Linear Algebra

Scalar :

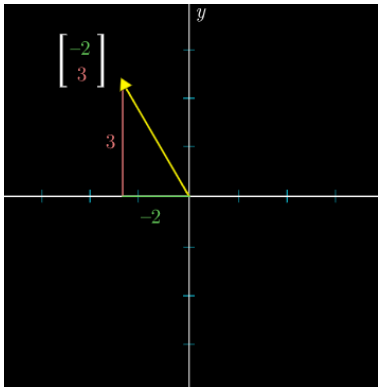
به اعداد در جبرخطی اسکالر می گویند. مانند ... ,2 ,1 و یا طول ، عرض ، جرم و... .

بعد (Dimension) :

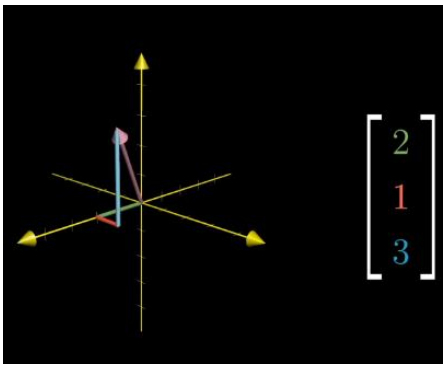
اگر تعداد ویژگی هایی که بر اساس آن یک مقدار را بیان می کنیم برابر یک باشد ، فضای یک بعدی داریم برای مثال معدل افراد که فقط یک عدد است. اما اگر تعداد ویژگی که بر اساس آن یک مقدار را بیان می کنیم برابر دو باشد برای مثال تعداد افراد در هر سن که در اینجا سن یک نوع مقدار است و تعداد افراد نوع دیگری از مقادیر است که در این مثال فضای دو بعدی داریم.

بردار (Vector) :

بردار به اشکال مختلف نشان داده می شود. در علم ریاضیات بردار را به شکل $\vec{v}$ نشان می دهند.
در علم فیزیک و یا به صورت فضایی بردار را به شکل ↗ نشان می دهند که دارای اندازه و جهت است.
در علم کامپیوتر بردار را به شکل یک آرایه که دارای یک ستون می باشد در نظر می گیرند. یعنی برای نشان دادن بردار در علم کامپیوتر $\begin{bmatrix}2\\7\end{bmatrix}$ عمل خواهیم کرد که 2 مقدار x بردار است و 7 مقدار y. ابتدای بردار را از مبدا مختصات یعنی صفر در نظر می گیرند. بردار های می توانند در فضاهای بیشتر از ۲ بعد نیز تعریف شوند در آن صورت بعدهای بیشتر سطرهای بعدی آرایه را تشکیل می دهند.
بردار در فضای دو بعدی :



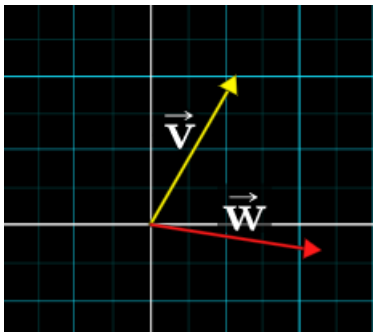
بردار در فضای سه بعدی :



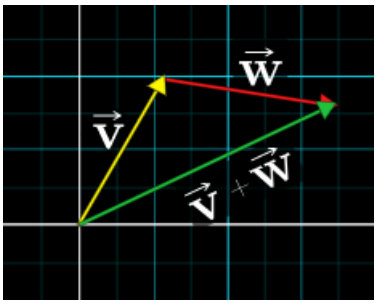
یکی از مثال های بردار نیرو می باشد. در محاسبات نیرو علاوه بر مقدار وارد شده نیرو بر یک جسم ، جهت وارد شدن نیرو نیز برای ما اهمیت دارد.

- **جمع بردار :**

برای جمع کردن دو بردار ، بردار دوم را به موازات جهت آن جا به جا کرده به طوری که ابتدای آن در انتهای بردار اول قرار گیرد و سپس از ابتدای بردار اول به انتهای بردار دوم برداری رسم می کنیم که این بردار حاصل جمع دو بردار خواهد بود.

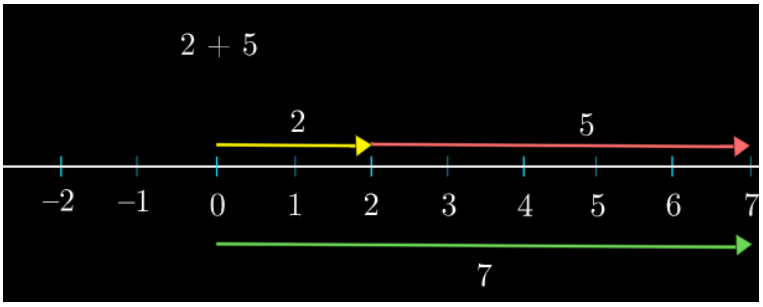


که حاصل جمع آن ها :



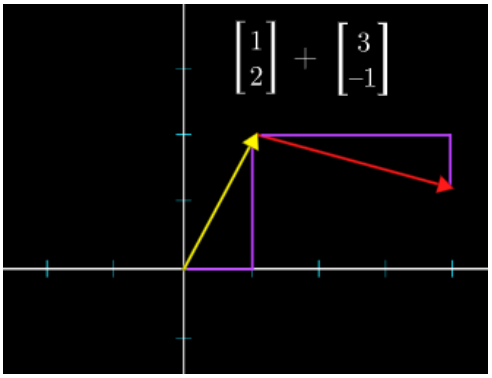
نکته ! ترتیب در جمع اهمیت ندارد.

نکته ! علت آن که باید حتما ابتدای بردار دوم در انتهای بردار اول قرار گیرد و سپس جمع شود این است که فرض کنید دو عدد قرار است با یکدیگر جمع شوند ، داریم :

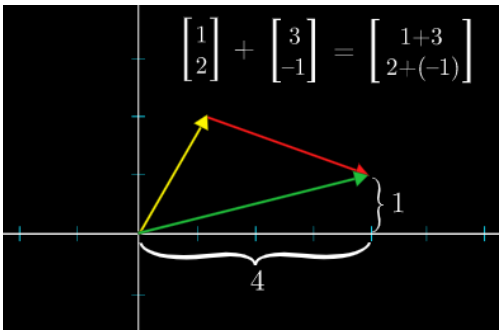


✓ **مثال :** حاصل جمع دو بردار $\begin{bmatrix}1\\2\end{bmatrix}$ و $\begin{bmatrix}3\\-1\end{bmatrix}$ را بدست بیاورید.

برای حل این سوال ابتدا دو بردار را در صفحه مختصات رسم می کنیم.



سپس :

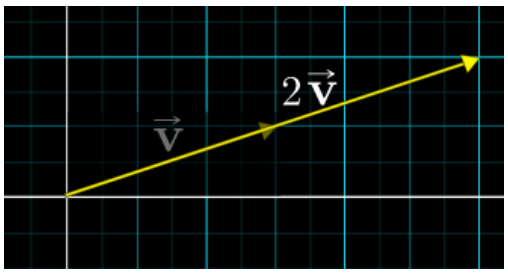


❖ ضرب عدد در بردار :

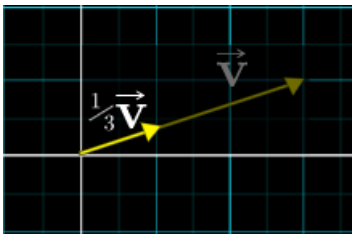
ضرب عدد در بردار باعث کشیدگی یا فشردگی یک بردار می شود.

داریم :

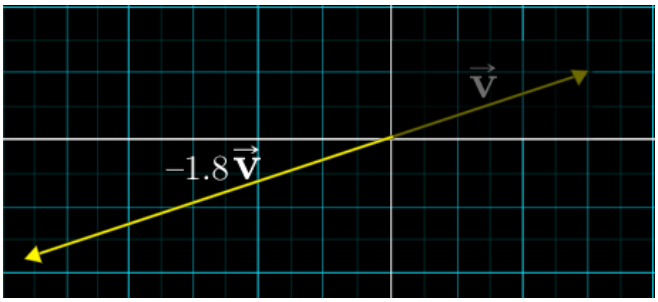
$$2\vec{v} = 2 \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$



یا :



نکته ! چنانچه یک عدد منفی در بردار ضرب شود ، جهت بردار معکوس خواهد شد.

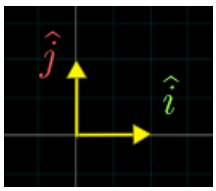


❖ **بردارهای پایه :**

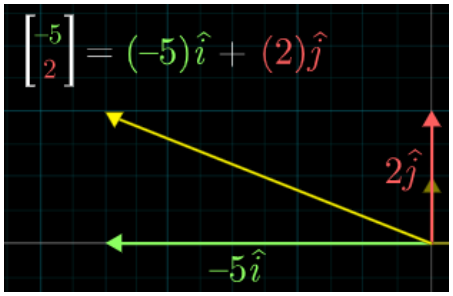
بردارهایی که به واسطه آن ها می توانیم هر برداری در فضا را مشخص کنیم.

بردارهای پایه به دو دسته بردارهای پایه استاندارد و بردارهای پایه غیر استاندارد تقسیم می شوند.

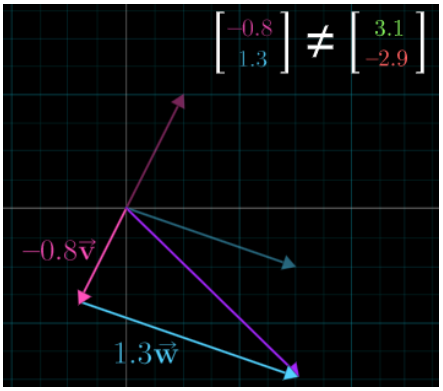
بردارهای پایه استاندارد برای محور x ها بردار i ، برای محور y ها بردار j و برای محور z ها بردار k است.



یعنی برای تعریف هر بردار با استفاده از بردارهای پایه داریم :



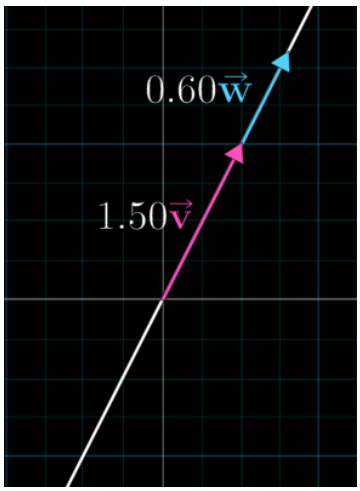
اما بردار پایه می تواند به صورت استاندارد نباشد یعنی هر بردار پایه ای را تعریف کنیم.
داریم :



می توان به این شکل نیز نشان داد :

$$-0.8 \underset{v}{\rightarrow} +1.3 \underset{w}{\rightarrow}$$

نکته ! اگر دو بردار پایه موازی باشند ، فقط و فقط می توان برداری هایی بر اساس آن ها ساخت که در راستای بردارهای پایه باشند.



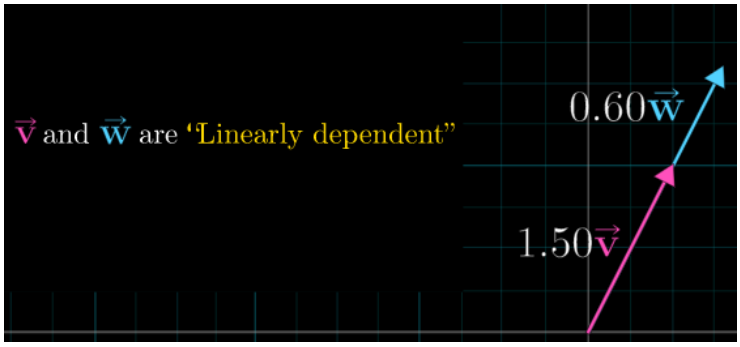
❖ **فضای Span :**

به مجموعه کل فضایی که توسط بردارهای پایه می توان ساخت ، فضای Span گفته می شود.

برای مثال اگر بردارهای پایه موازی نباشند و دو بردار پایه داشته باشیم ، فضای Span یک صفحه است ولی اگر موازی باشند ، فضای Span یک خط است و برای سه بردار پایه فضای Span یک حجم است.

❖ **وابسته خطی (Linearly Dependent) :**

اگر بردار پایه ای به واسطه بردار پایه دیگر ساخته شده باشد ، به آن بردار وابستگی خطی دارد. یعنی بتوان با استفاده از یک بردار ، بردار دیگر را بدست آورد.



❖ **مستقل خطی (Linearly Independent) :**

اگر دو بردار پایه به واسطه یکدیگر ساخته نشده باشند ، آن دو بردار از هم مستقل خطی هستند. یعنی نتوان با استفاده از یک بردار ، بردار دیگر را بدست آورد.

برای رسم یک بردار در پایتون داریم :

بردار دو بعدی :

```
import numpy as np
import matplotlib.pyplot as plt

v = np.array([5, 1])
w = np.array([7, -3])

start = np.array([0, 0])

a = v + w

ax = plt.axes()
ax.set_xlim(-5, 15)
ax.set_ylim(-5, 15)

ax.grid()
ax.quiver(*start, *v, scale=1, angles="xy", scale_units="xy", color="green")
ax.quiver(*start, *w, scale=1, angles="xy", scale_units="xy", color="yellow")
ax.quiver(*start, *a, scale=1, angles="xy", scale_units="xy", color="black")

plt.show()
```

بردار سه بعدی :

```
v = np.array([5, 1, 1])
w = np.array([7, -3, 5])

start = np.array([0, 0, 0])

a = v + w

ax = plt.axes(projection = "3d")
ax.set_xlim(-5, 15)
ax.set_ylim(-5, 15)
ax.set_zlim(0, 15)
ax.grid()
ax.quiver(*start, *v, color="green")
ax.quiver(*start, *w, color="yellow")
ax.quiver(*start, *a, color="black")

plt.show()
```

❖ Norm :

برای توصیف میزان بزرگی یا اندازه یک بردار از مقداری استفاده می کنند که به آن Norm می گویند.

روش های متفاوتی برای محاسبه مقدار Norm وجود دارد :

نرم L1 :

$$\|x\|_1 = \sum_{i=1}^{N} |x_i| = |x_1| + |x_2| + \cdots + |x_N|$$

نرم L2 :

$$\|x\|_2 = (\sum_{i=1}^{N} |x_i|^2)^{1/2} = \sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_N}^2}$$

به صورت کلی :

$$\|x\|_p = (\sum_{i=1}^{N} |x_i|^p)^{1/p} = \sqrt[p]{|x_1|^p + |x_2|^p + \cdots + |x_N|^p}$$

نرم بی نهایت :

$$\|v\|_\infty = \left\| \begin{matrix} -2 \\ 3 \\ 8 \end{matrix} \right\|_\infty = max|v_i| = |8| = 8$$

نکته ! نرم صفر یک بردار تعداد المان هایی از آن بردار که مقدار صفر ندارند را نشان می دهد.

برای محاسبه نرم یک بردار در پایتون داریم :

```python
from numpy import array
from numpy.linalg import norm
from math import inf

v = array([4, 5, 0, 6, 0])

print(v)

L0 = norm(v, 0)
L1 = norm(v, 1)
L2 = norm(v, 2)
Linf = norm(v, inf)

print("L0:", L0)
print("L1:", L1)
print("L2:", L2)
print("Linf:", Linf)
```

❖ **ضرب داخلی بردارها (Dot Product) :**

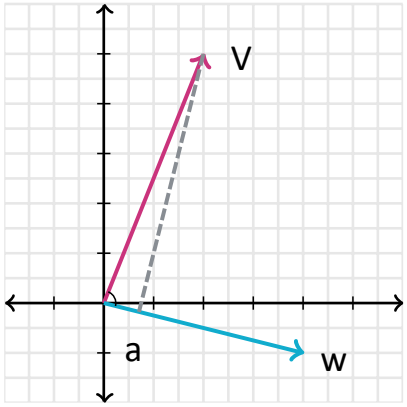
برای بدست آوردن ضرب داخلی دو بردار باید تک تک المان های نظیر به نظیر را در هم ضرب کرده و حاصل آن ها را باهم جمع کنید یعنی :

$$v = \begin{bmatrix} 3 \\ 8 \\ -6 \end{bmatrix}, w = \begin{bmatrix} 2 \\ -4 \\ 1 \end{bmatrix} \rightarrow v.w = 3 * 2 + 8 * (-4) + (-6) * 1 = -32$$

اما تعبیر هندسی ضرب داخلی دو بردار به این شکل است :

اگر فرض کنیم تصویر (Projection) بردار v روی بردار w برابر با a می باشد آنگاه ضرب داخلی بردار v و w برابر می شود با ضرب a در اندازه بردار w.

$$v.w = a \times |w|$$



نکته ! تصویر بردار v روی بردار w که برابر است با a اگر زاویه بین دو بردار را تتا در نظر بگیریم در واقع برابر است با اندازه v در $\cos(\theta)$. به عبارت دیگر مقدار ضرب داخلی دو بردار برابر است با $v.w = |v| * |w| \cos\theta$.

نکته ! اگر اندازه بردار w برابر با یک باشد یا بردار یکه باشد ، ضرب داخلی دو بردار برابر است با مقدار Project شده بردار v روی بردار w.

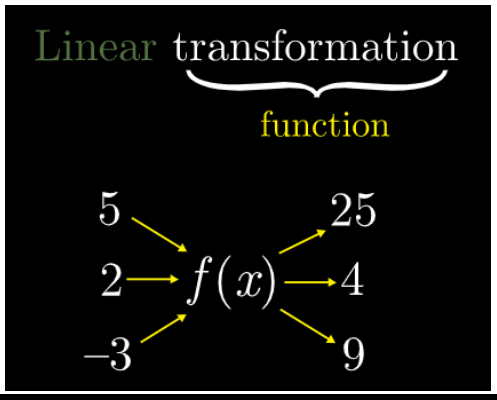
نکته ! اگر دو بردار بر هم عمود باشند ، ضرب داخلی آن ها برابر با صفر خواهد شد چرا که cos زاویه بین دو بردار برابر با صفر خواهد شد.

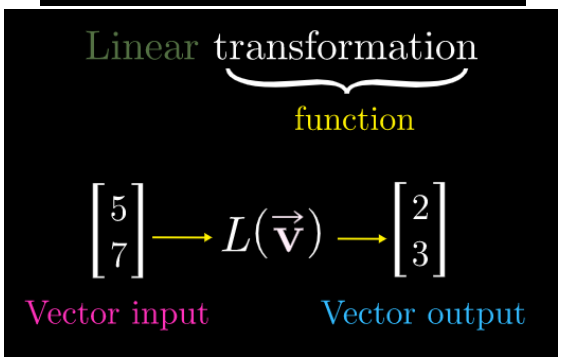
ماتریس (Matrix) :

به آرایه دو بعدی ماتریس می گویند. ماتریس ها دارای سطر و ستون هستند.

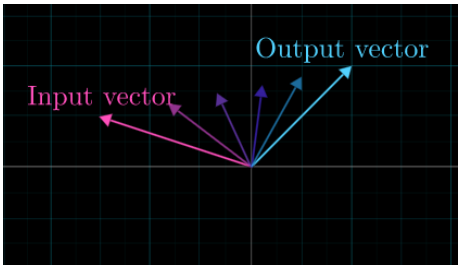
❖ Linear Transformation :

تبدیلات (Transformation) مانند توابع عمل می کنند. یعنی به عنوان ورودی ، ماتریسی به یک تبدیل داده می شود و تغییراتی روی آن ماتریس اعمال می شود و به عنوان خروجی ماتریس دیگری با احتساب تغییرات داریم.

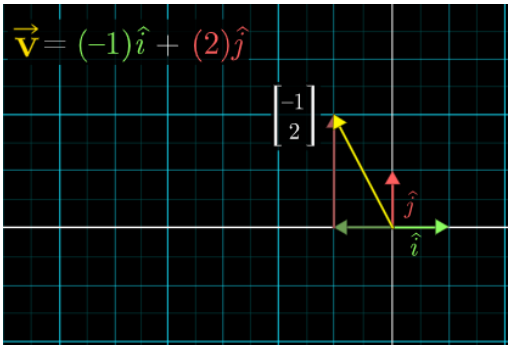




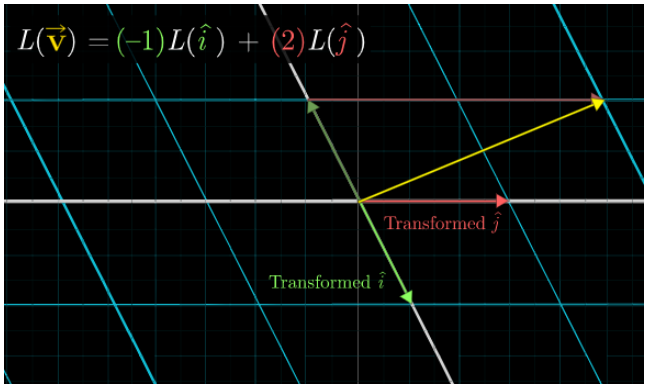
Transformation را مانند جا به جایی می توان در نظر گرفت که یک بردار را جا به جا می کند.



همینطور از Transformation برای تغییر فضا نیز استفاده می شود. برای مثال اگر بخواهیم از یک فضای دو بعدی به یک فضای دو بعدی جدید برویم.
برای مثال در فضای دو بعدی زیر بردار به شکل زیر می باشد :



اگر بردار را در فضای دو بعدی جدید بخواهیم داشته باشیم :



نکته ! برای انتقال بردارها به فضای جدید کافی است بردارهای پایه را تغییر بدهیم.

✓ **مثال :** بردار $v = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ را با استفاده از Transformation به فضای جدید انتقال دهید. $L(i) = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, L(j) = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$

داریم :

$$L(\vec{\mathbf{v}}) = L(-1\hat{i} + 2\hat{j})$$

$$= -1 \cdot L(\hat{i}) + 2 \cdot L(\hat{j})$$

$$= -1 \cdot \begin{bmatrix} 1 \\ -2 \end{bmatrix} + 2 \cdot \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

اگر حالت کلی تر آن را بررسی کنیم داریم :

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = x \begin{bmatrix} a \\ c \end{bmatrix} + y \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

نکته ! a,c مربوط به بردار پایه i است ، b,d مربوط به بردار پایه j است و x,y مربوط به بردار اصلی است.

نکته ! از به هم چسباندن بردارهای پایه ، ماتریس تبدیلات به وجود می آید.

نکته ! ضرب یک بردار در یک ماتریس را می توان تبدیل یک بردار در فضا در نظر گرفت.
پس ماتریس را می توان تبدیل (Transformation) دستگاه مختصات در نظر گرفت.

نکته ! ماتریس تبدیل Shear ، $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ است.

❖ **مرتبه ماتریس :**

تعداد سطر در تعداد ستون را مرتبه ماتریس می گویند.

✓ **مثال :** مرتبه ماتریس زیر چند است.

$$\begin{bmatrix} 5 & 2 & 7 \\ 8 & 9 & 3 \end{bmatrix}$$

ماتریس بالا دارای ۲ سطر و ۳ ستون است پس مرتبه ماتریس ۲ در ۳ است.

نکته ! ماتریسی که تعداد سطر و ستون آن صفر باشد یا مرتبه آن صفر در صفر باشد ، ماتریس تهی نام دارد.

❖ **جمع ماتریس :**

برای جمع کردن دو ماتریس تک تک المان های آن ها را باید به صورت نظیر به نظیر باهم جمع کرد.

✓ **مثال :** حاصل جمع دو ماتریس زیر را بدست بیاورید.

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, b = \begin{bmatrix} 12 & 7 & 5 \\ 9 & 8 & 7 \end{bmatrix}$$

داریم :

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 12 & 7 & 5 \\ 9 & 8 & 7 \end{bmatrix} = \begin{bmatrix} 13 & 9 & 8 \\ 13 & 13 & 13 \end{bmatrix}$$

نکته ! برای جمع دو ماتریس باید هر دو ماتریس هم مرتبه باشند یعنی تعداد سطر و ستون آن ها برابر باشد.

نکته ! به ماتریسی که تعداد سطر و ستون آن برابر باشد ، ماتریس مربعی می گویند.

❖ **ضرب ماتریس :**

ضرب دو ماتریس در هم از دیدگاه هندسی یعنی دو بار تبدیل انجام شود (تبدیلات پشت سرهم).

ماتریس ها به دو شکل در هم ضرب می شوند. ضرب هر عضو در عضو نظیرش (Element by Element) و ضرب ماتریسی (Dot Product).

✓ **مثال :** حاصل ضرب دو ماتریس زیر را به صورت عضو های نظیر بدست بیاورید.

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, b = \begin{bmatrix} 12 & 7 & 5 \\ 9 & 8 & 7 \end{bmatrix}$$

داریم :

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 12 & 7 & 5 \\ 9 & 8 & 7 \end{bmatrix} = \begin{bmatrix} 12 & 14 & 15 \\ 36 & 40 & 42 \end{bmatrix}$$

✓ **مثال :** حاصل ضرب دو ماتریس زیر را به صورت ضرب ماتریسی بدست بیاورید.

$$a = \begin{bmatrix} 2 & 5 \\ 7 & 4 \end{bmatrix}, b = \begin{bmatrix} 9 \\ 2 \end{bmatrix}$$

داریم :

$$\begin{bmatrix} 2 & 5 \\ 7 & 4 \end{bmatrix} \begin{bmatrix} 9 \\ 2 \end{bmatrix} = \begin{bmatrix} 18 + 10 \\ 63 + 8 \end{bmatrix} = \begin{bmatrix} 28 \\ 71 \end{bmatrix}$$

نکته ! در ضرب ماتریسی باید تعداد ستون ماتریس اول برابر با تعداد سطر ماتریس دوم باشد.

نکته ! اگر مرتبه ماتریس ها برابر باشند با $m \times n, n \times q$ ، مرتبه ماتریس حاصل ضرب ماتریسی برابر است با $m \times q$.

نکته ! ضرب ماتریسی خاصیت جا به جایی ندارد. یعنی $ab \neq ba$.

نکته ! اگر یک عدد در ماتریس ضرب شود ، در تک تک المان های آن ضرب خواهد شد.

برای ضرب ماتریس ها در پایتون داریم :

```python
import numpy as np

a = np.array([[2, 5], [7, 4]])

b = np.array([9, 2])

result = np.matmul(a, b)

print(result)
```

نکته ! از دستور dot() نیز می توان برای ضرب ماتریس ها استفاده کرد.

نکته ! چنانچه از عملگر ضرب یا * برای ضرب ماتریس ها استفاده کنید ، ضرب به شکل Element by Element در نظر گرفته می شود.

نکته ! چنانچه مرتبه ماتریس ها برای ضرب ماتریسی در پایتون درست نباشد ، پایتون روند پیمایش و در نظر گرفتن مرتبه ماتریس را تغییر می دهد.

❖ **ماتریس همانی (Identity Matrix) :**

به تبدیلی که بر اثر آن بردار تغییر نکند (همان بردار قبلی باقی بماند) ماتریس همانی یا ماتریس یکه می گویند.

نکته ! ماتریس همانی ، ماتریس مربعی است که المان های قطر اصلی آن همگی ۱ هستند و مابقی المان ها صفر هستند.

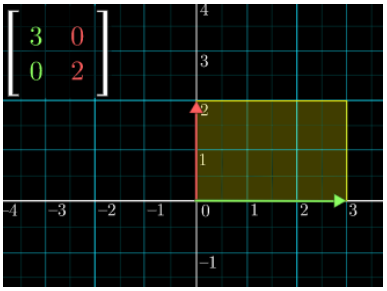
$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

برای استفاده از ماتریس همانی در پایتون داریم :

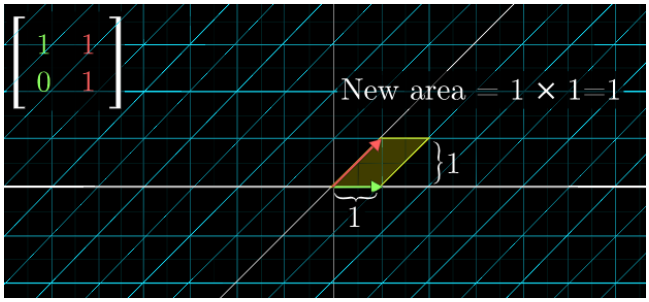
```
import numpy as np

v = np.array([4, 7])

I = np.eye(2)

result = np.matmul(I, v)

print(result)
```

❖ **دترمینان (Determinant) :**

به مساحت تولید شده توسط یک انتقال دترمینان می گویند. یعنی مساحت محصور بین دو بردار.



و یا :



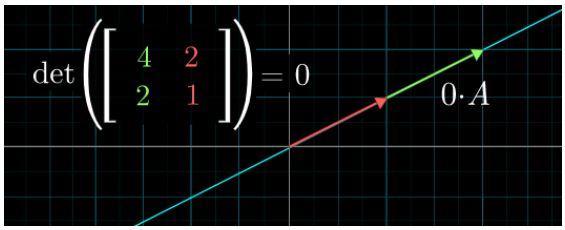
فرمول دترمینان به شکل زیر است :

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = (a * d) - (b * c)$$

نکته ! فقط برای ماتریس های مربعی می توان دترمینان را محاسبه کرد.

نکته ! اگر ماتریس تبدیل مرتبه ۳ بود ، آنگاه دترمینان حجم محصور بین بردارها را نشان می داد.

نکته ! اگر دترمینان یک ماتریس صفر شد یعنی مساحت تولید شده صفر است. به عبارت دیگر بردارهای انتقال یا ستون های ماتریس تبدیل به هم وابسته خطی هستند.



نکته ! اگر ستون های ماتریس تبدیل به هم وابسته خطی باشند ، بعد بردار کم خواهد شد (کاهش بعد) ، در نتیجه در یک محور بردار مقداری ندارد پس مساحتی بین بردار و آن محور تشکیل نمی شود که باعث می شود دترمینان صفر شود. (وابستگی خطی در ماتریس تبدیل دترمینان را صفر خواهد کرد)

برای محاسبه دترمینان در پایتون داریم :

```
import numpy as np

A = np.array([[3, 0],[0, 2]])

Det = np.linalg.det(A)

print("the Determinant is:", Det)
```

و یا :

```
B = np.array([[4, 2],[2, 1]])

Det = np.linalg.det(B)

print("the Determinant is:", Det)
```

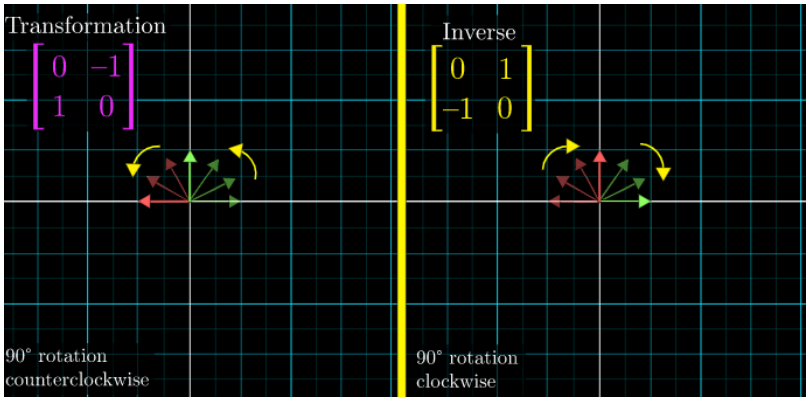
همچنین برای رسم بردارها داریم :

```python
import matplotlib.pyplot as plt

ax = plt.axes()
ax.set_xlim(-2, 5)
ax.set_ylim(-2, 5)

ax.grid()
ax.quiver(0, 0, *A[0], scale=1, angles="xy", scale_units="xy", color="blue")
ax.quiver(0, 0, *A[1], scale=1, angles="xy", scale_units="xy", color="red")

plt.show()
```

❖ ماتریس وارون یا معکوس (Inverse) :

ماتریسی که یک تبدیل را به حالت اولیه برگرداند ، ماتریس وارون یا معکوس نام دارد.

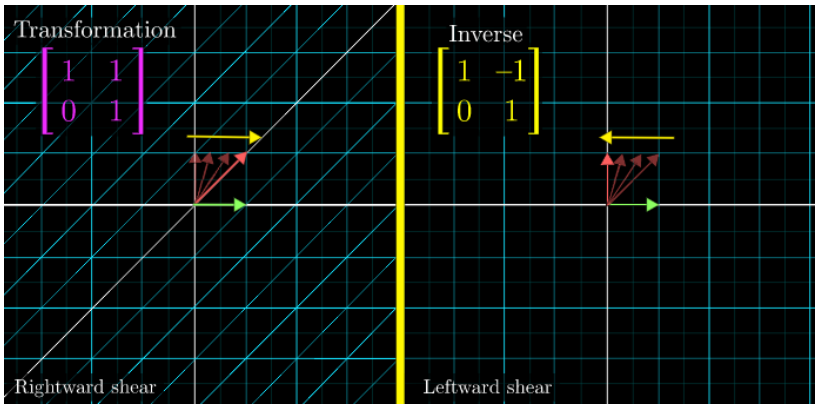
داریم :



بردار بالا تحت تبدیل دورانی ، ۹۰ درجه در خلاف عقربه های ساعت چرخیده است. وارون ماتریس تبدیل دورانی بردار را مجدد به حالت اولیه خود بر می گرداند.

نکته ! ماتریس تبدیل دورانی ، یک بردار را می چرخاند.

و یا داریم :



نکته ! $AA^{-1} = A^{-1}A = I$ است.

نکته ! اگر دترمینان ماتریس برابر صفر شود یعنی ماتریس باعث کاهش بعد شده باشد ، مثلا از یک صفحه به یک خط ، طبیعی است که ماتریس وارون نتواند بعد کاهش یافته را جبران کند و به صفحه برگرداند. بنابراین اگر دترمینان ماتریسی صفر باشد آن ماتریس وارون ناپذیر است.

برای محاسبه ماتریس وارون داریم :

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

برای محاسبه ماتریس وارون در پایتون داریم :

```python
import numpy as np

A = np.array([[1, 0], [1, 1]])

A_1 = np.linalg.inv(A)

print(A_1)
```

❖ ماتریس ترانهاده (Transpose) :

به ماتریسی که جای سطر و ستون آن با هم عوض شده باشد ، ماتریس ترانهاده می گویند.

اگر ماتریسی به شکل زیر باشد :

$$A = \begin{bmatrix} 5 & 6 & 4 \\ 8 & 3 & 2 \\ 9 & 12 & 1 \end{bmatrix}$$

ماتریس ترانهاده آن به شکل زیر است :

$$A^T = \begin{bmatrix} 5 & 8 & 9 \\ 6 & 3 & 12 \\ 4 & 2 & 1 \end{bmatrix}$$

برای بدست آوردن ماتریس ترانهاده در پایتون داریم :

```python
import numpy as np

A = np.array([[5, 6, 4], [8, 3, 2], [9, 12, 1]])

AT = A.T

# Or

AT = np.transpose(A)

print(AT)
```

❖ ماتریس متعامد (Orthogonal Matrix) :

ماتریسی که ستون های آن بر هم عمود باشند و یکه باشند (اندازه آن برابر با یک باشد) را ماتریس متعامد می گویند. زیرا که هر کدام از ستون ها بردارهای پایه هستند و زمانی که بردارهای پایه بر هم عمود باشند ، ماتریس متعامد است.

✓ **مثال :** Q^TQ برابر چه مقدار می شود.

داریم :

$$Q^TQ = \begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ - & v_3 & - \end{bmatrix}\begin{bmatrix} - & - & - \\ v_1 & v_2 & v_3 \\ - & - & - \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

برای بدست آوردن هر المان ، مانند ضرب داخلی بردارها در هم ضرب می کنیم. هر بردار در خودش ضرب می شود پس مقدار برابر با یک می شود. بردارهایی که در غیر خودشان ضرب شده اند چون برهم عمودند حاصل ضرب آن ها برابر صفر می شود. پس حاصل ضرب یک ماتریس در ماتریس ترانهاده برابر با ماتریس همانی می شود.

با توجه به مثال بالا داریم :

$$Q^{-1}QQ^T = Q^{-1}I$$

در نتیجه :

$$Q^T = Q^{-1}$$

با توجه به مثال بالا ماتریس متعامد را ماتریس مربعی تعریف می کنند که ماتریس ترانهاده آن با وارون آن برابر باشد.

❖ **Rank :**

به تعداد ابعاد در فضای خروجی ماتریس ، Rank ماتریس می گویند.

برای مثال اگر یک ماتریس باعث کاهش بعد شود ، یعنی از فضای سه بعدی به فضای دو بعدی تبدیل انجام دهد ، Rank ماتریس برابر ۲ خواهد بود. این ماتریس شامل دو ستون مستقل از هم خواهد بود و یک ستون وابسته به ستون دیگر است علاوه بر این دترمینان آن نیز صفر خواهد شد.

بنابراین می توان Rank ماتریس را نیز به تعداد ستون هایی از ماتریس که از یکدیگر مستقل خطی هستند ، تعریف کرد.

برای بدست آوردن Rank ماتریس در پایتون داریم :

```python
import numpy as np

A = np.array([[1, 0], [0, 1]])
B = np.array([[1, 2], [2, 4]])

A_Rank = np.linalg.matrix_rank(A)
B_Rank = np.linalg.matrix_rank(B)

print("A_Rank is:", A_Rank)
print("B_Rank is:", B_Rank)
```

❖ بردار ویژه و مقدار ویژه (Eigen Vector & Eigen Value) :

به صورت هندسی معادله زیر به چه معنی است؟

$$Av = \lambda v$$

منظور از v یک بردار ، منظور از A یک ماتریس و منظور از λ یک عدد است.

زمانی که حاصل ضرب یک ماتریس در بردار با حاصل ضرب یک عدد در بردار برابر است بدین معنی می باشد که ضرب ماتریس در بردار ، جهت بردار را تغییر نمی دهد و فقط بردار را کشیده تر ، فشرده تر و یا ثابت نگه می دارد.

در معادله بالا به دنبال بردارهایی هستیم که پس از تبدیل یا ضرب یک ماتریس در آن ها ، جهت بردار ثابت باشد. به این بردارها ، بردار ویژه (v) و به ضریب تغییر این بردارها نسبت به بردار اولیه ، مقدار ویژه (λ) می گویند.

✓ **مثال :** بردار ویژه و مقدار ویژه تبدیل زیر را بدست بیاورید.

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

داریم :

$$Ax = x \rightarrow \lambda = 1, any\ x$$

یعنی تمام بردارها بر اساس تبدیل بالا ، بردار ویژه هستند و مقدار ویژه برابر با ۱ است.

برای بدست آوردن بردار ویژه و محاسبه مقدار ویژه داریم :

$$Av = \lambda v \rightarrow Av = \lambda Iv \rightarrow Av - \lambda Iv = 0 \rightarrow (A - \lambda I)v = 0$$

نکته ! اگر ماتریس تبدیلی در برداری غیر صفر ضرب شود و حاصل برابر صفر شود ، زیرا که حاصل یک تبدیل برابر با عدد شده است پس کاهش بعد رخ داده است و می توان نتیجه گرفت دترمینان ماتریس تبدیل برابر با صفر است.

پس داریم :

$$\det(A - \lambda I) = 0$$

که با توجه به نکته بالا می توان مقدار ویژه را محاسبه کرد.

- ✓ **مثال :** بردار ویژه و مقدار ویژه تبدیل زیر را بدست بیاورید.

$$A = \begin{bmatrix} 0.7 & 0.6 \\ 0.4 & 0.1 \end{bmatrix}$$

داریم :

$$A - \lambda I = \begin{bmatrix} 0.7 & 0.6 \\ 0.4 & 0.1 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} = \begin{bmatrix} 0.7 - \lambda & 0.6 \\ 0.4 & 0.1 - \lambda \end{bmatrix}$$

سپس :

$$\det(A - \lambda I) = 0$$

مقادیر ویژه برابر است با :

$$\lambda = 0.97, -0.17$$

برای بدست آوردن بردار ویژه داریم :

$$Av = \lambda v \rightarrow Av = 0.97v \rightarrow \begin{bmatrix} 0.7 & 0.6 \\ 0.4 & 0.1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0.97 \begin{bmatrix} x \\ y \end{bmatrix}$$

سپس :

$$0.7x + 0.6y = 0.97x \rightarrow x = 2.22y \rightarrow v_1 = \begin{bmatrix} 0.91 \\ 0.42 \end{bmatrix}$$

برای بدست آوردن بردار ویژه و محاسبه مقدار ویژه در پایتون داریم :

```python
import numpy as np

A = np.array([[0.7, 0.6],
              [0.4, 0.1]])

Eigen = np.linalg.eig(A)

print(Eigen)
```

نکته ! اگر ماتریس تبدیل $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ باشد ، جمع المان های قطر اصلی (Trace) برابر با جمع مقادیر ویژه خواهد شد. یعنی $a + d = \lambda_1 + \lambda_2$.

نکته ! $m = \frac{\lambda_1 + \lambda_2}{2} = \frac{a+d}{2}$

نکته ! $det(A) = ad - bc = \lambda_1 \lambda_2$

نکته ! $\lambda_1 \lambda_2 = m^2 - d^2$ که d فاصله مقادیر ویژه از m است.

تجزیه (Decomposition) :

❖ تجزیه بردار (Vector Decomposition) :

روش جمع کردن دو بردار را یاد گرفتیم ، حال قرار است از برداری که حاصل جمع دو بردار است ، دو بردار اصلی را پیدا کنیم. به این کار تجزیه بردار می گویند.

✓ **مثال :** بردار $v = \begin{bmatrix} 5 \\ 6 \\ 7 \end{bmatrix}$ را به سه راستای $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ تجزیه کنید.

داریم :

$$a\,\vec{i} + b\,\vec{j} + c\,\vec{k} = v$$

پس :

$$\begin{bmatrix} a \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ b \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ c \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 7 \end{bmatrix}$$

✓ **مثال :** بردار $m = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$ را به سه راستای $w = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, v = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, u = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$ تجزیه کنید.

داریم :

$$a\,\vec{u} + b\,\vec{v} + c\,\vec{w} = m$$

پس :

$$\begin{bmatrix} a \\ 0 \\ a \end{bmatrix} + \begin{bmatrix} 0 \\ b \\ b \end{bmatrix} + \begin{bmatrix} c \\ c \\ 0 \end{bmatrix} = \begin{bmatrix} a+c \\ b+c \\ a+b \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$$

و از روش حل دستگاه معادلات به واسطه ماتریس ها مقادیر را بدست میاوریم :

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

A X m

سپس :

$$\vec{X} = A^{-1}\vec{m}$$

برای حل ادامه سوال از پایتون کمک می گیریم :

```python
import numpy as np

A = np.array([[1, 0, 1],
              [0, 1, 1],
              [1, 1, 0]])

m = np.array([2, 3, 4])

A_1 = np.linalg.inv(A)

result = np.matmul(A_1, m)

a, b, c = result
print("a is:", a)
print("b is:", b)
print("c is:", c)
```

❖ تجزیه ماتریس (Matrix Decomposition) :

در تجزیه ماتریس هدف آن است که ماتریس اصلی را به چند ماتریس ساده تر تجزیه کنیم.

تجزیه ماتریس ها معمولا به چند روش متفاوت انجام می پذیرد :

۱- LU Decomposition

۲- QR Decomposition

۳- Eigen Decomposition

۴- Singular Value Decomposition (SVD)

▪ LU Decomposition :

در تجزیه به این روش ماتریس مربعی A را به دو ماتریس یکی به شکل پایین مثلثی و دیگری به شکل بالا مثلثی تجزیه می کنیم.

$$A = L.U$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \times \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

✓ **مثال :** ماتریس زیر را به روش LU تجزیه کنید.

$$A = \begin{bmatrix} 6 & 3 \\ 4 & 8 \end{bmatrix}$$

داریم :

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \times \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}$$

سپس :

$$a_{11} = l_{11}u_{11} + 0 * 0$$
$$a_{12} = l_{11}u_{12} + 0 * u_{22}$$
$$a_{21} = l_{21}u_{11} + l_{22} * 0$$
$$a_{22} = l_{21}u_{12} + l_{22} * u_{22}$$

روابط بالا به ازای مقادیر مختلفی ممکن است صحیح باشد اما برای بدست آوردن یک جواب صریح ، یک قید در نظر می گیریم. معمولا قید در نظر گرفته شده بدین شکل است که المان های قطر اصلی ماتریس Lower را ۱ در نظر می گیریم.

با در نظر گرفتن این قید حل معادلات ساده تر خواهد شد.

برای حل ادامه سوال از پایتون کمک می گیریم :

```python
import numpy as np
import scipy.linalg
import pprint

A = np.array([[6, 3],
              [4, 8]])

P, L, U = scipy.linalg.lu(A)

print("P:")
pprint.pprint(P)
print("L:")
pprint.pprint(L)
print("U:")
pprint.pprint(U)
```

- **QR Decomposition** :

در تجزیه به این روش لزوما ماتریس A نباید مربعی باشد و ماتریس A را به دو ماتریس یکی به عنوان ماتریس متعامد و دیگری به عنوان ماتریس بالا مثلثی تجزیه می کند.

✓ **مثال** : ماتریس زیر را به روش QR تجزیه کنید.

$$A = \begin{bmatrix} 4 & 3 \\ 2 & 1 \\ 6 & 5 \end{bmatrix}$$

برای حل این سوال از پایتون کمک می گیریم :

```python
import numpy as np
import pprint

A = np.array([[4, 3],
              [2, 1],
              [6, 5]])

Q, R = np.linalg.qr(A, "complete")

print("Q:")
pprint.pprint(Q)
print("R:")
pprint.pprint(R)

Q.dot(R)
```

- **Eigen Decomposition** :

در تجزیه به این روش از مقادیر ویژه و بردارهای ویژه استفاده می کنیم. یعنی اگر ماتریس مربعی A را بخواهیم تجزیه کنیم ، داریم :

$$AU_1 = \lambda_1 U_1$$
$$AU_2 = \lambda_2 U_2$$

سپس :

$$A\begin{bmatrix} \vdots & \vdots \\ U_1 & U_2 \\ \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots \\ U_1 & U_2 \\ \vdots & \vdots \end{bmatrix}\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

$$AU = U\Lambda \rightarrow A = U\Lambda U^{-1}$$

در این روش ماتریس A به سه ماتریس تجزیه شد.

نکته ! ماتریس U باید شامل بردارهایی با اندازه یک (یکه) باشد. یعنی نرم آن ها باید یک باشد.

با استفاده از پایتون داریم :

```python
import numpy as np

A = np.array([[0.7, 0.6],
              [0.4, 0.1]])

L, U = np.linalg.eig(A)

U_1 = np.linalg.inv(U)

L_diag = np.diag(L)

print(U)
print(L)
print(U_1)
```

نکته ! $A^n = U\Lambda^n U^{-1}$

✓ **مثال :** حاصل ماتریس زیر به توان ۵ را محاسبه کنید.

$$A = \begin{bmatrix} 0.7 & 0.6 \\ 0.4 & 0.1 \end{bmatrix}$$

با استفاده از پایتون داریم :

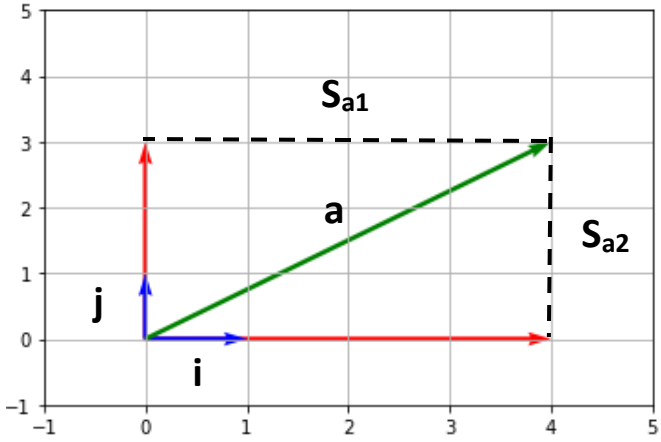
```python
import numpy as np

A = np.array([[0.7, 0.6],
              [0.4, 0.1]])

A_5 = np.linalg.matrix_power(A, 5)

print(A_5)
```

- **Singular Value Decomposition (SVD) :**

طبق مباحث گفته شده ، می توان یک بردار را به بردارهای پایه تجزیه کرد. در تجزیه بردار به بردارهای پایه دو نکته برای ما حائز اهمیت است.

۱- جهت بردار های پایه

۲- اندازه تجزیه شده بر روی بردارهای پایه

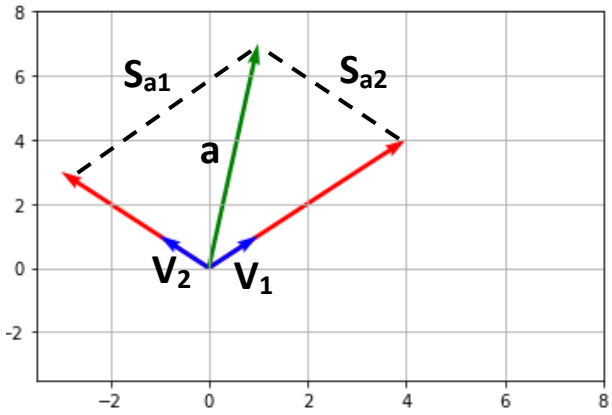


در اینجا بدلیل آن که در فضای دو بعد هستیم ، بردار را به دو بردار پایه تجزیه می کنیم اما اگر در فضای n بعدی بودیم به n بردار پایه تجزیه می کردیم.

طبق شکل بالا ، بردار a در دو راستای i و j تجزیه شده است. در تجزیه بردارها داشتیم ، اندازه تصویر بردار بر روی آن راستا اگر در بردار پایه ضرب شود ، بردار در راستای آن بردار پایه تجزیه شده است.

در شکل بالا اندازه تصویر بردار a روی راستای بردار پایه i برابر با S_{a1} است و برای بردار پایه j نیز شبیه به همان می شود.

اما بردارهای پایه ممکن است غیر استاندارد باشند ، یعنی :



در اینجا بردارهای پایه خود حاصل یک تبدیل هستند. یعنی داریم :

$$v_1 = \begin{bmatrix} v_{1x} \\ v_{1y} \end{bmatrix}, v_2 = \begin{bmatrix} v_{2x} \\ v_{2y} \end{bmatrix}$$

و تجزیه بردار a روی بردارهای پایه استاندارد بدین شکل است :

$$a = \begin{bmatrix} a_x \\ a_y \end{bmatrix}$$

می توان با استفاده از ضرب نقطه ای بردارها ، بردار a را تجزیه کرد :

$$a^T . v_1 = \begin{bmatrix} a_x & a_y \end{bmatrix} \begin{bmatrix} v_{1x} \\ v_{1y} \end{bmatrix} = s_{a1}$$

$$a^T . v_2 = \begin{bmatrix} a_x & a_y \end{bmatrix} \begin{bmatrix} v_{2x} \\ v_{2y} \end{bmatrix} = s_{a2}$$

ماتریس ها در واقع جمع تعدادی بردار بودند که هر ستون ماتریس مربوط به یک بردار بود ، پس :

$$a^T . V = \begin{bmatrix} a_x & a_y \end{bmatrix} \begin{bmatrix} v_{1x} & v_{2x} \\ v_{1y} & v_{2y} \end{bmatrix} = \begin{bmatrix} s_{a1} & s_{a2} \end{bmatrix}$$

می توان چندین بردار را تجزیه کرد :

$$A . V = \begin{bmatrix} a_x & a_y \\ b_x & b_y \end{bmatrix} \begin{bmatrix} v_{1x} & v_{2x} \\ v_{1y} & v_{2y} \end{bmatrix} = \begin{bmatrix} s_{a1} & s_{a2} \\ s_{b1} & s_{b2} \end{bmatrix} = S$$

بردار ها را به صورت سطری زیر هم قرار می دهیم و بردار های پایه را به صورت ستونی کنار هم قرار می دهیم.

می توان تعداد ابعاد را نیز بیشتر کرد. در این صورت تعداد ستون های ماتریس V بیشتر خواهد شد.

در ابعاد بالاتر ماتریس A ، ماتریس با ابعاد n*d و ماتریس V ، ماتریس با ابعاد d*d است.

ماتریس S ، ماتریس مقادیر تصویر شده روی بردارهای پایه است. هر ستون از این ماتریس مربوط به مقادیر تصویر شده تمامی بردار ها روی یکی از بردارهای پایه است.

نکته ! ماتریس V شامل بردارهای پایه ای است که برهم عمودند پس یک ماتریس متعامد است که یعنی وارون آن با ترانهاده آن برابر است.

داریم :

$$A.V = S \rightarrow A = SV^{-1} = SV^{T}$$

در نتیجه می توان گفت ماتریسی از نقاط یا بردار ها (A) که در ماشین لرنینگ دیتاست در نظر گرفته می شود را می توان به صورت اندازه تصویر آن بردار ها (S) روی تعدادی بردار پایه (V) نشان داد.

ماتریس S را می توان به دو ماتریس تجزیه کرد :

$$A = U\Sigma V^{T}$$

باید هر کدام از ستون های ماتریس S نرمالایز شوند. یعنی بر یک عدد مشخص تقسیم شوند. باید طوری تقسیم کرد که اندازه هر ستون برابر با ۱ شود.

در ماتریس ها تقسیم وجود ندارد اما با استفاده از ضرب ماتریسی می توان تقسیم را انجام داد.

✓ **مثال :** ستون دوم ماتریس زیر را بر ۴ تقسیم کنید.

$$A = \begin{bmatrix} 5 & 4 \\ 5 & 4 \end{bmatrix}$$

داریم :

$$A = \begin{bmatrix} 5 & 4 \\ 5 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 1 \\ 5 & 1 \end{bmatrix} \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}$$

سپس :

$$A = \begin{bmatrix} 5 & 1 \\ 5 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 5 & 4 \end{bmatrix}$$

با توجه به مثال بالا می توان نتیجه گرفت برای تقسیم کردن یک ستون از ماتریس بر عدد ، باید المان قطر اصلی ستونی که می خواهیم تقسیم شود از ماتریسی که در ماتریس حاصل ، ضرب می شود برابر با آن عدد باشد ، مابقی المان های قطر اصلی برابر با ۱ باشد و مابقی المان ها صفر باشد.

هدف آن بود که ماتریس S نرمالایز شود به طوری که اندازه هر ستون برابر با ۱ باشد. برای اینکار باید هر ستون از ماتریس S بر اندازه آن ستون تقسیم شود تا اندازه آن ستون برابر با ۱ شود.

یعنی داریم :

$$\sigma_1 = \sqrt{{s_{a1}}^2 + {s_{b1}}^2}$$

$$\sigma_2 = \sqrt{{s_{a2}}^2 + {s_{b2}}^2}$$

سپس :

$$S = \begin{bmatrix} \frac{s_{a1}}{\sigma_1} & \frac{s_{a2}}{\sigma_2} \\ \frac{s_{b1}}{\sigma_1} & \frac{s_{b2}}{\sigma_2} \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} = \begin{bmatrix} u_{a1} & u_{a2} \\ u_{b1} & u_{b2} \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}$$

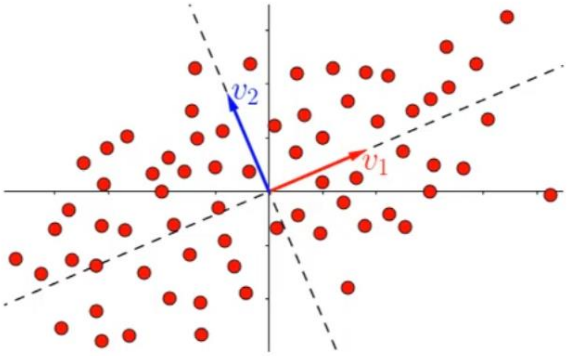
که به ماتریس σ ها ، Σ می گویند.

مقدار σ ، میزان نزدیک بودن بردار ها به یک بردار پایه یا محور مختصات است. هر σ ای که بیشتر باشد یعنی به همان محور مختصات نزدیک تر است.

نکته ! ماتریس سیگما میزان نزدیک بودن داده ها به محورهای مختصات را نشان می دهد. این بدین معنی است که کدام محور مختصات اهمیت بیشتری دارد.

در ماشین لرنینگ برای کاهش بعد از این موضوع استفاده می شود.

- ✓ **مثال :** پراکندگی دیتاها به شکل زیر است. اگر بخواهیم از فضای دو بعدی به فضای یک بعدی ، کاهش بعد داشته باشیم ، کدام محور را بهتر است حذف کنیم.



بدلیل آنکه دیتاها حول محور v_1 نزدیکتر هستند پس این محور اهمیت بیشتر و سیگمای بیشتری دارد بنابراین این محور را نباید حذف کنیم.

نکته ! می توان ماتریس سیگما و U را بر اساس اهمیت یعنی بر اساس سیگمای بزرگتر از بزرگ به کوچک مرتب کرد. پس ستون اول این ماتریس ها مهم تر از ستون دوم ، ستون دوم مهم تر از ستون سوم و... است.

✓ **مثال :** ماتریس زیر را به روش SVD تجزیه کنید.

$$A = \begin{bmatrix} 4 & 5 \\ 2 & 6 \\ 3 & 1 \end{bmatrix}$$

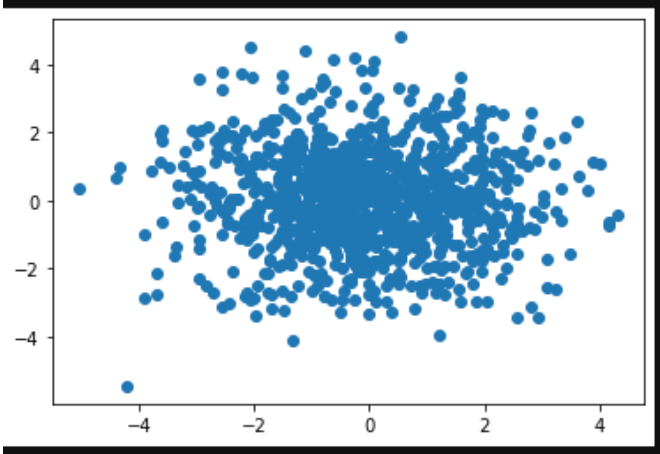
با استفاده از پایتون داریم :

```python
import numpy as np

A = np.array([[4, 5],
              [2, 6],
              [3, 1]])

U, S, V_T = np.linalg.svd(A)

print(U)
print(S)
print(V_T)
```

✓ مثال :

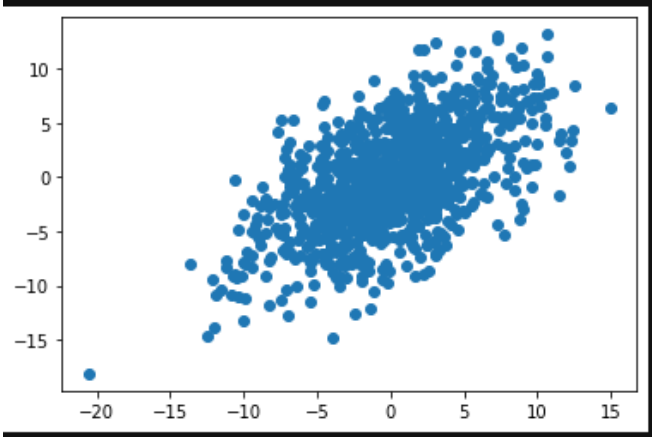
```
import numpy as np
import matplotlib.pyplot as plt
```

```
x1 = np.random.normal(loc = 0, scale = 1.5, size = 1000)
y1 = np.random.normal(loc = 0, scale = 1.5, size = 1000)
A = np.stack((x1, y1), axis=1)

plt.scatter(x1, y1)
plt.show()
```




```
M = np.array([[1, 3],
              [3, 1]])

data = np.matmul(A, M)
x = data.T[0]
y = data.T[1]
```

```
plt.scatter(x, y)
plt.show()
```




```
U, S, V_T = np.linalg.svd(data)

print(S)
```

```
[188.0186007   96.86798028]
```

```
sigma = np.zeros((data.shape))

print(sigma)
```

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

```
for i in range(len(S)):
    sigma[i, i] = S[i]

print(sigma)
```

```
[[188.0186007    0.        ]
 [  0.          96.86798028]
 [  0.           0.        ]
 ...
 [  0.           0.        ]
 [  0.           0.        ]
 [  0.           0.        ]]
```

```
v1 = V_T[0]*20
v2 = V_T[1]*20
```

```python
ax = plt.axes()
ax.set_xlim(-20, 15)
ax.set_ylim(-20, 15)

plt.scatter(data.T[0], data.T[1])

ax.quiver(0, 0, *v1, scale=1, angles="xy", scale_units="xy", color="green")
ax.quiver(0, 0, *v2, scale=1, angles="xy", scale_units="xy", color="red")

plt.show()
```