Métodos de Modelado y Optimización
Laboratorio 4

Descripción:

Los algoritmos de generación de números pseudoaleatorios se encuentran intrínsecamente relacionados con el campo de la simulación computacional. Los métodos de Monte Carlo, que se basan en el uso de simulaciones aleatorias para obtener el valor esperado a un problema, dependen por su naturaleza de números aleatorios. En este laboratorio implementaremos un generador de números aleatorios que luego se utilizará para estimar, por medio de simulación, la probabilidad de victoria de una mano inicial de Texas Hold'em.

Texas Hold'em es un juego de cartas de naipe (naipe de 52 cartas), en el cual a cada jugador se le entregan 2 cartas privadas (se desconocen las cartas recibidas por los demás jugadores) y luego se agregan 5 cartas públicas compartidas entre todos los jugadores. El juego consiste en que cada jugador elige una combinación de 5 cartas entre las 7 cartas a su disposición (2 en mano, 5 en mesa compartidas) y deben formar la mejor mano posible, superando las de sus contrincantes.

Al comparar dos manos de Texas Hold'em lo importante es considerar a qué categoría pertenecen, si las manos pertenecen a diferentes categorías, el jugador con la categoría superior gana. De pertenecer a la misma categoría se comparan acorde a las cartas que conforman la categoría, ganando el que posea la/s carta/s superior/es. De volver a ocurrir un empate se compara acorde a las cartas sobrantes de la categoría. Si ocurre un empate en los tres casos, entonces es un empate para los jugadores. Las categorías en las que se clasifican las manos se pueden observar en este link siendo la escalera real la mano más "alta" (de mayor valor) y carta alta la más "baja".

Implemente las siguientes estructuras solicitadas en lenguaje Python.

1. Clase `CongruentialGenerator` que consiste en un generador de números pseudoaleatorios congruencial lineal mixto y cuenta con los siguientes métodos:
    a. (5%) Método `__init__(a, b, m)` donde `a`, `b` y `m` corresponden a los valores esenciales para la operación del generador. De manera automática debe inicializar el valor `x` con el tiempo en milisegundos actual del sistema.
    b. (5%) Método `seed(s)` que permita establecer el estado del sistema a la semilla s.
    c. (10%) Método `random()` que retorna un número aleatorio generado con un valor en el intervalo [0,1[.
    d. (15%) Método `period()` que retorna el periodo del generador con base en sus parámetros especificados.
        i. Caso de prueba 1: (a=3, b=2, m=34) tendría periódo 16
        ii. Caso de prueba 2: (a=52, b=19, m=17) tendría periódo 17

2. (25%) Método `good_abm(n)` que recibe un número entero positivo `n` y encuentra una terna `(a,b,m)` "buena" para un generador congruencial lineal mixto donde los 3 valores sean mayores a `n`.
    a. Considere que en una buena terna maximiza el periódo del generador y ayuda a volverlo impredecible. Una terna `(1,1,n)` tiene periodo completo, sin embargo los números son absolutamente predecibles, lo cuál lo hace una pésima terna.
    b. Para volver el generador impredecible nos sirve que los valores `a` y `b` sean mayores que `m`, esto evitará que haya una "secuencialidad" en los valores. Además deberemos garantizar que se cumplan los requisitos de máximo periódo.
    c. El primer requisito: "*`m` y `b` son primos entre sí*" es fácil de cumplir si se garantiza que tanto como `m` y `b` sean primos (y no sean el mismo número). Por lo que solo se requiere encontrar dos números primos mayores a `n` que sean diferentes.
    d. El segundo requisito: "*Si `q` es un número primo que divide a `m`, entonces `q` divide a `a-1`*" es fácil de cumplir si `m` es primo, eso significa que `a` deberá ser un múltiplo de `m`, es decir `a=(p*m)+1` donde `p` es un número arbitrario.
    e. El tercer requisito: "*Si 4 divide a `m`, entonces 4 divide a `a-1`*", no nos afecta, dado que `m` es primo y por lo tanto no aplica.

3. (15%) Método `compare_hands(player, opponent)` que recibe dos arreglos compuestos por 5 números enteros (u objetos Carta, si se desea implementar una clase de soporte) y retorna si la mano del jugador player es superior a la del oponente. Para ello debe tomar en consideración las reglas de comparación de manos del Texas Hold' Em.

4. (20%) Método `simulate(initial_cards, rolls, generator)` que recibe un arreglo de 2 números enteros (u objetos Carta, si se desea implementar una clase de soporte) que representan las cartas en mano de un jugador, un número entero `rolls` que representa la cantidad de simulaciones a realizar y un generador de números aleatorios a utilizar para la aleatoriedad. Por cada simulación deberá generar al azar las posibles cartas en mesa y en la mano del oponente. Luego deberá calcular si el jugador gana, empata o pierde dicha simulación y, al terminar las simulaciones, deberá retornar una tupla con el porcentaje de simulaciones ganadas, empatadas y pérdidas.
    a. Note que el método `compare_hands` compara dos manos compuestas de 5 cartas, sin embargo los jugadores tienen 7 cartas a su disposición. Para resolver esta situación, es necesario comparar la **mejor** mano del jugador con la **mejor** mano del oponente.
    b. Para obtener la **mejor** mano de un jugador, es necesario comparar cuál de las 21 combinaciones (7C5: "*de 7 elija 5*") es la mejor, para ello es posible utilizar el método `compare_hands`.

5. (5%) Cree un código main que haga uso de sus métodos y estructuras generadas.

a. Debe inicializar un objeto de su clase `CongruentialGenerator` con una combinación arbitraria de parámetros generados por el método `good_abm` con un `n` superior a 10 millones, la decisión del `n` exacto queda a su decisión.
b. Imprima el m de su generador así como el periódo. ¿Son iguales?
c. Luego haga tres llamados a su método simulate, envíe su generador y dígale a su algoritmo que ejecute 100 000 simulaciones en cada caso.
    i. ¿Cuál es la probabilidad de ganar con un par de AA's en mano?
    ii. ¿Cuál es la probabilidad de ganar con un par de 22's en mano?
    iii. ¿Cuál es la probabilidad de ganar con un 2-7 de diferente palo en mano?
d. Documente las respuestas a las preguntas.

6. (+10% extra) Modifique su método simulate para incluir un parámetro adicional: `simulate(initial_cards, rolls, generator, opponents=1)` , este parámetro adicional se utilizará para indicar la cantidad de contrincantes en mesa. Para cada oponente deberá generarse una mano diferente y deberá tomarse en cuenta para determinar las probabilidades de victoria, empate y derrota.
    a. Considere que para que una mano cuente como una victoria, la mano del jugador debe "vencer" a las de todos los demás oponentes.
    b. Considere que con solo perder contra una mano oponente el resultado se considera una derrota (dado que en solo hay un ganador por mano, y sería el oponente con la mano más fuerte).
    c. Considere que aunque gane contra todos los demás oponentes, con solo que haya un empate contra una mano contrincante el resultado se considera un empate.

Nota: Se permite utilizar la biblioteca de soporte time, math, numpy, random, pero no se permite utilizar bibliotecas que realicen el trabajo por usted. Cualquier otra biblioteca que se desee usar debe ser consultada primero con el profesor.