Exercice d'ajustement

Essayons ensuite d'ajuster la fonction `y(x) = ax + b`.

```
import numpy as np
import math
```

x = np.linspace(-math.pi, math.pi, 20)

```
print(x)
```

```
[-3.14159265 -2.81089869 -2.48020473 -2.14951076 -1.8188168  -1.48812284
 -1.15742887 -0.82673491 -0.49604095 -0.16534698  0.16534698  0.49604095
  0.82673491  1.15742887  1.48812284  1.8188168   2.14951076  2.48020473
  2.81089869  3.14159265]
```

Remarquez que c'est `linspace`, et non `linespace`. Cela fait partie d'un exemple de code dans un tutoriel `PyTorch`. Ces décimales peuvent ne pas être très intuitives.

```
x = np.linspace(0, 100, 20)
```

```
import numpy as np
import math

x = np.linspace(0, 100, 20)
y = np.linspace(0, 100, 20)

print(x)
print(y)
```

Ainsi, nous obtenons deux ensembles de données. Comment les représenter graphiquement ?

Cependant, il s'avère que `x` et `y` sont identiques.

```
x = np.random.rand(2)
print(x)
```

```
[0.06094295 0.89674607]
```

Je continue à apporter des modifications.

```
x = np.random.rand(2)*100
print(x)
```

```
[39.6136151  66.15534011]
```

Continuer à modifier.

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

x = np.random.rand(10) * 100
y = np.random.rand(10) * 100

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

```
[20.1240488  59.69327146 58.05432614  3.14092909 82.86411091 43.23010476
 88.09796699 94.42222486 58.45253048 51.98479507]
[58.7129098   1.6457994  49.34115933 71.13738592 53.09736099 15.4485691
 45.12200319 20.46080549 67.48555147 91.10864978]
```

On peut voir que c'est de `(20.1,58.7)` à `(59.7,1.6)` puis à `(58,49.3)`. Notez que bien que ce graphique semble désordonné, il suit toujours une certaine logique. Il s'agit d'un tracé en un seul trait.

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

x = np.random.rand(2)*100
y = np.random.rand(2)*100

print(x)
print(y)
```

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

Remarquez que les échelles de `x` et `y` changent constamment. Ainsi, deux lignes qui semblent identiques peuvent en réalité être différentes. Alors, comment déterminer les valeurs de `a` et `b` dans l'équation `y(x) = ax + b` ? Supposons que nous connaissons maintenant deux points sur cette ligne droite. Notez que nous pouvons résoudre cela sur une feuille de brouillon. En soustrayant les deux équations, nous éliminons `b` et trouvons `a`. Ensuite, en substituant `a` dans une des équations, nous trouvons `b`.

Cependant, est-il possible d'utiliser une méthode de devinette ? Utilisons la méthode de dichotomie. Essayons.

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

x = np.random.rand(2)*100
y = np.random.rand(2)*100

a_max = 1000
a_min = -1000
b_max = 1000
b_min = -1000

def cal_d(da, b):
    y0 = x[0] * da + b
    y1 = x[1] * da + b
    d = abs(y0 - x[0]) + abs(y1 - x[1])
    return d

def cal_db(a, db):
    y0 = x[0] * a + db
    y1 = x[1] * a + db
    d = abs(y0 - y[0]) + abs(y1 - y[1])
    return d

def avg_a():
    return (a_max + a_min) / 2
```

```
def avg_b():
    return (b_max + b_min) / 2

for i in range(100):
    a = avg_a()
    b = avg_b()
    max_d = cal_d(a_max, b)
    min_d = cal_d(a_min, b)
    if max_d < min_d:
        a_min = a
    else:
        a_max = a

    a = avg_a()
    max_db = cal_db(a, b_max)
    min_db = cal_db(a, b_min)
    if max_db < min_db:
        b_min = b
    else:
        b_max = b

print(x)
print(y)
print('a = ', avg_a())
print('b = ', avg_b())
print(avg_a() * x[0] + avg_b())
print(avg_a() * x[1] + avg_b())
```

Exécutez-le.

```
[42.78912791 98.69284173]
[68.95535212 80.89946202]
a =  11.71875
b =  -953.125
-451.68990725289063
203.4317390671779
```

Cependant, les résultats ont montré une grande divergence.

Simplifions le problème. Supposons que `y(x) = ax`. On nous donne un ensemble de valeurs `x`, `y`, et nous devons trouver `a`. Bien que nous puissions le calculer directement, essayons de deviner.

```
import numpy as np
import math
import matplotlib.pyplot as plt
from numpy.random import rand, randint
```

x = randint(100) y = randint(100)

```
a_max = 1000
a_min = -1000

def cal_d(da):
    y0 = x * da
    return abs(y0 - y)

def avg_a():
    return (a_max + a_min) / 2

for i in range(1000):
    a = avg_a()
    max_d = cal_d(a_max)
    min_d = cal_d(a_min)
    if max_d < min_d:
        a_min = a
    else:
        a_max = a

print(x)
print(y)
print(avg_a())
print(avg_a()*x)
```

Le résultat est encourageant. La prédiction était assez précise.

```
96
61
0.6354166666666667
61.00000000000001
```

Cependant, il est plus courant d'écrire `for i in range(15):`, ce qui permet d'itérer 15 fois avec une précision suffisante. Pourquoi ? Remarquez que nos variables `x` et `y` sont toutes deux comprises entre 0 et 100. Par conséquent, la valeur de `a` est également comprise entre 0 et 100. Par exemple, `x=1, y=99` et `x=99, y=1`. Ainsi, les valeurs initiales de `a_min` et `a_max` peuvent être optimisées. Notez que `1/99` est égal à `0.01`. Donc, pour atteindre une précision de l'ordre de `0.01`, il faut approximativement calculer 2^n ≈ 10000. En utilisant log2(10000) = 13.28, cela signifie qu'une valeur d'environ 14 est suffisante.