Simulering af et Køsystem. Projekt 1, 02633&02525, januar '10

Projektet laves i grupper på 2-3 personer.
Rapporten skal afleveres *senest* fredag den 8. januar, klokken 12:00
til sekretær Mette Larsen, rum 011, bygning 321.

Mål

Målet med dette projekt er at udvikle et MATLAB -program, som kan benyttes til at analysere køsystemet i et supermarked. I projektet skal man foretage nogle beregninger og implementere en given algoritme ved at benytte forskellige typer af sætninger på en given datastruktur.

Specifikation af en model for køsystemet

En af de vigtigste anvendelser af edb er simulering af virkelige fænomener. Meteorologerne benytter computer-simulering til vejrforudsigelser, økonomerne til vurdering af samfundsmæssige konsekvenser, flyinstruktørerne til uddannelse af piloter, systemplanlæggerne til dimensionering af edb-systemer osv. osv. I dette projekt skal vi prøve at simulere udviklingen i køer af kunder, der venter på at blive ekspederet i et supermarked. Vi er nemlig af ledelsen blevet bedt om at producere data, som kan bruges til at vurdere, hvor mange ekspedienter der bør være, hvis disse køer ikke skal kunne blive alt for lange.

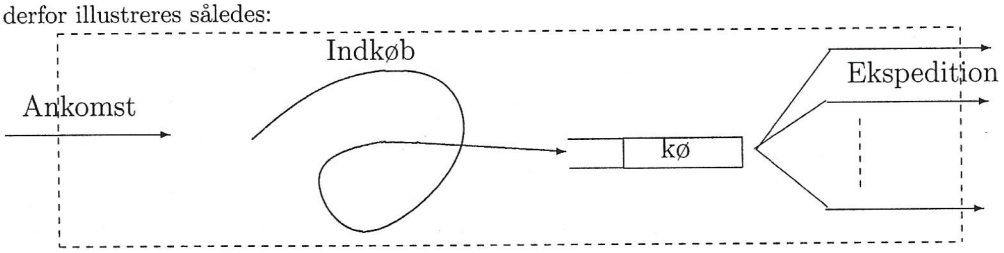
Antallet af ventende kunder afhænger af forholdet mellem tilstrømningen af kunder til ekspeditionsstederne og den tid, det tager at ekspedere en kunde, og da disse faktorer kan variere meget, har vi brug for at vide noget om denne variation.

På www.tidsskrift.dk kan man finde en artikel [1], der beskriver en undersøgelse af køsystemet i supermarkedet SALLING-SUPER, der ligger i kælderetagen til stormagasinet SALLING i Aarhus. I behøver ikke at læse denne artikel, men vi vil benytte nogle af resultaterne fra undersøgelsen til at danne en forholdsvis realistisk model for køsystemet.

Følgende undersøgelsesresultater danner baggrunden for vores model:

- Ledelsen er mest interesseret i en simulering i den travleste periode, som normalt er lørdag kl. 10 til kl. 12.30. Derfor vil vi simulere disse 150 minutter ved at sætte en konstant i programmet (variablen `antalminut`) til 150. Konstanten er en variabel, som vi let kan ændre, hvis der skal simuleres i kortere eller længere tid.
  Kommentar: Man kan som regel ikke forlange af brugerne, at de skal kunne rette konstanter i éns program, så hvis programmet skulle have været brugervenligt, skulle det have bedt brugeren om at angive antallet af minutter og derefter indlæst denne værdi i konstanten `antalminut`. I skal dog nok køre programmet nogle gange under selve programudviklingen, og det vil så virke irriterende at skulle angive værdien 150 (og flere andre konstante værdier) i hver kørsel. Derfor har vi valgt at lave denne ikke-brugervenlige udgave af et simuleringsprogram, der er let at ændre - for programmører!

- I SALLING-SUPER var der fire ekspeditionssteder, så vi sætter derfor en anden konstant (variablen `kasser`) til 4 i programmet.
- Der er ikke stor forskel på, hvor effektive de enkelte ekspedienter er til at ekspedere kunder. En ekspedient skal typisk bruge ca. $2.94/3.60 \approx 0.817$ minut på at ekspedere en kunde i den travle periode, hvor der i gennemsnit er ca. $1/4.78 \approx 0.209$ minut mellem de kunder, der kommer ind i supermarkedet. Der behøves derfor mindst fire ekspedienter i denne travle periode. På CampusNet har vi lagt en M-fil `gammarand.m`, der indeholder en funktion `gammarand(`α,β`)`. Hvis filen ligger i éns Current Dirctory, vil man kunne kalde `gammarand(`α,β`)` og hver gang få et pseudo-tilfældigt tal som værdi. Disse tal varierer (efter en såkaldt gamma(α,β)-fordeling) omkring middelværdien α/β, så vi kan altså simulere ekspeditionstiderne via en række `gammarand(2.94,3.60)`-kald, og ankomsttiderne mellem to kunder via `gammarand(1,4.78)`-kald.
- Efter ankomsten til supermarkedet benytter hver kunde i gennemsnit $3.04/0.341 \approx 8.915$ minutter på at finde varer og komme frem til ekspeditionsstederne, og da også disse indkøbstider er gamma-fordelte, kan vi simulere dem via `gammarand(3.04,0.341)`-kald.
- Der er som sagt ikke stor forskel på effektiviteten af de enkelte ekspedienter, og derfor vil kunder, der står i kø til et ekspeditionssted, som regel gå til et af de andre steder, hvis ekspedienten dér bliver ledig. Dette betyder, at vi i modellen kan slå køerne sammen til en "fælleskø", hvorfra ledige ekspedienter henter kunder frem (hvis de da ikke kommer af sig selv!). Vores generelle model for kunde-gennemstrømningen kan derfor illustreres således:



Den mindre delopgave

Før I laver simuleringsprogrammet, bør I teste, om funktionen `gammarand` virker efter hensigten. Derfor bedes I lave et script `gamparestim`, der tester funktionen ved 10 gange at gøre følgende:

1. opsamle 10000 `gammarand(3.04,0.341)`-værdier i variablene `g(1), g(2),..., g(10000)`.
2. beregne gennemsnittet `meang` af `g(i)`-værdierne, og gennemsnittet `meanlogg` af `log(g(i))`-værdierne
3. Udskrive (Maximum Likelihood) estimatet på middelværdien α/β, som er `meang`.
4. Udskrive (Maximum Likelihood) estimatet på parameteren α. Dette estimat fås som minimumspunktet for funktionen

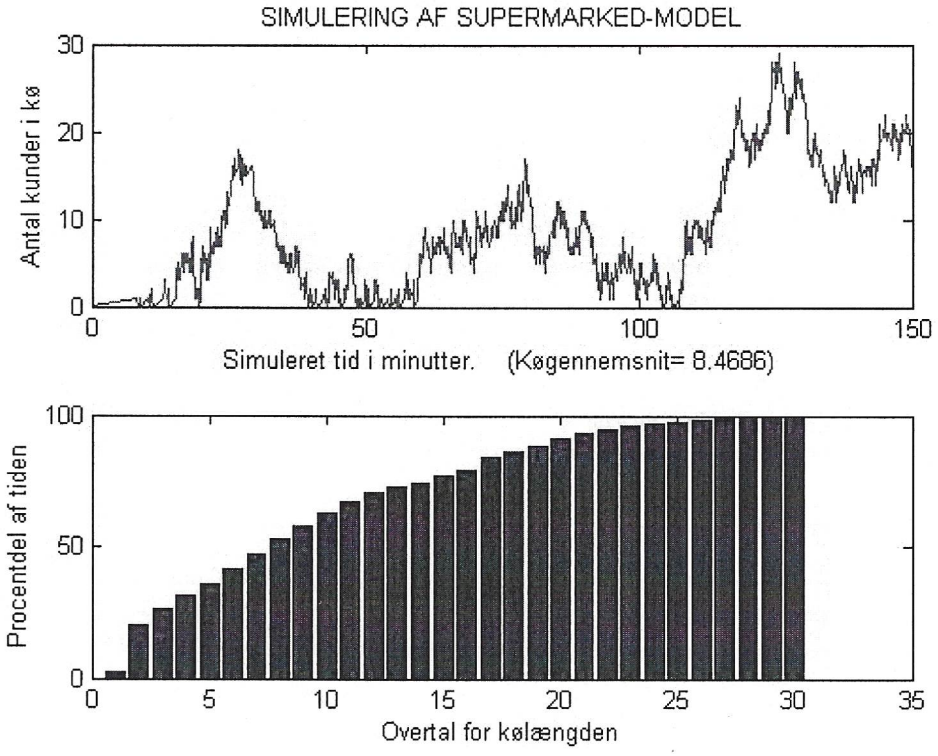
$$f = @(x)\texttt{log(gamma}(x)) - x * (\texttt{meanlogg} - 1 + \texttt{log}(x/\texttt{meang})),$$

og MATLAB -funktionen `fminbnd` kan benyttes til at finde dette punkt.

Til sidst skal `gamparestim` beregne og udskrive gennemsnittet af de 10 estimater på hhv. α/β og α. Før kørslen af `gamparestim` skal I 'cleare' Command Window, og efter kørslen printer I vinduets indhold ud. Udskriftformatet bør ligne dette:

```
>> gamparestim
Estimater 1:  middelværdi = 8.8631   alfa = 2.9951
Estimater 2:  middelværdi = 8.9657   alfa = 3.0346
Estimater 3:  middelværdi = 8.9095   alfa = 3.0557
Estimater 4:  middelværdi = 8.9493   alfa = 2.9968
Estimater 5:  middelværdi = 8.8742   alfa = 3.0035
Estimater 6:  middelværdi = 8.783    alfa = 3.0353
Estimater 7:  middelværdi = 8.9714   alfa = 3.0793
Estimater 8:  middelværdi = 8.8299   alfa = 3.0176
Estimater 9:  middelværdi = 8.9043   alfa = 3.0016
Estimater 10:  middelværdi = 8.9517   alfa = 3.1026
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Gennemsnit:  middelværdi = 8.9002   alfa = 3.0322
>>
```

På CampusNet har vi også lagt de to M-filer `super.m` og `visresultater.m`. Førstnævnte M-fil indeholder størstedelen af et MATLAB -program til simulering af køsystemet, og det er meningen, at I skal færdiggøre dette program. Til sidst i `super.m` kaldes den funktion, der ligger på `visresultater.m`, og resultatet af jeres simulering vil dermed blive vist i et Figur-vindue. Hvis `gammarand` ikke har været kaldt før simuleringen, vil figur-vinduet indeholde følgende efter den første simulerings-kørsel, og indholdet af dette vindue skal I printe ud:



Figur 1: Resultatet efter den første simulering af køsystemet.

Den nederste delfigur viser bl.a. at der i ca. 50% af tiden var *færre* end 7 kunder i kø, altså at 7 var et overtal for kølængden i ca. 50% af tiden i den første simulering af køsystemet.

Ved at studere M-filen `super.m` vil I kunne se, at den indeholder følgende:

1. Kommentarer, der beskriver formålet med programmet og de grundlæggende antagelser.
2. Konstanter svarende til SALLING-SUPER lørdage kl. 10 - 12.30.
3. Opsamling af ankomsttider til "fælleskøen" fra tid 0 og 150 minutter frem (svarende til lørdag kl. 10 - 12.30) i array-delen `ank(1 :antaleksp)`.
4. Beskrivelse af datastrukturen for "fælleskøen" `koe`, samt et array, der hedder `tider`.
5. Kommentarer, som I skal erstatte med jeres programdel.
6. Et kald af funktionen `visresultater`.

Formålet med array'et `tider` er at holde rede på de tidspunkter, hvor hver enkelt ekspedient bliver ledig, samt ankomsttiden til "fælleskøen" for den kunde, som man er nået til i simuleringen. Vi vil bede jer lave jeres programdel ud fra følgende skitse til algoritme:

For hver af de `antaleksp` kunder, der skal ekspederes, gøres følgende:

1. Ankomsttiden til køen indsættes sidst i array'et `tider`, idet vi reserverer de første elementer til ledighedstidspunkterne for ekspedienterne.
2. Kaldet `[value,index] = min(tider)` benyttes til at se, om en ekspedient er ledig på ankomsttidspunktet.
3. Er dette tilfældet, ændres køen ikke af kundeankomsten, men det gør ledighedstidspunktet for ekspedienten, som sættes til ankomsttiden + en ny ekspeditionstid (vi simulerer, at kunden ekspederes ved ankomsten).
4. Er der ikke nogen ekspedient ledig, øges køen. Dette sker ved at øge variablen `koeevents` og i element nr. `koeevents` af hhv. `koe.eventtid` og `koe.antal` notere kundens ankomsttid og en kølængde, der er 1 større end den nuværende.

   Sålænge der er kunder i køen, og den næste kunde ikke er ankommet, mindskes køen (på lignende, men ikke identisk måde som ved øgning) hver gang en af ekspedienterne bliver ledig, og ledigheds-tidspunktet for ekspedienten øges med en ny ekspeditionstid (vi simulerer, at den ledige ekspedient 'henter' og ekspederer en kunde fra køen).

Den større delopgave

Færdiggør `super.m` ved at indsætte jeres programdel dér, hvor kommentarerne angiver det. Print jeres script og det på side 3 nævnte Figur-vindue ud.

Kør simuleringsprogrammet nogle gange og konkludér, om det er nok med de 4 ekspeditionssteder i SALLING-SUPER.

Rapporten

I skal skrive en *kortfattet* rapport – det er ikke nok at kun indlevere de sider, som I er blevet bedt om at printe ud. I opfordres til at kommentere jeres resultater og skrive en konklusion, hvor I opsummerer, hvad I har lært om MATLAB -programmering i dette projekt.
Rapporten skal indeholde følgende:

- En forside med:
  - navne og studienumre
  - underskrifter, og
  - nummeret på den databar, hvor I har arbejdet (hvis I har siddet i databar).
- En kort beskrivelse af det script og den programdel, som I har lavet i de to delopgaver. Der skal bl.a. redegøres for, hvorvidt en bruger let kan ændre dem til brug i andre lignende situationer.
- Et appendix med en listning af `gamparestim.m`, det færdige `super.m`-program og de kørselsresultater, som I er blevet bedt om at printe ud.

MATLAB-koden

Følgende betragtes som god programmerings-stil, som I bør følge i jeres MATLAB-kodning.

- Kode bør så vidt muligt let kunne ændres af brugere/programmører, så den kan bruges i andre lignende situationer. I rapporten bør man angive, i hvor stor udstrækning dette er muligt, og begrunde evt. manglende generalitet af koden.
- Variable skal have *selv-forklarende navne*, som gør det nemmere at forstå koden. Hvor det er muligt, skal variabel-navne svare til dem i projektopgaven.
- Jeres kode skal inkludere *kommentarer*, der gør det klart for læseren hvad koden gør. I skal forklare *den generelle idé* bag hvert logisk sammenhængende stykke kode, og kommentere de dele, der trods passende variabel-navne ikke er selv-forklarende ud fra sammenhængen. I skal ikke skrive yderligere kommentarer (f.eks. for hver eneste instruktion), idet en erfaren programmør skal have mulighed for at se sammenhængen i koden ved at springe de for ham/hende unødvendige kommentarer over.
- Der bør anvendes et layout med passende indrykninger af indholdet i betingede sætninger og løkker, og relativt korte linier, der øger læsbarheden af koden.

Referencer

[1] J.P. Lassen, *Salling-Super analysen*, Ledelse og Erhvervsøkonomi/Handelsvidenskabeligt Tidsskrift/Erhvervsøkonomisk Tidsskrift, Bind 31 (1967).