On Linear Algebra

Some of the Very Basics

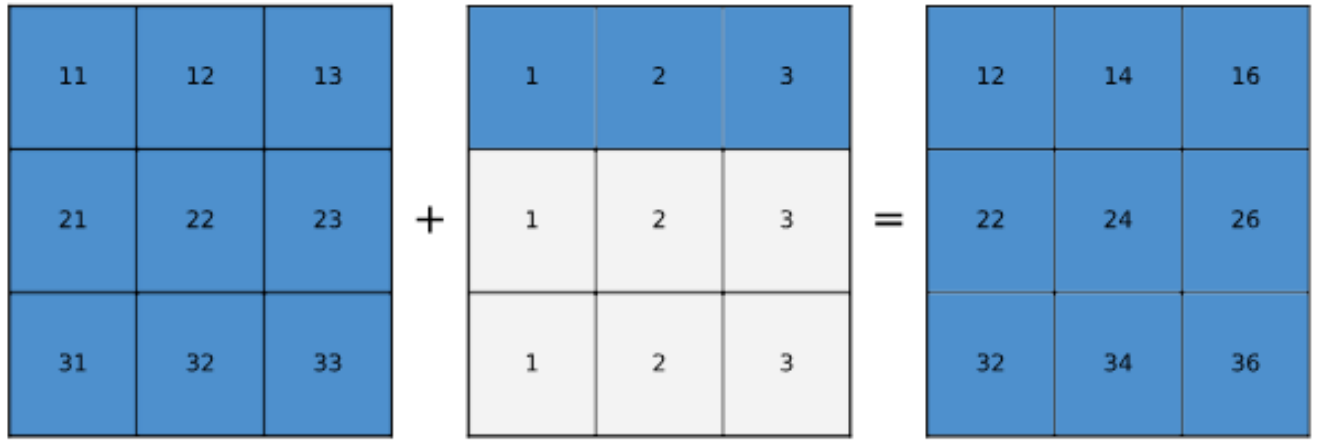
Alexander G. Ororbia II

COGS-621: Foundations of Scientific Computing

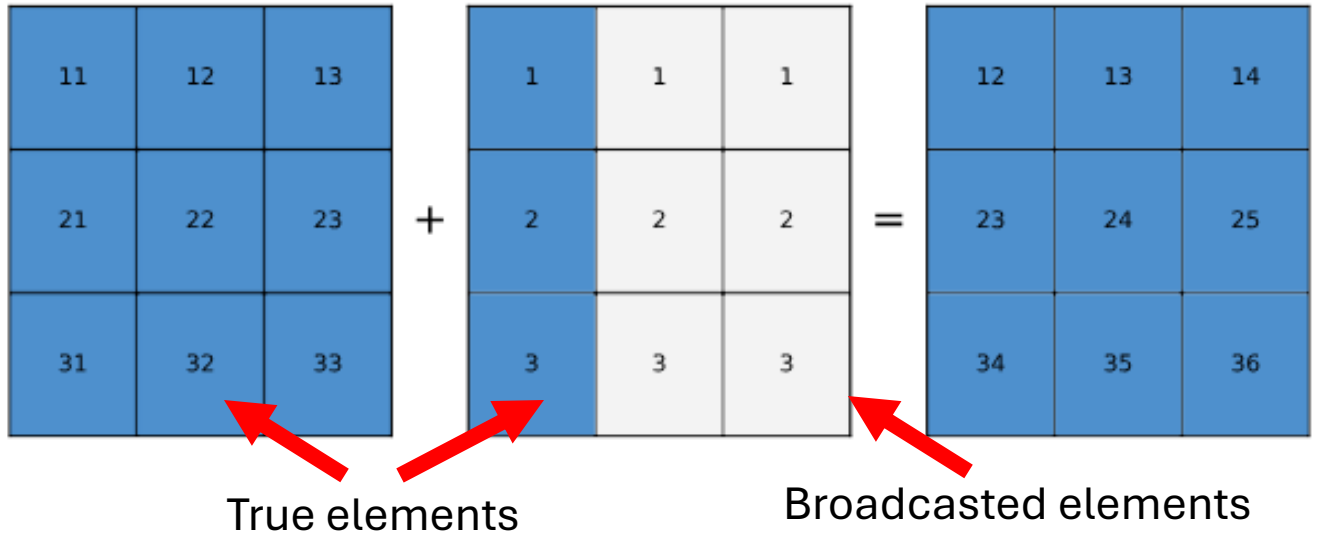
9/23/2025

Vectorization and broadcasting

Adding a 3x3 matrix to a 1x3 row vector:



Adding a 3x3 matrix to a 3x1 column vector:



- ***Vectorization***: store numerical data in arrays to use batch operations (applied to all values in array)
  - Avoids explicit loops
  - Makes code easier to maintain / more concise, better performance
  - Binary op well-defined if two arguments ae same shape

Broadcasting: effective array expansion

- Array + scalar – scalar distributed (& op applied) to each element in array
- Unequal shape arrays – if smaller array can be broadcasted to larger array (***rule***: if array axes on 1-on-1 basis have same length or length of 1)
  - If an array has fewer axes, dummy axes are padded on until dimensions of arrays agree

Elementwise / broadcasted mathematics

- **Addition** “+” (+)

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

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

- **Subtraction** “-” (-)
- **Multiplication** (Hadamard product) “*” ($\odot$)

0.5	-0.7
-0.69	1.8

$\odot$

0.5	0.7
-0.69	1.8

=

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

- **Division** “/” (/)

Elementwise arithmetic

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

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

- Arithmetic result => yields new independent array w/ own memory (complex expressions can trigger memory allocation/copy ops)
- In-place operations:
  - x = x + y (more readable) ***versus*** x += y (reduced memory footprint / updates x in-place)

Tensor logical expressions

- ***Standard base logical ops***: <, >, <=, >=, ==, !=

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

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

NumPy aggregation functions (aggregators)

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

NumPy Function	Description
`np.sum`	The sum of all elements
`np.prod`	The product of all elements
`np.min, np.max`	The minimum/maximum value in an array
`np.argmin, np.argmax`	The index of the minimum/maximum value in an array

Σ = summation (capital sigma)

Π = product (capital pi)

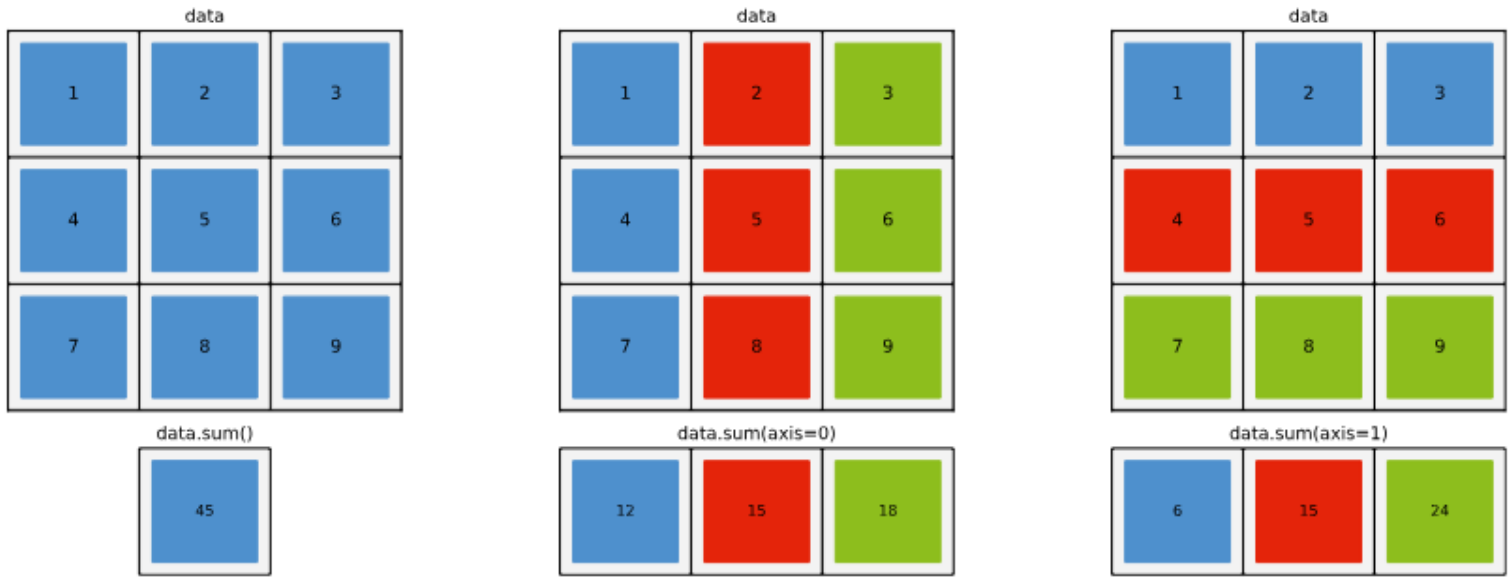


Figure 2-3. *Illustration of array aggregation functions along all axes (left), the first axis (center), and the second axis (right) of a two-dimensional array of shape 3 × 3*

Elementwise composed functions

- *Can build from simple routines:*
  cos(.), *sin(.)*, *exp(.)*, etc. (the "." means argument)

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

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

Let's write these out to our Python interpreter!

NumPy Function
np.cos, np.sin, np.tan
np.arccos, np.arcsin, np.arctan
np.cosh, np.sinh, np.tanh
np.arccosh, np.arcsinh, np.arctanh
np.sqrt
np.exp
np.log, np.log2, np.log10

NumPy Function
np.add, np.subtract, np.multiply, np.divide
np.power
np.remainder
np.reciprocal
np.real, np.imag, np.conj
np.sign, np.abs
np.floor, np.ceil, np.rint
np.round

Questions?