AI VIET NAM – COURSE 2022

Module 1 – Exam

Ngày 8 tháng 7 năm 2022

1. **Viết các dòng code theo yêu cầu sau để hoàn thành function apply_CLT() mẫu trong link:**

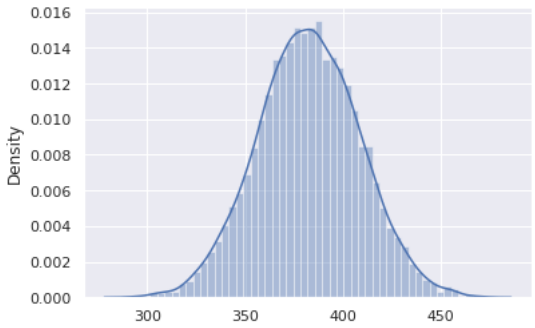
   »Nếu các bạn không quan tâm ý nghĩa của function này thì có thể bỏ qua phần này và sang phần CÁC BƯỚC THỰC HIỆN để thực hiện theo đề bài yêu cầu

   Central Limit Theorem (một định lý quang trọng trong xác suất): nói rằng nếu bạn có một population với mean và standard deviation và lấy một lượng đủ lớn các samples từ population (replacement), thì distribution của trung bình (mean) các samples sẽ xấp xỉ normal distribution (tương tự hình 1). Function apply_CLT() giúp chứng minh điều này.

   CÁC BƯỚC THỰC HIỆN

   - Đề bài cho trước ba tham số **num_of_samples** (population) = 10000, **sample_size** = 30, **side** = 500
   - Các bạn chỉ thực hiện viết bốn line code ở bốn vị trí (12, 17, 20, và 26) như trong link hoặc code mẫu bên dưới.
   - **Line 12**: giá trị mean nên được khởi tạo với giá trị bao nhiêu để thực hiện ở line 26 thì mean là tổng các distance và line 28 để tìm giá trị trung bình.
   - **Line 17**: tạo ra một giá trị **x** random trong range [0, **side**-1] (cụ thể [0, 499] trong bài này)
   - **Line 20**: tạo ra một giá trị **y** random trong range [0, **side**-1] (cụ thể [0, 499] trong bài này)
   - **Line 26**: các bạn phải thực hiện biến **mean** là tổng của các distance. Để khi code ở line 28 thực hiện được việc tính **mean** như công thức **mean** : $\mu = \frac{1}{n}\sum_{i=1}^{n} t_i$ where n is sample_size
   - Các bạn có thể kiểm tra kết quả của mình đúng hay sai bằng cách thực hiện code ở line 35, 36, 43, 44, 45 và kết quả sẽ đúng nếu thu được hình tương tự như hình 1

```python
import random
import math

num_of_samples = 10000
sample_size = 30
side = 500

def apply_CLT():
    means = []
    for _ in range(num_of_samples):
        ################ YOUR CODE HERE ###############
        mean =  # Write correct initial value
        for _ in range(sample_size):
            # generate a random point
            ################ YOUR CODE HERE ###############
            # x in range [0, 499] (use side variable)

            ################ YOUR CODE HERE ###############
            # y in range [0, 499] (use side variable)

            # distance to (0,0)
            distance = math.sqrt(x*x + y*y)

            ################ YOUR CODE HERE ###############
            # find total value (add all distance value to mean)

        # normalize to get mean
        mean = mean/sample_size

        # add to list
        means.append(mean)
    return means

# TEST
means = apply_CLT()
means
# Output
>> [384.7486582209122,
 354.722784658631,
 378.9724668716197,
 ...]

import seaborn as sns
sns.set_theme()
sns.distplot(apply_CLT())
```



Hình 1: Kết quả khi dùng code mẫu ở line 35, 36, 43, 44, 45

2. **Viết function để thực hiện các phép tính variance theo công thức sau:**

$$\boldsymbol{x} = [x_1, x_2, x_3, ..., x_n], \quad \forall \boldsymbol{x} \in \mathbb{R}^n$$

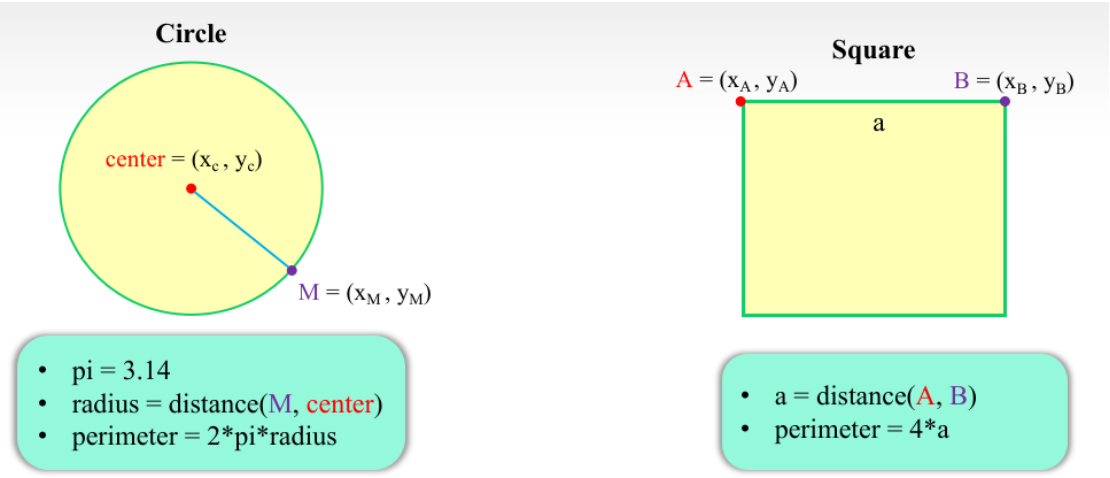
$$\textbf{mean}: \ \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\textbf{variance}: \ var(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

- **Input:** một list gồm có **n** element, mỗi element là một real number. $\mathbf{n} > 0$
- **Output:** Trả về variance ($var(\boldsymbol{x})$)
- **Note**: Các bạn phải xây dựng hàm từ đầu chỉ được sử dụng các built-in function (sum, len)

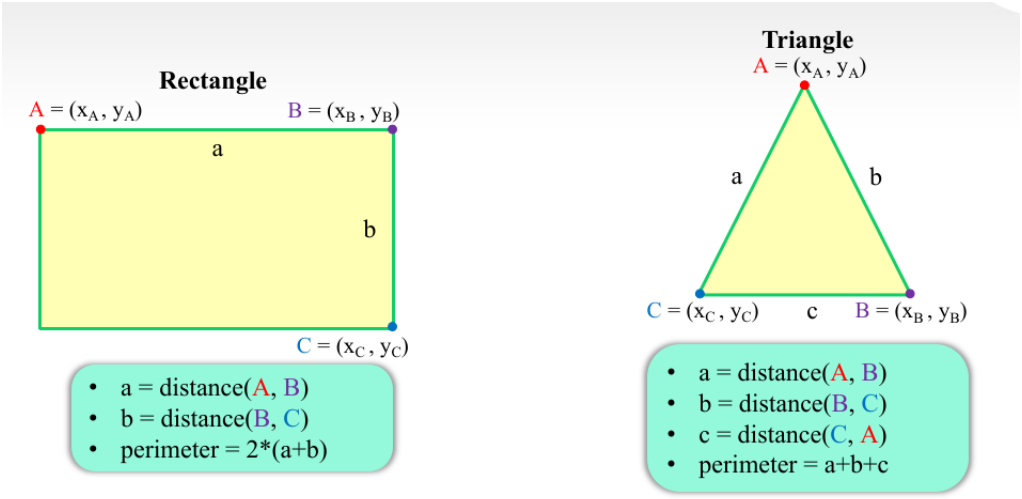
```
# Example
x = [5, 3, 6, 7, 4]
calc_variance(x)
#output
>> 2.0

x = [1.5, -3.4, 6.2, 3.7, -8.4]
calc_variance(x)
#output
>> 27.293600000000005
```

3. **Giả xử 1 tập của các shape 2D (2 chiều) gồm có Rectangle, Square, Triangle, và Circle. Một Circle gồm có hai cặp tọa độ x-y (M là điểm nằm trên đường tròn và center là điểm tâm của hình tròn). Một Square gồm có hai cặp tọa độ x-y. Một Rectangle hoặc Triangle sẽ có ba cặp tọa độ x-y (Xem 2 hình bên dưới). Một cặp tọa độ x-y được gọi là một Point**



Khoảng cách giữa 2 điểm $P1 = (x_1, y_1), P2 = (x_2, y_2)$ được tính theo công thức sau

$$distance(P1, P2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$



(a) Định nghĩa TwoDShape class có method distance(P1, P2) để tính distance giữa 2 Point, có calcPerimeter() method để tính chu vi của shape và show() method để print chu vi của shape.

(b) Viết bốn sub-class Rectangle, Square, Triangle, và Circle kế thừa từ TwoDShape class. Mỗi sub-class này đều có calcPerimeter() method để tính chu vi của các shape 2D này.

(c) Viết một function nhận một list các shape và print các shape với chu vi được sắp xếp theo thứ tự tăng dần. Tạo một list có 4 shape (2D) bao gồm một circle, một rectangle, một square, và một triangle, sau đó test thử nghiệm với function đã tạo ra ở trên.

```
# Example
#Circle
print("---Circle---")
M = Point(x=6, y=3)
center = Point(x=3, y=3)
circle1 = Circle(M, center)
circle1.show()

#output
>> ---Circle---
Perimeter: 18.84

#Square
print("---Square---")
A = Point(x=1, y=1)
B = Point(x=5, y=1)
square1 = Square(A, B)
square1.show()

#output
>> ---Square---
Perimeter: 16.0

#Rectangle
print("---Rectangle---")
A = Point(x=1, y=1)
B = Point(x=5, y=1)
C = Point(x=5, y=4)
rectangle1 = Rectangle(A, B, C)
rectangle1.show()

```

```
#output
>> ---Rectangle---
Perimeter: 14.0

#Triangle
print("---Triangle---")
A = Point(x=0, y=0)
B = Point(x=2, y=4)
C = Point(x=4, y=0)
triangle1 = Triangle(A, B, C)
triangle1.show()

#output
>> ---Triangle---
Perimeter: 12.94427190999916

print("---shape list---")
shapeList = [circle1, square1, rectangle1, triangle1]

#output
>> ---sorted shape list---
Perimeter: 12.94427190999916
Perimeter: 14.0
Perimeter: 16.0
Perimeter: 18.84
showShape(shapeList)
```

4. **Cho một function có N tham số,** $f(x_0, x_1, x_2 m..., x_{N-1}) = x_0^2 + x_1^2 x_2^2 + ... + x_{N-1}^2$**. Bài toán yêu cầu tìm nghiệm (**x_0 **đến** x_{N-1}**) để** $f(x_0, x_1, x_2, ..., x_{N-1}) = x_0^2 + x_1^2 x_2^2 + ... + x_{N-1}^2 = 0$

   »Nếu các bạn không quan tâm Genetic Algorithm (GA) thì có thể bỏ qua phần này và sang trang tiếp theo để thực hiện theo đề bài yêu cầu

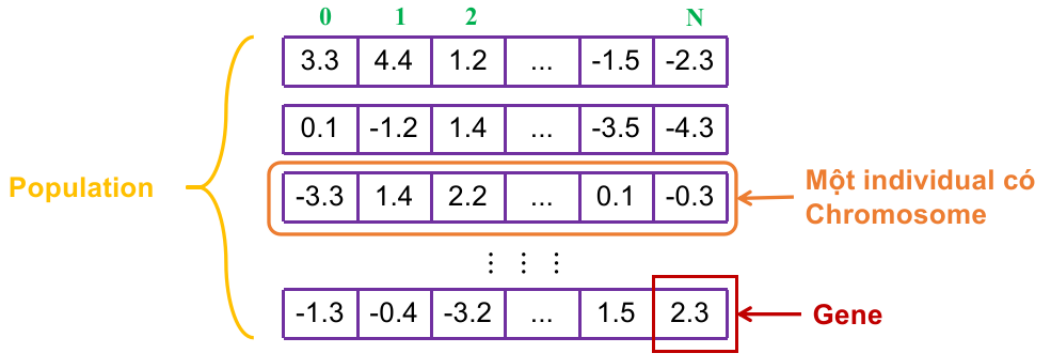
   Chúng ta biết trước rằng $f(x_0, x_1, x_2, ..., x_{N-1})$ chỉ bằng 0 khi toàn bộ nghiệm sẽ bằng 0. Nhưng để lập trình cho máy tính thì đây là một vấn đề phức tạp. Tuy nhiên, tồn tài một giải thuật có tên là giải thuật di truyền Genetic Algorithm (GA) giúp chúng ta tìm được nghiệm xấp xỉ để $f(x_0, x_1, x_2, ..., x_{N-1})$ có thể tiến càng gần về 0.

   GA sẽ khởi tạo một **population** là một list các **individual**, mỗi **individual** có một **chromosome** là một list các **gene**. (như hình 2). GA sẽ tạo random các **individual** trong một **population** sau đó dùng **fitness function** trong bài toán này sẽ gọi là **cost function**) tính toán **fitness** (trong bài toán này sẽ gọi là **cost**) cho mỗi **individual**. **Cost** dùng để chọn lọc **indiviudal** phù hợp và thực hiện quá trình lai tạo + đột biến để tạo ra **individual mới**. Quá trình này được thực hiện theo một vòng lặp liên tục để tạo ra các individual phù hợp nhất với yêu cầu đề bài.

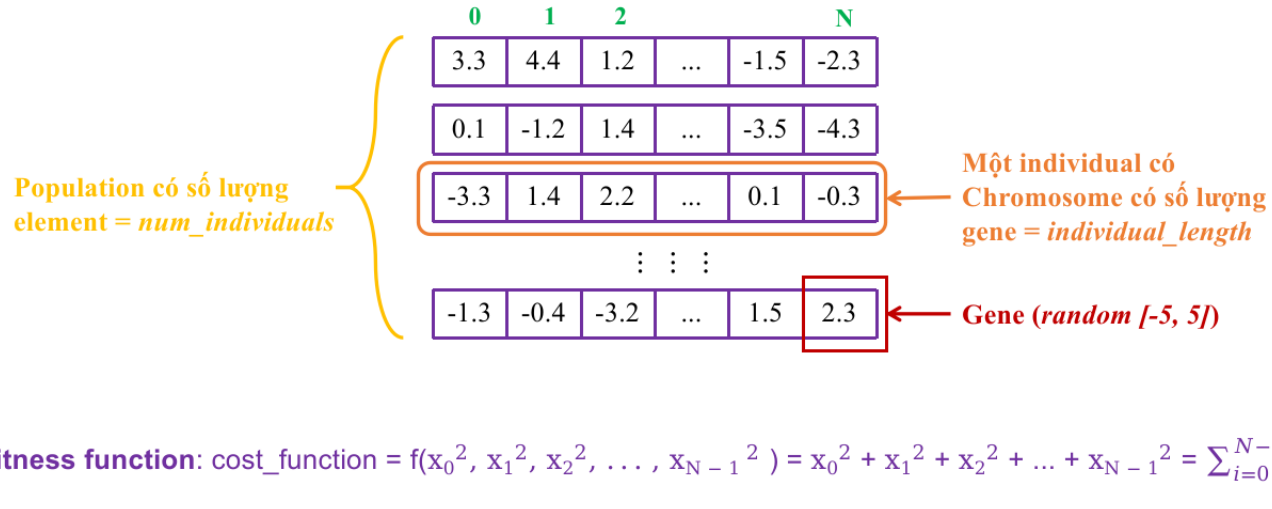
   Quay lại bài toán

   - Fitness function: cost_function $= f(x_0, x_1, ..., x_{N-1}) = x_1^2 + x_2^2 + ... + x_{N-1}^2 = \sum_{i=0}^{N-1} x_i^2$
   - Gene là x_i
   - Individual là solution của bài toán, chính là chromosome (một list chứa các gene) có length = N
   - Population: là list các indivdual (list các solution của bài toán)
   - GA sẽ tính cost của từng indivudal và chọn ra individual tốt (các cost càng gần 0 càng tốt vì là ta muốn tìm nghiệm để phương trình bằng 0). Sau đó lai tạo và đột biết bằng các thay đổi giá trị các gene để cuối cùng tìm được nghiệm là toàn bộ $x_i = 0$ để phương trình bằng 0.

- **=> Nhưng ở đây sẽ chỉ yêu cầu thực hiện một phần bài toán tính cost cho từng individual và tìm ra indivudal có cost nhỏ nhất (tối ưu nhất) trong một lần thực hiện.**



Hình 2: GA theo bài toán thông thường



Hình 3: GA theo bài toán hiện tại

(a) **Viết hai function**

- **create_individual(individual_length)**. Nhận input là một số nguyên dương và trả về một list có length = số nguyên dương này và mỗi element trong range [-5, 5]
  - **Input**: individual_length là một số nguyên dương
  - **Output**: list các element trong range [-5, 5], và số lượng element là individual_length
  - **Example**:
    - *create_individual(individual_length=10)*,
    - output là [0, 4, 5, 0, 4, -2, -3, -4, 3, -2]
- **cost_function(individual)** nhận kết quả từ function *create_individual(individual_length)*. Trả về cost là một con số theo công thức sau $\sum_{i=0}^{N-1} x_i^2$

  - **Input**: individual là output từ function *create_individual(individual_length)*
  - **Output**: cost (tổng bình phương của các element trong list input)
  - **Example**:
    * individual = *create_individual(individual_length=10)*, là một list [0, 4, 5, 0, 4, -2, -3, -4, 3, -2] được đưa vào **cost_function(individual)**.
    * Kết quả trả về 99.

(b) **Tương tự câu (a) nhưng lần này là một list các individual. Viết 2 function:**

- **create_population(num_individuals, individual_length)** nhận hai input **num_individuals** số lượng list individual trong một population, **individual_length** số lượng element trong mỗi list individual. Bài toán sẽ trả về một list của list các số trong range [-5 5]
  - **Input**: num_individuals là số nguyên dương định nghĩa số lượng list individual, individual_length là số nguyên dương định nghĩa số lượng element trong mỗi list individual
  - **Output**: list của các list individual
  - **Example**:
    * *create_population(num_individuals=3, individual_length=10)*
    * output trả về list có 3 list con, mỗi list con có 10 element trong range [-5, 5]
      [[-1, 5, -5, -1, -5, -4, -5, 2, -1, 5],
      [-1, -4, 5, -5, 2, 0, -3, 4, 1, -3],
      [-4, 2, -2, 1, 3, 1, 3, -1, 4, 0]]
- **compute_costs(population)** nhận list kết quả từ *create_population(num_individuals, individual_length)*. Trả về cost cho từng individual tương ứng.
  - **Input**: Nhận kết quả từ *create_population(num_individuals, individual_length)* list của list individual
  - **Output**: list kết quả của từng individual. Mỗi list individual sẽ tính theo function *cost_function(individual)* ở câu (a)
  - **Example**:
    * population = *create_population(num_individuals=3, individual_length=10)*
      [[-1, 5, -5, -1, -5, -4, -5, 2, -1, 5],
      [-1, -4, 5, -5, 2, 0, -3, 4, 1, -3]
      [-4, 2, -2, 1, 3, 1, 3, -1, 4, 0]]
    * **compute_costs(population)** trả về [148, 106, 61]

(c) **Viết function describe(population, costs)**. Nhận intput là population (output từ *create_population(num_individuals, individual_length)*) của câu (b) và costs (output từ *compute_costs(population)* của câu (b)). Print ra nội dung của từng indivudal và cost tương ứng

- **Input**: Nhận intput là population (output từ *create_population(num_individuals, individual_length)*) của câu (b) và costs (output từ *compute_costs(population)* của câu (b)). Input là hai list một list gồm các list indiviudal và một list cost với individual tương ứng
- **Output**: Print nội dung của từng indiviudal đi kiềm với cost tương ứng
- **Example**: như ví dụ cost list bên dưới (c)

(d) **Viết function get_best(population, costs)**. Nhận intput là population (output từ *create_population(num_individuals, individual_length)*) của câu (b) và costs (output từ *compute_costs(population)* của câu (b)). Trả về hai output một cái là individual tốt nhất có cost thấp nhất và output thứ hai là cost của individual thấp nhất đó

- **Input**: Nhận intput là population (output từ *create_population(num_individuals, individual_length)*) của câu (b) và costs (output từ *compute_costs(population)* của câu (b)). Input là hai list một list gồm các list indiviudal và một list cost với individual tương ứng
- **Output**: best_individual nội dung list individual có cost thấp nhất, best_cost cost bé nhất cũng chính là cost của indiviudal tương ứng được trả về
- **Example**: như ví dụ cost list bên dưới (d)

```
# Example
## (a)
individual_length = 10
individual = create_individual(individual_length)
print(individual)
print(cost_function(individual))
>> [0, 4, 5, 0, 4, -2, -3, -4, 3, -2]
>> 99

## (b)
num_individuals = 8
population = create_population(num_individuals, individual_length)
costs = compute_costs(population)
print(population)
print(costs)
>> [[-1, 5, -5, -1, -5, -4, -5, 2, -1, 5], [-1, -4, 5, -5, 2, 0, -3, 4, 1, -3],
    [-4, 2, -2, 1, 3, 1, 3, -1, 4, 0], [5, 5, 3, 5, -4, -1, -3, 1, 4, 2], [-5, -4,
     0, 2, -4, 1, 1, -3, 4, -1], [-4, 1, 2, 5, 4, -5, 1, 0, 1, 3], [4, -5, 3, -3,
    3, 5, 3, -4, 4, -5], [-2, -1, 5, 2, -4, -5, 2, 1, -2, 0]]

>> [148, 106, 61, 131, 89, 98, 159, 84]

## (c)
describe(population, costs)
>> Individual:[-1, 5, -5, -1, -5, -4, -5, 2, -1, 5] - Cost:148
Individual:[-1, -4, 5, -5, 2, 0, -3, 4, 1, -3] - Cost:106
Individual:[-4, 2, -2, 1, 3, 1, 3, -1, 4, 0] - Cost:61
Individual:[5, 5, 3, 5, -4, -1, -3, 1, 4, 2] - Cost:131
Individual:[-5, -4, 0, 2, -4, 1, 1, -3, 4, -1] - Cost:89
Individual:[-4, 1, 2, 5, 4, -5, 1, 0, 1, 3] - Cost:98
Individual:[4, -5, 3, -3, 3, 5, 3, -4, 4, -5] - Cost:159
Individual:[-2, -1, 5, 2, -4, -5, 2, 1, -2, 0] - Cost:84

## (d)
best_individual, best_cost = get_best(population, costs)
print(best_individual)
print(best_cost)
>> 61
>> [-4, 2, -2, 1, 3, 1, 3, -1, 4, 0]
```