Algorithmes, Programmation, IA

Cours 8

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/algo-prog-ia-25

Plan

- affichage de fonctions
- descente du gradient stochastique
- régression linéaire revisitée
- présentation algébrique
- classification linéaire

Machine Learning [Andrew Ng] http://cs229.stanford.edu
http://coursera.org/share/5aa61ab89328fc47de71f57999bf14b2

Deepmath [Bodin & Récher] http://exo7.emath.fr/cours/livre-deepmath.pdf

Quelques rappels

- le cours utilise le langage Python et l'environnement Visual Studio Code. (`vscode`)
- et pour la partie IA, la bibliothèque Pytorch

- en Python, l'opérateur ***** permet d'aplatir une liste, un n-uplet ou un tableau de *numpy*

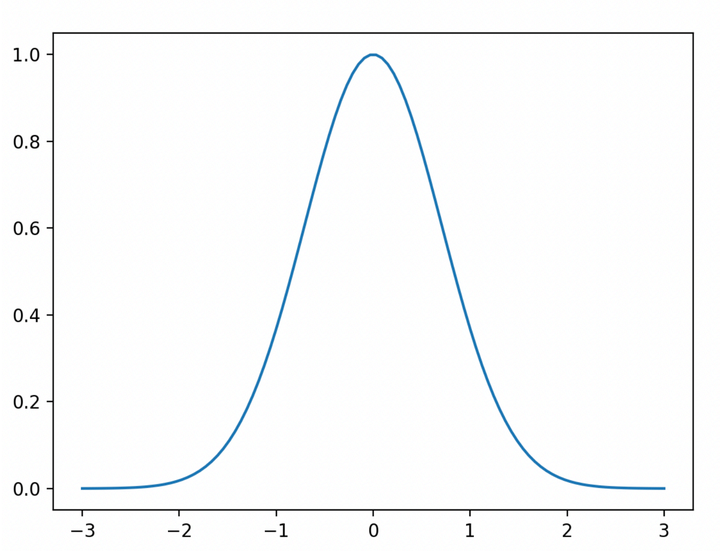
```
a = [i for i in range (8)]
print (a)
print (*a)
print (a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7])
```

```
[0, 1, 2, 3, 4, 5, 6, 7]
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
```

Bibliothèque `matplotlib`

- le module permet de tracer des courbes 2D et 3D à partir de Python et Numpy

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


def f(x):
    return np.exp(-x**2)

(a, b) = (-3, 3)
X = np.linspace(a,b,num=100)
Y = f(X)
plt.plot(X,Y)
plt.show()
```



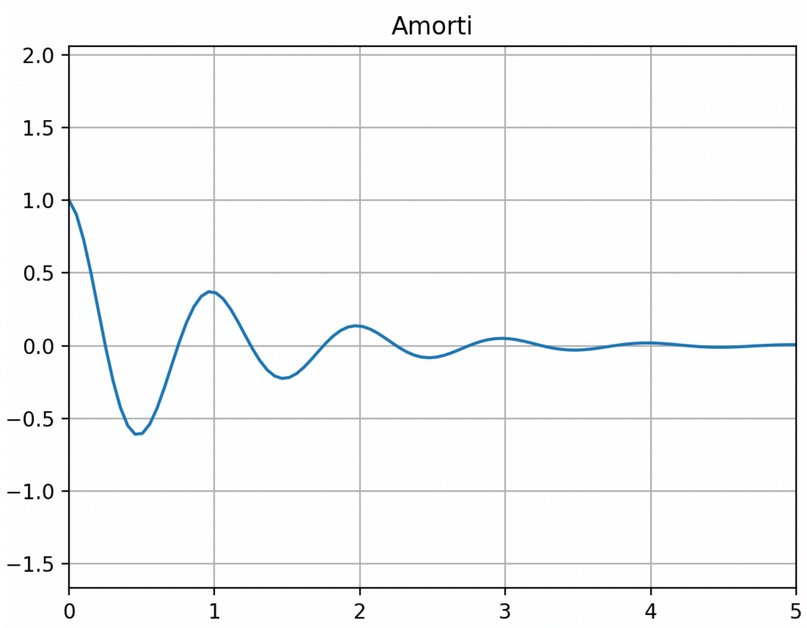
- ici, les majuscules sont utilisées pour des séries (`numpy.ndarray`), les minuscules pour les variables standard

  [une convention dans ce cours]

Bibliothèque `matplotlib`

- le module permet de tracer des courbes 2D et 3D à partir de Python et Numpy

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

def f(x):
    return np.exp(-x) * np.cos(2*np.pi*x)

(a, b) = (0, 5)
X = np.linspace(a, b, num=100)
Y = f(X)

plt.title('Amorti') # titre
plt.axis('equal') # repère orthonormé
plt.grid() # grille
plt.xlim(a,b) # bornes de l'axe des x
plt.plot(X,Y)
plt.savefig('amorti.png')
plt.show()
```

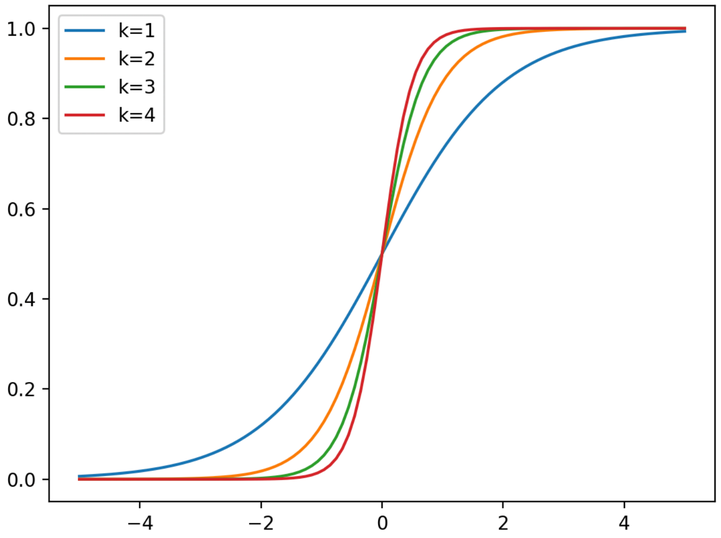


- ici, les majuscules sont utilisées pour des séries (`numpy.ndarray`), les minuscules pour les variables standard

Bibliothèque `matplotlib`

- le module permet de tracer des courbes 2D et 3D à partir de Python et Numpy

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


def f(x,k):
    return 1/(1+np.exp(-k*x))

(a, b) = (-5, 5)
X = np.linspace(a,b,num=100)
for k in range(1,5):
    Y = f(X,k)
    plt.plot(X,Y, label="k={}".format(k))

plt.legend()
plt.show()
```

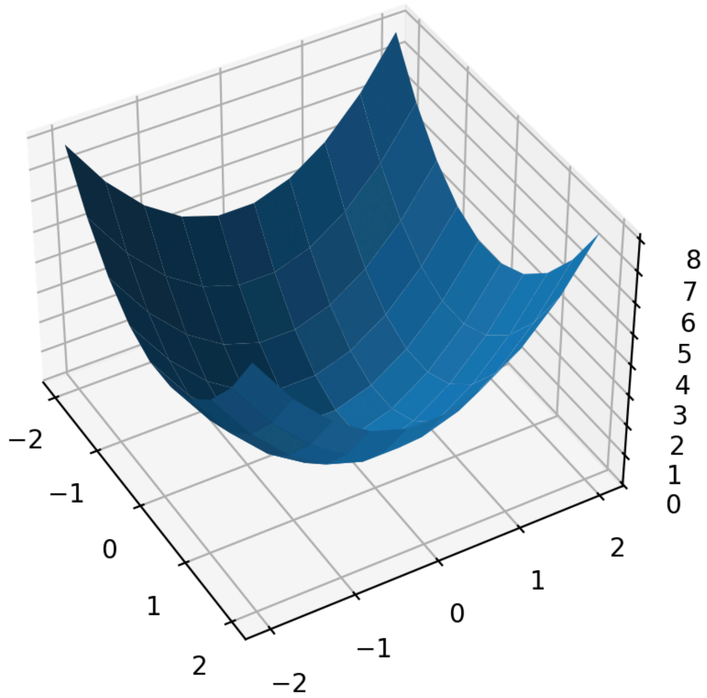


- ici, les majuscules sont utilisées pour des séries (`numpy.ndarray`), les minuscules pour les variables standard

Bibliothèque `matplotlib`

- enfin, un exemple d'affichage de surface 3D

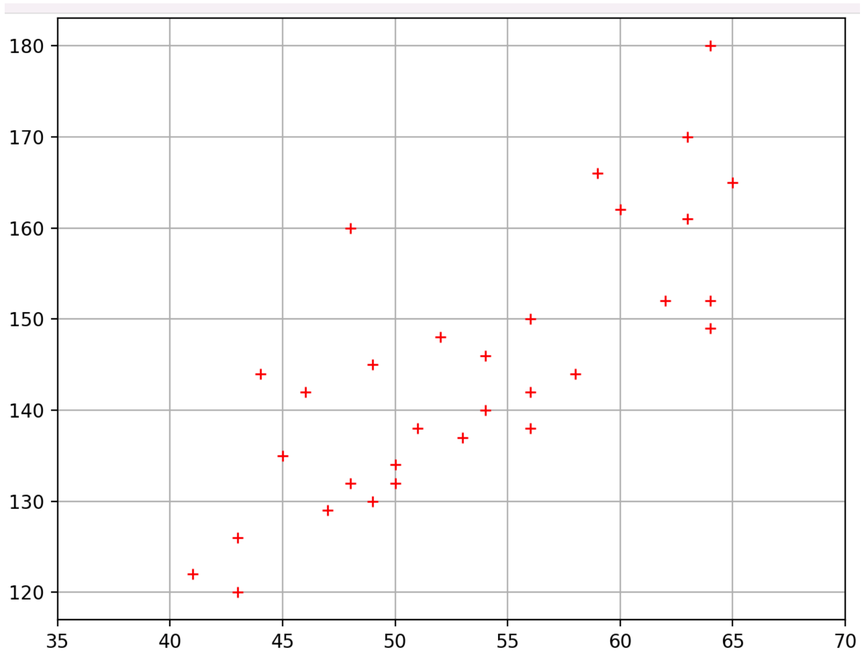
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def f(x):
    return X**2 + Y**2

n = 10
VX = np.linspace(-2.0, 2.0, n)
VY = np.linspace(-2.0, 2.0, n)
X,Y = np.meshgrid(VX, VY)
Z = f(X)

ax = plt.axes(projection='3d')
ax.view_init(40, -30)
ax.plot_surface(X, Y, Z)
plt.show()
```

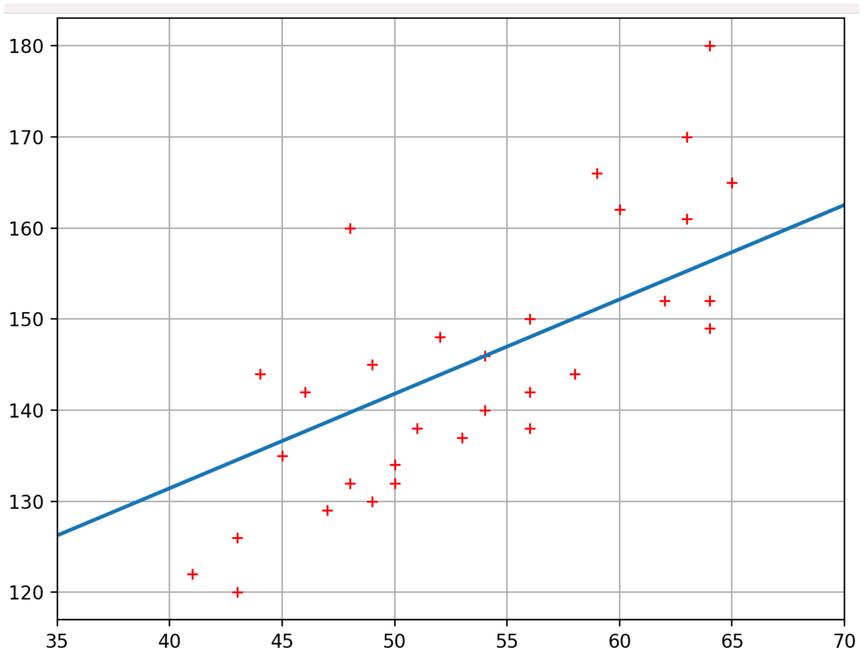


- ici, les majuscules sont utilisées pour des séries (`numpy.ndarray`), les minuscules pour les variables standard

Régression linéaire

- on se donne un jeu de données (ici la pression artérielle en fonction de l'âge)
- on cherche une approximation linéaire de ces données

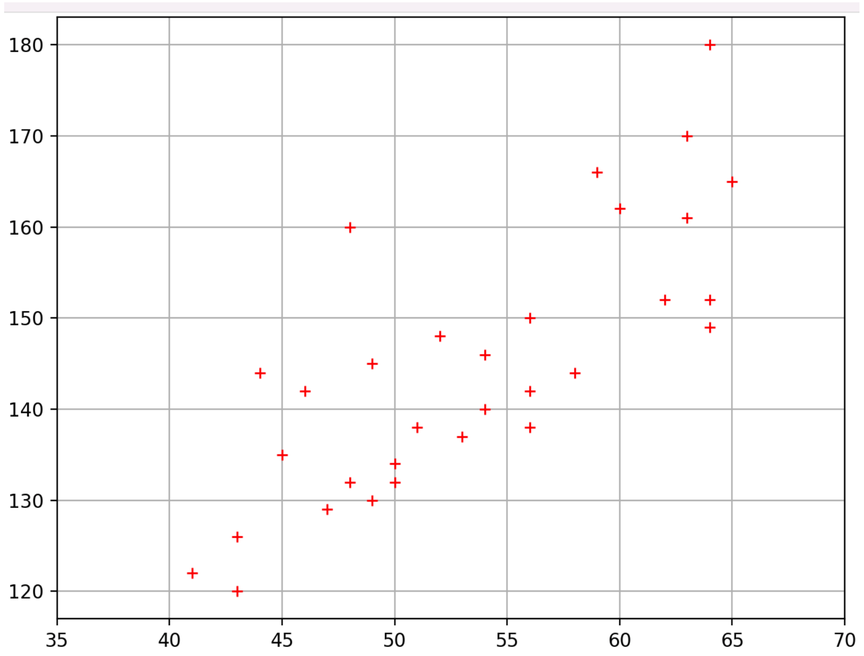




Régression linéaire

• le jeu de *n* données $(x^{(i)}, y^{(i)})$

```
dataset = [ (41, 122), (43, 120), (43, 126), (44, 144), (45, 135),
 (46, 142), (47, 129), (48, 132), (48, 160), (49, 130), (49, 145),
 (50, 132), (50, 134), (51, 138), (52, 148), (53, 137), (54, 140),
 (54, 146), (64, 149), (56, 138), (56, 142), (56, 150), (58, 144),
 (59, 166), (60, 162), (62, 152), (63, 161), (63, 170), (64, 152),
 (64, 180), (65, 165) ]
```



$x^{(0)} = 41,\ y^{(0)} = 122,\ x^{(1)} = 43,\ y^{(1)} = 120,\ x^{(2)} = 43,\ y^{(2)} = 126, \cdots$

• on cherche une fonction $f(x) = \theta_0 + \theta_1 x$
telle que la valeur absolue de la différence $|f(x^{(i)}) - y^{(i)}|$
soit la plus petite possible pour tous les $(x^{(i)}, y^{(i)})$

```
def f (x, theta0, theta1) :
    return theta0 + theta1 * x
```

• la différence à minimiser est la somme :

$$J = \sum_{i=0,n-1} \frac{1}{2} (f(x^{(i)}) - y^{(i)})^2$$

[calcul des moindres carrés]

```
def J(dataset) :
    global theta
    s = 0
    for (x, y) in dataset :
        s += 0.5 * (f (x, *theta) - y)**2
    return s
```

• *J* est aussi appelé le **coût** de l'approximation linéaire,
il dépend du jeu de données et des 2 **paramètres** θ_0 et θ_1

Régression linéaire

- on calcule les dérivées (partielles) du coût *J* par rapport à θ_0 et θ_1

$$\frac{\partial J}{\partial \theta_0} = \sum_{i=0,n-1} f(x^{(i)}) - y^{(i)}$$

```
def derJ0 (dataset) :
    global theta
    s = 0
    for (x, y) in dataset :
        s += (f (x, *theta) - y)
    return s
```

$$\frac{\partial J}{\partial \theta_1} = \sum_{i=0,n-1} x^{(i)} \left(f(x^{(i)}) - y^{(i)}\right)$$

```
def derJ1 (dataset) :
    global theta
    s = 0
    for (x, y) in dataset :
        s += x * (f(x, *theta) - y)
    return s
```

- on minimise le coût *J* par une descente du gradient en faisant varier les 2 paramètres θ_0 et θ_1

```
def descent (dataset) :
    global theta, alpha
    for i in range (100) :
        theta[0] -= alpha * derJ0 (dataset)
        theta[1] -= alpha * derJ1 (dataset)
```

- on choisit un bon *alpha* et 2 valeurs initiales arbitraires pour θ_0 et θ_1

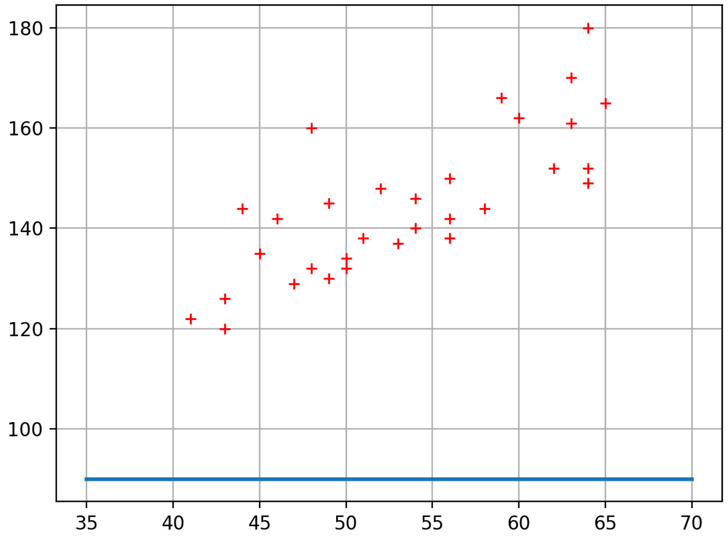
Régression linéaire

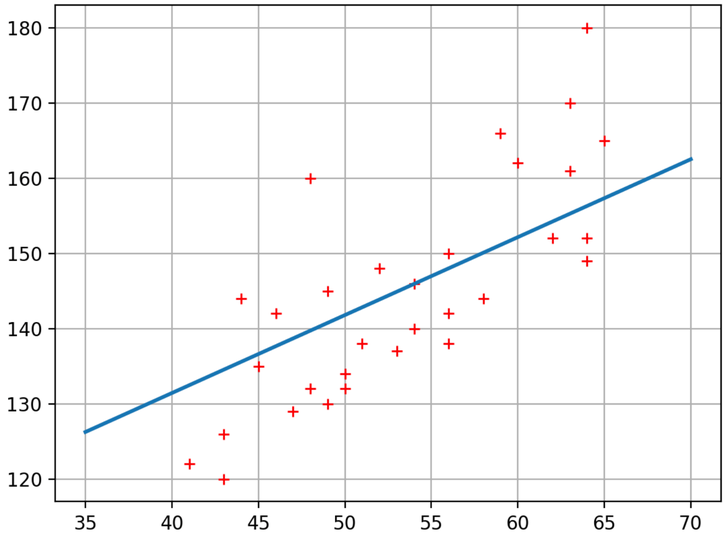
- on utilise matplotlib pour l'affichage

```
import matplotlib.pyplot as plt

def show (dataset) :
    global theta, xlim
    for (x, y) in dataset :
        plt.plot (x, y, marker='+', color='r')
    X = np.linspace (*xlim, 100)
    Y = f (X, *theta)
    plt.plot (X, Y, linewidth=2)
    plt.grid()
    plt.show()
```

- programme principal

```
xlim = (35, 70)
theta = [90, 0]; alpha = 0.00001
print (*theta, J(dataset)); show(dataset)
descent (dataset)
print (*theta, J(dataset)); show(dataset)
```





Descente stochastique

- à chaque variation de θ_0 et θ_1 , on parcourt l'ensemble des données qui peut être très grand

- la descente stochastique ne fait la variation que pour chaque donnée en espérant obtenir un résultat proche sans osciller

$$\frac{\partial J^{(i)}}{\partial \theta_0} = f(x^{(i)}) - y^{(i)}$$

$$\frac{\partial J^{(i)}}{\partial \theta_1} = x^{(i)} \left(f(x^{(i)}) - y^{(i)}\right)$$

```
def derJ0S (dataset, x, y) :
    global theta
    return (f (x, *theta) - y)
```

```
def derJ1S (dataset, x, y) :
    global theta
    return x * (f(x, *theta) - y)
```

- on itère la variation obtenue pour chaque donnée

```
def descentS (dataset) :
    global theta, alpha
    for i in range (100) :
        for (x, y) in dataset :
            theta[0] -= alpha * derJ0S (dataset, x, y)
            theta[1] -= alpha * derJ1S (dataset, x, y)
```

Régression linéaire avec *m* variables

- le jeu de *n* données $(x^{(i)}, y^{(i)})$ où *x* est fabriqué de *m* composantes

- par exemple, la quantité de CO_2 en fonction du volume et du poids d'une voiture

  $x = (x_1, x_2)$ x_1 est le volume, x_2 est le poids

  y est la quantité de CO_2

- on cherche une fonction $f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

Exercice 1 Trouver les paramètres $\theta_0, \theta_1, \theta_2$ par régression linéaire sur les données ci-contre.

En déduire la quantité de CO_2 pour une Renault Mégane: vol 1598, poids 1318

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1000	790	99
Mitsubishi	Space-Star	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Cooper	1500	1140	105
VW	Up!	1000	929	105
Skoda	Fabia	1400	1109	90
Mercedes	A-Class	1500	1365	92
Ford	Fiesta	1500	1112	98
Audi	A1	1600	1150	99
Hyundai	I20	1100	980	99
Suzuki	Swift	1300	990	101
Ford	Fiesta	1000	1112	99
Honda	Civic	1600	1252	94
Hundai	I30	1600	1326	97
Opel	Astra	1600	1330	97
BMW	1	1600	1365	99
Mazda	3	2200	1280	104
Skoda	Rapid	1600	1119	104
Ford	Focus	2000	1328	105
Ford	Mondeo	1600	1584	94
Opel	Insignia	2000	1428	99
Mercedes	C-Class	2100	1365	99
Skoda	Octavia	1600	1415	99
Volvo	S60	2000	1415	99
Mercedes	CLA	1500	1465	102
Audi	A4	2000	1490	104
Audi	A6	2000	1725	114
Volvo	V70	1600	1523	109
BMW	5	2000	1705	114
Mercedes	E-Class	2100	1605	115
Volvo	XC70	2000	1746	117
Ford	B-Max	1600	1235	104
BMW	2	1600	1390	108
Opel	Zafira	1600	1405	109
Mercedes	SLK	2500	1395	120

Régression linéaire avec *m* variables

```
dataset = [['Toyota', 'Aygo', 1000, 790, 99],
['Mitsubishi', 'Space-Star', 1200, 1160, 95],
['Skoda', 'Citigo', 1000, 929, 95],
['Fiat', '500', 900, 865, 90],
['Mini', 'Cooper', 1500, 1140, 105],
['VW', 'Up!', 1000, 929, 105],
['Skoda', 'Fabia', 1400, 1109, 90],
['Mercedes', 'A-Class', 1500, 1365, 92],
['Ford', 'Fiesta', 1500, 1112, 98],
['Audi', 'A1', 1600, 1150, 99],
['Hyundai', 'I20', 1100, 980, 99],
['Suzuki', 'Swift', 1300, 990, 101],
['Ford', 'Fiesta', 1000, 1112, 99],
['Honda', 'Civic', 1600, 1252, 94],
['Hundai', 'I30', 1600, 1326, 97],
['Opel', 'Astra', 1600, 1330, 97],
['BMW', '1', 1600, 1365, 99],
['Mazda', '3', 2200, 1280, 104],
['Skoda', 'Rapid', 1600, 1119, 104],
['Ford', 'Focus', 2000, 1328, 105],
['Ford', 'Mondeo', 1600, 1584, 94],
['Opel', 'Insignia', 2000, 1428, 99],
['Mercedes', 'C-Class', 2100, 1365, 99],
['Skoda', 'Octavia', 1600, 1415, 99],
['Volvo', 'S60', 2000, 1415, 99],
['Mercedes', 'CLA', 1500, 1465, 102],
['Audi', 'A4', 2000, 1490, 104],
['Audi', 'A6', 2000, 1725, 114],
['Volvo', 'V70', 1600, 1523, 109],
['BMW', '5', 2000, 1705, 114],
['Mercedes', 'E-Class', 2100, 1605, 115],
['Volvo', 'XC70', 2000, 1746, 117],
['Ford', 'B-Max', 1600, 1235, 104],
['BMW', '2', 1600, 1390, 108],
['Opel', 'Zafira', 1600, 1405, 109],
['Mercedes', 'SLK', 2500, 1395, 120]]
```

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1000	790	99
Mitsubishi	Space-Star	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Cooper	1500	1140	105
VW	Up!	1000	929	105
Skoda	Fabia	1400	1109	90
Mercedes	A-Class	1500	1365	92
Ford	Fiesta	1500	1112	98
Audi	A1	1600	1150	99
Hyundai	I20	1100	980	99
Suzuki	Swift	1300	990	101
Ford	Fiesta	1000	1112	99
Honda	Civic	1600	1252	94
Hundai	I30	1600	1326	97
Opel	Astra	1600	1330	97
BMW	1	1600	1365	99
Mazda	3	2200	1280	104
Skoda	Rapid	1600	1119	104
Ford	Focus	2000	1328	105
Ford	Mondeo	1600	1584	94
Opel	Insignia	2000	1428	99
Mercedes	C-Class	2100	1365	99
Skoda	Octavia	1600	1415	99
Volvo	S60	2000	1415	99
Mercedes	CLA	1500	1465	102
Audi	A4	2000	1490	104
Audi	A6	2000	1725	114
Volvo	V70	1600	1523	109
BMW	5	2000	1705	114
Mercedes	E-Class	2100	1605	115
Volvo	XC70	2000	1746	117
Ford	B-Max	1600	1235	104
BMW	2	1600	1390	108
Opel	Zafira	1600	1405	109
Mercedes	SLK	2500	1395	120

Présentation algébrique

- on peut regrouper les paramètres, variables et données dans des matrices

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \qquad x = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} \qquad X = \begin{bmatrix} {x^{(1)}}^T \\ \vdots \\ {x^{(n)}}^T \end{bmatrix} \qquad Y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

- calcul du coût

$$x^T = \begin{bmatrix} 1 & x_1 & \cdots & x_d \end{bmatrix}$$

$$f(x) = x^T\theta = \theta_0 + x_1\theta_1 + \cdots x_d\theta_d$$

$$f(X) - Y = \begin{bmatrix} f(x^{(1)}) - y^{(1)} \\ \vdots \\ f(x^{(n)}) - y^{(n)} \end{bmatrix} = X^T\theta - Y$$

$$J = \tfrac{1}{2}(f(X) - Y)^T(f(X) - Y) = \tfrac{1}{2}(X^T\theta - Y)^T(X^T\theta - Y)$$

- dérivée par rapport à θ

$$\nabla_\theta(J) = X^TX\theta - X^TY$$

- minimum **exact** quand la dérivée s'annule

$$\theta = (X^TX)^{-1}X^TY$$

Présentation algébrique

• minimum obtenu **exactement** en :

$$\theta = (X^T X)^{-1} X^T Y$$

Exercice 2 Trouver la solution exacte dans le cas du jeu de données avec une variable donnant la pression artérielle en fonction de l'âge. On a alors *d* = *2* et une seule variable *x*.

Le comparer au résultat obtenu par descente du gradient.

Rappel Dans le cas d'une matrice 2x2, on a:

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

Classification

• classer des points selon un critère binaire

$$f(x) = \theta^T x \in \{0, 1\}$$

• *f* n'est pas dérivable. On préfère utiliser la fonction **sigmoïde**

$$f(x) = \sigma(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

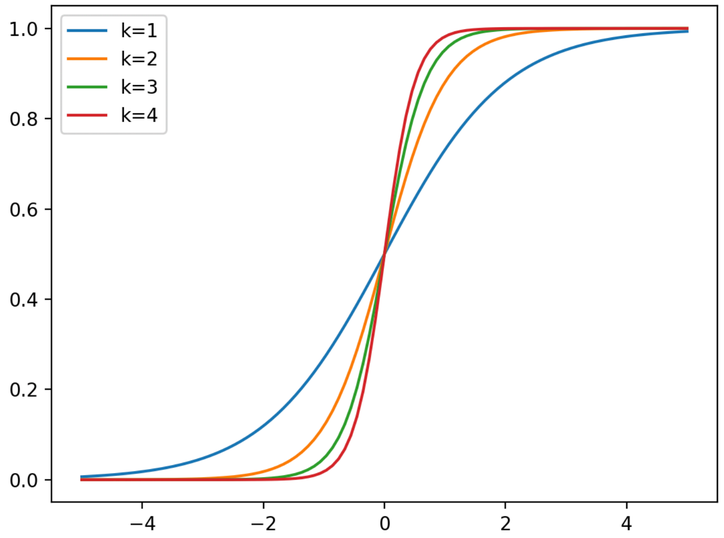
• *f* est alors dérivable et prend ses valeurs entre 0 et 1

• la dérivée de la fonction sigmoïde vérifie l'équation :

$$\sigma'(x) = \sigma(x)\,(1 - \sigma(x))$$

puisque

$$\sigma'(x) = \frac{1}{(1 + e^{-x})^2}\,(e^{-x}) = \frac{1}{1 + e^{-x}}\,(1 - \frac{1}{1 + e^{-x}})$$

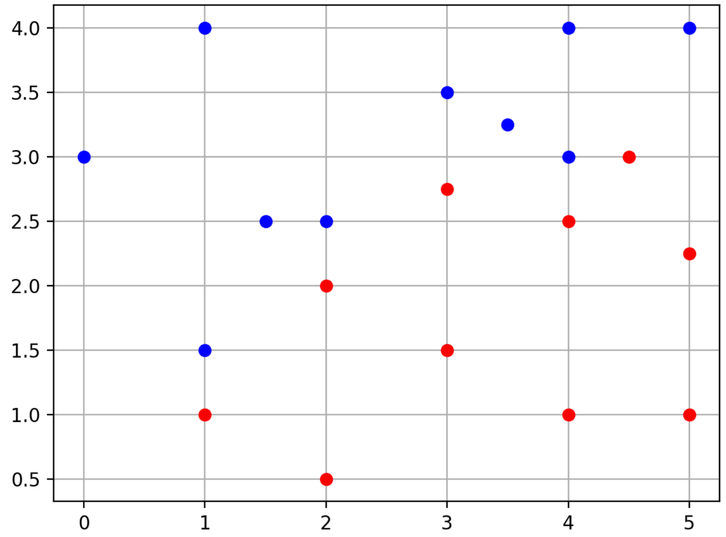


graphe de la fonction $\sigma(kx) = \frac{1}{1 + e^{-kx}}$

Classification

• le jeu de données est définit par deux ensembles de points rouges ou bleus dans le plan

```
bleus = [(0, 3), (1, 1.5), (1, 4), (1.5, 2.5), (2,2.5),
         (3, 3.5), (3.5, 3.25), (4,3), (4,4), (5,4)]

rouges = [(1,1), (2, 0.5), (2, 2), (3, 1.5), (3,2.75),
          (4,1), (4,2.5), (4.5, 3), (5, 1), (5, 2.25)]


dataset = [(0, 3, 0), (1, 1.5, 0), (1, 4, 0), (1.5, 2.5, 0),
           (2, 2.5, 0), (3, 3.5, 0), (3.5, 3.25, 0), (4, 3, 0),
           (4, 4, 0), (5, 4, 0), (1, 1, 1), (2, 0.5, 1),
           (2, 2, 1), (3, 1.5, 1), (3, 2.75, 1), (4, 1, 1),
           (4, 2.5, 1), (4.5, 3, 1), (5, 1, 1), (5, 2.25, 1)]
```



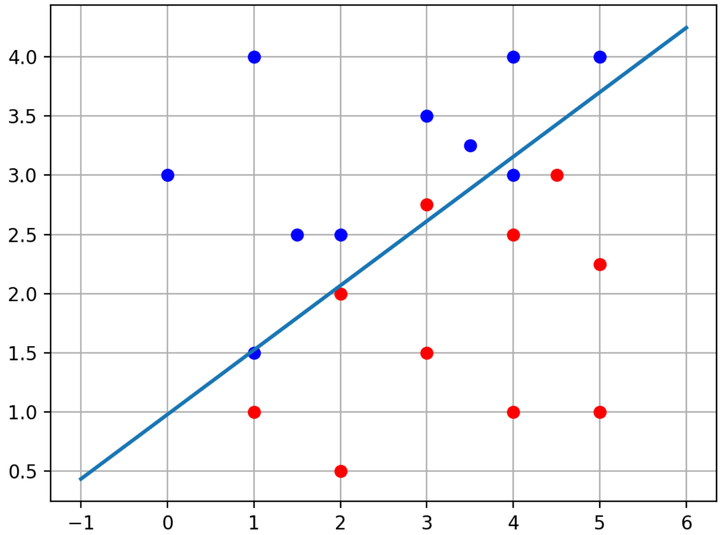
• pour chaque donnée $\left(x^{(i)}, y^{(i)}, z^{(i)}\right)$ la valeur bleu correspond à 0 et la valeur rouge à 1 pour $z^{(i)}$

Classification

- pour séparer par une **droite** les zones rouge et bleue, on cherche θ tel que :

(x, y) est rouge si $f(x, y) = \sigma(\theta_0 + \theta_1 x + \theta_2 y) > \frac{1}{2}$

(x, y) est bleu sinon



Remarque :

$$\sigma(\theta_0 + \theta_1 x + \theta_2 y) > \frac{1}{2} \quad \Longleftrightarrow \quad \theta_0 + \theta_1 x + \theta_2 y > 0$$

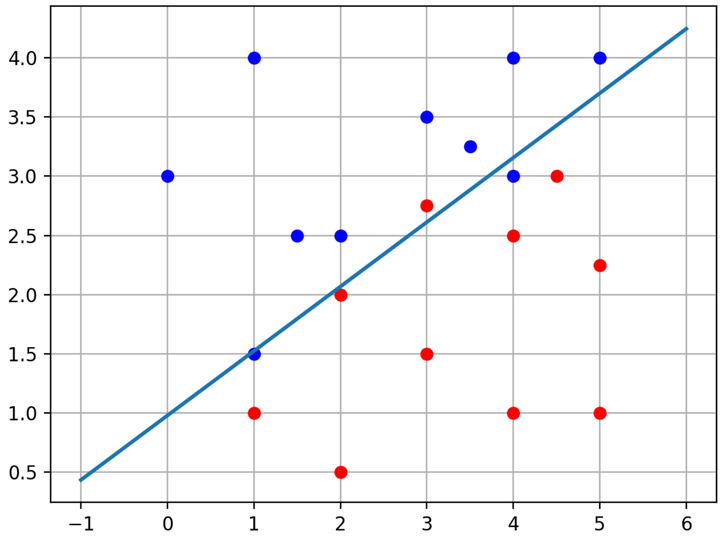
Classification

• l'erreur pour un θ donné est définie par la méthode des moindres carrés (*LMS - least mid squares*)

$$J = \sum_{i=0,n-1} \frac{1}{2} \left(f\left(x^{(i)}, y^{(i)}\right) - z^{(i)} \right)^2$$

• le minimum de l'erreur *J* est trouvé par descente du gradient
après 1000 itérations (stochastiques)

```
xlim = (-1, 6)
theta = [1.3, 0, 2]; alpha = 0.1
print (*theta, '--', J(dataset)); show(dataset)
descentS (dataset)
print (*theta, '--', J(dataset)); show(dataset)
```



Exercice 3 Programmer cette descente de gradient.

Exercice 4 Afficher en 3D les points *(x, y, z)* et le coût *J* en fonction de *x* et *y*.

Conclusion

- la régression linéaire permet des approximations linéaires d'un ensemble de données
- en ajoutant des fonctions d'**activation** comme la fonction **sigmoïde**, on étend sa puissance
- elle est à la base du fonctionnement des réseaux de neurones multi-couches