



On Linear Algebra

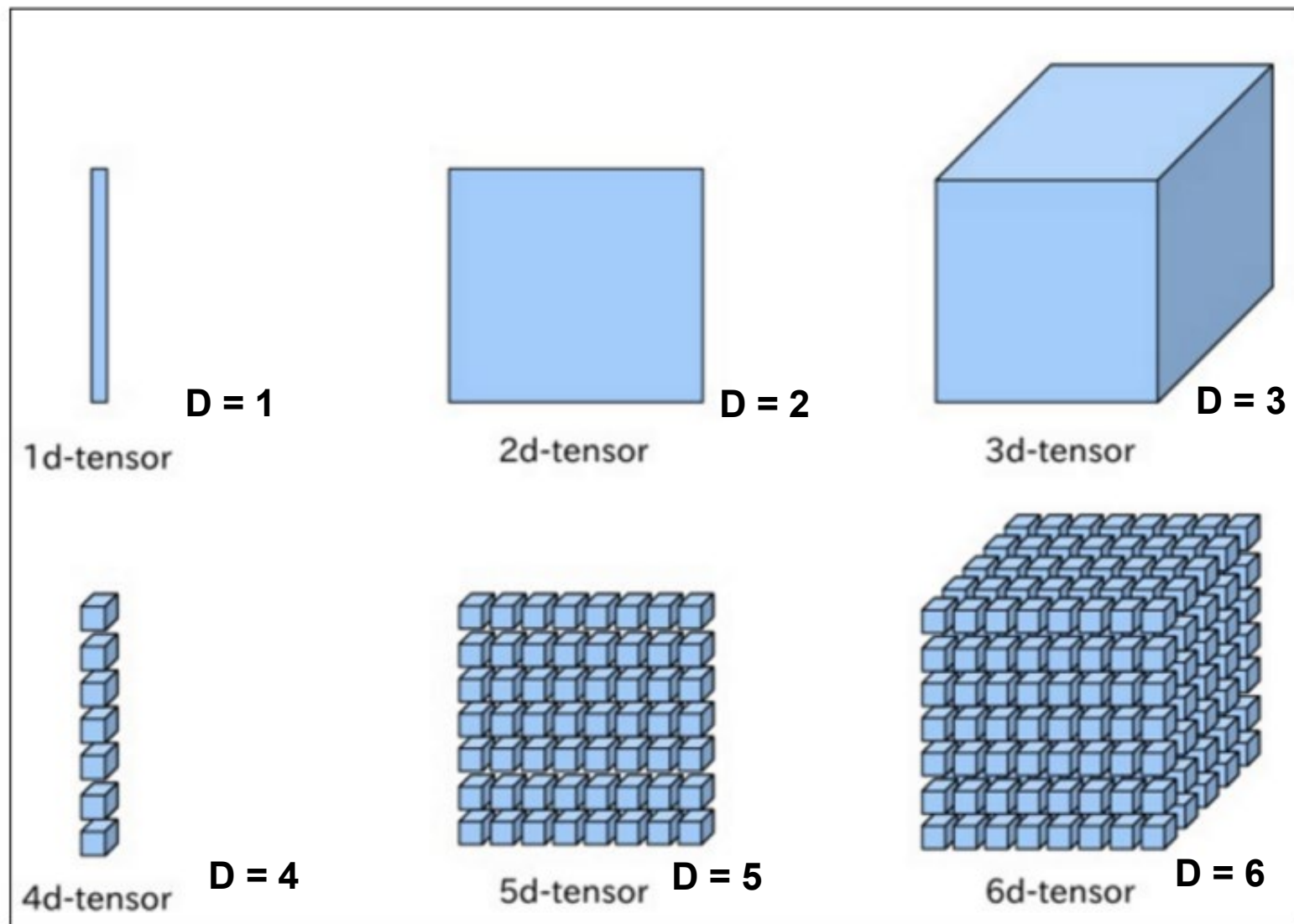
Some of the Very Basics

Alexander G. Ororbia II

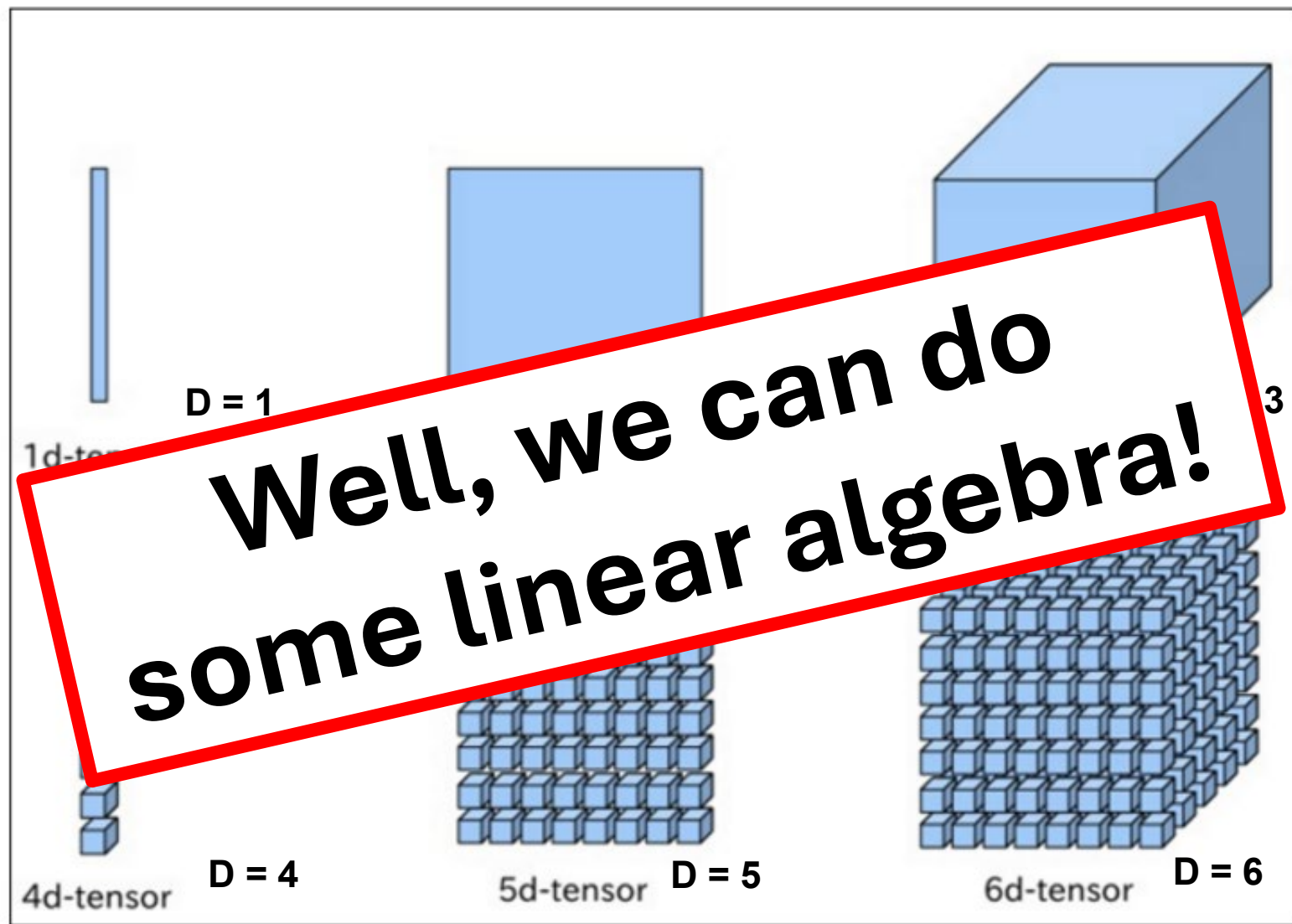
COGS-621: Foundations of Scientific Computing

9/11/2025

So we have tensors, what do we do with them?



So we have tensors, what do we do with them?



What is linear algebra?

- Linear algebra is the branch of mathematics concerning linear equations such as

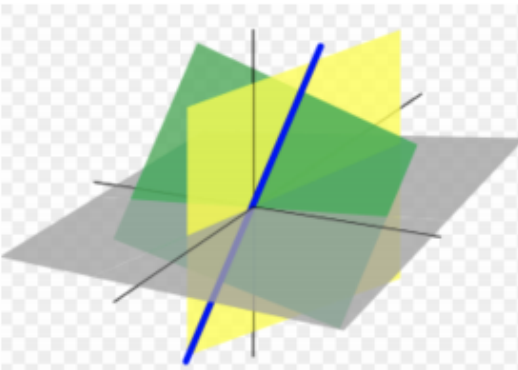
$$a_1x_1 + \dots + a_nx_n = b$$

- In vector notation we say $a^T x = b$
 - Called a linear transformation of x
- Linear algebra is fundamental to geometry, for defining objects such as lines, planes, rotations

For a larger subset/treatment:

Linear Algebra by Georgi E. Shilov

Also read: “**Linear Algebra for Dummies**”



Linear equation $a_1x_1 + \dots + a_nx_n = b$
defines a plane in (x_1, \dots, x_n) space
Straight lines define common solutions
to equations

Linear algebra is used
throughout engineering
(based on continuous math)

Transpose of a matrix

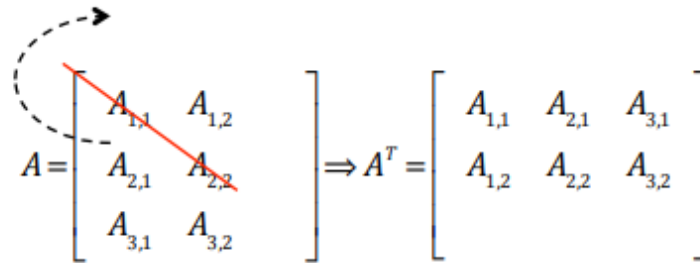
- An important operation on matrices
- The transpose of a matrix \mathbf{A} is denoted as \mathbf{A}^T
- Defined as

$$(\mathbf{A}^T)_{i,j} = A_{j,i}$$

– The mirror image across a diagonal line

- Called the main diagonal , running down to the right starting from upper left corner

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \\ A_{1,3} & A_{2,3} & A_{3,3} \end{bmatrix}$$


$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

Transposing/manipulation in NumPy

Table 2-12. Summary of NumPy Functions for Array Operations

Function	Description
<code>np.transpose,</code> <code>np.ndarray.transpose,</code> <code>np.ndarray.T</code>	Transpose (reverse axes) an array.
<code>np.fliplr/np.flipud</code>	Reverse the elements in each row/column.
<code>np.rot90</code>	Rotate the elements along the first two axes by 90 degrees.
<code>np.sort,</code> <code>np.ndarray.sort</code>	Sort an array's elements along a specified axis (which defaults to the last axis of the array). The <code>np.ndarray</code> method <code>sort</code> performs the sorting in place, modifying the input array.

Vector as a special case of the matrix

- Vectors are matrices with a single column
- Often written in-line using transpose

$$\mathbf{x} = [x_1, \dots, x_n]^T$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \Rightarrow \mathbf{x}^T = [x_1, x_2, \dots, x_n]$$

- A scalar is a matrix with one element

$$a = a^T \longleftarrow \mathcal{R}^{1 \times 1}$$

Matrix addition: an elementwise operation

- We can add matrices to each other if they have the same shape, by adding corresponding elements

– If A and B have same shape (height m , width n)

$$C = A + B \Rightarrow C_{i,j} = A_{i,j} + B_{i,j}$$

- A scalar can be added to a matrix or multiplied by a scalar

$$D = aB + c \Rightarrow D_{i,j} = aB_{i,j} + c$$

- Less conventional notation used in ML:

– Vector added to matrix $C = A + \mathbf{b} \Rightarrow C_{i,j} = A_{i,j} + b_j$

- Called broadcasting since vector \mathbf{b} added to each row of A

Matrix addition (and subtraction)

- Add / subtract operators follow basic properties of normal mathematical addition / subtraction
 - Matrix A + Matrix B is computed (element by element; ***elementwise***)

0.5	-0.7
-0.69	1.8

 +

0.5	0.7
-0.69	1.8


 =

.5 + .5 = 1.0	-.7 - .7 = -0.0
-.69 - .69 = -1.38	1.8 + 1.8 = 3.6

Hadamard product (elementwise multiplication)

- Multiply each $A(i, j)$ to each corresponding $B(i, j)$
 - Element-wise multiplication (similar in spirit to addition)
- Elementwise division follow same format

0.5	-0.7
-0.69	1.8



0.5	0.7
-0.69	1.8

 $=$

.5 * .5 = .25	-.7 * .7 = -.49
-.69 * -.69 = .4761	1.8 * 1.8 = 3.24

Elementwise operators

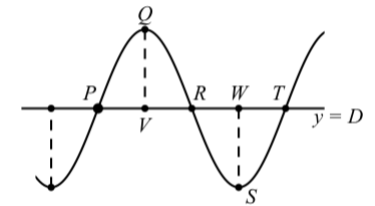
Table 2-6. Operators for Elementwise Arithmetic Operation on NumPy Arrays

Operator	Operation
<code>+, +=</code>	Addition
<code>-, -=</code>	Subtraction
<code>*, *=</code>	Multiplication
<code>/, /=</code>	Division
<code>//, //=</code>	Integer division
<code>**, **=</code>	Exponentiation

Elementwise functions (vectorized operators)

Table 2-7. Selection of NumPy Functions for Elementwise Elementary Mathematical Functions

NumPy Function	Description
<code>np.cos</code> , <code>np.sin</code> , <code>np.tan</code>	Trigonometric functions
<code>np.arccos</code> , <code>np.arcsin</code> , <code>np.arctan</code>	Inverse trigonometric functions
<code>np.cosh</code> , <code>np.sinh</code> , <code>np.tanh</code>	Hyperbolic trigonometric functions
<code>np.arccosh</code> , <code>np.arcsinh</code> , <code>np.arctanh</code>	Inverse hyperbolic trigonometric functions
<code>np.sqrt</code>	Square root
<code>np.exp</code>	Exponential
<code>np.log</code> , <code>np.log2</code> , <code>np.log10</code>	Logarithms of base e, 2, and 10, respectively



Sinusoid function

- NumPy contains many vectorized functions – elementwise evaluation of basic mathematical functions (over tensors)
 - Take in tensor/ndarray, return tensor/ndarray of same shape as input (though not necessarily of same type)

Table 2-8. Summary of NumPy Functions for Elementwise Mathematical Operations

NumPy Function	Description
<code>np.add</code> , <code>np.subtract</code> , <code>np.multiply</code> , <code>np.divide</code>	Addition, subtraction, multiplication, and division of two NumPy arrays
<code>np.power</code>	Raises first input argument to the power of the second input argument (applied elementwise)
<code>np.remainder</code>	The remainder of the division
<code>np.reciprocal</code>	The reciprocal (inverse) of each element
<code>np.real</code> , <code>np.imag</code> , <code>np.conj</code>	The real part, imaginary part, and the complex conjugate of the elements in the input arrays
<code>np.sign</code> , <code>np.abs</code>	The sign and the absolute value
<code>np.floor</code> , <code>np.ceil</code> , <code>np rint</code>	Converts to integer values
<code>np.round</code>	Rounds to a given number of decimals

- Elementwise functions can be single or multi-argument
- Can use the `np.vectorize` to convert some custom functions to those that operate on tensors/ndarrays

NumPy Statistics Functions

Table 2-9. NumPy Functions for Calculating Aggregates of NumPy Arrays

NumPy Function	Description
<code>np.mean</code>	The average of all values in the array
<code>np.std</code>	Standard deviation
<code>np.var</code>	Variance
<code>np.sum</code>	The sum of all elements
<code>np.prod</code>	The product of all elements
<code>np.cumsum</code>	The cumulative sum of all elements
<code>np.cumprod</code>	The cumulative product of all elements
<code>np.min, np.max</code>	The minimum/maximum value in an array
<code>np.argmin, np.argmax</code>	The index of the minimum/maximum value in an array
<code>np.all</code>	Returns True if all elements in the argument array are nonzero
<code>np.any</code>	Returns True if any of the elements in the argument array is nonzero

Σ = summation (capital sigma)

Π = product (capital pi)

- Can calculate certain statistics over tensors

Elementwise composed functions

We compose/create functions out of the basic/elemental elementwise functions we saw before!

- Applied to each element (i, j) of matrix/vector argument
 - *Can build from simple routines:*
 $\cos(\cdot)$, $\sin(\cdot)$, $\exp(\cdot)$, etc. (the “.” means argument)


- **Identity:** $\phi(\mathbf{v}) = \mathbf{v}$

- **Logistic Sigmoid:** $\phi(\mathbf{v}) = \sigma(\mathbf{v}) = \frac{1}{1+e^{-\mathbf{v}}}$

- **Softmax:** $\phi(\mathbf{v}) = \frac{\exp(\mathbf{v})}{\sum_{c=1}^C \exp(\mathbf{v}_c)}$

$$\mathbf{v} \in \mathbb{R}^C$$

- **Linear Rectifier:** $\phi(\mathbf{v}) = \max(0, \mathbf{v})$


$$\phi\left(\begin{array}{|c|c|}\hline 1.0 & -1.4 \\ \hline -0.69 & 1.8 \\ \hline\end{array}\right) = \begin{array}{|c|c|}\hline \phi(1.0) = 1 & \phi(-1.4) = 0 \\ \hline \phi(-0.69) = 0 & \phi(1.8) = 1.8 \\ \hline\end{array}$$

Tensor logical expression functions

Table 2-10. NumPy Functions for Conditional and Logical Expressions

Function	Description
<code>np.where</code>	Chooses values from two arrays depending on the value of a condition array
<code>np.choose</code>	Chooses values from a list of arrays depending on the values of a given index array
<code>np.select</code>	Chooses values from a list of arrays depending on a list of conditions
<code>np.nonzero</code>	Returns an array with indices of nonzero elements
<code>np.logical_and</code>	Performs an elementwise AND operation
<code>np.logical_or</code> , <code>np.logical_xor</code>	Elementwise OR/XOR operations
<code>np.logical_not</code>	Elementwise NOT operation (inverting)

Questions?

