BOOK: Linear Algebra: Theory, Intuition, Code

AUTHOR: Mike X Cohen

WEBSITE: sincxpress.com

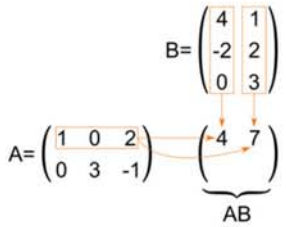
CHAPTER: Matrix multiplications (chapter 6)

In [2]:
```
## import libraries for the entire chapter
import numpy as np
```

Section 6.1, code block 6.1 -- "tandard" matrix multiplication

MOLTIPLICAZIONE "Standard" delle Matrici 6.1

Indichiamo con AB = moltiplicazione standard delle Matrici



In [7]:
```
import numpy as np
# two matrices
#M1 = np.random.randn(4,3)
M1 = np.random.randint(5,size=(2,3))
#M2 = np.random.randn(3,5)
M2 = np.random.randint(5,size=(3,4))
print(M1)
print(' ')
print(M2)
print(' ')

# and their product
C = M1 @ M2
print(C)
```

```
[[2 2 2]
 [1 2 0]]

[[3 3 2 4]
 [3 0 1 2]
 [0 1 0 4]]

[[12  8  6 20]
 [ 9  3  4  8]]
```

TERMINOLOGIA

Il prodotto non è commutativo $\mathbb{AB} \neq \mathbb{BA}$

ECCEZIONE: $\mathbb{AI} = \mathbb{IA}$

$$
\begin{aligned}
&\text{"}\mathbf{A}\text{ times }\mathbf{B}\text{"}\\
&\text{"}\mathbf{A}\text{ left-multiplies }\mathbf{B}\text{"}\\
&\text{"}\mathbf{A}\text{ pre-multiplies }\mathbf{B}\text{"}\\
&\text{"}\mathbf{B}\text{ right-multiplies }\mathbf{A}\text{"}\\
&\text{"}\mathbf{B}\text{ post-multiplies }\mathbf{A}\text{"}
\end{aligned}
$$

Section 6.2, code block 6.3 -- Non-commutivity of matrix multiplication

In [15]:
```
#A = np.random.randn(2,2)
#B = np.random.randn(2,2)
A = np.random.randint(5,size=(2,2))
B = np.random.randint(5,size=(2,2))
# notice that C1 != C2
C1 = A@B
C2 = B@A
print(A,' = A')
print(' ')
print(B,' = B')
print(' ')
print(C1,' = AB')
print(' ')
print(C2,' = BA')
```

```
[[1 1]
 [3 4]]  = A

[[1 2]
 [4 3]]  = B

[[ 5  5]
 [19 18]]  = AB

[[ 7  9]
 [13 16]]  = BA
```

Sesction 6.3, Matrix multiplication with a diagonal matrix

MOLTIPLICAZIONI e EQUAZIONI

MOLTIPLICAZIONE di una MATRICE con una MATRICE DIAGONALE

$$\begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1a & 2a & 3a \\ 4b & 5b & 6b \\ 7c & 8c & 9c \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} = \begin{bmatrix} 1a & 2b & 3c \\ 4a & 5b & 6c \\ 7a & 8b & 9c \end{bmatrix}$$

$$\begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} d & 0 & 0 \\ 0 & e & 0 \\ 0 & 0 & f \end{bmatrix} = \begin{bmatrix} ad & 0 & 0 \\ 0 & be & 0 \\ 0 & 0 & cf \end{bmatrix}$$

The LIVE EVIL rule: Reverse matrix order

$$(\mathbf{A} \dots \mathbf{B})^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}} \dots \mathbf{A}^{\mathrm{T}} \tag{6.7}$$

Section 6.4, LIE EVIL! (a.k.a. order of operations)

LIVE-EVIL si applica anche all'inversa di una matrice

Section 6.5, Matrix-vector multiplication

PRODOTTO MATRICE-VETTORE

Section 6.8, code block 6.5

Symmetric matrix times a vector

$$if\ \mathbf{A} = \mathbf{A}^{\mathrm{T}}\ then\ \mathbf{Ab} = (\mathbf{b}^{\mathrm{T}}\mathbf{A})^{\mathrm{T}} \tag{6.8}$$

In [ ]:
```
M1 = np.random.randn(4,3)
M2 = np.random.randn(4,3)
C = M1 * M2 # note the * instead of @
```

Section 6.6 -- Creating symmetric matrices

$$\mathbf{C} = \frac{1}{2}(\mathbf{A}^{\mathrm{T}} + \mathbf{A}) \tag{6.9}$$

Section 6.7, Multiplication of two symmetric matrices

in general it is not a symmetric operation

Section 6.8, Element-wise (Hadamard) multiplication

$$\mathbf{C} = \mathbf{A} \odot \mathbf{B} \tag{6.19}$$

The formal definition of Hadamard multiplication is

Hadamard (element-wise) multiplication

$$c_{i,j} = a_{i,j} \times b_{i,j} \tag{6.20}$$

Section 6.9, code block 6.7 -- Creating symmetric matrices

In [17]:
```
A = np.array([ [1,2,3],[4,5,6] ])
B = A.flatten(order='F')
print(A)
print(' ')
print(B)
```

```
[[1 2 3]
 [4 5 6]]

[1 4 2 5 3 6]
```

Section 6.9, code block 6.9 -- Frobenius dot product

FROBENIUS PRODUCT

Frobenius dot product as the trace of $\mathbf{A}^{\mathrm{T}}\mathbf{B}$

$$\langle \mathbf{A}, \mathbf{B} \rangle_F = tr(\mathbf{A}^{\mathrm{T}}\mathbf{B}) \tag{6.23}$$

The code below shows the trace-transpose trick for computing the Frobenius product

Frobenius matrix norm

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{i,j})^2} \tag{6.24}$$

Euclidean distance between two matrices

$$\|\mathbf{A} - \mathbf{B}\|_F = \sqrt{\sum_{i,j}(a_{i,j} - b_{i,j})^2} \tag{6.25}$$

In [3]:
```
A = np.random.randint(5,size=(4,3))
B = np.random.randint(5,size=(4,3))

# the transpose-trace trick for the Frobenius dot product
C = A.T@B
f = np.trace(C)
print(A)
print(' ')
print(B)
print(' ')
print(C)
print(' ')
print(f)
```

```
[[4 1 2]
 [3 4 0]
 [1 4 0]
 [4 2 2]]

[[1 4 0]
 [1 1 0]
 [1 1 0]
 [2 1 1]]

[[16 24  4]
 [13 14  2]
 [ 6 10  2]]

32
```

Section 6.10, Matrix norms

Section 6.10, Codeblock 6.11 -- Frobenius norm

In [5]:
```
A = np.random.randint(5,size=(4,3))
B = np.linalg.norm(A,'fro')
# questa e la somma dei qudrati di ogni elemento, quindi la loro radice quadrata
print(A)
print(' ')
print(B)
```

```
[[4 3 2]
 [3 4 0]
 [0 3 1]
 [1 0 2]]

8.306623862918075
```

Section 6.11, From a square matrix A, we get a skew-symmetrix matrix K, and the symmetric one L

In [41]:
```
# MATRICE GENERICA
A1 = ([[1, 2, 3],[4, 5, 6],[7, 8, 9]])
A = Aa = np.array(A1)
B = A.transpose()
K = (A-B)/2
#print(K)
L = (A-K)
#print(L)
K_frobenius_norm = pow(np.linalg.norm(K),2)
A_frobenius_norm = pow(np.linalg.norm(A),2)
A_sigma = int(K_frobenius_norm/A_frobenius_norm*1000)/1000
print('Asymmetry index of a generic matrix A = ',A_sigma)

A1 = K
B1 = K.transpose()
K1 = (A1-B1)/2
K1_frobenius_norm = pow(np.linalg.norm(K1),2)
A1_frobenius_norm = pow(np.linalg.norm(A1),2)
A1_sigma = K1_frobenius_norm/A1_frobenius_norm
#print(K1)
print('Asymmetry index of a skew-matrix A1 = ',A1_sigma)
```

```
A2 = L
B2 = L.transpose()
K2 = (A2-B2)/2
K2_frobenius_norm = pow(np.linalg.norm(K2),2)
A2_frobenius_norm = pow(np.linalg.norm(A2),2)
A2_sigma = K2_frobenius_norm/A2_frobenius_norm
#print(K1)
print('Asymmetry index of a symmetric matrix A2 = ',A2_sigma)
```

```
Asymmetry index of a generic matrix A =  0.042
Asymmetry index of a skew-matrix A1 =  1.0
Asymmetry index of a symmetric matrix A2 =  0.0
```

Section 6.12, What about matrix division?

Section 6.13, Exercises

Section 6.14, Answers

Section 6.15, Code challenges

Section 6.16 - Code block 6.13 -- very small numbers should be considered as zeros

In [11]:
```
# two matrices
#A = np.random.randn(2,4)
#B = np.random.randn(4,3)
A = np.random.randint(5,size=(2,4))
B = np.random.randint(5,size=(4,3))
print(A@B)

# initialize product
C1 = np.zeros((2,3))

# sum over columns
for i in range(4):
    C1 += np.outer(A[:,i],B[i,:])
print(C1)
# show equality
D = C1 - A@B
print(D)
```

```
[[27 19 27]
 [17 13 16]]
[[27. 19. 27.]
 [17. 13. 16.]]
[[0. 0. 0.]
 [0. 0. 0.]]
```

Section 6.15, code block 6.15 -- The diagonals of the two product matrices are the same.

In [44]:
```
# a diagonal matrix and a full one
D = np.diag(np.arange(1,5))
```

```
#A = np.random.randn(4,4)
A = np.random.randint(5,size=(4,4))
#print(D)  # matrice diagonale
#print(A)  # matrice 4x4

# two kinds of products
C1 = D*A   # una matrice diagonale moltiplicata per A
C2 = D@A   # prodotto standard delle due matrici
#print(C1)
#print(C2)

# are they the same?
print(np.diag(C1))
print(np.diag(C2))
```

```
[1 8 3 8]
[1 8 3 8]
```

Section 6.15, code block 6.17 -- Square root of a special matrix

La radice quadrata funziona perché A è diagonale. Il motivo per cui funziona è lo stesso per cui funziona $x = (\sqrt{x})^2$, per $x \geq 0$.

In [42]:
```
# diagonal matrix
A = np.diag(np.random.rand(3))
#print(A)
# two symmetric matrices
C1 = (A.T+A)/2
C2 = A.T@A
#print(C1)     # C1 e C2 sono uguali
#print(C2)
# are they the same?
D = C1-np.sqrt(C2)
print(D)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

Section 6.15, code block 6.19 -- With this method the "inequaliry" is always > 0

In [43]:
```
# a matrix and a vector (walk into a bar...)
m = 5
A = np.random.randn(m,m)
v = np.random.randn(m)

# the two sides of the equation
LHS = np.linalg.norm(A@v)
RHS = np.linalg.norm(A,ord='fro') * np.linalg.norm(v)
#print(LHS)
#print(RHS)
# should always be positive
D = RHS-LHS
print(D)
```

```
4.476710510613195
```

In [ ]: