

# **Gaussian Elimination (+ Numerics)**

**Geometric Algorithms  
Lecture 4**

# Practice Problem

$$x + z = 1$$

$$x + y + 3z = 3$$

$$x - y - z = -1$$

*Write down the general forms solution of the above linear system.*

# Solution

$$x + z = 1$$

$$x + y + 3z = 3$$

$$x - y - z = -1$$

# Objectives

1. (Finally) discuss Gaussian elimination
2. Think more carefully about number representations
3. Look at the consequences of floating point representations
4. Introduce NumPy and talk about best best practices

# Keywords

forward elimination

back substitution

floating point numbers

IEEE-754

relative error

`numpy.isclose`

ill-conditioned problems

# Defining the Gaussian Elimination (GE) Algorithm

**At a High Level**

# At a High Level

eliminations + back-substitution



# At a High Level

eliminations + back-substitution

*we've already done this*

# At a High Level

eliminations + back-substitution

*we've already done this*

but we'll take one step further and write down  
the algorithm as pseudocode

# At a High Level

eliminations + back-substitution

*we've already done this*

but we'll take one step further and write down  
the algorithm as pseudocode

**Keep in mind.** How do we turn our intuitions  
into a formal procedure?

# **A Word of Warning**

# **A Word of Warning**

The details of Gaussian elimination are tricky.

# **A Word of Warning**

The details of Gaussian elimination are tricky.

The goal is not to understand it entirely, but to get enough intuition to emulate it.

# **A Word of Warning**

The details of Gaussian elimination are tricky.

The goal is not to understand it entirely, but to get enough intuition to emulate it.

**You should roughly use Gaussian Elimination when solving a system by hand.**

demo  
(step-throughs)



# The Algorithm

# Gaussian Elimination (Specification)

**FUNCTION** GE(A):

**# INPUT:**  $m \times n$  matrix A

**# OUTPUT:** equivalent  $m \times n$  RREF matrix

...

# Gaussian Elimination (High Level)

**FUNCTION** fwd\_elim(A):

# **INPUT:**  $m \times n$  matrix A

# **OUTPUT:** equivalent  $m \times n$  echelon form matrix

...

**FUNCTION** back\_sub(A):

# **INPUT:**  $m \times n$  echelon form matrix A

# **OUTPUT:** equivalent  $m \times n$  RREF matrix

...

**FUNCTION** GE(A):

**RETURN** back\_sub(fwd\_elim(A))

# Elimination Stage

# Elimination Stage (High Level)

# Elimination Stage (High Level)

**Input:** matrix  $A$  of size  $m \times n$

**Output:** echelon form of  $A$

# Elimination Stage (High Level)

**Input:** matrix  $A$  of size  $m \times n$

**Output:** echelon form of  $A$

starting at the top left and move down, find a leading entry and eliminate it from latter equations

# Edge cases



# Edge cases

What if the first equation doesn't have the variable  $x_1$ ?

# Edge cases

What if the first equation doesn't have the variable  $x_1$ ?

**Swap rows with an equation that does.**

# Edge cases

What if the first equation doesn't have the variable  $x_1$ ?

**Swap rows with an equation that does.**

What if *none* of the equations have the variable  $x_1$ ?

# Edge cases

What if the first equation doesn't have the variable  $x_1$ ?

**Swap rows with an equation that does.**

What if *none* of the equations have the variable  $x_1$ ?

**Find the *leftmost* variable which appears in *any* of the remaining equations.**

# Elimination Stage (Pseudocode)

# Elimination Stage (Pseudocode)

**FUNCTION** fwd\_elim(A):

# Elimination Stage (Pseudocode)

**FUNCTION** fwd\_elim(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

# Elimination Stage (Pseudocode)

**FUNCTION** fwd\_elim(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

**IF** [rows i...m are all-zeros]: # if remaining rows are zero



# Elimination Stage (Pseudocode)

**FUNCTION** fwd\_elim(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

**IF** [rows i...m are all-zeros]: # if remaining rows are zero

**RETURN** A

# Elimination Stage (Pseudocode)

**FUNCTION** fwd\_elim(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

**IF** [rows i...m are all-zeros]: # if remaining rows are zero

**RETURN** A

**ELSE:**

# Elimination Stage (Pseudocode)

**FUNCTION** fwd\_elim(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

**IF** [rows  $i \dots m$  are all-zeros]: # if remaining rows are zero

**RETURN** A

**ELSE:**

(j, k)  $\leftarrow$  [position of leftmost entry in the rows  $i \dots m$ ]

# Elimination Stage (Pseudocode)

**FUNCTION** fwd\_elim(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

**IF** [rows  $i \dots m$  are all-zeros]: # if remaining rows are zero

**RETURN** A

**ELSE:**

(j, k)  $\leftarrow$  [position of leftmost entry in the rows  $i \dots m$ ]

[swap row i and row j]

# Elimination Stage (Pseudocode)

**FUNCTION** fwd\_elim(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

**IF** [rows  $i \dots m$  are all-zeros]: # if remaining rows are zero

**RETURN** A

**ELSE:**

(j, k)  $\leftarrow$  [position of leftmost entry in the rows  $i \dots m$ ]

[swap row i and row j]

**FOR** [l from  $i + 1$  to m]: # for all remaining rows

# Elimination Stage (Pseudocode)

**FUNCTION** fwd\_elim(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

**IF** [rows  $i \dots m$  are all-zeros]: # if remaining rows are zero

**RETURN** A

**ELSE:**

(j, k)  $\leftarrow$  [position of leftmost entry in the rows  $i \dots m$ ]

[swap row i and row j]

**FOR** [l from i + 1 to m]: # for all remaining rows

[zero out  $A[l, k]$  using a replacement operation]

# Elimination Stage (Pseudocode)

**FUNCTION** fwd\_elim(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

**IF** [rows  $i \dots m$  are all-zeros]: # if remaining rows are zero

**RETURN** A

**ELSE:**

(j, k)  $\leftarrow$  [position of leftmost entry in the rows  $i \dots m$ ]

[swap row i and row j]

**FOR** [l from  $i + 1$  to m]: # for all remaining rows

[zero out  $A[l, k]$  using a replacement operation]

**RETURN** A

# Elimination Stage (Example)

$$\begin{bmatrix} 0 & 3 & -6 & 6 & 4 & -5 \\ 3 & -7 & 8 & -5 & 8 & 9 \\ 3 & -9 & 12 & -9 & 6 & 15 \end{bmatrix}$$



# Elimination Stage (Example)

leftmost  
nonzero  
entry

$$\begin{bmatrix} 0 & 3 & -6 & 6 & 4 & -5 \\ 3 & -7 & 8 & -5 & 8 & 9 \\ 3 & -9 & 12 & -9 & 6 & 15 \end{bmatrix}$$

# Elimination Stage (Example)

leftmost  
nonzero  
entry

$$\begin{bmatrix} 0 & 3 & -6 & 6 & 4 & -5 \\ 3 & -7 & 8 & -5 & 8 & 9 \\ 3 & -9 & 12 & -9 & 6 & 15 \end{bmatrix}$$

Swap  $R_1$  and  $R_3$

# Elimination Stage (Example)

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 3 & -7 & 8 & -5 & 8 & 9 \\ 0 & 3 & -6 & 6 & 4 & -5 \end{bmatrix}$$

# Elimination Stage (Example)

next entry  
to zero

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 3 & -7 & 8 & -5 & 8 & 9 \\ 0 & 3 & -6 & 6 & 4 & -5 \end{bmatrix}$$

# Elimination Stage (Example)

next entry  
to zero

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 3 & -7 & 8 & -5 & 8 & 9 \\ 0 & 3 & -6 & 6 & 4 & -5 \end{bmatrix}$$

$$R_3 \leftarrow R_3 - R_1$$

# Elimination Stage (Example)

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 3 & -6 & 6 & 4 & -5 \end{bmatrix}$$

# Elimination Stage (Example)

leftmost  
nonzero  
entry

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 3 & -6 & 6 & 4 & -5 \end{bmatrix}$$

# Elimination Stage (Example)

leftmost  
nonzero  
entry

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 3 & -6 & 6 & 4 & -5 \end{bmatrix}$$

swap  $R_2$  with  $R_2$



# Elimination Stage (Example)

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 3 & -6 & 6 & 4 & -5 \end{bmatrix}$$

# Elimination Stage (Example)

next entry  
to zero

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 3 & -6 & 6 & 4 & -5 \end{bmatrix}$$

# Elimination Stage (Example)

next entry  
to zero

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 3 & -6 & 6 & 4 & -5 \end{bmatrix}$$

$$R_3 \leftarrow R_3 - \frac{3R_2}{2}$$

# Elimination Stage (Example)

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Elimination Stage (Example)

leftmost  
nonzero  
entry

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Elimination Stage (Example)

leftmost  
nonzero  
entry

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

swap  $R_3$  with  $R_3$

# Elimination Stage (Example)

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Elimination Stage (Example)

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

done with elimination stage  
going to back substitution stage



**Back Substitution Stage**

# **Back Substitution Stage (High Level)**

# Back Substitution Stage (High Level)

**Input:** matrix  $A$  of size  $m \times n$  in echelon form

**Output:** reduced echelon form of  $A$

# Back Substitution Stage (High Level)

**Input:** matrix  $A$  of size  $m \times n$  in echelon form

**Output:** reduced echelon form of  $A$

scale pivot positions and eliminate the variables for that column from the other equations

# Back Substitution (Pseudocode)

# Back Substitution (Psuedocode)

**FUNCTION** back\_sub(A) :

# Back Substitution (Psuedocode)

```
FUNCTION back_sub(A):
```

```
  FOR [i from 1 to m]: # for each row from top to bottom
```

# Back Substitution (Pseudocode)

```
FUNCTION back_sub(A):
```

```
  FOR [i from 1 to m]: # for each row from top to bottom
```

```
    IF [row i has a leading entry]:
```



# Back Substitution (Pseudocode)

**FUNCTION** back\_sub(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

**IF** [row i has a leading entry]:

j ← index of leading entry of row i

# Back Substitution (Pseudocode)

**FUNCTION** back\_sub(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

**IF** [row i has a leading entry]:

j ← index of leading entry of row i

$R_i(A) \leftarrow R_i(A) / A[i, j]$  # divide by leading entry

# Back Substitution (Pseudocode)

**FUNCTION** back\_sub(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

**IF** [row i has a leading entry]:

j ← index of leading entry of row i

$R_i(A) \leftarrow R_i(A) / A[i, j]$  # divide by leading entry

**FOR** [k from 1 to i - 1]: # for the rows above the current one

# Back Substitution (Pseudocode)

**FUNCTION** back\_sub(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

**IF** [row i has a leading entry]:

j ← index of leading entry of row i

$R_i(A) \leftarrow R_i(A) / A[i, j]$  # divide by leading entry

**FOR** [k from 1 to i - 1]: # for the rows above the current one

$R_k(A) \leftarrow R_k(A) - R[k, j] * R_i(A)$

# zero out R[k, j] above the leading entry

# Back Substitution (Pseudocode)

**FUNCTION** back\_sub(A):

**FOR** [i from 1 to m]: # for each row from top to bottom

**IF** [row i has a leading entry]:

j ← index of leading entry of row i

$R_i(A) \leftarrow R_i(A) / A[i, j]$  # divide by leading entry

**FOR** [k from 1 to i - 1]: # for the rows above the current one

$R_k(A) \leftarrow R_k(A) - R[k, j] * R_i(A)$

# zero out R[k, j] above the leading entry

**RETURN** A

You will have to implement  
this part in HW2...

# Gaussian Elimination (Example)

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Gaussian Elimination (Example)

pivot  
position

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$



# Gaussian Elimination (Example)

pivot  
position

$$\begin{bmatrix} 3 & -9 & 12 & -9 & 6 & 15 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

$$R_1 \leftarrow R_1 / 3$$

# Gaussian Elimination (Example)

$$\begin{bmatrix} 1 & -3 & 4 & -3 & 2 & 5 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Gaussian Elimination (Example)

pivot  
position

$$\begin{bmatrix} 1 & -3 & 4 & -3 & 2 & 5 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Gaussian Elimination (Example)

pivot  
position

$$\begin{bmatrix} 1 & -3 & 4 & -3 & 2 & 5 \\ 0 & 2 & -4 & 4 & 2 & -6 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

$$R_2 \leftarrow R_2 / 2$$

# Gaussian Elimination (Example)

$$\begin{bmatrix} 1 & -3 & 4 & -3 & 2 & 5 \\ 0 & 1 & -2 & 2 & 1 & -3 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Gaussian Elimination (Example)

next entry  
to zero

$$\begin{bmatrix} 1 & -3 & 4 & -3 & 2 & 5 \\ 0 & 1 & -2 & 2 & 1 & -3 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Gaussian Elimination (Example)

next entry  
to zero

$$\begin{bmatrix} 1 & -3 & 4 & -3 & 2 & 5 \\ 0 & 1 & -2 & 2 & 1 & -3 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

$$R_1 \leftarrow R_1 + 3R_2$$

# Gaussian Elimination (Example)

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 5 & -4 \\ 0 & 1 & -2 & 2 & 1 & -3 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$



# Gaussian Elimination (Example)

pivot  
position

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 5 & -4 \\ 0 & 1 & -2 & 2 & 1 & -3 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Gaussian Elimination (Example)

pivot  
position

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 5 & -4 \\ 0 & 1 & -2 & 2 & 1 & -3 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

$$R_3 \leftarrow R_3 / 1$$

# Gaussian Elimination (Example)

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 5 & -4 \\ 0 & 1 & -2 & 2 & 1 & -3 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Gaussian Elimination (Example)

next entry  
to zero

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 5 & -4 \\ 0 & 1 & -2 & 2 & 1 & -3 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Gaussian Elimination (Example)

next entry  
to zero

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 5 & -4 \\ 0 & 1 & -2 & 2 & 1 & -3 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

$$R_2 \leftarrow R_2 - R_1$$

# Gaussian Elimination (Example)

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 5 & -4 \\ 0 & 1 & -2 & 2 & 0 & -7 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Gaussian Elimination (Example)

next entry  
to zero

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 5 & -4 \\ 0 & 1 & -2 & 2 & 0 & -7 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Gaussian Elimination (Example)

next entry  
to zero

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 5 & -4 \\ 0 & 1 & -2 & 2 & 0 & -7 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

$$R_1 \leftarrow R_1 - 5R_3$$



# Gaussian Elimination (Example)

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 0 & -24 \\ 0 & 1 & -2 & 2 & 0 & -7 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Gaussian Elimination (Example)

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 0 & -24 \\ 0 & 1 & -2 & 2 & 0 & -7 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

done with back substitution phase

# Question

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 0 & -24 \\ 0 & 1 & -2 & 2 & 0 & -7 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

*Write down the general form solution from the given RREF.*

# Solution

$$\begin{bmatrix} 1 & 0 & -2 & 3 & 0 & -24 \\ 0 & 1 & -2 & 2 & 0 & -7 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}$$

# Solution

$$x_1 = (-24) + 2x_3 - 3x_4$$

$$x_2 = (-7) + 2x_3 - 2x_4$$

$x_3$  is free

$x_4$  is free

$$x_5 = 4$$

# How-To: Solving a System of Linear Equations

# How-To: Solving a System of Linear Equations

1. Write your system as an augmented matrix

# How-To: Solving a System of Linear Equations

1. Write your system as an augmented matrix
2. Find the RREF of that matrix



# How-To: Solving a System of Linear Equations

1. Write your system as an augmented matrix
2. Find the RREF of that matrix
3. Read off the solution from the RREF

# How-To: Solving a System of Linear Equations

1. Write your system as an augmented matrix

2. Find the RREF of that matrix  
Gaussian elimination

3. Read off the solution from the RREF

# Numerics

demo  
(mini-GE)

# **Significant Figures (Sig Figs)**

# Significant Figures (Sig Figs)

*Have you ever been docked points in a science class for having incorrect sig figs?*

# Significant Figures (Sig Figs)

*Have you ever been docked points in a science class for having incorrect sig figs?*

when you use a ruler, you can't do better than  $\pm 1\text{mm}$ , so we can't say anything about nanometer differences

# Significant Figures (Sig Figs)

*Have you ever been docked points in a science class for having incorrect sig figs?*

when you use a ruler, you can't do better than  $\pm 1\text{mm}$ , so we can't say anything about nanometer differences

we run into a similar problem with decimal numbers  
in programs



# Number Representations

# Number Representations

your computer is a collection of fixed size registers

# Number Representations

your computer is a collection of fixed size registers

each register holds a sequence of bits

# Number Representations

your computer is a collection of fixed size registers

each register holds a sequence of bits

**The Goal.** represent numbers so they fit in those registers

# Number Representations

your computer is a collection of fixed size registers

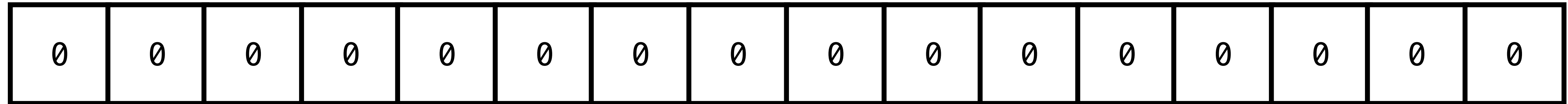
each register holds a sequence of bits

**The Goal.** represent numbers so they fit in those registers

this is, of course, ~~a lie~~ an abstraction

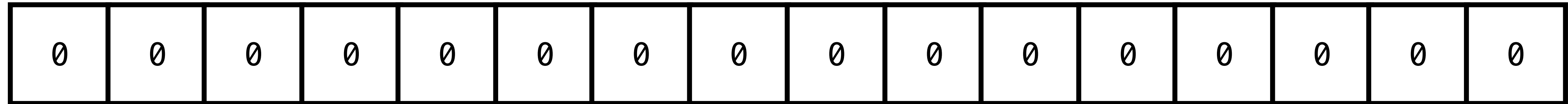


# Number Representations



**Question.** How do we slice up our fixed sequence to represent numbers?

# Number Representations



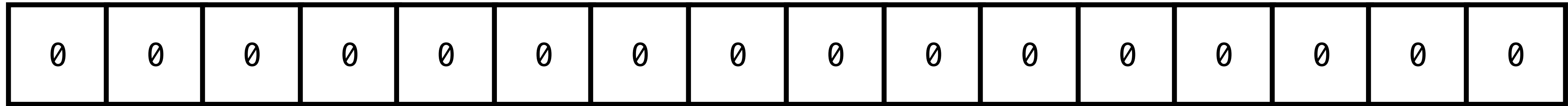
**Question.** How do we slice up our fixed sequence to represent numbers?

things to consider:

- simple idea (easy to understand)
- maximize coverage (not too redundant)
- simple numeric operations (easy to use)



# Unsigned Integers



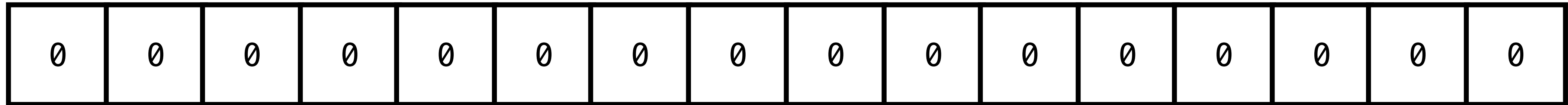
value

binary value (we should know this by now)

e.g. 10001010 represents

$$1(2^7) + 0(2^6) + 0(2^5) + 0(2^4) + 0(2^3) + 1(2^2) + 0(2^1) + 1(2^0)$$

# Signed Integers



sign value

sign bit + binary value

e.g. 10001010 represents

$$-1 \times (0(2^6) + 0(2^5) + 0(2^4) + 0(2^3) + 1(2^2) + 0(2^1) + 1(2^0))$$

# **Floating-Point Numbers (Some Figures)**

# Floating-Point Numbers (Some Figures)

floats in python use 64 bits

# Floating-Point Numbers (Some Figures)

floats in python use 64 bits

That's  $1.8 \times 10^{19}$  possible values

# Floating-Point Numbers (Some Figures)

floats in python use 64 bits

That's  $1.8 \times 10^{19}$  possible values

*We can't represent everything. We'll have to choose and then round*

# Floating-Point Numbers (Some Figures)

floats in python use 64 bits

That's  $1.8 \times 10^{19}$  possible values

*We can't represent everything. We'll have to choose and then round*

**Question.** Which ones should we represent?

# Floating-Point Numbers (An Idea)



# Floating-Point Numbers (An Idea)

Integers work because they are **discrete and evenly spaced**

# Floating-Point Numbers (An Idea)

Integers work because they are **discrete** and **evenly spaced**

**What if we evenly discretize a range of values?**

# Floating-Point Numbers (An Idea)

Integers work because they are **discrete and evenly spaced**

**What if we evenly discretize a range of values?**

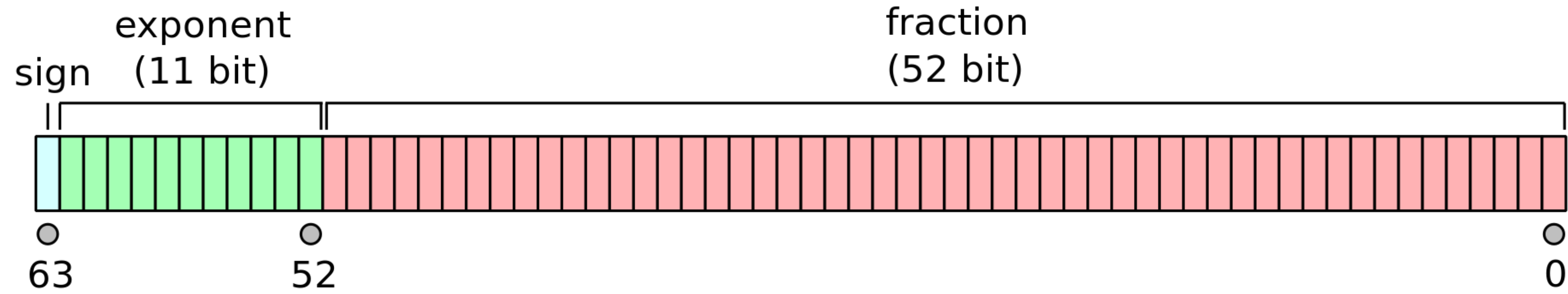
i.e., represent

**..., -0.001, 0, 0.0001, 0.002, 0.003, 0.004, ...**

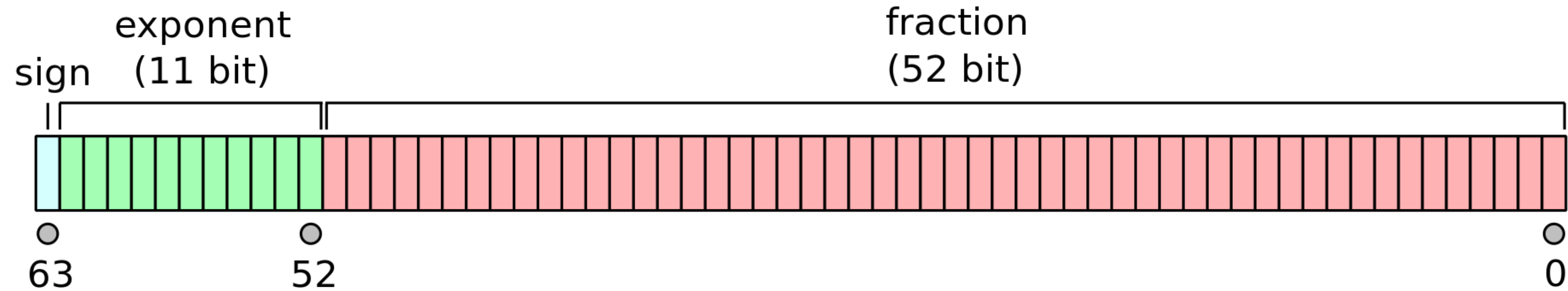
# Question

*Discuss the advantages and disadvantages of this approach*

# Floating-Point Numbers (IEEE-754)

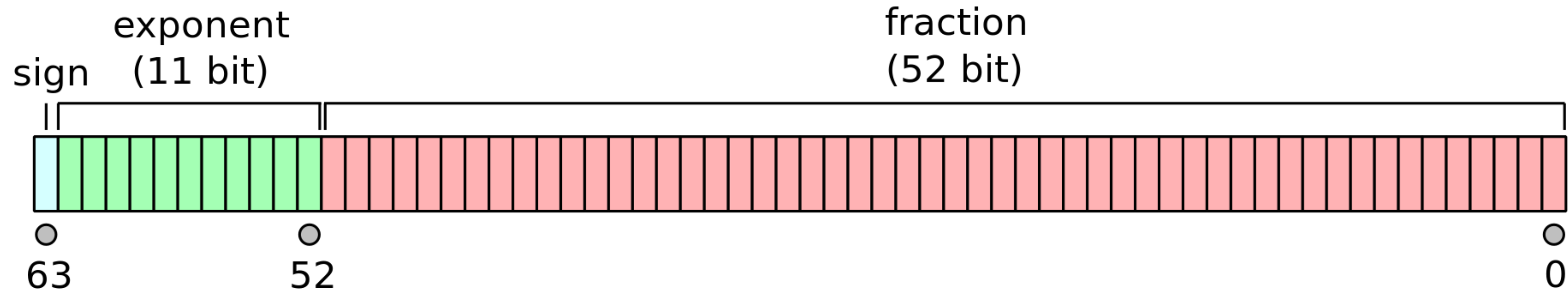


# Floating-Point Numbers (IEEE-754)



like scientific notation, but binary

# Floating-Point Numbers (IEEE-754)

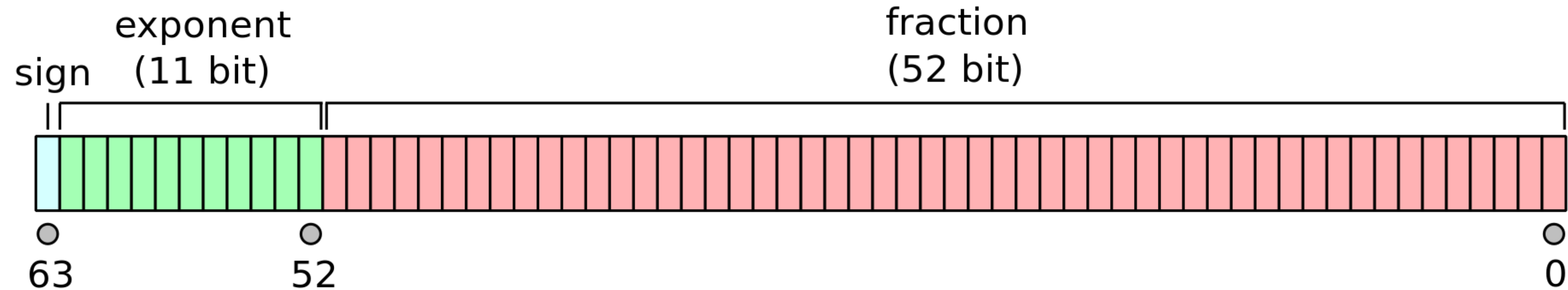


like scientific notation, but binary

the equation:

$$(-1)^{\text{sign}} \times \left( 1 + \frac{\text{fraction}}{2^{52}} \right) \times 2^{\text{exponent} - (2^{10} - 1)}$$

# Floating-Point Numbers (IEEE-754)



like scientific notation, but binary

the equation:

$$(-1)^{\text{sign}} \times \left( 1 + \frac{\text{fraction}}{2^{52}} \right) \times 2^{\text{exponent} - (2^{10} - 1)}$$

it's an accepted standard, not perfect, but it works well



# Question

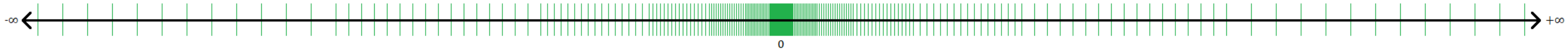
$$(-1)^{\text{sign}} \times \left( 1 + \frac{\text{fraction}}{2^{52}} \right) \times 2^{\text{exponent} - (2^{10} - 1)}$$

*Any ideas why this is better/worse?*

*And why not have a sign bit for the exponent?*

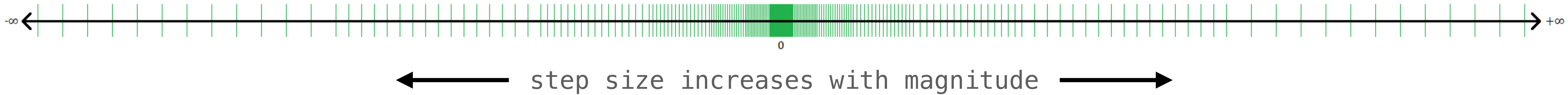
# Step Size

$$(-1)^{\text{sign}} \times \left( 1 + \frac{\text{fraction}}{2^{52}} \right) \times 2^{\text{exponent} - (2^{10} - 1)}$$



# Step Size

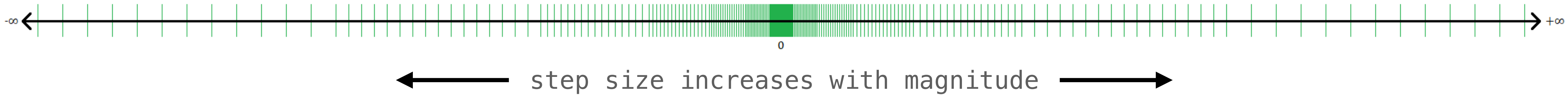
$$(-1)^{\text{sign}} \times \left( 1 + \frac{\text{fraction}}{2^{52}} \right) \times 2^{\text{exponent} - (2^{10} - 1)}$$



**Definition.** step size is the space between two floating-point representations

# Step Size

$$(-1)^{\text{sign}} \times \left( 1 + \frac{\text{fraction}}{2^{52}} \right) \times 2^{\text{exponent} - (2^{10} - 1)}$$



**Definition.** step size is the space between two floating-point representations

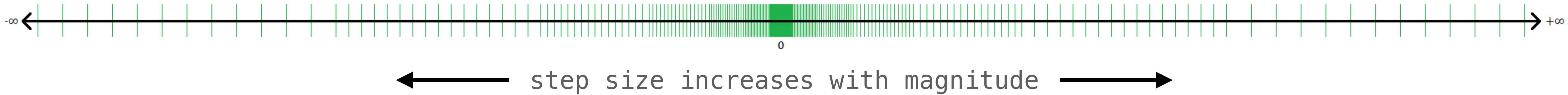
for fixed exponent  $n$  two numbers are at least

$$0.00\dots001 \times 2^n = 2^{-52} \times 2^n$$

away (why?)

# Step Size

$$(-1)^{\text{sign}} \times \left( 1 + \frac{\text{fraction}}{2^{52}} \right) \times 2^{\text{exponent} - (2^{10} - 1)}$$



**Definition.** step size is the space between two floating-point representations

for fixed exponent  $n$  two numbers are at least

$$0.00\dots001 \times 2^n = 2^{-52} \times 2^n$$

away (why?)

Step size doubles for each exponent

# Things to Keep in Mind

# Things to Keep in Mind

IEEE-754 defines a subset of decimal numbers

# Things to Keep in Mind

IEEE-754 defines a subset of decimal numbers

operations on floating point numbers attempt to give you the closest to the actual value, though there will be errors.



# Things to Keep in Mind

IEEE-754 defines a subset of decimal numbers

operations on floating point numbers attempt to give you the closest to the actual value, though there will be errors.

we can assume when we write down a number like '0.3' we get the closest IEEE-754 value

# Relative Error

**Observation.**  $\pm 0.001$  is *tiny* error for  $10^{20}$  but *massive* for  $10^{-20}$

# Relative Error

**Observation.**  $\pm 0.001$  is *tiny* error for  $10^{20}$  but *massive* for  $10^{-20}$

**Relative Error.**

$$\text{err}_{\text{rel}} = \frac{\text{err}}{\text{val}}$$

# Relative Error

**Observation.**  $\pm 0.001$  is *tiny* error for  $10^{20}$  but *massive* for  $10^{-20}$

**Relative Error.**

$$\text{err}_{\text{rel}} = \frac{\text{err}}{\text{val}}$$

IEEE-754 keeps relative error small

# Relative Error (Calculation)

$$(-1)^{\text{sign}} \times \left( 1 + \frac{\text{fraction}}{2^{52}} \right) \times 2^{\text{exponent} - (2^{10} - 1)}$$

(fix an exponent  $n$ )

# Relative Error (Calculation) $(-1)^{\text{sign}} \times \left(1 + \frac{\text{fraction}}{2^{52}}\right) \times 2^{\text{exponent} - (2^{10} - 1)}$

(fix an exponent  $n$ )

error is determined by step-size

$$\text{err} \leq 2^{-52} \times 2^n$$

# Relative Error (Calculation) $(-1)^{\text{sign}} \times \left(1 + \frac{\text{fraction}}{2^{52}}\right) \times 2^{\text{exponent} - (2^{10} - 1)}$

(fix an exponent  $n$ )

the smallest number we can represent at least  
 $1.0 \times 2^n$

$$\text{val} \geq 1.0 \times 2^n$$

(why do we care about a lower bound on val?)

# Relative Error (Calculation)

$$(-1)^{\text{sign}} \times \left( 1 + \frac{\text{fraction}}{2^{52}} \right) \times 2^{\text{exponent} - (2^{10} - 1)}$$

(fix an exponent  $n$ )



# Relative Error (Calculation)

$$(-1)^{\text{sign}} \times \left( 1 + \frac{\text{fraction}}{2^{52}} \right) \times 2^{\text{exponent} - (2^{10} - 1)}$$

(fix an exponent  $n$ )

the relative error is *small*

$$\text{val} \geq 1.0 \times 2^n$$

$$\text{err} \leq 2^{-52} \times 2^n$$

# Relative Error (Calculation)

$$(-1)^{\text{sign}} \times \left(1 + \frac{\text{fraction}}{2^{52}}\right) \times 2^{\text{exponent} - (2^{10} - 1)}$$

(fix an exponent  $n$ )

the relative error is *small*

$$\text{val} \geq 1.0 \times 2^n$$

$$\text{err} \leq 2^{-52} \times 2^n$$

$$\text{err}_{\text{rel}} = \frac{\text{err}}{\text{val}} \leq \frac{2^{-52} \times 2^n}{1.0 \times 2^n} = 2^{-52} \approx 10^{-16}$$

# Relative Error (Calculation)

$$(-1)^{\text{sign}} \times \left( 1 + \frac{\text{fraction}}{2^{52}} \right) \times 2^{\text{exponent} - (2^{10} - 1)}$$

(fix an exponent  $n$ )

the relative error is *small*

$$\text{val} \geq 1.0 \times 2^n$$

$$\text{err} \leq 2^{-52} \times 2^n$$

$$\text{err}_{\text{rel}} = \frac{\text{err}}{\text{val}} \leq \frac{2^{-52} \times 2^n}{1.0 \times 2^n} = 2^{-52} \approx 10^{-16}$$

$\approx 16$  digits of accuracy

Not bad, but also not great

demo

(example from the notes)

# The Takeaways

# The Takeaways

operations on floating-point numbers are not exact

# The Takeaways

operations on floating-point numbers are not exact

properties like  $(ab)c = a(bc)$  (associativity) may not hold



# The Takeaways

operations on floating-point numbers are not exact

properties like  $(ab)c = a(bc)$  (associativity) may not hold

it's a trade-off for large range and low relative error

# The Takeaways

operations on floating-point numbers are not exact

properties like  $(ab)c = a(bc)$  (associativity) may not hold

it's a trade-off for large range and low relative error

What do we do about it?

# Best Practices

1. don't compare floating points for equality
2. be aware of ill-conditioned problems
3. be aware of small differences

# Principle 1: Closeness

# Principle 1: Closeness

*When doing floating-point calculations in a program, define an error margin and use that for equality checking*

# Principle 1: Closeness

*When doing floating-point calculations in a program, define an error margin and use that for equality checking*

## **In Practice.**

Replace  
with

```
x == y  
numpy.isclose(x, y)
```

demo

# **Principle 2: Ill-Conditioned Problems**



# Principle 2: Ill-Conditioned Problems

*Make sure your problem is not sensitive to small errors.*

# Principle 2: Ill-Conditioned Problems

*Make sure your problem is not sensitive to small errors.*

**In Practice.** for example, don't divide by numbers much smaller than your error tolerance

demo

# **Principle 3: Small Differences**

# Principle 3: Small Differences

*Make sure you understand your error tolerance when looking that the small differences of large numbers.*

# Principle 3: Small Differences

*Make sure you understand your error tolerance when looking that the small differences of large numbers.*

**In Practice.** Don't expect  $a - b$  to be small when  $a$  and  $b$  are "close" but very large.

demo

# One Last Note: Special Numbers

`0` (we can't already represent 0?)

`nan` stands for not a number, .e.g, `sqrt(-2)`

`inf` symbolic infinity, behaves as expected



**NumPy**

# NumPy

# NumPy

NumPy is a library for doing linear algebra in Python.

# NumPy

NumPy is a library for doing linear algebra in Python.

Its **fast** and very widely used.

# NumPy

NumPy is a library for doing linear algebra in Python.

Its **fast** and very widely used.

**We will primarily be using numpy (and scipy) instead of sympy in this course.**

# NumPy vs. SymPy

NumPy is **fast**

NumPy is **approximate**

NumPy is **widely used in applications**

Sympy is **slow**

Sympy is **exact**

Sympy is a **teaching tool** (and useful in symbolic computation research)

# NumPy vs. SymPy

`numpy.array(...)`

`a[i] #row access`

`a[:,j] #col access`

`a.shape[0]`

`a.shape[1]`

`Matrix(...)`

`a[i,:] #row access`

`a[:,j] #col access`

`a.rows`

`a.cols`

demo



# **Extra Topic: Analyzing Gaussian Elimination**

# Analyzing the Algorithm

# Analyzing the Algorithm

We will not use  $O(\cdot)$  notation!

# Analyzing the Algorithm

We will not use  $O(\cdot)$  notation!

For numerics, we care about number of **F**loating-**O**perations (FLOPs):

- >> addition
- >> subtraction
- >> multiplication
- >> division
- >> square root

# Analyzing the Algorithm

We will not use  $O(\cdot)$  notation!

For numerics, we care about number of **F**loating-**O**perations (FLOPs):

- >> addition
- >> subtraction
- >> multiplication
- >> division
- >> square root

*2n vs. n is very different  
when  $n \sim 10^{20}$*

# Dominant Terms

# Dominant Terms

that said, we don't care about *exact* bounds

# Dominant Terms

that said, we don't care about *exact* bounds

A function  $f(n)$  is ***asymptotically equivalent*** to  $g(n)$  if

$$\lim_{i \rightarrow \infty} \frac{f(i)}{g(i)} = 1$$



# Dominant Terms

that said, we don't care about *exact* bounds

A function  $f(n)$  is ***asymptotically equivalent*** to  $g(n)$  if

$$\lim_{i \rightarrow \infty} \frac{f(i)}{g(i)} = 1$$

for polynomials, they are equivalent to their dominant term

# Dominant Terms

the dominant term of a polynomial is the monomial with the highest degree

$$\lim_{i \rightarrow \infty} \frac{3x^3 + 100000x^2}{3x^3} = 1$$

$3x^3$  dominates the function even though the coefficient for  $x^2$  is so large

# Parameters

$n$  : number of variables

$m$  : number of equations (we will assume  $m = n$ )

$n + 1$  : number of rows in the augmented matrix

# The Cost of a Row Operation

$$R_i \leftarrow R_i + aR_j$$

$n + 1$  multiplications for the scaling

$n + 1$  additions for the row additions

Tally:  $2(n + 1)$  FLOPS

# Cost of First Iteration of Elimination

$$R_2 \leftarrow R_2 + a_2 R_1$$

$$R_3 \leftarrow R_3 + a_3 R_1$$

⋮

$$R_n \leftarrow R_n + a_n R_1$$

repeated row operations for each row except the first

Tally:  $\approx 2n(n+1)$  FLOPS

# Rough Cost of Elimination

repeating this last process at most  $n$  times  
gives us a dominant term  $2n^3$

we can give a better estimation...

Tally:  $\approx 2n^2(n + 1)$  FLOPS

# Cost of Elimination

0	■	*	*	*	*	*	*	*	*
0	0	0	■	*	*	*	*	*	*
0	0	0	0	*	*	*	*	*	*
0	0	0	0	*	*	*	*	*	*
0	0	0	0	*	*	*	*	*	*
0	0	0	0	0	0	0	0	0	0

At iteration  $i$ , we're only interested in rows after  $i$

And to the right of column  $i$

# Cost of Elimination

$$\begin{array}{l} \text{Iteration 1: } 2n(n+1) \\ \text{Iteration 2: } 2(n-1)n \\ \text{Iteration 3: } 2(n-2)(n-1) \\ \quad \quad \quad \vdots \end{array} \quad +$$

---

$$\sum_{k=1}^n 2k(k+1) \approx \frac{2n(n+1)(2n+1)}{6} \sim (2/3)n^3$$

Tally:  $\sim (2/3)n^3$  FLOPS



# Cost of Back Substitution

(Let's assume no free variables)

for each pivot, we only need to:

>> zero out a position in 1 row (0 FLOPS)

>> add a value to the last row (1 FLOP)

**at most 1 FLOP per row per pivot  $\sim n^2$**

Tally:  $\sim (2/3)n^3$  FLOPS

# Cost of Gaussian Elimination

Tally:  $\sim (2/3)n^3$  FLOPS

(dominated by elimination)

# Summary

floating point numbers are represented in your computer

floating point operations are not exact

this can have unintended consequences

we get 16 digits of accuracy