Chapter 0.1:
Numpy: Membuat dan Memanipulasi Data Numerik

February 20, 2021

Numpy (*Numerical Python*) menyediakan paket ekstensi untuk Python berhubungan dengan array multi-dimensi, lebih efisien dalam hardware, didesain untuk komputasi saintifik, dan dikenal juga dengan *array-oriented computing*. Apapun datanya, langkah pertama yang harus dilakukan untuk membuat data dapat dianalisa adalah mentransformasikan data tersebut ke dalam sebuah array.

Array dari *Numpy* mirip dengan *built-in* tipe `list`, tetapi Numpy menyediakan kemampuan menyimpan data dan operaisi pada data yang lebih efisien, terutama ketika data membesar. Array dari *Numpy* merupakan inti dari hampir semua ekosistem tool-tool data science di Python.

[3]:
```
import numpy as np
a = np.array([0,1,2,3])
a
```

[3]:
```
array([0, 1, 2, 3])
```

Untuk melihat versi numpy yang sudah diinstall

[4]:
```
np.__version__
```

[4]:
```
'1.18.1'
```

Array biasanya digunakan untuk menyimpan: * sinyal rekaman dari alat ukur * piksel-piksel dari gambar * harga-harga suatu eksperimen/simulasi pada waktu diskrit * 3-D data hasil pengukuran posisi X-Y-Z, mis. *MRI Scan*

Array berguna karena efisien dalam penggunaan memori dan menyediakan komputasi numerik yang cepat. Berbeda dengan Python-List, numpy hanya support tipe data yang sama dalam sebuah array. Array pada numpy bisa didefinisikan sebagai multidimensi, sedangkan pada Python-List tidak.

Eksekusi array pada numpy lebih cepat dibanding pada Python.

[5]:
```
L = range(1000)
```

[6]:
```
%timeit [i**2 for i in L]
```

```
328 µs ± 16.6 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Bandingkan dengan berikut:

[8]:
```
a = np.arange(1000)
```

[9]:
```
%timeit a**2
```

```
950 ns ± 4.77 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

Terdapat dokumentasi untuk mencari sesuatu menyangkut numpy:

```
>>> np.lookfor('create array')
```

perhatikan apa yang terjadi. Atau:

```
>>> np.con*?
```

perhatikan juga apa keluaran perintah berikut:

```
>>> np?
```

[6]:
```
np.con*?
```

```
np.concatenate
np.conj
np.conjugate
np.convolve
```

Untuk numpy biasanya dibuat konvensi dengan simbol `np`

```
>>> import numpy as np
```

1 Membuat Array

- **1-D**:

[12]:
```
a = np.array([0,1,2,4,5])
```

[13]:
```
a
```

[13]:
```
array([0, 1, 2, 4, 5])
```

[16]:
```
a.ndim # dimensi array
```

[16]:
```
1
```

[21]:
```
a.shape # bentuk array
```

[21]:
```
(5,)
```

[22]:
```
len(a) # panjang data
```

[22]:
```
5
```

- **2-D, 3-D,...**:

[72]:
```
b = np.array([[0,1,2,3],[4,5,6,7]]) # Array 2 x 4
```

[30]:
```
b
```

[30]:
```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

[31]:
```
b.ndim
```

[31]:
```
2
```

[32]:
```
b.shape
```

[32]:
```
(2, 4)
```

[33]:
```
len(b) # mengembalikan ukuran dari dimensi pertama
```

[33]:
```
2
```

[34]:
```
c = np.array([[[1],[2]],[[3],[4]]])
```

[35]:
```
c
```

[35]:
```
array([[[1],
        [2]],

       [[3],
        [4]]])
```

[36]:
```
c.ndim
```

[36]:
```
3
```

[37]:
```
c.shape
```

[37]:
```
(2, 2, 1)
```

[38]:
```
len(c)
```

[38]:
```
2
```

1.1 Fungsi-fungsi untuk membuat Array

Dalam praktek, kita jarang menyisipkan elemen-elemen array satu persatu

- Array dengan elemen sekuens, jarak bilangan yang sama *equally/evenly spaced*

[41]:
```
a = np.arange(10) # 0,1,...,n-1
a
```

[41]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[44]:
```
b = np.arange(2,14,2) # start, end(exclusive), step
b
```

[44]:
```
array([ 2,  4,  6,  8, 10, 12])
```

- Array dengan jumlah titik

[46]:
```
c = np.linspace(0,1,6)  # start, end, jumlah titik
c
```

[46]:
```
array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])
```

[48]:
```
d = np.linspace(0,1,5, endpoint = False)
d
```

[48]:
```
array([0. , 0.2, 0.4, 0.6, 0.8])
```

[50]:
```
e = np.linspace(0,1,6, endpoint=False)
e
```

[50]:
```
array([0.        , 0.16666667, 0.33333333, 0.5       , 0.66666667,
       0.83333333])
```

- *Common* Array

[3]:
```
a = np.ones((3,3)) # reminder: (3,3) adalah tuple
```

[6]:
```
a
```

[6]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

[7]:
```
b = np.zeros((2,2))
```

[8]:
```
b
```

[8]:
```
array([[0., 0.],
       [0., 0.]])
```

[9]:
```
c = np.eye(3) # Identity matrix 3 x 3
```

[10]:
```
c
```

[10]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

[7]:
```
d = np.diag(np.array([1,2,3,4])) # Membuat matriks diagonal dengan elemen␣
↪diagonal dari array [1,2,3,4]
```

[8]:
```
d
```

[8]:
```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

Membuat array ukuran 3 x 5, dengan entri 3.14

[9]:
```
np.full((3,5),3.14)
```

[9]:
```
array([[3.14, 3.14, 3.14, 3.14, 3.14],
       [3.14, 3.14, 3.14, 3.14, 3.14],
       [3.14, 3.14, 3.14, 3.14, 3.14]])
```

- Membangkitkan bilangan random

[16]:
```
a = np.random.rand(4)  # Distribusi uniform di [0,1]
a
```

[16]:
```
array([0.27512146, 0.93048162, 0.84654969, 0.95753028])
```

[22]:
```
b = np.random.randn(4,2) # Distribusi Gaussian 4 x 2
b
```

[22]:
```
array([[ 0.53144423, -1.32853226],
       [ 0.69211962,  0.24272433],
       [-1.48183158, -2.24778021],
       [ 0.32328004,  0.15745643]])
```

[31]:
```
np.random.seed(1234)  # Setting random seed
```

[32]:
```
np.random.rand(3)
```

[32]:
```
array([0.19151945, 0.62210877, 0.43772774])
```

[33]:
```
np.empty()
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-33-d18d581e1fd9> in <module>
----> 1 np.empty()


TypeError: empty() missing required argument 'shape' (pos 1)
```

[34]:
```
np.empty?
```

[35]:
```
np.empty([3,3])
```

[35]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

1.2 Tipe data dasar

Bisa digunakan dalam array dengan `dtype`.

[36]:
```
a = np.array([1,2,3,4,5,6])
```

[37]:
```
a.dtype
```

[37]:
```
dtype('int64')
```

[38]:
```
b = np.array([1., 2.4, 5])
```

[39]:
```
b.dtype
```

[39]:
```
dtype('float64')
```

Pada contoh-contoh di atas, Numpy melakukan *auto-detects* tipe-tipe data dari input. Kita juga bisa menspesifikasikan tipe data pada array.

[45]:
```
c =  np.array([1,2,3], dtype = float)
```

[46]:
```
c.dtype
```

[46]:
```
dtype('float64')
```

Defaultnya adalah tipe *float*

[47]:
```
a = np.ones((3,3))
```

[48]:
```
a.dtype
```

[48]:
```
dtype('float64')
```

Tipe-tipe data lain

Complex

[49]:
```
d = np.array([1+2j, 3+4J])
```

[50]:
```
d.dtype
```

[50]:
```
dtype('complex128')
```

Boolean

[56]:
```
e = np.array([True, False, False, True])
```

[57]:
```
e
```

[57]:
```
array([ True, False, False,  True])
```

[59]:
```
e.dtype
```

[59]:
```
dtype('bool')
```

Strings

[68]:
```
f = np.array(['Fiky','Yosef','Suratman'])
```

[73]:
```
f.dtype  # String dengan maksimum 8 Karakter
```

[73]:
```
dtype('<U8')
```

1.3 Dasar Visual

Setelah mendapatkan data array, kemudian adanya kemungkinan diplot.

- **1D-Plotting**

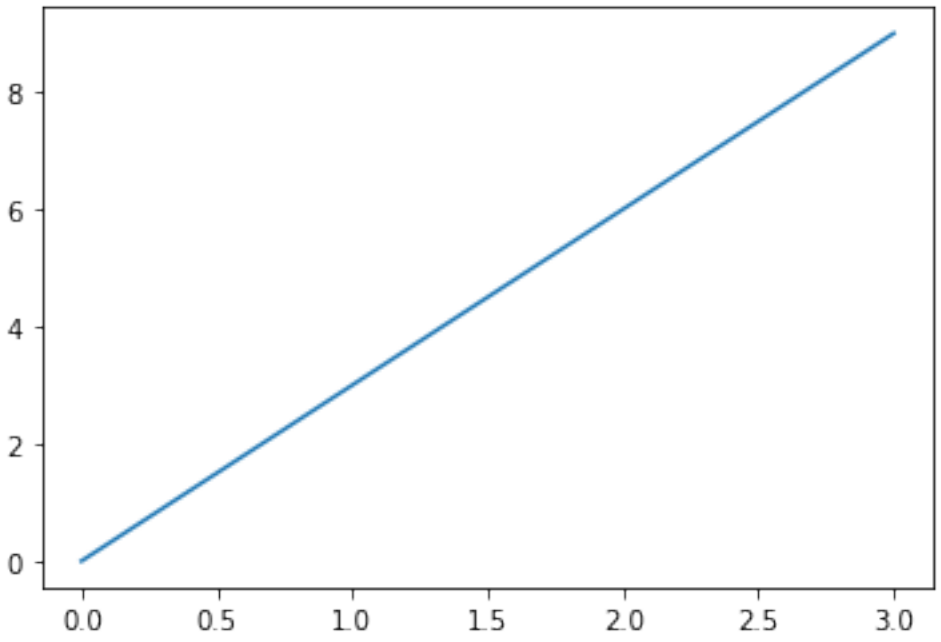
[74]:
```
x = np.linspace(0,3,20)
```

[75]:
```
y = np.linspace(0,9,20)
```

[76]:
```
import matplotlib.pyplot as plt
```

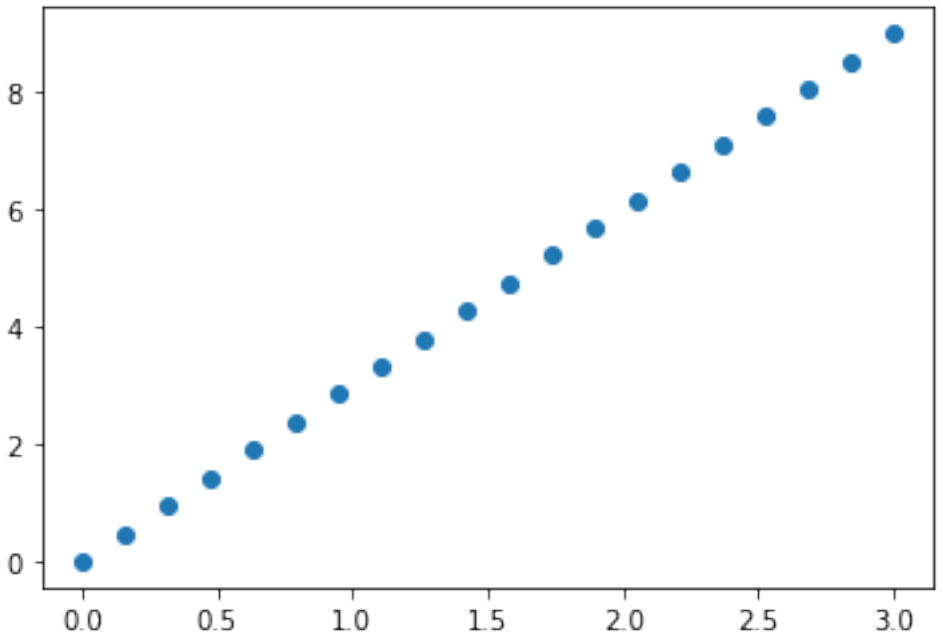
[77]:
```
plt.plot(x,y)
```

[77]:
```
[<matplotlib.lines.Line2D at 0x112872be0>]
```



[79]:
```
plt.plot(x,y,'o') # dot plot
```

[79]:
```
[<matplotlib.lines.Line2D at 0x112ad0160>]
```



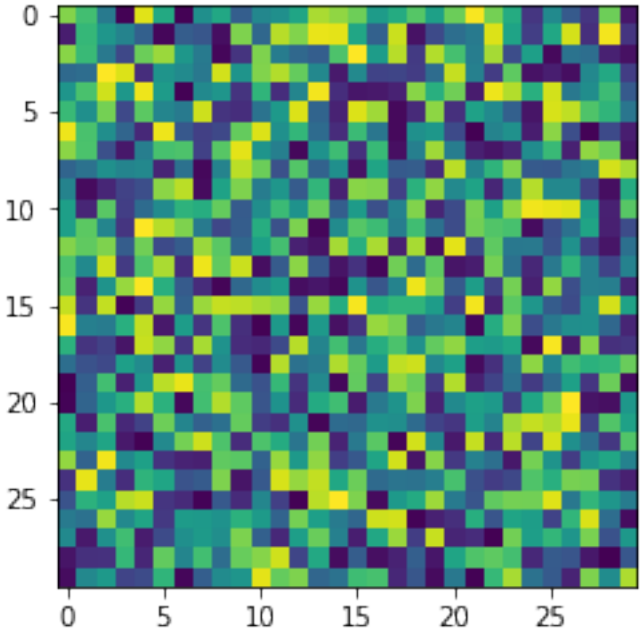
- **2-D Plotting**

[81]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[87]:
```
image = np.random.rand(30,30)
```

[88]:
```
plt.imshow(image)
```

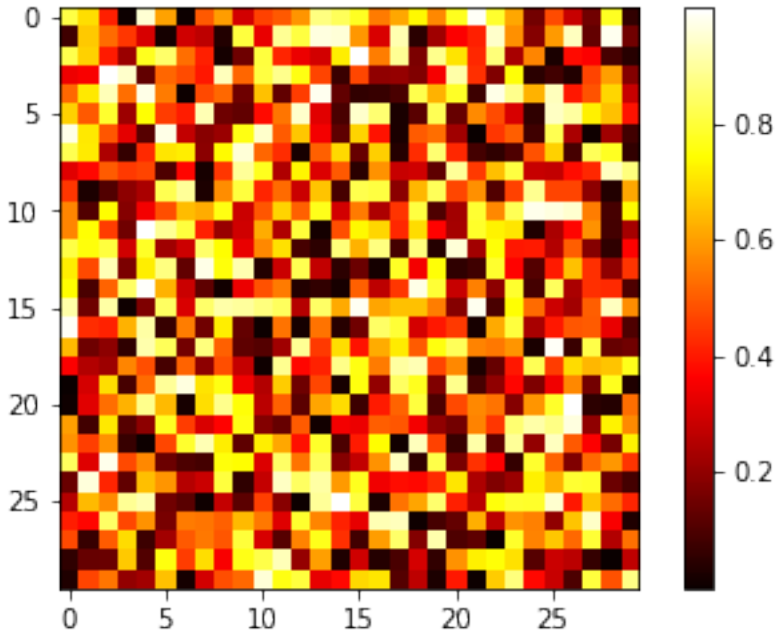
[88]:
```
<matplotlib.image.AxesImage at 0x112b98700>
```



[92]:
```
%matplotlib inline
```

[95]:
```
plt.imshow(image, cmap=plt.cm.hot)
plt.colorbar()
```

[95]:
```
<matplotlib.colorbar.Colorbar at 0x112f71070>
```



1.4 Indexing dan Slicing

Indexing Elemen-elemen dari array dapat diakses dengan menggunakan indeks seperti pada *list*:

[97]:
```
a = np.arange(10)
```

[98]:
```
a[0], a[2], a[-1]
```

[98]:
```
(0, 2, 9)
```

[99]:
```
type(_)
```

[99]:
```
tuple
```

Ingat bahwa indeks dimulai dari `0`.

[100]:
```
a[::-1] # Idiom python untuk membalik sequence
```

[100]:
```
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

Untuk array multidimensi, indeks merupakan *tuples* dari integer:

[101]:
```
a = np.diag(np.arange(3))
```

[102]:
```
a
```

[102]:
```
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 2]])
```

[103]:
```
a[1,1]
```

[103]:
```
1
```

[104]:
```
a[2,1] = 4 # baris ke-3, kolom ke-2
```

[105]:
```
a
```

[105]:
```
array([[0, 0, 0],
       [0, 1, 0],
       [0, 4, 2]])
```

[107]:
```
a[2] # akses baris ke-2
```

[107]:
```
array([0, 4, 2])
```

[108]:
```
a[:,1]
```

[108]:
```
array([0, 1, 4])
```

[109]:
```
_.shape
```

[109]:
```
(3,)
```

[110]:
```
a[2].shape
```

[110]:
```
(3,)
```

Slicing Mengakses elemen-elemen array dapat menggunakan *slicing* seperti pada Python *sequence* umumnya:

[112]:
```
a = np.arange(10)
a
```

[112]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[114]:
```
a[2:9:3]  # start:end:step, indeks terakhir (9) tidak diikutsertakan
```

[114]:
```
array([2, 5, 8])
```

[115]:
```
a[:4]
```

[115]:
```
array([0, 1, 2, 3])
```

Semua indeks *slicing* tidak diperlukan, secara default, *start* = 0, *end* = yang terakhir, *step* = 1.

[116]:
```
a[::2]
```

[116]:
```
array([0, 2, 4, 6, 8])
```

[117]:
```
a[3:]
```

[117]:
```
array([3, 4, 5, 6, 7, 8, 9])
```

[120]:
```
import numpy
numpy.__version__
```

[120]:
```
'1.18.2'
```

Mengkombinasikan pemberian nilai dan *slicing*:

[121]:
```
a = np.arange(10)
```

[122]:
```
a[5:]=10
```

[123]:
```
a
```

[123]:
```
array([ 0,  1,  2,  3,  4, 10, 10, 10, 10, 10])
```

[124]:
```
b = np.arange(5)
```

[125]:
```
a[5:]=b[::-1]
```

[126]:
```
a
```

[126]:
```
array([0, 1, 2, 3, 4, 4, 3, 2, 1, 0])
```

1.5 *Copy* dan *View*

Slicing membuat **view** dari array yang original, yang merupakan cara untuk mengakses data array. Sehingga, array yang original tidak di copy ke memori lain.

[127]:
```
a = np.arange(10)
```

[128]:
```
b = a[::2]
```

[129]:
```
b
```

[129]:
```
array([0, 2, 4, 6, 8])
```

[131]:
```
np.may_share_memory(a,b) # Untuk melihat apakah dua array share memory block␣
↪yang sama.
```

[131]:
```
True
```

[135]:
```
b[0]= 17 # Assignment ini akan merubah a juga, karena b merupakan view dari a
```

[136]:
```
b
```

[136]:
```
array([17,  2,  4,  6,  8])
```

[137]:
```
a
```

```
[137]: array([17,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

Untuk membuat *copy* (bukan *view*) maka bisa digunakan metode `copy()`.

```
[138]: a = np.arange(10)
```

```
[139]: c = a[::2].copy()  # Memaksa untuk terjadi copy
```

```
[144]: c[0]=100 # Assignment ini tidak akan merubah a, karena variabel c merupakan␣
       ↪copy dari a
```

```
[145]: c
```

```
[145]: array([100,   2,   4,   6,   8])
```

```
[146]: a
```

```
[146]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
[147]: np.may_share_memory(a,c)
```

```
[147]: False
```

Sifat *view* ini membingungkan pertama kali, tetapi akan *save* memori dan waktu.

1.6 *Fancy Indexing*

Numpy array bisa diindeks dengan *slicing*, tetapi juga dengan *boolean* dan array integer (**mask**). Metode ini dinamakan ***fancy indexing***.

- Menggunakan *boolean masking*:

```
[49]: np.random.seed(3)
```

```
[50]: a = np.random.randint(0,21,(2,5))
```

Untuk melihat detail penggunaan `randint`, gunakan perintah berikut pada konsol:

```
>>> np.random.randint?
```

```
[51]: a
```

```
[51]: array([[10,  3,  8,  0, 19],
             [10, 11,  9, 10,  6]])
```

```
[54]: mask = (a%3 == 0)
```

```
[55]: mask
```

```
[55]: array([[False,  True, False,  True, False],
             [False, False,  True, False,  True]])
```

```
[56]: extract_a = a[mask]
```

[60]:
```
extract_a
```

[60]:
```
array([3, 0, 9, 6])
```

Indexing menggunakan *mask* sangat berguna untuk memberikan harga baru ke *sub-array*.

[61]:
```
a[a%3 == 0]=-1
```

[62]:
```
a
```

[62]:
```
array([[10, -1,  8, -1, 19],
       [10, 11, -1, 10, -1]])
```

- Menggunakan array integer

[63]:
```
a = np.arange(0,100,10)
```

[64]:
```
a
```

[64]:
```
array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

Indexing dapat dilakukan dengan array integer, dimana indeks yang sama dapat muncul beberapa kali.

[66]:
```
a[[2,4,5,2,4]] # catatan: [2,4,5,2,4] adalah python list.
```

[66]:
```
array([20, 40, 50, 20, 40])
```

Indexing dengan metode ini sangat berguna saat assign harga baru:

[67]:
```
a[[1,7]]=45
```

[68]:
```
a
```

[68]:
```
array([ 0, 45, 20, 30, 40, 50, 60, 45, 80, 90])
```

Ketika sebuah array baru dibuat dengan *indexing* menggunakan array integer, maka array baru yang telah dibuat mempunyai *shape* yang sama dengan array integer. Lihat contoh berikut:

[73]:
```
a = np.random.randn(10)
```

[74]:
```
a
```

[74]:
```
array([-1.62915743, -0.63893553, -0.56080987,  1.81898259,  0.15657257,
       -1.20327619, -0.7040594 ,  1.01922044, -0.53463016, -1.32688896])
```

[75]:
```
idx = np.array([[3,4],[9,7]])
```

[76]:
```
a[idx]
```

[76]:
```
array([[ 1.81898259,  0.15657257],
       [-1.32688896,  1.01922044]])
```

2 Operasi Numerik pada Array

Bagian ini terdiri dari: > * Operasi berdasarkan elemen > * Dasar reduksi > * Broadcasting > * Manipulasi *shape* array > * Sorting data > * Summary

2.1 Operasi Berdasarkan Elemen

Dengan skalar

[17]:
```
import numpy as np
a = np.array([1,2,3,4])
```

[18]:
```
a + 1
```

[18]:
```
array([2, 3, 4, 5])
```

[19]:
```
2**a
```

[19]:
```
array([ 2,  4,  8, 16])
```

[20]:
```
a**2
```

[20]:
```
array([ 1,  4,  9, 16])
```

- Operasi aritmetik per-elemen:

[21]:
```
b = np.ones(4) + 1
```

[22]:
```
a - b
```

[22]:
```
array([-1.,  0.,  1.,  2.])
```

[23]:
```
a * b # perkalian per-elemen, bukan perkalian matrix
```

[23]:
```
array([2., 4., 6., 8.])
```

[24]:
```
j = np.arange(5)
```

[25]:
```
2**(j+1)-j
```

[25]:
```
array([ 2,  3,  6, 13, 28])
```

Operasi-operasi tersebut dilakukan lebih cepat dibandingkan dengan python:

[26]:
```
a = np.arange(10000)
```

[27]:
```
%timeit a + 1
```

```
4.05 µs ± 39.3 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

[28]:
```
l = range(10000)
```

[29]:
```
%timeit [i+1 for i in l]
```

```
538 µs ± 6.39 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Ingat, perkalian array bukan perkalian matrix:

[30]:
```
c = np.ones((3,3))
```

[31]:
```
c * c
```

[31]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

- Perkalian matrix:

[32]:
```
c.dot(c)
```

[32]:
```
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

[34]:
```
b = np.random.randn(3,3)
```

[35]:
```
c.dot(b)
```

[35]:
```
array([[-4.11186577,  0.85036091,  0.49569695],
       [-4.11186577,  0.85036091,  0.49569695],
       [-4.11186577,  0.85036091,  0.49569695]])
```

[36]:
```
b.dot(c)
```

[36]:
```
array([[-1.29577488, -1.29577488, -1.29577488],
       [-1.53422599, -1.53422599, -1.53422599],
       [ 0.06419295,  0.06419295,  0.06419295]])
```

- Perbandingan:

[37]:
```
a = np.array([1,2,3,4])
```

[38]:
```
b = np.array([4,2,2,4])
```

[39]:
```
a == b
```

[39]:
```
array([False,  True, False,  True])
```

[43]:
```
a > b
```

[43]:
```
array([False, False,  True, False])
```

- Perbandingan per-array (bukan perbandingan per-elemen):

[44]:
```
a = np.array([1,2,3,4])
b = np.array([4,2,2,4])
c = np.array([1,2,3,4])
```

[45]:
```
np.array_equal(a,b)
```

[45]:
```
False
```

[48]:
```
np.array_equal(a,c)
```

[48]:
```
True
```

[55]:
```
!export PATH=/Library/TeX/texbin:$PATH
```

- Operasi logika

[58]:
```
a = np.array([1,1,0,0], dtype = bool)
```

[59]:
```
b = np.array([1,0,1,0], dtype = bool)
```

[62]:
```
np.logical_or(a,b)
```

[62]:
```
array([ True,  True,  True, False])
```

[63]:
```
np.logical_and(a,b)
```

[63]:
```
array([ True, False, False, False])
```

- Fungsi-fungsi transcenden:

[64]:
```
a = np.arange(5)
```

[65]:
```
np.sin(a)
```

[65]:
```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ])
```

[66]:
```
np.log(a)
```

```
/Users/fikyy.suratman/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:1: RuntimeWarning: divide by zero encountered in
log
  """Entry point for launching an IPython kernel.
```

[66]:
```
array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436])
```

[67]:
```
np.exp(a)
```

[67]:
```
array([ 1.        ,  2.71828183,  7.3890561 , 20.08553692, 54.59815003])
```

- Transposition:

```
[75]: a = np.triu(np.ones((3,3)),0)

[76]: a

[76]: array([[1., 1., 1.],
             [0., 1., 1.],
             [0., 0., 1.]])

[77]: a = np.triu(np.ones((3,3)),1)

[78]: a

[78]: array([[0., 1., 1.],
             [0., 0., 1.],
             [0., 0., 0.]])
```

Untuk melihat pengertian `np.triu()` gunakan:

```
>>> help(np.triu)
```

Selain itu, gunakan juga `help()` untuk melihat `np.allclose`, `np.tril`

```
[84]: b = a.T # b adalah transpose dari matriks a

[85]: b

[85]: array([[0., 0., 0.],
             [1., 0., 0.],
             [1., 1., 0.]])
```

Transpose adalah sebuah **view**, menggunakan alamat yang sama:

```
[87]: np.may_share_memory(a,b)

[87]: True

[88]: c = a.T.copy()

[89]: c

[89]: array([[0., 0., 0.],
             [1., 0., 0.],
             [1., 1., 0.]])

[90]: np.may_share_memory(a,c)

[90]: False
```

CATATAN: Untuk operasi aljabar dasar seperti pemecahan persamaan linear, SVD dll, terdapat sub-module `numpy.linalg` tetapi direkomendasikan menggunakan `scipy.linalg`.

2.2 Dasar Agregasi Data

- Menghitung penjumlahan

```
[92]: x = np.array([1,2,3,4])
```

```
[93]: np.sum(x)
```

```
[93]: 10
```

atau

```
[94]: x.sum()
```

```
[94]: 10
```

Menjumlahkan *rowwise* atau *columnwise*:

```
[116]: x = np.array([[1,1],[6,7]])
```

```
[117]: x
```

```
[117]: array([[1, 1],
              [6, 7]])
```

```
[118]: x.sum(axis=0)  # rowwise, dimensi 1
```

```
[118]: array([7, 8])
```

```
[119]: x.sum(axis=1) # columnwise, dimensi 2
```

```
[119]: array([ 2, 13])
```

```
[120]: x[0,:].sum() # jumlahkan row ke-1
```

```
[120]: 2
```

```
[121]: x[1,:].sum() # jumlahkan row ke-2
```

```
[121]: 13
```

```
[122]: x[:,0].sum() # jumlahkan k
```

```
[122]: 7
```

Untuk dimensi yang lebih besar

```
[126]: x = np.random.rand(2,2,2)
```

```
[127]: x
```

```
[127]: array([[[0.02450443, 0.50294089],
               [0.14392422, 0.05893191]],
```

```
       [[0.13293748, 0.21327736],
        [0.38211652, 0.06629647]]])
```

[128]:
```
x.sum(axis=2)
```

[128]:
```
array([[0.52744531, 0.20285613],
       [0.34621484, 0.44841299]])
```

Untuk 3-D agak membingungkan, berikut contoh yang memudahkan:

[147]:
```
td = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
```

[148]:
```
td
```

[148]:
```
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
```

[149]:
```
td.sum(axis=0) # dimensi pertama
```

[149]:
```
array([[ 6,  8],
       [10, 12]])
```

[150]:
```
td.sum(axis=1) # dimensi kedua
```

[150]:
```
array([[ 4,  6],
       [12, 14]])
```

[151]:
```
td.sum(axis=2) # dimensi ketiga
```

[151]:
```
array([[ 3,  7],
       [11, 15]])
```

[199]:
```
td.cumsum(axis=0) # dimensi pertama
```

[199]:
```
array([[[ 1,  2],
        [ 3,  0]],

       [[ 6,  8],
        [10,  8]]])
```

[200]:
```
td.cumsum(axis=1) # dimensi kedua
```

[200]:
```
array([[[ 1,  2],
        [ 4,  2]],

       [[ 5,  6],
        [12, 14]]])
```

[201]:
```
td.cumsum(axis=2) # dimensi ketiga
```

[201]:
```
array([[[ 1,  3],
        [ 3,  3]],

       [[ 5, 11],
        [ 7, 15]]])
```

- Menghitung ekstrim

[152]:
```
x = np.array([1,5,3])
```

[155]:
```
x.min() # minimum
```

[155]:
```
1
```

atau bisa juga `np.min(x)`

[156]:
```
x.max() # maksimum
```

[156]:
```
5
```

atau bisa juga `np.max(x)`.

[159]:
```
x.argmin()  # indeks yang bernilai minimum
```

[159]:
```
0
```

atau bisa juga `np.argmin(x)`.

[162]:
```
x.argmax() # indeks yang bernilai maksimum
```

[162]:
```
1
```

atau bisa juga `np.argmax(x)`.

- Operasi Logika

[179]:
```
np.all([True,True,False])
```

[179]:
```
False
```

[180]:
```
np.any([True,True,False])
```

[180]:
```
True
```

[181]:
```
td
```

[181]:
```
array([[[1, 2],
        [3, 0]],

       [[5, 6],
        [7, 8]]])
```

[183]:
```
td.all(axis=0)
```

[183]:
```
array([[ True,  True],
       [ True, False]])
```

[184]:
```
np.all(td,axis=0)
```

[184]:
```
array([[ True,  True],
       [ True, False]])
```

[185]:
```
td.all(axis=1)
```

[185]:
```
array([[ True, False],
       [ True,  True]])
```

Fungsi-fungsi tersebut diatas `any()` dan `all`, bisa digunakan untuk perbandingan array

[186]:
```
a = np.zeros((100,100))
```

[187]:
```
np.any(a!=0)
```

[187]:
```
False
```

[188]:
```
a[0,0]=1
```

[190]:
```
np.any(a!=0)
```

[190]:
```
True
```

[191]:
```
a = np.array([1,2,3,2])
```

[192]:
```
b = np.array([2,2,3,2])
```

[193]:
```
c = np.array([6,4,4,5])
```

[197]:
```
a.all()
```

[197]:
```
True
```

[202]:
```
((a <=b) & (b <=c)).all()
```

[202]:
```
True
```

- Dasar statistik

[203]:
```
x = np.array([1,2,3,1])
```

[204]:
```
y = np.array([[1,2,3],[5,6,1]])
```

[205]:
```
y
```

```
[205]: array([[1, 2, 3],
              [5, 6, 1]])
```

```
[206]: x.mean()
```

```
[206]: 1.75
```

```
[207]: np.median(x)
```

```
[207]: 1.5
```

```
[210]: np.median(y,axis=-1) # axis terakhir, sama dengan np.median(y,axis=1)
```

```
[210]: array([2., 5.])
```

```
[214]: x.std()
```

```
[214]: 0.82915619758885
```

```
[215]: x.var()
```

```
[215]: 0.6875
```

- Contoh: data statistics

```
[1]: !cat data/Populations.txt
```

```
# year  hare    lynx    carrot
1900    30e3    4e3     48300
1901    47.2e3  6.1e3   48200
1902    70.2e3  9.8e3   41500
1903    77.4e3  35.2e3  38200
1904    36.3e3  59.4e3  40600
1905    20.6e3  41.7e3  39800
1906    18.1e3  19e3    38600
1907    21.4e3  13e3    42300
1908    22e3    8.3e3   44500
1909    25.4e3  9.1e3   42100
1910    27.1e3  7.4e3   46000
1911    40.3e3  8e3     46800
1912    57e3    12.3e3  43800
1913    76.6e3  19.5e3  40900
1914    52.3e3  45.7e3  39400
1915    19.5e3  51.1e3  39000
1916    11.2e3  29.7e3  36700
1917    7.6e3   15.8e3  41800
1918    14.6e3  9.7e3   43300
1919    16.2e3  10.1e3  41300
1920    24.7e3  8.6e3   47300
```

```
[3]: import numpy as np
     data = np.loadtxt('data/Populations.txt')
```

```
[4]: data
```

```
[4]: array([[ 1900., 30000.,  4000., 48300.],
            [ 1901., 47200.,  6100., 48200.],
            [ 1902., 70200.,  9800., 41500.],
            [ 1903., 77400., 35200., 38200.],
            [ 1904., 36300., 59400., 40600.],
            [ 1905., 20600., 41700., 39800.],
            [ 1906., 18100., 19000., 38600.],
            [ 1907., 21400., 13000., 42300.],
            [ 1908., 22000.,  8300., 44500.],
            [ 1909., 25400.,  9100., 42100.],
            [ 1910., 27100.,  7400., 46000.],
            [ 1911., 40300.,  8000., 46800.],
            [ 1912., 57000., 12300., 43800.],
            [ 1913., 76600., 19500., 40900.],
            [ 1914., 52300., 45700., 39400.],
            [ 1915., 19500., 51100., 39000.],
            [ 1916., 11200., 29700., 36700.],
            [ 1917.,  7600., 15800., 41800.],
            [ 1918., 14600.,  9700., 43300.],
            [ 1919., 16200., 10100., 41300.],
            [ 1920., 24700.,  8600., 47300.]])
```

```
[5]: data.shape
```

```
[5]: (21, 4)
```

```
[8]: year = data[:,0]
```

```
[9]: year
```

```
[9]: array([1900., 1901., 1902., 1903., 1904., 1905., 1906., 1907., 1908.,
            1909., 1910., 1911., 1912., 1913., 1914., 1915., 1916., 1917.,
            1918., 1919., 1920.])
```

Tidak perlu satu persatu dengan indexing, bisa dilakukan dengan *trick* sebagai berikut:

```
[10]: year, hares, lynxes, carrots = data.T # assign column ke variable
```

```
[11]: year
```

```
[11]: array([1900., 1901., 1902., 1903., 1904., 1905., 1906., 1907., 1908.,
             1909., 1910., 1911., 1912., 1913., 1914., 1915., 1916., 1917.,
             1918., 1919., 1920.])
```

```
[12]: hares
```

```
[12]: array([30000., 47200., 70200., 77400., 36300., 20600., 18100., 21400.,
             22000., 25400., 27100., 40300., 57000., 76600., 52300., 19500.,
             11200.,  7600., 14600., 16200., 24700.])
```

Kemudian akan di-plot:

[13]:
```
import matplotlib.pyplot as plt
```

[39]:
```
plt.axes([0.2, 0.1, 0.5,0.8])
plt.plot(year,hares,year,lynxes,year,carrots)
plt.legend(('Hare','Lynx','Carrot'), loc=(1.05,0.5))
```

[39]:
```
<matplotlib.legend.Legend at 0x11a273e10>
```

[42]:
```
populations = data[:,1:] # menghilangkan kolom tahun
```

[43]:
```
populations.mean(axis=0) # mencari mean tiap kolom
```

[43]:
```
array([34080.95238095, 20166.66666667, 42400.        ])
```

[44]:
```
populations.std(axis=0) # mencari standar deviasi tiap kolom
```

[44]:
```
array([20897.90645809, 16254.59153691,  3322.50622558])
```

Apakah populasi terbesar tiap tahun?

[46]:
```
np.argmax(populations,axis=1)
```

[46]:
```
array([2, 2, 0, 0, 1, 1, 2, 2, 2, 2, 2, 2, 0, 0, 0, 1, 2, 2, 2, 2, 2])
```

Atau bisa juga dengan:

[47]:
```
populations.argmax(axis=1)
```

[47]:
```
array([2, 2, 0, 0, 1, 1, 2, 2, 2, 2, 2, 2, 0, 0, 0, 1, 2, 2, 2, 2, 2])
```

Latihan: Random Walk n_stories = 1000 # Jumlah walkers

[2]:
```
t_max = 200 # durasi waktu kita mengikuti walkers
```

[3]:
```
import numpy as np
```

[5]:
```
t = np.arange(t_max)
```

Generate random integer {0,1}, dikalikan dengan 2, dan dikurangi 1. Sehingga akan digenerate random {-1,+1}.

[8]:
```
steps = 2*np.random.randint(0,1+1,(n_stories,t_max))-1 # ukuran n_stories x␣
↪t_max
```

Untuk meyakinkan bilangan hanya -1 dan +1:

[10]:
```
np.unique(steps)
```

[10]:
```
array([-1,  1])
```

[11]:
```
positions = np.cumsum(steps,axis = 1) # sum column menyatakan posisi
```

[12]:
```
sq_distance = positions**2
```

[16]:
```
mean_sq_distance = np.mean(sq_distance, axis = 0)
```

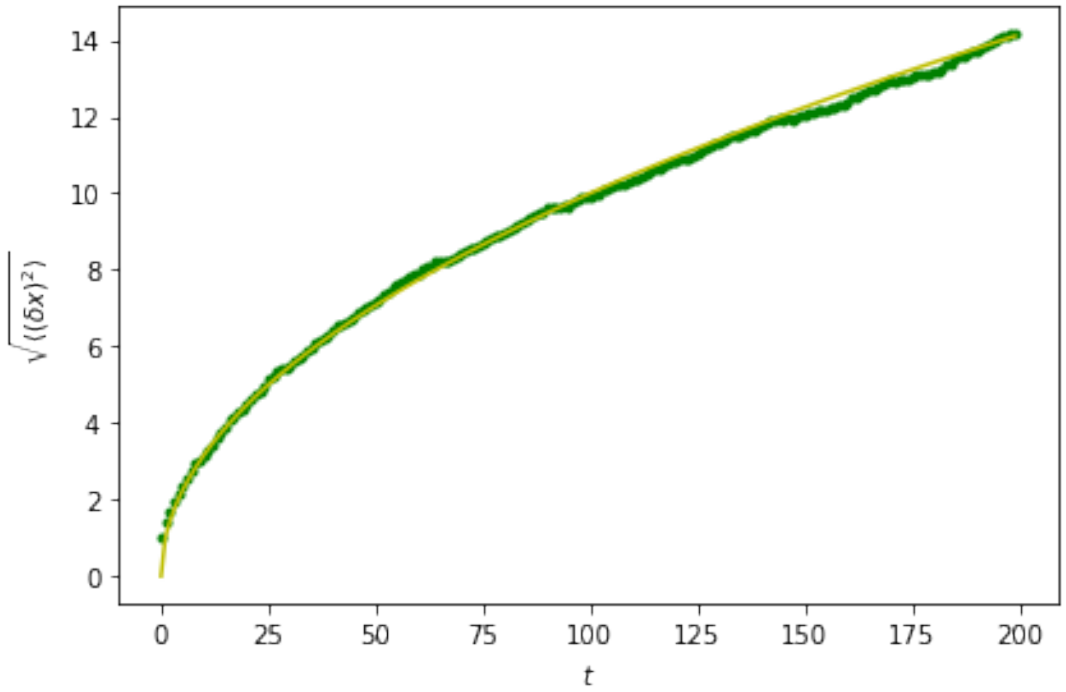
[17]:
```
import matplotlib.pyplot as plt
```

[18]:
```
plt.figure(figsize = (4,3))
```

[18]:
```
<Figure size 288x216 with 0 Axes>
```

<Figure size 288x216 with 0 Axes>

[23]:
```
plt.plot(t,np.sqrt(mean_sq_distance), 'g.', t, np.sqrt(t),'y-')
plt.xlabel(r"$t$")
plt.ylabel(r"$\sqrt{\langle(\delta x)^2\rangle}$")
plt.tight_layout() # to provide sufficient space for labels
```



Terlihat bahwa *known result in physics*: **the RMS distance grows as the square root of the time!**

2.3 Broadcasting

Operasi dasar dari `numpy` adalah *elementwise*, yang bekerja untuk array dengan ukuran yang sama. Kendatipun demikian, di `numpy` memungkinkan untuk melakukan operasi antara array dengan ukuran yang berbeda, jika `numpy` mampu mengkonversi menjadi ukuran yang sama. Mekanisme ini disebut dengan ***Broadcasting***.

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

[5]:
```
x = np.arange(0,40,10)
```

[8]:
```
x
```

[8]:
```
array([ 0, 10, 20, 30])
```

[11]:
```
a = np.tile(x,(3,1)).T
```

[12]:
```
a
```

[12]:
```
array([[ 0,  0,  0],
       [10, 10, 10],
       [20, 20, 20],
       [30, 30, 30]])
```

[14]:
```
b = np.array([0,1,2])
```

[16]:
```
a + b # Ukuran b di-broadcasting jadi seukuran dengan a
```

[16]:
```
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])
```

Kita sudah mempergunakan *broadcasting* di atas.

[17]:
```
a = np.ones((4,5))
```

[25]:
```
a[0] = 2  # Assign nilai 2 ke baris pertama dari matriks a, dengan␣
↪broadcasting
```

[26]:
```
a
```

[26]:
```
array([[2., 2., 2., 2., 2.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

Trik yang berguna:

[27]:
```
a = np.arange(0,40,10)
```

[28]:
```
a.shape
```

[28]:
```
(4,)
```

[29]:
```
a = a[:,np.newaxis] # Menambahkan axis baru 2-D array
```

[31]:
```
a.shape
```

[31]:
```
(4, 1)
```

[34]:
```
a + b # array a mengulang kolom 2 kali sehingga 4 x 3, dan array b mengulang␣
↪baris 3 kali jadi 4 x 3
```

[34]:
```
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])
```

Catatan: *Broadcasting* sangat berguna untuk mengatasi permasalahan dimana data output merupakan array dengan dimensi yang lebih besar dibandingkan dengan data input.

Latihan: ***Broadcasting*** Dimisalkan kita akan mengkonstruksi array dari jarak (dalam miles) antara kota-kota di Amerika pada *Route 66* yaitu: Chicago, Springfield, Saint-Louis, Tulsa, Oklahoma City, Amarillo, Santa Fe, Albuquerque, Flagstaff and Los Angeles.

[35]:
```
mileposts = np.array([0,198,303,736,871,1175,1475,1544,1913,2448])
```

[40]:
```
distance_array = np.abs(mileposts-mileposts[:,np.newaxis])
```

[41]:
```
distance_array
```

[41]:
```
array([[   0,  198,  303,  736,  871, 1175, 1475, 1544, 1913, 2448],
       [ 198,    0,  105,  538,  673,  977, 1277, 1346, 1715, 2250],
       [ 303,  105,    0,  433,  568,  872, 1172, 1241, 1610, 2145],
       [ 736,  538,  433,    0,  135,  439,  739,  808, 1177, 1712],
       [ 871,  673,  568,  135,    0,  304,  604,  673, 1042, 1577],
       [1175,  977,  872,  439,  304,    0,  300,  369,  738, 1273],
       [1475, 1277, 1172,  739,  604,  300,    0,   69,  438,  973],
       [1544, 1346, 1241,  808,  673,  369,   69,    0,  369,  904],
       [1913, 1715, 1610, 1177, 1042,  738,  438,  369,    0,  535],
       [2448, 2250, 2145, 1712, 1577, 1273,  973,  904,  535,    0]])
```

Permasalahan lain adalah *grid-based or network-based* bisa juga menggunakan *broadcasting*. Misalkan kita ingin menghitung jarak dari titik referensi `(0,0)` ke grid `10x10`.

[42]:
```
x, y = np.arange(10), np.arange(10)[:,np.newaxis]
```

[43]:
```
distance = np.sqrt(x ** 2 + y ** 2)
distance
```

[43]:
```
array([[ 0.        ,  1.        ,  2.        ,  3.        ,  4.        ,
         5.        ,  6.        ,  7.        ,  8.        ,  9.        ],
       [ 1.        ,  1.41421356,  2.23606798,  3.16227766,  4.12310563,
         5.09901951,  6.08276253,  7.07106781,  8.06225775,  9.05538514],
       [ 2.        ,  2.23606798,  2.82842712,  3.60555128,  4.47213595,
         5.38516481,  6.32455532,  7.28010989,  8.24621125,  9.21954446],
       [ 3.        ,  3.16227766,  3.60555128,  4.24264069,  5.        ,
         5.83095189,  6.70820393,  7.61577311,  8.54400375,  9.48683298],
       [ 4.        ,  4.12310563,  4.47213595,  5.        ,  5.65685425,
         6.40312424,  7.21110255,  8.06225775,  8.94427191,  9.8488578 ],
       [ 5.        ,  5.09901951,  5.38516481,  5.83095189,  6.40312424,
         7.07106781,  7.81024968,  8.60232527,  9.43398113, 10.29563014],
       [ 6.        ,  6.08276253,  6.32455532,  6.70820393,  7.21110255,
         7.81024968,  8.48528137,  9.21954446, 10.        , 10.81665383],
       [ 7.        ,  7.07106781,  7.28010989,  7.61577311,  8.06225775,
         8.60232527,  9.21954446,  9.89949494, 10.63014581, 11.40175425],
       [ 8.        ,  8.06225775,  8.24621125,  8.54400375,  8.94427191,
         9.43398113, 10.        , 10.63014581, 11.3137085 , 12.04159458],
       [ 9.        ,  9.05538514,  9.21954446,  9.48683298,  9.8488578 ,
        10.29563014, 10.81665383, 11.40175425, 12.04159458, 12.72792206]])
```

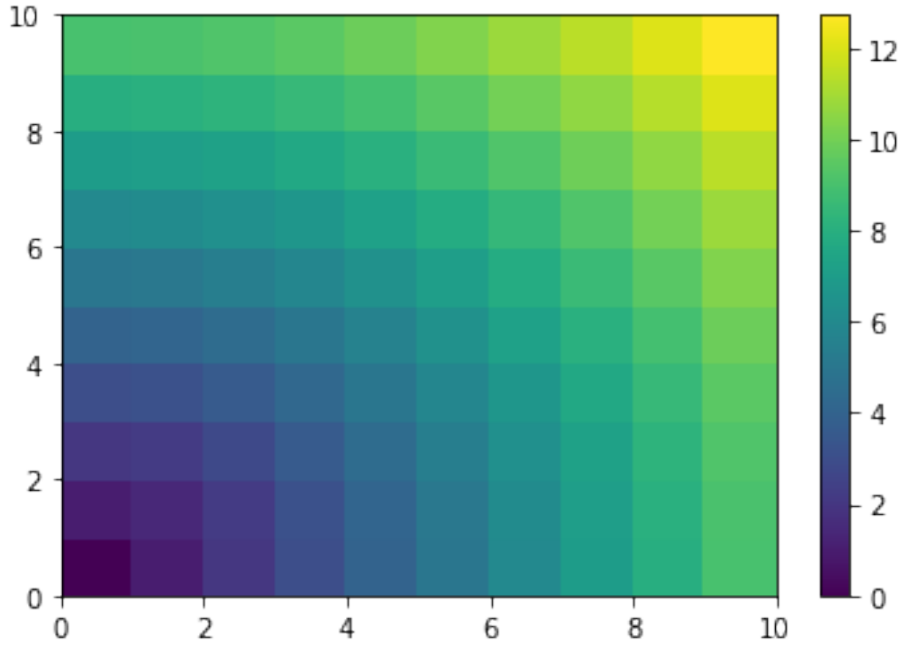
[44]:
```
distance.shape
```

[44]:
```
(10, 10)
```

[46]:
```
import matplotlib.pyplot as plt
```

[47]:
```
plt.pcolor(distance)
plt.colorbar()
```

[47]:
```
<matplotlib.colorbar.Colorbar at 0x120e17610>
```



Assignment terhadap `x,y` di atas dapat dibuat juga secara langsung dengan fungsi `numpy.ogrid()`.

[48]:
```
x, y = np.ogrid[0:10,0:10]
```

[54]:
```
x,y
```

[54]:
```
(array([[0],
        [1],
        [2],
        [3],
        [4],
        [5],
        [6],
        [7],
        [8],
        [9]]),
 array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]]))
```

[53]:
```
x.shape, y.shape
```

[53]:
```
((10, 1), (1, 10))
```

[55]:
```
distance = np.sqrt(x ** 2 + y ** 2)
distance
```

[55]:
```
array([[ 0.        ,  1.        ,  2.        ,  3.        ,  4.        ,
         5.        ,  6.        ,  7.        ,  8.        ,  9.        ],
       [ 1.        ,  1.41421356,  2.23606798,  3.16227766,  4.12310563,
         5.09901951,  6.08276253,  7.07106781,  8.06225775,  9.05538514],
       [ 2.        ,  2.23606798,  2.82842712,  3.60555128,  4.47213595,
         5.38516481,  6.32455532,  7.28010989,  8.24621125,  9.21954446],
       [ 3.        ,  3.16227766,  3.60555128,  4.24264069,  5.        ,
         5.83095189,  6.70820393,  7.61577311,  8.54400375,  9.48683298],
       [ 4.        ,  4.12310563,  4.47213595,  5.        ,  5.65685425,
         6.40312424,  7.21110255,  8.06225775,  8.94427191,  9.8488578 ],
       [ 5.        ,  5.09901951,  5.38516481,  5.83095189,  6.40312424,
         7.07106781,  7.81024968,  8.60232527,  9.43398113, 10.29563014],
       [ 6.        ,  6.08276253,  6.32455532,  6.70820393,  7.21110255,
         7.81024968,  8.48528137,  9.21954446, 10.        , 10.81665383],
       [ 7.        ,  7.07106781,  7.28010989,  7.61577311,  8.06225775,
         8.60232527,  9.21954446,  9.89949494, 10.63014581, 11.40175425],
       [ 8.        ,  8.06225775,  8.24621125,  8.54400375,  8.94427191,
         9.43398113, 10.        , 10.63014581, 11.3137085 , 12.04159458],
       [ 9.        ,  9.05538514,  9.21954446,  9.48683298,  9.8488578 ,
        10.29563014, 10.81665383, 11.40175425, 12.04159458, 12.72792206]])
```

Sehingga `np.ogrid` sangat berguna untuk mengatasi komputasi berdasarkan *grid*. Tapi, jika tidak ingin memanfaatkan broadcasting, `np.mgrid` bisa digunakan langsung untuk menghasilkan matriks penuh dengan indeks.

[56]:
```
x, y = np.mgrid[0:5,0:5]
```

[58]:
```
x
```

[58]:
```
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]])
```

[59]:
```
y
```

[59]:
```
array([[0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4]])
```

2.4 Manipulasi Bentuk Array

Flattening:

[12]:
```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
```

[13]:
```
a.ravel()
```

[13]:
```
array([1, 2, 3, 4, 5, 6])
```

[14]:
```
a.T.ravel()
```

[14]:
```
array([1, 4, 2, 5, 3, 6])
```

Untuk dimensi yang lebih besar, maka dimensi terahir akan *ravel out* pertamakali

[15]:
```
a = np.random.randn(2,3,4)
```

[16]:
```
a.shape
```

[16]:
```
(2, 3, 4)
```

[17]:
```
a
```

[17]:
```
array([[[-0.2359398 , -1.70950758, -0.75672927,  0.65824228],
        [ 2.12637777, -0.15425137,  0.11638403,  0.84068461],
        [-0.46476587, -0.15470568, -0.80960971,  0.73302427]],

       [[ 0.51352798, -0.07209589,  0.09379882, -0.09254156],
        [ 0.47479125, -1.50586348,  1.44728525,  0.89117417],
        [-0.60521243,  0.05395525,  0.02113071,  0.13477008]]])
```

[18]:
```
a.ravel()
```

[18]:
```
array([-0.2359398 , -1.70950758, -0.75672927,  0.65824228,  2.12637777,
       -0.15425137,  0.11638403,  0.84068461, -0.46476587, -0.15470568,
       -0.80960971,  0.73302427,  0.51352798, -0.07209589,  0.09379882,
       -0.09254156,  0.47479125, -1.50586348,  1.44728525,  0.89117417,
       -0.60521243,  0.05395525,  0.02113071,  0.13477008])
```

- Reshaping

Kebalikan dari *flattening*

[19]:
```
a = np.array([[1,2,3],[4,5,6]])
```

[20]:
```
a.shape
```

[20]:
```
(2, 3)
```

[23]:
```
b = a.ravel()
```

[24]:
```
b = b.reshape((2,3))
```

[25]:
```
b
```

```
[25]: array([[1, 2, 3],
             [4, 5, 6]])
```

```
[26]: b[0,0]=99
```

```
[27]: a
```

```
[27]: array([[99,  2,  3],
             [ 4,  5,  6]])
```

Perhatikan bahwa `ndarray.reshape` bisa jadi mengembalikan *view* seperti contoh di atas, atau bisa juga *copy* seperti contoh di bawah ini. Untuk mengerti perbedaan, kita harus mengerti lebih jauh tentang *layout* memory untuk numpy array.

```
[32]: a = np.zeros((3,2))
```

```
[33]: b = a.T.reshape(3*2)
```

```
[34]: b[0]=9
```

```
[35]: a
```

```
[35]: array([[0., 0.],
             [0., 0.],
             [0., 0.]])
```

- Menambahkan Dimensi

Pengindeksan dengan `np.newaxis` memperbolehkan kita untuk menambahkan *axis* ke array, seperti pada contoh-contoh *Broadcasting*.

```
[36]: z = np.array([1,2,3])
```

```
[37]: z
```

```
[37]: array([1, 2, 3])
```

```
[38]: z.shape
```

```
[38]: (3,)
```

```
[42]: z[:,np.newaxis]
```

```
[42]: array([[1],
             [2],
             [3]])
```

```
[43]: z[:,np.newaxis].shape
```

```
[43]: (3, 1)
```

```
[44]: z[np.newaxis,:]
```

[44]:
```
array([[1, 2, 3]])
```

[45]:
```
z[np.newaxis,:].shape
```

[45]:
```
(1, 3)
```

- *Shuffling* dimensi

[46]:
```
a = np.arange(4*3*2).reshape(4,3,2)
```

[47]:
```
a.shape
```

[47]:
```
(4, 3, 2)
```

[48]:
```
a
```

[48]:
```
array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5]],

       [[ 6,  7],
        [ 8,  9],
        [10, 11]],

       [[12, 13],
        [14, 15],
        [16, 17]],

       [[18, 19],
        [20, 21],
        [22, 23]]])
```

[49]:
```
a[0,2,1]
```

[49]:
```
5
```

[50]:
```
b = a.transpose(1,2,0)
```

[51]:
```
b
```

[51]:
```
array([[[ 0,  6, 12, 18],
        [ 1,  7, 13, 19]],

       [[ 2,  8, 14, 20],
        [ 3,  9, 15, 21]],

       [[ 4, 10, 16, 22],
        [ 5, 11, 17, 23]]])
```

[52]:
```
b.shape
```

[52]:
```
(3, 2, 4)
```

[53]:
```
b[2,1,0]
```

[53]:
```
5
```

Proses ini juga akan menghasilkan *view*

[54]:
```
b[2,1,0]=-1
```

[55]:
```
a
```

[55]:
```
array([[[ 0,  1],
        [ 2,  3],
        [ 4, -1]],

       [[ 6,  7],
        [ 8,  9],
        [10, 11]],

       [[12, 13],
        [14, 15],
        [16, 17]],

       [[18, 19],
        [20, 21],
        [22, 23]]])
```

[56]:
```
a[0,2,1]
```

[56]:
```
-1
```

- Resize

Ukuran array bisa dirubah dengan `ndarray.resize`

[60]:
```
a = np.arange(4)
```

[61]:
```
a.resize((8,))
```

[62]:
```
a
```

[62]:
```
array([0, 1, 2, 3, 0, 0, 0, 0])
```

Tetapi tidak bisa di-*resize* kalau sudah di-*assign* ke yang lain:

[63]:
```
b = a
```

[64]:
```
a.resize((4,))
```

```
␣
↪---------------------------------------------------------------------------
```

```
        ValueError                                Traceback (most recent call␣
↪last)

        <ipython-input-64-9bca6e65e6ff> in <module>
    ----> 1 a.resize((4,))


        ValueError: cannot resize an array that references or is referenced
    by another array in this way.
    Use the np.resize function or refcheck=False
```

2.5 Sorting Data

[81]:
```
a = np.array([[4,3,5],[1,2,1]])
```

[82]:
```
b = np.sort(a, axis = 1)
```

[83]:
```
b
```

[83]:
```
array([[3, 4, 5],
       [1, 1, 2]])
```

[85]:
```
c = np.sort(a[0],axis=0)
```

[86]:
```
c
```

[86]:
```
array([3, 4, 5])
```

Atau bisa juga digunakan

[87]:
```
a.sort(axis=1)
```

[88]:
```
a
```

[88]:
```
array([[3, 4, 5],
       [1, 1, 2]])
```

Sorting dengan pengindeksan:

[89]:
```
a = np.array([4,3,1,2])
```

[90]:
```
j = np.argsort(a)
```

[91]:
```
j
```

[91]:
```
array([2, 3, 1, 0])
```

[92]:
```
a[j]
```

[92]:
```
array([1, 2, 3, 4])
```

Penentuan maksimum dan minimum:

[93]:
```
a = np.array([4,3,1,2])
```

[94]:
```
j_max = np.argmax(a)
```

[95]:
```
j_min = np.argmin(a)
```

[96]:
```
j_max, j_min
```

[96]:
```
(0, 2)
```

Sampai dengan di sini boleh dilanjutkan ke **Chapter 2 - Matplotlib**, tidak akan mengganggu kontinuitas pengajaran. Tetapi akan lebih baik lagi apabila setelahnya kembali lagi ke bagian ini untuk meneruskan yang belum dipelajari. _____

3 Lebih Jauh Menyangkut Array

Bagian ini terdiri dari: > * Lebih jauh menyangkut tipe data > * Tipe data terstruktur > * *Maskedarray*

3.1 Lebih jauh menyangkut tipe data

- Casting

Tipe data yang lebih besar akan menang pada operasi dengan tipe data beragam:

[97]:
```
np.array([1,2,3])+1.5 # akan menghasilkan tipe floating point
```

[97]:
```
array([2.5, 3.5, 4.5])
```

Assignment tidak akan merubah tipe data:

[102]:
```
a = np.array([1,2,3])
```

[103]:
```
a.dtype
```

[103]:
```
dtype('int64')
```

[104]:
```
a[0]=1.9
```

[105]:
```
a
```

[105]:
```
array([1, 2, 3])
```

a tidak berubah karena `1.9` dipotong menjadi `1`, pada saat assignment.

- *Forced Cast*:

[109]:
```
a = np.array([1.7, 0.2, -2.6])
```

```
[110]: b = a.astype(int)  # truncated ke integer
```

```
[111]: b
```

```
[111]: array([ 1,  0, -2])
```

- Pembulatan:

```
[116]: a = np.array([1.2, -1.4, -7.77, 3.8, 4.5])
```

```
[117]: b = a.round()
```

```
[118]: b
```

```
[118]: array([ 1., -1., -8.,  4.,  4.])
```

Atau:

```
[123]: c = np.around(a)
```

```
[124]: c
```

```
[124]: array([ 1., -1., -8.,  4.,  4.])
```

```
[126]: c.dtype
```

```
[126]: dtype('float64')
```

```
[127]: c = np.around(a).astype(int)
```

```
[128]: c
```

```
[128]: array([ 1, -1, -8,  4,  4])
```

```
[129]: c.dtype
```

```
[129]: dtype('int64')
```

3.2 Structured Data Types

Misalkan ingin dibuat struktur data sebagai berikut:

```
[130]: samples = np.
       ↪zeros((6,),dtype=[('sensor','S4'),('position',float),('value',float)])
```

```
[131]: samples.ndim
```

```
[131]: 1
```

```
[132]: samples.shape
```

```
[132]: (6,)
```

```
[136]: samples.dtype
```

```
[136]: dtype([('sensor', 'S4'), ('position', '<f8'), ('value', '<f8')])
```

```
[137]: samples.dtype.names
```

```
[137]: ('sensor', 'position', 'value')
```

```
[142]: samples[:]=[('ALFA',1,0.37),('BETA',1,0.11),('TAU',1,0.13),('ALFA',1,0.
       ↪37),('ALFA',3.0,0.11),('TAU',1.2,0.13)]
```

```
[143]: samples
```

```
[143]: array([(b'ALFA', 1. , 0.37), (b'BETA', 1. , 0.11), (b'TAU', 1. , 0.13),
              (b'ALFA', 1. , 0.37), (b'ALFA', 3. , 0.11), (b'TAU', 1.2, 0.13)],
             dtype=[('sensor', 'S4'), ('position', '<f8'), ('value', '<f8')])
```

```
[145]: samples[0]
```

```
[145]: (b'ALFA', 1., 0.37)
```

```
[146]: samples[5]
```

```
[146]: (b'TAU', 1.2, 0.13)
```

```
[147]: samples[0]['sensor']
```

```
[147]: b'ALFA'
```

```
[150]: samples[0]['sensor']='KAYAKNY' # terpotong menjadi 4 karakter
```

```
[151]: samples
```

```
[151]: array([(b'KAYA', 1. , 0.37), (b'BETA', 1. , 0.11), (b'TAU', 1. , 0.13),
              (b'ALFA', 1. , 0.37), (b'ALFA', 3. , 0.11), (b'TAU', 1.2, 0.13)],
             dtype=[('sensor', 'S4'), ('position', '<f8'), ('value', '<f8')])
```

```
[152]: samples[1][['position','value']]
```

```
[152]: (1., 0.11)
```

```
[153]: samples[['position','value']]
```

```
[153]: array([(1. , 0.37), (1. , 0.11), (1. , 0.13), (1. , 0.37), (3. , 0.11),
              (1.2, 0.13)],
             dtype={'names':['position','value'], 'formats':['<f8','<f8'],
       'offsets':[4,12], 'itemsize':20})
```

Fancy Indexing juga berlaku di sini:

```
[154]: samples[samples['sensor'] == b'ALFA']
```

[154]:
```
array([(b'ALFA', 1., 0.37), (b'ALFA', 3., 0.11)],
      dtype=[('sensor', 'S4'), ('position', '<f8'), ('value', '<f8')])
```

Untuk deskripsi yang lebih detail menyangkut struktur data dapat dilihat di * Scipy-Link.

4 Operasi Lanjut

Terdiri dari: > * Polinomial > * Loading file-file data

4.1 Polinomial

Numpy juga dapat merepresentasikan polinomial dengan basis yang berbeda-beda:

Misalkan, $3x^2 + 2x - 1$ dapat dinyatakan dengan:

[156]:
```
p = np.poly1d([3,2,-1])
```

[157]:
```
p(0)
```

[157]:
```
-1
```

[158]:
```
type(p)
```

[158]:
```
numpy.poly1d
```

[159]:
```
p.roots
```

[159]:
```
array([-1.        ,  0.33333333])
```

[160]:
```
p.order
```

[160]:
```
2
```

Contoh lain untuk *fitting* dengan polinomial

[161]:
```
x = np.linspace(0,1,20)
```

[162]:
```
y = np.cos(x)+0.3*np.random.rand(20)
```

[174]:
```
fxy = np.polyfit(x,y,1)  # menggunakan regresi linier, orde = 1
p = np.poly1d(fxy)
```

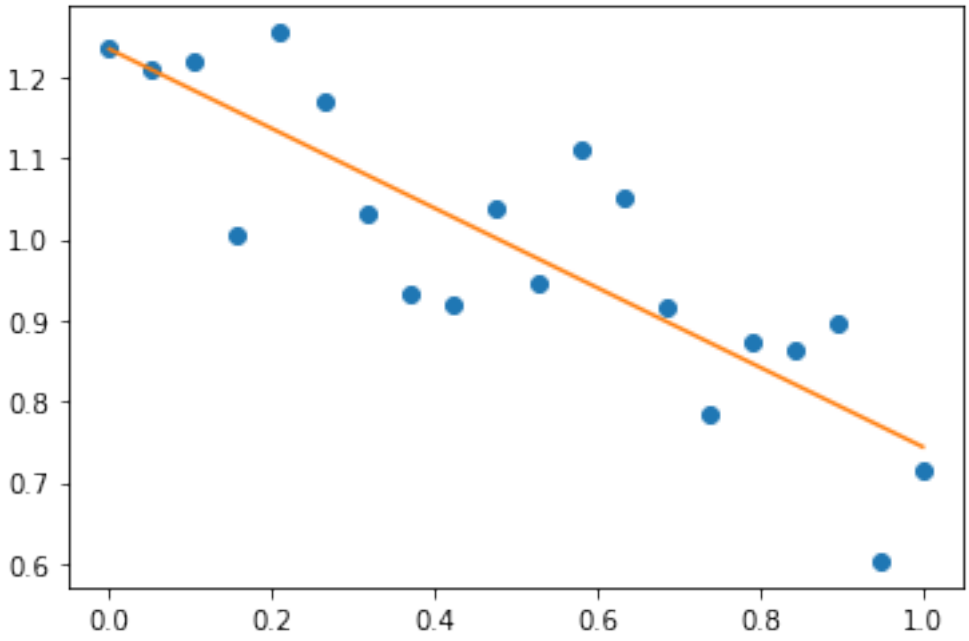
[169]:
```
fxy
```

[169]:
```
array([-0.49074405,  1.23474547])
```

[170]:
```
import matplotlib.pyplot as plt
```

[173]:
```
t = np.linspace(0,1,200)
plt.plot(x,y,'o',t,p(t),'-')
```

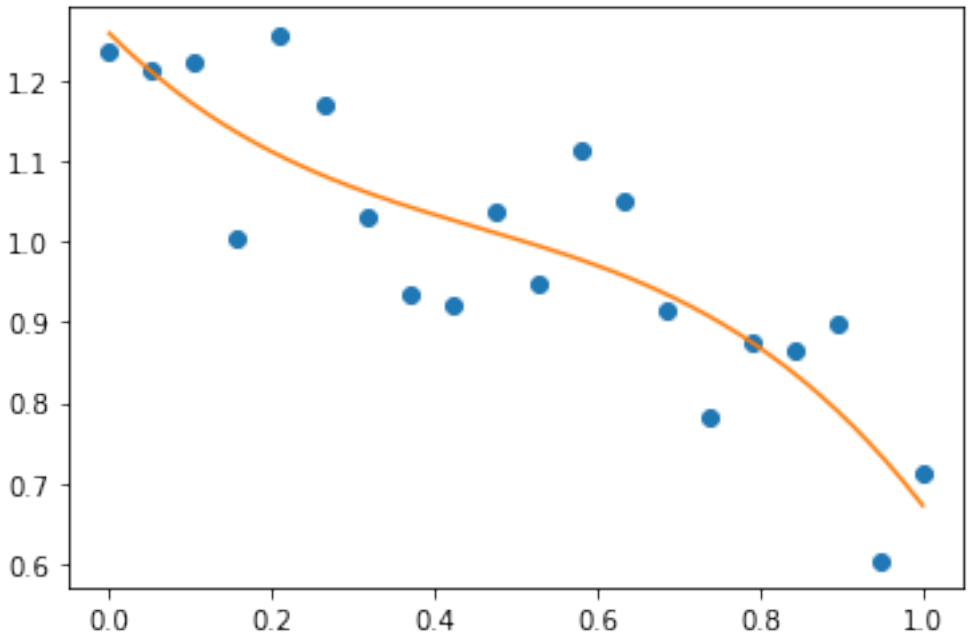
[173]:
```
[<matplotlib.lines.Line2D at 0x11ff40e50>,
 <matplotlib.lines.Line2D at 0x1209d7ad0>]
```



Dapat juga digunakan orde n:

[185]:
```
n = 3
fxy = np.polyfit(x,y,n)  # menggunakan regresi linier, orde = n
p = np.poly1d(fxy)
t = np.linspace(0,1,200)
plt.plot(x,y,'o',t,p(t),'-')
```

[185]:
```
[<matplotlib.lines.Line2D at 0x1213c2b10>,
 <matplotlib.lines.Line2D at 0x1213c2d50>]
```



- Polinomial dengan basis lain untuk : $3x^2 + 2x - 1$

[220]:
```
p = np.polynomial.Polynomial([-1,2,3]) # bandingkan dengan yang sebelumnya␣
↪poly1d, urutan penulisan koefisien terbalik
```

[221]:
```
p(0)
```

[221]:
```
-1.0
```

[222]:
```
p.roots()
```

[222]:
```
array([-1.        ,  0.33333333])
```

[223]:
```
p.degree()
```

[223]:
```
2
```

Contoh menggunakan basis *Chebyshev* untuk polinomial pada range `[-1,1]`

[224]:
```
x = np.linspace(-1,1,2000)
```

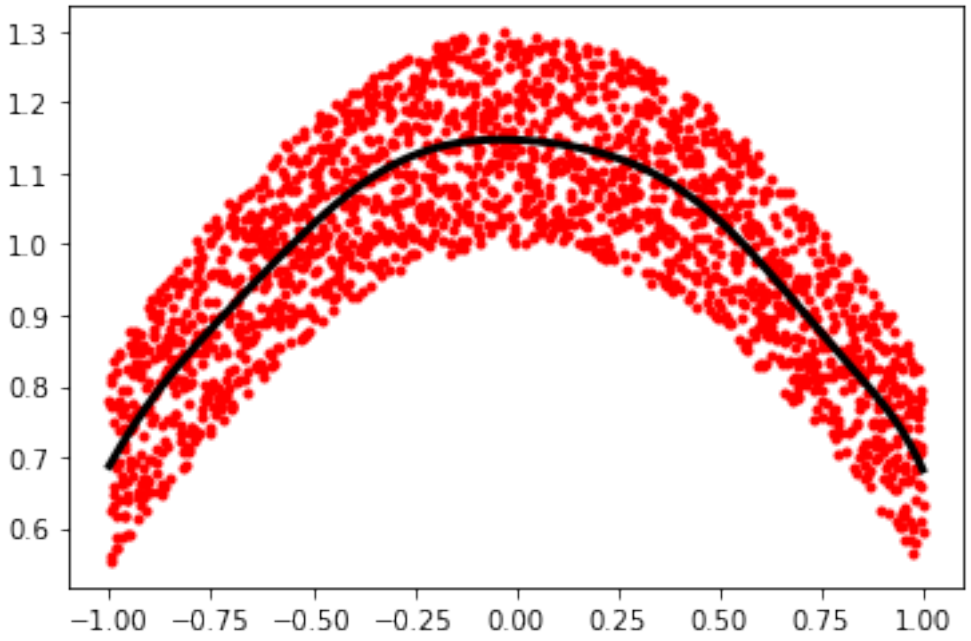
[225]:
```
y = np.cos(x)+0.3*np.random.rand(2000)
```

[226]:
```
n = 10
p = np.polynomial.Chebyshev.fit(x,y,n)
```

[227]:
```
t = np.linspace(-1,1,200)
```

[228]:
```
%matplotlib inline
plt.plot(x,y,'r.',t,p(t),'k-',lw=3)
```

[228]:
```
[<matplotlib.lines.Line2D at 0x1214c6a50>,
 <matplotlib.lines.Line2D at 0x121796e50>]
```



4.2 Loading File Data

- Membaca file text

[6]:
```
import numpy as np
data = np.loadtxt('data/populations.txt')
```

[7]:
```
data
```

[7]:
```
array([[ 1900., 30000.,  4000., 48300.],
       [ 1901., 47200.,  6100., 48200.],
       [ 1902., 70200.,  9800., 41500.],
       [ 1903., 77400., 35200., 38200.],
       [ 1904., 36300., 59400., 40600.],
       [ 1905., 20600., 41700., 39800.],
       [ 1906., 18100., 19000., 38600.],
       [ 1907., 21400., 13000., 42300.],
       [ 1908., 22000.,  8300., 44500.],
       [ 1909., 25400.,  9100., 42100.],
       [ 1910., 27100.,  7400., 46000.],
       [ 1911., 40300.,  8000., 46800.],
       [ 1912., 57000., 12300., 43800.],
       [ 1913., 76600., 19500., 40900.],
       [ 1914., 52300., 45700., 39400.],
       [ 1915., 19500., 51100., 39000.],
       [ 1916., 11200., 29700., 36700.],
       [ 1917.,  7600., 15800., 41800.],
```

```
       [ 1918., 14600.,  9700., 43300.],
       [ 1919., 16200., 10100., 41300.],
       [ 1920., 24700.,  8600., 47300.]])
```

[16]:
```
np.savetxt('pop2.txt', data[:,1:])
```

[17]:
```
data2 = np.loadtxt('pop2.txt')
```

[18]:
```
data2
```

[18]:
```
array([[30000.,  4000., 48300.],
       [47200.,  6100., 48200.],
       [70200.,  9800., 41500.],
       [77400., 35200., 38200.],
       [36300., 59400., 40600.],
       [20600., 41700., 39800.],
       [18100., 19000., 38600.],
       [21400., 13000., 42300.],
       [22000.,  8300., 44500.],
       [25400.,  9100., 42100.],
       [27100.,  7400., 46000.],
       [40300.,  8000., 46800.],
       [57000., 12300., 43800.],
       [76600., 19500., 40900.],
       [52300., 45700., 39400.],
       [19500., 51100., 39000.],
       [11200., 29700., 36700.],
       [ 7600., 15800., 41800.],
       [14600.,  9700., 43300.],
       [16200., 10100., 41300.],
       [24700.,  8600., 47300.]])
```

Catatan: Untu file teks yang cukup kompleks bisa digunakan: - np.genfromtxt - Menggunakan Python's I/O dan misalkan `regexp` untuk parsing (Python cukup tepat untuk hal-hal semacam ini)

- Membaca file gambar

[19]:
```
import matplotlib.pyplot as plt
```

[20]:
```
img = plt.imread('data/Gambar-MachineLearning.jpg')
```

[21]:
```
img.shape
```

[21]:
```
(470, 450, 3)
```

[22]:
```
img.dtype
```

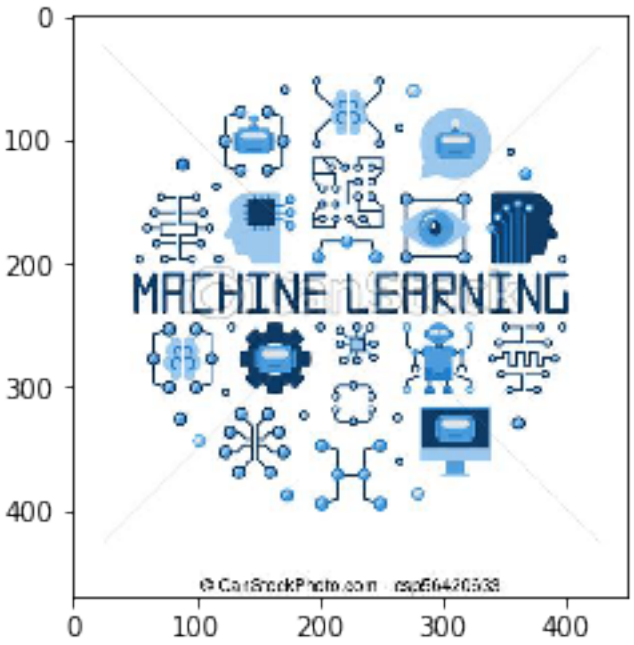
[22]:
```
dtype('uint8')
```

[23]:
```
img
```

[23]:
```
array([[[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       ...,

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]]], dtype=uint8)
```

[24]:
```
plt.imshow(img)
```

[24]: <matplotlib.image.AxesImage at 0x10a157790>



[26]:
```
img[:,:,0].shape
```

[26]: (470, 450)

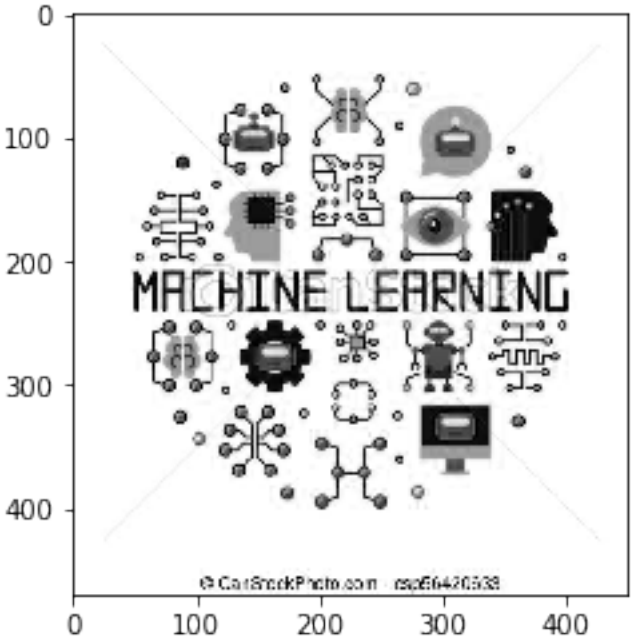
[27]:
```
plt.savefig('MesinLearning.png')
```

```
<Figure size 432x288 with 0 Axes>
```

[55]:
```
plt.imsave('MesinLearning0.png', img[:,:,0], cmap=plt.cm.gray)
```

[56]:
```
plt.imshow(plt.imread('MesinLearning0.png'))
```

[56]: <matplotlib.image.AxesImage at 0x11ed17710>



[62]:
```
from scipy import misc
```

[64]:
```
face = misc.face(gray=True)
```

[65]:
```
plt.imshow(face)
```

[65]:
```
<matplotlib.image.AxesImage at 0x11baf9650>
```