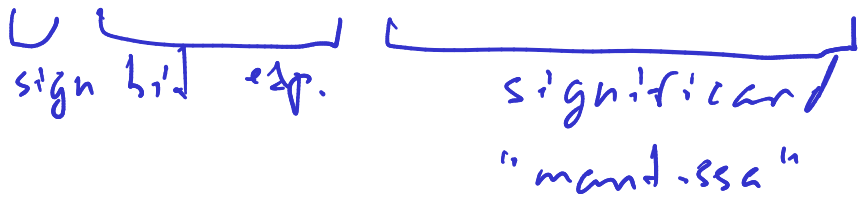


Floating Point



float: 8 exp, 23 sig

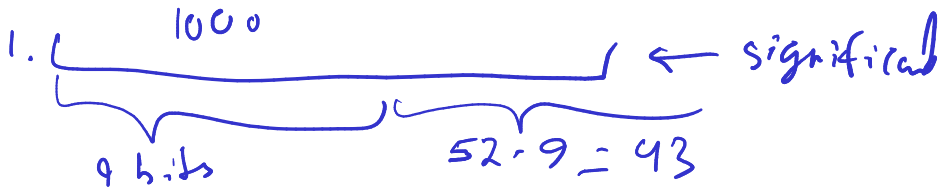
double: 11 exp, 52 sig

$$\underline{4 \cdot (2^{10})^5} = 4 \cdot 1024^5 \approx 4 \cdot 10^{15} \quad \left| \quad 2^{52} \approx 10^x \right.$$

$$.1 = \frac{1}{16} + \frac{1}{32} + \dots$$

$$.1 > \frac{3}{32}$$

$$1.\underbrace{0001} \cdot 10^3$$



exp	sign	stgnit.	type
0 ... 0	matters		subnormal
0 ... 0	+/-	000	
111111	+/-	000	infinity
111111		$\neq 0$	NaN

$$\frac{0}{0} = \text{NaN} = 0 \cdot \infty$$

$$x.yz \cdot 10^6 \cdot a.bc \cdot 10^d$$

$$\underbrace{(x.yz \cdot a.bc)} \cdot 10^{6+d}$$

$$|(\text{true answer} - x.yz \cdot a.bc)|$$

$$\leq 10^{-2} \cdot a.bc + 10^{-2} \cdot x.yz$$

$$\leq O(10^{-2})$$

Outline

Python, Numpy, and Matplotlib
Making Models with Polynomials
Making Models with Monte Carlo

Error, Accuracy and Convergence
Floating Point

Modeling the World with Arrays

The World in a Vector
What can Matrices Do?
Graphs
Sparsity

Norms and Errors
The 'Undo' Button for Linear Operations: LU

LU: Applications
Linear Algebra Applications
Interpolation

Repeating Linear Operations:
Eigenvalues and Steady States
Eigenvalues: Applications
Approximate Undo: SVD and Least Squares

SVD: Applications

Solving Funny-Shaped Linear Systems
Data Fitting
Norms and Condition Numbers
Low-Rank Approximation

Iteration and Convergence

Solving One Equation
Solving Many Equations
Finding the Best: Optimization in 1D

Optimization in n Dimensions

Outline

Python, Numpy, and Matplotlib
Making Models with Polynomials
Making Models with Monte Carlo

Error, Accuracy and Convergence
Floating Point

Modeling the World with Arrays

The World in a Vector

What can Matrices Do?

Graphs

Sparsity

Norms and Errors
The 'Undo' Button for Linear Operations: LU

LU: Applications

Linear Algebra Applications

Interpolation

Repeating Linear Operations:
Eigenvalues and Steady States

Eigenvalues: Applications

Approximate Undo: SVD and Least Squares

SVD: Applications

Solving Funny-Shaped Linear Systems

Data Fitting

Norms and Condition

Numbers

Low-Rank Approximation

Iteration and Convergence

Solving One Equation

Solving Many Equations

Finding the Best: Optimization in 1D

Optimization in n Dimensions

Some Perspective

- ▶ We have so far (mostly) looked at what we can do with single numbers (and functions that return single numbers).
- ▶ Things can get *much* more interesting once we allow not just one, but *many* numbers together.
- ▶ It is natural to view an *array of numbers* as one object with its own rules.
The simplest such set of rules is that of a **vector**.
- ▶ A 2D array of numbers can also be looked at as a **matrix**.
- ▶ So it's natural to use the tools of **computational linear algebra**.
- ▶ 'Vector' and 'matrix' are just *representations* that come to life in many (*many!*) applications. The purpose of this section is to explore some of those applications.

Vectors

What's a vector?

$$c = a + b$$

↑ ↑
————— vectors

$$c = \alpha \cdot a$$

↑
————— scalar

$$c = a \otimes b$$

$$c_i = a_i \cdot b_i$$

$$d = \sum_i a_i \cdot b_i \quad c_{ij} = a_i \cdot b_j$$

properties

$$a + b = c = a + (b_i)$$

$$a + b = b + a$$

$$\alpha(a + b)$$

$$= \alpha a + \alpha b$$

Vectors from a CS Perspective

What would the concept of a vector look like in a programming language (e.g. Java)?

interface vector

op add \rightarrow scalars

op mul

vector vadd(vector a, vector b)

vector vscale(scalar a, vector b)

Vectors in the 'Real World'

Demo: Images as Vectors

Demo: Sound as Vectors

Demo: Shapes as Vectors

Outline

Python, Numpy, and Matplotlib
Making Models with Polynomials
Making Models with Monte Carlo

Error, Accuracy and Convergence
Floating Point

Modeling the World with Arrays

The World in a Vector

What can Matrices Do?

Graphs

Sparsity

Norms and Errors
The 'Undo' Button for Linear Operations: LU

LU: Applications

Linear Algebra Applications

Interpolation

Repeating Linear Operations:
Eigenvalues and Steady States

Eigenvalues: Applications

Approximate Undo: SVD and Least Squares

SVD: Applications

Solving Funny-Shaped Linear Systems

Data Fitting

Norms and Condition

Numbers

Low-Rank Approximation

Iteration and Convergence

Solving One Equation

Solving Many Equations

Finding the Best: Optimization in 1D

Optimization in n Dimensions

Matrices

What does a matrix do?

It represents a *linear function* between two vector spaces $f : U \rightarrow V$ in terms of bases $\mathbf{u}_1, \dots, \mathbf{u}_n$ of U and $\mathbf{v}_1, \dots, \mathbf{v}_m$ of V . Let

$$\mathbf{u} = \alpha_1 \mathbf{u}_1 + \dots + \alpha_n \mathbf{u}_n$$

and

$$\mathbf{v} = \beta_1 \mathbf{v}_1 + \dots + \beta_m \mathbf{v}_m.$$



Then f can *always* be represented as a matrix that obtains the β s from the α s:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_m \end{pmatrix}.$$



Example: The 'Frequency Shift' Matrix

Assume both \mathbf{u} and \mathbf{v} are linear combination of sounds of different frequencies:

$$\mathbf{u} = \alpha_1 \mathbf{u}_{110 \text{ Hz}} + \alpha_2 \mathbf{u}_{220 \text{ Hz}} + \cdots + \alpha_4 \mathbf{u}_{880 \text{ Hz}}$$

(analogously for \mathbf{v} , but with β s). What matrix realizes a 'frequency doubling' of a signal represented this way?

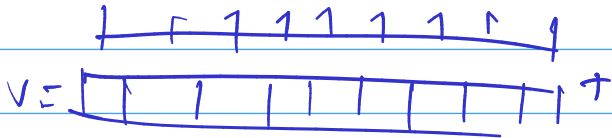
$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} 0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}$$

Example: The 'Frequency Shift' Matrix

Assume both \mathbf{u} and \mathbf{v} are linear combination of sounds of different frequencies:

$$\mathbf{u} = \alpha_1 \mathbf{u}_{110 \text{ Hz}} + \alpha_2 \mathbf{u}_{220 \text{ Hz}} + \cdots + \alpha_4 \mathbf{u}_{880 \text{ Hz}}$$

(analogously for \mathbf{v} , but with β s). What matrix realizes a 'frequency doubling' of a signal represented this way?



$$\begin{bmatrix} -1 & 0 & 1 & 0 & 0 & \dots \\ 0 & -1 & 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & -1 & 0 & 1 & \dots \end{bmatrix}$$

$$V = V[2:] - V[:-2]$$