1. Statystyka

1 Średnia i wariancja próby losowej

Szablon programu należy uzupełnić o definicję funkcji `mean_variance(...)`, która wypełnia tablicę liczbami całkowitymi z przedziału $[a, b]$ a następnie oblicza średnią arytmetyczną oraz wariancję zapisanych w tablicy liczb.

1.1 Test

- **Wejście**
  Numer testu, zarodek generatora `seed`, liczba prób n, dolna i górna granica przedziału generacji a i b
- **Wyjście**
  Wartości średniej arytmetycznej i wariancji zapisane z dwoma cyframi znaczącymi po kropce dziesiętnej.
- **Przykład:**
  Wejście: 1 2001 10 1 10
  Wyjście: 5.50 7.65

2 Tablica wyników prób Bernoulliego

Dla przypomnienia:

Próba Bernoulliego to eksperyment losowy z dwoma możliwymi wynikami, np. rzut monetą z wynikami 0 (reszka, porażka), 1 (orzeł, sukces). Przyjmujemy, że moneta nie jest symetryczna, tzn. zadajemy, jakie jest prawdopodobieństwo p rezultatu "orzeł" – wyniku 1 (dla monety symetrycznej byłoby równe 0.5). Symulację takiego eksperymentu należy zrealizować stosując biblioteczny generator liczb pseudolosowych.

Dla powtarzalności wyników programu należy przyjąć, że wynik próby jest równy 1 gdy wylosowana z przedziału $[0, RAND\_MAX]$ liczba jest mniejsza od $p \cdot (RAND\_MAX + 1)$.

Szablon programu należy uzupełnić o definicję funkcji `bernoulli_gen(...)`, która generuje losowo tablicę n prób Bernoulliego. Elementami tej tablicy mają być wyniki prób.

2.1 Test

Test symuluje wykonanie rzutów monetą. Wczytuje liczbę rzutów i założone prawdopodobieństwo wypadnięcia orła (wyniku 1) oraz wyprowadza wyniki kolejnych prób.

- **Wejście**
  Numer testu, zarodek generatora `seed`, liczba prób n, prawdopodobieństwo p wypadnięcia orła (jedynki).
- **Wyjście**
  Wyniki symulacji n prób.
- **Przykład:**
  Wejście: 2 2001 20 0.3
  Wyjście: 1 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0

3 Dyskretny rozkład prawdopodobieństwa

Dyskretny rozkład prawdopodobieństwa jest to rozkład prawdopodobieństwa zmiennej losowej, której zbiór możliwych wartości jest przeliczalny. Zdarzeniem losowym w tym zadaniu jest rzut dwoma sześciennymi kostkami do gry w kości. Wartością zmiennej losowej jest suma oczek, które wypadły na tych dwóch kostkach, czyli liczba z przedziału $[2, 12]$. Rezultatem tego zadania ma być przybliżony rozkład prawdopodobieństwa tej zmiennej losowej. Przybliżony, bo

1. zamiast rzutu kostkami stosujemy generator liczb pseudolosowych,
2. wykonujemy tylko skończoną liczbę prób.

Szablon programu należy uzupełnić o definicję funkcji `pmf(...)` (probability mass function), która symuluje wykonanie n rzutów dwoma kostkami i zapisuje wartości otrzymanego przybliżonego rozkładu prawdopodobieństwa w tablicy. Losując liczbę oczek jednej kostki – dla uzyskania powtarzalności wyników – należy raz wywołać funkcję `rand()` i jej wynik sprowadzić do przedziału $[1, 6]$.

Szablon programu należy również uzupełnić o definicję funkcji `print_histogram(double v[], int n, int x_start, double y_scale, char mark)`, która w trybie znakowym przedstawia histogram funkcji o n wartościach zapisanych w argumencie v. Należy przyjąć założenia:

1. Oś zmiennej niezależnej jest pionowa, skierowana w dół. Oś zmiennej zależnej jest pozioma, skierowana w prawo (nie jest rysowana).
2. Wartości zmiennej niezależnej są kolejnymi liczbami naturalnymi, począwszy od `x_start`. Są one pisane od pierwszej lewej kolumny, w polu o szerokości 2 znaków z wyrównaniem w prawo.
3. W trzeciej kolumnie wyprowadzane są znaki | tworząc oś x.
4. Począwszy od 4. kolumny pisane są znaki `mark`. Liczba znaków jest przeskalowaną i zaokrągloną wartością funkcji. Parametr `y_scale` jest wartością zmiennej zależnej odpowiadającej szerokości jednego znaku na wykresie.

5. Wartości funkcji (liczby nieujemne typu `double`) są wyprowadzane z dokładnością do 3 cyfr po kropce dziesiętnej w każdym wierszu, po jednej spacji na prawo od ostatniego znaku `mark`.

3.1 Test

Test wczytuje dane, wywołuje funkcję `pmf(...)` i rysuje histogram obliczonego rozkładu.

- **Wejście**
  Numer testu, zarodek generatora `seed`, liczba prób n, znak do rysowania wykresu
- **Wyjście**
  Wartości rozkładu prawdopodobieństwa w postaci histogramu
- **Przykład:**
  Wejście: 3 2001 1000 *
  Wyjście:

```
 2 |***** 0.024
 3 |************** 0.068
 4 |******************* 0.094
 5 |************************ 0.120
 6 |**************************** 0.140
 7 |********************************* 0.165
 8 |************************* 0.125
 9 |******************* 0.096
10 |***************** 0.083
11 |********** 0.051
12 |******* 0.034
```

4 Dystrybuanta (ang. Cumulative Distribution Function)

Szablon programu należy uzupełnić o definicję funkcji `cdf(...)`, która na podstawie danego dyskretnego rozkładu prawdopodobieństwa (funkcja `mdf()`) oblicza wartości dystrybuanty w punktach skokowych (w punktach nieciągłości dystrybuanty, przy założeniu jej prawostronnej ciągłości[1]). Obliczone wartości zapisuje na miejscu danych wejściowych.

4.1 Test

Test wyznacza wartości dystrybuanty wywołując funkcję `cdf(...)`.

- **Wejście**
  Numer testu, zarodek generatora `seed`, liczba prób (rzutów) n, znak do rysowania wykresu
- **Wyjście**
  Wartości obliczonej dystrybuanty w kolejnych punktach skokowych przedstawione w postaci histogramu (przy pomocy funkcji `print_histogram(...)`) jak w teście 3.1).

[1]Dla przypomnienia definicji i własności dystrybuanty: `https://en.wikipedia.org/wiki/Cumulative_distribution_function`, w szczególności wykresy w części Properties ilustrujące punkty skokowe i prawostronną ciągłość dystrybuanty dyskretnej.

- **Przykład:**
  Wejście: 4 2001 1000 #
  Wyjście:

```
 2 |# 0.024
 3 |##### 0.092
 4 |######### 0.186
 5 |############### 0.306
 6 |###################### 0.446
 7 |############################### 0.611
 8 |##################################### 0.736
 9 |########################################## 0.832
10 |############################################## 0.915
11 |################################################ 0.966
12 |################################################## 1.000
```

5 Monty Hall problem, czyli jak wybierać “drzwi”, aby zwiększyć prawdopodobieństwo wygranej

Paradoks Monty’ego Halla, w przypadku trojga drzwi (bramek) do wyboru, polega na tym, że intuicyjnie przypisujemy równe szanse dwóm sytuacjom — wskazanie wygranej w jednej z dwóch zakrytych ciągle bramek wydaje się równie prawdopodobne jak wskazanie bramki pustej, bo przecież “nic nie wiadomo”. Tymczasem układ jest warunkowany przez początkowy wybór zawodnika i obie sytuacje nie pojawiają się równie często.

Opis problemu: `https://pl.wikipedia.org/wiki/Paradoks_Monty’ego_Halla`.

Szablon programu należy uzupełnić o definicję funkcji `monty_hall(...)`, która symuluje n rozgrywek. Założenia:

1. W każdej rozgrywce funkcja wywołuje `rand()` dokładnie 2 razy.
2. W pierwszym losowaniu jest wybierany numer drzwi, za którymi jest nagroda.
3. W drugim losowaniu – numer drzwi, które gracz wybiera na początku gry.

Funkcja oblicza (i przekazuje do funkcji ją wywołującej) ile razy w ciągu n rozgrywek wygrywał gracz, który po otwarciu jednych drzwi zmieniał pierwotną decyzję.

5.1 Test

Test wczytuje n - liczbę prób (rozgrywek) i wywołuje funkcję `monty_hall(...)` przekazując jej wartość n oraz adresy zmiennych, do których funkcja ma wpisać wyniki. Test wypisuje wyniki symulacji.

- **Wejście**
  Numer testu, zarodek generatora `seed`, liczba prób (rozgrywek) n
- **Wyjście**
  Liczba wygranych po decyzji „zmień wybór” i po decyzji „nie zmieniaj” (skąd wziąć tę wartość?)
- **Przykład**
  Wejście: 5 2001 1000
  Wyjście: 659 341