EBA3500 Fall 2021
Lecture 1: Matrices and Python

Jonas Moss

August 26, 2021

Recall the definition of an $m \times n$ matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- The matrix A has elements a_{ij}.
- These are written in lower case letters for some reason; I didn't choose this, it's the convention!
- Sometimes we describe matrices using their elements only.
- Remember that m is the number of rows and n the number of columns, $r \times c$. It's RC for RC cars or "remote controller".

We will use numpy matrices.

- The key module is called `linalg`.
- See https://numpy.org/doc/stable/reference/routines.linalg.html.
- For instance `A = np.array([[1, 0, 1], [0, 1, 0], [1, 0, 1]])`
- Then $A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$.

Four operations:

- Addition
- Multiplication
- Inversion (the matrix variant of division!)
- Transpose

All of these are important!

- Wikipedia is an excellent resource for mathematics facts.
- Sometimes difficult, but you should be able to read it anyway.
- Plenty of free linear algebra resources online too.
- An example is http://linear.ups.edu/html/fcla.html
- You will probably have to study and restudy linear algebra during your career, as it's the foundation of most data science.

Matrix addition and scalar multiplication

Matrix addition

Two $m \times n$ matrices A and B can be added to each other element-wise, where $(a+b)_{ij} = a_{ij} + b_{ij}$, i.e.,

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{21}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{11} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

This is just like elementwise addition of Numpy arrays, which we have already covered.

Example

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 3 \\ 2 & 2 & 3 \\ 4 & 2 & 2 \end{bmatrix}$$

```
>>> A = np.array([[1,0,1], [0,1,0], [1,0,1]])
>>> B = np.array([[1,2,2], [2,1,3], [3,2,1]])
>>> A + B
array([[2, 2, 3],
       [2, 2, 3],
       [4, 2, 2]])
```

Minus is defined in the same way.

Scalar multiplication

In $c \in \mathbb{R}$ is a number, then cA has elements ca_{ij}.

$$cA = \begin{bmatrix} ca_{11} & ca_{12} & \cdots & ca_{1n} \\ ca_{21} & ca_{22} & \cdots & ca_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ ca_{m1} & ca_{m2} & \cdots & ca_{mn} \end{bmatrix}$$

This is just like vectorized multiplication in Numpy.

Example

$$2\begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 4 \\ 4 & 2 & 6 \\ 6 & 4 & 2 \end{bmatrix}$$

```
>>> B = np.array([[1,2,2], [2,1,3], [3,2,1]])
>>> 2 * B
array([[2, 4, 4],
       [4, 2, 6],
       [6, 4, 2]])
```

Matrix multiplication

Matrix multiplication is *not* like vectorized multiplication of arrays, but something else entirely.

The matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Can be written in *column form*:

$$\begin{aligned} A &= \begin{bmatrix} A_{\cdot 1} & A_{\cdot 2} & \ldots & A_{\cdot n} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \ldots & \boldsymbol{a}_n \end{bmatrix} \end{aligned}$$

Here

$$\boldsymbol{a}_j = \begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{bmatrix}$$

is the jth column of A.

Multiplication

Let A be an $m \times n$ and $\boldsymbol{x}$ be a column.

$$Ax = \boldsymbol{a}_1 x_1 + \boldsymbol{a}_2 x_2 + \cdots + \boldsymbol{a}_n x_n$$

A *linear combination* of the columns of A.

Example

$$
\begin{aligned}
\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} &= \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \cdot 1 + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \cdot 2 + \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \cdot 3 \\
&= \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix} + \begin{bmatrix} 3 \\ 0 \\ 3 \end{bmatrix} \\
&= \begin{bmatrix} 4 \\ 2 \\ 4 \end{bmatrix}
\end{aligned}
$$

Explanation

Matrices are used to represent linear equations! Matrix multiplication ensures that

$$
Ax = \begin{bmatrix} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n \\ \vdots & & \vdots & & \ddots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_1 & + & \cdots & + & a_{mn}x_n \end{bmatrix}
$$

For instance,

$$
\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 + x_3 \\ x_2 \\ x_1 + x_3 \end{bmatrix}
$$

Extension to matrices

Multiplying two matrices is like having a bunch of matrix equations at the same time!

If $B \in \mathbb{R}^{n \times k}$

$$
\begin{aligned}
AB &= A \left[\begin{array}{cccc} \boldsymbol{b}_1 & \boldsymbol{b}_2 & \cdots & \boldsymbol{b}_k \end{array} \right], \\
&= \left[\begin{array}{cccc} A\boldsymbol{b}_1 & A\boldsymbol{b}_2 & \cdots & \boldsymbol{A}\boldsymbol{b}_k \end{array} \right].
\end{aligned}
$$

For example,

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

- This definition of multiplication equivalent to the definition you probably learned in your math course.
- The definiton you learned is easier to use when calculating by hand, but not for understanding and theory.
- To multiply to numpy arrays using matrix multiplication, write `A@B`.

```
>>> A = np.array([[1,0,1], [0,1,0], [1,0,1]])
>>> B = np.array([[1,2,2], [2,1,3], [3,2,1]])
>>> A@B
array([[4, 4, 3],
       [2, 1, 3],
       [4, 4, 3]])
```

Matrix inverse and matrix equations

A matrix equation is on the form

$$Ax = b.$$

May also be called a *linear equation* or *system of linear equations*.

From the definition of matrix multiplication, this means that

$$\boldsymbol{a}_1 x_1 + \boldsymbol{a}_2 x_2 + \cdots + \boldsymbol{a}_n x_n = b.$$

In other words, there is a linear combination of the columns of A that add up b!

Example

$$\begin{aligned} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} &= \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \\ x_1 + x_3 &= b_1 \\ x_2 &= b_2 \\ x_1 + x_3 &= b_3 \end{aligned}$$

Three essential facts about linear equations

A system of linear equations $Ax = b$ has either:

1. One solution.
2. Infinitely many solutions.
3. No solution at all.

Example

$$\begin{aligned} x_1 + x_3 &= b_1 \\ x_2 &= b_2 \\ x_1 + x_3 &= b_3 \end{aligned}$$

- When $b_1 \neq b_3$, this system of equations has no solution, as $x_1 + x_3 = x_1 + x_3$.

- If $b_1 = b_3$, it has infinitely many solutions. This happens because $x_1 = b_1 - x_3$ is the equation for a line!
- To solve the system $Ax = b$ in Python, use

$$\texttt{np.linalg.solve}(\texttt{A}, \texttt{b})$$

- If $Ax = b$ has a unique solution for every b, then A is *invertible.*
- In this case there is a matrix A^{-1} so that $x = A^{-1}b$, which is unique.
- To find the inverse of a matrix in Python, use

$$\texttt{np.linalg.inv}(\texttt{A})$$

```
>>> B = np.array([[1,2,2], [2,1,3], [3,2,1]])
>>> np.linalg.inv(B)
array([[-0.45454545,  0.18181818,  0.36363636],
       [ 0.63636364, -0.45454545,  0.09090909],
       [ 0.09090909,  0.36363636, -0.27272727]])
```

Matrix inverse

The $n \times n$ identity matrix is the unique matrix satisfying

$$AI = A$$

And equals

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \cdots & \boldsymbol{e}_k \end{bmatrix}$$

where $\boldsymbol{e}_k$ is the kth unit vector. For instance

$$\boldsymbol{e}_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}.$$

Using the definition of matrix multiplication, we can verify that

$$A\boldsymbol{e}_i = \boldsymbol{a}_i.$$

Proof

$$\begin{aligned} AI &= A \begin{bmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \cdots & \boldsymbol{e}_k \end{bmatrix} \\ &= \begin{bmatrix} A\ \boldsymbol{e}_1 & A\boldsymbol{e}_2 & \cdots A & \boldsymbol{e}_k \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_k \end{bmatrix} \end{aligned}$$

Inversion facts

- A square matrix $A \in \mathbb{R}^{n\times n}$ is invertible (non-singular) if there is a matrix A^{-1} so that $AA^{-1} = A^{-1}A = I$. The *inverse* A^{-1} is unique if it exists.
- Not every matrix has an inverse!
- Regarding addition and inversion: $(A+B)^{-1} \neq A^{-1} + B^{-1}$
  - Why? If a, b are numbers, then $1/(a+b) \neq 1/a + 1/b = (a+b)/(ab)$!

Basic facts about inversion:

- If $c \neq 0$ and A is invertible, then $(cA)^{-1} = \frac{1}{c}A^{-1}$.
- If A, B are invertible, then AB is invertible, and $(AB)^{-1} = B^{-1}A^{-1}$.

Gaining a solid understanding of when matrices are invertible and not is important, but beyond the scope of this lecture. But there are three ways to check if a matrix is invertible in Python:

- Is its determinant different from 0? The determinant is the signed volume of the parallelotope defined by the matrix – think of it as the "volume of the matrix".
  - `np.linalg.det(B)`
- Find its eigenvalues. The matrix is invertible if all eigenvalues are different from 0. A matrix is *numerically singular* (non-invertible) if its smallest eigenvalue is really close to 0. Then it's hard to work with!
  - `np.linalg.eigvals(A)`
- Just try to invert it!
  - `np.linalg.inv(A)`

```
>>> B = np.array([[1,2,2], [2,1,3], [3,2,1]])
>>> np.linalg.det(B)
11.000000000000002
```

Remember: Wikipedia is an excellent resource! https://en.wikipedia.org/wiki/Invertible_matrix

Matrix transpose

The transpose A^T is the matrix flipped along the diagonal.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}.$$

Example

$$\begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix}^T = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix}$$

Transposition facts

To transpose matrices in Python, use either:

- `np.transpose(A)`
- `A.T`

```
>>> B = np.array([[1,2,2], [2,1,3], [3,2,1]])
>>> B.T
array([[1, 2, 3],
       [2, 1, 2],
       [2, 3, 1]])
```

Three basic transposition facts:

- Addition and transposes: $(A+B)^T = A^T + B^T$
- Multiplication and transposes: $(AB)^T = B^T A^T$
- Connection with transposition: $(A^{-1})^T = (A^T)^{-1}$

Element-wise multiplication

Also known as *Hadamard multiplication.*

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1n}b_{1n} \\ a_{21}b_{21} & a_{21}b_{22} & \cdots & a_{2n}b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{11} & a_{m2}b_{m2} & \cdots & a_{mn}b_{mn} \end{bmatrix}$$

Not that useful for matrices, but absolutely useful in Python practice!

```
>>> A = np.array([[1,0,1],
                  [0,1,0],
                  [1,0,1]])
>>> B = np.array([[1,2,2],
                   [2,1,3],
                   [3,2,1]])
>>> A * B
array([[1, 0, 2],
       [0, 1, 0],
       [3, 0, 1]])
```

The dot product

- The dot product between to vectors in $\mathbb{R}^n$ is $x \cdot y = x_1 y_1 + \ldots + x_n y_n$.
- Then $x \cdot y = x^T y$.
- May use `numpy.dot(x,y)` or `x.dot(y)`.

```
>>> x = np.array([2,2,1])
>>> y = np.array([1,3,3])
>>> x.dot(y) # 2 * 1 + 2 * 3 + 1 * 3
11
>>> x.T @ y
11
>>> x @ y # Works because Python is kind.
11
```