Miniprojektuppgift i TSRT04: Femtal i Yatzy

21 mars 2017

1 Uppgift

I tärningsspelet Yatzy används fem vanliga sexsidiga tärningar. Deltagarna slår tärningarna i tur och ordning och försöker få vissa tärningskombinationer. Högst poäng ger femtal, vilket kallas för Yatzy. En deltagare har i varje omgång tre kast: ett originalkast och två då man kan spara valfritt antal tärningar och slå om de övriga.

I det här miniprojektet begränsar vi oss inte till tre kast, utan undrar: Hur stor är sannolikheten att man har samlat ihop till femtal efter ett visst antal kast? Och hur många kast behövs i genomsnitt (se beskrivning av väntevärde i avsnitt 3) för att få femtal? Ett mått på spridningen i antal kast (hur vanligt det är att man ligger nära medelvärdet) är variansen, som också beskrivs i avsnitt 3.

1.1 Redovisning

Detta är ett av miniprojekten i kursen TSRT04. För att bli godkänd på kursen måste ni lösa ett av miniprojekten enligt anvisningarna och redovisa det för en lärare på något av examinationstillfällena. Redovisningen sker på engelska så skriv er kod och era kodkommentarer på engelska (det är en bra övningen för framtiden då engelsk kodning är standard på företag). Uppgiften ska lösas i grupper om två, eller individuellt. Eftersom det rör sig om ett examinationsmoment är det inte tillåtet att dela eller visa MATLAB-kod/anteckningar för andra studenter. Det är däremot okej att diskutera uppgiften muntligen med andra grupper, exempelvis för att dela med sig av goda råd!

- Ni ska skriva en MATLAB-funktion `yatzy` som simulerar ett stort antal försök att få femtal, och beräknar medelvärdet och variansen för hur många kast som behövs. Funktionen ska även rita ett histogram över hur många kast som behövs och jämför resultatet med den analytiska lösningen i avsnitt 3. Funktionen `yatzy` ska anropa andra egenskrivna funktioner som löser vissa delproblem.
- Lösningen ska demonstreras och koden ska visas upp för lärarna på laboration 3 eller 5. Vi kommer kontrollera att ni följt anvisningarna, det finns en rimlig mängd kommentarer i koden, att plottarna är lättförståeliga och att ni kan förklara vad olika delar av koden gör. Det finns en stilguide på kurshemsidan som berättar mer om hur en bra lösningen ska se ut. Se till att läsa den och följa rekommendationerna!

- När lärarna sagt att ni blivit godkända på projektet så ska ni även skicka koden till Urkund för plagiatskontroll. Ni skickar ett e-brev till `hakan.johansson.liu@analys.urkund.se` med era fullständiga namn i textfältet. Koden bifogas i ett text-dokument döpt till kurskod_år_studentid1_studentid2.txt (t.ex. TSRT04_2017_helan11_halvan22.txt). Detta dokument ska innehålla alla funktioner och skript som examinerats, inklusive ett kort exempel på hur man kan anropa dessa funktioner för att lösa projektet.

2 Förslag på arbetsgång

Börja med att läsa hela dokumentet för att sätta dig in i problemställningen och vårt förslag till hur man kan dela upp den i delproblem.

Nedan har vi gett ett förslag på hur den stora programmeringsdelen av uppgiften kan lösas bit för bit. Vi har delat upp problemet så att det finns möjlighet att kontrollera att varje bit fungerar innan man tar sig an nästa bit. Utnyttja dessa möjligheter! Trots att arbetsgången föreslår att ni ska skriva små funktioner som löser delproblem innebär detta inte att det nödvändigtvis är bra/effektivt/snyggt att använda sig av (anropa) alla de små funktionerna när man går över till att lösa större delproblem. Vissa (alla?) funktioner löser helt enkelt så små bitar att det kan vara bättre att kopiera programkoden från den lilla funktionen till den större, istället för att låta den större funktionen anropa den mindre.

Observera att den föreslagna arbetsgången är anpassad för studenter med ingen eller liten tidigare programmeringserfarenhet. En erfaren programmerare skulle troligen dela upp problemet på ett annat sätt baserat på tidigare erfarenheter. Om du känner dig erfaren och har ett förslag till en bättre lösning så är det inget som hindrar dig att använda den istället, men om du behöver hjälp så har assistenterna mer erfarenhet av den av oss föreslagna lösningsgången. Ifall ni frångår den föreslagna arbetsgången så måste ni se till att lösningen ger åtminstone samma funktionalitet.

2.1 Många tärningskast

Skriv en funktion som simulerar ett kast med ett givet antal tärningar och returnerar resultatet. Låt funktionen ta önskat antal tärningar som inargument och låt den returnera en vektor med resultaten från tärningskasten. Ledning: Skapa en vektor med ett slumptal mellan 0 och 1 för varje tärning, multiplicera med sex och avrunda uppåt.

Testa att sannolikheten för att en tärning visar 1, 2, 3, 4, 5 eller 6 är lika stor genom att t ex kasta 1000 tärningar och plotta ett histogram över resultatet (titta på `help hist`). Hur bör histogrammet se ut?

2.2 Räkna antalet tärningar av varje sort

Skriv nu en funktion som tar ett tärningskast med fem tärningar som inargument och returnerar en vektor som talar om hur många tärningar som var ettor, hur många som var tvåor osv. Fråga assistenten om hjälp om ni känner att ni kör fast på uppgiften!

Exempel:

- Tärningskastet [1 4 2 2 4] ska ge resultatet [1 2 0 2 0 0] (en etta, två tvåor, noll treor, två fyror, och noll femmor eller sexor).

2.3 Hitta vilket resultat det finns flest av

Modifiera funktionen från förra avsnittet så att den istället returnerar vilken sort (ettor, tvåor, treor, ...) det finns flest av. För att avgöra det har ni hjälp av vektorn som ni skapade i förra avsnittet. Om tärningarna visar två par räcker det att returnera valören för det ena paret. Det kvittar vilket som returneras, så ni kan själv välja ifall ni föredrar små eller stora tal. Testa er funktion så att den verkar stämma!

Exempel:

- Tärningskastet [4 5 4 4 1] ska ge resultatet 4 (det finns flest fyror).
- Tärningskastet [1 4 2 2 4] kan ge resultatet 2 eller 4 (välj själv).

2.4 Hitta vilka tärningar som ska slås om

När man har hittat vilket resultat det finns flest av är det dags att avgöra vilka tärningar som ska sparas och vilka som ska slås om. Här finns det två varianter att välja mellan:

1. Ena alternativet är att modifiera funktionen från förra avsnittet så att den returnerar en vektor med numren på de tärningar som ska slås om. Det är alltså de tärningar som inte har det värde som räknades ut i avsnitt 2.3. Testa er funktion!

   Exempel:

   - Tärningskastet [4 5 4 4 1] ska ge resultatet [2 5] (tärning nr 2 och 5 ska slås om).
   - Tärningskastet [1 4 2 2 4] kan ge resultatet [1 2 5] eller [1 3 4] (välj själv).

2. Andra alternativet är att lägga de tärningar som ska sparas först i vektorn med tärningskast. Eftersom ni vet vilken sorts resultat som ska sparas (från deluppgift 2.3) och hur många sådana tärningar ni hade (från deluppgift 2.2) så kan ni sätta ihop en vektor med rätt antal tärningar av rätt sort. Modifiera er funktion på detta sätt och prova att den fungerar innan ni går vidare!

Exempel:

- Tärningskastet $[4\ 5\ 4\ 4\ 1]$ ska ge resultatet $[4\ 4\ 4\ \star\ \star]$ (där $\star$ ska slås om och därför kan vara vad som helst).
- Tärningskastet $[1\ 4\ 2\ 2\ 4]$ kan ge resultatet $[2\ 2\ \star\ \star\star]$ eller $[4\ 4\ \star\ \star\star]$ (välj själv).

2.5 Femtal

Utvidga nu funktionen så att den slår om de tärningar som bör slås om, och upprepar hela proceduren tills man har fått femtal (dvs. tills inga tärningar längre behöver slås om). Hur upprepar man bäst proceduren? Håll reda på hur många kast som behövdes och låt funktionen returnera detta antal.

Testa att er funktion verkar fungera! För att ni ska kunna se att den räknar antal kast rätt kan ni (tillfälligtvis) låta den skriva ut varje kast och granska siffrorna.

Se upp med om ni får t.ex. först 2 lika på ett kast och sen 3 lika av något annat på kastet efteråt, då skall ni spara på det som ni fick 3 av.

2.6 Monte-Carlo-simulering

En Monte-Carlo-simulering innebär att man utför ett experiment många gånger för att få en uppfattning om hur den underliggande sannolikhetsfunktionen ser ut. Skriv en (ny) funktion som utför experimentet med att kasta tärningar tills man får femtal ett stort antal gånger. Er kod ska kunna genomföra 10000 experiment på en eller ett par minuter, annars får ni gå tillbaka och optimera er kod. Det kan vara bra att skriva ut något med `disp` så att det går att följa hur långt MATLAB kommit i experimentet. Spara antalet kast som behövdes i varje experiment i en vektor. Plotta ett histogram över resultatet (dvs. antal kast i experimenten). Histogrammens staplar skall ha bredd 1. Funktionens inargument kan vara antalet experiment som ska utföras.

2.7 Beräkna uppskattningar av väntevärde och varians

Modifiera funktionen så att den även returnerar uppskattningar av väntevärdet och variansen för antal kast som behövs. En skattning av väntevärdet ges helt enkelt av medelvärdet av antal kast i de olika experimenten (se också ekvation (1)). En formel för hur variansen kan skattas ges i (2) i avsnitt 3. I detta fall är x_i antal kast i experiment i, och n är antalet experiment. Fråga assistenten om ni inte lyckas använda formlerna!

I avsnitt 3 ges även det teoretiska väntevärdet och variansen. Testa att er funktion ger skattningar som är ungefär rätt!

2.8 Jämför med den analytiska lösningen

Utöka nu funktionen så att den dessutom plottar den analytiska sannolikhetsfunktionen i samma figur som histogrammet från Monte-Carlo-simuleringen (Tips: Plotta först histogrammet och sedan sannolikhetsfunktionen med hjälp av `hold on`). Den analytiska sannolikhetsfunktionen finns i (3) i avsnitt 3 och ni bör beräkna den för k från 1 upp till något lämpligt tal. För att man ska kunna jämföra dem bör man tänka på följande: Om man summerar staplarnas höjd i histogrammet så ska summan bli lika med antalet utförda experiment i Monte-Carlo-simuleringen. Summerar vi sannolikhetsfunktionens värden för olika kast, däremot, blir summan ett (sannolikheten är ju ett för att vi ska få femtal förr eller senare). Alltså måste vi multiplicera antingen histogrammet eller sannolikhetsfunktionen med en skalfaktor för att de ska bli av ungefär samma storleksordning.

Observera: Histogrammet och den analytiska sannolikhetsfunktionen ska ha ungefär samma form, särskilt när ni kör 10000 experiment i Monte-Carlo simuleringen. Om detta inte är fallet så kanske något är fel i er lösning. Diskutera detta med assistenten!

Om ni har kallat din funktion `yatzy` så ska MATLAB-kommandot nedan ge en plot med den skattade sannolikhetsfunktionen (histogrammet) och den analytiska funktionen, samt det skattade medelvärdet `mhat` och variansen `s2hat`.

```
>> [mhat, s2hat]= yatzy(experiments)
```

där `experiments` är antalet experiment i Monte-Carlo-simuleringen, alltså hur många femtal som simuleras fram. Jämför vad som händer när ni har ett litet antal respektive många experiment.

2.9 Förberedelse inför redovisning

Som en förberedelse inför att presentera projektet bör ni läsa igenom avsnittet ”Redovisning” i början av detta dokument samt läsa examinations- och kodningsstil-guiden som ni hittar på kurshemsidan. På dessa ställen framgår saker som att ni måste ha minst två funktioner i er kod, funktioner och variabler måste ha beskrivande namn, det ska finnas en rimligt mängd kommentarer i koden, alla bilder ska vara självförklarande, kommentarer ska vara på engelska, etc. Om det är något av kraven i examinations- och kodningsstil-guiden som ni *inte* uppfyller så bör ni åtgärda detta *innan* ni redovisar projektet, annars kanske ni får vänta länge på att få en andra redovisningschans.

Slutligen: Glöm inte att skicka in koden till Urkund, när ni blivit godkända. Instruktionerna finns i avsnittet ”Redovisning” i början av detta dokument.

3 Användbara fakta

Sannolikhetsteori: Definitioner

Låt X beteckna utfallet (resultatet) av ett tärningskast med en tärning (X är en så kallad slumpmässig eller stokastisk variabel). P betecknar sannolikheten för ett visst utfall, $P(X = 3)$ är t.ex. sannolikheten för att få en trea.

Sannolikhetsfunktionen $p_X(k)$ definieras som $p_X(k) = P(X = k)$.

För tärningskast med en perfekt tärning gäller $p_X(k) = \frac{1}{6}$ för $k = 1, 2, \ldots, 6$.

Att kasta med flera tärningar, eller samma tärning flera gånger efter varandra, är oberoende händelser. Då gäller att $P(X_1 = k_1, X_2 = k_2) = P(X_1 = k_1) \cdot P(X_2 = k_2)$, vilket innebär att de enskilda sannolikheterna ska multipliceras.

Som ett annat exempel på en stokastisk variabel, låt Y vara antal kast vi behöver innan vi får femtal.

Väntevärdet (medelvärdet) för en stokastisk variabel Y kan tolkas som det genomsnittliga utfallet av ett försök: "Hur många kast behöver vi slå i genomsnitt innan vi får femtal?"

Varians är ett spridningsmått som anger hur stort avståndet (i kvadrat) till väntevärdet är i genomsnitt. En liten varians betyder att vi oftast behöver slå om ungefär lika många gånger innan vi får femtal, medan en stor varians betyder att antalet slag varierar väldigt från gång till gång.

Om $x_1, \ldots, x_n$ är n slumpmässiga utfall så kan väntevärdet skattas som

$$\hat{m} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{1}$$

och variansen som

$$\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \hat{m})^2. \tag{2}$$

Sannolikhetsfunktionen i uppgiften

Som beskrivet nedan, kan sannolikhetsfunktionen i uppgiften skrivas med hjälp av en matris: Låt $p(k)$ vara sannolikheten för att vi behöver k kast för att få femtal. Då gäller

$$p(k) = e_1^T A^k e_5 \tag{3}$$

där $k = 1, 2, 3, \ldots$, där ^{T} är notationen för matristransponat och

$$A = \begin{bmatrix} 0 & \frac{1}{6} & \frac{1}{36} & \frac{1}{216} & \frac{1}{1296} \\ 0 & \frac{5}{6} & \frac{10}{36} & \frac{15}{216} & \frac{25}{1296} \\ 0 & 0 & \frac{25}{36} & \frac{80}{216} & \frac{250}{1296} \\ 0 & 0 & 0 & \frac{120}{216} & \frac{900}{1296} \\ 0 & 0 & 0 & 0 & \frac{120}{1296} \end{bmatrix}, \qquad e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \qquad e_5 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Väntevärdet för antal kast som behövs är $\frac{191283}{17248} \approx 11.0902$, och variansen är $\frac{12125651655}{297493504} \approx 40.7594$.

För den intresserade: Ovanstående formler är härledda genom att tänka i följande banor: Sannolikheten för att få fem lika i kast k måste vara lika med sannolikheten för att ha fyra lika i kast $k-1$ och i kast k få ytterligare en lika, plus sannolikheten för tre lika efter kast $k-1$ och i kast k få ytterligare två lika, etc. Vi kan då ställa upp en vektor $x(k)$ enligt

$$x(k) = \begin{bmatrix} P(\text{fem lika för första gången i kast } k) \\ P(\text{fyra lika i kast } k) \\ P(\text{tre lika i kast } l) \\ P(\text{två lika i kast } k) \\ P(\text{alla olika i kast } k) \end{bmatrix}$$

och räkna ut sannolikheterna rekursivt genom $x(k) = Ax(k-1) = A^2x(k-2) = \ldots$, där A innehåller sannolikheter för resultatet i ett enskilt kast.

4 Frivilliga extrauppgifter

I det här avsnittet beskrivs några frivilliga extrauppgifter. Ifall ni är vana programmerare och känner att projektet var för enkelt så rekommenderar vi att ni även gör några av dessa extrauppgifter, så att ni lär känna MATLAB ordentligt.

MaxiYatzy: I spelet MaxiYatzy använder man sex tärningar istället för fem. Om alla sex tärningarna visar samma sak så har man fått MaxiYatzy. Redigera er kod så att antalet tärningar är ett inargument. Ni får gärna göra det till ett frivilligt inargument så att man kan utelämna det och därmed få fem tärningar. Ni kan använda variabeln `nargin` för detta.

Andra tärningstyper: Vanliga sexsidiga tärningar brukar kallas T6:or (där T står för tärning). Det finns andra tärningstyper med fler och färre antal sidor, men där sidorna fortfarande har samma form och därmed samma sannolikhet. Några exempel är T4, T8, T12 och T20. För att se hur dessa tärningar ser ut så kan ni googla på "Platonska kroppar". Redigera er kod så att antalet sidor på tärningarna är ett inargument och se hur detta förändrar resultatet och antalet experiment som krävs i Monte-Carlo-simuleringen för att få en jämn graf.

Falska tärningar: På kasinon är det viktigt att tärningarna har exakt rätt form så att sannolikheten är exakt $\frac{1}{6}$ för varje av de sex sidorna. Anta att Yatzy-tärningarna inte är exakta utan att mer sannolikt att få en sexa. Redigera er kod för att hantera detta fall. Sannolikheten för att få en sexa kan vara en in-parameter. Undersök vad som händer när sannolikheten för att få en sexa går mot ett.