Projekt 1 - Monte Carlo

Marta Gacek

November 2024

1 Wprowadzenie

Celem niniejszego projektu jest wykorzystanie generatorów liczb pseudolosowych do przeprowadzenia testów statystycznych w symulacjach. Chcemy sprawdzić czy dany generator losuje liczby z rozkładu jednostajnego.

1.1 Testy statystyczne

Zanim przejdziemy do części programistycznej, opiszemy stosowane przez nas testy od strony teoretycznej, aby wiadomo było jak funkcjonują.

1.1.1 Test χ^2

Test *chi kwadrat* jest dość popularnym testem statystycznym. Jego działanie opiera się na wygenerowaniu pewnej liczby obserwacji, powiedzmy n, z których każda dopasowana jest do jednej z k kategorii (obrazowo możemy opisać tę sytuację jako wrzucanie n kul do k kubełków).

Zakładając, że Y_i oznacza liczbę obserwacji z i-tej kategorii, natomiast p_i jest prawdopodobieństwem dopasowania obserwacji do i-tej kategorii, możemy się spodziewać, że dla dużych wartości n zajdzie zależność

$$Y_i \approx np_i.$$

Ze względu na założenie o niezależności obserwacji, statystyka

$$\hat{\chi}^2 = \sum_{j=1}^{k} \frac{(Y_j - np_j)^2}{np_j}$$

ma rozkład χ^2 z $k-1$ stopniami swobody, co oznaczamy jako $\chi^2(k-1)$.

1.1.2 Test Kołmogorowa-Smirnowa

Kolejnym testem, którego będziemy używać, jest test Kołmogorowa-Smirnowa. Zakładamy w nim, że mamy n obserwacji X_i, $i = 1, ..., n$, gdzie zmienna

losowa X ma ciągły rozkład z dystrybuantą $F_X(x)$. Statystyka Kołmogorowa-Smirnowa ma postać

$$D_n = \sup_x |F_n(x) - F_X(x)|,$$

gdzie $F_n(x) = \frac{1}{n}\sum_{i=1}^{n} 1(X_i \leq x)$ oznacza dystrybuantę empiryczną. Sprawdzamy zatem czy rozkład naszej próby różni się w istotny sposób od rozkładu teoretycznego.

1.1.3 Test serii

Celem tego testu jest sprawdzenie czy dane są rozmieszczone w losowy sposób. Przyjmujemy, że seria to ciąg kolejnych wartości tego samego typu w danych. Dzielimy dane na dwie kategorie, na przykład wartości poniżej i powyżej średniej. Następnie liczymy serie w ciągu i porównujemy ich liczbę z wartością oczekiwaną.

Dla dużych prób liczba serii (K) jest zbliżona do rozkładu normalnego ze średnią oraz wariancją odpowiednio

$$\mu = \frac{2xy}{x+y} + 1, \quad \sigma^2 = \frac{2xy(2xy-(x+y))}{(x+y)^2(x+y-1)}.$$

Oznaczenia x i y to liczby wartości w odpowiednio pierwszej i drugiej kategorii. Statystyka testowa ma zatem postać

$$Z = \frac{K-\mu}{\sigma}.$$

Pozostaje porównać ją z odczytaną z tablic wartością Z_α dla wybranego poziomu istotności. Jeśli $|Z| > Z_\alpha$, to odrzucamy hipotezę zerową, czyli ciąg nie jest losowy.

1.1.4 Frequency monobit test

Chcemy sprawdzić losowość ciągu bitów, który oznaczamy jako $b_1, , , b_i \in \{0, 1\}$. Przekształcamy bity na $x_1, ..., x_i$, przyjmujące wartości $\{-1, 1\}$, według wzoru $x_k = 2b_k - 1$. Statystyka testowa ma postać

$$S_n = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} x_i.$$

Jeśli $b_k,\ k = 1, 2, ...$ są niezależne i rozłożone jednostajnie na $\{0, 1\}$, to $x_k,\ k = 1, 2, ...$ są niezależne i rozłożone jednostajnie na $\{-1, 1\}$. Z tego względu statystyka S_n ma w przybliżeniu rozkład $N(0, 1)$. Wobec tego

$$p - value = P(|N| > |S_n|) = 2(1 - \phi(|S_n|),$$

gdzie N jest zmienną losową standardowego rozkładu normalnego, a ϕ jego dystrybuantą. Możemy porównać wartość statystyki S_n z tablicami statystycznymi dla rozkładu normalnego, odczytując Z_α dla konkretnego poziomu istotności. Jeśli $|S_n| > Z_\alpha$, to ciąg nie jest losowy.

1.2 Generatory liczb pseudolosowych

Krótko opiszemy generatory, których będziemy używać, aby lepiej zrozumieć jak działają.

1.2.1 LCG (Linear Congruential Generator)

LCG jest popularnym, prostym w implementacji generatorem. W danych wejściowych wymagany jest początkowy warunek x_0 (pojedyncza wartość). Działanie algorytmu opiera się na wyliczaniu kolejnych liczb w sposób rekurencyjny, ze wzoru

$$x_i = (ax_{i-1} + c) \ modulo \ M,$$

gdzie a oraz c są pewnymi parametrami, a M określoną wartością modulo.

1.2.2 GLCG (Generalized Linear Congruential Generator)

Ten generator jest podobny w budowie do poprzednika, ale uogólniony. Tym razem w danych wejściowych mamy k początkowych wartości, a kolejne liczby wyznaczane są rekurencyjnie przy użyciu k poprzedników:

$$x_i = (a_1x_{i-1} + ... + a_kx_{i-k}) \ modulo \ M.$$

Oczywiście $a_1, ..., a_k$ są pewnymi stałymi.

1.2.3 RC4

Generator RC4 ma bardziej złożoną budowę. Na wejściu bierze on "klucz", czyli liczby ze zbioru wybranej wielkości. Najpierw tworzona jest permutacja identycznościowa, która następnie zostaje przekształcona za pomocą klucza na inną permutację. Odbywa się to w obrębie funkcji KSA. Kolejnym etapem jest użycie funkcji $PRGA$, która bierze wspomnianą permutację i odpowiednio ją aktualizuje, zwracając komplet liczb pseudolosowych.

1.2.4 Xorshift

Ostatnim z generatorów jest Xorshift, który opiera się na przekształcaniu liczby za pomocą kilku czynności. Zaczynamy od pojedynczej niezerowej wartości początkowej. Następnie aktualizujemy ją za pomocą operacji XOR i kilku przesunięć bitowych w celu "wymieszania" bitów. Zaktualizowana wartość w każdej takiej iteracji jest liczbą pseudolosową.

2 Symulacje

Gdy już znamy teoretyczną stronę naszego projektu, możemy przejść do symulacji.