Unit 2.1 – Linear Algebra

Let's start by considering some basic algebra.

$$5x = 5$$

Solution: $x = 1$

Generalize: $ax = b$ (2.1-1)

Solution: $x = b/a$

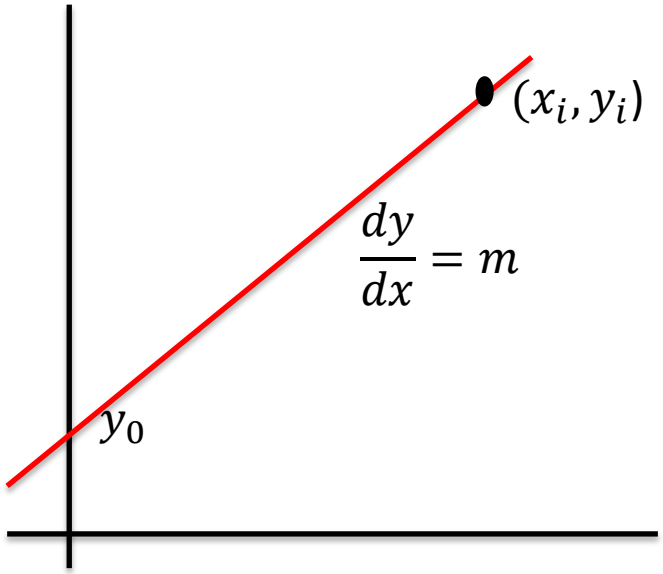
$$ax = b$$
$$(1/a)ax = (1/a)b$$
$$x = b/a \quad (2.1\text{-}2)$$

$(1/a) = a^{-1}$ the inverse of a (2.1-3)

$x = b/a$ is true if $a \neq 0$ – nonsingular

Our equation, $ax = b$, is in 1-space. Can we generalize? Consider 2-space:

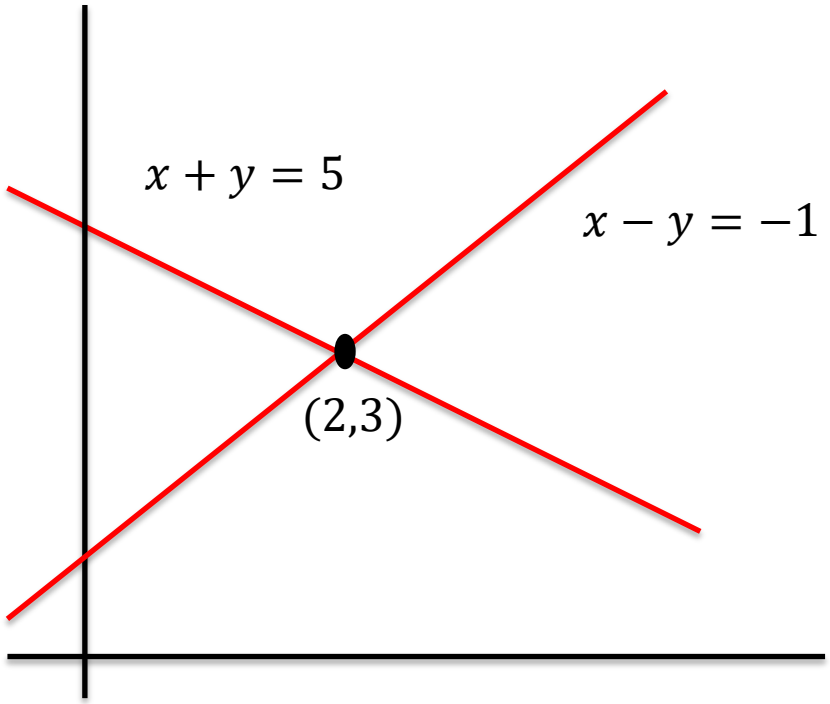
$$y = mx + y_0 \quad (2.1\text{-}4)$$
$$y - y_i = m(x - x_i) \quad (2.1\text{-}5)$$
$$ax + by = c \quad (2.1\text{-}6)$$



What if we had 2 equations?

$$a_1 x + b_1 y = c_1 \qquad (2.1\text{-}7)$$
$$a_2 x + b_2 y = c_2$$

Could we write a general solution to this equation?



$$x + \frac{b_1}{a_1} y = \frac{c_1}{a_1}$$
$$x + \frac{b_2}{a_2} y = \frac{c_2}{a_2}$$

$$\left(\frac{b_1}{a_1} - \frac{b_2}{a_2}\right) y = \frac{c_1}{a_1} - \frac{c_2}{a_2}$$

$$y = \frac{\frac{c_1}{a_1} - \frac{c_2}{a_2}}{\frac{b_1}{a_1} - \frac{b_2}{a_2}} = \frac{c_1 a_2 - a_1 c_2}{b_1 a_2 - a_1 b_2} \qquad (2.1\text{-}8)$$

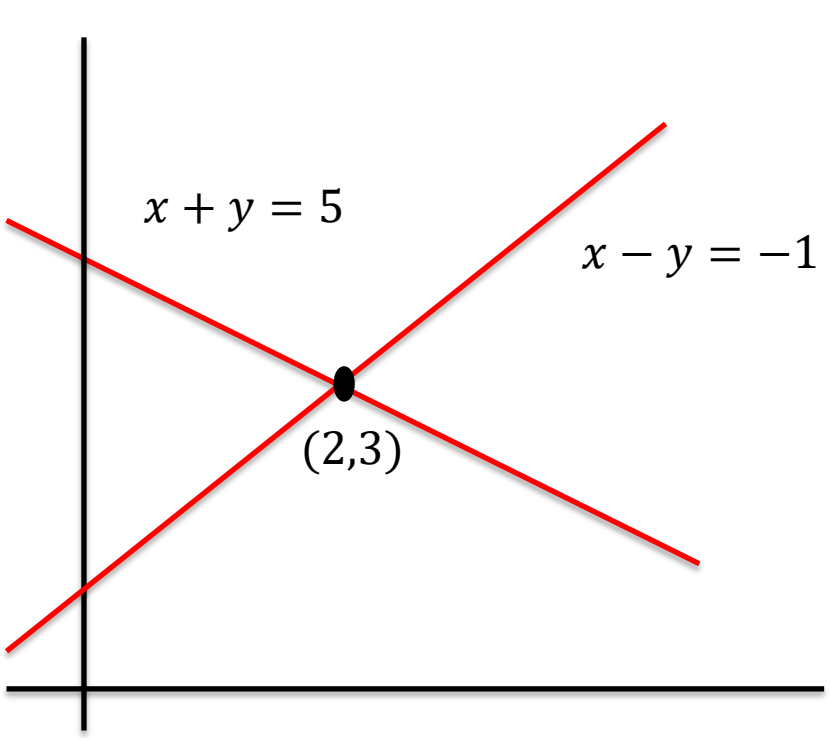
$$x = c_1 - \frac{b_1}{a_1}\left[\frac{c_1 a_2 - a_1 c_2}{b_1 a_2 - a_1 b_2}\right] \qquad (2.1\text{-}9)$$

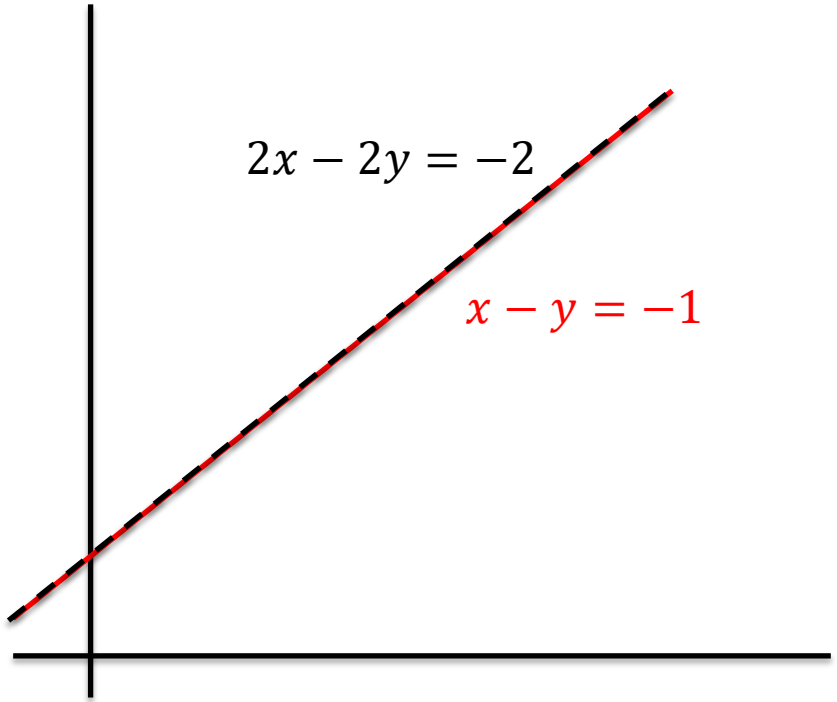
Which works so long as

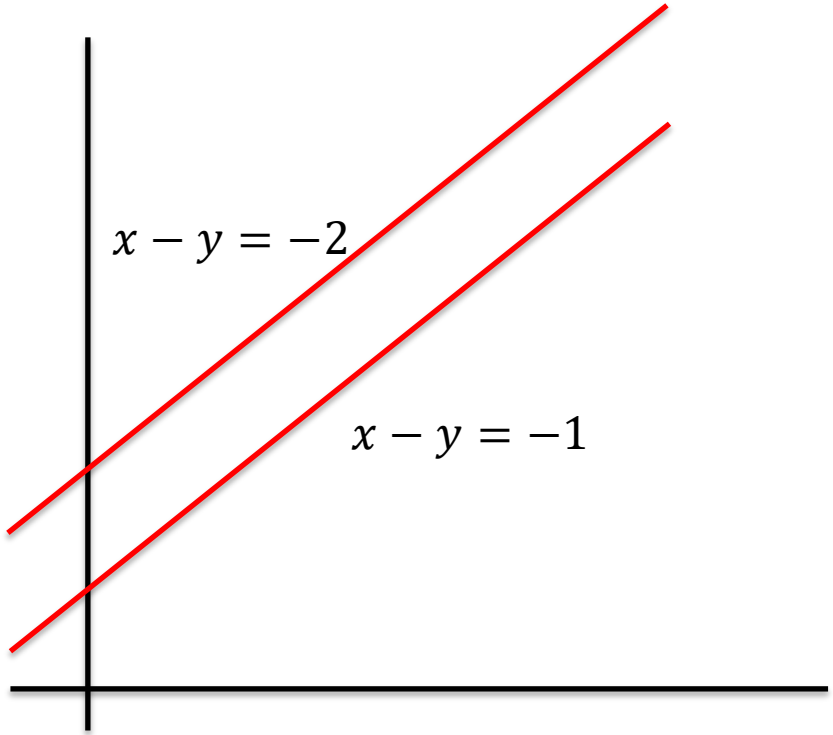
$$b_1 a_2 - a_1 b_2 \neq 0 \qquad (2.1\text{-}10)$$

Since this is in both *x* and *y* it helps us to determine whether we have a solution.

In general, there are 3 possibilities.







1- Unique Solution

$b_1a_2 - a_1b_2 \neq 0$

2- Infinite Solutions

$b_1a_2 - a_1b_2 = 0$ & $c_1a_2 - a_1c_2 = 0$

3- No Solutions

$b_1a_2 - a_1b_2 = 0$

The term $b_1a_2 - a_1b_2$ is called a determinant because of this.

Solutions in 2-space are quite easy to visualize, since we are talking about lines. Even 3-space, where we deal with planes,

$$ax + by + cz = d \qquad (2.1\text{-}11)$$

is not hard to imagine. However, the same concepts apply to n-space, we just need a generalized algebra to explain things.

Consider again the notation of Eq. 2.1-7

$$a_1 x + b_1 y = c_1 \qquad (2.1\text{-}7)$$
$$a_2 x + b_2 y = c_2$$

Which we write as a matrix and vectors using the text's notation

$$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} c_1 \\ c_2 \end{Bmatrix} \qquad (2.1\text{-}12)$$

Note that matrix vector multiplication implies that the sum of the products of a given row of the matrix times the column of the vector equals the corresponding row of the right-hand side (RHS) vector.

A more standard matrix notation writes 2.1-12 as

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \end{Bmatrix} \qquad (2.1\text{-}13)$$

So that you can write

$$[A]\vec{x} = \vec{b} \qquad (2.1\text{-}14)$$

Corresponding to the 1-space notation

$$ax = b \qquad (2.1\text{-}1)$$

Like the 1-space case, if we can find the inverse of $[A]$ we can solve for $\vec{x}$.

$$[A]\vec{x} = \vec{b}$$
$$[A]^{-1}[A]\vec{x} = [A]^{-1}\vec{b}$$
$$[I]\vec{x} = [A]^{-1}\vec{b}$$
$$\vec{x} = [A]^{-1}\vec{b} \qquad (2.1\text{-}15)$$

Where $[I]$ is the identity matrix, sometimes denoted $[I]^n$ to define its dimensions, a matrix of all zeros except ones on the diagonal. In 3 dimensions it has the form:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (2.1\text{-}16)$$

However, it is not a requirement that we find $[A]^{-1}$ to find $\vec{x}$. Consider our earlier example equation:

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} 5 \\ -1 \end{Bmatrix}$$

Add the first equation to the second.

$$\begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} 5 \\ 4 \end{Bmatrix}$$

Immediately, we can solve for x, since

$$2x = 4$$
$$x = 2$$

Then using the first equation we see that

$$y = 5 - (1)(2)$$
$$\mathrm{y} = 3$$

This last step is an example of something called back-substitution.

Our simple example followed the approach we would naturally use, as it "looked like the easier way to solve." A systematic algorithm that we can code is needed.

Given the system:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \end{Bmatrix}$$

The first approach is to alter it by standard algebraic operations on the equations to transform the system into the form:

$$\begin{bmatrix} 1 & a'_{12} \\ 0 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b'_1 \\ b'_2 \end{Bmatrix}$$

Where ' recognizes that the coefficient has changed by some combination of equation addition/subtraction/multiplication/division, allowing us to easily back-solve.

The basic idea is to change the system into one with an upper-triangular matrix.

$$[U] = \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} \qquad (2.1\text{-}17)$$

Note that this is different than what we did in the previous example and is called Gauss-Elimination, the first of the direct methods we will present in this unit.

Starting again with our example problem:

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} 5 \\ -1 \end{Bmatrix}$$

We subtract equation 1 from equation 2, i.e.

$$\begin{bmatrix} 1 & 1 \\ 1-1 & -1-1 \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} 5 \\ -1-5 \end{Bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 0 & -2 \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} 5 \\ -6 \end{Bmatrix}$$

Multiply the second equation by -½ to get

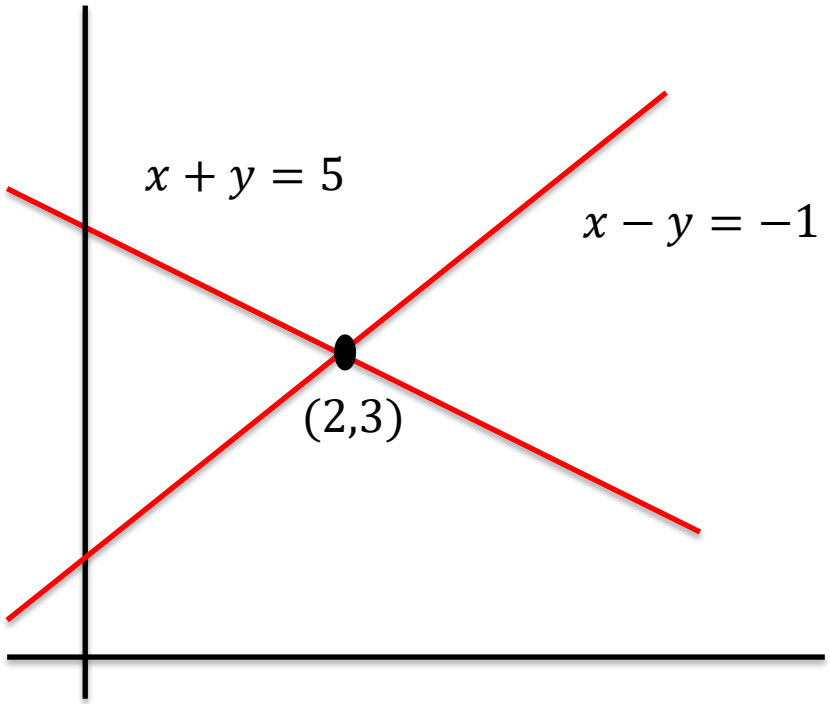
$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} 5 \\ 3 \end{Bmatrix}$$

And can now back-substitute to get

$$y = 3$$

$$x = 5 - (1)3 = 2$$

Hence, our solution is *(2,3)*.



Unit 2.2 – Gauss Elimination (GE)

The basic Gauss Elimination (GE) algorithm modifies the system of equations by a series of operations, such that we obtain an upper triangular system with 1's on the diagonal.

Consider the 3-equation/3-variable system:

$$\begin{bmatrix} 4 & 2 & -2 \\ 2 & 4 & 2 \\ -2 & 3 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 2 \\ 16 \\ 7 \end{Bmatrix}$$

and perform the following operations:

1. Make $a_{11} = 1$ ($\frac{1}{4}$xEq.1)

$$\begin{bmatrix} 1 & \frac{1}{2} & -\frac{1}{2} \\ 2 & 4 & 2 \\ -2 & 3 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{2} \\ 16 \\ 7 \end{Bmatrix}$$

2. Eliminate a_{21} (Eq.2-2xEq.1)

$$\begin{bmatrix} 1 & \frac{1}{2} & -\frac{1}{2} \\ 0 & 3 & 3 \\ -2 & 3 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{2} \\ 15 \\ 7 \end{Bmatrix}$$

3. Eliminate a_{31} (Eq.3+2xEq.1)

$$\begin{bmatrix} 1 & \frac{1}{2} & -\frac{1}{2} \\ 0 & 3 & 3 \\ 0 & 4 & 0 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{2} \\ 15 \\ 8 \end{Bmatrix}$$

4. Make $a_{22} = 1$ ($\frac{1}{3}$xEq.2)

$$\begin{bmatrix} 1 & \frac{1}{2} & -\frac{1}{2} \\ 0 & 1 & 1 \\ 0 & 4 & 0 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{2} \\ 5 \\ 8 \end{Bmatrix}$$

5. Eliminate a_{32} (Eq.3-4xEq.2)

$$\begin{bmatrix} 1 & \frac{1}{2} & -\frac{1}{2} \\ 0 & 1 & 1 \\ 0 & 0 & -4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{2} \\ 5 \\ -12 \end{Bmatrix}$$

6. Make $a_{33} = 1$ (-$\frac{1}{4}$xEq.3)

$$\begin{bmatrix} 1 & \frac{1}{2} & -\frac{1}{2} \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{2} \\ 5 \\ 3 \end{Bmatrix}$$

Now back-solve

$$x_3 = 3$$

$$x_2 = 5 - 3 = 2$$

$$x_1 = \frac{1}{2} + \frac{1}{2}(3) - \frac{1}{2}(2) = 1$$

$$\vec{x} = \begin{Bmatrix} 1 \\ 2 \\ 3 \end{Bmatrix}$$

UNIVERSITY OF Cincinnati

```
## module gaussElimin
''' x = gaussElimin(a,b).
    Solves [a]{b} = {x} by Gauss elimination.
'''
import numpy as np

def gaussElimin(a,b):
    n = len(b)
  # Elimination Phase
    for k in range(0,n-1):
        for i in range(k+1,n):
            if a[i,k] != 0.0:
                lam = a [i,k]/a[k,k]
                a[i,k+1:n] = a[i,k+1:n] - lam*a[k,k+1:n]
                b[i] = b[i] - lam*b[k]
  # Back substitution
    for k in range(n-1,-1,-1):
        b[k] = (b[k] - np.dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
    return b
```

The basic question of will the algorithm work involves finding the equivalent of Eq. 2.1-10, $b_1 a_2 - a_1 b_2 \neq 0$, the so-called Determinant.

Notation for determinant of $[A]$:

$$\det(A) = |A|$$

The simplest case is 1-space is $|a| = a$ for the case of $ax = b$.

In 2-space

$$\det \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = a_{11}\, a_{22} - a_{12}\, a_{21} \quad (2.2\text{-}1)$$

For k-space we need to introduce the idea of Minors.

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1k} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & a_{k3} & \cdots & a_{kk} \end{vmatrix} = a_{11} \begin{vmatrix} a_{22} & a_{23} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k2} & a_{k3} & \cdots & a_{kk} \end{vmatrix}$$

$$\ominus a_{12} \begin{vmatrix} a_{21} & a_{23} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k3} & \cdots & a_{kk} \end{vmatrix} + \ldots \oplus a_{1k} \begin{vmatrix} a_{21} & a_{22} & \cdots & a_{2(k-1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{k(k-1)} \end{vmatrix}.$$

mathworld.wolfram.com/Determinant.html

Example, 3x3:

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} =$$

$$a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} \ominus a_{12} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

The minors approach reduces a nxn determinant to a series of (n-1)x(n-1) determinants.

Easy Approach: Use the function in numpy

numpy.org/doc/stable/reference/routines.linalg.html#module-numpy.linalg

Its use is

numpy.linalg.det(a)

If $|A| = 0$, we either have no solution or indeterminate solutions. To illustrate we will introduce a very simple direct method that uses determinants.

Cramer's rule (not in the text)

For the system $[A]\vec{x} = \vec{b}$ $\begin{bmatrix} 4 & 2 & -2 \\ 2 & 4 & 2 \\ -2 & 3 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 2 \\ 16 \\ 7 \end{Bmatrix}$

$$x_i = \frac{\det(A_i)}{\det(A)}$$

Where the matrix $[A_i]$ is obtained by replacing the *i-th* column of $[A]$ with $\vec{b}$.

For our previous example:

$$x_2 = \frac{\begin{vmatrix} 4 & 2 & -2 \\ 2 & 16 & 2 \\ -2 & 7 & 1 \end{vmatrix}}{\begin{vmatrix} 4 & 2 & -2 \\ 2 & 4 & 2 \\ -2 & 3 & 1 \end{vmatrix}} = \frac{-95.999\cdots}{-47.999\cdots} \approx 2$$

If we have:

$\det(A_i) \neq 0 \ \forall i$, but $\det(A)$=0 – no solution

(same matrix row with different RHS)

$\det(A_i) = 0$ for any *i* and $\det(A)$=0 – indeterminate

(linearly dependent equations)

Note: Cramer's rule isn't often used because of computation time. A single determinant calculation requires at best $\mathcal{O}(n^{2.373})^*$ operations and you need *n* of them, so it is at best an $\mathcal{O}(n^{3.373})$ algorithm.

GE takes $\mathcal{O}(n^3/_3)$ operations for the elimination phase + $\mathcal{O}(n^2/_2)$, not great but less than Cramer's rule. We'll talk about quicker methods later.

*en.Wikipedia.org/wiki/Computational_complexity_of_mathematical_operations#Matrix_algebra

Condition Number

We know we can't get a unique solution for $\det(A) = 0$, but will we have a problem if $\det(A)$ is small?

In a formal sense, the condition number answers this question.

$$\text{cond}(A) = \|A\|\|A^{-1}\|$$

Where $\|A\|$ could mean several things.

$$\|A\|_e = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2}$$ - Euclidean Norm

$$\|A\|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|$$ - Row-sum Norm

But these are super expensive to compute. Best bet is to compare the smallest element of $[A]$ to $\det(A)$.

If $\det(A)$ is much smaller than the smallest element of $[A]$ you will have problems.

$$1 \le \text{cond}(A) \le \infty$$

Well conditioned — Ill conditioned

A large condition number says that small changes in the matrix coefficients will make large changes in the results.

So, if the numbers we use have any errors, the solution will be suspect.

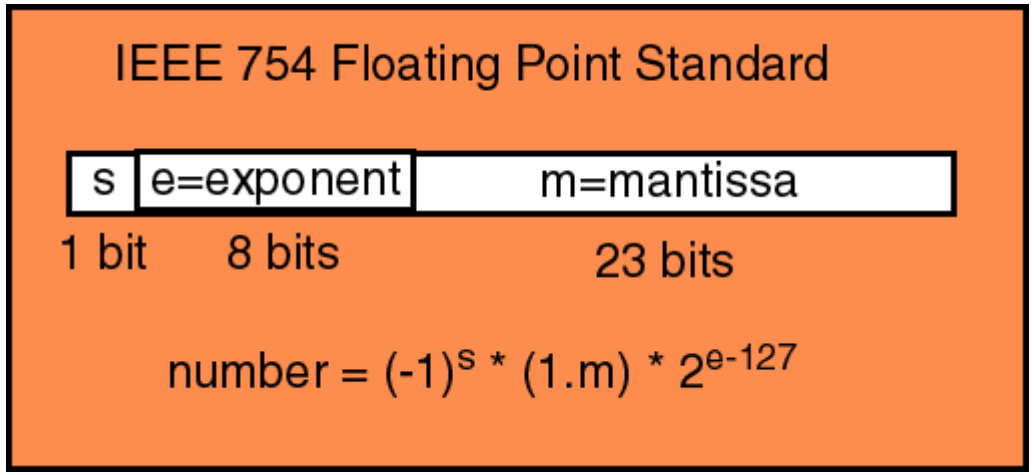
If it is ill-conditioned we need to be careful about the number of operations.

This may seem confusing, since we have the exact algorithms, why should it matter?

The quick answer is that we solving numerically on a computer and we don't have infinite numbers.

Unit 2.3 – Symbolic vs. Computer Arithmetic

Computers store numbers using binary representation, a series of 0/1s. There are several standards used, but IEEE-754 gives a reasonable understanding of how it works for the simple case of 32-bit numbers.



The idea is best summarized by visualizing a 5-bit number system, 1 for the sign, 2 for the exponent, and 2 for the mantissa.

In this 5-bit system a number takes the form:

$$Number=(-1)^{s}*(1.mm)*2^{ee} \qquad (2.3\text{-}1)$$

This is understood to mean

$$Number = (-1)^{s}\left(1+\frac{m_1}{2}+\frac{m_2}{4}\right)2^{(-1)^{e_1}e_2}$$

In a more compact notation, the bits can be written

$$seemm \qquad (2.3\text{-}2)$$

Then 00111 becomes

$$00111 = (-1)^{0}\left(1+\frac{1}{2}+\frac{1}{4}\right)2^{(-1)^{0}1}$$

$$=\left(\frac{7}{4}\right)2=\frac{7}{2}=3.5$$

The represented numbers are:

$00000 = (-1)^0(1)2^0$ = 1

$00001 = (-1)^0\left(1+\frac{1}{4}\right)2^0$ = $1\frac{1}{4}$

$00010 = (-1)^0\left(1+\frac{1}{2}\right)2^0$ = $1\frac{1}{2}$

$00011 = (-1)^0\left(1+\frac{1}{2}+\frac{1}{4}\right)2^0$ = $1\frac{3}{4}$

$00100 = (-1)^0(1)2^1$ = 2

$00101 = (-1)^0\left(1+\frac{1}{4}\right)2^1$ = $2\frac{1}{2}$

$00110 = (-1)^0\left(1+\frac{1}{2}\right)2^1$ = 3

$00111 = (-1)^0\left(1+\frac{1}{2}+\frac{1}{4}\right)2^1$ = $3\frac{1}{2}$

$01000 = (-1)^0(1)2^{-0}$ = ?

$01001 = (-1)^0\left(1+\frac{1}{4}\right)2^{-0}$ = ?

$01010 = (-1)^0\left(1+\frac{1}{2}\right)2^{-0}$ = ?

$01011 = (-1)^0\left(1+\frac{1}{2}+\frac{1}{4}\right)2^{-0}$ = ?

$01100 = (-1)^0(1)2^{-1}$ = $\frac{1}{2}$

$01101 = (-1)^0\left(1+\frac{1}{4}\right)2^{-1}$ = $\frac{5}{8}$

$01110 = (-1)^0\left(1+\frac{1}{2}\right)2^{-1}$ = $\frac{3}{4}$

$01111 = (-1)^0\left(1+\frac{1}{2}+\frac{1}{4}\right)2^{-1}$ = $\frac{7}{8}$

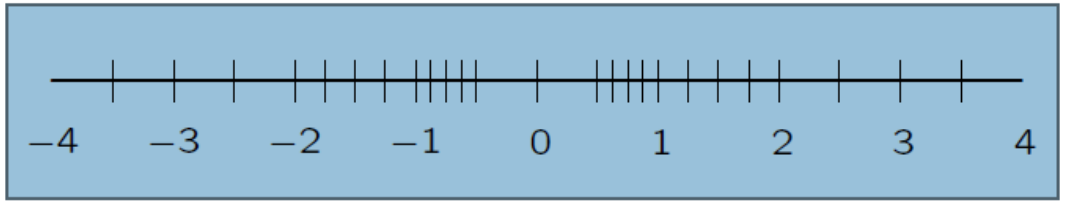
and their opposites with s=1

The only question being: What of the -0 exponent? Define then

$$(01000)_2=0_{10} \qquad (2.3\text{-}3)$$

and either take the others to be the same or undefined.

That means we have a system with 25 total numbers and the number line.



$$OFL = (1.11)_2 \times 2^1 = (3.5)_{10}$$

$$UFL = (1.00)_2 \times 2^{-1} = (0.5)_{10}$$

Max nonzero

Min nonzero

All discrete computational number systems show the same kind of behavior at some magnification. What you notice is even spacing between powers of 2 and more numbers in the vicinity of 1.

Numbers that don't exist in the computer are rounded to ones that do. Therefore, roundoff error is reduced near 1.

The natural question is, “how does this affect an algorithm like GE?”

The problem essentially occurs when you have small values along the diagonals.

Suppose:

$$[A \quad |b] = \left[\begin{array}{ccc|c} \epsilon & -1 & 1 & 0 \\ -1 & 2 & -1 & 0 \\ 2 & -1 & 0 & 1 \end{array}\right]$$

One step of GE gives:

$$\left[\acute{A} \quad |\acute{b}\right] = \left[\begin{array}{ccc|c} \epsilon & -1 & 1 & 0 \\ 0 & 2-\frac{1}{\epsilon} & -1+\frac{1}{\epsilon} & 0 \\ 0 & -1+\frac{2}{\epsilon} & -\frac{2}{\epsilon} & 1 \end{array}\right]$$

But if ϵ is small, $\frac{1}{\epsilon}$ is large. So that if they encounter roundoff, the system might become

$$\left[\acute{A} \quad |\acute{b}\right] = \left[\begin{array}{ccc|c} \epsilon & -1 & 1 & 0 \\ 0 & -\frac{1}{\epsilon} & +\frac{1}{\epsilon} & 0 \\ 0 & +\frac{2}{\epsilon} & -\frac{2}{\epsilon} & 1 \end{array}\right]$$

And we might change from a linearly independent to a linearly dependent system, or simply produce a result with considerable error.

The solution is to switch rows, so that the largest element is always on the diagonal.

This idea is called pivoting and can be applied in many instances.

Example:

$$\left[\begin{array}{ccc|c} \frac{1}{100} & 2 & 3 & \frac{501}{100} \\ 1 & \frac{1}{200} & 5 & \frac{1201}{200} \\ 2 & 1 & 2 & 5 \end{array}\right] \qquad (2.3\text{-}3)$$

Pivot

$$\left[\begin{array}{ccc|c} 2 & 1 & 2 & 5 \\ 1 & \frac{1}{200} & 5 & \frac{1201}{200} \\ \frac{1}{100} & 2 & 3 & \frac{501}{100} \end{array}\right]$$

$$\left[\begin{array}{ccc|c} 2 & 1 & 2 & 5 \\ 0 & \frac{399}{200} & \frac{299}{100} & \frac{997}{200} \\ 0 & \frac{-99}{200} & 4 & \frac{701}{200} \end{array}\right]$$

$$\left[\begin{array}{ccc|c} 2 & 1 & 2 & 5 \\ 0 & \frac{399}{200} & \frac{299}{100} & \frac{997}{200} \\ 0 & 0 & \frac{189210}{39900} & \frac{378402}{79800} \end{array}\right]$$

~5

Making the diagonals 1

$$\left[\begin{array}{ccc|c} 1 & \frac{1}{2} & 1 & \frac{5}{2} \\ 0 & 1 & \frac{598}{399} & \frac{997}{399} \\ 0 & 0 & 1 & 1 \end{array}\right]$$

Back substitution gives:

$$x_3 = 1$$

$$x_2 = \frac{997}{399} - \frac{598}{399} = \frac{399}{399} = 1$$

$$x_1 = \frac{5}{2} - 1 - \frac{1}{2} = 1$$

However, taking 1 step of GE from Eq. 2.3-1 gives:

$$\left[\begin{array}{ccc|c} \frac{1}{100} & 2 & 3 & \frac{501}{100} \\ 0 & -\frac{39999}{200} & -295 & \frac{98999}{200} \\ 0 & -99 & -147 & -\frac{491}{2} \end{array}\right]$$

~500

So clearly not a problem for 32-bit numbers at this point, but scale this up and consider a number like 1e-15, so that the straight GE gives numbers on the order of 1e+15, and we see that we begin to develop roundoff problems.

The Gauss Elimination with Pivoting algorithm solves this problem.

UNIVERSITY OF Cincinnati

■ gaussPivot

The function gaussPivot performs Gauss elimination with row pivoting. Apart from row swapping, the elimination and solution phases are identical to gaussElimin in Section 2.2.

```
## module gaussPivot
''' x = gaussPivot(a,b,tol=1.0e-12).
    Solves [a]{x} = {b} by Gauss elimination with
    scaled row pivoting

import numpy as np
import swap
import error

def gaussPivot(a,b,tol=1.0e-12):
    n = len(b)

  # Set up scale factors
    s = np.zeros(n)
    for i in range(n):
        s[i] = max(np.abs(a[i,:]))

    for k in range(0,n-1):

      # Row interchange, if needed
        p = np.argmax(np.abs(a[k:n,k])/s[k:n]) + k
        if abs(a[p,k]) < tol: error.err('Matrix is singular')
        if p != k:
            swap.swapRows(b,k,p)
            swap.swapRows(s,k,p)
            swap.swapRows(a,k,p)

      # Elimination
        for i in range(k+1,n):
            if a[i,k] != 0.0:
                lam = a[i,k]/a[k,k]
                a[i,k+1:n] = a[i,k+1:n] - lam*a[k,k+1:n]
                b[i] = b[i] - lam*b[k]
    if abs(a[n-1,n-1]) < tol: error.err('Matrix is singular')

  # Back substitution
    b[n-1] = b[n-1]/a[n-1,n-1]
    for k in range(n-2,-1,-1):
        b[k] = (b[k] - np.dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
    return b
```

Unit 2.4 – LU Decomposition and Matrix Inverse

The text shows a comparison of different direct methods. All have $\mathcal{O}(n^3)$ operations.

Method	Initial form	Final form
Gauss elimination	$\mathbf{Ax} = \mathbf{b}$	$\mathbf{Ux} = \mathbf{c}$
LU decomposition	$\mathbf{Ax} = \mathbf{b}$	$\mathbf{LUx} = \mathbf{b}$
Gauss-Jordan elimination	$\mathbf{Ax} = \mathbf{b}$	$\mathbf{Ix} = \mathbf{c}$

Table 2.1. Three Popular Direct Methods

The big difference between the approaches is that only LU decomposition leaves the RHS unchanged. That means whenever a new RHS is needed, no new elimination work is needed, the LU decomposition remains the same.

LU Decomposition

The basic idea is to decompose $[A]$ into lower and upper diagonal matrices $[L]$ & $[U]$.

$$[A] = [L][U] \qquad (2.4\text{-}1)$$

This requires a quick description of matrix-matrix multiplication. The *i*-th row, *j*-th column element of $[A]$, is equal to the sum of the products of the *i*-th row of $[L]$ and the *j*-th column of $[U]$.

Example:

$$\begin{bmatrix} 8 & -6 & 2 \\ -4 & 11 & -7 \\ 4 & -7 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ -1 & 2 & 0 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 4 & -3 & 1 \\ 0 & 4 & -3 \\ 0 & 0 & 2 \end{bmatrix}$$

$$\begin{aligned} a_{22} &= l_{21}u_{12} + l_{22}u_{22} + l_{23}u_{32} \\ &= (-1)(-3) + (2)(4) + (0)(0) \\ &= 11 \end{aligned}$$

So for

$$[A]\vec{x} = \vec{b} \qquad (2.4\text{-}2)$$

$$[L][U]\vec{x} = \vec{b}$$

If we know $[L]$ and $[U]$, we forward-substitute to find intermediate vector $\vec{y}$

$$[L]\vec{y} = \vec{b} \qquad (2.4\text{-}3)$$

And solve for $\vec{x}$ via

$$[U]\vec{x} = \vec{y} \qquad (2.4\text{-}4)$$

Note that *LU* decomposition is NOT unique.

A useful approach is Doolittle's, which defines the diagonal elements of $[L]$ to be 1. Allowing both $[L]$ and $[U]$ to be written to the memory locations holding $[A]$.

The technique is summarized by

$$[A] = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ U_{11}L_{21} & U_{12}L_{21} + U_{22} & U_{13}L_{21} + U_{23} \\ U_{11}L_{31} & U_{12}L_{31} + U_{22}L_{32} & U_{13}L_{31} + U_{23}L_{32} + U_{33} \end{bmatrix} \qquad (2.4\text{-}5)$$

We see that the upper elements of $[U]$are identical to $[A]$, and, once known, a process is easily found for the off diagonal elements of $[L]$ and the remaining elements of $[U]$.

LUdecomp is found in the text to implement the scheme as described.

The idea of pivoting again applies and is summarized in the LUpivot algorithm also found in the text.

Matrix Inverse

The previously presented GE and LU decomposition methods can be applied to find $[A]^{-1}$ if the system can be treated as

$$[A]\,[X] = [B] \qquad (2.4\text{-}6)$$

Where $[X]$ and $[B]$ are *n*x*m* matrices whose columns consist, respectively, of the solution vectors for the associated RHSs. The *LU* decomposition is particularly well suited for this since, once computed, the $[L]$ and $[U]$ matrices do not need to be again computed.

Example code is provided in the text.

However, with this approach we can find $[A]^{-1}$ using the idea that

$$[A][A]^{-1} = [I] \qquad (2.4\text{-}7)$$

So that in Eq. 2.4-6 the $[B]$ matrix consists of the unit vectors that define $[I]$, and the $[X]$ represents $[A]^{-1}$.

Warning, if GE is used, this technique requires *n* $[A]\vec{x} = \vec{b}$ problems. Hence, it requires $\mathcal{O}(n^4)$ operations to find $[A]^{-1}$.

Other Methods

We won't have time to review every method in the text, but, depending on the form of $[A]$, faster methods are possible. Methods are available for symmetric matrices, where

$$a_{ij} = a_{ji} \qquad (2.4\text{-}8)$$

And for banded matrices, where the diagonal terms and some next to them are nonzero, but the majority are zeros.