फिटिंग अभ्यास

अगले चरण में `y(x) = ax + b` को फिट करने का प्रयास करते हैं।

```
import numpy as np
import math

□ = □□.□□□□□□□□□(-□□□□.□□, □□□□.□□, 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]
```

ध्यान दें कि यह `linspace` है, `linespace` नहीं। यह `PyTorch` ट्यूटोरियल के एक उदाहरण का हिस्सा है। ये दशमलव संख्याएँ शायद बहुत स्पष्ट नहीं हैं।

```
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)
```

इस तरह से हमें दो सेट डेटा मिलता है। इसे ग्राफ़िकल रूप में कैसे प्रदर्शित किया जाए?

हालांकि, आश्चर्यजनक रूप से `x` और `y` एक जैसे हैं।

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

```
[0.06094295 0.89674607]
```

फिर इसे संशोधित करें।

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

(यह कोड ब्लॉक को हिंदी में ट्रांसलेट नहीं किया गया है क्योंकि यह प्रोग्रामिंग कोड है और इसे बदलने की आवश्यकता नहीं है।)

```
[39.6136151  66.15534011]
```

जारी रखें और सुधार करें।

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

□ = □□.□□□□□□□.□□□□□(10)100 □ = □□.□□□□□□□.□□□□□(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]
```

देखा जा सकता है कि यह `(20.1,58.7)` से `(59.7,1.6)` और फिर `(58,49.3)` तक जाता है। ध्यान दें कि हालांकि यह ग्राफ बहुत अव्यवस्थित दिखता है, फिर भी इसमें एक पैटर्न है। यह एक ही स्ट्रोक में बनाया गया है।

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

□ = □□.□□□□□□□.□□□□□(2)100 □ = □□.□□□□□□□.□□□□□(2)100

print(x)
print(y)

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

ध्यान दें कि `x` और `y` के स्केल हमेशा बदलते रहते हैं। इसलिए, दो सीधी रेखाएं जो एक जैसी दिखती हैं, वास्तव में अलग हो सकती हैं। तो हम `y(x) = ax + b` में `a` और `b` कैसे निकालें? मान लें कि हमें इस रेखा के दो बिंदु पता हैं। ध्यान दें कि हम इसे ड्राफ्ट पेपर पर हल कर सकते हैं। दो समीकरणों को घटाकर `b` को हटा दें और `a` निकालें। फिर `a` को एक समीकरण में प्रतिस्थापित करके `b` निकालें।

हालांकि, क्या अनुमान लगाने की विधि का उपयोग किया जा सकता है? द्विआधारी खोज (बाइनरी सर्च) का उपयोग करके देखें। कोशिश करके देखें।

```
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 - y[0]) + abs(y1 - y[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
```

यह फ़ंक्शन `cal_db` दो पैरामीटर `a` और `db` लेता है और एक मान `d` लौटाता है। यह फ़ंक्शन `x` नामक एक सूची के पहले दो तत्वों का उपयोग करके `y0` और `y1` की गणना करता है। फिर यह `y0` और `y[0]` के बीच तथा `y1` और `y[1]` के बीच के अंतर का निरपेक्ष मान लेकर उन्हें जोड़ता है और परिणाम `d` के रूप में लौटाता है।

```
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())
```

चलाएं।

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

परिणाम में काफी अंतर होता है।

आइए समस्या को सरल बनाएं। मान लीजिए कि `y(x) = ax` है। हमें `x` और `y` के एक सेट दिए गए हैं, और हमें `a` का मान ज्ञात करना है। हालांकि हम इसे सीधे गणना कर सकते हैं, लेकिन आइए इसे अनुमान लगाकर करने का प्रयास करें।

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

□ = □□□□□□□□(100) □ = □□□□□□□□(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)
```

परिणाम प्रसन्नताप्रद है। अनुमान बहुत सटीक था।

```
96
61
```

```
0.6354166666666667
61.00000000000001
```

(नोट: कोड ब्लॉक में संख्याएं और गणितीय मान हैं, इसलिए उन्हें अनुवादित नहीं किया गया है।)

हालांकि, आमतौर पर इसे इस तरह लिखा जाता है `for i in range(15):`, जो 15 बार पुनरावृत्ति करता है और यह काफी सटीक होता है। क्यों? ध्यान दें कि हमारे `x` और `y` दोनों 0 से 100 के बीच हैं। इसलिए `a` का मान भी 0 से 100 के बीच होगा। उदाहरण के लिए, `x=1, y=99` और `x=99, y=1`। इसलिए `a_min` और `a_max` के प्रारंभिक मानों को थोड़ा ऑप्टिमाइज़ किया जा सकता है। ध्यान दें कि `1/99` लगभग `0.01` है। इसलिए शायद `0.01` की सटीकता तक पहुंचने के लिए, हमें लगभग 2^□ □ 10000 की गणना करनी होगी। □□□2(10000) = 13.28। इसका मतलब है कि इसे लगभग 14 पर सेट करना पर्याप्त होगा।