BLM442 Büyük Veri Analizine Giriş

Lineer Cebir

Dr. Süleyman Eken

Bilgisayar Mühendisliği
Kocaeli Üniversitesi

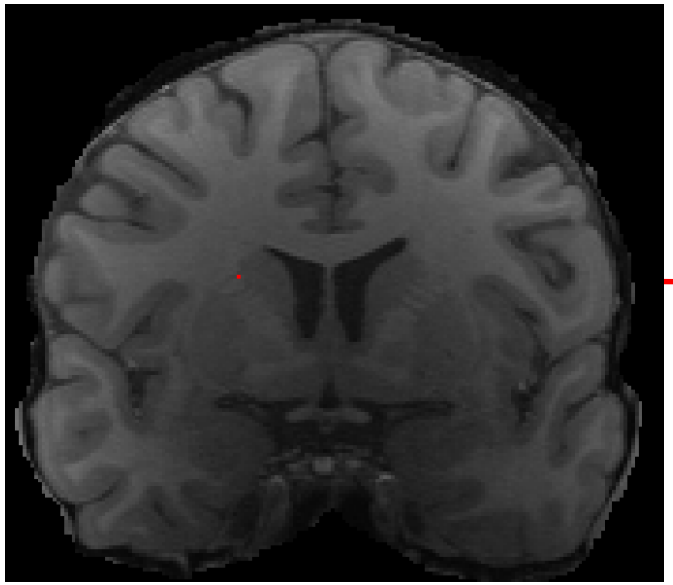
Sunum Planı

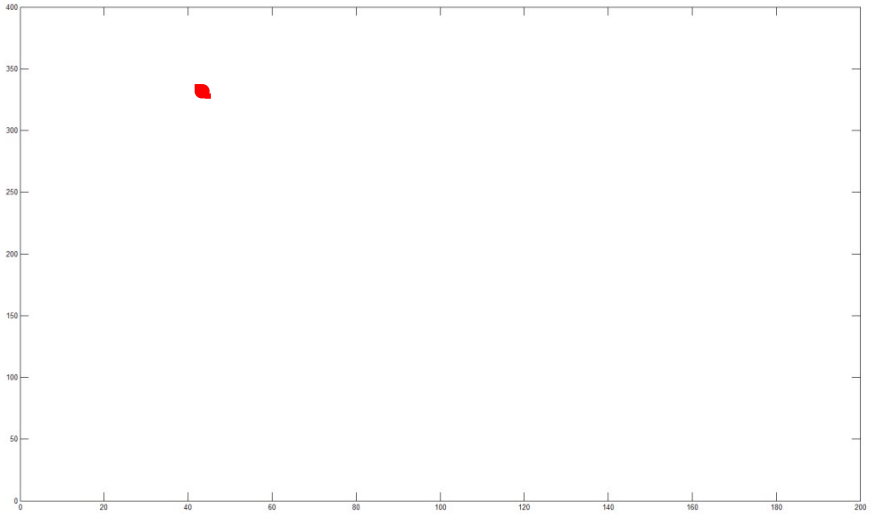
- Tanımlar: skaler, vektörler ve matrisler
- Vektör ve matris hesaplamaları
- Birim, ters matrisler ve determinantlar
- Özvektörler ve vektörlerin iç çarpımı
- Lineer birleşim, bağımlılık, rank
- Matris determinantı, tersi, yalancı tersinir
- Numpy, SciPy

Scaler

- Tek bir gerçek sayı ile tanımlanan bir miktar (değişken) 1 ve 22 gibi

Bir MRI taramasındaki her vokselin yoğunluğu (intensity) değeri

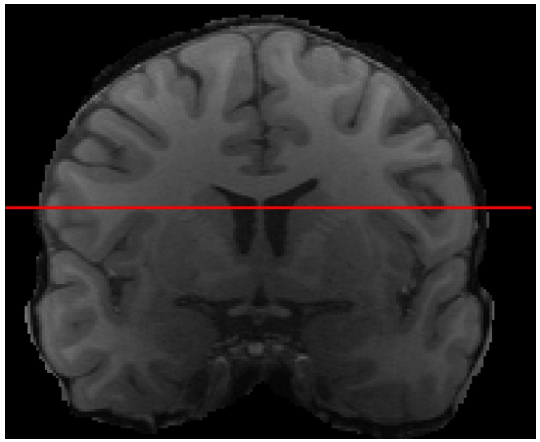


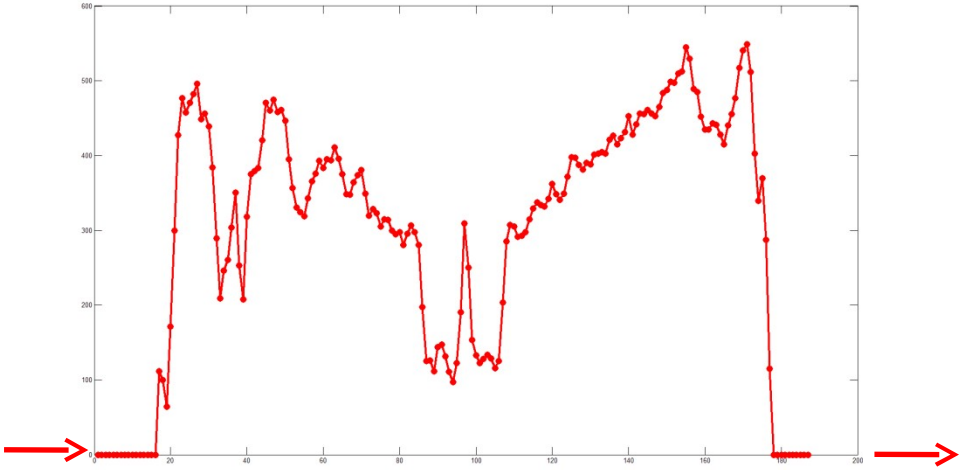


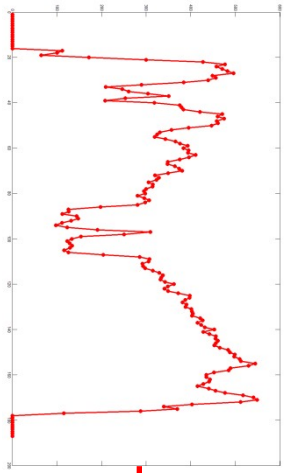
Vektör

- Bir fizik vektörü değil (büyüklük, yön gösteren)

Örneğin:







- Tek satır veya sütundan oluşan sayı dizisi

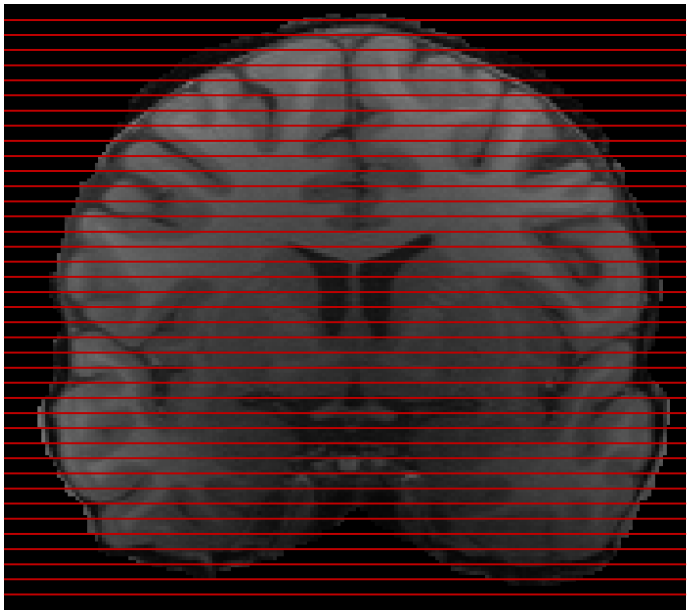
a=

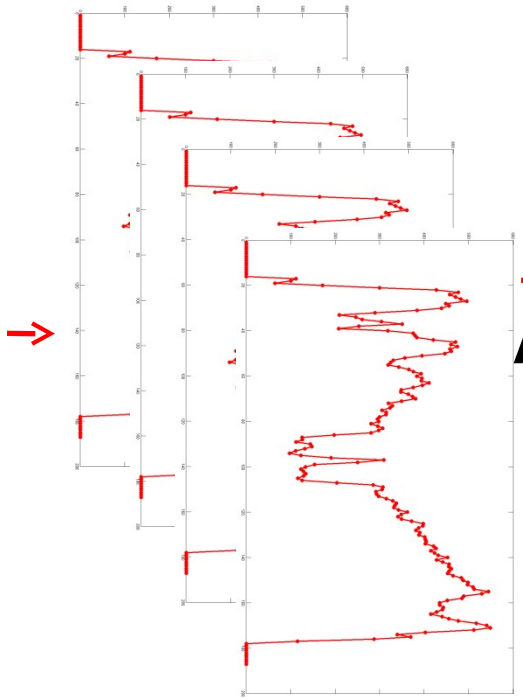
sayılar kolonu
(sütun vektörü)

$$\begin{bmatrix} x1 \\ x2 \\ xn \end{bmatrix}$$

Matrisler

- Vektörlerin satır ve sütunlarda dikdörtgen olarak gösterilmesi
- Farklı zamanlarda aynı vektör yoğunluğu veya aynı anda farklı vokseller hakkında bilgi verebilir.
- Vektör, sadece bir nx1 matrisidir.





$$\mathbf{A}=\begin{bmatrix} x11 & x12 & x13 \\ x21 & x22 & x23 \\ x31 & x32 & x33 \end{bmatrix}$$

Matrisler

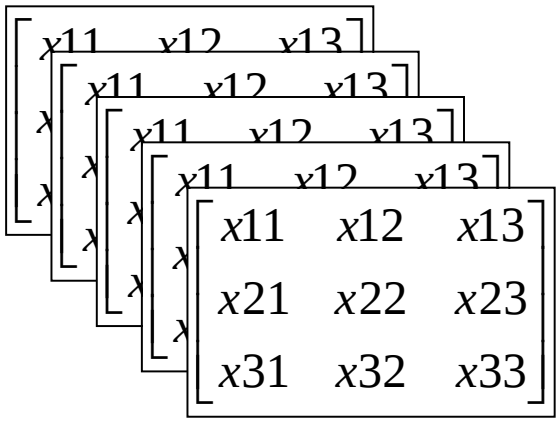
Matris büyüklüğü satır x sütun (R x C) şeklinde gösterilir.
Her bir matris elemanına (d_{ij}) satır, sütun numaraları ile erişilir.

$$\mathbf{D} = \begin{bmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{bmatrix}$$

d_{ij} : i^{th} satır, j^{th} sütun

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

Kare (3 x 3)



3-boyutlu (3 x 3 x 5)

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

dikdörtgen (3 x 2)

Tersyüz etme (transposition)

sütun

$$\mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

satır

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

satır

$$\mathbf{d} = \begin{bmatrix} 3 & 4 & 9 \end{bmatrix}$$

sütun

$$\mathbf{d}^T = \begin{bmatrix} 3 \\ 4 \\ 9 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 4 & 1 \\ 6 & 7 & 4 \end{bmatrix}$$

$$\mathbf{A}^T = \begin{bmatrix} 1 & 5 & 6 \\ 2 & 4 & 7 \\ 3 & 1 & 4 \end{bmatrix}$$

A = A' ise A simetriktir.

Matris işlemleri

Toplama

- Değişme özelliği: **A**+**B**=**B**+**A**
- Birleşme özelliği: (**A**+**B**)+**C**=**A**+(**B**+**C**)

Çıkarma

- Negatiflenmiş matrisle toplama

$$\mathbf{A}-\mathbf{B}=\begin{bmatrix}2 & 4\\ 5 & 3\end{bmatrix}-\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}=\begin{bmatrix}2 & 4\\ 5 & 3\end{bmatrix}+\begin{bmatrix}-1 & -2\\ -3 & -4\end{bmatrix}=\begin{bmatrix}1 & 2\\ 2 & -1\end{bmatrix}$$

Populer bir matris: **X'X**

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} \\ 1 & x_{12} \\ \vdots & \vdots \\ 1 & x_{1n} \end{bmatrix}$$

$$\boldsymbol{X'X} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_{11} & x_{12} & \cdots & x_{1n} \end{bmatrix} \times \begin{bmatrix} 1 & x_{11} \\ 1 & x_{12} \\ \vdots & \vdots \\ 1 & x_{1n} \end{bmatrix} = \begin{bmatrix} n & \sum_{i=1}^{n} x_{1i} \\ \sum_{i=1}^{n} x_{1i} & \sum_{i=1}^{n} x_{1i}^{2} \end{bmatrix}$$

Populer bir matris: **e'e**

$$\boldsymbol{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$$

$$\boldsymbol{e'e} = \begin{bmatrix} e_1 & e_2 & \cdots & e_n \end{bmatrix} \times \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = \sum_{i=1}^{n} e_i^2$$

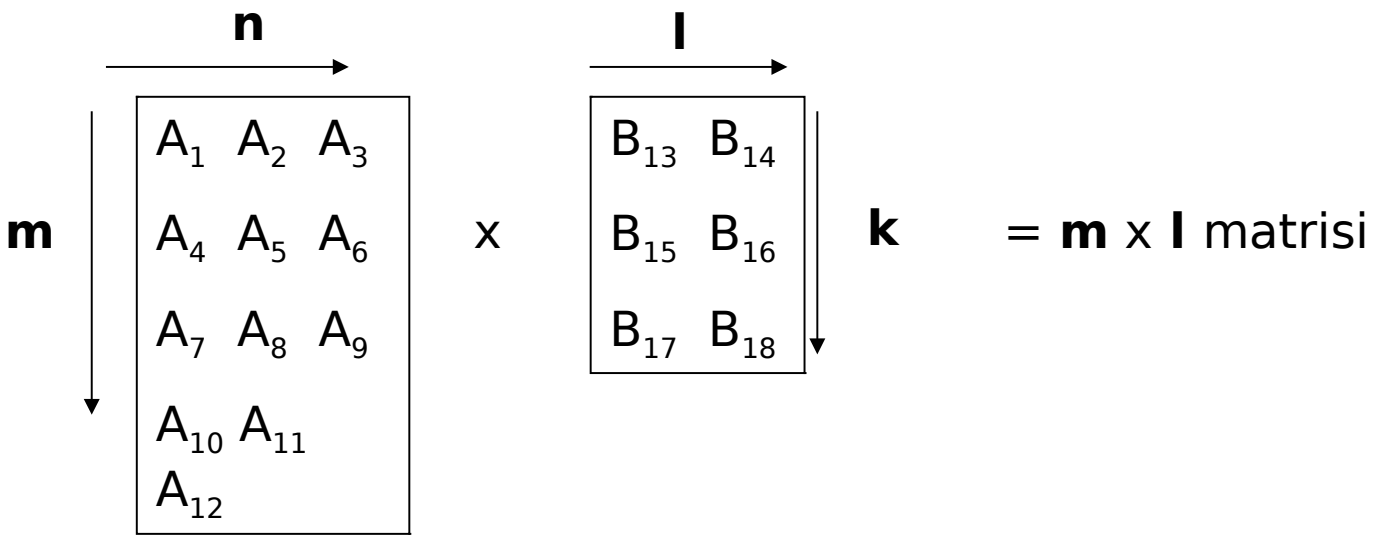
Skaler ile çarpım

Skaler * matris = skaler çarpım

$$\lambda \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} = \begin{pmatrix} \lambda a & \lambda b & \lambda c \\ \lambda d & \lambda e & \lambda f \end{pmatrix}$$

Matris çarpımı

*mxn boyutunda **A** matrisi ile **k**x**l** boyutunda **B** matrisi, n=k eşitliğini sağlaması durumunda çarpılabilir. Sonuç matris mxl boyutludur.*



Matris çarpımı

- Çarpma metodu:

  İlgili satır ve sütunların çarpımlarının toplamı

$$AB = \underset{2\times3}{\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}} \cdot \underset{3\times2}{\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}} = \underset{2\times2}{\begin{pmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32} \end{pmatrix}}$$

Matris çarpımı

- Matris çarpımında değişme özelliği YOKTUR, yani matris sırası önemlidir!
  - $AB \neq BA$
- Matris çarpımının birleşme özelliği vardır.
  - $A(BC)=(AB)C$
- Matris çarpımının dağılma özelliği vardır.
  - $A(B+C)=AB+AC$
  - $(A+B)C=AC+BC$

Birim matris

Sayı çarpımında 1 sayısı ile benzer rol oynayan özel bir matris

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

$$\underset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}} \times \underset{I_3}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}} = \underset{A}{\begin{bmatrix} 1+0+0 & 0+2+0 & 0+0+3 \\ 4+0+0 & 0+5+0 & 0+0+6 \\ 7+0+0 & 0+8+0 & 0+0+9 \end{bmatrix}}$$

nxn boyutlu bir ***A*** matrisi için $\boldsymbol{A\, I_n = I_n\, A = A}$

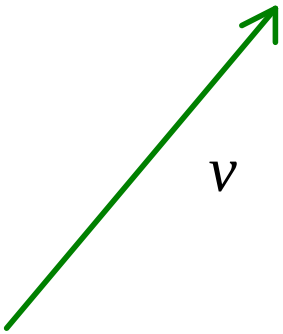
nxm boyutlu bir ***A*** matrisi için $\boldsymbol{I_n\, A = A}$ ve $\boldsymbol{A\, I_m = A}$

Cevaplar her zaman A ise neden bir birim matrisi kullanılır?

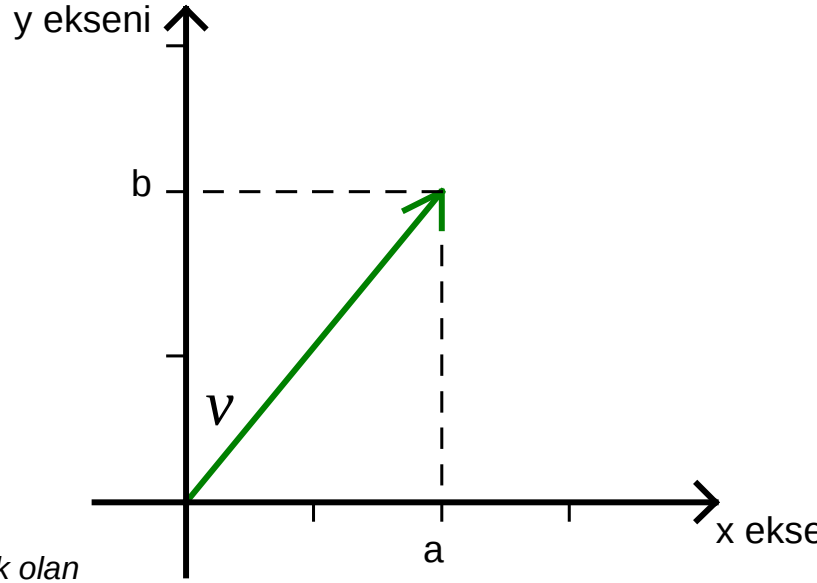
Matrisleri bölemezsiniz, bu yüzden problemleri çözmek için tersini kullanmak zorunda kalabilirsiniz. Birim matris, bu tür hesaplamalarda önemlidir.

Vektör bileşenleri ve taban

- Belirli bir vektör (a b) *bileşenleri* tarafından özetlenebilir. *Taban* (eksen seti; vektörün kendisi bu özel bazın seçiminden bağımsız olabilir).



a ve b, V'nin bileşenleridir.



Taban (Orthonormal base)*: Normu (uzunluğu) 1'e eşit olan birbirlerine dik olan bileşenleri ifade etmek için seçilen vektör seti*

Lineer birleşim & boyutluluk

Vektörel uzay: farklı vektörler tarafından tanımlanan uzaydır.
Lineer birleşim : Bir vektörü başka vektörler cinsinden yazmak için kullanılan bir yöntemdir. herhangi bir v vektör uzayının s alt kümesini oluşturan vektörlerin belli katsayılarla çarpılıp toplanması sonucu elde edilen yeni vektör uzayı s kümesinin vektörlerinin lineer bileşimidir şeklinde ifade edilir.

Örneğin 2-boyutlu uzayda (2,2) ve (3,1) vektörlerinin bir lineer bileşimi
2(2,2) – 3(3,1) = (-5,1) dir.

Bir matris A ($m \times n$) , sütun sayısı (n) kadar çok sayıda vektörde ayrıştırılabilir. Ayrıştırıldığında ; bir vektör, matrisin her sütununu temsil edebilir. n-vektör sütunu topluluğu, A matrisine uygun bir vektör uzayını tanımlar. Benzer şekilde bileşenleri bir tabanda (*base*) ifade edilen vektörlerin bir araya gelmesiyle A matrisi oluşmuş olur.

Örneğin 3-boyutlu uzayda (1,0,0), (0,1,0) ve (0,0,1) tabanlarından bir lineer bileşimi 3(1,0,0) - 2(0,1,0) + (0,0,1) = (3,-2,1) dir.

Lineer bağımlılık ve rank

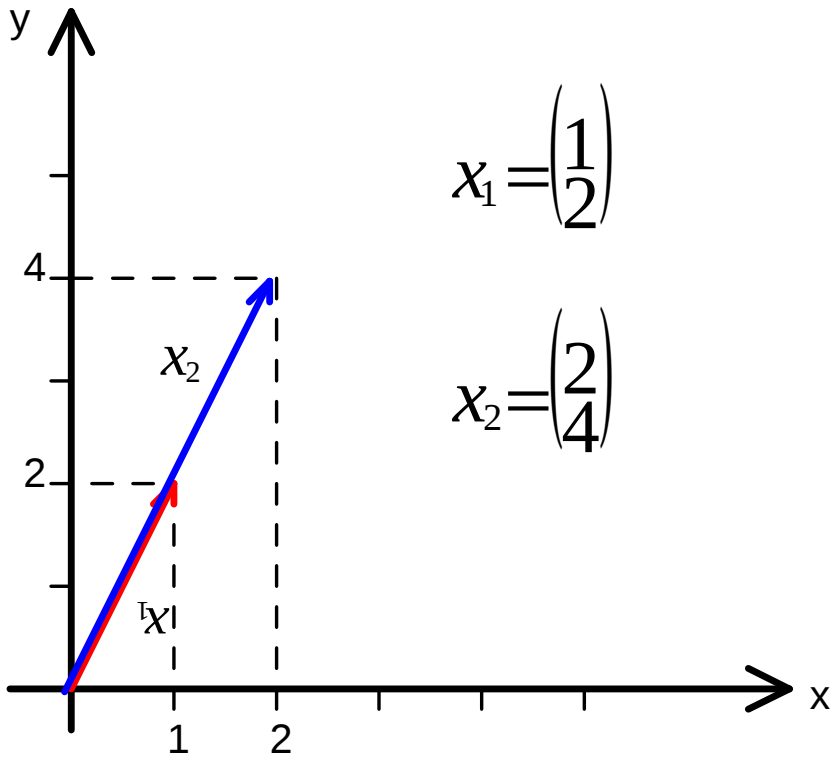
Bir matrisin satırları veya sütunları arasında doğrusal bir bağılılık varsa, o zaman matrisin rankı (vektör uzayının boyutlarının sayısı) sütun sayısına eşit olmaz. Matrisin rank eksik (rank-deficient) olduğu söylenir.

Örneğin

$$X = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$$

x1 ve x2 üst üste getirilebilir.
x2 = 2 x1.

Matris rankı 1.



$$x_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$$x_2 = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$$

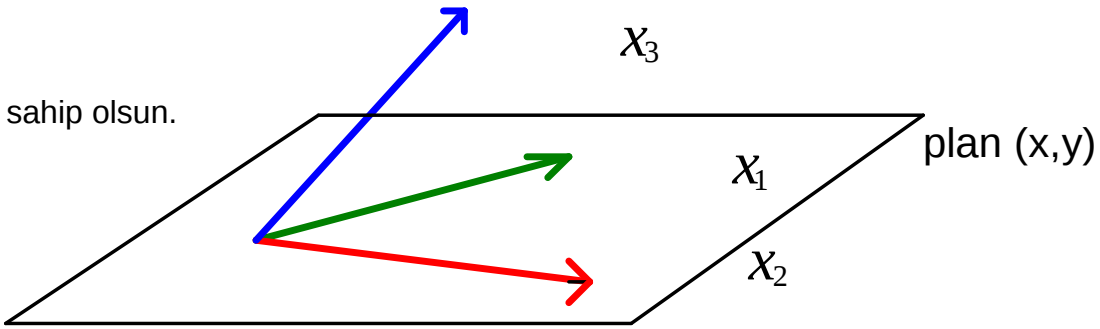
Lineer bağımlılık ve rank

• Bir matrisin rankı, bu matris tarafından tanımlanan vektör uzayının boyutuna karşılık gelir. Birbirlerinden doğrusal olarak bağımsız olan matris tarafından tanımlanan vektör sayısına karşılık gelir.

• Doğrusal olarak bağımsız vektörler (*Linealy independent*), diğerlerinin doğrusal bir kombinasyonu ile ifade edilemez.

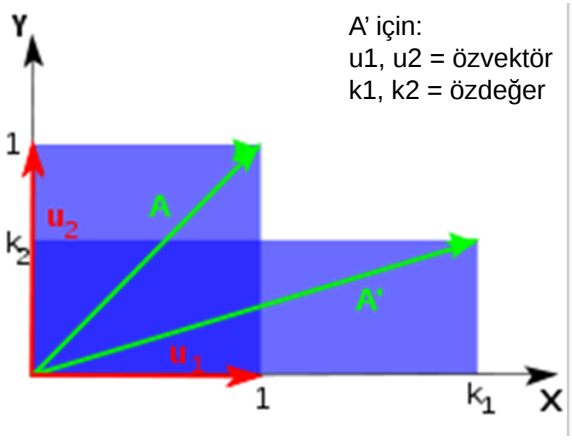
Not. Lineer bağımsız vektörler ortogonal değildir.
x1, x2 ve x3 lineer bagımsız vektörler olsun.

Vektörler x1 ve x2 (x,y) duzelemini tanımlasın.
Vektör x3 z-ekseninde sıfır olmayan bir bileşene sahip olsun.
X3, x1 veya x2'ye dik degildir.

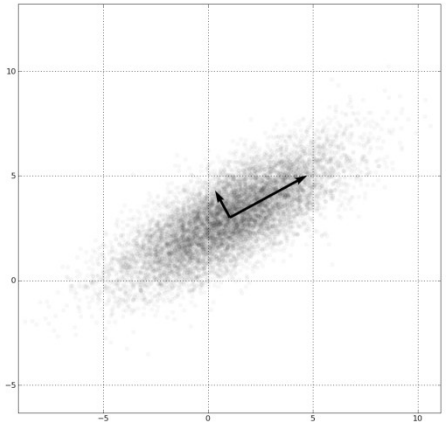


Özdeğerler ve özvektörler

Özdeğerler, bir matrisin orijinal yapısını görmek için kullanılan alternatif bir yoldur.
Özdeğer kavramını açıklamak için öncelikle özvektör kavramı ele alınsın.
*Bazı vektörler bir **A** matrisi ile çarpıldıkları zaman yön değiştirir, bazıları ise değiştirmezler. Bazı özel **x** vektörleri, A**x** vektörü ile aynı yönde kalmaktadır. İşte bu vektörlere "özvektörler" denir.*
*Bir özvektörün **A** matrisi ile çarpımı olan A**x** vektörü, orijinal **x** vektörünün* $\lambda \in IR$ *olmak üzere* λ*katıdır.*
(Ax= x) λ
Not: bir matris rank eksik (rank-deficient) ise, öz değerlerinden en az biri sıfırdır.



A' için:
u1, u2 = özvektör
k1, k2 = özdeğer



Temel Bileşenler Analizi (TBA), bir boyut azaltma işlemidir. Eğer bir dizi değişken mevcut ise (muhtemelen çok sayıda) ve değişkenlerden bazılarını fazlalık ki burada bahsedilen aynı yapı içerisinde birbirleriyle ilişkili olan değişkenler olduğuna inanılıyorsa TBA'nin kullanımına uygundur. Bu sistem veri seti için yeni bir koordinat sistemi seçip en büyük varyansa sahip olanı ilk eksene yerleştirir, ikinci en büyük varyansa sahip olanı ikinci eksene yerleştirerek devam eden bir süreç izler.

Vektörel çarpım

iki vektör: $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$

iç çarpım = skaler

İç çarpım $X^T Y$ sonucu skaler
(1xn) (nx1)

$$\mathbf{x}^T\mathbf{y} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = x_1y_1 + x_2y_2 + x_3y_3 = \sum_{i=1}^{3} x_i y_i$$

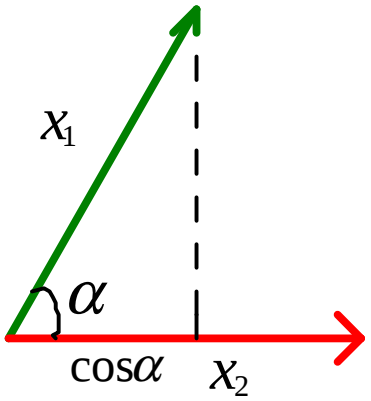
dış çarpım = matris

$$\mathrm{xy}^T = \begin{bmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \\ \mathrm{x}_3 \end{bmatrix} \begin{bmatrix} \mathrm{y}_1 & \mathrm{y}_2 & \mathrm{y}_3 \end{bmatrix} = \begin{bmatrix} \mathrm{x}_1\mathrm{y}_1 & \mathrm{x}_1\mathrm{y}_2 & \mathrm{x}_1\mathrm{y}_3 \\ \mathrm{x}_2\mathrm{y}_1 & \mathrm{x}_2\mathrm{y}_2 & \mathrm{x}_2\mathrm{y}_3 \\ \mathrm{x}_3\mathrm{y}_1 & \mathrm{x}_3\mathrm{y}_2 & \mathrm{x}_3\mathrm{y}_3 \end{bmatrix}$$

Dış çarpım XYT sonucu bir matris
(nx1) (1xn)

Vektörlerin skaler çarpımı

İki vektörün skaler çarpımı, bir vektörün diğerine projeksiyonuna eşdeğerdir.



$$\vec{x1} \bullet \vec{x2} = |\vec{x_1}| \,.|\vec{x_2}| \,.\, \cos\alpha$$

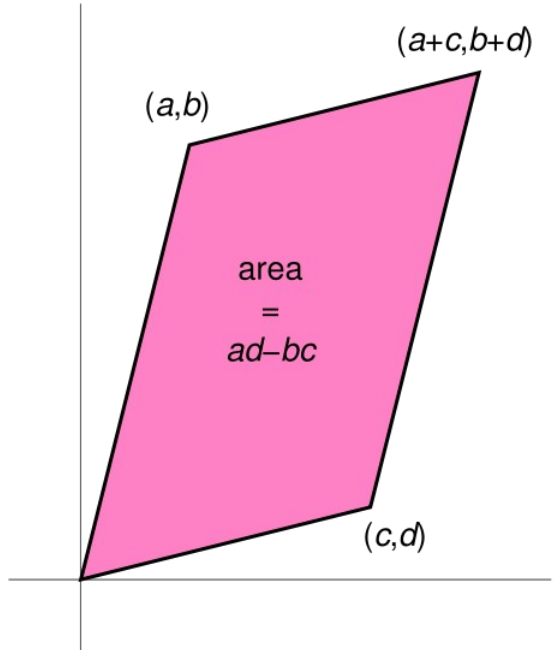
İki vektör ortogonal ise, skaler çarpımı sıfırdır: birinin diğerine projeksiyonu sıfır olacaktır.

i.i = j.j = k.k = 1

i.j = i.k = j.k = 0

Matris determinantı

- Determinant, kare matrisleri bir sayıya eşleyen fonksiyondur.
- Determinant fonksiyonunun, kare matrisi eşlediği o sayıya matrisin determinantı denir.
- A matrisinin determinantı, detA veya |A| biçiminde gösterilir.
- |A|, matrislerde mutlak değer anlamına gelmez. |A| sıfır veya negatif de olabilir.
  - Sarrus Kuralı
  - İşaretli Minör (Kofaktör)
  - Gauss Eleminasyon Yöntemi
- Determinant, vektör uzayındaki matrisin işgal ettiği "hacim" hakkında bir fikir verir.
- Bir A matrisinin tersinin alınabilmesi için **det(A)≠0** olmalı



Matris determinantı

- Bir matrisin determinantı, eğer ve sadece matrisin sütunları arasında doğrusal bir ilişki varsa sıfır olur.
- O zaman bir A matrisinin rankı, sıfır olmayan bir determinanta sahip olan A'nın en büyük kare alt matrisinin boyutu olarak tanımlanabilir.

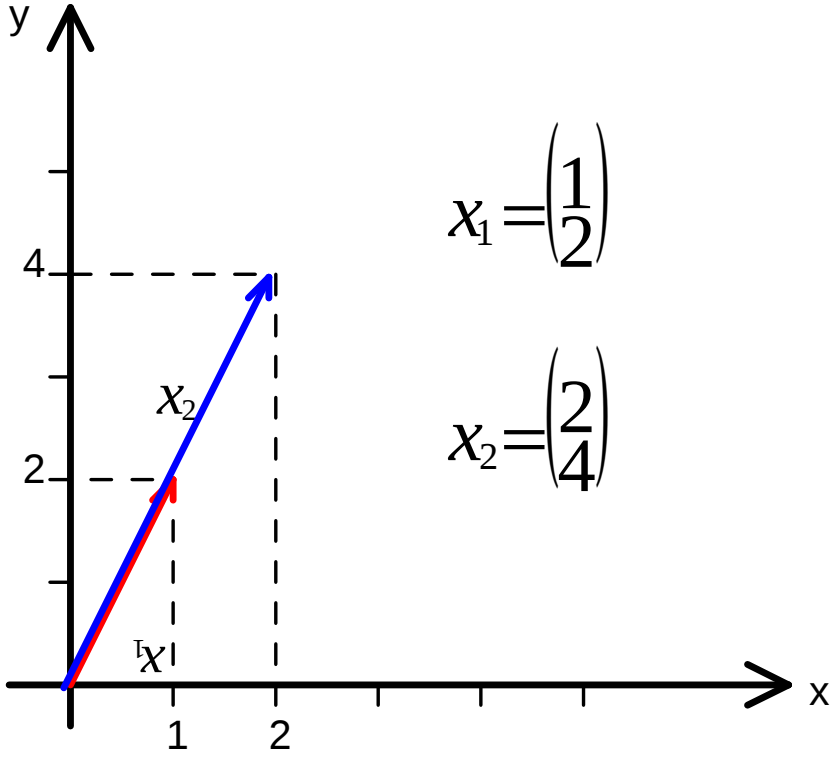
$$X = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$$

x1 ve x2 üst üste getirilebilir.
x2 = 2 x1.

matris X'in determinantı sıfırdır.

Determinantı sıfır olmayan en büyük kare alt matrisi 1x1'dir.

Dolayısıyla rank(X)=1.



$$x_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$$x_2 = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$$

Matris tersi

- **Tanım.** A matrisi, aşağıdaki gibi bir ***B*** matrisi varsa tersi alınabilirdir.

$$A\,B = B\,A = I_n$$

$$\begin{bmatrix} 1 & 1 \\ -1 & 2 \end{bmatrix} \times \begin{bmatrix} \frac{2}{3} & \frac{-1}{3} \\ \frac{1}{3} & \frac{1}{3} \end{bmatrix} = \begin{bmatrix} \frac{2}{3}+\frac{1}{3} & \frac{-1}{3}+\frac{1}{3} \\ \frac{-2}{3}+\frac{2}{3} & \frac{1}{3}+\frac{2}{3} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Notasyon. A^{-1} , ***A****'nın tersi*

$$A\,A^{-1} = A^{-1}\,A = I_n\,.$$

- ***A*** tersi alınabilirse, A^{-1} de tersi alınabilir. $(A^T)^{-1} = (A^{-1})^T$

Matris tersi

- Kare matris A:

$$A = \begin{pmatrix} x_{1,1} & \dots & x_{1,j} \\ \vdots & \ddots & \vdots \\ x_{i,1} & \dots & x_{i,j} \end{pmatrix}$$

- Tersi:

$$A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} \operatorname{cof}(A, x_{1,1}) & \dots & \operatorname{cof}(A, x_{1,j}) \\ \vdots & \ddots & \vdots \\ \operatorname{cof}(A, x_{i,1}) & \dots & \operatorname{cof}(A, x_{i,j}) \end{pmatrix}^T$$

- 2x2 matris

$$\mathbf{A}^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

Bir matrsin tersi alınabilirse, determinantı sıfırdan farklıdır (kare matris olmalı). Tersinir olmayan bir matrisin tekil (singular) olduğu söylenir.

Ters matris özellikleri

$$\left(\boldsymbol{AB}\right)^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$$

$$\left(\boldsymbol{A}'\right)^{-1} = \left(\boldsymbol{A}^{-1}\right)'$$

$$\left(\boldsymbol{A}^{-1}\right)^{-1} = \boldsymbol{A}$$

Yalancı tersinir (pseudoinverse)

- Eğer matrisiniz kare değilse (bkz: kare matris) bu durumda ters matrisi elde edemezsiniz.
- Bunun yerine, sanki ters matrismiş gibi bir matris veren moore–penrose yöntemi kullanılabilir.
- En küçük kareler yönetmi ile hesaplanır. Bunun sonucunda elde edilen pseudoinverse matris, ters matrisin bütün özelliklerini içermese de fikir verir.
- Genelleştirilmiş ters matris de denir.

Lineer sistem denklemleri

- Matrislerin en önemli uygulamalarından biri, elektrik devreleri, istatistikler ve diferansiyel denklemler için sayısal yöntemler dahil olmak üzere birçok farklı problemde ortaya çıkan doğrusal denklem sistemlerini çözmek içindir.

$$
\begin{aligned}
a_{11}x_1 + \ldots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + \ldots + a_{2n}x_n &= b_2 \\
&\vdots \\
a_{m1}x_1 + \ldots + a_{mn}x_n &= b_m
\end{aligned}
$$

m adet denklem n-adet bilinmeyen

Scientific Python

- Ekstra özellikler gerekli:
  - hızlı, çok boyutlu diziler
  - güvenilir, test edilmiş bilimsel fonksiyonların kütüphaneleri
  - görselleştirme araçları
- NumPy, neredeyse her bilimsel Python uygulamasının veya modülünün çekirdeğinde yer almaktadır çünkü vektörel bir biçimde manipüle edilebilen hızlı bir N-d array veri türü sağlar.

Arrays – Numerical Python (Numpy)

- Küçük boyutlu bir boyutlu verilerin depolanması için listeler yeterli

```
>>> a = [1,3,5,7,9]
>>> print(a[2:4])
[5, 7]
>>> b = [[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
>>> print(b[0])
[1, 3, 5, 7, 9]
>>> print(b[1][2:4])
[6, 8]
```

```
>>> a = [1,3,5,7,9]
>>> b = [3,5,6,7,9]
>>> c = a + b
>>> print c
[1, 3, 5, 7, 9, 3, 5, 6, 7, 9]
```

- Fakat, aritmetik operatörler direk uygulanamaz (+, -, *, /, …)
- Aritmetik operasyonlara uygun çok boyutlu veriler için
- **Numpy**

```
>>> import numpy
```

Numpy – N-boyutlu Array manipulasyonları

Python ile bilimsel hesaplama için gerekli temel kütüphaneye NumPy denir. Bu Açık Kaynak kütüphanesi şunları içerir:

- güçlü bir N boyutlu dizi nesnesi
- gelişmiş dizi dilimleme (slicing) yöntemleri (dizi elemanlarını seçmek için)
- uygun dizi yeniden şekillendirme (reshaping) yöntemleri

ve hatta sayısal rutinleri olan 3 kütüphaneyi içerir:

- temel lineer cebir fonksiyonları
- temel Fourier dönüşümleri
- karmaşık rasgele sayı üretme

Numpy – array oluşturma

- numpy array oluşturmanın birçok yolu var
  - Python list veya tuples
  - arange, linspace vb. gibi numpy dizileri oluşturmaya adanmış fonksiyonları kullanarak
  - Dosyadan veri okuyarak

Numpy – array oluşturma

- list kullanrak
  - numpy.array

```
# as vectors from lists
>>> a = numpy.array([1,3,5,7,9])
>>> b = numpy.array([3,5,6,7,9])
>>> c = a + b
>>> print c
[4, 8, 11, 14, 18]

>>> type(c)
(<type 'numpy.ndarray'>)

>>> c.shape
(5,)
```

Numpy – matris oluşturma

```
>>> l = [[1, 2, 3], [3, 6, 9], [2, 4, 6]]  # create a list
>>> a = numpy.array(l)  # convert a list to an array
>>>print(a)
[[1 2 3]
 [3 6 9]
 [2 4 6]]
>>> a.shape
(3, 3)
>>> print(a.dtype)  # get type of an array
int64

# or directly as matrix
>>> M = array([[1, 2], [3, 4]]
>>> M.shape
(2,2)
>>> M.dtype
dtype('int64')
```

```
#only one type
>>> M[0,0] = "hello"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for long() with base 10: 'hello‘

>>> M = numpy.array([[1, 2], [3, 4]], dtype=complex)
>>> M
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

Numpy – Matris kullanımı

```
>>> print(a)
[[1 2 3]
 [3 6 9]
 [2 4 6]]
>>> print(a[0])  # this is just like a list of lists
[1 2 3]
>>> print(a[1, 2])  # arrays can be given comma separated indices
9
>>> print(a[1, 1:3])  # and slices
[6 9]
>>> print(a[:,1])
[2 6 4]
>>> a[1, 2] = 7
>>> print(a)
[[1 2 3]
 [3 6 7]
 [2 4 6]]
>>> a[:, 0] = [0, 9, 8]
>>> print(a)
[[0 2 3]
 [9 6 7]
 [8 4 6]]
```

Numpy – array oluşturma

- Fonksiyonlar yoluyla

```
>>> x = arange(0, 10, 1) # arguments: start, stop, step
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> numpy.linspace(0, 10, 25)
array([  0.        ,   0.41666667,   0.83333333,   1.25      ,
         1.66666667,   2.08333333,   2.5       ,   2.91666667,
         3.33333333,   3.75      ,   4.16666667,   4.58333333,
         5.        ,   5.41666667,   5.83333333,   6.25      ,
         6.66666667,   7.08333333,   7.5       ,   7.91666667,
         8.33333333,   8.75      ,   9.16666667,   9.58333333,  10.        ])
>>> numpy.logspace(0, 10, 10, base=numpy.e)
array([  1.00000000e+00,   3.03773178e+00,   9.22781435e+00,
         2.80316249e+01,   8.51525577e+01,   2.58670631e+02,
         7.85771994e+02,   2.38696456e+03,   7.25095809e+03,
         2.20264658e+04])
```

Numpy – array oluşturma

```
# a diagonal matrix
>>> numpy.diag([1,2,3])
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])

>>> b = numpy.zeros(5)
>>> print(b)
[ 0.  0.  0.  0.  0.]
>>> b.dtype
dtype('float64')
>>> n = 1000
>>> my_int_array = numpy.zeros(n, dtype=numpy.int)
>>> my_int_array.dtype
dtype('int32')

>>> c = numpy.ones((3,3))
>>> c
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

Numpy – array oluşturma ve kullanımı

```
>>> d = numpy.arange(5)  # just like range()
>>> print(d)
[0 1 2 3 4]

>>> d[1] = 9.7
>>> print(d)  # arrays keep their type even if elements changed
[0 9 2 3 4]

>>> print(d*0.4)  # operations create a new array, with new type
[ 0.   3.6  0.8  1.2  1.6]

>>> d = numpy.arange(5, dtype=numpy.float)
>>> print(d)
[ 0.  1.  2.  3.  4.]

>>> numpy.arange(3, 7, 0.5)  # arbitrary start, stop and step
array([ 3. , 3.5, 4. , 4.5, 5. , 5.5, 6. , 6.5])
```

Numpy – array oluşturma ve kullanımı

```
>>> x, y = numpy.mgrid[0:5, 0:5] # similar to meshgrid in MATLAB
>>> x
array([[0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1],
       [2, 2, 2, 2, 2],
       [3, 3, 3, 3, 3],
       [4, 4, 4, 4, 4]])
# random data
>>> numpy.random.rand(5,5)
array([[ 0.51531133,  0.74085206,  0.99570623,  0.97064334,  0.5819413 ],
       [ 0.2105685 ,  0.86289893,  0.13404438,  0.77967281,  0.78480563],
       [ 0.62687607,  0.51112285,  0.18374991,  0.2582663 ,  0.58475672],
       [ 0.72768256,  0.08885194,  0.69519174,  0.16049876,  0.34557215],
       [ 0.93724333,  0.17407127,  0.1237831 ,  0.96840203,  0.52790012]])
```

Numpy – array oluşturma

- Dosyadan okuma

```
>>> os.system('head DeBilt.txt')
"Stn", "Datum", "Tg", "qTg", "Tn", "qTn", "Tx", "qTx"
001, 19010101,   -49, 00,   -68, 00,   -22, 40
0
0
0
0
0
0
0
>
>
(25568, 8)
```

```
>>> numpy.savetxt('datasaved.txt', data)
>>> os.system('head datasaved.txt')
1.0000000000000000e+00 1.9010101000000000e+07 -4.9000000000000000e+01
0.0000000000000000e+00 -6.8000000000000000e+01 0.0000000000000000e+00
-2.2000000000000000e+01 4.0000000000000000e+01
1.0000000000000000e+00 1.9010102000000000e+07 -2.1000000000000000e+01
0.0000000000000000e+00 -3.6000000000000000e+01 3.0000000000000000e+01
-1.3000000000000000e+01 3.0000000000000000e+01
1.0000000000000000e+00 1.9010103000000000e+07 -2.8000000000000000e+01
0.0000000000000000e+00 -7.9000000000000000e+01 3.0000000000000000e+01
-5.0000000000000000e+00 2.0000000000000000e+01
```

Numpy – array oluşturma

```
>>> M = numpy.random.rand(3,3)
>>> M
array([[ 0.84188778,  0.70928643,  0.87321035],
       [ 0.81885553,  0.92208501,  0.873464  ],
       [ 0.27111984,  0.82213106,  0.55987325]])
>>>
>>> numpy.save('saved-matrix.npy', M)
>>> numpy.load('saved-matrix.npy')
array([[ 0.84188778,  0.70928643,  0.87321035],
       [ 0.81885553,  0.92208501,  0.873464  ],
       [ 0.27111984,  0.82213106,  0.55987325]])
>>>
>>> os.system('head saved-matrix.npy')
NUMPYF{'descr': '<f8', 'fortran_order': False, 'shape': (3, 3), }
Ï<
£¾ðê?sy²æ?$÷ÒVñë?Ù4ê?%dn¸í?Ã[Äjóë?Ä,ZÑ?Ç
ÎåNê?ó7L{êá?0
>>>
```

Numpy - ndarray

- NumPy'nin ana nesnesi “ndarray” adı verilen homojen çok boyutlu bir dizidir.
  - Bu, bir tamsayı dizisi ile indekslenen, aynı tipte bir elementler tablosu (genellikle sayılar). Çok boyutlu dizilerin tipik örnekleri, vektörleri, matrisleri, görüntüleri ve elektronik tabloları içerir.
  - Boyutlar genellikle eksen olarak adlandırılır, eksen sayısı rankı verir.

[7, 5, -1]	rankı 1'dir. 3-uzunluklu bir eksene sahip
[[1.5, 0.2, -3.7] , [0.1, 1.7, 2.9]]	rankı 2'dir. 2-boyutludur, ilk eksen 3 uzunluklu, ikinci eksen 3 uzunlukludur (matris 2 satır 3 sütundan oluşmakta

Numpy – ndarray attributes

- **ndarray.ndim**
  - the number of axes (dimensions) of the array i.e. the rank.
- **ndarray.shape**
  - the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with n rows and m columns, shape will be (n,m). The length of the shape tuple is therefore the rank, or number of dimensions, ndim.
- **ndarray.size**
  - the total number of elements of the array, equal to the product of the elements of shape.
- **ndarray.dtype**
  - an object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. NumPy provides many, for example bool_, character, int_, int8, int16, int32, int64, float_, float8, float16, float32, float64, complex_, complex64, object_.
- **ndarray.itemsize**
  - the size in bytes of each element of the array. E.g. for elements of type float64, itemsize is 8 (=64/8), while complex32 has itemsize 4 (=32/8) (equivalent to ndarray.dtype.itemsize).
- **ndarray.data**
  - the buffer containing the actual elements of the array. Normally, we won't need to use this attribute because we will access the elements in an array using indexing facilities.

Numpy – array oluşturma ve kullanımı

Two ndarrays are mutable and may be views to the same memory:

```
>>> x = np.array([1,2,3,4])
>>> y = x
>>> x is y
True
>>> id(x), id(y)
(139814289111920, 139814289111920)
>>> x[0] = 9
>>> y
array([9, 2, 3, 4])

>>> x[0] = 1
>>> z = x[:]
>>> x is z
False
>>> id(x), id(z)
(139814289111920, 139814289112080)
>>> x[0] = 8
>>> z
array([8, 2, 3, 4])
```

```
>>> x = np.array([1,2,3,4])
>>> y = x.copy()
>>> x is y
False
>>> id(x), id(y)
(139814289111920, 139814289111840)
>>> x[0] = 9
>>> x
array([9, 2, 3, 4])
>>> y
array([1, 2, 3, 4])
```

Numpy – array oluşturma ve kullanımı

```
>>> a = numpy.arange(4.0)
>>> b = a * 23.4
>>> c = b/(a+1)
>>> c += 10
>>> print c
[ 10.   21.7  25.6  27.55]

>>> arr = numpy.arange(100, 200)
>>> select = [5, 25, 50, 75, -5]
>>> print(arr[select])  # can use integer lists as indices
[105, 125, 150, 175, 195]

>>> arr = numpy.arange(10, 20 )
>>> div_by_3 = arr%3 == 0  # comparison produces boolean array
>>> print(div_by_3)
[ False False  True False False  True False False  True False]
>>> print(arr[div_by_3])  # can use boolean lists as indices
[12 15 18]

>>> arr = numpy.arange(10, 20) . reshape((2,5))
[[10 11 12 13 14]
 [15 16 17 18 19]]
```

Numpy – array metotları

```
>>> arr.sum()
145
>>> arr.mean()
14.5
>>> arr.std()
2.8722813232690143
>>> arr.max()
19
>>> arr.min()
10
>>> div_by_3.all()
False
>>> div_by_3.any()
True
>>> div_by_3.sum()
3
>>> div_by_3.nonzero()
(array([2, 5, 8]),)
```

Numpy – array metotları - sıralama

```
>>> arr = numpy.array([4.5, 2.3, 6.7, 1.2, 1.8, 5.5])
>>> arr.sort()  # acts on array itself
>>> print(arr)
[ 1.2  1.8  2.3  4.5  5.5  6.7]

>>> x = numpy.array([4.5, 2.3, 6.7, 1.2, 1.8, 5.5])
>>> numpy.sort(x)
array([ 1.2,  1.8,  2.3,  4.5,  5.5,  6.7])

>>> print(x)
[ 4.5  2.3  6.7  1.2  1.8  5.5]

>>> s = x.argsort()
>>> s
array([3, 4, 1, 0, 5, 2])
>>> x[s]
array([ 1.2,  1.8,  2.3,  4.5,  5.5,  6.7])
```

Numpy – array fonksiyonları

- Most array methods have equivalent functions

```
>>> arr.sum()
45
>>> numpy.sum(arr)
45
```

- Ufuncs provide many element-by-element math, trig., etc. operations
  - e.g., add(x1, x2), absolute(x), log10(x), sin(x), logical_and(x1, x2)

- See http://numpy.scipy.org

Numpy – array operasyonları

```
>>> a = array([[1.0, 2.0], [4.0, 3.0]])
>>> print a
[[ 1. 2.]
 [ 3. 4.]]

>>> a.transpose()
array([[ 1., 3.],
       [ 2., 4.]])

>>> inv(a)
array([[-2. , 1. ],
       [ 1.5, -0.5]])

>>> u = eye(2) # unit 2x2 matrix; "eye" represents "I"

>>> u
array([[ 1., 0.],
       [ 0., 1.]])

>>> j = array([[0.0, -1.0], [1.0, 0.0]])

>>> dot (j, j) # matrix product
array([[-1., 0.],
       [ 0., -1.]])
```

Numpy – statistics

In addition to the mean, var, and std functions, NumPy supplies several other methods for returning statistical features of arrays. The median can be

```
>>> a = np.array([1, 4, 3, 8, 9, 2, 3], float)
>>> np.median(a)
3.0
```

The correlation coefficient (iki rassal arasındaki lineer bagımlılık) for multiple variables observed at multiple instances can be found for arrays of the form [[x1, x2, ...], [y1, y2, ...], [z1, z2, ...], ...] where x, y, z are different observables and the numbers indicate the observation times:

```
>>> a = np.array([[1, 2, 1, 3], [5, 3, 1, 8]], float)
>>> c = np.corrcoef(a)
>>> c
array([[ 1.        ,  0.72870505],
       [ 0.72870505,  1.        ]])
```

Here the return array c[i,j] gives the correlation coefficient for the ith and jth observables. Similarly, the covariance for data can be found::

```
>>> np.cov(a)
array([[ 0.91666667,  2.08333333],
       [ 2.08333333,  8.91666667]])
```

Numpy – arrays, matrisler

For **two dimensional** arrays NumPy defined a special matrix class in module matrix. Objects are created either with matrix() or mat() or converted from an array with method asmatrix().

```
>>> import numpy
>>> m = numpy.mat([[1,2],[3,4]])
or
>>> a = numpy.array([[1,2],[3,4]])
>>> m = numpy.mat(a)
or
>>> a = numpy.array([[1,2],[3,4]])
>>> m = numpy.asmatrix(a)
```

Note that the statement m = mat(a) creates a copy of array 'a'.
Changing values in 'a' will not affect 'm'.
On the other hand, method m = asmatrix(a) returns a new reference to the same da
Changing values in 'a' will affect matrix 'm'.

Numpy – matrisler

Array and matrix operations may be quite different!

```
>>> a = array([[1,2],[3,4]])
>>> m = mat(a) # convert 2-d array to matrix
>>> m = matrix([[1, 2], [3, 4]])
>>> a[0]            # result is 1-dimensional
array([1, 2])
>>> m[0]            # result is 2-dimensional
matrix([[1, 2]])
>>> a*a             # element-by-element multiplication
array([[ 1, 4], [ 9, 16]])
>>> m*m             # (algebraic) matrix multiplication
matrix([[ 7, 10], [15, 22]])
>>> a**3            # element-wise power
array([[ 1, 8], [27, 64]])
>>> m**3            # matrix multiplication m*m*m
matrix([[ 37, 54], [ 81, 118]])
>>> m.T             # transpose of the matrix
matrix([[1, 3], [2, 4]])
>>> m.H             # conjugate transpose (differs from .T for complex matrices)
matrix([[1, 3], [2, 4]])
>>> m.I             # inverse matrix
matrix([[-2. , 1. ], [ 1.5, -0.5]])
```

Numpy – matrisler

- Operator *, dot(), and multiply():
  - For array, '*' **means element-wise multiplication**, and the dot() function is used for matrix multiplication.
  - For matrix, '*'**means matrix multiplication**, and the multiply() function is used for element-wise multiplication.
- Handling of vectors (rank-1 arrays)
  - For array, the vector shapes 1xN, Nx1, and N are all different things. Operations like A[:,1] return a rank-1 array of shape N, not a rank-2 of shape Nx1. Transpose on a rank-1 array does nothing.
  - For matrix, rank-1 arrays are always upgraded to 1xN or Nx1 matrices (row or column vectors). A[:,1] returns a rank-2 matrix of shape Nx1.
- Handling of higher-rank arrays (rank > 2)
  - array objects can have rank > 2.
  - matrix objects always have exactly rank 2.
- Convenience attributes
  - array has a .T attribute, which returns the transpose of the data.
  - matrix also has .H, .I, and .A attributes, which return the conjugate transpose, inverse, and asarray() of the matrix, respectively.
- Convenience constructor
  - The array constructor takes (nested) Python sequences as initializers. As in array([[1,2,3],[4,5,6]]).
  - The matrix constructor additionally takes a convenient string initializer. As in matrix("[1 2 3; 4 5 6]")

Numpy – array mathematics

```
>>> a = np.array([1,2,3], float)
>>> b = np.array([5,2,6], float)
>>> a + b
array([6., 4., 9.])
>>> a – b
array([-4., 0., -3.])
>>> a * b
array([5., 4., 18.])
>>> b / a
array([5., 1., 2.])
>>> a % b
array([1., 0., 3.])
>>> b**a
array([5., 4., 216.])

>>> a = np.array([[1, 2], [3, 4],
>>> b = np.array([-1, 3], float)
>>> a
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.]])
>>> b
array([-1.,  3.])
>>> a + b
array([[ 0.,  5.],
       [ 2.,  7.],
       [ 4.,  9.]])
```

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> b = np.array([-1, 3], float)

>>> a * a
array([[  1.,   4.],
       [  9.,  16.],
       [ 25.,  36.]])
>>> b * b
array([ 1.,  9.])
>>> a * b
array([[ -1.,   6.],
       [ -3.,  12.],
       [ -5.,  18.]])
>>>
```

Numpy – array mathematics

```
>>> A = np.array([
>>> v1 = arange(0,
>>> A
array([[ 0, 1, 2,
[10, 11, 12, 13, 1
[20, 21, 22, 23, 2
[30, 31, 32, 33, 3
[40, 41, 42, 43, 4
>>> v1
array([0, 1, 2, 3,
>>> np.dot(A,A)
array([[ 300,  310
       [1300, 1360
       [2300, 2410
       [3300, 3460
       [4300, 4510
>>>
>>> np.dot(A,v1)
array([ 30, 130, 2
>>> np.dot(v1,v1)
30
>>>
```

```
Alternatively, we can cast the array objects to the type matrix. This changes
the behavior of the standard arithmetic operators +, -, * to use matrix
algebra.
>>> M = np.matrix(A)
>>> v = np.matrix(v1).T
>>> v
matrix([[0],
        [1],
        [2],
        [3],
        [4]])
>>> M*v
matrix([[ 30],
        [130],
        [230],
        [330],
        [430]])
>>> v.T * v   # inner product
matrix([[30]])
# standard matrix algebra applies
>>> v + M*v
matrix([[ 30],
        [131],
        [232],
        [333],
        [434]])
```

Broadcasting

a = a + 1 # add one to every element

- When operating on multiple arrays, broadcasting rules are used.
  - Each dimension must match, from right-to-left
  - Dimensions of size 1 will broadcast (as if the value was repeated).
  - Otherwise, the dimension must have the same shape.
  - Extra dimensions of size 1 are added to the left as needed.

Broadcasting

- Suppose we want to add a color value to an image
  - a.shape is 100, 200, 3
  - b.shape is 3
- a + b will pad b with two extra dimensions so it has an effective shape of 1 x 1 x 3.
- So, the addition will broadcast over the first and second dimensions.

Broadcasting

If a.shape is 100, 200, 3 but b.shape is 4 then a + b will fail. The trailing dimensions must have the same shape (or be 1)

SciPy

- SciPy is an Open Source library of scientific tools for Python. It depends on the NumPy library, and it gathers a variety of high level science and engineering modules together as a single package. SciPy provides modules for
  - file input/output
  - statistics
  - optimization
  - numerical integration
  - linear algebra
  - Fourier transforms
  - signal processing
  - image processing
  - ODE solvers
  - special functions
  - and more...
- See http://docs.scipy.org/doc/scipy/reference/tutorial/

SciPy: Lineer Cebir

scipy.linalg

SciPy is built using the optimized ATLAS LAPACK (Linear Algebra PACKage) and BLAS (Basic Linear Algebra Subprograms) libraries, it has very fast linear algebra capabilities. All of these linear algebra routines expect an object that can be converted into a 2-dimensional array. The matrix class is initialized with the SciPy command mat which is just convenient short-hand for matrix.

```
>>> A = matrix('1.0 2.0; 3.0 4.0')
>>> A
[[ 1. 2.]
 [ 3. 4.]]

>>> type(A) # file where class is defined
<class 'numpy.matrixlib.defmatrix.matrix'>

>>> B = mat('[1.0 2.0; 3.0 4.0]')
>>> B
[[ 1. 2.]
 [ 3. 4.]]

>>> type(B) # file where class is defined
<class 'numpy.matrixlib.defmatrix.matrix'>
```

SciPy: Lineer Cebir

Matrs tersi

```
>>> A = mat('[1 3 5; 2 5 1; 2 3 8]')
>>> A
matrix([[1, 3, 5],
        [2, 5, 1],
        [2, 3, 8]])
>>> A.I
matrix([[-1.48,  0.36,  0.88],
        [ 0.56,  0.08, -0.36],
        [ 0.16, -0.12,  0.04]])
>>> from scipy import linalg
>>> linalg.inv(A)
array([[-1.48,  0.36,  0.88],
       [ 0.56,  0.08, -0.36],
       [ 0.16, -0.12,  0.04]])
```

SciPy: Lineer Cebir

Lineer sistem çözümü

$$\begin{aligned} x + 3y + 5z &= 10 \\ 2x + 5y + z &= 8 \\ 2x + 3y + 8z &= 3 \end{aligned}$$

S = A^{-1} B where S=[x y z] and B = [10 8 3]

```
>>> A = mat('[1 3 5; 2 5 1; 2 3 8]')
>>> b = mat('[10;8;3]')
>>> A.I*b
matrix([[-9.28],
        [ 5.16],
        [ 0.76]])
>>> linalg.solve(A,b)
array([[-9.28],
       [ 5.16],
       [ 0.76]])
```

SciPy: Lineer Cebir

Determinant bulma

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix}$$

$$\begin{aligned} |\mathbf{A}| &= 1\begin{vmatrix} 5 & 1 \\ 3 & 8 \end{vmatrix} - 3\begin{vmatrix} 2 & 1 \\ 2 & 8 \end{vmatrix} + 5\begin{vmatrix} 2 & 5 \\ 2 & 3 \end{vmatrix} \\ &= 1\,(5 \cdot 8 - 3 \cdot 1) - 3\,(2 \cdot 8 - 2 \cdot 1) + 5\,(2 \cdot 3 - 2 \cdot 5) = -25. \end{aligned}$$

```
>>> A = mat('[1 3 5; 2 5 1; 2 3 8]')
>>> linalg.det(A)
-25.000000000000004
```

Uygulama

- https://github.com/gertingold/euroscipy-numpy-tutorial
- Piazza resources altında

  lecture_07_numpy.ipynb