Uygulama Notları: 3

FİZ220 - Bilgisayar Programlama II | 20/03/2020

Matrisler (NumPy Matrix (*numpy.matrix*) Nesneleri)

- NumPy Kütüphanesine Giriş
  - NumPy ve SciPy kütüphaneleri
  - Matrix değişken tipi
  - Temel Matris İşlemleri
    - Matris hakkında bilgi almak için kullanılan metotlar
      - ndim
      - shape
      - size
    - Matris elemanlarına erişim ve değiştirme
    - Transpozesini (devriğini) almak
    - Tersini (inverse) almak
    - Matrislerin birbirleri ile toplanması ve çarpılması
      - Toplama işlemi
      - Çarpma işlemi
      - Matrisin üssünü alma
      - Matrisin 'şeklini değiştirme'
  - Yolun sonu
  - Kurtuluş: NumPy dizileri
  - Aklınızdaki "o soru"

Dr. Emre S. Taşcı, emre.tasci@hacettepe.edu.tr (mailto:emre.tasci@hacettepe.edu.tr)
Fizik Mühendisliği Bölümü
Hacettepe Üniversitesi

NumPy Kütüphanesine Giriş

NumPy ve SciPy kütüphaneleri

Python'da matematiksel işlemler için iki adet temel kütüphane vardır: NumPy ve SciPy. NumPy, temel matematik ve asıl olarak sayısal hesaplamaların olmazsa olmazı matris işlemlerini içerirken, SciPy daha ileri özel fonksiyonlarını ve çözüm yöntemlerini bünyesinde barındırır. Dersimizde işlemlerimize NumPy ile başlayıp, daha ileri seviyedeki problemler için ileride SciPy'dan da yararlanacağız.

NumPy nesneleri ve işlemleri, GNU Octave ile büyük parallellik gösterir. Bu benzerliği daha ilk aşamada "matrix" değişkeni ile hemen görmek mümkündür.

Matrix değişken tipi

NumPy'da `matrix` türündeki matrisler tırnak içinde, -tıpkı Octave'da olduğu gibi- sütunlar virgül veya boşlukla; satırlar ise noktalı virgülle ayrılır (Octave'ın aksine, satır ayrımı yapmak için bir alt satıra geçmek hataya sebep olur).

Tırnağın içerisinde köşeli parantez kullanmak opsiyoneldir.

3x3'lük bir matris tanımlayalım:

In [1]:
```
import numpy as np

# Koseli parantez kullanmadan tanimlayalim:
a = np.matrix("1 2, 3;4 5 6 ; 7 8 9")
print(a)

print("---------------")

# Ayni matrisi bu sefer koseli parantez
# kullanarak tanimlayalim:
a_2 = np.matrix("[1 2, 3;4 5 6 ; 7 8 9]")
print(a_2)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
---------------
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Temel Matris İşlemleri

Matris hakkında bilgi almak için kullanılan metotlar(*):

- `ndim` ile matrisimizin kaç boyutlu olduğunu,
- `shape` ile kaça kaçlık bir matris olduğunu,
- `size` ile de matrisimizdeki eleman sayısını

öğreniriz:

In [2]:
```
a = np.matrix("1 2, 3;4 5 6 ; 7 8 9")
print("Matrisimiz",a.ndim,"boyutlu bir matris olup,")
kaca_kac = a.shape
print(kaca_kac[0],"x",kaca_kac[1],"dağılımlı bir matristir.")
print("Matrisimizde toplam ",a.size,"adet eleman vardır.")
```

```
Matrisimiz 2 boyutlu bir matris olup,
3 x 3 dağılımlı bir matristir.
Matrisimizde toplam  9 adet eleman vardır.
```

(*) Her ne kadar "metot" terimini kullanmış olsam da, teknik olarak `matrix` nesnesinin özelliklerinden ("attribute") bahsetmekteyiz -- işleri fazla karıştırmamak için nesne özellikleri ve metotları geçişli olarak kullanılacaktır.

Matris elemanlarına erişim ve değiştirme

Matris elemanlarına köşeli parantez içerisinde indisi belirterek erişebiliriz (indislerin 0'dan başladığına ve bütün indislerin aynı köşeli parantez içinde belirtildiğine dikkat edin!).

Örneğin, matrisimizin 2. satırının 3. elemanına erişmek için:

In [3]:
```
a = np.matrix("1 2, 3;4 5 6 ; 7 8 9")
print(a)
print("Matrisimizin 2. satırının 3. sütundaki elemanı:",a[1,2])
print("----------")
# Bu elemana yeni bir deger atayalim:
a[1,2] = 12
print("Matrisimizin 2. satırının 3. sütundaki elemanı:",a[1,2])
print(a)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Matrisimizin 2. satırının 3. sütundaki elemanı: 6
----------
Matrisimizin 2. satırının 3. sütundaki elemanı: 12
[[ 1  2  3]
 [ 4  5 12]
 [ 7  8  9]]
```

Matrisimizin bir kısmına erişmek için aralıkları kullanabiliriz:

In [4]:
```
a = np.matrix("1 2, 3;4 5 6 ; 7 8 9")
print("Matrisimizin 2. ve 3. satırlarının, 1. ve 2. elemanları:")
print(a[1:3,0:2])
```

```
Matrisimizin 2. ve 3. satırlarının, 1. ve 2. elemanları:
[[4 5]
 [7 8]]
```

Aralıkları belirtirken bitiş elemanının "kadar" anlamına geldiğini, onu **içermediğini** unutmayın!

Hazır aralıklardan söz açmışken, aralıklardaki artış miktarını 3. parametre olarak belirtiriz *(GNU Octave'da bu parametre başlangıç ve bitiş parametrelerinin ortasında yer almaktaydı)*.

In [5]:
```
b = np.matrix("1 2, 3,9;4 5 6,9 ; 7 8 9 9")
print(b)
print("Matrisimizin 1. ve 3. satırlarının, 2. ve 4. sütunları")
print(b[0:4:2,1:4:2])
```

```
[[1 2 3 9]
 [4 5 6 9]
 [7 8 9 9]]
Matrisimizin 1. ve 3. satırlarının, 2. ve 4. sütunları
[[2 9]
 [8 9]]
```

Transpozesini (*devriğini*) almak: 'T' metodu bu iş içindir:

In [6]:
```
a = np.matrix("1 2, 3;4 5 6 ; 7 8 9")
print(a.T)
```

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

Aynı işi uzun uzadıya `transpose()` fonksiyonu ile de yapabiliriz:

In [7]:
```
a = np.matrix("1 2, 3;4 5 6 ; 7 8 9")
print(np.transpose(a))
```

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

Fonksiyon NumPy kütüphanesinde tanımlı olduğundan, çağırırken 'np' isim alanı (*namespace*) ile `np.transpose()` şeklinde çağırdığımıza dikkat edin.

(Matrisin kompleks eşlenik transpozesini almak içinse 'H' metodu kullanılır)

Tersini (*inverse*) almak: 'I' metodu kullanılır:

In [8]:
```
a = np.matrix("0 2 1;1 0 0;2, 0 1")
print(a.I)
```

```
[[ 0.   1.   0. ]
 [ 0.5  1.  -0.5]
 [-0.  -2.   1. ]]
```

Matrislerin birbirleri ile toplanması ve çarpılması
İki matris (boyutları uyumlu olduğu sürece) birbirleri ile toplanabilir, çarpılabilir (bir matrisin üssü de alınabilir).

- **Toplama işlemi:**

In [9]:
```
a = np.matrix("1 2, 3;4 5 6 ; 7 8 9")
b = np.matrix("9 1 2; 3 2 5;1 2 0")
print("a matrisi:\n",a)
print("b matrisi:\n",b)
print("---------")
print("a+b matrisi:\n",a+b)
```

```
a matrisi:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
b matrisi:
 [[9 1 2]
 [3 2 5]
 [1 2 0]]
---------
a+b matrisi:
 [[10  3  5]
 [ 7  7 11]
 [ 8 10  9]]
```

- **Çarpma işlemi:**

In [10]:
```
a = np.matrix("1 2, 3;4 5 6")
b = np.matrix("9 1; 3 2;1 2")
print("a matrisi:\n",a)
print("b matrisi:\n",b)
print("---------")
print("a*b matrisi:\n",a*b)
```

```
a matrisi:
 [[1 2 3]
 [4 5 6]]
b matrisi:
 [[9 1]
 [3 2]
 [1 2]]
---------
a*b matrisi:
 [[18 11]
 [57 26]]
```

- **Matrisin üssünü alma:**

In [11]:
```
a = np.matrix("1 2;3 4")
print("a matrisi:\n",a)
print("a^2 matrisi:\n",a**2)
print("a^3 matrisi:\n",a**3)
```

```
a matrisi:
 [[1 2]
 [3 4]]
a^2 matrisi:
 [[ 7 10]
 [15 22]]
a^3 matrisi:
 [[ 37  54]
 [ 81 118]]
```

(Sadece tam sayı üsler alınabilir -- örneğin matrisin kökünü hesaplamak için 0.5 üssünü kullanamayız -- biraz sabır, bu sorunun da üstesinden geleceğiz! 8)

Matrisin 'şeklini değiştirme':
Elimizde 3x4'lük bir matris olsun. Bu matrisin düzenini değiştirip, onu 6x2'lik bir matrise dönüştürmek istiyoruz. Bu durumda `shape()` metotu yardımımıza koşar:

In [12]:
```
a = np.matrix("1,2,3,4;5,6,7,8;9,10,11,12")
print(a)
kaca_kac = a.shape
print(kaca_kac[0],"x",kaca_kac[1],"dağılımlı bir matristir.")
print("-------------")
b = a.reshape(6,2)
print(b)
kaca_kac = b.shape
print(kaca_kac[0],"x",kaca_kac[1],"dağılımlı bir matristir.")
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
3 x 4 dağılımlı bir matristir.
-------------
[[ 1  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]
 [11 12]]
6 x 2 dağılımlı bir matristir.
```

Bazen de matrisimizi tamamıyla *düzleştirmek* yani onu 1 boyutlu bir matrise indirmek ihtiyacını duyarız. Bu durumda `flat` metotunu kullanırız:

In [13]:
```
a = np.matrix("1,2,3,4;5,6,7,8;9,10,11,12")
print(a)
kaca_kac = a.shape
print(kaca_kac[0],"x",kaca_kac[1],"dağılımlı bir matristir.")
print("-------------")
b = np.matrix(a.flat)
print(b)
kaca_kac = b.shape
print(kaca_kac[0],"x",kaca_kac[1],"dağılımlı bir matristir.")
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
3 x 4 dağılımlı bir matristir.
-------------
[[ 1  2  3  4  5  6  7  8  9 10 11 12]]
1 x 12 dağılımlı bir matristir.
```

`flat` metotu diğer metotların aksine bir `matrix` nesnesi değil, daha özel bir nesne (`flatiter`) döndürür (`matrix` nesnesi olmadığından ötürü de `matrix` nesnesinin sahip olduğu metotlara (örn. `shape`) sahip değildir). Bu nedenle yeni b matrisini tanımlarken `b = np.matrix(a.flat)` şeklinde, `flat` in sonucunu tekrardan `matrix` olarak atamaktayız.

Herhangi bir nesnenin cinsini `type()` fonksiyonu ile öğrenebiliriz:

In [14]:
```
a = np.matrix("1,2,3,4;5,6,7,8;9,10,11,12")
print("a'nın cinsi:",type(a))
print("'a.flat'ın cinsi:",type(a.flat))
b = np.matrix(a.flat)
print("b'nin cinsi:",type(b))
```

```
a'nın cinsi: <class 'numpy.matrix'>
'a.flat'ın cinsi: <class 'numpy.flatiter'>
b'nin cinsi: <class 'numpy.matrix'>
```

Yolun Sonu

NumPy'ın `matrix` nesnesi çok başta, dil ilk kurulurken çalışmalara hız kazandırsın diye oluşturulmuş bir nesne idi: sonrasında yeterli gelmediği için, daha gelişmiş olan *NumPy dizileri* geliştirildi (bir sonraki dersimizin konusu).

Bu yetersizliği anlamak için, 1 boyutlu bir `matrix` nesnesi tanımlayalım:

In [15]:
```
a1 = np.matrix([1,2,3])
print("1 boyutlu matrisimiz:\n",a1)
```

```
1 boyutlu matrisimiz:
 [[1 2 3]]
```

buraya kadar bir sıkıntı yok.

Şimdi de 2 boyutlu bir `matrix` nesnesi tanımlayalım:

In [16]:
```
a2 = np.matrix("[1,2,3;4,5,6]")
print("2 boyutlu matrisimiz:\n",a2)

# veya, alternatif olarak:
a2_2 = np.matrix([[1,2,3],[4,5,6]])
print("2 boyutlu matrisimiz:\n",a2_2)
```

```
2 boyutlu matrisimiz:
 [[1 2 3]
 [4 5 6]]
2 boyutlu matrisimiz:
 [[1 2 3]
 [4 5 6]]
```

Hâlâ iyi gidiyoruz.

Daha da yüksek boyuta çıkalım:

In [17]:
```
a3 = np.matrix([[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call
 last)
<ipython-input-17-9c82791b4b4c> in <module>
----> 1 a3 = np.matrix([[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]])

~/anaconda3/lib/python3.8/site-packages/numpy/matrixlib/defmatrix.py i
n __new__(subtype, data, dtype, copy)
    147         shape = arr.shape
    148         if (ndim > 2):
--> 149             raise ValueError("matrix must be 2-dimensional")
    150         elif ndim == 0:
    151             shape = (1, 1)

ValueError: matrix must be 2-dimensional
```

Hata mesajına ("ValueError: matrix must be 2-dimensional"), hatta daha da açık olarak, orijinal tanımın içine yerleştirilmiş kontrol mekanizmasına ("if (ndim > 2):") bakacak olursanız, `matrix` nesnelerinin en fazla 2 boyutlu olabileceğini, daha yüksek boyutlara izin verilmediğini göreceksiniz.

Daha da dramatik olarak, [bizzat resmi NumPy matrix nesnesinin dökümanında (https://docs.scipy.org/doc/numpy/reference/generated/numpy.matrix.html)](https://docs.scipy.org/doc/numpy/reference/generated/numpy.matrix.html) kullanıcı bu nesneye karşı uyarılmakta:

Note:

It is no longer recommended to use this class, even for linear algebra. Instead use regular arrays. The class may be removed in the future.

(Bu nesnenin lineer cebirde dahi kullanılması artık tavsiye edilmemektedir. Bunun yerine normal dizileri kullanın. Bu nesne gelecekte kaldırılabilir.)

Kurtuluş: NumPy dizileri

Biz ne yapacağız? Tavsiyeyi dinleyip, çok daha esnek ve daha geniş desteklenen NumPy dizilerini kullanacağız (bkz. bir sonraki ders). Elimizdeki bir `matrix` nesnesini doğrudan ve kolaylıkla NumPy dizisine çevirmek için 'A' metotu tanımlıdır:

In [18]:
```
a = np.matrix("1,2,3,4;5,6,7,8;9,10,11,12")
print(a)
print("a nesnesinin cinsi:",type(a))
print("------------")
b = a.A
print(b)
print("b nesnesinin cinsi:",type(b))
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
a nesnesinin cinsi: <class 'numpy.matrix'>
------------
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
b nesnesinin cinsi: <class 'numpy.ndarray'>
```

Paniğe mahal yok: şimdiye kadar gördüğümüz pek çok `matrix` metodu NumPy dizilerinde de geçerlidir (olmayanlar da numpy.linalg kütüphanesi ile gelecek -- bir sonraki derste 8)

In [19]:
```
a = np.matrix("1,2,3,4;5,6,7,8;9,10,11,12")
b = a.A
print("a matrisinden cevirdigimiz b dizisi:\n",b)
print("2. satirin, 3. elemani:",b[1,2])
print("Cinsi:",type(b))
print("Boyut sayisi:",b.ndim)
print("Eleman sayisi:",b.size,"\n------------")
kaca_kac = b.shape
print("Kaca kaclik: (",kaca_kac[0],"x",kaca_kac[1],")")
print("Transpozesi:\n",b.T,"\n------------")
print("Toplama islemi\n",b+b,"\n------------")
print("Carpma islemi\n",b*b,"\n------------")
print("Us alma islemi\n",b**2,"\n------------")
print("Artik kok alma gibi tam sayi olmayan usler de desteklenmekte:")
print("Koku\n",b**(0.5),"\n------------")
```

```
a matrisinden cevirdigimiz b dizisi:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
2. satirin, 3. elemani: 7
Cinsi: <class 'numpy.ndarray'>
Boyut sayisi: 2
Eleman sayisi: 12
------------
Kaca kaclik: ( 3 x 4 )
Transpozesi:
 [[ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]
 [ 4  8 12]]
------------
Toplama islemi
 [[ 2  4  6  8]
 [10 12 14 16]
 [18 20 22 24]]
------------
Carpma islemi
 [[  1   4   9  16]
 [ 25  36  49  64]
 [ 81 100 121 144]]
------------
Us alma islemi
 [[  1   4   9  16]
 [ 25  36  49  64]
 [ 81 100 121 144]]
------------
Artik kok alma gibi tam sayi olmayan usler de desteklenmekte:
Koku
 [[1.         1.41421356 1.73205081 2.        ]
 [2.23606798 2.44948974 2.64575131 2.82842712]
 [3.         3.16227766 3.31662479 3.46410162]]
------------
```

Aklınızdaki "o soru"##

Madem ileride `matrix` nesnesini kullanmayıp dizileri (`ndarray`) kullanacaktık, ne demeye bu kadar şeyi gördük?

Programlama dillerinde dizi kavramı çok geniş bir kavram. Baştan o kapıdan geçseydik, "dizi... dizi..." dedikçe kafanız karışacaktı (çünkü NumPy'da dizi dediğimiz şey pratikte matrisin ta kendisi). Kaldı ki -bir sonraki derste göreceğimiz üzere- dizi tanımında bizim Octave'dan alışageldiğimiz ve matrislerde kullandığımız satır/sütun - noktalı virgül/virgül tanımına izin verilmemekte. Bu nedenle yumuşak bir geçiş olsun istedim. 'A' metotu yardımıyla hiçbir kayıp vermeden elimizdeki matrisleri hop diye diziye çevirebildiğimiz için, bu dersi dizilere bir giriş olarak ele aldık.

-Çok düşük ihtimalle de olsa- aklınızda olabilecek bir başka soru:

Hocam, "nesne... nesne..." (_object... object... ;) deyip duruyorsunuz, ama sonra `type()` fonksiyonu ile cinsini sorduğumuzda bize "bir şey bir şey sınıfı (class)" diye yanıt geliyor. Nesne ile sınıf arasındaki fark nedir?

(Öncelikle bu güzel soruyu sorduğunuz için teşekkür ederim... <kem küm...> 8) Sınıf dediğimiz şey, nesnenin ait olduğu yapıdır: nesne ise, bu yapıdan tanımladığımız bir elemandır, projeksiyon/anlık-gerçeklemedir (*instance*). Örneğin "araba" bir sınıftır: siz programınızda (hayatınızda) kullanmak üzere kendinize bu sınıftan bir eleman çektiğinizde (bir anlık-gerçekleme yaptığınızda) "Volkswagen", "Anadol", "Renault" gibi adı konulmuş bir nesne oluşturmuş olursunuz. Kafanız karıştıysa boşverin, teknik ve semantik bir ayrım var. Özetle: Sınıflar soyut (*abstract*) tanımlardır (üzerine binip bir yere gidemezsiniz), nesneler ise bu tanımlar doğrultusunda oluşturduğunuz, elle tutulur, üzerlerinde işlemler yapıp değerler atayabileceğiniz elemanlardır. Bu nedenle bir nesnenin cinsini sorduğumuzda cevap olarak onun sınıfı verilmektedir.

In [ ]: