Adam Wieliczko

Projekt numer 3: “Generator liczb losowych”

1. Opis projektu:

Projekt zgodnie z nazwą opiera się na implementacji generatorów pseudolosowych o rozkładach: -jednostajnym -jednostajnym na przedziale (0,1) -Bernoulliego (dwupunktowego) -dwumianowego -Poissona -wykładniczego -normalnego Po ich utworzeniu sprawdzam czy są losowe oraz czy generują zmienne w podanym rozkładzie

2. Opis generatorów:

Każdy generator przyjmuje w argumentach seed, fact, con oraz div. Użytkownik chcąc wygenerować liczbę danym generatorem może wpisać wyłącznie seed, ale jeśli chce może również wpisać wartości wpływające na pracę generatora G. Jeśli jednak chce to zrobić warto, żeby m.in. liczby wpisywane nie miały wspólnych dzielników, w przeciwnym wypadku (dla przykładowo fact = 50, con = 0, div = 25) możemy otrzymać wyłącznie 0, bez względu na seed.

Generator G:

Jest to generator addytywny tworzący zmienne o rozkładzie jednostajnym o wartościach od 0 do 2147483647 (czyli inaczej mówiąc do $2^{31} - 1$). Możemy skrócić zakres wartości możliwych do wygenerowania wpisując argument div.

Działanie generatora:

Podany seed jest najpierw mnożony przez wartość podaną przez użytkownika jako fact lub jeśli nie zostanie podana, a następnie dodawana jest do wyniku wartość stała (można ustawić wpisując wartość con) po wykonujemy operację modulo przez wartość div o ile została podana zwiększając lub zmniejszając tym samym przedział liczb, bądź przez podstawową wartość ustawioną na 2147483647 (maxInt). Ważne jest, jeśli chcemy wpisać wartości mnożnika oraz stałej, żeby nie miały wspólnych dzielników z wartością div/maxInt. Warto też pamiętać, że najlepszym możliwym mnożnikiem jest wartość o jeden mniejsza od wartości modulo, gdyż dla con = 0 otrzymujemy wtedy generator multiplikatywny zamiast addytywnego i podany mnożnik da nam największy możliwy zakres generatora. Ze względu na to, że generator w Pythonie nie bierze pod uwagę typów wpisywanych generator pomimo podstawowego zastosowania na liczbach całkowitych potrafi działać na liczbach zmiennoprzecinkowych oraz spełniać niejako rolę generatora J (dla div = 1 oraz mnożnika, bądź seeda z liczbą po przecinku), jednakże nie jest to zalecane, gdyż strikte do liczb w przedziale (0,1) służy generator J.

Generator J:

Jest to generator, który na podstawie generatora G wylicza zmienną o rozkładzie jednostajnym na przedziale (0,1).

Działanie generatora:

Generuję zmienną w generatorze G, dodaję do wyniku 1 a następnie dzielę ją przez liczbę użytą jako modulo + 1. W ten sposób dodając jedynki otrzymujemy przedział nie uwzględniający zera oraz jedynki.

Generator B:

Jest to generator, który na podstawie generatora J wylicza liczbę z rozkładu Bernoulliego (dokładniej chodzi o rozkład dwupunktowy, nie o dwumianowy jak jest najczęściej uznawane w Polsce). W tym generatorze dodatkowym argumentem możliwym do wpisania jest p, które oznacza prawdopodobieństwo sukcesu danej próby.

Działanie generatora:

Generuję zmienną w generatorze J, a następnie porównuję wielkość zmiennej do ustalonego p. Jeśli zmienna jest równa lub większa od p to uznaję próbę za nieudaną i zwracam 0, w przeciwnym wypadku zwracam 1.

Generator D:

Jest to generator, który na podstawie generatora J tworzy zmienną z rozkładu dwumianowego. Dodatkowym argumentem, który możemy wpisać jest p (analogiczne do generatora B) oraz n oznaczające ilość prób.

Działanie generatora:

Można nieformalnie określić działanie jako zliczenie sukcesów w n próbach z poprzedniego generatora (B). W implementacji zastosowałem fakt, że dla J oznaczającego zmienną losową z generatora J mamy twierdzenie, iż $min(\frac{J}{p}, \frac{1-J}{1-p})$ jest niezależną próbą wobec J. Dzięki temu uniknąłem generowania wielokrotnie generatora J i ograniczając się tylko do jednego wywołania.

Generator P:

Jest to generator, który na podstawie generatora J tworzy zmienną z rozkładu Poissona. W tym wypadku możemy podać lambdę.

Działanie generatora:

Generator sprawdza ile razy uzyskamy niezależne zdarzenia do momentu gdy wydarzy się zdarzenie zależne od lambdy.

Generator W:

Jest to generator, który na podstawie generatora J tworzy zmienną z rozkładu wykładniczego. Można opcjonalnie podać lambdę.

Działanie generatora:

Można opisać schemat następująco: korzystam z odwróconej dystrybuanty i podstawiam za (1 - p) zmienną losową o rozkładzie równomiernym z przedziału (0, 1) i otrzymujemy wzór $(-1)\frac{\ln J}{\lambda}$ gdzie J to owa zmienna.

Generator N:

Jest to generator, który na podstawie generatora J tworzy zmienną z rozkładu normalnego. Argumentami dodatkowymi jest drugi seed, factToN oraz conToN. Drugi seed wynika z tego, że algorytm boxa-mullera wymaga dwóch zmiennych z generatora J, factToN i conToN odpowiadają za odstandaryzowanie otrzymanego wyniku.

Działanie generatora:

Najpierw generator postępuje zgodnie z algorytmem Boxa-Mullera. Otrzymujemy dzięki temu zmienną o rozkładzie normalnym standaryzowanym. Wystarczy zatem odstandaryzować zmienną i otrzymamy oczekiwany wynik.

3. Opis testów:

Zaimplementowałem dwa typy testowania: testy serii sprawdzające losowość próby oraz testy MMVK sprawdzające czy dany generator ma właściwości odpowiadającemu rozkładowi. Przeprowadziłem przykładowe testy serii na generatorach G, J, D, W oraz N dla kilku losowych seedów. Testy MMVK z kolei zostały przeprowadzone na wszystkich generatorach z kilkoma różnymi argumentami startowymi. Testy serii kończą się sukcesem, a testy MMVK pokazują bliskość utworzonych generatorów do właściwych wartości dla ich średniej, mediany, wariancji oraz kurtozy. Dokładniejsze wyniki testów są dość długie i aby nie zaśmiecać dokumentacji zostały umieszczone w kodzie, dzięki temu można faktycznie sprawdzić poprawność generatorów również na innych danych. Są tam również umieszczone histogramy dla każdego z utworzonych generatorów.

4. Wnioski:

Sądzę, że generatory można uznać za losowe ze względu na pozytywne testy serii dla różnych wartości. Co do poprawności ich rozkładów - uważam, że testy MMVK kończące się bardzo bliskimi do poprawnych wartościami są dostatecznym potwierdzeniem.