7 Cost of Matrix Inversion

☛ View at edX

Let us now discuss the cost of matrix inversion via various methods. In our discussion, we will ignore pivoting. In other words, we will assume that no zero pivot is encountered. We wil start with an $n \times n$ matrix A.

A very naive approach

Here is a very naive approach. Let X be the matrix in which we will compute the inverse. We have argued several times that $AX = I$ means that

$$A \left(\begin{array}{c|c|c|c} x_0 & x_1 & \cdots & x_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} e_0 & e_1 & \cdots & e_{n-1} \end{array} \right)$$

so that $Ax_j = e_j$. So, *for each column* x_j, we can perform the operations

- Compute the LU factorization of A so that $A = LU$. We argued in Week 6 that the cost of this is approximately $\frac{2}{3}n^3$ flops.
- Solve $Lz = e_j$. This is a lower (unit) triangular solve with cost of approximately n^2 flops.
- Solve $Ux_j = z$. This is an upper triangular solve with cost of approximately n^2 flops.

So, for *each* column of X the cost is approximately $2n^3 + n^3 + n^3 = 2n^3 + 2n^2$. There are n columns of X to be computed for a total cost of approximately

$$n(2n^3 + 2n^2) = 2n^4 + 2n^3 \text{ flops.}$$

To put this in perspective: A relatively small problem to be solved on a current supercomputer involves a $100,000 \times 100,000$ matrix. The fastest current computer can perform approximately 55,000 Teraflops, meaning 55×10^{15} floating point operations per second. On this machine, inverting such a matrix would require approximately an hour of compute time.

(Note: such a supercomputer would not attain the stated peak performance. But let's ignore that in our discussions.)

A less naive approach

The problem with the above approach is that A is redundantly factored into L and U for every column of X. Clearly, we only need to do that once. Thus, a less naive approach is given by

- Compute the LU factorization of A so that $A = LU$ at a cost of approximately $\frac{2}{3}n^3$ flops.
- For each column x_j
  - Solve $Lz = e_j$. This is a lower (unit) triangular solve with cost of approximately n^2 flops.
  - Solve $Ux_j = z$. This is an upper triangular solve with cost of approximately n^2 flops.

  There are n columns of X to be computed for a total cost of approximately

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

Thus, the total cost is now approximately

$$\frac{2}{3}n^3 + 2n^3 = \frac{8}{3}n^3 \text{ flops.}$$

Returning to our relatively small problem of inverting a $100,000 \times 100,000$ matrix on the fastest current computer that can perform approximately 55,000 Teraflops, inverting such a matrix with this alternative approach would require approximately 0.05 seconds. Clearly an improvement.

The cost of the discussed Gauss-Jordan matrix inversion

Now let's consider the Gauss-Jordan matrix inversion algorithm that we developed in the last unit:

Algorithm: $[B] := \text{GJ\_INVERSE\_ALT}(A, B)$

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

where A_{TL} is 0×0, B_{TL} is 0×0

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

Repartition

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

where α_{11} is 1×1, β_{11} is 1×1

$$\begin{array}{c|c|c} & a_{01} := a_{01}/\alpha_{11} & A_{02} := A_{02} - a_{01}a_{12}^T \\ \hline & & \\ \hline & a_{21} := a_{21}/\alpha_{11} & A_{22} := A_{22} - a_{21}a_{12}^T \end{array} \qquad \begin{array}{c|c|c} & B_{00} := B_{00} - a_{01}b_{10}^T & b_{01} := -a_{01} \\ \hline & & \\ \hline & B_{20} := B_{20} - a_{21}b_{10}^T & b_{21} := -a_{21} \end{array}$$

(Note: above a_{01} and a_{21} must be updated before the operations to their right.)

$$\begin{array}{c|c|c} & a_{01} := 0 & \\ \hline & \alpha_{11} := 1 & a_{12}^T := a_{12}^T/\alpha_{11} \\ \hline & a_{21} := 0 & \end{array} \qquad \begin{array}{c|c|c} & & \\ \hline b_{10}^T := b_{10}^T/\alpha_{11} & \beta_{11} = 1/\alpha_{11} & \\ \hline & & \end{array}$$

(Note: above α_{11} must be updated last.)

Continue with

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

endwhile

During the kth iteration, A_{TL} and B_{TL} are $k \times k$ (starting with $k = 0$). After repartitioning, the sizes of the different subma-

trices are

$$\begin{array}{r} \\ k\{ \\ 1\{ \\ n-k-1\{ \end{array} \begin{array}{c} \overbrace{\phantom{A_{00}}}^{k} \overbrace{\phantom{a_{01}}}^{1} \overbrace{\phantom{A_{02}}}^{n-k-1} \\ \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{02} \end{array}\right) \end{array}$$

The following operations are performed (we ignore the other operations since they are clearly "cheap" relative to the ones we do count here):

- $A_{02} := A_{02} - a_{01}a_{12}^T$. This is a rank-1 update. The cost is $2k \times (n-k-1)$ flops.
- $A_{22} := A_{22} - a_{21}a_{12}^T$. This is a rank-1 update. The cost is $2(n-k-1) \times (n-k-1)$ flops.
- $B_{00} := B_{00} - a_{01}b_{10}^T$. This is a rank-1 update. The cost is $2k \times k$ flops.
- $B_{02} := B_{02} - a_{21}b_{12}^T$. This is a rank-1 update. The cost is $2(n-k-1) \times k$ flops.

For a total of, approximately,

$$\begin{array}{rcl} \underbrace{2k(n-k-1)+2(n-k-1)(n-k-1)}_{2(n-1)(n-k-1)} + \underbrace{2k^2+2(n-k-1)k}_{2(n-1)k} & = & 2(n-1)(n-k-1)+2(n-1)k \\ & = & 2(n-1)^2 \text{ flops.} \end{array}$$

Now, we do this for n iterations, so the total cost of the Gauss-Jordan inversion algorithms is, approximately,

$$n(2(n-1)^2) \approx 2n^3 \text{ flops.}$$

Barring any special properties of matrix A, or high-trapeze heroics, this turns out to be the cost of matrix inversion. Notice that this cost is less than the cost of the (less) naive algorithm given before.

A simpler analysis is as follows: The bulk of the computation in each iteration is in the updates

$$\left(\begin{array}{c|c} B_{00} := B_{00} - a_{01}b_{10}^T & A_{02} := A_{02} - a_{01}a_{12}^T \\ \hline B_{20} := B_{20} - a_{21}b_{10}^T & A_{22} := A_{22} - a_{21}a_{12}^T \end{array}\right)$$

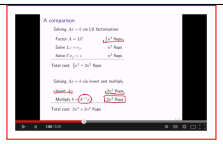
Here we try to depict that the elements being updated occupy almost an entire $n \times n$ matrix. Since there are rank-1 updates being performed, this means that essentially every element in this matrix is being updated with one multiply and one add. Thus, in this iteration, approximately $2n^2$ flops are being performed. The total for n iterations is then, approximately, $2n^3$ flops.

Returning one last time to our relatively small problem of inverting a $100,000 \times 100,000$ matrix on the fastest current computer that can perform approximately 55,000 Teraflops, inverting such a matrix with this alternative approach is further reduced from approximately 0.05 seconds to approximately 0.036 seconds. Not as dramatic a reduction, but still worthwhile.

Interestingly, the cost of matrix inversion is approximately the same as the cost of matrix-matrix multiplication.

8.3 (Almost) Never, Ever Invert a Matrix

8.3.1 Solving $Ax = b$



☛ View at edX

Solving $Ax = b$ via LU Factorization

Homework 8.3.1.1 Let $A \in \mathbb{R}^{n \times n}$ and $x, b \in \mathbb{R}^n$. What is the cost of solving $Ax = b$ via LU factorization (assuming there is nothing special about A)? You may ignore the need for pivoting.

Solving $Ax = b$ by Computing A^{-1}

Homework 8.3.1.2 Let $A \in \mathbb{R}^{n \times n}$ and $x, b \in \mathbb{R}^n$. What is the cost of solving $Ax = b$ if you first invert matrix A and than compute $x = A^{-1}b$? (Assume there is nothing special about A and ignore the need for pivoting.)

Just Don't Do It!

The bottom line is: LU factorization followed by two triangular solves is cheaper!

Now, some people would say "What if we have many systems $Ax = b$ where A is the same, but b differs? Then we can just invert A once and for each of the bs multiply $x = A^{-1}b$."

Homework 8.3.1.3 What is wrong with the above argument?

There are other arguments why computing A^{-1} is a bad idea that have to do with floating point arithmetic and the roundoff error that comes with it. This is a subject called "numerical stability", which goes beyond the scope of this course.

So.... You should be very suspicious if someone talks about computing the inverse of a matrix. There are very, very few applications where one legitimately needs the inverse of a matrix.
However, realize that often people use the term "inverting a matrix" interchangeably with "solving $Ax = b$", where they don't mean to imply that they explicitly invert the matrix. So, be careful before you start arguing with such a person! They may simply be using awkward terminology.

Of course, the above remarks are for general matrices. For small matrices and/or matrices with special structure, inversion may be a reasonable option.

8.3.2 But...

No Video for this Unit

Inverse of a general matrix

Ironically, one of the instructors of this course has written a paper about high-performance inversion of a matrix, which was then published by a top journal:

> Xiaobai Sun, Enrique S. Quintana, Gregorio Quintana, and Robert van de Geijn.
> A Note on Parallel Matrix Inversion.
> *SIAM Journal on Scientific Computing*, Vol. 22, No. 5, pp. 1762–1771.
> Available from http://www.cs.utexas.edu/users/flame/pubs/SIAMMatrixInversion.pdf.
> (This was the first journal paper in which the FLAME notation was introduced.)

The algorithm developed for that paper is a blocked algorithm that incorporates pivoting that is a direct extension of the algorithm we introduce in Unit 8.2.5. It was developed for use in a specific algorithm that required the explicit inverse of a general matrix.

Inverse of a symmetric positive definite matrix

Inversion of a special kind of symmetric matrix called a symmetric positive definite (SPD) matrix is sometimes needed in statistics applications. The inverse of the so-called covariance matrix (which is typically a SPD matrix) is called the precision matrix, which for some applications is useful to compute. We talk about how to compute a factorization of such matrices in this week's enrichment.

If you go to wikipedia and seach for "precision matrix" you will end up on this page:

Precision (statistics)

that will give you more information.

We have a paper on how to compute the inverse of a SPD matrix:

Paolo Bientinesi, Brian Gunter, Robert A. van de Geijn.
Families of algorithms related to the inversion of a Symmetric Positive Definite matrix.
ACM Transactions on Mathematical Software (TOMS), 2008
Available from http://www.cs.utexas.edu/~flame/web/FLAMEPublications.html.

Welcome to the frontier!

Try reading the papers above (as an enrichment)! You will find the notation very familiar.

8.4 (Very Important) Enrichment

8.4.1 Symmetric Positive Definite Matrices

Symmetric positive definite (SPD) matrices are an important class of matrices that occur naturally as part of applications. We will see SPD matrices come up later in this course, when we discuss how to solve overdetermined systems of equations:

$$Bx = y \quad \text{where} \quad B \in \mathbb{R}^{m\times n} \text{ and } m > n.$$

In other words, when there are more equations than there are unknowns in our linear system of equations. When B has "linearly independent columns," a term with which you will become very familiar later in the course, the best solution to $Bx = y$ satisfies $B^T Bx = B^T y$. If we set $A = B^T B$ and $b = B^T y$, then we need to solve $Ax = b$, and now A is square and nonsingular (which we will prove later in the course). Now, we could solve $Ax = b$ via any of the methods we have discussed so far. However, these methods ignore the fact that A is symmetric. So, the question becomes how to take advantage of symmetry.

Definition 8.1 *Let $A \in \mathbb{R}^{n\times n}$. Matrix A is said to be symmetric positive definite (SPD) if*

- *A is symmetric; and*
- *$x^T Ax > 0$ for all nonzero vectors $x \in \mathbb{R}^n$.*

A nonsymmetric matrix can also be positive definite and there are the notions of a matrix being negative definite or indefinite. We won't concern ourselves with these in this course.

Here is a way to relate what a positive definite matrix is to something you may have seen before. Consider the quadratic polynomial

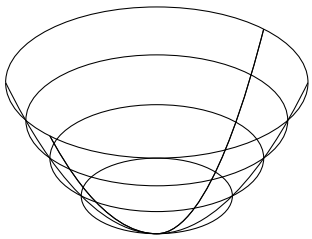
$$p(\chi) = \alpha\chi^2 + \beta\chi + \gamma = \chi\alpha\chi + \beta\chi + \gamma.$$

The graph of this function is a parabola that is "concaved up" if $\alpha > 0$. In that case, it attains a minimum at a unique value χ.

Now consider the vector function $f : \mathbb{R}^n \to \mathbb{R}$ given by

$$f(x) = x^T Ax + b^T x + \gamma$$

where $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, and $\gamma \in \mathbb{R}$ are all given. If A is a SPD matrix, then this equation is minimized for a unique vector x. If $n = 2$, plotting this function when A is SPD yields a paraboloid that is concaved up:



8.4.2 Solving $Ax = b$ when A is Symmetric Positive Definite

We are going to concern ourselves with how to solve $Ax = b$ when A is SPD. What we will notice is that by taking advantage of symmetry, we can factor A akin to how we computed the LU factorization, but at roughly half the computational cost. This new factorization is known as the Cholesky factorization.

Cholesky factorization theorem

Theorem 8.2 *Let $A \in \mathbb{R}^{n\times n}$ be a symmetric positive definite matrix. Then there exists a lower triangular matrix $L \in \mathbb{R}^{n\times n}$ such that $A = LL^T$. If the diagonal elements of L are chosen to be positive, this factorization is unique.*

We will not prove this theorem.

Unblocked Cholesky factorization

We are going to closely mimic the derivation of the LU factorization algorithm from Unit 6.3.1.

Partition

$$A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array}\right), \quad \text{and} \quad L \rightarrow \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right).$$

Here we use $\star$ to indicate that we are not concerned with that part of the matrix because A is symmetric and hence we should be able to just work with the lower triangular part of it.

We want L to satisfy $A = LL^T$. Hence

$$\begin{aligned}
\overbrace{\left(\begin{array}{c|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array}\right)}^{A} &= \overbrace{\left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right)}^{L} \overbrace{\left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right)^T}^{L^T} \\
&= \overbrace{\left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right)}^{L} \overbrace{\left(\begin{array}{c|c} \lambda_{11} & l_{21}^T \\ \hline 0 & L_{22}^T \end{array}\right)}^{L^T} \\
&= \overbrace{\left(\begin{array}{c|c} \lambda_{11}^2 + 0\times 0 & \star \\ \hline l_{21}\lambda_{11} + L_{22}\times 0 & l_{21}l_{21}^T + L_{22}L_{22}^T \end{array}\right)}^{LL^T}. \\
&= \overbrace{\left(\begin{array}{c|c} \lambda_{11}^2 & \star \\ \hline l_{21}\lambda_{11} & l_{21}l_{21}^T + L_{22}L_{22}^T \end{array}\right)}^{LL^T}.
\end{aligned}$$

where, again, the $\star$ refers to part of the matrix in which we are not concerned because of symmetry.

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

$$\left(\begin{array}{c|c} \alpha_{11} = \lambda_{11}^2 & \star \\ \hline a_{21} = l_{21}\lambda_{11} & A_{22} = l_{21}l_{21}^T + L_{22}L_{22}^T \end{array}\right)$$

or, rearranging,

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

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

Algorithm: $[A] := \text{CHOL\_UNB\_VAR3}(A)$

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

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

$\alpha_{11} := \sqrt{\alpha_{11}}$

$a_{21} := a_{21}/\alpha_{11}$

$A_{22} := A_{22} - a_{21}a_{21}^T$

(updating only the lower triangular part)

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 8.8: Algorithm for overwriting the lower triangular part of A with its Cholesky factor.

- Partition
$$A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array}\right).$$
- $\alpha_{11} = \sqrt{\alpha_{11}} \quad (= \lambda_{11})$.
- Update $a_{21} = a_{21}/\alpha_{11} \quad (= l_{21})$.
- Update $A_{22} = A_{22} - a_{21}a_{12}^T (= A_{22} - l_{21}l_{21}^T)$
Here we use a "symmetric rank-1 update" since A_{22} and $l_{21}l_{21}^T$ are both symmetric and hence only the lower triangular part needs to be updated. This is where we save flops.
- Overwrite A_{22} with L_{22} by repeating with $A = A_{22}$.

This overwrites the lower triangular part of A with L.

The above can be summarized in Figure 8.8. The suspicious reader will notice that $\alpha_{11} := \sqrt{\alpha_{11}}$ is only legal if $\alpha_{11} > 0$ and $a_{21} := a_{21}/\alpha_{11}$ is only legal if $\alpha_{11} \neq 0$. It turns out that *if* A is SPD, then

- $\alpha_{11} > 0$ in the first iteration and hence $\alpha_{11} := \sqrt{\alpha_{11}}$ and $a_{21} := a_{21}/\alpha_{11}$ are legal; and
- $A_{22} := A_{22} - a_{21}a_{21}^T$ is again a SPD matrix.

The proof of these facts goes beyond the scope of this course. The net result is that the algorithm will compute L if it is executed starting with a matrix A that is SPD. It is useful to compare and contrast the derivations of the unblocked LU factorization and the unblocked Cholesky factorization, in Figure 8.9.

LU factorization	Cholesky factorization
$A \rightarrow \left(\begin{array}{c\|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right), L \rightarrow \left(\begin{array}{c\|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array}\right), U \rightarrow \left(\begin{array}{c\|c} \upsilon_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array}\right)$	$A \rightarrow \left(\begin{array}{c\|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array}\right), L \rightarrow \left(\begin{array}{c\|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right).$
$\left(\begin{array}{c\|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right) = \underbrace{\left(\begin{array}{c\|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array}\right)\left(\begin{array}{c\|c} \upsilon_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array}\right)}_{\left(\begin{array}{c\|c} \upsilon_{11} & u_{12}^T \\ \hline l_{21}\upsilon_{11} & l_{21}u_{12}^T + L_{22}U_{22} \end{array}\right)}$	$\left(\begin{array}{c\|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array}\right) = \underbrace{\left(\begin{array}{c\|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right)\left(\begin{array}{c\|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right)^T}_{\left(\begin{array}{c\|c} \lambda_{11}^2 & \star \\ \hline l_{21}\lambda_{11} & l_{21}l_{12}^T + L_{22}L_{22}^T \end{array}\right)}.$
$\begin{array}{c\|c} \alpha_{11} = \upsilon_{11} & a_{12}^T = u_{12}^T \\ \hline a_{21} = l_{21}\upsilon_{11} & A_{22} = l_{21}u_{12}^T + L_{22}U_{22} \end{array}$	$\begin{array}{c\|c} \alpha_{11} = \lambda_{11}^2 & \star \\ \hline a_{21} = l_{21}\lambda_{11} & A_{22} = l_{21}l_{12}^T + L_{22}L_{22}^2 \end{array}$
$\alpha_{11} := \alpha_{11}$ $a_{12}^T := a_{12}^T$ $a_{21} := a_{21}/\alpha_{11}$ $A_{22} := A_{22} - a_{21}a_{12}^T$	$\alpha_{11} := \sqrt{\alpha_{11}}$ $a_{21} := a_{21}/\alpha_{11}$ $A_{22} := A_{22} - a_{21}a_{12}^T$ (update only lower triangular part)

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
	Algorithm: $[A] := \text{CHOL\_UNB\_VAR3}(A)$
	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
	$\left(\begin{array}{c\|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$
	$\alpha_{11} := \sqrt{\alpha_{11}}$
	$a_{21} := a_{21}/\alpha_{11}$
	$A_{22} := A_{22} - a_{21}a_{21}^T$
	(updating only the lower triangular part)
	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 8.9: Side-by-side derivations of the unblocked LU factorization and Cholesky factorization algorithms.

Once one has computed the Cholesky factorization of A, one can solve $Ax = b$ by substituting

$$\overbrace{LL^T}^{A} x = b$$

and first solving $Lz = b$ after which solving $L^T x = z$ computes the desired solution x. Of course, as you learned in Weeks 3 and 4, you need not transpose the matrix!

Blocked (and other) algorithms

If you are interested in blocked algorithms for computing the Cholesky factorization, you may want to look at some notes we wrote:

Robert van de Geijn.
Notes on Cholesky Factorization
http://www.cs.utexas.edu/users/flame/Notes/NotesOnCholReal.pdf

These have since become part of the notes Robert wrote for his graduate class on Numerical Linear Algebra:

Robert van de Geijn.
Linear Algebra: Foundations to Frontiers - Notes on Numerical Linear Algebra, Chapter 12.

Systematic derivation of Cholesky factorization algorithms

The above video was created when Robert was asked to give an online lecture for a class at Carnegie Mellon University. It shows how algorithms can be systematically derived (as we discussed already in Week 2) using goal-oriented programming. It includes a demonstration by Prof. Paolo Bientinesi (RWTH Aachen University) of a tool that performs the derivation automatically. It is when a process is systematic to the point where it can be automated that a computer scientist is at his/her happiest!

More materials

You will find materials related to the implementation of this operations, including a video that demonstrates this, at

http://www.cs.utexas.edu/users/flame/Movies.html#Chol

Unfortunately, some of the links don't work (we had a massive failure of the wiki that hosted the material).

8.4.3 Other Factorizations

We have now encountered the LU factorization,

$$A = LU,$$

the LU factorization with row pivoting,

$$PA = LU,$$

and the Cholesky factorization,

$$A = LL^T.$$

Later in this course you will be introduced to the QR factorization,

$$A = QR,$$

where Q has the special property that $Q^T Q = I$ and R is an upper triangular matrix.

When a matrix is *indefinite symmetric*, there is a factorization called the LDL^T (pronounce as L D L transpose) factorization,

$$A = LDL^T,$$

where L is unit lower triangular and D is diagonal. You may want to see if you can modify the derivation of the Cholesky factorization to yield an algorithm for the LDL^T factorization.

8.4.4 Welcome to the Frontier

Building on the material to which you have been exposed so far in this course, you should now be able to fully understand significant parts of many of our publications. (When we write our papers, we try to target a broad audience.) Many of these papers can be found at

http://www.cs.utexas.edu/~flame/web/publications.

If not there, then Google!

Here is a small sampling:

- The paper I consider our most significant contribution to science to date:

  Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique S. Quintana-Orti, Robert A. van de Geijn.
  The science of deriving dense linear algebra algorithms.
  ACM Transactions on Mathematical Software (TOMS), 2005.

- The book that explains the material in that paper at a more leisurely pace:

  Robert A. van de Geijn and Enrique S. Quintana-Orti.
  The Science of Programming Matrix Computations.
  www.lulu.com, 2008.

- The journal paper that first introduced the FLAME notation:

  Xiaobai Sun, Enrique S. Quintana, Gregorio Quintana, and Robert van de Geijn.
  A Note on Parallel Matrix Inversion.
  SIAM Journal on Scientific Computing, Vol. 22, No. 5, pp. 1762–1771.
  http://www.cs.utexas.edu/~flame/pubs/SIAMMatrixInversion.pdf.

- The paper that discusses many operations related to the inversion of a SPD matrix:

  Paolo Bientinesi, Brian Gunter, Robert A. van de Geijn.
  Families of algorithms related to the inversion of a Symmetric Positive Definite matrix.
  ACM Transactions on Mathematical Software (TOMS), 2008.

- The paper that introduced the FLAME APIs:

  Paolo Bientinesi, Enrique S. Quintana-Orti, Robert A. van de Geijn.
  Representing linear algebra algorithms in code: the FLAME application program interfaces.
  ACM Transactions on Mathematical Software (TOMS), 2005.

- Our papers on high-performance implementation of BLAS libraries:

  Kazushige Goto, Robert A. van de Geijn.
  Anatomy of high-performance matrix multiplication.
  ACM Transactions on Mathematical Software (TOMS), 2008.

  Kazushige Goto, Robert van de Geijn.
  High-performance implementation of the level-3 BLAS.
  ACM Transactions on Mathematical Software (TOMS), 2008

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

- A classic paper on how to parallelize matrix-matrix multiplication:

  Robert A van de Geijn, Jerrell Watts.
  SUMMA: Scalable universal matrix multiplication algorithm.
  Concurrency Practice and Experience, 1997.

For that paper, and others on parallel computing on large distributed memory computers, it helps to read up on collective communication on massively parallel architectures:

Ernie Chan, Marcel Heimlich, Avi Purkayastha, Robert van de Geijn.
Collective communication: theory, practice, and experience.
Concurrency and Computation: Practice & Experience , Volume 19 Issue 1, September 2007

- A paper that gives you a peek at how to parallelize for massively parallel architectures:

Jack Poulson, Bryan Marker, Robert A. van de Geijn, Jeff R. Hammond, Nichols A. Romero.
Elemental: A New Framework for Distributed Memory Dense Matrix Computations.
ACM Transactions on Mathematical Software (TOMS), 2013.

Obviously, there are many people who work in the area of dense linear algebra operations and algorithms. We cite our papers here because you will find the notation used in those papers to be consistent with the slicing and dicing notation that you have been taught in this course. Much of the above cite work builds on important results of others. We stand on the shoulders of giants.

8.5 Wrap Up

8.5.1 Homework

8.5.2 Summary

Equivalent conditions

The following statements are equivalent statements about $A \in \mathbb{R}^{n \times n}$:

- A is nonsingular.
- A is invertible.
- A^{-1} exists.
- $AA^{-1} = A^{-1}A = I$.
- A represents a linear transformation that is a bijection.
- $Ax = b$ has a unique solution for all $b \in \mathbb{R}^n$.
- $Ax = 0$ implies that $x = 0$.
- $Ax = e_j$ has a solution for all $j \in \{0, \ldots, n-1\}$.
- The determinant of A is nonzero: $\det(A) \neq 0$.
- LU with partial pivoting does not break down.

Algorithm for inverting a matrix

See Figure 8.10.

Cost of inverting a matrix

Via Gauss-Jordan, taking advantage of zeroes in the appended identity matrix, requires approximately

$$2n^3 \text{ floating point operations.}$$

Algorithm: $[A] := \text{GJ\_INVERSE\_INPLACE}(A)$

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

	$a_{01} := a_{01}/\alpha_{11}$	$A_{02} := A_{02} - a_{01}a_{12}^T$	$A_{00} := A_{00} - a_{01}a_{10}^T$	$a_{01} := -a_{01}$	
	$a_{21} := a_{21}/\alpha_{11}$	$A_{22} := A_{22} - a_{21}a_{12}^T$	$A_{20} := A_{20} - a_{21}a_{10}^T$	$a_{21} := -a_{21}$	

(Note: above a_{01} and a_{21} must be updated before the operations to their right.)

		$a_{12}^T := a_{12}^T/\alpha_{11}$	$a_{10}^T := a_{10}^T/\alpha_{11}$	$\alpha_{11} = 1/\alpha_{11}$	

(Note: above α_{11} must be updated last.)

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 8.10: Algorithm for inplace inversion of a matrix (when pivoting is not needed).

(Almost) never, ever invert a matrix

Solving $Ax = b$ should be accomplished by first computing its LU factorization (possibly with partial pivoting) and then solving with the triangular matrices.