

4.1 Wprowadzenie do modelowania

Uwaga!!! Rzut monetą nie jest eksperymentem losowym.

Znając warunki początkowe oraz wiedząc wszystko o otoczeniu, wyposażeni w znajomość zasad dynamiki jesteśmy w stanie dokładnie określić na którą stronę upadnie moneta. Problem polega na tym, że nie jesteśmy w stanie poznać dokładnie warunków początkowych (dokładność pomiaru jest ograniczona), równanie ruchu uwzględniające wszystkie czynniki było by skomplikowane, nie mamy wystarczająco dużo czasu by wyznaczyć dokładny wynik. W tej sytuacji możemy modelować przebieg działania pewnego procesu w sposób probabilistyczny. Podobna sytuacja występuje przy opracowaniu prognozie pogody, określaniu struktury przestrzennej białek, opisie ruchu pojazdów na drogach itp.

Modelowanie stochastyczne to szeroka gałąź nauki, obejmująca problematykę budowy modelu, badania właściwości modelu, weryfikacje adekwatności modelu itp. Poniżej przedstawimy trzy popularne modele, oraz zajmiemy się badaniem ich właściwości.

4.2 Rozgrzewka, czyli Centralne Twierdzenie Graniczne

Przy odpowiednich założeniach (jakich?) średnia z n zmiennych losowych zbiega do rozkładu granicznego.

Co to właściwie oznacza? Dla przykładu zobaczmy jak zachowuje się średnia ze 100 zmiennych losowych o rozkładzie wykładniczym $\lambda(1)$.

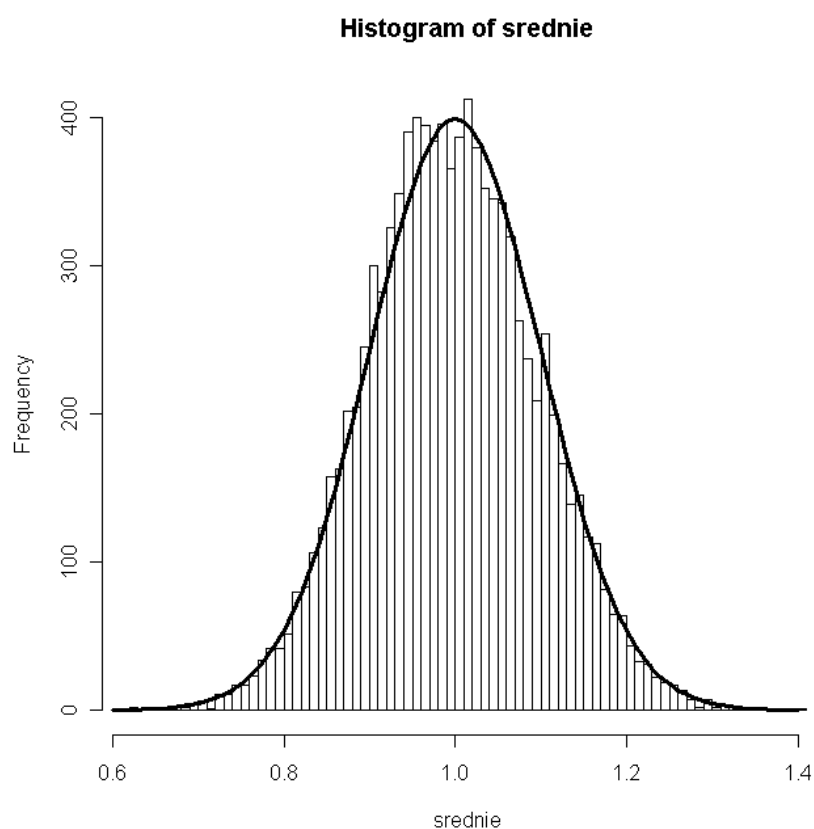
```
> n ← 10000
> m ← 100
>
> obs ← matrix(rexp(n*m,1), n,m)
> srednie ← apply(obs, FUN=mean, 1)
> hist(srednie,100)
```

Wynik tych symulacji przedstawiony jest na rysunku poniżej. Widzimy, że rozkład średnich jest całkiem podobny do rozkładu normalnego (właśnie o tym mówi CTG).

Oczywiście pojawiają się naturalne pytania:

- jakie jest tempo zbieżności?
- czy średnia z innych rozkładów też zachowa się równie dobrze (proponuje zobaczyć średnią z 12 zmiennych losowych o rozkładzie jednostajnym),
- dla jakich rozkładów to nie działa (proponuje sprawdzić rozkłady dyskretne, rozkład cauchego i rozkład log normalny),
- jakie są konsekwencje CTG (o tym będziemy dużo mówić).

Zainteresowany czytelnik znajdzie na te pytania odpowiedź modyfikując powyżej umieszczony program w języku R.



Rysunek 4.1: Przeskalowana gęstość rozkładu normalnego $\mathcal{N}(1, 0.1^2)$ i histogram obserwowanych średnich.

4.3 Model kasyna

Przypuśćmy, że mamy możliwość gry w grę o następujących zasadach

- gracz stawia kwotę x ,
- z prawdopodobieństwem p przegrywa i traci kwotę x ,
- z prawdopodobieństwem $1 - p$ wygrywa i zyskuje kwotę x .

Przypuśćmy, że pewien Bardzo Pilnie Potrzebny Element Komputera kosztuje 700\$ a Ty masz tylko 100\$ i jedynym wyjściem aby ten element kupić jest gra w wyżej wymienioną grę. Oszacuj prawdopodobieństwo wygrania 700\$ dla każdej z poniższych strategii, dla parametrów $p = 0.5$, $p = 0.45$ i $p = 0.55$.

- **Strategia cierpliwa** w każdym kroku stawiasz kwotę 100\$.

```
gotowka ← 100
p       ← 0.45
krok    ← 1
while (gotowka[krok]>0 & gotowka[krok]<700) {
  stawka ← 100
  krok   ← krok+1
  gotowka[krok] ← gotowka[krok-1] - stawka + 2*stawka*(runif(1)>p)
}
plot(gotowka, type="l", lwd=3)
```

- **Strategia narwana** w każdym kroku stawiasz kwotę „ile możesz”, czyli $\min(700\$-y, y)$, gdzie y to kwota pieniędzy, które posiadasz.

```
gotowka ← 100
p       ← 0.45
krok    ← 1
while (gotowka[krok]>0 & gotowka[krok]<700) {
  stawka ← min(gotowka[krok], 700 - gotowka[krok])
  krok   ← krok+1
  gotowka[krok] ← gotowka[krok-1] - stawka + 2*stawka*(runif(1)>p)
}
plot(gotowka, type="l", lwd=3)
```

4.3.1 Zadania

- Spróbuj rozwiązać ten problem analitycznie (Model Markowa), a następnie napisz program rozwiązujący ten problem symulacyjnie (uwaga! rozwiązanie tego problemu analitycznie pozwala na otrzymanie plusa po uprzednim przedstawieniu rozwiązania prowadzącemu).
- Wyznacz zależność pomiędzy prawdopodobieństwem wygrania w jednej grze p a prawdopodobieństwem zdobycia kwoty 700\$.

- Wyznacz symulacyjnie rozkład czasu trwania gry.
- Zbadaj jak prawdopodobieństwo wygrania zmienia się w zależności od kwoty, którą chcemy wygrać.
- Jaka jest oczekiwana wartość portfela po 5 grach?
- Która strategia jest lepsza, od czego to zależy?

4.4 Model Bomby Atomowej

Poniżej opiszemy uproszczony opis zagadnienia, nad którym pracował Stanisław Ulam w Los Alamos. Prowadzone przez Niego badania doprowadziły do powstania bomby wodorowej (to źle) oraz rozwoju metod Monte Carlo (to dobrze).

Zakładam, że każdy wie jak działa bomba atomowa (jeżeli nie, to polecam artykuł <http://science.howstuffworks.com/nuclear-bomb.htm>). W skrócie, atomy niektórych pierwiastków w wyniku zderzenia z pojedynczym neutronem ulegają rozpadowi, temu rozpadowi towarzyszy wydzielenie się energii, oraz rozpad atomu na atomy innych pierwiastków i wolne neutrony. Wolne neutrony o ile jest ich wystarczająco dużo są w stanie doprowadzić do reakcji łańcuchowej rozszczepiając jądra kolejnych pierwiastków. Za tą historyjką kryje się bardziej skomplikowana fizyka, ale na potrzeby tego modelu całą tą fizykę jeszcze bardziej uprościmy.

Przypuśćmy, że rozpędzony neutron trafi w jądro atomu Uranu z prawdopodobieństwem p . Neutron, który „nie trafi” nie jest już dla nas interesujący. Jeżeli neutron trafi, to w wyniku rozpadu zostanie uwolniona trójka nowych neutronów. Każdy neutron z tej trójki, jak również „stary” neutron z prawdopodobieństwem p trafi w kolejne jądro. Jeżeli dojdzie do bardzo dużej liczby rozszczepień (dużo to np. 10^8) w krótkim czasie to w efekcie obserwuje się wydzielenie dużej ilości energii (nazwywane zwyczajowo wybuchem).

Przypuśćmy, że eksperyment rozpoczęliśmy z użyciem n wolnych neutronów.

Poniższy fragment kody opisuje rozwój wydarzeń dla pojedynczego eksperymentu

```
l.neutronow ← 10
p           ← 0.26
krok       ← 1
while (l.neutronow[krok]>0 & l.neutronow[krok]<10^8) {
  krok      ← krok+1
  l.neutronow[krok] ← 4*rbinom(1,l.neutronow[krok-1],p)
}

plot(l.neutronow, type="l", lwd=3)
```

4.4.1 Zadania

- Wyznacz prawdopodobieństwo, że dojdzie do reakcji łańcuchowej, to znaczy, że liczba „gotowych do działania” neutronów przekroczy 10^8 , dla $n = 10$ i $p = 0.26$.
- Wykonaj wykres przedstawiający zależność prawdopodobieństwa wybuchu od parametru n .
- Wykonaj wykres przedstawiający zależność prawdopodobieństwa wybuchu od parametru p .

4.5 Model obciążenia serwera

Mamy serwer (w przeciwieństwie do poprzednich modeli, ten mam nadzieję nie wymaga wprowadzenia). Do serwera są wysyłane zadania do realizacji. Wykonanie zadania zabiera określoną ilość czasu. Interesuje nas jak poradzi sobie serwer z realizacją zleconych zadań. Model ten znany jest w wielu modyfikacjach, rozważmy jedną z prostszych.

Do serwera wysyłane są zadania w sposób losowy z jednostajną częstością. Czas pomiędzy kolejnymi „wysłaniami” opisać można rozkładem wykładniczym z częstością λ (pytanie: jakim rozkładem można opisać liczbę zadań wysłanych w ciągu jednej godziny?). Realizacja zadania i tego zabiera $x_i \sim \chi_1^2$ czasu.

Serwer działa w następujący sposób:

- jeżeli otrzymuje nowe zadanie do realizacji, to informacja o tym zadaniu dodawana jest na koniec kolejki zadań,
- jeżeli serwer nie realizuje żadnego zadania a lista zadań do wykonania nie jest pusta, to rozpoczyna natychmiast realizacja kolejnego zadania z kolejki.

Zadanie do wykonania, to ocena rozkładu czasu bezczynności serwera, w zależności od parametru λ .

Poniżej umieszczamy kod opisujący „dane wejściowe” dla serwera

```
wejscie ← function(T, lambda) {
  liczba.zadan ← rpois(1, T*lambda)
  czasy.wyslania.zadan ← sort(runif(liczba.zadan, 0, T))
  czasy.trwania.zadan ← rchisq(liczba.zadan,1)

  list(liczba.zadan = liczba.zadan,
       czasy.wyslania.zadan = czasy.wyslania.zadan,
       czasy.trwania.zadan = czasy.trwania.zadan)
}
```

Opiszmy teraz działanie serwera w okresie czasu T

```
serwer ← function(dane.in) {
  aktualny.czas ← 0
  czas.rozporzeczcia ← NULL
  czas.zakonczenia ← NULL
  czas.bezczynnosci ← NULL
  for (i in 1:dane.in$liczba.zadan) {
    aktualny.czas ← max(aktualny.czas, dane.in$czasy.wyslania.zadan[i])
    czas.rozporzeczcia[i] ← aktualny.czas
    czas.zakonczenia[i] ← aktualny.czas + dane.in$czasy.trwania.zadan[i]
    aktualny.czas ← czas.zakonczenia[i]
  }
  czas.bezczynnosci ← czas.rozporzeczcia[-1] - czas.zakonczenia[-dane.in$liczba.zadan]
  list(liczba.zadan = dane.in$liczba.zadan, czas.rozporzeczcia = czas.rozporzeczcia,
       czas.zakonczenia = czas.zakonczenia, czas.bezczynnosci = czas.bezczynnosci)
}
```

Zobaczymy histogram czasów bezczynności

```

T      ← 3000
lambda ← 0.5
dane.in ← wejscie(T, lambda)
dane.out ← serwer(dane.in)

czas.do.rozgrzania ← 100
zadania.rozgrzewajace ← which(dane.out$czas.roz poczenia <= czas.do.rozgrzania)
hist(dane.out$czas.bezczynnosci[-zadania.rozgrzewajace],
     xlab="czas beczynnosci", main="Czas beczynnosci po ,,rozgrzaniu'' serwera", 100)

```

Jak widzimy rozkład czasu beczynności ma atom w zerze. Masa tego atomu może być oceniona następująco

```

>mean(dane.out$czas.bezczynnosci[-zadania.rozgrzewajace]==0)
[1] 0.4805

```

Przyjmijmy, że jeżeli w 90% przypadków czas beczynności wynosi 0, to serwer jest przeciążony.

4.5.1 Zadania:

- oceń prawdopodobieństwo przeciążenia, dla $lambda = 0.92$,
- oceń prawdopodobieństwo, że serwer będzie beczynny przez ponad 5 jednostek czasu (dla różnych $lambda$),
- oceń, jaka frakcja zadań musi czekać ponad 1 jednostkę czasu na rozpoczęcie działania.