Fitting Practice

This blog post was translated by Mistral

Try fitting `y(x) = ax + b` next.

In this context, "拟合" (fitting) refers to the process of approximating a relationship between two vari

```
II. import math

III. x = np.linspace(-1.570796326795, 1.570796326795, 20)

IV. print(x)

[...]
```

Translation:

I. Import numpy as np, import math
II.
III. Assign to variable x, a sequence of 20 evenly spaced values ranging from - to
IV. Print the sequence

~~~
[[-3.14159265, -2.81089869, -2.48020473, -2.14951076, -1.8188168, -1.48812284,
  -1.16559676, -0.84316047, -0.52359878, -0.2058861, 0.2058861, 0.52359878,
   0.84316047, 1.16559676, 1.48812284, 1.8188168, 2.14951076, 2.48020473,
   2.81089869, 3.14159265]
```[0.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, 45.0, 50.0, 55.0, 60.0, 65.0, 70.0, 75.0, 80.0,
~~~

The numbers in the text are outputs from a `numpy linspace()` function, which generates evenly spaced n

Here's the English translation without any Chinese characters or punctuation:

```
-1.15742887 -0.82673491 -0.49604095 -0.16534698 0.16534698 0.49604095 0.82673491 1.15742887 1.48812284
import math

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

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

However, it turns out that x and y are the same.

[Image of a line]

Despite this,

[Note: The Chinese text does not provide any English translation for the image or the text "Despite thi

Here's the English translation of the Chinese text without any Chinese characters or punctuation:

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

Output:
[0.06094295 0.89674607]:
Here is the English translation of the Chinese text without any Chinese characters or punctuation:

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

Output:
[0.39613615 0.6615534 ] continue to modify.

Here's the Python code without any Chinese characters or punctuation:

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

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

continue modify.1. plt.plot(x, y)
```

```
2. plt.show()

# Inputs:
# x: 1xN array or list of x-values
# y: 1xN array or list of y-values

# This code snippet uses matplotlib library to plot a line chart with x and y data. The plt.plot() func

# The following numbers are the x and y values respectively, which should be replaced by the actual dat

# x: [20.1240488, 59.69327146, 58.05432614, 3.14092909, 82.86411091, 43.23010476,
#     88.09796699, 94.42222486, 58.45253048, 51.98479507]

# y: [58.7129098, 1.6457994, 49.34115933, 71.13738592, 53.09736099, 15.4485691,
#     [53.05437123, 87.6892582, 45.6789654, 82.6452012, 66.03452356, 27.8962677,
#     78.94655813, 35.53394168, 67.26885836, 49.13991689]] The given points are approximately `(45.122,
```

In English, the given numbers are approximately:

[45.122, 20.461, 67.485, 91.109]

To convert these numbers to cartesian coordinates, we can use the following formula:

```
x = [45.122, 20.461, 67.485, 91.109]
y = [20.461, 67.485, 91.109, ?]
```

We can find the y-coordinate of the last point by using the slope-intercept form of a line:

```
m = (y2 - y1) / (x2 - x1)
b = y1 - m * x1
```

where (x1, y1) = (20.461, 67.485) and (x2, y2) = (91.109, ?)

```
m = (?, 91.109 - 67.485) / (91.109 - 20.461)
b = 67.485 - m * 20.461
```

Solving for m and b, we get:

m = 1.4558536318544855

b = 22.35418333868325

So, the last point is approximately (91.109, 113.7105170248278).

Therefore, the points form a line approximately passing through the points (45.122, 20.461), (20.461, 6

Generates random numbers x and y, each with 2 elements and values between 0 and 100 using NumPy's rando

Prints the values of x and y.

Plots a line graph of x against y using matplotlib.pyplot.

Displays the graph. I. Image of x1:

II. Image of y11:

Note: The text does not contain any Chinese characters or punctuation, only image descriptions and Engl

II. However, can we use a guessing method? Let's try using the binary search method.

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

# Given points
x1, y1 = 1, 2
x2, y2 = 3, 5

# Find slope (coefficient a)
a = (y2 - y1) / (x2 - x1)

# Find y-intercept (coefficient b) using one of the given points
```

```
b = y1 - a * x1

# Create x and y arrays
x = np.linspace(np.min([x1, x2]), np.max([x1, x2]), 100)
y = a * x + b

# Plot the line
plt.plot(x, y, 'r-')
plt.scatter(x1, y1, color='g')
plt.scatter(x2, y2, color='g')
plt.xlabel('x')
plt.ylabel('y')
plt.show()

# Binary search for the y-intercept (coefficient b)
low = np.min(np.min(np.abs(np.array([x1, x2])), np.abs(y1 - a * x1)))
high = np.max(np.max(np.abs(np.array([x1, x2])), np.abs(y1 - a * x1)))

while low <= high:
 mid = (low + high) / 2
 error = abs(np.array([x1, x2]) * a - np.array([y1, y2]) + mid)
 if np.all(error < 0):
 low = mid + 1
 else:
 high = mid

 b = mid
 print('The y-intercept (coefficient b) is approximately:', b)
```

This Python script finds the coefficients `a` and `b` for the given line using the method described in

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

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

```
def cal_d(da, b):
# Calculate the difference between two arrays
return np.abs(da - b)

# Ensure 'da' and 'b' have the same shape
if len(da.shape) != len(b.shape):
raise ValueError("Shape mismatch: 'da' and 'b' must have the same shape")

# Calculate the difference element-wise
result = np.abs(da - b)

# Clamp the results to the range [a_min, a_max] and [b_min, b_max] respectively
result = np.minimum(np.maximum(result, a_min*np.ones_like(result)), b_max*np.ones_like(result))

return result y0 = x[0] * a + db;
y1 = x[1] * a + db;
d = abs(y0-y[0]) + abs(y1-y[1]);
return d;
```

Function for calculating d with different a value:

```
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():
# calculate the average of a_max and a_min
return (a_max + a_min) / 2

def avg_b():
# calculate the average of b_max and b_min
return (b_max + b_min) / 2

for i in range(100):
# assign the average of a to variable a, and the average of b to variable b
a = avg_a()
```

```
b = avg_b() max_d = calculation_d(a_max, b)
min_d = calculation_d(a_min, b)
if max_d < min_d:
a_min = a
else:
a_max = a

a = average_a()
max_db = calculation_db(a, b_max)
min_db = calculation_db(a, b_min) if max_db < min_db:
else:

print(x[0])
print(y)
print("a = ")
print(avg_a())
print("b = ")
print(avg_b())
print(avg_a() * x[0] + avg_b())

If max_db is less than min_db, then assign b_min as b. Otherwise, assign b_max as b.
Print x[0], y, average of a, average of b, and the sum of their averages multiplied by x[0]. print(avg_a
# Output:
# print(avg_a() * 98.69284173 + avg_b())

# Translation:
print(avg_a() * x[1] + avg_b())

# Function definitions:
def avg_a():
 a = 11.71875
 return a

def avg_b():
 b = -953.125
 return b
```

```
# Input:
x = [42.78912791, 98.69284173]

# Calculation:
print(avg_a() * x[1] + avg_b())
# Output: 1035.538438625
```

So, the English translation of the given Chinese code without any Chinese characters or punctuation would be:

```
print(avg_a() * x[1] + avg_b())
```

With the provided function definitions and input, the output of the code would be `1035.538438625`.: The results show significant differences.

Let' s simplify the problem. Given a function `y(x) = ax`, find `a` with a given set of `x, y`. Although we can directly calculate it, let' s guess. import numpy as np import math import matplotlib.pyplot as plt from numpy.random import rand, randint

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

No English translation needed as the code is already in English. a_max = 1000

a_min = -1000

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

def avg_a(): return (a_max + a_min) / 2: For i in range of 1000: ⋯..: avg_a = average_function() ⋯..: max_d = calculate_d(a_max) ⋯..: min_d = calculate_d(a_min) ⋯..: if max_d < min_d: ⋯⋯: a_min = avg_a ⋯⋯: else: ⋯⋯: a_max = avg_a

English Translation:

For i from 0 to 999: assign a as the average value returned by avg_a() assign max_d as the difference value calculated by cal_d with the current value of a_max assign min_d as the difference value calculated by cal_d with the current value of a_min if max_d is less than min_d: assign the current value of avg_a to a_min else: assign the current value of avg_a to a_max- print(x) - print(y) - print(avg_a()) - print(avg_a() * x)

Output:

96 “‘

This code snippet is written in Python. The Chinese text translates to “The result is delightful. I guessed it correctly.” However, the text does not provide any information related to the code itself. Therefore, the English translation of the code remains the same as the original. However, writing it as `for i in range(15)` iterates more accurately, about 15 times. Why is that? Notice that both `x` and `y` are within the range of 0 to 100. Therefore, the `a` value is also within 0 to 100. For example, `x=1, y=99` and `x=99, y=1`. So, the initial value of `a_min` and `a_max` can be optimized. Since `1/99` is approximately `0.01`, the accuracy is likely to be around 2^n being roughly equal to 10000. The logarithm of 10000 to the base 2 is approximately 13.28. This means setting it to around 14 would be sufficient.

Therefore, the English translation of the provided Chinese text is:

However, writing it as `for i in range(15)` iterates more accurately, about 15 times. Why is that? Notice that both `x` and `y` are within the range of 0 to 100. Therefore, the `a` value is also within 0 to 100. For example, `x=1, y=99` and `x=99, y=1`. So, the initial value of `a_min` and `a_max` can be optimized. Since `1/99` is approximately `0.01`, the accuracy is likely to be around 2^n being roughly equal to 10000. The logarithm of 10000 to the base 2 is approximately 13.28. This means setting it to around 14 would be sufficient.