Linear Systems I (part 2)

Steve Marschner

Cornell CS 322

Outline

Solving linear systems

We all know how to approach these systems by hand, taking advantage of certain transformations we can apply without changing the answer:

- Multiply both sides of an equation by a number
- Add or subtract two equations

In a system expressed as a matrix, these operations correspond to scaling and combining rows of the matrix (as long as we do the same thing to the RHS).

Gauss-Jordan elimination

A procedure you may have learned for systematically eliminating variables from a linear system.

- Subtract first equation from all others, scaled appropriately to cancel the first variable.
- Subtract second equation from all others, again scaled appropriately
  - does not disturb the canceled first variable
- Continue with all other rows
- Answer can be read directly from reduced equations

Example from Moler p. 54.

Gauss-Jordan example

$$10x_1 - 7x_2 = 7$$

$$-3x_1 + 2x_2 + 6x_3 = 4$$

$$5x_1 - x_2 + 5x_3 = 6$$

To eliminate x_1, add $\frac{3}{10}$ of row 1 to row 2 and add $-\frac{1}{2}$ of row 1 to row 2.

Gauss-Jordan example

$$10x_1 - 7x_2 = 7$$

$$-0.1x_2 + 6x_3 = 6.1$$

$$2.5x_2 + 5x_3 = 2.5$$

second round: multipliers are

$(-7/0.1)$ and $(2.5/0.1)$

Gauss-Jordan example

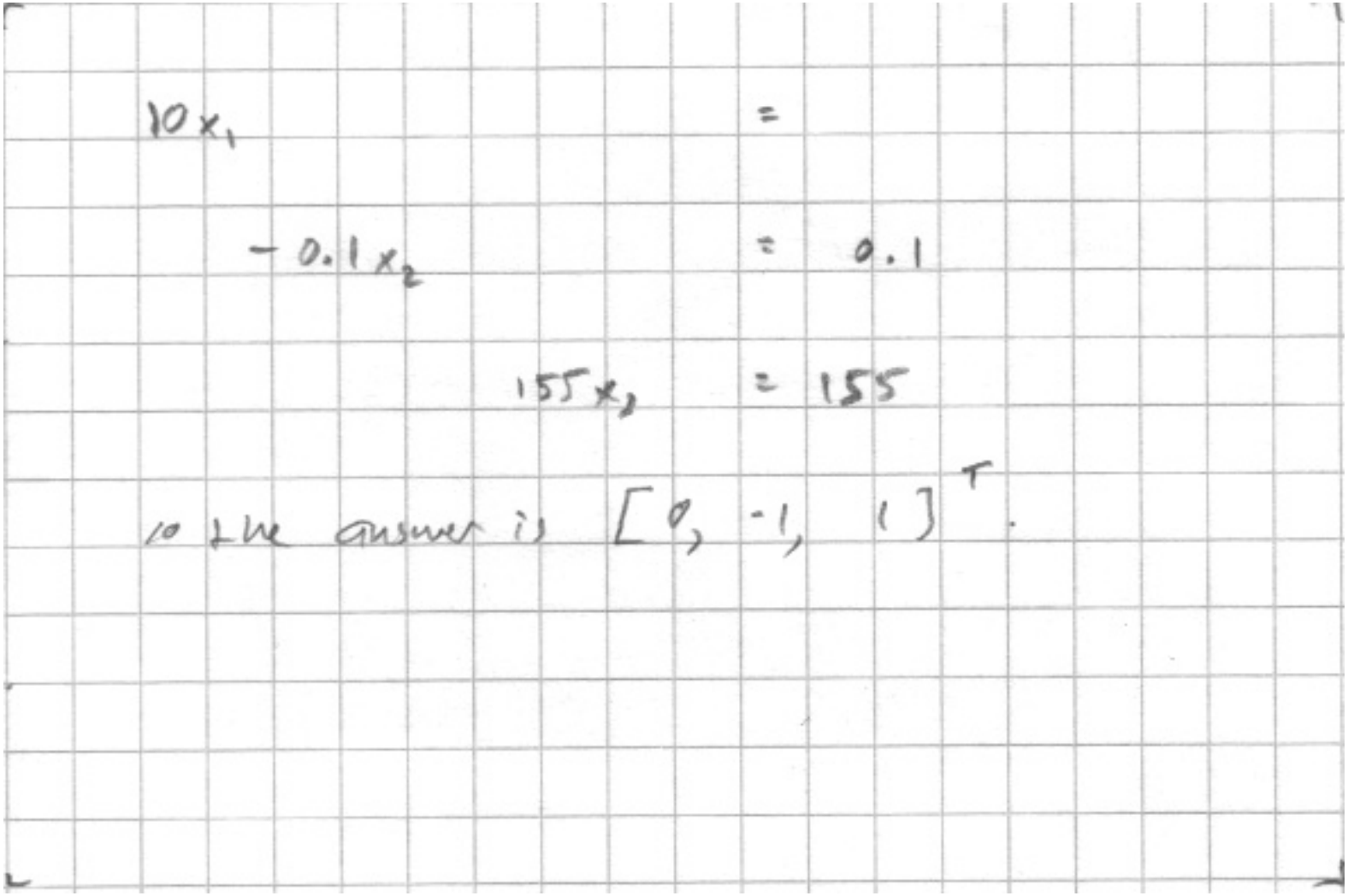
$$10x_1 - 420x_3 = -420$$

$$-0.1x_2 + 6x_3 = 6.1$$

$$155x_3 = 155$$

last round: multipliers are $6/155$ and $-420/155$

Gauss-Jordan example



Gauss-Jordan in matrix operations

In matrix terms, why are these scaling and combination operations allowed?

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

$$\mathbf{A}\mathbf{x} - \mathbf{b} = 0$$

Then for any "reasonable" matrix $\mathbf{M}$:

$$\mathbf{M}(\mathbf{A}\mathbf{x} - \mathbf{b}) = 0$$

holds if and only if $\mathbf{M}\mathbf{x} - \mathbf{b} = 0$. So the system with $\mathbf{MA}$ and $\mathbf{Mb}$ is equivalent to the system with $\mathbf{A}$ and $\mathbf{b}$.

What is $\mathbf{M}$ for a Gauss-Jordan step?

Row operations as matrix multiplication

For the first step of a 3×3 system:

$$\begin{bmatrix} \frac{1}{a_{11}} & & \\ -\frac{a_{21}}{a_{11}} & 1 & \\ -\frac{a_{31}}{a_{11}} & & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & a_{12}^{+} & a_{13}^{+} \\ & a_{22}^{+} & a_{23}^{+} \\ & a_{32}^{+} & a_{33}^{+} \end{bmatrix}$$

The row operations amount to multiplying by a specially designed matrix to zero out $\mathbf{A}(2:3,1)$.

Row operations as matrix multiplication

The general case for step j in an $n \times n$ system:

$$\mathbf{M}_j = \begin{bmatrix} 1 & & -a_{1j}/a_{jj} & & \\ & \ddots & \vdots & & \\ & & 1/a_{jj} & & \\ & & \vdots & \ddots & \\ & & -a_{nj}/a_{jj} & & 1 \end{bmatrix}$$

In Matlab we could write (though we wouldn't form $\mathbf{M}_j$ in practice):

```
M = eye(n);
M(:,j) = -A(:,j)/A(j,j);
M(j,j) = 1/A(j,j);
```

Gauss-Jordan in matrix terms

Applying the row operations to the matrix and RHS amounts to hitting a wide matrix containing A and b with a sequence of matrices on the left:

$$\mathbf{M}_n \cdots \mathbf{M}_2 \mathbf{M}_1 \begin{bmatrix} \mathbf{A} & \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{x} \end{bmatrix}$$
$$\mathbf{M} \begin{bmatrix} \mathbf{A} & \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{x} \end{bmatrix}$$

(Recall that multiplying by a matrix on the left transforms the columns, independently.) The sequence of $\mathbf{M}_j$s (row operations) reduces $\mathbf{A}$ to $\mathbf{I}$ and $\mathbf{b}$ to $\mathbf{x}$.

Gauss-Jordan in matrix terms

So this product matrix $\mathbf{M}$ is something that when multiplied by $\mathbf{A}$ gives the identity. There's a name for that!

$$\mathbf{M} = \mathbf{M}_n \cdots \mathbf{M}_2 \mathbf{M}_1 = \mathbf{A}^{-1}$$

The product of all the Gauss-Jordan matrices is the inverse of $\mathbf{A}$. This is another way to see why $\mathbf{Mb} = \mathbf{x}$.

In an implementation we wouldn't really form the $\mathbf{M}_j$s and multiply them. Instead start with $\mathbf{I}$ and apply the same row operations we apply to $\mathbf{A}$.

$$\mathbf{Y}_0 = \mathbf{I} \quad ; \quad \mathbf{Y}_k = \mathbf{M}_k \mathbf{Y}_{k-1} \quad \rightsquigarrow \quad \mathbf{Y}_n = \mathbf{M}$$

Multiple right-hand sides

There might be more than one set of RHS values that are of interest.

- Geometry example: find domain points that map to a bunch of range points
- Circuit example: solve same circuit with different source voltages
- Radiosity example: solve energy balance with different emissions

Each of these examples requires solving many systems with the same matrix; e.g. find $\mathbf{x}_1, \mathbf{x}_2$ such that

$$A\mathbf{x}_1 = \mathbf{b}_1 \quad \text{and} \quad A\mathbf{x}_2 = \mathbf{b}_2$$

Multiple right-hand sides

It's easy to package a multi-RHS system as a matrix equation, using the interpretation of matrix multiplication as transforming the columns of the right-hand matrix:

$$\mathbf{A} \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 \end{bmatrix}$$
$$\mathbf{AX} = \mathbf{B}$$

This kind of system is *not* any harder to solve than $\mathbf{Ax} = \mathbf{b}$; we just use the same method but apply the operations to all the columns of $\mathbf{B}$ at once.

Sometimes a system that you form from matrices that you are not thinking of as a stack of columns surprises you by turning out to be a multi-RHS system.

Pivoting in Gauss-Jordan

In our G-J example, the second pivot was 0.1.

This led to a large multiplier (70 and 25) and big numbers in the intermediate results where there were only small numbers in the problem and in the solution.

Worse, we computed the RHS of row 2 as $6.1 - 6.0$. If the pivot was 10^{-6} this would result in complete loss of accuracy in single precision.

It wasn't obvious from the initial problem that this was coming!

Pivoting in Gauss-Jordan

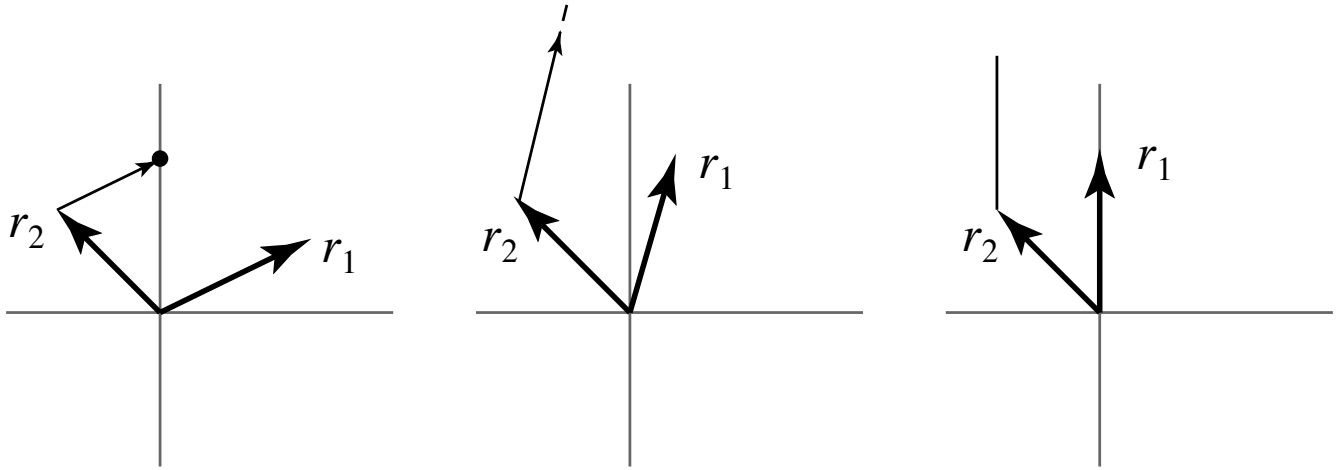
An extreme example

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Obviously $x_1 = 3$ and $x_2 = 2$, but our G-J algorithm will not figure this out.

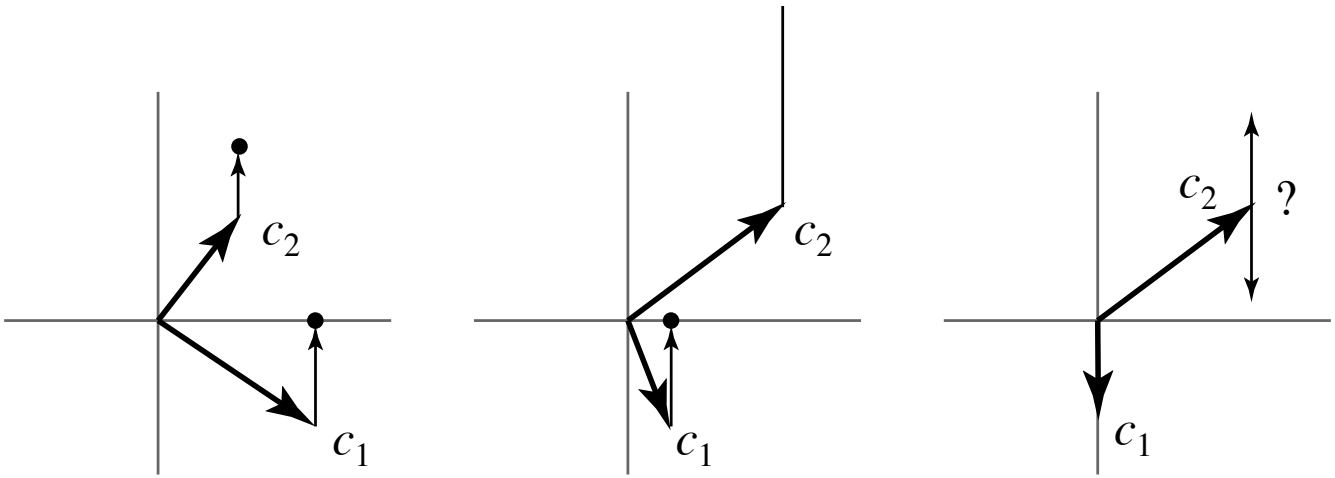
Pivoting in Gauss-Jordan

Geometric intuition: G-J is adding a multiple of row i to the other rows to get them into the plane perpendicular to the i^{th} coordinate axis.



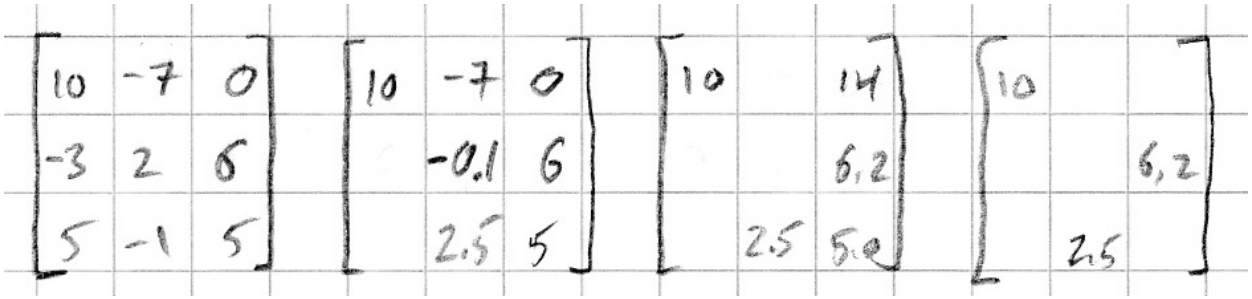
Pivoting in Gauss-Jordan

Geometric intuition: G-J is shearing perpendicular to dimension i to get the i^{th} column onto the i^{th} coordinate axis. (That is, it projects each column in turn onto an axis.)



Pivoting in Gauss-Jordan

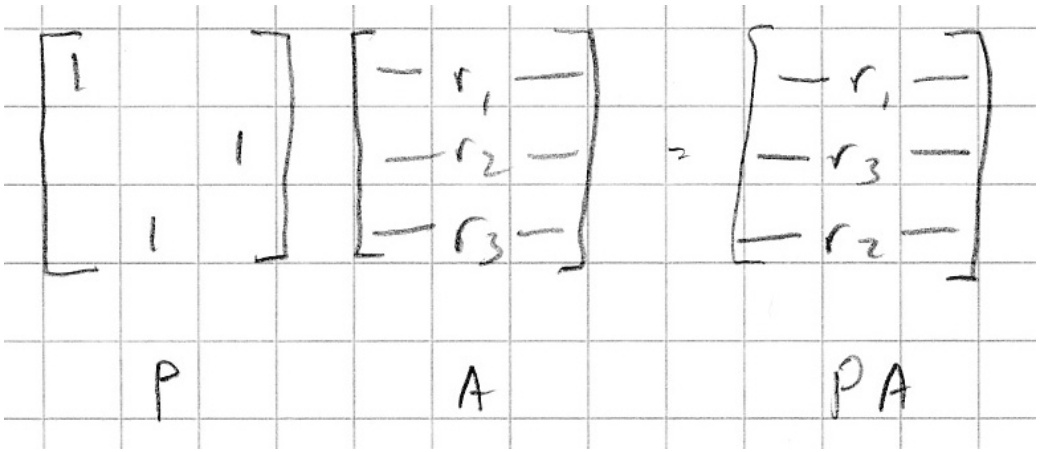
So the pivot problem is just a bad choice of arbitrary axis onto which to project. The solution is simply to process the axes in a different order.



To keep the bookkeeping simple we do this by reordering the rows and then processing them in order.

Pivoting in Gauss-Jordan

Can express this in matrices using a permutation matrix, which is a matrix with a single 1 in each row and column.



Note that we don't physically move the rows of $\mathbf{A}$ around in memory; we just keep track of $\mathbf{P}$ so that we know where to find them.

Computational complexity of G-J

How much arithmetic is involved in G-J?

- One step updates all of columns $i-n$. this takes $n(n-i+1)$ multiply-add operations.
- Do this for $i=1:n$ results in an operation count with leading term:

$$n \sum_{i=0}^{n-1} i \approx \frac{1}{2} n^3$$

Operation counts don't tell the whole story but are indicative.

Gaussian elimination

We can do better than G-J on two fronts:

- Overall operation count
- G-J can't handle new RHS vectors that come along after the fact, unless you compte the inverse.

Can fix both problems by only doing part of the work!

Gaussian elimination

No need to reduce $\mathbf{A}$ all the way down to the identity—only to an "easy" matrix to solve.

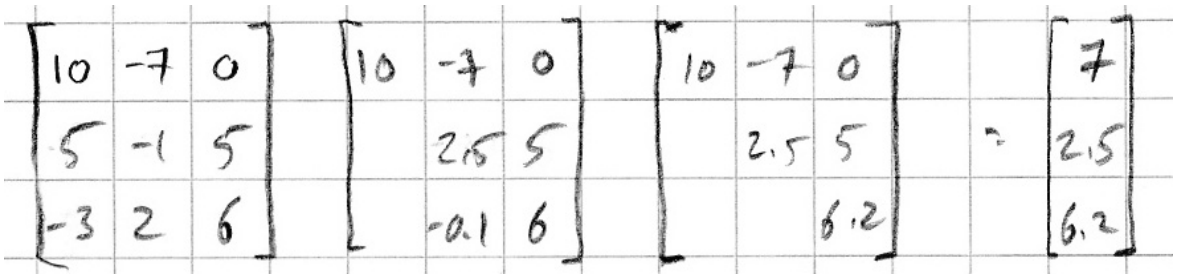
E.g. the version we used for the by-hand examples, without dividing through to make the diagonal entries equal to 1, does this:

$$\mathbf{MA} = \mathbf{D}$$

where $\mathbf{D}$ is diagonal.

Gaussian elimination

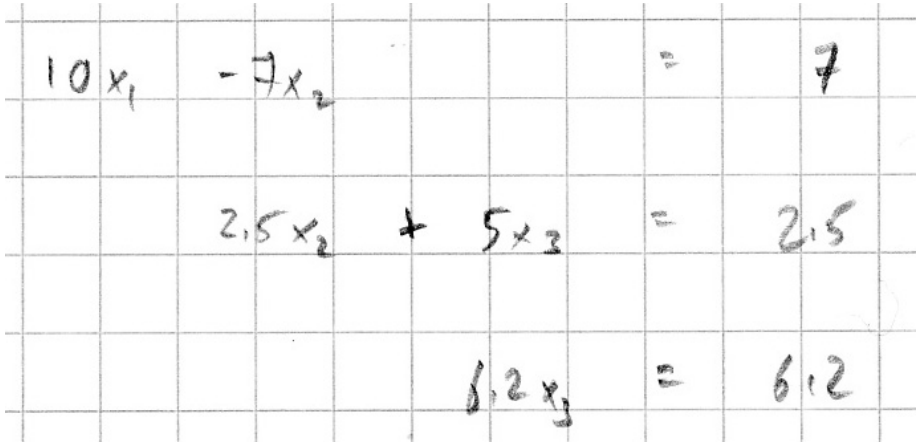
Further laziness leads to only operating on rows *below* the row we're working on:



This is an *upper triangular* matrix—that is, it has zeros below the diagonal ($a_{ij} = 0$ for $i > j$).

Gaussian elimination

The reduced system can be solved easily:



From these equations we can compute the values of the unknowns, starting from x_3. Once we have x_3 we substitute into the second equation to get x_2, and so forth. This is known as *back substitution*.

The LU factorization

This approach can be expressed as a series of Gauss transformations (slightly different from the ones used in G-J because they have zeros above the diagonal):

$$\mathbf{M}_{n-1} \cdots \mathbf{M}_2 \mathbf{M}_1 \mathbf{A} = \mathbf{U}$$
$$\mathbf{M}\mathbf{A} = \mathbf{U}$$

or

$$\mathbf{A} = \mathbf{M}_1^{-1} \mathbf{M}_2^{-1} \cdots \mathbf{M}_{n-1}^{-1} \mathbf{U}$$
$$\mathbf{A} = \mathbf{L}\mathbf{U}$$

Multiplying together the inverse $\mathbf{M}$s in this order serves simply to lay out the multipliers in $\mathbf{L}$, below the diagonal. Try it!

Gaussian elimination

If we keep the multipliers around in this way, Gaussian elimination becomes the *LU factorization*

$$\mathbf{A} = \mathbf{LU}$$

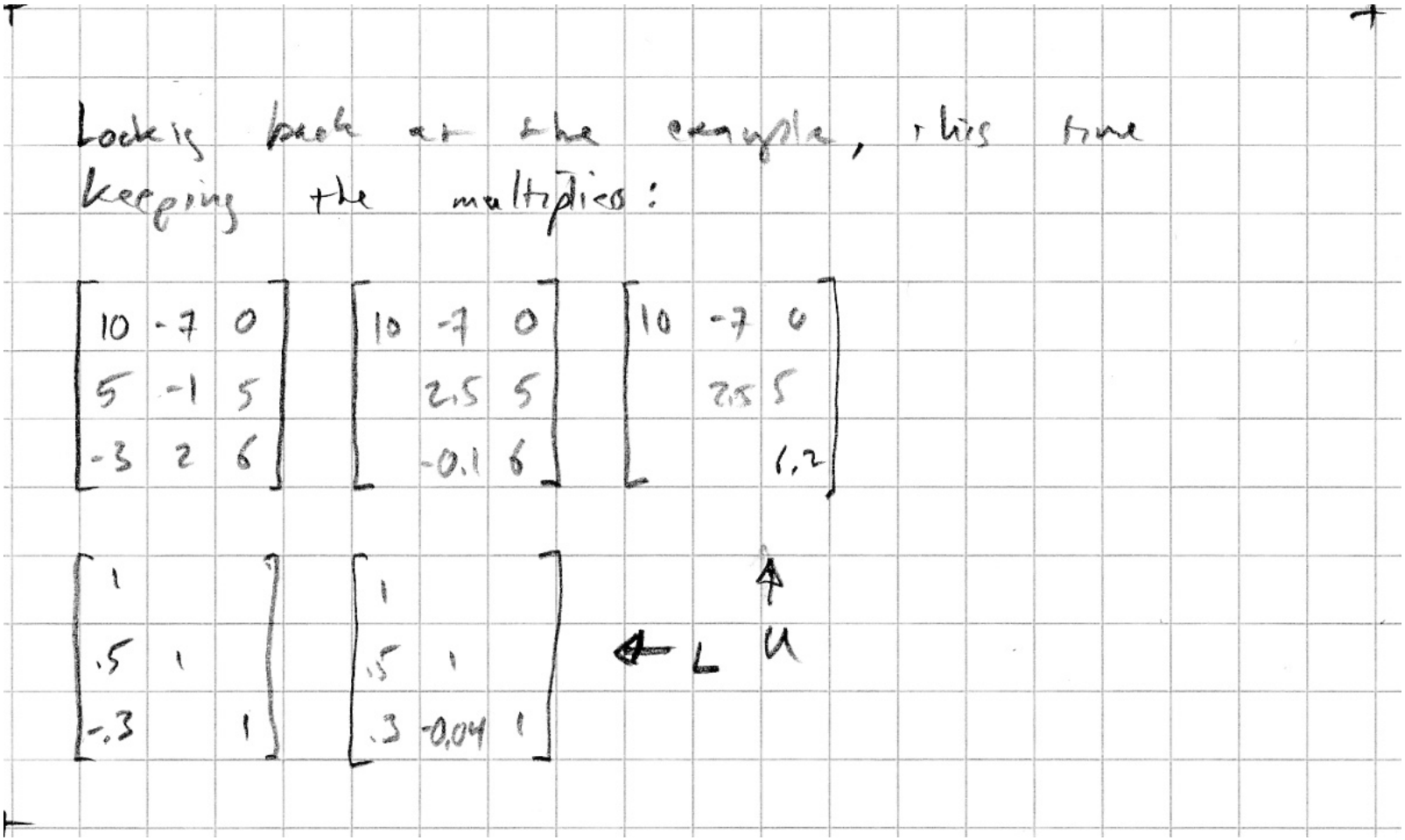
One way to think of this is that the matrix $\mathbf{U}$ is a reduced form of $\mathbf{A}$, ant $\mathbf{L}$ contains the instructions for how to reconstitute $\mathbf{A}$ from $\mathbf{U}$.

An advantage over Gauss-Jordan is that we factor $\mathbf{A}$ with no RHS in sight. Then, to solve $\mathbf{LUx} = \mathbf{b}$:

- solve $\mathbf{Ly} = \mathbf{b}$
- solve $\mathbf{Ux} = \mathbf{y}$.

(another way to say this is we compute $\mathbf{U}^{-1}(\mathbf{L}^{-1}\mathbf{b})$).

The LU factorization



The LU factorization

LU needs pivoting to be stable.

row pivoting like what we've already seen is usually sufficient. This leads to a LU factorization of a permutation of A:

$$PA = LU$$

This is "LU factorization with partial pivoting" and is a good first choice for a general linear system.

Linear systems in Matlab

Solving a linear system in Matlab is simple: to solve $\mathbf{Ax} = \mathbf{b}$, write

```
x = A \ b
```

The idea behind this name for the operator is:

$$ax = b \quad \Longrightarrow \quad x = b/a$$

except that for a matrix $\mathbf{A}$ you have to keep track of whether it is on the left or the right. Since $\mathbf{A}$ is on the left, we use the backslash to remind us that $\mathbf{A}$ is on the left and "in the denominator."

Note that there is also a corresponding forward slash operator that solves systems with $\mathbf{A}$ on the right (basically it solves a system with $\mathbf{A}^T$).

Summary

- Gauss-Jordan is a method for directly solving systems
  - Process the RHS with the matrix
  - Can compute the inverse as a side effect
- LU factorization is both faster and more flexible
  - Process matrix first, then RHSs as they come along
  - Based on a factorization into two triangular matrices
- Both require pivoting for stability
- LU is a good default for general system solving