Grundlæggende Programmering

—

Workshop I: Kanin Hop Hop

J. J. Nielsen, T. Kallehauge

October 27, 2022

Introduktion

I denne workshop simuleres brætspillet Kanin Hop Hop, med formålet at bestemme, om der en fordel ved starte spillet, ved at være anden spiller, tredje spiller og så videre.

Modelleringsproblemer kan nogle gange formuleres som matematisk analytiske/symbolske problemer og således løses direkte. Eksempler på dette er differentialligninger og lineær regression. Imidlertid er mange problemer dog for komplicerede til at kunne opstille og løse problemet med matematiske modeller, og I disse tilfælde kan en *simulerings model* ofte bruges istedet. Givet en sådan model, kan vi blot simulere systemet tilpas mange gange og så bruge Monte-Carlo metoden til at drage konklusioner om systemet. Mange fysiske systemer er tilfældige i natur, og simuleringsmodeller er derfor ofte baseret på generering af tilfældige tal. Kanin Hop Hop er et godt eksempel på et sådant system, da det er tilpas kompliceret til at vi ikke umildtbart kan analysere det matematisk, men, da der ikke er nogle *valg* i spillet (spillere slår blot med en terning), kan vi nemt simulere spillet.

Spillets regler

Se Moodle for spillets regler. Bemærk følgende:

- Spillet starter med **20 kaniner** i kaninhulen
- Der kan være så mange spillere med, som man har lyst til.
- Spillet kan spilles med tre forskellige regelsæt, når der slås en kanin. Vi refererer til de tre regelsæt som *normal*, *hurtig* (1. variant) og *langsom* (2. variant) jævnfør side 7 i regelbogen.
- Ved den hurtige variant, må man udover at slå igen **også** tage en kanin fra midten af bordet.
- I denne workshop indfører vi husreglen, at hvis to spillere har lige mange kaniner ved spillets afslutning, da vinder begge spillere.

Modellering

Spillet kan simuleres umiddelbart ud fra beskrivelsen af reglerne. Her er et par hints til at komme igang:

- Et terningeslag kan simuleres som et tilfældigt heltal mellem 0 og 5.
- Udgangshullerne kan repræsenteres som en liste af længde 5, et element for hver farve, med værdien `True`/`False` afhængigt af, om der er en kanin i hullet eller ej.

- Spillet kan simuleres i en `while`-løkke med betingelsen, at der er flere kaniner tilbage i hulen.

Der er to indstillinger i modelleringen; henholdvis antal spillere $s \geq 1$ og spilleregler normal/hurtig/langsom.

Problemet

Som der kan læses i spillereglerne, er det spilleren med de længste øre som starter. Men er det egentlig en fordel at starte? Hvem har fordelen i spillet hvis man spiller mere end 2? I workshoppen stilles problemstillingen:

Hvad er sandsynligheden p for at vinde Kanin Hop Hop for spiller $1, 2, \ldots, s$*, når der er s spillere i alt?*

For at undersøge dette skal Kanin Hop Hop implementeres og simuleres. Ved at simulere spillet et stort antal gange, kan man regne sandsynligheden ud for at hver spiller vinder ved $\hat{p}_i = n_i/n$, hvor n_i er antallet af gange spiller i vandt, og n er det totale antal simulerede antal spil. Dette er princippet bag Monte Carlo simulering.

Krav til løsning

Problemet skal, som sagt, løses ved Monte Carlo simulering ved implementation i Python. Koden til dette afleveres i slutningen af workshoppen. Følgende krav er givet til programmet:

- Programmet skal kunne estimere sandsynligheden for at spiller $1, 2, \ldots, s$ vinder ved Monte Carlo simulering. Det skal være muligt at ændre n, og det skal være muligt at ændre spillereglerne (normal/hurtig/langsom).
- Programmet skal opbygges fortrinvis af Python funktioner — undgå således kode uden for funktioner og globale variable så vidt muligt. Til hver funktion skrives en passende docstring.
- Programmet skal opdeles i minimum 2 moduler (Python scripts)[1]. Til hvert modul skrives en docstring i starten af filen.
- Programmet skal præsenteres for brugeren via en grafisk brugeroverflade (GUI). Som input skal brugeren kunne ændre spillets indstillinger (antal spillere og spilleregler) og antallet af simuleringer. Som output vises de Monte Carlo estimerede sandsynligheder $\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_s$. I øvrigt skal der vises et histogram over estimerede sandsynligheder for den seneste simulering — se et eksempel i figur 1.

[1] Her er det oplagt at lave et modul til simulering og analyse af spillet, og et modul til GUI hvor brugeren kan sætte indstillinger og resultaterne vises.

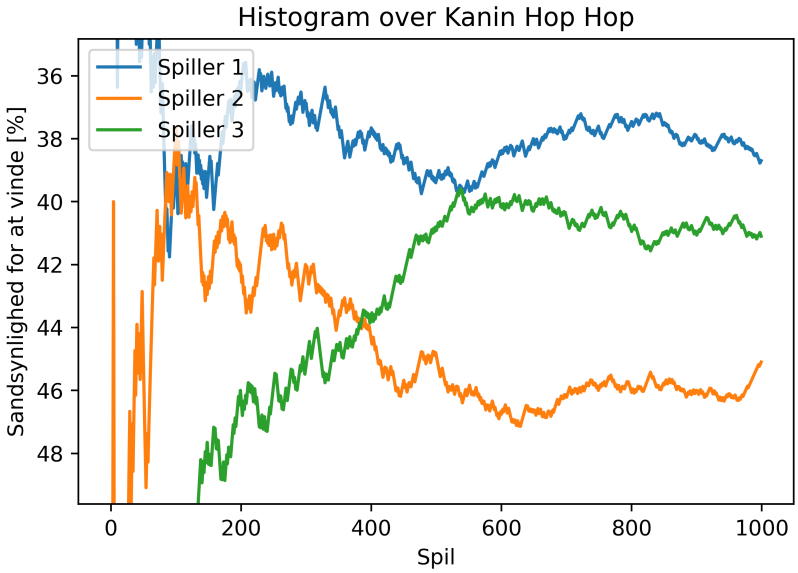

Figure 1: Histogram over 1000 spil Kanin Hop Hop med 3 spillere. Langs x-aksen er antal spil n og langs y-aksen er den estimerede sandsynlighed for at hver spiller vinder $\hat{p}_{1,n}$, $\hat{p}_{2,n}$ og $\hat{p}_{3,n}$. Som det ses på grafen opdateres disse estimater for hvert spillet spil, og når $n \to \infty$ går $\hat{p}_{i,n} \to p_i$, den sande sandsynlighed for at spiller i vinder.

Ekstra opgaver

Udover kravene til programmet, gives forslag til en række ekstra opgaver. Disse er frivillige. Følgende foreslås:

- Implementer konfidensinterval for de estimerede sandsynligheder. Et konfidensinterval er et interval hvori den sande værdi ligger med stor sandsynlighed. For N simulerede spil og den estimerede værdi for at vinde $\hat{p}$, er 95% sandsynligheds konfidens intervallet givet ved:[2]

$$I_{95\%} = \left[\hat{p} - 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}, \hat{p} + 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\right]$$

  Dette betyder at sandsynligheden for at den sande værdi p ligger i $I_{95\%}$ er 95%. Dette kan også skrives som $P(p \in I_{95\%}) = 95\%$.

- Gør ekstra meget ud af GUI delen af opgaven. Lav et flot design, der er intuitivt for brugeren at interagere med.

- Lav, udover statistik delen af GUI, mulighed for at kunne simulere enkelte spil og vise spillene grafisk. Man kan gøre så spillene spiller sig selv, eller man kan implementere at rigtige spillere skal slå med terningen, eksempelvis ved at trykke på en knap. Brug her PySimpleGUI's `Graph` element til at vise spillet. Brug eventuelt PySimpleGUI's `Menu` element, der kan bruges til at skifte imellem forskellige menuer.

Opgaver i workshop I

Følgende opgaver er givet:

- Forstå problemet, og hvordan Monte Carlo simulering kan løse det.
- Lav en plan hvor hvordan programmet skal opbygges. Hvilke funktioner og moduler skal implementeres?
- Implementer programmet i Python jævnfør ovenstående krav. Husk god dokumentation.
- Aflever programmet senest d. 6. november kl. 23:59.
- Lav peer review af de andre studerendes løsninger. Dette starter d. 7. november, og skal være indsendt senest d. 8. november kl. 23:59.

[2] Konfidensintervallet givet her er approksimativt, altså ikke eksakt. Approksimationen er dog god hvis n er høj, og når den sande værdi p ikke er tæt på 0 eller 1. Sørg derfor for at simulere mange gange.