# Sannsynlighet ved Monte Carlo

Skrevet av: Kristian Sørnes

Monte Carlo-metoden hjelper oss å finne sannsynligheten for at noe skal skje når alternativene er så mange at de ikke kan telles. I denne oppgaven skal vi lage et dataprogram som gjør dette for oss.

| Biologi | Kjemi | Fysikk | Elektronikk | Informatikk | Matematikk |
|---|---|---|---|---|---|
| ○○○○○ | ○○○○○ | ○○○○○ | ○○○○○ | ●●○○○ | ●●○○○ |

## Introduksjon

Problemet vi skal ta for oss er følgende: hva er sannsynligheten for å få minst én 6'er fra fire terninger?

Normalt finner vi sannsynligheten ved å dele antallet gunstige utfall på antall mulige utfall. Dersom resultatet fra et kast med fire terninger er én 1'er, to 3'ere og én 5'er, skriver vi det som (1-3-3-5). Vi kan nå telle hvor mange av kastene som resulterer i minst én 6'er:

(1-1-1-6), (1-1-2-6), (1-1-3-6), (1-1-4-6),
(1-1-5-6), (1-1-6-6), (1-2-1-6), (1-2-2-6),
(1-2-3-6), (1-2-4-6), (1-2-5-6), (1-2-6-6),
(1-3-1-6), (1-3-2-6), (1-3-3-6), ...

Når 6'eren i tillegg kan dukke opp på en av de andre terningene, blir det fort veldig mange alternativer. Derfor lager vi et program.

## Hensikt

Hensikten med dette forsøket er å vise hvordan man beregner sannsynligheten til relativt kompliserte forsøk gjennom Monte Carlo-metoden, samt vise fordelene av programmering og hvor lett det er å endre problemstillingen. Deretter, kan vi gjøre om programmet til å finne andre sannsynligheter fra fire terninger.

## Teori

Monte Carlo-metoden følger det samme prinsippet som alle sannsynlighetsforsøk: sannsynligheten for at hendelse *A* skjer er alltid gitt ved antall utfall som gir ønsket resultat, *m*, delt på antall mulige utfall, *n*:

$$\boldsymbol{P(A) = \frac{m}{n}}$$

Vi sier gjerne at hendelse *A* skjedde ved *m* tilfeller av *n* eksperimenter, gitt at alle eksperimentene er identiske. Altså, hvis sannsynligheten er 20% vil hendelse *A* inntreffe 2 av 10 ganger, 20 av 100 ganger, 200 av 1000 ganger osv.

Monte Carlo-metoden finner sannsynligheten "baklengs" fra en slik tankegang; vi undersøker hvor mange ganger hendelse *A* inntraff fra et gitt antall forsøk, og beregner deretter den nevnte brøken. Matematisk skriver vi at

$$\boldsymbol{P(A) \approx \frac{M}{N}}$$

der *M* er antall eksperimenter der hendelse *A* inntraff, og *N* er antall eksperimenter som ble utført. Dette er bare en forenkling. Dersom vi utfører et myntkast to ganger, og får krone begge gangene, ville Monte Carlo-metoden sagt det var 100% sikkert at man fikk krone i et myntkast. I realiteten er det kun en 50% sjanse. For at Monte Carlo-metoden skal fungere må det derfor utføres et enormt antall forsøk. Formelt har vi sammenhengen

$$\boldsymbol{P(A) = \lim_{N\to\infty} \frac{M}{N}}$$

altså at Monte Carlo-metoden gir den virkelige sannsynligheten *P(A)* etter hvert som det utføres flere og flere eksperimenter. *N* blir så stort at den "går mot uendelig" ("lim" står for "limit", engelsk for "grense").

Vi bruker Monte Carlo-metoden hovedsaklig gjennom programmering, som enkelt tillater oss å gjenta eksperimentene mange ganger på kort tid. I prinsippet skal alle sannsynlighetsforsøk kunne beregnes på denne måten, men i praksis er det ikke alltid like lett. Metoden fungerer best på forsøk der resultatene er helt tilfeldige, som ved myntkast og terninger. Disse forsøkene kan datamaskinene enkelt gjenskape ved å lage sine egne tilfeldige tall. Samtidig finnes det sannsynlighetsforsøk der resultatene er få, som i et presidentvalg, men der det er så mange faktorer som spiller inn at det ikke lar seg programmere på en realistisk måte.

## Utstyr

Til dette forsøket behøver man bare en datamaskin med et valgfritt programmeringsspråk installert. Vi anbefaler Python, som er et fleksibelt språk det er lett å forstå, og som er effektivt nok for problemer som disse. Vi brukte versjon 2.7.9 av Python.

## Programmering

I Python finnes det en egen funksjon for det å lage et tilfeldig heltall: randint. Den må importeres fra numpy.random, et slags bibliotek for slike tilfeldige funksjoner. Ved å skrive **randint(1,7)** dannes et tilfeldig heltall mellom 1 og 6 hvor 7 ikke er inkludert. Det er dette som simulerer en terning. Vi lager derfor fire variabler på denne måten, **t1**, **t2**, **t3** og **t4**, som representerer en terning hver. I tillegg lager vi variabel **M** med startverdi 0 og som teller hvor mange ganger vi får minst én 6'er.

Deretter utfører vi en test, nettopp for å sjekke om noen av terningene er tallet 6. Nøkkelordet if får programmet til å sjekke om påstanden til høyre er sann eller ikke. Hvis den er det, og minst en av dem er en 6'er, øker M med 1.

```
from numpy.random import randint
M = 0
t1 = randint(1,7)
t2 = randint(1,7)
t3 = randint(1,7)
t4 = randint(1,7)
if t1 == 6 or t2 == 6 or t3 == 6 \
or t4 == 6:
      M += 1
```

Både "terningkastet" og "if-testen" må plasseres i en såkalt for-løkke, som lar oss gjenta prosessen så mange ganger vi vil. Kodelinjen for i in range(10) gjør at prosessen kjøres 10 ganger. Til slutt beregner vi sannsynligheten ved Monte Carlo-metoden; vi deler M på det totale antallet forsøk, i dette tilfellet 10. Det fullstendige programmet blir følgende:

```
#Navn: 4terninger_v1.py

from numpy.random import randint
M = 0
for i in range(10):
  t1 = randint(1,7)
  t2 = randint(1,7)
  t3 = randint(1,7)
  t4 = randint(1,7)
  if t1 == 6 or t2 == 6 or t3 == 6 \
  or t4 == 6:
    M += 1

print ''\nP(minst en 6): '', M/10.0
```

Legg merke til at vi skriver **10.0** istedenfor **10**. Dette må vi gjøre for at svaret skal være et desimaltall, og ikke blir rundet av til nærmeste heltall. I den siste linja gjør **\n** at det dannes en blank linje i utskriften, før resten av teksten skrives.

## Resultater og diskusjon

En gjennomgang av programmet ga følgende utskrift:

```
Terminal> python 4terninger_v1.py

P(minst en 6):  0.8
```

Dette gir altså 80% sjanse for å få en 6'er. Når det er 1/6 sjanse for at en terning viser tallet 6, virker dette resultatet for høyt. Vi kjører programmet noen ganger til:

**Tabell 1:** Resultatet fra 4 ulike kjøringer av programmet 4terminer_v1.py.

| Kjøring # | P(minst en 6'er) |
|---|---|
| 1 | 0,3 |
| 2 | 0,5 |
| 3 | 0,6 |
| 4 | 0,5 |

Vi ser det er en store forskjeller i disse resultatene, sannsynligheten for å få minst én 6'er er tydeligvis et sted mellom 30% og 80%. Dette skyldes at 10 gjennomganger er et for lite til at programmet får gått gjennom alle mulige resultater. Vi øker antallet gjennomganger til 100**.** Nå er det lurt å definere en egen variabel **N** lik totalt antall forsøk, slik at det kun er én kodelinje som må endres senere.

Det nye programmet ser slik ut:

```
#Navn: 4terninger_v2.py

from numpy.random import randint
M = 0
N = 100
for i in range(N):
      t1 = randint(1,7)
t2 = randint(1,7)
t3 = randint(1,7)
t4 = randint(1,7)
if t1 == 6 or t2 == 6 or t3 == 6 or t4
== 6:
      M += 1

print ''\nP(minst en 6): '', M/float(N)
```

At vi skriver **float(N)** istedenfor **N** er tilsvarende det å skrive **10.0** framfor **10** slik vi gjorde i forrige versjon.

Det nye programmet ga noen andre resultater:

**Tabell 2:** Resultatet fra 4 ulike kjøringer av programmet 4terminer_v2.py med N = 100.

| Kjøring # | P(minst en 6'er) |
|---|---|
| 1 | 0,57 |
| 2 | 0,47 |
| 3 | 0,59 |
| 4 | 0,43 |

Med kun én liten endring har vi senket usikkerheten dramatisk, til kun noen få prosenter! Likevel, vi kan gjøre bedre og øker **N** til 1000:

**Tabell 3:** Resultatet fra 4 ulike kjøringer av programmet 4terminer_v2.py med N = 1000.

| Kjøring # | P(minst en 6'er) |
|---|---|
| 1 | 0,547 |
| 2 | 0,503 |
| 3 | 0,512 |
| 4 | 0,536 |

Innen nå er mønsteret tydelig: nøyaktigheten til resultatet øker med antall gjennomganger. Dette samsvarer med at Monte Carlo-metoden fungerer best når **N** er et veldig stort tall.
Med en million gjennomganger får vi følgende:

**Tabell 4:** Resultatet fra 4 ulike kjøringer av programmet 4terminer_v2.py med N = 1 000 000.

| Kjøring # | P(minst en 6'er) |
|---|---|
| 1 | 0,518553 |
| 2 | 0,518144 |
| 3 | 0,517431 |
| 4 | 0,519033 |

Sannsynligheten for å få minst én 6'er fra fire terninger er altså i underkant av 52%. Vi kan sammenligne disse resultatene med en matematisk metode:

$$\begin{aligned} \boldsymbol{P(\text{minst én 6})} &= \mathbf{1} - \boldsymbol{P(\text{ingen 6})} \\ &= \mathbf{1} - \left(\frac{\mathbf{5}}{\mathbf{6}}\right)^{\mathbf{4}} \\ &= \mathbf{0.5177469\ldots} \end{aligned}$$

## Konklusjon

Vi brukte Monte Carlo-metoden og fant at sannsynligheten for å få minst én 6'er fra fire terninger er rett under 52%. Da vi økte **N** ble variasjonen i resultatene mindre. Usikkerheten minsket helt til under to tusendeler. Vi ser at resultatene fra programmet er svært nære den matematiske sannsynligheten.

### Prøv dette hjemme

Dette programmet kan brukes som en mal for de fleste sannsynlighetsproblemer med fire terninger. Legg merke til at "**if**-testen" er den avgjørende kodelinja i hele programmet; det er denne som sier hvilke utfall vi skal telle. Vi kunne for eksempel funnet sannsynligheten for å få fire 6'ere på ett kast ved å skrive:

```
if t1 == 6 and t2 == 6 and t3 == 6 \
and t4 == 6:
```

Se om du kommer på flere problemer som kan oppstå med fire terninger og ta utgangspunkt i programmet mitt for å løse dem. Kombiner gjerne flere enn én test!