Institut for Telekommunikation

Danmarks Tekniske Universitet

NUMERISK SIMULATION

Villy Bæk Iversen

June 1, 2007

Contents

Chapter 1

Introduktion

1.1 Introduktion

At simulere betyder at efterligne. Begrebet anvendes i forbindelse med f.eks. optræning af piloter i simulatorer, modelforsøg i vindtunneller, og i mange former for spil. Ovennævnte eksempler er alle fysiske eller ikke-numeriske simulationsmodeller. Det er ikke altid nødvendigt at materialisere en model. På datamaskiner udføres simulationer med matematiske (abstrakte) modeller under betegnelsen numerisk simulation. Det er denne form for simulering, vi skal se på i det følgende.

Ved numerisk simulation vil vi fremover forstå både det

- at opstille en model af et system eller en proces,
- at implementere denne på en datamaskine, og
- at eksperimentere med denne model for f.eks. at få en bedre forståelse af modellen eller for at analysere forskellige strategier for udnyttelsen af det betragtede system.

Vi får derfor brug for systemanalytiske værktøjer for modelbygning, datalogiske værktøjer for implementering af modeller, og statistiske og matematiske værktøjer for at fastlægge parametre (inddata) og analysere resultater (uddata) fra modellen. Numerisk simulation er i dag et af de vigtigste værktøjer til analyse af komplekse systemer og modeller.

Hvor selve implementeringen af en model på en datamaskine er håndværk, er de andre faser i højere grad en kunst.

En vigtig fordel ved en simulationsmodel er, at man kan udføre eksperimenter under kontrollerede forhold billigt og hurtigt (f.eks. simulere 100 års forløb på nogle få minutter på en datamaskine). Man kan endvidere undgå fejltagelser, der kunne få uoverskuelige konsekvenser, hvis de blev udført i virkeligheden. Endvidere kan man simulere systemer, som endnu ikke eksisterer, men hvor man ønsker at undersøge forskellige alternative løsningsmuligheder, før en af dem udvælges til konstruktion.

I sammenligning med fysiske modeller er simulationsmodeller på datamaskiner langt billigere at arbejde med. Analytiske modeller er yderligere et par størrelsesordner billigere end simulations-modeller, men ulemperne ved de billigere modeller er i almindelighed, at de bliver mindre realistiske.

1.2 Klassifikation af simulationsmodeller

Numeriske simulationsmodeller kan klassificeres på flere måder. Man opdeler således modellerne i

diskrete versus kontinuerte
deterministiske versus stokastiske

Diskret simulation

Diskrete simulationsmodeller anvendes til modellering af systemer, der ændrer sig til diskrete tidspunkter. Normalt har disse systemer et diskret tilstandsrum. Vi betragter f.eks. et system, der udvikler sig med tiden, idet systemets tilstand til et bestemt tidspunkt kan beskrives med et heltal (f.eks. antal personer i en elevator). Denne type modeller anvendes til beskrivelse af alle former for trafiksystemer (kø-systemer): vejtrafik, elevatorer, flytrafik, butikker, tele- og datasystemer etc. Ændrer systemets tilstand sig, er det i spring fra en diskret tilstand til en anden diskret tilstand, og tilstandsændringer sker momentant, dvs. til et bestemt tidspunkt. Tiden kan være enten diskret (et heltal) eller kontinuert (et reelt tal). Ofte vil man med disse modeller være interesserede i at studere systemets tilstand i statistisk ligevægt, dvs. vi interesserer os for den tilstand, systemet befinder sig i efter meget lang tids forløb.

Kontinuert simulation

Kontinuerte simulationsmodeller anvendes til modellering af systemer, hvis tilstand ændrer sig kontinuert gennem tiden. Normalt vil tilstandsrummet for disse systemer være kontinuert. Eksempelvis vil vandstanden i en flod kunne antage alle ikke-negative reelle talværdier. I disse

modeller vil man ofte være interesserede i den tidsmæssige udvikling (det dynamiske forløb) af det betragtede system. Modellerne vil ofte blive formuleret ved hjælp af differentialligninger. Systemets tilstand ændrer sig løbende med tiden, som derfor principielt er kontinuert. I analogregnemaskiner kan man direkte studere systemernes tidsmæssige udvikling. I digitale regnemaskiner vil man normalt digitalisere den kontinuerte tid i tidsenheder (time slices) og antage, at der kun sker ændringer til disse diskrete tidspunkter. Ovennævnte opdeling er ikke altid entydig, og i visse tilfælde betragter vi hybride modeller, der både omfatter diskrete og kontinuerte elementer. I praksis vil diskrete simulationsmodeller ofte være kontinuerte i tid, medens kontinuerte simulationsmodeller vil være diskrete i tid.

Deterministiske modeller

Dette er modeller, hvor alle elementer er deterministiske, dvs. der optræder ikke tilfældigheder i modellen. Disse modeller har derfor en entydig løsning, og denne form for simulering kaldes også for konsekvensberegning. Ved budgetsimulering beregner man således konsekvensen af forskellige ændringer i modellens parametre (f.eks. renteniveau). Denne form for simulering kan derfor siges at omfatte alle sædvanlige former for edb-programmer.

Stokastiske modeller = Monte Carlo modeller

Dette er modeller, hvor der indgår tilfældige (= stokastiske) elementer. Skal vi modellere et vejkryds, er vi ikke i stand til at sige eksakt, hvornår de enkelte køretøjer ankommer, men kun at der med en vis sandsynlighed kommer så og så mange biler i løbet af en tidsenhed, eller at afstanden mellem to køretøjer med en vis sandsynlighed ligger i et vist interval. Disse tilfældige elementer modelleres i datamaskinen ved hjælp af (pseudo-) tilfældige tal. I det følgende vil vi se på to eksempler på simulationsmodeller: en deterministisk model og en stokastisk model.

1.3 Eksempel på deterministisk simulation

PRODUKTIONS- OG LAGERSTYRING

Vi betragter en fabrik, der producerer et enkelt produkt. Råvarer købes udefra og ligger på lager, indtil de bruges i produktionen. Når varen er produceret ligger den på færdigvarelager, indtil den leveres til kunderne. Fabrikanten vil gerne med rimelige lagerbeholdninger kunne klare kundernes efterspørgsel uden forsinkelser. Endvidere skal arbejdsstyrken, der antages at være proportional med den producerede mængde, helst være nogenlunde stabil. Fabrikanten beslutter ved hver uges start, hvor stor en mængde råvarer, der skal indkøbes,

og hvor meget, der skal produceres i den kommende uge.

For at kunne opstille en model for dette system indfører vi følgende variable til at beskrive processen:

Ved begyndelsen af uge t har vi følgende variable til bestemmelse af **systemets tilstande**:

$R(t)$ = Råvarelager
$F(t)$ = Færdigvarelager
$B(t)$ = Ordrebeholdningen
$T(t)$ = Tilstræbt færdigvarelager

I løbet af uge t har vi følgende **ændringer** i systemets tilstande:

$X(t, t+1)$ = Ordrermængden, der indløber fra kunder
$M(t, t+1)$ = Råvarer, der leveres
$P(t, t+1)$ = Produktion
$D(t, t+1)$ = Leverede færdigvarer (effektuerede ordrer)

Det tidsmæssige forløb af størrelsen af råvarelager, produktion og færdigvarelager kan udtrykkes ved en række ligninger, der samtidig beskriver den af fabrikanten anvendte strategi:

Ordrebeholdningen:

$$B(t+1) = B(t) + X(t, t+1) - D(t, t+1) \tag{1.1}$$

Tilstræbt færdigvarelager: Antag firmaet ønsker at have varer til 5 ugers salg på lager. Denne mængde kan findes som 5 gange den gennemsnitlige ordreindgang de foregående 4 uger (glidende middelværdi). Dette er således en del af ledelsens politik (strategi):

$$T(t+1) = (5/4) \cdot (X(t, t+1) + X(t-1, t) + X(t-2, t-1) + X(t-3, t-2)) \tag{1.2}$$

Råvarelager:

$$R(t+1) = R(t) + M(t,t+1) - P(t,t+1) \tag{1.3}$$

Færdigvarelager:

$$F(t+1) = F(t) + P(t,t+1) - D(t,t+1) \tag{1.4}$$

Leveringshastighed:

$$D(t,t+1) = \begin{cases} B(t) & hvis\ B(t) < F(t) \\ \\ F(t) & ellers \end{cases}$$

Der sælges indtil færdigvarelageret er tomt.

Antag fabrikanten i sin visdom har fastlagt følgende politik for:

Råvareindkøb:

$$M(t,t+1) = P(t-1,t) \tag{1.6}$$

Dvs. der indkøbes en råvaremængde, der svarer til den foregående uges produktion.

Produktion:

$$P(t,t+1) = T(t) - F(t) + D(t,t+1) \tag{1.7}$$

forudsat at

$$P(t,t+1) = \begin{cases} 0 & \text{hvis mængden bliver negativ} \\ R(t) & \text{hvis mængden overstiger råvarelageret.} \end{cases}$$

Dvs. produktionsmængden vil være lig med det, der kræves for at opnå det tilstræbte lager, idet vi tager højde for den mængde, der skal leveres i ugens løb. Denne strategi virker umiddelbart naturlig.

Kender vi et sæt startværdier, kan vi simulere hvorledes systemet reagerer på forskellige ordremængder. Antag indledningsvis, at systemet er i ligevægt, og at det de 5 foregående uger har opereret med følgende parametre:

$D(t, t+1)$	Leverede færdigvarer	= 50 per uge
$P(t, t+1)$	Produktion	= 50 per uge
$M(t, t+1)$	Råvareforsyning	= 50 per uge
$B(t)$	Ordrebeholdning	= 50 per uge
$T(t)$	Tilstræbt færdigvarelager	= 250
$F(t)$	Færdigvarelager	= 250
$R(t)$	Råvarelager	= 150
$X(t, t+1)$	Ordremængde	= 50 per uge

Antag at denne udvikling fortsætter i endnu en uge. Derefter kommer der en uge med den dobbelte ordreindgang (salg) = 100. I den tredje uge falder salget til 0, da efterspørgslen blev dækket den foregående uge. Den fjerde og de følgende uger vender salget tilbage til 50 enheder per uge.

Hvad sker der med systemet?

For at besvare dette spørgsmål er det ligetil at udføre en deterministisk simulation med en diskret tidsakse (tidsenhed = en uge):

(a) Beregn værdierne af ligninger (1.1) - (1.4) ved start af uge t.
(b) Beregn værdierne af ligningerne (1.5) - (1.7), dvs. de nye værdier for den følgende uge.
(c) Opdater klokken til begyndelsen af næste uge.
(d) Gentag trin (a)-(c) indtil vi er færdige.

Resultaterne for 10 ugers simulation er vist i følgende tabel.

Uge	$X(t,t+1)$	$B(t)$	$D(t,t+1)$	$T(t)$	$M(t,t+1)$	$R(t)$	$P(t,t+1)$	$F(t)$	$D(t,t+1)$
1	50	50	50	250	50	150	50	250	50
2	50	50	50	250	50	150	50	250	50
3	100	50	50	250	50	150	50	250	50
4	0	100	100	313	50	150	150	300	100
5	50	0	0	250	150	50	0	300	0
6	50	50	50	250	0	200	0	250	50
7	50	50	50	188	0	200	0	200	50
8	50	50	50	250	0	200	100	250	50
9	50	50	50	250	100	100	50	250	50
10	50	50	50	250	50	150	50	250	50
11	50	50	50	250	50	150	50	250	50

Først efter uge 9 er systemet atter i ligevægt. Det er klart, strategien ikke virker tilfredsstillende. Produktionen og dermed behovet for arbejdskraft varierer voldsomt, og råvareforsyningen varierer i samme takt med en uges forsinkelse. (Råvareleverandøren får derfor endnu større problemer end vores fabrikant, hvis han anvender samme strategi).

Vi indser, at selv dette uhyre simple system er vanskeligt at behandle matematisk, og at det er nyttigt at eksperimentere med forskellige strategier. Et realistisk system med mange varetyper, lageromkostninger etc. bliver langt mere komplekst.

Øvelse 1.1: Forbedret strategi
Foreslå en bedre strategi for lager- og produktionsstyring, som ikke giver så voldsomme variationer. Implementer modellen i et Pascal-program. □

Eksempel på strategi:

$$M(t, t+1) = 50 \cdot T(t)/F(t)$$
$$P(t, t+1) = R(t)/3$$

Gennemfør simuleringen med samme efterspørgsel som ovenfor i 10 uger. Lav en grafisk afbildning af ordremængde og produktion som funktion af tiden. Forsøg selv med en anden strategi.

1.4 Eksempel på stokastisk simulation

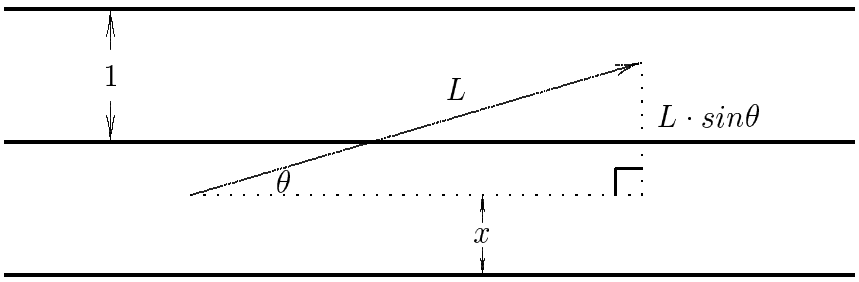
Buffons Nåleeksperiment (1777)

Værdien af konstanten π kan eksperimentelt estimeres ved følgende forsøg, der blev beskrevet af Buffon i 1777 (Buffon, 1777 [2]) (Wood & Robertson 1998 [8]). En nål af længden $L \leq 1$ kastes tilfældigt ned på en plan, hvorpå der er indtegnet parallelle linier med afstanden 1 (valgt som længdeenhed, tænk på et plankegulv). Vi ønsker at finde sandsynligheden for at nålen skærer en linie. Udledningen, der foretages i det følgende, forudsætter nogen sandsynlighedsregning, men det er primært resultatet, der af interesse.

Nålens position kan i forbindelse med vores problem beskrives ved to parametre:

θ = nålens vinkel med liniesystemet
x = er nålens afstand fra en linie (f.eks. den nærmeste linie "nedenunder").

Tilfældighedskravet medfører, at θ vil være ligeligt fordelt i intervallet $(0, \pi)$ (terminologi fra statistikken), hvilket kan beskrives med en sandsynlighedsfordeling med følgende frekvensfunktion:



$$g(\theta) = \begin{cases} 1/\pi & for\ 0 \leq \theta < \pi \\ 0 & ellers \end{cases}$$

På samme måde vil nålens afstand fra en ret linie være tilfældigt fordelt i intervallet (0,1):

$$f(x) = \begin{cases} 1 & for\ 0 \leq x < 1 \\ 0 & ellers \end{cases}$$

For en bestemt værdi af θ har vi følgende sandsynlighed for at nålen skærer en ret linie:

$$\begin{aligned} P(\text{skæring}|\theta) &= P(x + L \cdot sin(\theta) \geq 1) \\ &= P(1 - x \leq L \cdot sin(\theta)) \\ &= P(x \leq L \cdot sin(\theta)) \\ &= L \cdot sin(\theta) \end{aligned}$$

Den ubetingede sandsynlighed for skæring fås da ved at integrere over alle værdier af θ :

$$P(\text{skæring}) = \int_0^{\pi} ((L \cdot sin(\theta)/\pi) d\theta = 2 \cdot L/\pi$$

Efter n kast kan denne sandsynlighed estimeres som antal kast n_s, der resulterer i skæring, divideret med det totale antal kast n:

$$2 \cdot L/\hat{\pi} = n_s/n$$

Af denne ligning får vi følgende estimat for ã:

$$\hat{\pi} = 2 \cdot L \cdot n/n_s$$

Er nålens længde større end 1, får vi stadigvæk samme formel, hvis vi lader n_s være det totale antal linier, der skæres af nålen. Foretages der 101 kast, kan vi ved hjælp af teori fra matematisk statistik vise, at vi får et såkaldt 90% konfidensinterval (2,75 - 3,53), dvs. efter 101 kast er det kun det første ciffer, vi kan regne med. (Dvs. at med 90% sandsynlighed kan vi regne med, at dette interval indeholder den ukendte teoretiske værdi af π).

I litteraturen er der publiceret mere end 100 eksperimenter med Buffon's nål. Således foretog den amerikanske kaptajn O. C. Fox i 1864 forsøg med en 5 tommer lang nål på et gulv med 2 tommers planker. Efter 590 kast havde han registreret 939 skæringer, hvilket gav følgende estimat af π:

$$\pi = 2 \cdot 5 \cdot 590/2 \cdot 939 = 3.1416$$

Havde han foretaget et kast mere, ville han havde fået en af de følgende værdier: (3.1470, 3.1436, 3.1403, 3.1369), svarende til 0, 1, 2 eller 3 skæringer mere.

En italiener Lazzerini foretog eksperimentet i 1901 med 3408 kast og fandt følgende estimat:

$$\pi = 3.1415929$$

Havde han foretaget et kast mere eller mindre, ville han havde mistet halvdelen af cifrene. Ovennævnte viser, at det kun er successer, der publiceres - eller at forfatterne kendte resultatet og lavede fusk.

Ønsker vi at opnå flere cifre i π, end vi fandt med 101 forsøg, skal vi generelt (resultat fra teoritisk statistik) foretage 4 gange så mange forsøg for at få den dobbelte nøjagtighed. For at få et ciffer mere (10-dobbelt nøjagtighed), skal vi derfor foretage 100 gange flere kast, dvs. ca. 10000 kast. For at få yderligere et ciffer, skal vi foretage yderligere 100 gange flere kast, dvs. en million kast. *Vi ser, at selv med de største regnemaskiner, kan vi kun opnå ganske få cifres nøjagtighed.*

Øvelse 1.2: Simulering af Buffons nåleproblem
Skriv et program i Pascal til simulering af Buffons nåleproblem. □

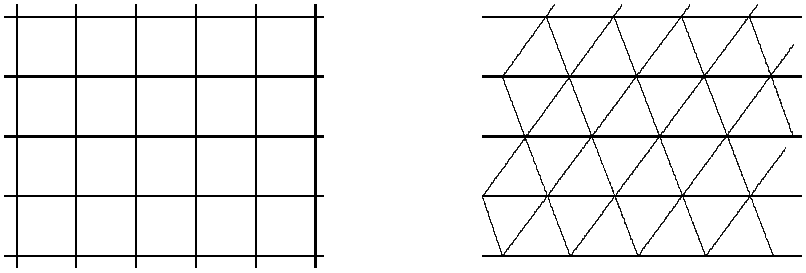
Variansreduktion

Af det foregående ser vi, at det ikke nytter blot at "køre løs" på en datamaskine for at få mere pålidelige oplysninger om en model. En anden og langt mere effektiv måde at reducere

usikkerheden på demonstreres i det følgende. Kaster vi nålen på et gulv med firkantede fliser, har vi to sæt parallelle linier, og kan derfor foretage 2 aflæsninger (forsøg) per kast, således at 101 kast giver 202 aflæsninger. Men de to aflæsninger i hvert kast er ikke uafhængige. Er der således ingen skæringer med det ene sæt linier, ligger nålen formodentlig nogenlunde parallelt med disse, og skærer derfor med større sandsynlighed det andet sæt linier. Det totale antal skæringer for de to aflæsninger i et kast varierer derfor mindre end hver enkelt af de to aflæsninger.

Med en lang nål kan det vises, at vi efter 101 kast får ca 12 gange så meget information som i det første forsøg med et sæt linier, idet vi får følgende 90% konfidensinterval: (3.09, 3.19).

Anvender vi et gulv med tre sæt rette linier (trekantede fliser) får vi yderligere 29.5 gange så meget information, dvs. 357 gange så meget information pr. kast som i første forsøg. Sådan kan vi fortsætte med at forøge antallet af parallelle liniesystemer og få mere information. I grænsen ender vi op med et deterministisk problem, idet det totale antal skæringer bliver det samme i alle forsøg.



I dette specielle tilfælde kunne vi også se på variationen (standard afvigelsen) af antal skæringer. I tilfældet med kvadratiske fliser så vi, at få skæringer med det ene liniesystem medførte flere skæringer med det andet system. Der er en negativ korrelation mellem disse, og variationen i det totale antal skæringer per kast vil være lille. Ved at anvende variationen i antal skæringer kan vi derfor (for tilfældet med firkantede fliser) med 101 kast opnå et 90% konfidensinterval på (3.138, 3.146). Dette estimat har yderligere den fordel, at det er ret ufølsomt overfor usikkerhed i bestemmelse af nålens længde. En målefejl i nålens længde på 10% vil kun give en usikkerhed på 0.1% i estimatet.

Ovennævnte har illustreret en del karakteristiske træk ved simulationsmodeller:

- simulering ved hjælp af en fysisk model sammenlignet med en numerisk datamaskinmodel.
- Usikkerhedsproblematikken. Der er grænser for, hvor nøjagtigt et resultat vi kan opnå ved blot at forøge antal forsøg.

- Med visse metoder (variansreducerende teknikker) kan man undertiden opnå langt mere information på en enkel måde.

1.5 Historiske bemærkninger

Simulation har været anvendt længe før datamaskinernes fremkomst. Buffon's nåleeksperiment omtalt ovenfor betragtes normalt, som den geometriske sandsynligheds fødsel. Et andet eksempel er W.S. Gossett's udledelse af t-fordelingen. (Gossett skrev under pseudonymet Student, hvorfor fordelingen også kaldes Student's t-fordeling). I 1908 skriver W.S. Gosset:

> "Before I had succeeded in solving my problem analytically, I had endeavoured to do so empirically (i.e. by simulation). The material used was a table containing the height and left finger measurements of 3000 criminals. The measurements were written out on 3000 pieces of cardboard, which were then very thoroughly shuffled and drawn at random ... each consecutive set of 4 was taken as a sample and the mean and standard deviation of each sample determined. This provided us with two sets of 750 z's on which to test the theoretical results arrived at. The height and left middle finger table was chosen because the distribution of both was approximately normal."

Dette er et typisk eksempel på anvendelse af numerisk simulation til dels af få ideer om en model, dels til at afprøve en teori. t-fordelingen er en fordeling vi vender tilbage til senere, idet det er fordelingen af middelværdien af en parameter, vi har observeret i en række simulationer (delkørsler).

(Man kan naturligvis undre sig over Gossett's sidste antagelse, idet kriminelle ofte anses for at være mere langfingrede end andre !!).

Inden for telefonien har man siden tyverne bygget store "trafikmaskiner" til simulering af telefonsystemer. Disse maskiner, der var baseret på trådet logik (programmeres ved at trække kabler), blev anvendt til at finde trafikkapaciteten af forskellige telefonisystemer.

Bibliography

[1] Banks, J. og J.S. Carson: Discrete-Event System Simultion. Prentice-Hall 1984. 514 pp.

[2] Buffon, G. (1777): Essai d'arithmètique morale. Suppl. Hist. Naturelle, Vol. 4 (1977) : 685–713.

[3] Datalære 1: Simulering. Danmarks Radio 1985. 40 pp.

[4] Morgan, B.J.T. : Elements of Simulation. Chapman and Hall. London 1984. 351 pp.

[5] Pidd, Michael: Computer Simulation in management Science. J. Wiley & Sons 1986. 238 pp.

[6] Pidd, Michael: Computer Modelling for Discrete Simulation. J. Wiley & Sons 1989. 274 pp.

[7] Student: The Probable Error of a Mean. Biometrika, Vol. 6 (1908):1, pp. 1-25.

[8] Wood, G.R. & Robertson, J.M. (1998): Buffon got it straight. Statistics & Probability Letters, Vol. 37 (1008) : 415-421.

Chapter 2

Modelleringsprocessen og Løsningsmetoder

Anvendelse af stokastisk simulation til løsning af problemer inden for områder som teknik, biologi, politik, økonomi, etc. er baseret på symbolske modeller af disse systemer. Ved at beskrive disse komplekse interaktive systemer med modeller, kan man drage konklusioner vedrørende virkningen af alternative forslag.

2.1 Systemer

2.2 Modeller

2.3 Modelleringsprocessen

1. Problemidentifikation
2. Målsætning
3. Systemanalyse og dataindsamling
4. Systemsyntese (modelformulering)
5. Estimation af modelparametre
6. Prælimin'r modelvalidering
7. Udvikling af program

8. Endelig validering
9. Planlægning af kørsler
10. Analyse af resultater

2.4 Løsningsmetoder

1. Analytiske metoder
2. Numeriske metoder
3. Heuristiske metoder
4. Simulering

Bibliography

[1] Mihram, G.A: The Modelling Process. IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-2 (1972):5, 621-629.

Chapter 3

Generering af tilfældige tal

3.1 Indledning

Tilfældige tal er af stor betydning for praktisk anvendelse af statistik og sandsynlighedsregning (f.eks. forsøgsplanlægning, stikprøveudtagning), og de udgør det nødvendige grundlag for al simulation på datamaskiner, som involverer stokastiske variable (Monte Carlo metoder).

Vi skal i det følgende gennemgå de mest almindelige metoder til generering af tilfældige tal. I denne sammenhæng bruger vi en del plads på de historiske metoder, idet disse er velegnede til at bidrage læseren en god forståelse af hele idegrundlaget.

Generelt kan det siges, at jo nyere en metode er, desto bedre er den. Der bør altid være adgang til fuld dokumentation for de algoritmer, der er implementeret datamaskiner. Principielt bør man altid trygt kunne anvende de tilfældigtalsgeneratorer, der er implementeret i datamaskiner. Men ser man på historien, har dette langfra været tilældet i praksis. Der findes en omfattende litteratur om tilfældigtals generatorer, og specielt følgende kilder kan fremhæves: en generel fremstilling af (Knuth, 1981 [23]), (Ripley, 1983 [38]) og bibliografierne (Sowey, 1972 [42]), (Sowey, 1978 [43]), (Sahai, 1980 [39]), (Schmeiser, 1980 [40]) og (Schmeiser, 1981 [41]).

3.2 Tilfældige tal

Definition: Ved et tilfældigt tal forstås et realiseret udfald af en rektangulært fordelt stokastisk variabel U med tæthedsfunktionen

$$f(u) = \begin{cases} 1 & 0 < u < 1 \\ 0 & \text{ellers} \end{cases}$$

U er således ligeligt fordelt i intervallet (0,1): U $\in$ U(0,1). En følge af tilfældige tal U_i (i = 1,2,...) er uafhængige ens fordelte (IID Independent and Identically Distributed) observationer $U_i \in$ U(0,1).

Det bemærkes, at ifølge ovennævnte definition kan et tilfældigt tal hverken antage værdien 0 eller 1. Dette har praktisk betydning, når man f.eks. skal dividere med et tilfældigt tal eller tage logaritmen til dette. (e.g. generering af eksponentialfordelte tilfældige tal).

Man har valgt den rektangulære fordeling, fordi

1. Den kan transformeres til en vilkårlig anden fordeling (se næste kapitel). Man behøver derfor kun én funktion for tilfældige tal i en datamaskine.
2. Man skal kun udvikle og statistisk teste én generator for tilfældige tal.

3.2.1 Tilfældige cifre

Ifølge ovennævnte definition er et tilfældigt tal et reelt tal, og de tilfældige tal ligger derfor vilkårligt tæt i intervallet (0,1). I datamaskiner er der indlagt en standardfunktion, som ved kald giver et tilfældigt tal. Da man i de digitale datamaskiner har en endelig ordlængde (opløselighed), vil denne funktion altid være en approksimation til den ideale generator (jf. definitionen).

I datamaskiner arbejder man med tilfældige cifre, som en approksimation til et tilfældigt tal.

Definition: Ved et tilfældigt ciffer forstås et realiseret udfald af en ligefordelt stokastisk variabel U med tæthedsfunktionen

$$f(u) = \begin{cases} \frac{1}{m+1} & u = 0, 1, \ldots, m \\ 0 & \text{ellers} \end{cases}$$

hvor m er det største ciffer i det betragtede talsystem.

I en digital datamaskine, der arbejder i det binære talsystem, kan et tilfældigt tal defineres som et tal, hvor de enkelte bit er tilfældige cifre, dvs. resultatet af et Bernoulli-forsøg. Har det tilfældige tal d binære cifre (bortset fra evt. fortegnsbit), normeres det ved division med (2^d-1), så at det ligger i intervallet (0,1). Der ses bort fra endepunkterne 0 og 1.

I en datamaskine, der arbejder i titals systemet, er et tilfældigt tal på tilsvarende måde sammensat af cifre, der er ligefordelt UD(0,1,...,9). Skal de tilfældige tal have middelværdien 1/2, er man nødt til at addere 1/2 til det sidste ciffer.

3.3 Metoder til generering af tilfældige tal

I tidens løb har man anvendt mange forskellige metoder til generering af tilfældige tal. Disse metoder kan inddeles i 3 grupper: mekaniske, fysiske og matematiske.

3.3.1 Mekaniske metoder

Disse metoder anvendes ofte i forbindelse med spil og i de tilfælde, hvor man kun har brug for et begrænset antal tilfældige tal (cifre). Det er let at overbevise sig om, at der er et ægte tilfældigt element i disse metoder.

Urnemetoden

(tombolametoden)
I en urne med 2 typer (farver) kugler kan man ved tilfældige trækning med tilbagelægning generere binære tilfældige cifre. Har man brikker med de decimale cifre 0,1,...,9, kan man generere cifre i ti-tals systemet. Notarius Publicus sælger således tilfældige tal genereret på denne måde. Lotto-gevinster udtrækkes på denne måde med kugler uden tilbagelægning.

Terningemetoden

Ved kast med en mønt får man tilfældige binære cifre (et Bernoulli forsøg). Med en ikosaeder-formet terning, der netop har 20 kongruente sider, kan man generere tilfældige decimale cifre, idet hvert af de 10 decimale cifre netop anbringes 2 steder på terningen. En alm. terning og en roulette vil også give tilfældige cifre.

Maskinelle metoder

Erfaringen har vist, at ovennævnte metoder ikke er tilstrækkeligt tilfældigt, idet en terning, kugle eller mønt aldrig er helt symmetrisk. Man har derfor konstrueret maskiner til generering af tilfældige tal.

I årene indtil 1960 byggede man eksempelvis elektro-mekaniske trafikmaskiner til automatisk simulering af telefonsystemer. Den svenske trafikmaskine anvendte en roterende cylinder, hvorpå der var anbragt ca. 100 huller, til generering af tilfældige tal. En føler, der bevægede sig langs en skruegang på cylinderen, gav ved kontakt med et hul en impuls, som angav et

opkaldstidspunkt. I alt var der 36 cylindere, og hullerne var anbragt, således at afstanden mellem tidspunktet for impulser blev eksponentialfordelt.

Der findes talrige andre eksempler på "Storm P" - maskiner til generering af tilfældige hændelser.

Tabelmetoden

I tabeller angiver man tilfældige cifre i 10-tals systemet. Der er udgivet talrige tabeller med tilfældige tal, f.eks. (Tippett, 1959 [44]), (Rand Corporation, 1955 [37]) og (Malmberg, 1971 [28]). Tippettis tabel er frembragt ved at vælge kort fra arkiver over befolkningsstatistik. Rand Corporations tabeller er genereret med en fysisk metode (afsnit 3.3.2), medens Malmberg's tabel er baseret på en matematisk algoritme (eksempel 3.7.6).

Det bemærkes, at cifrene ikke bliver mindre tilfældige af at blive nedskrevet. Ved gentagne anvendelser skal man naturligvis ikke starte samme sted i tabellen, men starte et andet sted eller evt. kombinere cifrene på forskellige måder.

3.3.2 Fysiske metoder

Med fremkomsten af de første elektroniske datamaskiner opstod der behov for generatorer, der var hurtigere end de mekaniske metoder, og som endvidere leverede tallene på en for datamaskinen bekvem form.

Man udviklede i datamaskinernes barndom generatorer af tilfældige tal baseret på atomare processer; f.eks. baseret på termisk støj i passive kredsløb (hvid støj), hagleffekter i elektronrør (støjdioder), radioaktiv stråling m.fl. (Isaksson, 1959 [16]), (Tocher, 1963 [45]).

Fælles ulemper for alle disse metoder er imidlertid, at

1. De er for langsomme for nutidens datamaskiner, idet man
   i en enkelt simulation ofte bruger flere millioner
   tilfældige tal.
2. Man kan ikke rekonstruere en given sekvens af
   tilfældige tal, og dermed kan man heller ikke gentage et forsøg.
3. Af samme grund er det vanskeligt at teste disse generatorer for tilfældighed.
4. Endelig kræver de en omfattende vedligeholdelse.

Disse metoder er næsten helt opgivet i forbindelse med simulation på datamaskiner.

3.3.3 Matematiske metoder

Tilfældige tal til anvendelse i datamaskiner genereres i dag så godt som altid ved hjælp af *matematiske algoritmer*. Generelt er disse metoder baseret på en rekursionsformel, hvor et tilfældigt tal bestemmes ud fra det foregående tilfældige tal samt nogle konstanter:

$$R_i = f(R_{i-1}, k_1, k_2, \ldots, k_k) \pmod{c}$$

hvor k_i'erne er konstanter, og c er det talområde, der opereres med.

Vi ser, at hele sekvensen er fastlagt ud fra konstanterne og det første tilfældige tal. Tallene er således ikke ægte tilfældige mere. Man bruger derfor betegnelsen *pseudo-tilfældig* eller kvasi-tilfældig. Fra et teoretisk synspunkt er det afgørende imidlertid, at det statistisk er umuligt at afgøre, om tallene er ægte- eller pseudo-tilfældige. Har man ikke kendskab til den algoritme, hvormed tallene genereres, kan det være særdeles vanskeligt at afgøre, om de er pseudo-tilfaeldige eller ægte-tilfældige.

For lægfolk kan de pseudo-tilfældige tal give forståelsesmæssige problemer, idet mange er tilbøjelige til (f.eks. på en lommeregner med programmer til forskellige former af spil) altid at bruge de samme startsværdier for generatoren og dermed få samme sekvens af tilfældige tal hver gang; dette gør det naturligvis muligt at forudsige spillets forløb.

I virkeligheden er det en stor fordel, at man kan rekonstruere et forsøg. Dermed kan man kontrollere andres forsøg, og man kan f.eks. udsætte et system for nøjagtig de samme tilfældige påvirkninger igen, efter at have foretaget en ændring i systemets struktur. Man bør derfor altid sammen med forsøgsresultatet opbevare de værdier, der fastlægger sekvensen af tilfældige tal.

I det følgende skal vi udelukkende se på pseudo-tilfældige tal, som vi fremover blot vil kalde tilfældige tal, hvor dette ikke kan give anledning til misforståelse.

3.4 Test for tilfældighed

Principielt bør man trygt kunne anvende de tilfældigtalsgeneratorer, som findes i bibliotek-sprogrammellet til datamaskiner, på samme måde som man anvender andre standardfunktioner. Der bør være adgang til såvel algoritmen til generering af de tilfældige tal som til resultaterne af de statistiske tests, der er udført på de tilfældige tal.

Erfaringen hidtil viser imidlertid, at ovennævnte naturlige krav ikke altid er opfyldte. Vi skal

derfor kort se på principperne for, hvorledes man tester kvaliteten af en generator. Kendskab til disse principper er også nødvendigt, når man selv skal udvikle en generator. Endvidere vil en implementeret generator normalt have både gode og dårlige egenskaber, og i forskellige anvendelser kan det være formålstjenligt at lægge vægt på forskellige egenskaber.

3.4.1 Tilfældighed

De definitioner, der i afsnit 3.2 er givet på tilfældige tal og cifre, er matematisk set præcise; men med henblik på test af generatorer og forståelse af begrebet tilfældighed findes der definitioner, som er mere operationelle.

Man kan således definere et tilfældigt tal som et tal, der består af en række nummererede cifre (binære, oktale, decimale, etc.), der har følgende egenskaber: sandsynligheden for at finde et givet ciffer på en given plads i rækken skal være uafhængig af

1. det givne ciffer,
2. cifrene på de øvrige pladser, og
3. cifrets nummer i rækken

Man kan også definere tilfældighed som kravet om, at k på hinanden følgende tilfældige tal skal, når de betragtes som koordinaler til et punkt i en k-dimensionel enhedsterning, være tilfældigt placeret i denne.

Der findes talrige andre definitioner, hvoraf nogle fremgår af det følgende.

3.4.2 Test for tilfældighed

For de grundlæggende principper og metoder for test af tilfældighed henvises der til grundkurset i statistik (Conradsen, 1976 [8]), specielt afsnit 4.1: Test for tilfældighed).

Der findes ikke et enkelt test, som analyserer alle aspekter ved en generator. Eksempler på anvendelse af en række tests er givet i (Jørgensen & al., 1974i [20]). For mere avancerede tests henvises der til (Christiansen, 1975 [5]) og (Knuth, 1981i [23]), afsnit 3.3.

Oversigt over tests

For en vilkårlig generator vil man kunne finde et test, som forkaster hypotesen, om at generatoren er tilfældig; og omvendt vil man kunne finde en generator, som består et givet test.

Man bør ikke hænge sig for meget i dette, men have i erindring, at *det er usandsynligt, at det usandsynlige ikke indtræffer på et eller andet tidspunkt.* Vi skal omtale de almindelige tests.

(a) **Frekvenstest** Det er et grundlæggende krav, at de tilfældige tal skal være ligeligt fordelte i intervaller (0,1). Deler vi derfor dette interval op i en række lige store delintervaller, kan vi forvente lige mange observationer i hver klasse. Dette kan let kontroleres med et χ^2-test.
Men dette er ikke tilstrækkeligt. En række tilfældigtalsgeneratorer antager således i løbet af en cyklus samtlige værdier i (0,1) (med datamaskinens opløslighed) netop én gang. Udfører vi derfor ovennævnte test på en hel cyklus, vil vi få lige mange observationer i hver klasse, dvs. χ^2-værdien nul. Generatoren er i virkeligheden for pæn. De enkelte bitpositioner kan også være cykliske med korte perioder (se eksempler senere), uden at dette afsløres med dette test.

(b) **Korrelationstest** Et andet grundlæggende krav er, at et tilfældigt tal skal være uafhængigt af alle tidligere tal.
Udregner vi:

$$c_h = \frac{1}{n} \cdot \sum_{\nu=1}^{n} U_\nu \cdot U_{\nu+h},$$

hvor U_ν er det ν'te tilfældige tal i rækken, så vil c_h, hvis de tilfældige tal er uafhængige, og n er stor, være normaltfordelt:

$$N\left(0.25, \frac{7}{144 \cdot n}\right)$$

For h=1, vil vi undersøge nabo observationer, men også højereordens korrelationer bør undersøges.

(c) **Run test** I (Conradsen, 1976 [8]) afsnit 4.1 er der beskrevet et run above/below test. I næste afsnit skal vi studere et run up/down test.

(d) **Serietest - test i flere dimensioner** Se afsnit 3.4.1 og (Christiansen, 1975 [5]). For en række af de hyppigste anvendte generatorer gælder det, at 3 på hinanden følgende tilfældige tal opfattet som en koordinat i den 3-dimensionale enhedsterning vil ligge på en af et begrænset antal planer i rummet. Disse generatorer er således uegnede til f.eks. at udregne rumintegraler ved hjælp af Monte Carlo metoder.

(e) **Interval test (Gap test)** Man studerer her afstanden mellem successive forekomster af tilfældige tal i et delinterval af (0,1).

(f) **Maksimum test** Fordelingen af den største observation blandt n tilfældige tal sammenlignes med fordelingen F(t) = t^n. På tilsvarende måde kan man studere den mindste.

(g) **Spektral test** Den diskrete Fouriertransformation af n tilfældige tal sammenlignes med den teoretiske, som er 1 i nulpunktet og 0 ellers (ingen periodicitet). Se (Christiansen, 1975 [5]).

(h) Permutationstest, Pokertest Hyppigheden af forekomsten af bestemte mønstre analyseres.

(i) Run test (up and down) Disse behandles mere udførligt i følgende afsnit.

3.4.3 Run test (up and down)

Vi betragter en følge af tilfældige tal og indsætter mellem disse det korrekte ulighedstegn ($\leq$ eller $>$). En følge af ens ulighedstegn kaldes en run, og antallet af ens tegn angiver længden af en run.

Eksempelvis har vi

$$U_0 < U_1 > U_2 > U_3 < U_4 < U_5 > U_6 > U_7 > U_8 < U_9 > \cdots$$

hvilket giver runs af længden

$$(1), 2, 2, 3, 1, \ldots$$

Hvis de tilfældige tal er uafhængige, er det umuligt at beregne det forventede antal runs af længden i, når vi har n tilfældige tal:

$$R(i,n) = \begin{cases} \dfrac{2(i^2+3i+1)\cdot n - (i^3+3i^2-i-4)}{(i+3)!}, & i < n-1 \\ \dfrac{2}{i!}, & i = n-1 \end{cases}$$

Det forventede antal runs af længden i eller derover er givet ved

$$CR(i,n) = \frac{2(i+1)\cdot n - i^2 - i + 1}{(i+2)!}, \quad 1 \leq i \leq n-1$$

Specielt er det forventede totale antal runs givet ved

$$CR(1,n) = \frac{2n-1}{3}$$

To på hinanden følgende runs vil være korrelerede, og man bør se bort fra den første run, der er atypisk.

For n = 10.000 tilfældige tal finder vi følgende forventede værdier:

i	R(i,n)
1	4166,7
2	1833,1
3	527,6
4	115,0
5	20,3
$\geq$	3,5

For en given generator er det meget simpelt at udregne runs og derefter med et χ^2-test afgøre, om resultatet er acceptabelt. Testet er egnet til at afsløre, om der er afhængighed mellem de tilfældige tal.

3.5 Generering af pseudo-tilfældige tal

Vi skal se på de kriterier, der er af betydning for karakteriseringen af algoritmer til generering af pseude-tilfældige tal.

a. **Hurtighed.** I mange simulationer skal man bruge flere millioner tilfældige tal. Algoritmerne bør derfor være hurtige og bestå af så få simple maskinoperationer som muligt. Der er imidlertid generelt en sammenhæng mellem algoritmens kompleksitet og de tilfældige tals statistiske egenskaber.

b. **Stor cyklus.** Da de digitale datamaskiner har en endelig ordlængde, vil sekvensen af tilfældige tal have en begrænset periodelængde. Dette gælder f.eks. klart for de algoritmer, hvor det næste tilfældige tal bestemmes ud fra det foregående. Normalt vil de eksisterende generatorer have en cyklus, som er langt større end det antal tilfældige tal, man har brug for. Dette kriterium volder derfor ingen problemer.
Man må dog være opmærksom på, at de enkelte cifre (bits eller decimale cifre) i en bestemt position for visse generatorer kan være cykliske med en meget lille periodelængde.

c. **Reproducerbarhed.** Det er muligt ud fra startsværdierne at reproducere en følge af tilfældige tal. Med forskellige startsværdier fås forskellige sekvenser. Det er muligt at teste en generator, før den bruges.

d. **Tilfældighed.** Vi har allerede i afsnit 3.3 og 3.4 set på dette kriterium. Kommercielt tilgængelige generatorer bør altid være omhyggeligt testede for tilfældighed, og de bør altid være dokumenterede. Tidligere har man set eksempler på meget dårlige generatorer. Generelt vil det gælde, at jo nyere en generator er, desto bedre statistiske egenskaber vil den have.

e. **Lagerkrav.** En generator bør kun beslaglægge en lille del af lageret i datamaskinen. Der bør således kun indgå tabeller af begrænset størrelse. Dette kriterium spiller en meget ringe rolle med nutidens datamaskinteknologi.

f. **Portabilitet.** Såvel maskinel (grundprincipper for aritmetriske operationer, ordlængde m.v.) som programmel (sprog, oversættere, afrundingsprincipper m.v.) kan medføre problemer, hvis man ukritisk overfører en generator, der er afprøvet på en datamaskine til en anden type maskine. Det er ønskeligt, at det bør være enkelt at overføre en algoritme til generering at tilfældige tal fra en datamaskine til en anden.
Tilfældigtals generatoren i IMSL (et omfattende bibliotelk af matematiske og statistiske

rutiner) vil således altid give de samme værdier for de første 23 bit på de maskiner, som biblioteket er tilpasset (Lewis, 1980 [26]).

Ovennævnte kriterier vil i praksis være i indbyrdes modstrid, og man må afveje betydningen af de enkelte kriterier mod hinanden. Generelt er det i dag vigtigere at have en god generator (pkt. d og b) fremfor en hurtig generator (pkt. a).

3.6 Simple algoritmer for pseudo-tilfældige tal

Fælles for alle algoritmer er, at man genererer et heltal R_i i intervallet (1,c-1), der kaldes generatorerens udfaldsområde. R_i transformeres til det tilfældige tal U_i ved division med c:

$$U_i = R_i/c \tag{3.1}$$

I en binær datamaskine vælges c ofte som en potens af 2, hvorved divisionen bliver meget simpel at udføre. Som vi skal se senere, har valget af c imidlertid ofte stor betydning for kvaliteten af de tilfældige tal.

Som nævnt i afsnit 3.5 vil en generator være cyklisk med en begrænset periodelængde. Betegner vi generatorens startsværdi med R_0, vil en generator antage værdierne

$$R_0, R_1, R_2, \ldots, R_s, R_{s+1}, \ldots, R_{s+p}, \ldots$$

Hvis $R_{s+p} = R_s$ vil generatorens periodelængde være p.

Sekvensen af tilfældige tal vil da bestå af et acyklisk indsvingningsforløb af længde s efterfulgt af et periodisk gentaget forløb med periodelængden p. I specielle tilfælde kan indsvingningen bortfalde (s=0), eller generatoren kan degenerere til en periodelængde p=1 (eksempel 3.6.1 og 3.6.2).

I praksis vil periodelængden dog normalt være meget stor ($\sim 10^{10}$), så selv en moderne datamaskine vil kræve meget lang tid til at generere en hel cyklus.

3.6.1 Midtkvadratmetoden (Midsquare - metoden)

Denne algoritme fra datamaskinernes barndom omtales af historiske grunde. Den blev foreslået af (von Neumann, 1951). Den er hurtig og kræver så godt som ingen lagerplads, men statistisk set er den dårlig, og den anvendes ikke mere.

Har datamaskinen en ordlængde på 2d bit, genereres et tilfældigt tal ved at kvadrere det foregående (derved fås 4d bit) og beholde de 2d midterste bit. Startsværdien R_0 bestemmer således hele sekvensen.

Tilnærmelsen til den rektangulære fordeling er ofte dårlig, og der er ingen sikkerhed for, at alle værdier i udfaldsområdet kan forekomme. Generatoren når ofte sin cykliske tilstand meget hurtigt, og periodelængden er ofte meget lille.

Eksempel 3.6.1: 8 bit datamaskine
Betragt en datamaskine med ordlængde 8 bit. Antager generatoren blot værdien 0 eller 165 én gang, vil den fortsætte med denne værdi. Alternativt antager den aldrig disse værdier. □

Eksempel 3.6.2:
Antager de n første/sidste bit på et tidspunkt værdien 0, vil de $2n$ første/sidste bit blive 0 i det kvadrerede tal. Det næste tilfældige tal vil derfor indeholde 0 på de $2n - d$ første/sidste pladser. Hvis $n > d$, ses det, at sekvensen hurtigt degenererer til at antage værdien 0. □

En tilsvarende metode, midtproduktmetoden, genererer et tilfældigt tal ved at multiplicere de to foregående tilfældige tal med hinanden og beholde de midterste bit i resultatet. Selv om denne algoritme statistisk set er bedre, er den dog stadigvæk langtfra tilfredsstillende.

3.6.2 Fibonaccimetoden

Denne generator er opkaldt efter Leonardo af Pisa, der under pseudonymet Fibonacci i 1202 i sin bog "Liber Abaci" behandlede talfølgen

$$x_i = x_{i-1} + x_{i-2}, \quad i \geq 2, \quad x_0 = x_1 = 1$$

Denne algoritme anvendes i Fibonaccimetoden, idet vi modificerer den på følgende måde:

$$R_i = (R_{i-1} + R_{i-2}) \pmod{c}, \tag{3.2}$$

hvor R_i er rest modulo c af $(R_{i-1} + R_{i-2})$ (modulusdivision). Til belysning af dette anføres følgende definitioner fra talteorien.

Definition: Kongruens modulo c To tal a og b siges at være kongruente modulo c, hvis deres differens er et helt multiplum af c:

$$a = b \pmod{c} \tag{3.3}$$

Hvis b-a er delelig med c, så er a og b altså kongruente modulo c.

Definition: Rest modulo c (modulusdivision) For en given værdi af b kaldes det mindste positive heltal x, for hvilket

$$x = b \pmod{c} \tag{3.4}$$

for rest modulo c af b.

Eksempel 3.6.3: b = 13, c = 3
For $a = 1, 4, 7, 10, 13, 16, \ldots i$ vil a og b være kongruente modulo 3. Rest modulo 3 af 13 er $a = 1$. □

Med Fibonaccimetoden, der undertiden kaldes en additiv kongruensmetode (se afsnit 3.7), bestemmes et tilfældigt tal således ud fra de to foregående, og den vil derfor have en maksimal periodelængde på $c^2 - 1$. Denne periodelængde kan imidlertid kun opnås for $c = 2$ og $c = 3$.

Valget af c er kritisk for generatorens egenskaber. Ofte vælges c som en potens af 2. Derved bliver modulusdivisionen meget simpel, idet vi blot adderer R_{i-1} og R_{i-2} og ser bort fra et eventuelt overløb (mente). Dermed fås en hurtig generator, men kun på bekostning af alvorlige defekter i den genererede sekvens.

De n første (dvs. højrestillede) bit i R_i afhænger således kun af de n første bit i R_{i-1} og R_{i-2}, og det kan vises, at det n'te bit vil have en cyklus på $3 \cdot 2^{n-1}$.

Øvelse 3.1: 4 bit datamaskine, $R_0 = R_1 = 1$, $c = 2^4$
Denne generator har med de givne startsværdier intet indsvingningsforløb. Dens cyklus er 24. Periodelængden for de enkelte bit er (fra højre) henholdsvis 3, 6, 12 og 24. Rekonstruer generatoren, idet 0 af nemhedsgrunde medtages, og vis, at de statistiske egenskaber er dårlige. □

Ovennævnte uheldige cykliske egenskab for de enkelte bitpositioner kan undgås ved at vælge c anderledes. Det kan således vises (Marqvardsen, 1974 [29]), at hvis c vælges som et primtal med det sidste decimale ciffer 3 eller 7, så fås en periodelængde på $2(c+1)$ for *alle* bitpositioner. Denne periodelængde opnås for alle startsværdier, naturligvis undtagen $(R_0, R_1) = (0, 0)$.

Da generatoren således kan antage $(c^2 - 1)$ forskellige startsværdier, der alle har periodelængden $2(c + 1)$, får vi

$$\frac{(c^2 - 1)}{2(c + 1)} = \frac{c - 1}{2}$$

forskellige cykler. Startværdien afgør, hvilken af de betragtede cykler vi genererer i et givet tilfælde.

Eksempel 3.6.4: 4 bit datamaskine, $c = 13$
For denne generator opnås en cyklus på 28 for alle startsværdier. Der er 168 forskellige startsværdier, som giver 6 forskellige cykler. □

Eksempel 3.6.5:
Som et eksempel på valg af generatorkontanter for en 16 bit datamaskine anføres følgende (Jørgensen & al., 1974 [20]):

$$c = 2^{16} - 39 = 65.497$$

Denne generator har en periodelængde på 130.996, og antallet af forskellige cykler er 32.748. □

Fibonaccigeneratoren (3.2) er meget simpel og enkel og brugbar til visse anvendelser, hvor man ikke overstiger periodelængden.

Til andre anvendelser er den uanvendelig. For fastholdt værdi af R_{i-2} vil (R_i, R_{i-1}) betragtet som koordinat til et punkt i enhedskvadratet ligge på én ud af to rette linier (fig. 1), idet vi har:

$$R_i = R_{i-1} + R_{i-2}$$

eller

$$R_i = R_{i-1} + R_{i-2} - c \tag{3.5}$$

(R_i, R_{i-1}, R_{i-2}) fastlægger netop to planer i en enhedsterning. Der er således en kraftig korrelation mellem to på hinanden følgende tilfældige tal, og generatoren er helt uegnet til at generere tilfældige tal i flere dimensioner.

Hvis vi f.eks. bortkaster hvert andet tilfældige tal, vil vi fordoble antallet af planer til 4 ækvidistante planer, men til gengæld bliver generatoren langsommere. Generatoren kan også forbedres ved at lægge de k (k > 2) foregående tilfældige tal sammen, men også dette gør generatoren langsommere, så at den ikke kan konkurrere med de generatorer, vi omtaler i det følgende.

3.7 Lineære kongruensmetoder

Metoder til generering af tilfældige tal baseret på lineære kongruensmetoder har følgende form:

$$R_i = (a \cdot R_{i-1} + b) \pmod{c} \tag{3.6}$$

R_i er netop rest modulo c af den lineære transformation (a $\cdot$ R_{i-1} + b).

Vi indfører følgende betegnelder:

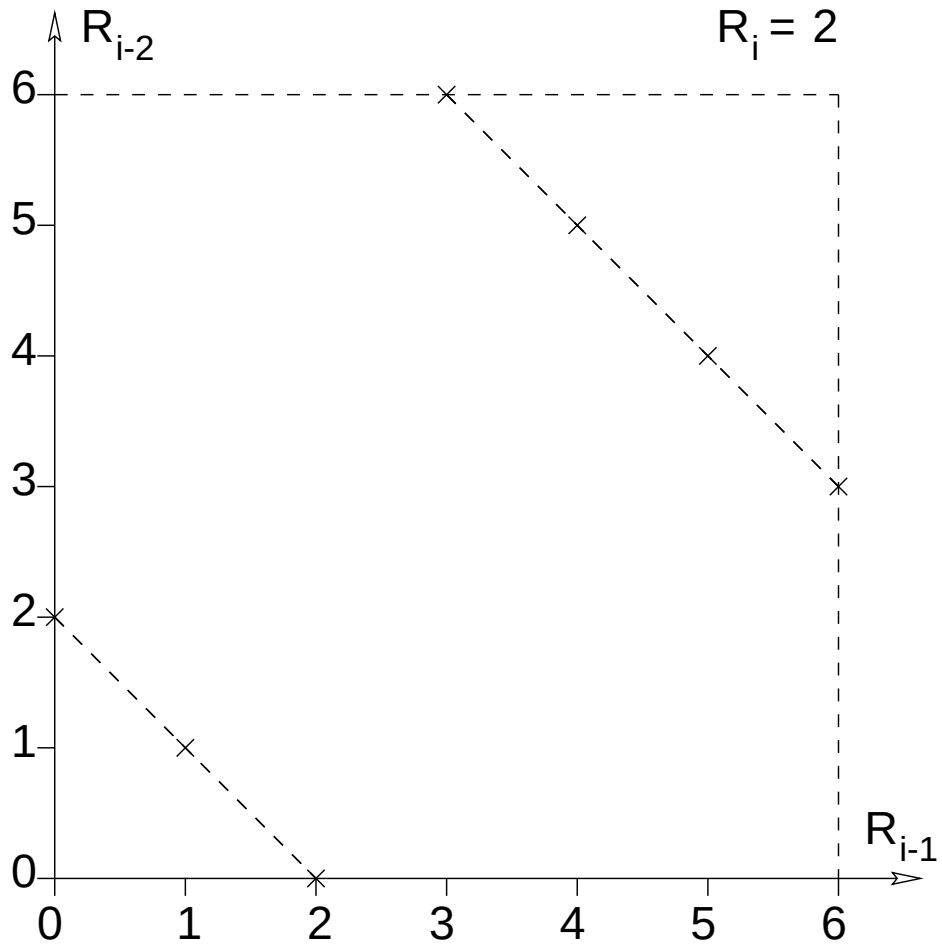


Figure 3.1: *Tre på hinanden følgende værdier* $(R_i, R_{i-1}, Ri-2)$ *af en Fibonaccigenerator* $(c = 7)$*, hvor* $R_i = 2$*. Betragtet som koordinater til et punkt i rummet bestemmer observationerne to planer i en terning. For* $R_i =$ *konstant fås et snit gennem terningen.*

a = multiplikator
b = tilvækst
c = modulus

Vi skelner mellem følgende algoritmer:

a = 1 : additiv kongruensmetode,
b = 0 : multiplikativ kongruensmetode,
b $\neq$ 0, a $\neq$ 1 : Blandet kongruensmetode.

Det ses umiddelbart, at den maksimale periodelængde for disse generatorer er c.

Kongruensmetoder er de mest udbredte i praksis.

3.7.1 Den multiplikative kongruensmetode

Denne metode går også under betegnelserne "den rent multiplikative metode", "power residue metoden" og "Lehmers kongruensmetode" (se eksempel 3.7.2). Den er baseret på følgende algoritme:

$$R_i = a \cdot R_{i-1} \pmod{c} \tag{3.7}$$

Kvaliteten af denne generator er stærkt afhængig af, hvorledes konstanterne vælges, og vi får i det følgende brug for yderligere et par resultater fra talteorien (jf. (Marqvardsen, 1974 [29])):

Definition: Primitiv rod En primitiv rod a af et primtal c er et tal, for hvilket det gælder, at

$$x_i = a^n \pmod{c}$$

for n = 1, 2, ...,c-1 vil antage enhver af værdierne 1, 2, ..., c-1 én og kun én gang.

Eksempel 3.7.1:
Primitive rødder i $c = 7$ er $a = 3$ og 5. □

Er a en primitiv rod i c vil

$$x_i = (x_{i-1} \cdot a) \pmod{c} \tag{3.8}$$

naturligvis også antage enhver af værdierne $1, 2, \ldots, c-1$ for $i = 1, 2, \ldots, c-1$, når x_0 vælges blandt heltallene $(1, 2, \ldots, c-1)$.

Sætning 3.1 Hvis a er en primitiv rod af primtallet c, så vil også alle tal

$$a^n \pmod{c}$$

være primitive rødder af c, såfremt n er primisk med $c - 1$.

For bestemmelse af et tals primitive rødder henvises der til (Marqvardsen, 1974 [29]).

Ovennævnte talteori kan udnyttes til at opnå den maksimale periodelængde, som af (3.7) umiddelbart ses at være $c - 1$.

Vi skal i det følgende give en generel vejledning i valg af konstanter i (3.7), jf. (Marqvardsen, 1974 [29]). De angivne retningslinier er baseret på en kombination af teori og erfaring.

1. a og c må ikke have en fælles faktor, da R_i ellers kan antage værdien 0 for alle i større end en vis værdi.

2. c bør være et stort tal, da generatorer af denne type har en maksimal periodelængde på $c - 1$.

3. a bør være af samme størrelsesorden som $\sqrt{c}$, da korrelationen mellem to på hinanden følgende tal bliver mindst mulig omkring denne værdi. Man bør være opmærksom på, at man ofte vil *forringe* generatorens statistiske egenskaber ved kun at anvende hvert andet eller hvert tredie tal i rækken.

4. c bør være et primtal. Man kan da opnå den maksimale periodelængde $(c - 1)$ for alle bitpositioner ved at vælge a som en primitiv rod af c. Periodelængden bliver uafhængig af startværdien R_0. Anvender vi netop en hel cyklus af tilfældige tal, vil disse i visse henseender være for pæne (jf. afsnit 3.4.2, pkt. a). Man bør derfor ikke anvende en hel cyklus, og man bør starte med forskellige værdier af R_0.

5. Lille eksekveringstid. Multiplikationer og divisioner er langsommere at udføre end additioner, subtraktioner og skifteoperationer. Ved at vælge c som en potens af 2 kan modulusdivisionen udføres meget hurtigt. Herefter kan a ikke vælges som en potens af 2 (jf. punkt a). Det anbefales da at vælge

   $$a = 8 \cdot h \pm 3,$$

   hvor h er et heltal. Hvis R_0 da vælges som et ulige tal, fås periodelængden $c/4$. (jf. eksempel 3.7.1.3). Modulusdivisionen kan i maskinkode dog udføres meget hurtigt for vilkårlige værdier af c ved en teknik kaldet "divisions simulation" (Kobayashi, 1978 [24], p. 237). Man bør derfor alligevel vælge c som et primtal (pkt. d.).

   I en datamaskine er det sidste bit ofte fortegnsbit. I en 32 bit datamaskine har man derfor kun 31 bit til rådighed til de aritmetriske operationer.

   Hvis man i et højere programmeringssprog ønsker at kunne vælge c frit, må man sikre sig, at a · (c-1) ikke overstiger det største tal, som den anvendte datamaskine kan repræsentere.

Vi skal i det følgende omtale en lang række eksempler på generatorer.

Eksempel 3.7.2:
Lehmer anvendte i 1951 følgende generator til simulation på ENIAC:

$$a = 23 \ , \ c = 10^8 + 1$$

Denne generator har periodelængden 5.882.352, som er fundet empirisk. Det bemærkes, at ENIAC arbejdede i det decimale talsystem (ligesom lommeregnere), medens moderne datamater arbejder i det binære talsystem. □

Eksempel 3.7.3:
Dette er et eksempel på, at c er valgt som en potens af 2:

$$a = 2^{16} \ , \ c = 2^{31}$$

Vælges startværdien som et ulige tal, fås periodelængden 2^{29}. Er startværdien et lige tal, bliver den "kun" 2^{28}. Betragter vi 3 på hinanden følgende tilfældige tal som koordinater til et punkt i enhedsterningen, vil dette ligge på én ud af 15 ækvidistante planer. Denne generator er derfor helt uegnet til en række anvendelser, og den er blevet skiftet ud med GGL-generatoren i IBM's bibliotek. □

Eksempel 3.7.4: IBM's CGL-generator
Følgende generator er væsentlig bedre end den foregående:

$$a = 7^5 \ , \ c = 2^{31} - 1$$

I en 32 bit datamaskine er det største primtal mindre end 2^{31} netop (2^{31}-1=2147483647). Endvidere er 7 en primitiv rod af dette tal c. Da $n = 5$ ikke er en faktor i $c - 1$ ($= 2^{31} - 2 = 2 \cdot 3^2 \cdot 7 \cdot 11 \cdot 31 \cdot 151 \cdot 331$), er 7^5 også en primitiv rod af c.

Denne generator anvendes nu i IBM's scientific subroutine library i stedet for RANDU.

For a = 455470314 og samme værdi af c fås også en god generator. □

Eksempel 3.7.5:
(Jørgensen & al., 1974 [20]) Dette er en generator for en 16 bit datamaskine

$$a = 721 \ , \ c = 2^{15} - 19$$

c er det største primtal mindre end 2^{15}, og a er en positiv primitiv rod i c. Generatoren har derfor periodelængden $c - 1$. Dette gælder for alle bitpositioner. □

Eksempel 3.7.6: Erlang-T tabellen
(Malmberg, 1971 [28]). Til generering af denne tabel er anvendt generatoren

$$a = 5^{15} \ , \ c = 2^{35}$$

Dette er følgelig for en 36-bit datamaskine. Til generering af en tabel vil man normalt aldrig anvende en så simpel og dårlig generator. □

Eksempel 3.7.7: IBM 1401
Dette er en datamaskine, der arbejder med decimale tal:

$$a = 1011\ ,\ c = 10^8$$

Periodelængden er $5 \cdot 10^6$. Man bør være opmærksom på, at det er af betydning for valg af konstanter, om datamaskinen arbejder binært eller decimalt (f.eks. alle lommeregnere). □

Eksempel 3.7.8: 32-bit datamaskine
En generator med konstanterne

$$a = 1943\ ,\ c = 10^6 + 9$$

giver udmærkede resultater for en 32 bit datamaskine (Marqvardsen, 1974 [29]). □

Eksempel 3.7.9: GPSS
I simulationssproget GPSS anvendes den multiplikative komgruensmetode til generering af tilfældige tal. I GPSS III findes der én generator. I GPSS V findes der 8 generatorer (RNi, i = 1, 2, . . .,8). Generatorerne er ens, men tildeles automatisk forskellige startværdier. Man kan evt. selv bestemme startværdierne med RMULT-kommandoen. Man kan f.eks. reservere en generator til generering af kundeankomster, medens betjeningstid, valg af betjeningssted etc. bestemmes med de andre generatorer. Man kan da sende nøjagtigt den samme kundestrøm ind i et system, hvor man har ændret f.eks. betjeningstid eller struktur. Se også eksempel 3.8.1. □

Eksempel 3.7.10: Lommeregnere
I større lommeregnere bør man anvende samme algoritmer som til datamaskiner, der arbejder i det decimale talsystem. Til små lommeregnere kan anføres et par udbredte algoritmer (reelle tal), som også kan bruges på moderne lommeregnere::

$$R_i = \text{Fraction}\{147 \cdot R_{i-1}\} \quad \text{, (”147-generatoren”)}$$
$$R_i = \text{Fraction}\{(R_{i-1} + \pi)^5\} \quad \text{, (HP-25)}$$

Fraction fjerner tallets heltalsdel. Det er vanskeligt at udtale sig om kvaliteten af denne type generatorer.

En fordel ved den multiplikative kongruensmetode er, at den hverken antager værdierne 0 eller 1. □

Eksempel 3.7.11: Mikrodatamater
(Jennergren, 1984 [18]). Til anvendelse på mikrodatamater (PASCAL/MT+) foreslås følgende generatorer:

$$a = 5^{13} \ , \ c = 2^{35}$$

(sml. eksempel 3.7.1.6). I (Jennergren, 1984 [18]) beskrives i detaljer, hvorledes den implementeres i PASCAL. □

3.7.2 Den blandede kongruensmetode

Denne metode, der også kaldes den blandede multiplikative metode, adskiller sig fra den multiplikative kongruensmetode ved, at tilvæksten b er forskellig fra nul:

$$R_i = (a \cdot \ R_{i-1} + b) \ (\text{mod } c) \, . \tag{3.9}$$

Denne metodes fordel frem for den multiplikative kongruensmetode er, at man kan vælge c som en potens af 2 og alligevel opnå den maksimale periodelængde, der er c, idet den også tillader R_i at antage værdien nul. Derudover har den ingen fordele. Ud fra rekursionsformlen (3.8) kan det vises, at

$$R_i = (a^i R_0 + \frac{a^i - 1}{a - 1} b) \ \ (\text{mod } c) \, . \tag{3.10}$$

Også her er valget af konstanter af afgørende betydning, og man anbefaler følgende retningsregler for valg af disse. Den maksimale periodelængde c opnås for:

1. b ulige og relativt primisk med c
2. $a \ (\text{mod } q) = 1$ for enhver primfaktor q i c.
3. $a \ (\text{mod } 4) = 1$ hvis 4 er en faktor i c.

Der findes yderligere en række anbefalinger:

1. $a \ (\text{mod } 8) = 5$ hvis c er en potens af 2, dvs. for en binær datamaskine.
   $a \ (\text{mod } 200) = 21$ for en decimal datamaskine.
2. a bør være af samme størrelsesorden som $\sqrt{c}$.
3. c bør være stor, af samme størrelsesorden som datamaskinens ordlængde. Ofte vælges $c = 2^d + 1$ for binære datamaskiner, og $c = 10^d + 1$ for decimale datamaskiner, idet disse værdier også giver mulighed for hurtig eksekvering.

Uanset hvilke heltallige værdier af a og c vi vælger, så vil de mindst betydende bit i de genererede tal være stærkt periodiske. Generelt er det n'te bit periodisk med en maksimal periodelængde 2^n. Dette kan undgås ved at vælge a ikke-heltallig som:

$$a = 1 + 2^{-i}$$

(Maqvardsen, 1974 [29]).

Eksempel 3.7.12: Dårlig generator
$c = 10$, $a = 13$ og $b = 10$. Med startværdien $R_0 = 2$ finder vi følgende sekvens

$$2, 6, 8, 4, 2, \ldots$$

Generatoren har periodelængden 4 og antager kun 4 af de 10 mulige værdier. Den er derfor uegnet til at generere tilfældige cifre i 10-talssystemet. □

Eksempel 3.7.13:
(Kobayashi, 1978 [24])

$$a = 314159269, \qquad b = 453806245, \qquad c = 2^{31}$$

Denne generator opfylder de ovennævnte regler 1 - 4 og har den maksimale periodelængde c. (Cifrene i a er hentet fra π). □

Eksempel 3.7.14: (Jørgensen et al., 1974)

$$a = 2^d + 1, \quad b \simeq c \cdot (1/2 - \frac{1}{6\sqrt{3}}), \quad c = 2^{16}$$

Denne generator har periodelængden 2^n for det n'te bit, dvs. generatorens cyklus er 2^{16}. □

Eksempel 3.7.15: TI-59 lommeregner

$$a = 24298, \qquad b = 99991, \qquad c = 199017\,.$$

Denne algoritme anvendtes af TI-59, som var en programmerbar lommeregner, der arbejdede i det decimale talsystem. □

Eksempel 3.7.16: Lommeregnere og mikrodatamater
I (van Es & al., 1983 [11]) undersøges to generatorer for lommeregnere og mikrodatamater, hvoraf følgende er den bedste:

$$a = 314159221, b = 211324863, c = 10^9$$

Dette er en algoritme for decimale maskiner. □

3.8 Kombination af generatorer

De i afsnit 3.7 omtalte generatorer dominerer helt i datamaskinernes programbiblioteker, og de er tilstrækkelige til mange formål. Til en lang række opgaver er deres statistiske egenskaber imidlertid for dårlige, og man har brug for endnu bedre generatorer.

Jo bedre en generator skal være, desto mere regnetid og lagerplads vil det kræve. Disse ressourcer bliver i disse år af stadig mindre betydning sammenholdt med arbejdskraften. Med en god generator kan man fjerne en række potentielle fejlkilder ved programudviklingen og således spare udviklingstid. Endvidere vil en perfekt generator normalt hurtigere give det ønskede resultat.

Den mest effektive måde at opnå en bedre generator på er at kombinere flere simple generatorer til én.

3.8.1 Kortblandingsgeneratoren

Dette er en generator, hvor man søger at fjerne korrelationen mellem to på hinanden følgende tal. Den består af en hovedgenerator og en indeksgenerator.

Hovedgeneratoren er den generator, man vil forbedre, og dette sker ved at blande de tilfældige tal. Man starter med at generere n tilfældige tal (f.eks. $n = 64$), som lægges ind i en vektor.

Med indeksgeneratoren genereres et ligefordelt tilfældigt tal UD(1,2,$\cdots$,n) (se næste kapitel), som udpeger det element i vektoren, vi skal anvende som næste observation. Den ledige plads fyldes op ved hjælp af hovedgeneratoren. Enhver observation kræver således generering af to tilfældige tal. Indekxgeneratoren kan dog være meget simpel.

Det er ikke muligt kvantitativt at afgøre, hvor stor forbedringen bliver, men grundideen bag metoden er umiddelbart forståelig.

Eksempel 3.8.1: GPSS V
(jf. eksempel 3.7.9) I GPSS V (et meget udbredt simuleringssprog) anvender man 8 ens generatorer af typen multiplikativ kongruensgenerator. Hver generator veksler imidlertid mellem 8 forskellige værdier af konstanten a (c er ens for alle). Visse bit i det sidst genererede tilfældige tal afgør, hvilken konstant, der skal anvendes næste gang. De enkelte generatorer er således helt uafhængige af hinanden, men man har forsøgt at forbedre dem ved at udnytte kortblandingsprincippet på de enkelte multiplikatorers a. □

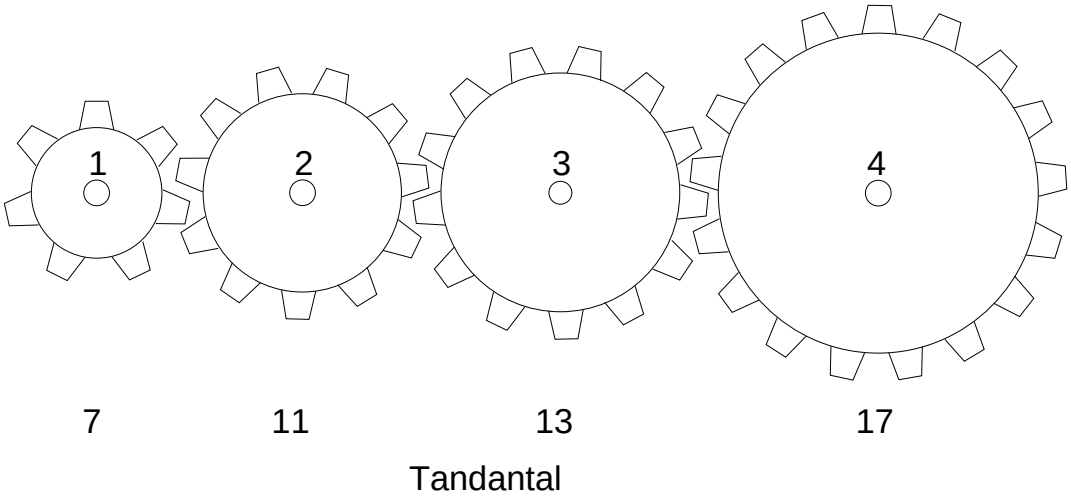


Figure 3.2: *Fysisk model af en Tandhjulsgenerator. Se Eksempel 3.8.3.*

3.8.2 Tandhjulsgeneratoren

En række generatorer kan føres tilbage til følgende princip (the multicyclic vector method-MCV):

Eksempel 3.8.2:
Vi vælger 3 binære tal bestående af 3, 5 og 7 bit og skriver dem på følgende måde:

0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1
1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1
0	1	1	1	1	0	1	0	1	1	1	1	0	1	0	1	1	1

Den første linie indeholder gentagelser af det første binære tal (011), den anden gentagelsen af det andet binære tal (10110) o.s.v.

Vi bestemmer nu en følge af tilfældige binære cifre ved at tage summen af de enkelte søjler modulo 2:

$$1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1$$

Da antallet af bit i de 3 tal er primtal, fås en generator med periodelængden $3 \cdot\ 5 \cdot\ 7 = 105$.
□

Generatorer efter ovennævnte princip er foreslået af svenskeren G. Dahlquist. P. Pohl (1975), H.M. Marqvardsen (1973) og H. Dalgas Christiansen (1975) har studeret og implementeret metoden. Navnet *tandhjulsgenerator* forklares som følger (fig. 2). Vi forestiller os, vi har et tandhjul for hvert af de binære tal. Antallet af tænder på et hjul svarer til antallet af bit

i det tilsvarende binære tal, idet der anbringes et binært ciffer på hver tand. Ud for hvert tandhjul er der en viser, som peger på den "første" tand i hvert hjul. Bittene på de tænder, som viseren peger på, adderes modulo 2 og giver det første tilfældige ciffer. Derefter drejes tandhjulene, som griber ind i hinanden, en tand frem, og det næste tilfældige ciffer genereres, etc.

Generatoren kaldes også *EXOR-generatoren*, idet addition modulo to i datamater lettest udføres ved den logiske operation "exclusive-or". Betragter vi f.eks. de første 4 cifre i ovennævnte eksempel 3.8.2, får vi

(1)	0	1	1	0
(2)	1	0	1	1
EXOR	1	1	0	1
(3)	0	1	1	1
EXOR	1	0	1	0

I praksis anvendes simple tilfældigtalsgeneratorer (basisgeneratorer) til generering af de enkelte binære tal, der således kommer til at bestå af et antal bit, der er lig med den aktuelle generators cyklus gange ordlængden.

Tandhjulsgeneratoren kan umiddelbart implementeres i ethvert talsystem, f.eks. det decimale (eksempel 3.8.3 og afsnit 3.8.3)

Den resulterende generator vil altid være bedre end den bedste undergenerator. Kombinerer vi to generatorer, og giver den ene generator altid værdien 0 (eller 1), fås netop den anden generator som resultat. Har de to basisgeneratorer sandsynligheden P_1, hendholdsvis P_2 for at antage værdien "1", vil EXOR-generatoren have sandsynligheden

$$P_x = P_1(1 - P_2) + P_2(1 - p_1)$$

for at antage værdien "1". Denne værdi vil aldrig være fjernere fra 1/2 end den bedste af P_1 og P_2.

Jo flere basisgeneratorer, der er, desto bedre bliver resultatet naturligvis, og den kan principielt gøres vilkårlig god. De i praksis implementerede generatorer har særdeles gode statistiske egenskaber.

Eksempel 3.8.3: Udtrækning af obligationer
(Jensen, 1967 [19]) Tandhjulsgeneratoren er implementeret på decimal form (addition modulo 10) til udtrækning af obligationer. Mængden af obligationer er i dag så stor, at det er uoverkommeligt for Notarius Publicus at foretage udtrækninger efter tombolametoden. Man lader derfor Notarius Publicus udtrække de cifre, der anvendes som startværdier (48 decimale cifre) på en tandhjulsgenerator med 7, 11, 13 og 17 tænder. Denne generator anvendes til at

generere 251 startcifre til hovedgeneratoren, der består af 7 tandhjul med tandantallene 23, 29, 31, 37, 41, 43 og 47. Denne generator anvendes til udtrækning af obligationer. Den har en periodelængde på 6.339.272.519, idet tandantallene er primtal. □

Eksempel 3.8.4: EXOR-generator til HP 2100-serien
De generatorer, der er omtalt i eksemplerne 3.6.2.4, 3.7.1.5 og 3.7.2.3, er anvendt som basisgeneratorer til en EXOR-generator for Hewlett & Packard datamaskiner. Den har en periodelængde på ca. 10^{13} og fremragende statistiske egenskaber (Jørgensen & al., 1974 [20]). □

Eksempel 3.8.5: RANDXR på UNI-C
Dette er en tandhjulsgenerator (EXOR-generator), der er baseret på 2 basisgeneratorer (Christiansen, 1975):

1. Fibonacci
2. Blandet kongruensmetode:
   $a = 2^7 + 1, \quad b = 907.633.385, \quad c = 2^{32}$

Den blev implementeret på NEUCC og har meget gode statistiske egenskaber.

I Fortran kaldes den på følgende måde:

CALL RANDXR (Z,NR,IVEC).

IVEC er en vektor, der indeholder 3 heltal som startværdier for generatoren. Den ændres automatisk, og skal således kun initieres en gang for alle, eller når generatoren skal genstartes.

Z er en vektor af dimension NR, der ved kaldet returneres med NR tilfældige tal.

I Algol kaldes generatoren på følgende måde:

PROCEDURE RANDXR:

RANDXR (NR,Z(∗),J,K,L)

(J,K og L) svarer til IVEC ovenfor.

REAL PROCEDURE RANDXRl:

Z := RANDXRl(J,K,L)

Herved tildeles Z værdien af et tilfældigt tal.

Ved at generere mange tilfældige tal ad gangen, kan der spares en del tid. □

3.8.3 Tandhjulsgenerator baseret på heltal (L'Ecuyer 1988)

Den følgende metode til generering af tilfældige tal er baseret på addition af tilfældige tal fra simple generatorer. Det er i virkeligheden i den oprindelige tandhjulsgenerator.

Metoden er baseret på følgende to lemmaer. Den er implementeret i afsnit 3.9.

Lemma 1: Lad $W_1, W_2, \ldots, W_n$ være n uafhængige diskrete tilfældighedsvariable, sådan at W_i er UD$(0, 1 \ldots, d-1)$ (uniformt fordelt mellem 0 og d-1), hvor d er et positivt heltal:

$$P\{W_i = n\} = \frac{1}{d}$$

Da vil

$$W = \left(\sum_{j=1}^{n} W_j \right) (\text{mod}) \ d$$

også være UD$(0, 1 \ldots, d-1)$.

Lemma 2: Betragt en familie af n generatorer, hvor (for $j = 1, 2, \ldots, n$) generator j har perioden p_j og udvikles efter

$$s_{j,i} = f_j(S_{j,i-1})$$

Da er perioden p af sekvensen $\{S_i = (S_{1,i}, S_{2,i}, \ldots, S_{j,i}), i = 0, 1, 2, \ldots\}$ den mindste fælles multiplikator af $p_1, p_2, \ldots, p_j$.

3.9 Implementering af algoritmer

Ved programmering af tilfældigtals generatorer må vi være omhyggelige med at sikre, at beregningerne udføres som forudsat i talteorien.

F.eks. i formel (3.7.2):

$$R_i = a \cdot \ R_{i-1} \ (\text{mod } c) \tag{3.11}$$

kan både a og R_{i-1} være store heltal. Ved udførelse af multiplikationen $a \cdot \ R_{i-1}$ må vi derfor sikre os, at ordlængden er tilstrækkelig til at opbevare resultatet. Dette kan ske ved udvidet ordlængde eller ved at dele heltallene op i flere heltal. Tilfældigtals generatorer i biblioteker vil ofte være programmeret i maskinkode.

Ved at vælge c som f.eks. 2^{31} i en datamaskine med ordlængde 32 bit (arbejder i det binære talsystem) udføres modulo-divisionen ved blot kun at gemme de 31 mindst betydende bit i resultatet. Dette er simpelt, hvis beregningerne ikke standses på grund af "overflow". Er det 32. bit et fortegnsbit, må dette elimineres.

For at undgå "overflow" kan vi i stedet for (3.11) anvende følgende ækvivalente, men langsommere rekursionsformel, hvor $U_i = R_i/c$:

$$U_i = a \cdot \ U_{i-1} \ (\text{mod } 1) \tag{3.12}$$

hvor (mod 1) betyder, vi bortkaster heltalsdelen.

For alle tal i sekvensen U_i vil det gælde, at U_i = heltal/c, hvis initialværdien er et heltal. Konstanterne a og c skal vælges således, at det reelle produkt $(a \cdot U_{i-1})$ beregnes uden afrunding.

En anden måde til at undgå afrundingsproblemer illustreres i det følgende.

Ved implementering af Den multiplikative kongruensmetode

$$f(s) = a \cdot \ s \ \text{ MOD } m$$

på en maskine med b bit kan vi kun repræsentere heltal, dvs.

$$m < 2^b - 1.$$

hvorved en multiplikation får "overflow", hvis faktorerne er på mere end b/2 bit.

Dog kan vi omskrive metoden hvis

$$a^2 < m$$

Vi definerer flg.:

$$q = |m/a|$$

$$r = m \text{ MOD } a$$

hvorved m er dekomponeret som

$$m = aq + r,$$

hvor

$$r < a.$$

For

$$0 < s < m$$

får man

$$\begin{aligned} as \text{ MOD } m &= (as - |s/q|m) \text{ MOD } m \\ &= (as - |s/q|(aq + r)) \text{ MOD } m \\ &= (a(s - |s/q|q) - |s/q|r) \text{ MOD } m \\ &= (a(s \text{ MOD } q) - |s/q|r) \text{ MOD } m \end{aligned}$$

Når man beregner

$$as \text{ MOD } m$$

ved hjælp af ovenstående får man alle mellemresultater i intervallet fra -m til +m.

Af ovenstående resultater vælger man den kombinerede generator, således

$$m_j < 2^{b-1}$$

og

$$a_j \leq \sqrt{M_j}$$

For 32-bit computere (som PC'ere, HP-Unix m.fl.) vælger vi 2 MLCG'er:

$$m_1 = 2147483563, \quad a_1 = 40014$$

$$m_2 = 2147483399, \quad a_2 = 40692$$

Disse to MLCG'er har den fremragende egenskab, at m_1 og m_2 er indbyrdes primiske, hvorved maksimal periode opnås.

Tilfældigtalsgeneratoren MLCG

Selve implementeringen af MLCG er skrevet i programmeringssproget PASCAL.

```
VAR
  s1, s2 : LONGINT;
BEGIN
  k := s1 DIV 53668;
  s1 := 40014 * (s1 - k * 53668) - k * 12211;
  IF s1<0 then s1 := s1 + 2146483563;
  k := s2 DIV 52774;
  s2 := 40692 * (s2 - k * 52774) - k * 3791;
  IF s2<0 then s2 := s2 + 2146483399;
  d := s1 - s2;
  IF d<1 THEN d := d + 2147483562;
  RandomX := d * 4.656613E-10;
END;

BEGIN
  s1 := 12345678;
  s2 := 81726354;
END.
```

Eksempel 3.9.1: IBM's RANDU-generator
Generatoren i eksempel 3.7.3 kan implementeres i FORTRAN på IBM-maskiner med følgende program:

```
FUNCTION RAND(I)
I = I x 65539
IF (I.LT.0) I = I + 2147483647 + 1
RAND = I x .4656613E-9
RETURN
END
```

□

Bibliography

[1] Andersson, S.L. (1990): Random Number Generators on Vector Supercomputers and other Advanced Architecture SIAM Review, Vol. 32 (1990) : 2, 221–251.

[2] Andréasson, I. (1973): *Simulationsmetodik.* KTH, Stockholm 1973. 107 pp.

[3] Blum, L. & Blum, M. & Shub, M. (1986): A Simple Unpredictable Pseudo-Random Number Generator. SIAM J. Comput., Vol. 15 (1986) : 2, 364–383.

[4] Bratley, P. & Bennett, L.F. & Schrage, L.E. (1983): *A Guide to Simulation.* Springer–Verlag 1983. 383 pp.

[5] Christiansen, H. Dalgas (1975): Random Number Generators in Several Dimensions. Theory, Tests and Examples. IMSOR, Technical University of Denmark 1975. 66 pp.

[6] Coates, R.F.W. & Janacek, G.J. & Lever, K.V. (1988): Monte Carlo Simulation and Random Number Generation. IEEE Journal of Selected Areas in Communications, Vol. 6 (1988) : 1, 58–66.

[7] Collings, B.J. (1988): Compound Random Number Generators. JASA, Vol. 82 (1988): 398, 525–527.

[8] Conradsen, K. (1976): *En Introduktion til Statistik.* IMSOR, Technical University of Denmark 1976. 484 pp.

[9] Dodge, Y. (1996): A Natural Random Number Generator. International Statistical Review, Vol. (64) : 3, 329–344.

[10] Eichenauer–Herrmann, J. (1992): Inversive Congruential Pseudorandom Numbers: a Tutorial. International Statistical Institute, Vol. 60 (1992) : 2, 167–176.

[11] van Es, A.J. & Gill, r.D. & van Putten, C. (1983): Random Number Generators for a Pocket Calculator. Statistica Neerlandica, Vol. 37 (1983) : 3, pp. 95–102.

[12] Glasserman, P. (1992): Some Guidelines and Guarantees for Common Random Numbers. Management Science, Vol. 38 (1992) : 6, 884–908.

[13] Haas, A. (1987): The Multiple Prime Random Number Generator. ACM Transactions on Mathematical Software, Vol. 13 (1992) : 4, 368–381.

[14] Hammersley, I.M. & Handscomb, D.C. (1964): *Monte Carlo Methods*. London 1964. 178 pp.

[15] Iversen, V.B. (1985): *Data– og Teletrafikteori*. Institut for Telekommunikation, Danmarks Tekniske Højskole 1996. 440 pp.

[16] Isaksson, H. (1958): Generator for Tilfældige Tal. Teleteknik 1958 Nr. 4, pp. 175–186.

[17] Jansson, B. (1966): *Random Number Generators*. Stockholm 1966. Kapitel 8: Pseudo–Random Numbers from Various Statistical Distributions. pp. 170–191.

[18] Jennergren, L.P. (1984): *Discrete–Events Simulations Models in PASCAL/MT+ on a Microcomputer*. Studentlitteratur 1984. 135 pp.

[19] Jensen, Arne (1967): Trækning af Kreditforeningers Kasseobligationer. IMSOR, Danmarks Tekniske Universitet 1967. 61 pp.

[20] Jørgensen, F.G. & Fischer–Madsen, A. & Marqvardsen, H. (1974): Documentation and Test of Exor-Generator. IMSOR, Danmarks Tekniske Universitet 1974. 70 pp.

[21] Kao, Chiang (1989): A Random–Number Generator for Microcomputers. J. Opl. Res. Soc., Vol. 40 (1989) : 7, pp. 687–691.

[22] Kleijnen, J.P.C. & Annink, B. (1992): Pseudorandom Number Generators for Supercomputers and Classical Computers: A Practical Introduction. European Journal of Operations Research, Vol. 63 (1992), 76–85.

[23] Knuth, D.E. (1981): *The Art of Computer Programming*. Vol. 2: Seminumerical Algorithms. Chapter 3: Random Numbers. pp. 1–177. Second edition. Addison–Wesley Publ. Co. 1981.

[24] Kobayashi, H. (1978): Modeling and Analysis: An Introduction to System Performance Evaluation Methodology. Addison–Wesley Publ. Co. 1978. Chapter 4: The Simulation Method. pp. 221–314.

[25] L'Ecuyer, P. (1988): Efficient and Portable Combined Random Number Generators. Communications of the ACM, Vol. 31 (1988) : 6, 742–749 + 770.

[26] Lewis, P.A.W. (1980): Chapter G of the IMSL Library – Generation and Testing of Random Deviates: Simulation. Proceedings of the 1980 Winter Simulation Conference, pp. 357–360.

[27] Maclaren, N.M. (1989): The Generation of Multiple Independent Sequences of Pseudorandom Numbers. Appl. Statistics, Vol. 38 (1989) : 2, 351–359.

[28] Malmberg, A.C. (1971): Matematiske Tabeller: Erlang T. København 1971. 40 pp.

[29] Marqvardsen, H.M. (1974): Generering af Tilfældige Tal. Kapitel 5 i "Skemalægning ved Numerisk Simulation". IMSOR 1974, pp. 91–136.

[30] Marsaglia, G. (1961): Generating Exponential Random Variables. Ann. Math. Stat., Vol. 32 (1961) 899–902.

[31] Marsaglia, G. & Zaman, A. (1990): Towards a Universal Random Number Generator. Statistics & Probability Letters, Vol. 8 (1990) 35–39.

[32] Marsaglia, G. & Zaman, A. (1991): A New Class of Random Number Generators. The Annals of Applied Probability, Vol. 1 (1991) : 3, 462–480.

[33] Newman, T.G. & Odell, P.L.(1971) *The Generation of Random Variates*. London 1971. 88 pp.

[34] Niederreiter, H. (1991): Recent Trends in Random Numbers and Random Vector Generation. Annals of Operations Research, Vol. 31 (1991) 323–346.

[35] Park, S.K. & Miller, K.W. (1988): Random Number Generators: Good Ones are Hard to Find. Communications of the ACM, Vol. 31 (1988) : 10, pp. 1192–1201.

[36] Pohl, P. (1975): MCV – A Fast Pseudo–Random Number Generator with Extremely Good Statistical Properties. Afhandling, KTH, Stockholm 1975. 17 pp.

[37] Rand Corporation (1955): A Million Random Digits with 100.000 Normal Deviates. Glencoe, Illinois 1955.

[38] Ripley, B.D. (1983): Computer Generation of Random Variables: A Tutorial. International Statistical Rewiew, Vol. 51 (1983) 301–319.

[39] Sahai, H. (1980): A supplement to Sowey's bibliography on Random Number Generation and related topics. Biometrical Journal, Vol. 22 (1980) 447–461.

[40] Schmeiser, B.W. (1980): Random Variate Generation: A Survey. Simulation with Discrete Models: A State–of–the–Art Survey. IEEE 1980. pp. 79–104.

[41] Schmeiser, B.W. (1981): Random Variate Generation. 1981 Winter Simulation Conference Proceedings, pp. 227–242.

[42] Sowey, E.R. (1972): A Chronologocal and Classified Bibliography on Random Number Generation and Testing. Int. Stat. Rev., Vol. 40 (1972) : 3, pp. 355–371.

[43] Sowey, E.R. (1978): A Second Classified Bibliography on Random Number Generation and Testing. Int. Stat. Rev., Vol. 46 (1978) : 1, pp. 89–102.

[44] Tippett, L.H.C. (1959): Random Sampling Numbers. Tracts for Computers No. XV. Cambridge 1959.

[45] Tocher, K.D. (1963): The Art of Simulation. London 1963. 184 pp.

[46] Yakowitz, S.J. (1977): Computational Probability and Simulation. Reading, Massachusetts 1977. Kapitel 2, pp. 39–76.

Chapter 4

Tilfældige tal fra statistiske fordelinger

I dette kapitel går vi ud fra, at vi har en ideel tilfældigtals generator til rådighed. Den genererer en følge af pseudotilfældige tal

$$U_1,\ U_2,\ U_3,\ \cdots$$

som er rektangulært fordelt i intervallet (0,1), dvs. *U(0,1)*.

Fordelingen *U(0,1)* er uegnet til direkte at beskrive de fleste stokastiske systemer. Hyppigt optræder der andre fordelinger som f.eks. Normalfordelingen, Γ-fordelingen, eksponentialfordelingen og Poissonfordelingen. I praksis bruger man ofte empiriske fordelinger baseret på observationer.

Det er upraktisk at skulle konstruere og teste tilfældigtals generatorer for alle mulige fordelinger. I det følgende skal vi se, hvorledes en rektangulær fordeling kan transformeres til en vilkårlig fordeling, således at vi kun skal teste en enkelt tilfældigtals generator, der kan indlægges som en standardfunktion i vores regnemaskine.

4.1 Sandsynlighedstransformationer

Den rektangulære (= ligelige) fordeling *U(0,1)* indtager en central rolle for alle statistiske fordelinger, idet en vilkårlig fordeling kan transformeres til en rektangulær fordeling på simpel måde.

Omvendt gælder det, at en rektangulær fordeling kan transformeres til en vilkårlig fordeling. Det er denne sidstnævnte egenskab vi netop udnytter ved genereringen af tilfældige tal med en vilkårlig statistisk fordeling ud fra ligeligt fordelte tal.

Vi betragter en stokastisk variabel T med fordelingsfunktionen $F(t)$ (Fig. 4.1):

$$P\{T \leq t\} = F(t) = \int_{-\infty}^{t} dF(t) \tag{4.1}$$

$$= \int_{-\infty}^{t} f(t)dt$$

hvor $f(t)$ er frekvensfunktionen for T.

Sætning 1: Hvis $F(t)$ er fordelingsfunktionen for den stokastiske variabel T, og vi foretager transformationen:

$$U = F(T) \tag{4.2}$$

så vil U være en stokastisk variabel, der er rektangulært fordelt *U(0,1)*.

Bevis: Da $F(t)$ kun antager værdier i intervallet (0,1), må U også være begrænset til samme interval. Lad den søgte fordelingsfunktion for U hedde $g(u)$, og lad x og y ($> x$) være værdier af U, der svarer til værdierne u og v af T:

$$x = F(u), \;\; y = F(v)$$

Sandsynligheden for, at U ligger i intervallet (x, y) er da lig sandsynligheden for, at T ligger i intervallet (u, v). Vi har derfor:

$$\begin{aligned} G(y) - G(x) &= F(v) - F(u) \\ &= y - x \end{aligned}$$

idet $0 < x < y < 1$.
Erstattes y med $x + \Delta x$, fås

$$\frac{G(x + \Delta x) - G(x)}{\Delta x} = 1$$

Ved grænseovergangen Δx $\rightarrow 0$ fås

$$\frac{dG(x)}{dx} = 1 \qquad 0 < x < 1$$

hvilket viser, at U er rektangulært fordelt *U(0,1)*.

Dette resultat er ikke overraskende, og anvendes f.eks. i forbindelse med fraktildiagrammer.

Da $F(t)$ er monotont voksende, er den inverse funktion af $y = F(t)$ éntydigt bestemt. Den betegnes her med

$$t = F^{-1}(y) \tag{4.3}$$

Der vil gælde følgende:

Sætning 2: Hvis U er en stokastisk variabel, der er rektangulært fordelt U(0,1), så vil:

$$T = F^{-1}(U) \tag{4.4}$$

være en stokastisk variabel med fordelingsfunktionen F(t).

Bevis: Vi ønsker at bevise, at

$$\begin{aligned} P\{F^{-1}(U) \leq t\} &= P\{T \leq t\} \\ &= F(t) \end{aligned} \tag{4.5}$$

Det er givet, at U er rektangulært fordelt:

$$P\{U \leq u\} = u \tag{4.6}$$

Da F(t) er en fordelingsfunktion, er den monotont voksende, og det gælder derfor, at

$$\mathrm{T} \leq \mathrm{t} \iff \mathrm{F(T)} \leq \mathrm{F(t)}$$

Vi har derfor:

$$\begin{aligned} \mathrm{P}\ \{\mathrm{T} \leq \mathrm{t}\} &= \mathrm{P}\{\mathrm{F(T)} \leq \mathrm{F(t)}\} \\ &= \mathrm{p}\{\mathrm{U} \leq \mathrm{F(t)}\ \} \end{aligned}$$

som med (4.6) netop giver det ønskede resultat (4.5).

Forudsat U er rektangulært fordelt $U(0,1)$, er $T = F^{-1}(U)$ altså fordelt som $F(t)$ (Fig. 4.1). Dette kan direkte anvendes til generering af tilfældige tal, der er fordelt som $F(t)$. Vi skal se på en række fordelinger, hvor den inverse funktion kan findes eksplicit.

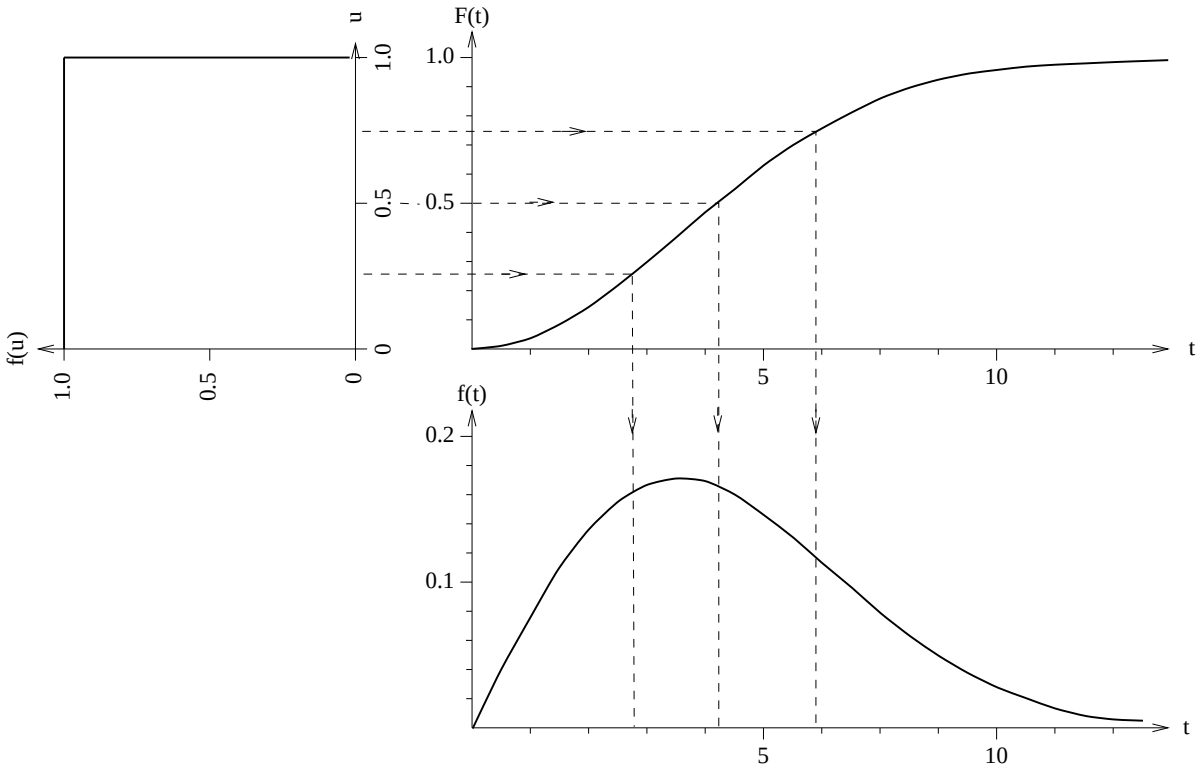


Figure 4.1: *Generering af $F(t)$–fordelte tilfældige observationer ud fra U(0,1)–fordelte tilfældige tal ved invers transformation. Fordelingens kvartiler er indtegnede. ($F(t)$ er det viste eksempel Weibull–fordelt We(2,5)).*

4.2 Invers transformation af kontinuerte fordelinger

Vi skal her anføre de vigtigste eksempler på anvendelse af den inverse fordelingsfunktion.

4.2.1 Rektangulær fordeling U(a,b)

Fordelingsfunktionen U(a,b) for en rektangulært fordelt stokastisk variabel T er givet ved

$$F(t) = \begin{cases} 0 & t \leq a \\ \dfrac{t-a}{b-a} & a < t < b \\ 1 & b \leq t \end{cases} \tag{4.7}$$

Den inverse fordelingsfunktion bliver

$$T = a + (b-a)U \tag{4.8}$$

Er U rektangulært fordelt U(0,1), bliver T rektangulært fordelt U(a,b).

4.2.2 Eksponentialfordelingen Ex(λ)

Eksponentialfordelingen med intensiteten $\lambda > 0$ (middelværdi $\frac{1}{\lambda}$) er givet ved

$$F(t) = 1 - e^{-\lambda t} \qquad , t \geq 0, \lambda > 0 \tag{4.9}$$

Den inverse funktion er givet ved

$$T = -\frac{1}{\lambda}\, ln(1-U)$$

Hvis U er ligeligt fordelt U(0,1), vil 1 - U også være det, og vi får derfor

$$T = -\frac{1}{\lambda}\, ln(U) \tag{4.10}$$

Er U ligeligt fordelt U(0,1), vil T være eksponentialfordelt med intensiteten λ.

Dette er den mest anvendte metode til generering af eksponentialfordelte tal. Metoden effektivitet afhænger af, hvor hurtigt logaritmen beregnes.

4.2.3 Weibull fordelingen We(k,λ)

Weibull fordelingen er givet ved fordelingsfunktionen

$$F(t) = 1 - e^{-\left(\frac{t-\gamma}{\lambda}\right)^k} \quad , \; t > \gamma, \; \lambda > 0, \; k > 0 \tag{4.11}$$

hvor γ blot er en positionsparameter.

Den inverse fordelingsfunktion er givet ved

$$\begin{aligned} T &= \gamma + \lambda \, [-ln(1-U)]^{1/k} \\ T &= \gamma + \lambda \, [-ln(U)]^{1/k} \end{aligned} \tag{4.12}$$

idet 1-U igen erstattes med U. For $\gamma = 0$ og k = 1 fås netop eksponentialfordelingen Ex($1/\lambda$).

4.2.4 Cauchy-fordelingen Ca(α,β)

Den standardiserede Cauchy-fordeling med parametrene (α,β) = (0,1) er givet ved tætheds-funktionen

$$f(t) = \frac{1}{\pi} \cdot \frac{1}{1+t^2} \qquad -\infty < t < +\infty \tag{4.13}$$

Cauchy-fordelingen Ca(α,β) med positionsparameteren α og skalaparameteren β fås ved transformationen

$$y = \alpha + \beta t \tag{4.14}$$

Fordelingen er symmetrisk om α, og summen af en række Cauchy-fordelinger vil igen være en Cauchy-fordeling (reproduktivitetssætningen, jf. Conradsen 1976).

Den kumulerede fordelingsfunktion bliver:

$$\begin{aligned} F(t) &= \int_{-\infty}^{t} \frac{1}{\pi} \cdot \frac{1}{1+t^2} dt \\ &= \frac{1}{\pi} \, tan^{-1}(t) + \frac{\pi}{2} \end{aligned} \tag{4.15}$$

hvor $\tan^{-1}$ = arctan er den inverse tangensfunktion.

Den inverse fordelingsfunktion bliver derfor enkel:

$$T = tan[\pi(U - 1/2)] \tag{4.16}$$

hvor U er et tilfældigt tal U(0,1).

Cauchy-fordelingen er specielt kendt, fordi dens middelværdi og dermed varians ikke eksisterer. Der kan ved anvendelse af (4.16) opstå numeriske problemer, når det tilfældige tal ligger i nærheden af 0 eller 1.

Cauchy-fordelingen kan anvendes som indhyllingskurve ved generering af gammafordelte variable, idet dens tæthedsfunktion ved multiplikation med en konstant vil ligge helt over Γ-fordelingens tæthedsfunktion.

4.2.5 Laplace-fordelingen La(α,β)

Den standardiserede Laplace-fordeling La(0,1) er givet ved tæthedsfunktionen

$$f(t) = 1/2e^{-|t|} \quad , \quad -\infty < t < +\infty \, . \tag{4.17}$$

Den inverse transformation kan umiddelbart anvendes. Ligger det tilfældige tal U(0,1) i intervallet (0,1/2), får vi en negativ observation, der bestemmes på samme måde som ved eksponentialfordelingen (afsnit 4.2.2), idet det tilfældige tal multipliceres med 2, så det bliver U(0,1). Ligger det tilfældige tal i intervallet (1/2,1), er observationen positiv, og før den inverse eksponentialtransformation udføres, multipliceres det tilfældige tal med 2, og der subtraheres 1.

I praksis vil man anvende to tilfældige tal: ét til fortegnsbestemmelse og ét til eksponentialtransformationen.

4.2.6 Pareto-fordelingen Par(k,β)

Fordelingsfunktionen for Pareto-fordelingen med skalaparameteren β er givet ved (Conradsen 1976):

$$F(t) = 1 - \left(\frac{\beta}{t}\right)^k \quad , \quad t \geq \beta \tag{4.18}$$

Den inverse transformation kan umiddelbart anvendes.

$$T = \beta(1-U)^{-1/k} \tag{4.19}$$

4.2.7 Den logistiske fordeling L(α,β)

Fordelingsfunktionen for den logistiske fordeling med positionsparameter α og skalaparameter β er givet ved

$$F(t) = \left(1 + e^{-\frac{t-\alpha}{\beta}}\right)^{-1} \tag{4.20}$$

Den inverse transformation giver umiddelbart:

$$T = \alpha \ - \beta \cdot ln\left(\frac{1}{U} - 1\right) \tag{4.21}$$

4.3 Invers Transformation af diskrete fordelinger

Den generelle metode til generering af diskrete stokastiske variable svarer helt til metoden for kontinuerte variable. Vi skal se på de mest betydningsfulde eksempler.

4.3.1 Bernoulli fordelingen B(1,p)

Fordelingsfunktionen for Bernoulli fordelingen er givet ved :

$$F(t) = \begin{cases} 0 & t < 0 \\ 1-p & 0 \leq t < 1 \\ 1 & t \geq 1 \end{cases} \tag{4.22}$$

Den inverse fordelingsfunktion bliver

$$T = \begin{cases} 0 & U < 1-p \\ 1 & U \geq 1-p \end{cases}$$

eller idet U erstattes med 1 - U

$$T = \begin{cases} 0 & U > p \\ 1 & U \leq p \end{cases} \tag{4.23}$$

Eksempel: møntkast En stokastisk variabel til generering af møntkast fås således ved at vælge p = 0,5.

4.3.2 Ligefordeling på {0,1,$\cdots$,n} UD(n)

Fordelingsfunktionen for denne er givet ved

$$F(t) = \begin{cases} 0 & t < 0 \\ \frac{t+1}{n+1} & t = 0, 1, \ldots, n \\ 1 & t \geq n \end{cases} \tag{4.24}$$

T antager værdien t, når

$$\frac{t}{n+1} \leq \ F(t) < \frac{t+1}{n+1}$$

Erstattes F(t) derfor med U, fås

$$\frac{t}{n+1} \leq U < \frac{t+1}{n+1}$$

For en given værdi af U vil T derfor antage en heltallig værdi bestemt ved

$$(n+1)U - 1 < T \leq (n+1)U$$

eller

$$T = Int[(n+1)U] \tag{4.25}$$

idet Int angiver heltalsdelen af argumentet.

Eksempel: terningekast

Udfaldet af et terningekast er givet ved en ligefordeling på $\{1, 2, \cdots, 6\}$. Vi får derfor

$$T = 1 + Int[6 \cdot U],$$

idet n + 1 erstattes af n og fordelingen starter i 1 (Fig. 4.2).

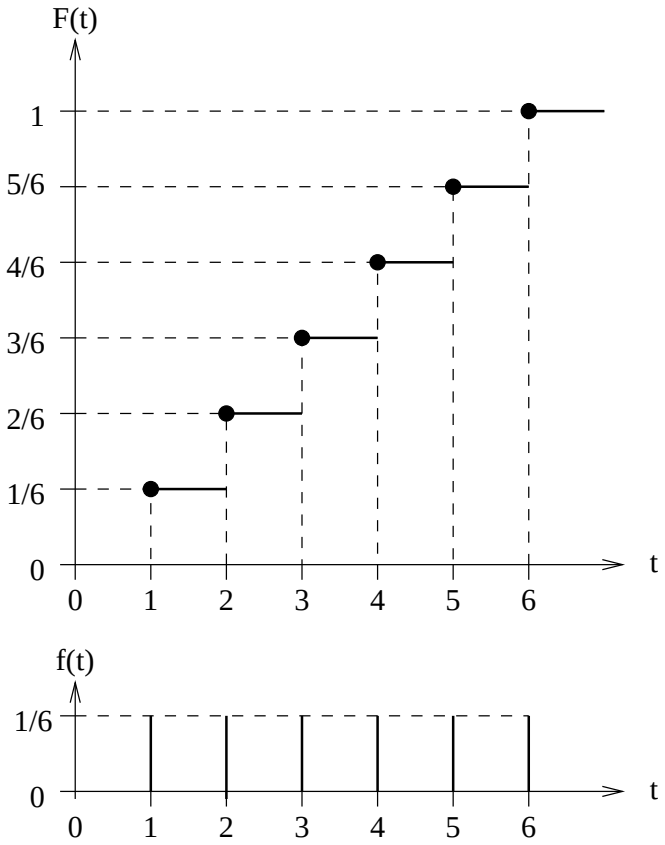


Figure 4.2: *Simulation af terningekast ved invers transformation af en ligefordeling på* $\{1, 2, \ldots, 6\}$.

4.3.3 Geometrisk fordeling NB(1,p)

Fordelingsfunktionen for den geometriske fordeling, der er en speciel negativ binomialfordeling, er givet ved

$$F(t) = 1 - (1-p)^{t+1} \qquad t = 0,\ 1,\ 2,\ \cdots \tag{4.26}$$

T antager værdien t når

$$1 - (1-p)^t \leq\ F(t) < 1 - (1-p)^{t+1}$$

Den inverse transformation bliver da som følger:

$$\begin{array}{lcccl} 1-(1-p)^t & \leq & U & < & 1-(1-p)^{t+1} \\ (1-p)^{t+1} & < & 1-U & \leq & (1-p)^t \\ (t+1)ln(1-p) & < & ln(1-U) & \leq & t\ ln(1-p) \end{array}$$

hvilket, da ln(1 - p) < 0, resulterer i

$$1 > \frac{ln(1-U)}{ln(1-p)} - t \geq\ 0$$

Hvis U er rektangulært fordelt U(0,1), vil T derfor være geometrisk fordelt, hvis

$$T = Int\left[\frac{ln\ U}{ln(1-p)}\right] \tag{4.27}$$

idet (1-U) erstattes med U.

Geometrisk fordelte variable kan genereres ud fra Bernoulli fordelte variable. Dette vil kræve flere tilfældige tal pr. observation. Ovennævnte metode kræver kun ét tilfældigt tal pr. observation, men til gengæld to logaritmeberegninger.

Den geometriske fordeling er den eneste diskrete fordeling uden hukommelse og er den diskrete parallel til eksponentialfordelingen, som er den eneste kontinuerte fordeling uden hukommelse.

Ovennævnte metode kunne også udledes ved at diskretisere eksponentialfordelingen. Idet vi betragter en heltallig værdi af t findes:

$$\begin{aligned} P\{T > t+1 \mid T > t\} &= \frac{P\{T > t+1\}}{P\{T > t\}} \\ &= \frac{e^{-\lambda(t+1)}}{e^{-\lambda t}} \\ &= e^{-\lambda} \end{aligned}$$

Denne sandsynlighed svarer til parameteren 1-p i den geometriske fordeling:

$$\begin{aligned} 1-p &= e^{-\lambda} \\ \lambda &= -ln(1-p) \end{aligned}$$

Fra afsnit 4.2.2 har vi en eksponentielfordelt variabel givet ved

$$T = -\frac{1}{\lambda}\, ln(U)$$

Ved trunkering til en heltallig værdi fås følgelig

$$T = Int\left[\frac{ln(U)}{ln(1-p)}\right]$$

hvilket netop er (4.27)

4.4 Numerisk invers transformation

Den teknik, der anvendes ved invers transformation, kan altid anvendes numerisk på datamat for en vilkårlig fordeling.

I praksis vil der i mange tilfælde ikke foreligge et analytisk udtryk for sandsynligheden, som kan være givet ved empiriske observationer i form af histogram.

I andre tilfælde kan den inverse fordelingsfunktion ikke findes eksplicit (e.g. normalfordelingen). Numerisk kan man imidlertid altid eksakt beregne enkeltværdier af alle fordelingsfunktioner. Dette er imidlertid tidskrævende og benyttes sjældent.

Selv i de tilfælde, hvor den inverse funktion er kendt analytisk (e.g. normalfordelingen). Numerisk kan man imidlertidig altid eksakt beregne enkeltværdier af alle fordelingsfunktioner. Dette er imidlertid tidskrævende og benyttes sjældent.

Selv i de tilfælde, hvor den inverse funktion er kendt analytisk (e.g. eksponentialfordelingen), bruger man ofte approksimative numeriske metoder for af spare beregningstid.

Blandt de anvendte numeriske metoder kan man skelne mellem eksakte og approksimative metoder. Approksimationer kan foretages til en af følgende funktioner:

1. fordelingens tæthedsfunktion,
2. den kumulerede fordelingsfunktion eller
3. den inverse kumulerede fordelingsfunktion.

Den hyppigst anvendte approksimation består i at tilnærme den kumulerede fordelingsfunktion med rette liniestykker, svarende til at tæthedsfunktionen bliver konstant i delintervaller. Denne approksimation svarer til en *kompositionsmetode* (afsnit 4.6), hvor den kumulerede fordelingsfunktion skrives som en vægtet sum af rektangulære fordelinger (Fig. 4.3).

I praksis implementeres metoden ved at oprette en tabel, hvor man en gang for alle beregner de punktpar, der beskriver approksimationen. For et givet tilfældigt tal U(0,1) går man ind i tabellen og finder den tilsvarende værdi af den søgte stokastiske variabel.

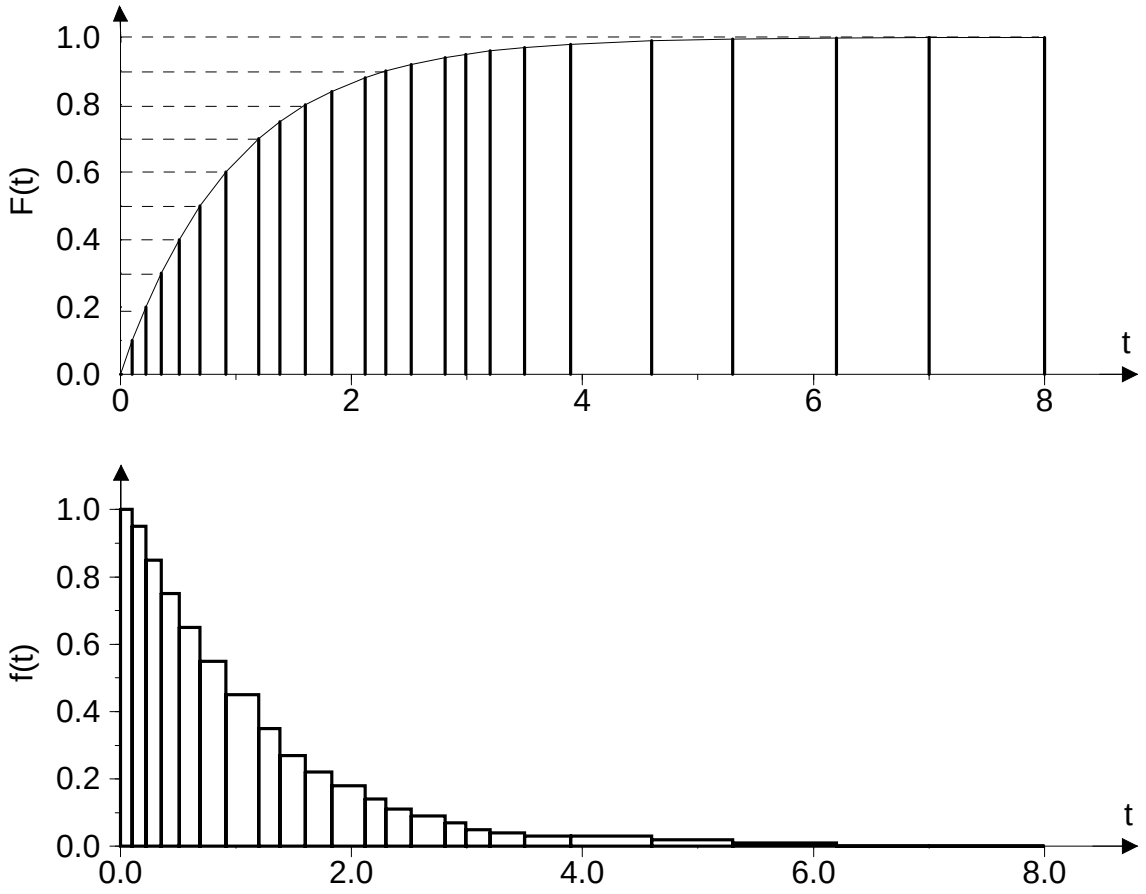


Figure 4.3: *Eksempel på hvorledes en kontinuert fordeling (eksponentialfordelingen) approksimeres med en vægtet sum af 23 rektangulære fordelinger (GPSS, se Tab. 4.1, Kompositionsmetoden, Afsn. 4.6).*

Det er en fordel, hvis intervallet (0,1) er inddelt i lige store delintervaller (fraktiler), idet det søgte delinterval da umiddelbart kan beregnes uden anvendelse af søgeprocedurer.

Hvis den betragtede stokastiske variabel er diskret, kan man evt. anvende én klasse for hver værdi. Er den stokastiske variabel kontinuert, vil man normalt foretage lineær interpolation inde i det enkelte delinterval.

4.4.1 Eksponentialfordelingen - fraktiltransformation

En metode til generering af eksponentialfordelte tal består i at oprette en tabel med 256 (=2^8) klasser og approksimere eksponentialfordelingen med en vægtet sum af 256 rektangulære fordelinger.

De 256 værdier ligger ækvidistant fordelt i intervallet (0,1). For at få en eksponentialfordelt observation genereres et tilfældigt tal, fra hvilket man bruger de 8 mest betydende bit som direkte indgang i tabellen.

Uanset hvor mange eksponentialfordelte tilfældige observationer, man har brug for (e.g. 100.000), skal man med denne metode kun benytte logaritmefunktionen 256 gange. Dette vil i mange tilfælde medføre store tidsbesparelser.

Principielt kan man med n klasser få mindst de (n-1) første momenter af fordelingen korrekt. Normalt vil man kun sørge for, at middelværdien er korrekt. Denne kan skrives som

$$M_1 = E\{T\} = \int_{-\infty}^{+\infty} t \cdot f(t)dt = \int_{-\infty}^{+\infty} (1 - F(t))dt \tag{4.28}$$

hvorfor man f.eks. i hvert delinterval skal sørge for, at dette integrale giver det korrekte bidrag. Samtidigt vil alle højereordens momenter da blive approksimativt korrekte.

4.4.2 Eksponentialfordelingen - GPSS

En af de store mangler ved simulationssproget GPSS er, at man ikke har adgang til matematiske funktioner og derfor f.eks. ikke kan generere eksponentialfordelte tilfældige tal ved den inverse transformation (afsnit 4.2.2)

Alle kontinuerte funktioner approksimeres i GPSS med rette liniestykker, og punktparrene må indlæses som data. Eksponentialfordelingen er den vigtigste fordeling ved diskret simulation og beskrives med 24 punktpar, mellem hvilke der foretages lineær interpolation.

For at opnå den bedst mulige approksimation med kun 24 punkter, er disse ikke anbragt ækvidistant i intervallet (0,1), men med størst tæthed dér, hvor den inverse eksponentialfordeling vokser hurtigst. Endvidere er den største opnåelige værdi begrænset til 8 gange middelværdien. De numeriske værdier for den anvendte approksimation er vist i Tab. 4.1.

De resulterende observationer får teoretisk middelværdien

$$M_1 = 0.9956$$

og formfaktoren

$$\varepsilon = \frac{M_2}{{M_1}^2} = \frac{1.98099}{0.9956^2} = 1.9985$$

hvilket ligger tæt på de teoretiske værdier M_1 og $\varepsilon = 2$.

I GPSS sker der yderligere en trunkering af den beregnede observation, idet der arbejdes med heltal. Man bør derfor kun arbejde med middelværdier på mindst 30. Dette medfører imidlertid andre problemer.

Da intervallet (0,1) ikke er opdelt i ækvidistante punkter, kræves der en del sorteringsarbejde for at finde den korrekte klasse, selv om de 5 første klasser omfatter halvdelen af alle observationer.

4.4.3 Empiriske fordelinger

Fordelinger, der opnås via observationer, kan behandles på samme måde som eksponentialfordelingen (og alle andre fordelinger) i GPSS.

Normalt vil de observerede værdier være givet i form af et histogram med et begrænset antal klasser.

Er den observerede stokastiske variabel diskret, kan hver klasse svare til en mulig observation. Er den stokastiske variabel kontinuert, vil man normalt anvende en rektangulær fordeling (lineær interpolation i den kumulerede fordelingsfunktion) i hver klasse.

Lad fordelingsfunktionen være givet ved rette liniestykker gennem punkterne (f~fraktil):

$$(t_0, f_0), (t_1, f_1), \ldots, (t_i, f_i), \ldots, (t_n, f_n) \tag{4.29}$$

hvor $f_0 = 0$ og $f_n = 1$.

f_i er den kumulerede relative hyppighed af observationer i klasserne 1,2,...,i. Klasse i omfatter intervallet $t_{i-1} \leq t < t_i$.

Den kumulerede fordelingsfunktion er da givet ved:

$$F(t) = f_i + \frac{(f_{i+1} - f_i)}{(t_{i+1} - t_i)} (t - t_i) \qquad t_i \leq t < t_{i+1} \tag{4.30}$$

og frekvensfunktionen er givet ved:

$$f(t) = \frac{(f_{i+1} - f_i)}{(t_{i+1} - i)} \qquad t_i \leq t < t_{i+1} \tag{4.31}$$

Den inverse fordelingsfunktion bliver:

$$F^{-1}(f) = t_i + \frac{(t_{i+1} - t_i)}{(f_{i+1} - f_i)} (f - f_i) \qquad f_i \leq f < f_{i+1} \tag{4.32}$$

i	f_i	t_i (GPSS)	$-\ln(1-f_i)$
1	0	0	0
2	0.1	0.104	0.1054
3	0.2	0.222	0.2231
4	0.3	0.355	0.3567
5	0.4	0.509	0.5108
6	0.5	0.69	0.6931
7	0.6	0.915	0.9163
8	0.7	1.2	1.2040
9	0.75	1.38	1.3863
10	0.80	1.6	1.6094
11	0.84	1.83	1.8326
12	0.88	2.12	2.1203
13	0.90	2.3	2.3026
14	0.92	2.52	2.5257
15	0.94	2.81	2.8134
16	0.95	2.99	2.9957
17	0.96	3.2	3.2189
18	0.97	3.5	3.5066
19	0.98	3.9	3.9120
20	0.99	4.6	4.6052
21	0.995	5.3	5.2983
22	0.998	6.2	6.2146
23	0.999	7	6.9078
24	0.9997	8	8.1117

Table 4.1: *I GPSS approksimeres fordelingsfunktionen for eksponentialfordelingen med 24 rette liniestykker, idet man søger at begrænse antallet af decimaler. Fraktilerne f_i vælges, og den inverse funktionsværdi t_i beregnes, så at middelværdien (4.28) bliver korrekt. Derved bliver de beregnede værdier t_i generelt mindre end $-\ln(1-f_i)$ på grund af eksponentialfordelingens konkavitet (jf. Fig. 4.3).*

Genererer vi derfor et ligeligt fordelt tilfældigt tal U ∈ U(0,1), vil vi få en observation fra den empiriske fordeling ved den inverse transformation:

$$T = t_i + \frac{(t_{i+1} - t_i)}{(f_{i+1} - f_i)} (U - f_i) \qquad f_i \leq U < f_{i+1} \tag{4.33}$$

Ved diskrete fordelinger undgår man den lineære interpolation.

Er der mange klasser, bliver det tidskrævende at fastlægge klassen ($f_i \leq U < f_{i+1}$) ved en søgningsalgoritme.

4.5 Afvisningsmetoden

(von Neumann's rejection method, acceptance-rejection technique).

I de tilfælde, hvor det analytisk ikke er muligt at finde den inverse fordelingsfunktion t = $F^{-1}(y)$, er det ofte simpelt at udregne frekvensfunktionen f(t) for en given værdi af t.

Dette udnyttes i von Neumann's afvisningsmetode, der er baseret på samme princip som det, der anvendes til Monte Carlo beregning af bestemte integraler.

En fordelings frekvensfunktion har som bekendt den egenskab, at arealet f(t)dt er lig med sandsynligheden for at få en observation i intervallet (t,t+dt).

Genererer vi derfor punktpar, der er ligeligt fordelt på arealet under frekvensfunktionen, så vil disse punktpars abscisser være fordelt som den givne frekvensfunktion.

Hvis vi omvendt genererer en observation T fra fordelingen f(t), så vil {T, U · f(T)}, hvor U er et tilfældigt tal, være ligeligt fordelt på arealet under f(t).

For at anvende afvisningsmetoden til generering af observationer fra en fordeling med frekvensfunktionen f(t) må man først finde en indhyldningskurve h(t,β), for hvilken det gælder, at

$$f(t) \leq M \cdot h(t, \beta) \qquad -\infty < t < +\infty \tag{4.34}$$

hvor M er en konstant. Indhyllingsfordelingen $h(t, \beta)$ vil normalt være en frekvensfunktion for en anden fordeling, fra hvilken det er let at generere observationer, og vi vil da have $M > 1$.

Den generelle fremgangsmåde bliver da som følger:

1. Generer en tilfældig observation U_1 fra h(t,β).
2. Generer et tilfældigt tal $U_2 \in$ U(0,1).

3. Beregn
$$T(U_1) = \frac{f(U_1)}{M \cdot \ h(U_1, \beta)} \leq \ 1 \tag{4.35}$$

4. Hvis $U_2 \leq T(U_1)$, så er U_1 en tilfældig observation fra fordelingen f(t).

5. Hvis $U_2 > T(U_1)$, forkastes forsøget (deraf metodens navn), og der startes forfra med punkt 1.

Der genereres altså et tilfældigt punkt under h(t,β). Hvis punktet yderligere ligger under kurven f(t), accepteres t som en tilfældig observation fra f(t), ellers forkastes forsøget.

Det ses, at hvis h(t,β) også er en frekvensfunktion, så skal der i gennemsnit foretages M forsøg, før vi accepterer en observation. Det er derfor ønskeligt, at M ligger så tæt på 1 som muligt, dvs. de to frekvensfunktioner f(t) og h(t,β) skal have samme form.

Det fremgår, at metoden principielt er eksakt.

4.5.1 Rektangulær indhyldningskurve h(t,β) $\in$ U(a,b)

Lad frekvensfunktionen f(t) være begrænset til intervallet $a \leq \ t \leq \ b$, og lad den største værdi, den antager, være lig med c (Fig. 4.4).

Som h(t,β) vælges en rektangulær fordeling U(a,b), og den mindst mulige værdi af M bliver
$$M = \frac{c}{1/(b-a)} = c\,(b-a) \tag{4.36}$$

Vi genererer da en observation U_1 fra U(a,b) og et tilfældigt tal U_2. Dernæst beregnes:
$$T(U_1) = \frac{f(U_1)}{c} \tag{4.37}$$

Hvis $U_2 \leq T(U_1)$, så er U_1 accepteret, ellers forkastes U_1, og der gøres et nyt forsøg.

Vi genererer altså et punkt (U_1, U_2), der er tilfældigt placeret i rektanglet, og accepterer punktet, hvis det ligger under frekvensfunktionen f(t).

Det antal punktpar, der i gennemsnit er nødvendige for at generere en observation fra f(t), er $M = c(b-a)$, og der bruges hver gang to tilfældige tal.

Eksempel 4.5.1:
: Vi betragter frekvensfunktionen
$$f(t) = 2t \qquad 0 \leq \ t \leq \ 1 \tag{4.38}$$

Vi har umiddelbart a = 0, b = 1 og c = 2, og dermed

$$T(U_1) = U_1$$

Trækker vi derfor to tilfældige tal U_1 og U_2, bliver U_1 accepteret som observation, hvis

$$U_2 \leq \ T(U_1) = U_1 \tag{4.39}$$

dvs. U_1 skal være det største af to tilfældige tal, og metoden er ækvivalent med at vælge det største af to tilfældige tal. Vi bruger således to tilfældige tal og en logisk sammenligning pr. observation.

Med den inverse transformation får vi:

$$F(t) \ = \ t^2 \,, \tag{4.40}$$

$$T \ = \ \sqrt{U} \,. \tag{4.41}$$

Denne metode kræver altså ét tilfældigt tal og en kvadratrodsuddragning pr. observation. □

4.5.2 Betafordelingen Be(α,β)

Betafordelingen er givet ved frekvensfunktionen:

$$f(t) = \frac{\Gamma(\alpha \ + \beta)}{\Gamma(\alpha) \cdot \ \Gamma(\beta)} \ t^{\alpha-1} \ (1-t)^{\beta-1} \qquad 0 \leq \ t \leq \ 1 \tag{4.42}$$

Anvendelse af afvisningsmetoden vil derfor i den umiddelbare form forudsætte, at gammafunktionen Γ(x) kan beregnes. På større datamater findes Γ(x) normalt som standardfunktion.

Betafordelingen er begrænset til intervallet (a,b) = (0,1) , og den antager en maksimalværdi

$$c = \frac{df(t)}{dt}\bigg|_{t=0} = \frac{\alpha - 1}{\alpha + \beta - 2} \tag{4.43}$$

Vi kan derfor ligesom i afsnit 4.5.1 bruge et rektangel som indhyllingskurve (Fig. 4.4).

En anden metode anvender en Normalfordeling

$$N\left(\frac{\alpha-1}{\alpha+\beta-2}, \frac{2}{\sqrt{\alpha+\beta-2}}\right)$$

som indhyllingskurve og anvender kun observationer i (0,1).

Er formparametrene α og β heltallige, vil det α'te mindste af $(\alpha+\beta-1)$ tilfældige tal U(0,1) være fordelt som Be(α,β).

Endelig skal der anføres en afvisningsmetode, der undgår beregning af gammafunktionen:

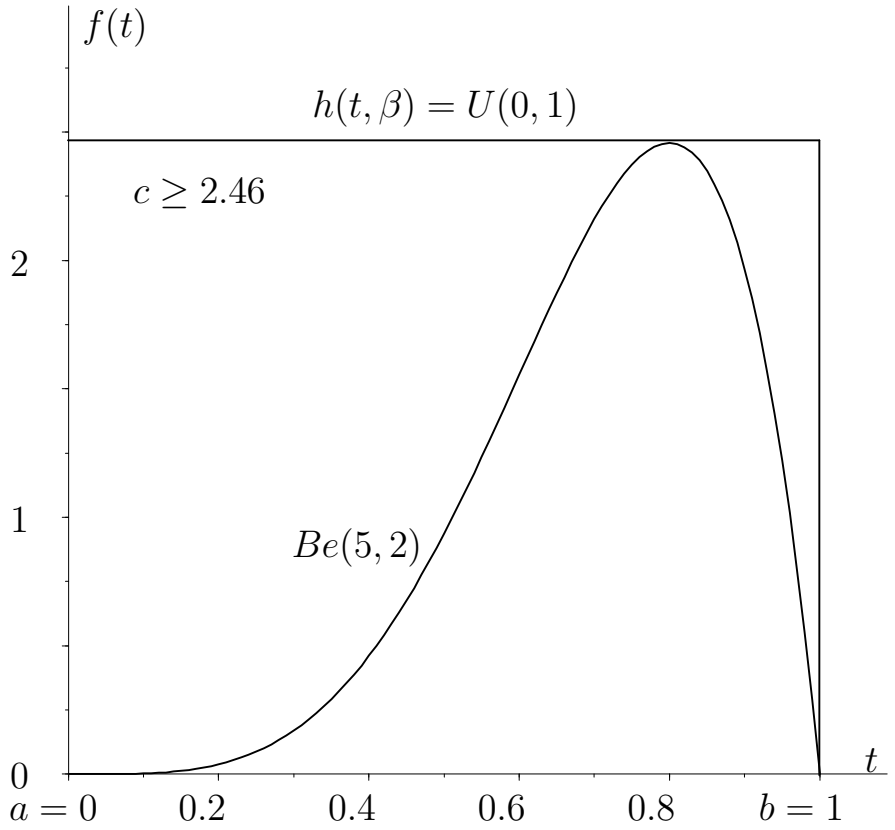


Figure 4.4: *Generering af beta–fordelte {Be(5,2)} tilfældige observationer ved afvisningsmetoden med en rektangulær indhyllingskurve. En observation kræver i gennemsnit 4.92 tilfældige tal.*

1. Generer to tilfældige tal U_1 og U_2 ,
2. Beregn $S_1 = U_1^{1/\alpha}$ og $S_2 = U_2^{1/\beta}$,
3. Hvis $S_1 + S_2 \leq 1$, vil
$$\frac{S_1}{S_1 + S_2} \text{ være Be}(\alpha, \beta)\text{-fordelt}$$
4. Hvis $S_1 + S_2 > 1$, startes forfra med 1.

Metoden er udviklet af Johnk og kan danne grundlaget for generering af Γ-fordelte observationer (afsnit 4.5.3). Metoden virker umiddelbart meget tiltalende. Den eneste ulempe er, at for store parameterværdier skal der foretages et stort antal forsøg, før $S_1 + S_2 \leq 1$. Det forventede antal mislykkede forsøg før en accepteret observation er for en række værdier af parametre (α, β) (der er symmetri i parametrene):

$(\alpha, \beta) = (1,1)$:	1 forsøg	$(\alpha, \beta) = (3,3)$:	30 forsøg
$(\alpha, \beta) = (3,1)$:	4 forsøg	$(\alpha, \beta) = (5,3)$:	56 forsøg
$(\alpha, \beta) = (5,1)$:	6 forsøg	$(\alpha, \beta) = (5,5)$:	252 forsøg

En praktisk anvendelse af metoden kræver således, at mindst den ene parameter er lille. Et bevis for metoden kan f.eks. findes i (Yakowitz 1977).

4.5.3 Gammafordelingen G(k,β)

Denne fordeling kaldes undertiden for Pearson type-III-fordelingen. Gammafordelingen, der betegnes med G(k,β), har tæthedsfunktionen

$$f(t) = \frac{1}{\Gamma(k)} \cdot \left(\frac{t-\gamma}{\beta}\right)^{k-1} \frac{1}{\beta} e^{-\frac{t-\gamma}{\beta}} \quad , \qquad t \geq \gamma \tag{4.44}$$

hvor γ blot er en positionsparameter.

Den standardiserede gammafordeling fås for $\gamma = 0$ og $\beta = 1$, G(k,1):

$$f(t) = \frac{1}{\Gamma(k)} t^{k-1} \cdot e^{-t} \quad , \qquad t \geq 0 \tag{4.45}$$

Det er tilstrækkeligt at betragte den standardiserede gammafordeling, idet den generelle fås ved transformationen

$$y = t \cdot \beta + \gamma \tag{4.46}$$

γ er således blot en skalaparameter. Den inverse fordelingsfunktion kan ikke udledes.

Fra statistikken (e.g. Conradsen 1976, reproduktivitetssætningen) ved vi, at summen af de to standardiserede Γ-fordelinger resulterer i en ny standardiseret Γ-fordeling:

$$G(k,1) + G(l,1) = G(k+l,1) \tag{4.47}$$

G(1,β) er en eksponentialfordeling, som vi behandler flere steder.

For heltallige værdier af k er G(k,β) en Erlang-k-fordeling (afsnit 4.6.3), som jo er en sum af k eksponentialfordelinger.

G $\left(\frac{n}{2}, 2\right)$ er en chi-i-anden fordeling med n frihedsgrader (afsnit 4.7.3).

Det fremgår således, at Γ-fordelingen indtager en central plads. Kan vi generere observationer fra denne, kan vi også generere observationer fra flere andre vigtige fordelinger.

De fleste metoder til generering af G(k,β)-fordelte observationer er baseret på afvisningsmetoder. For store værdier af k kan dog bruge en Normalfordelingstilnærmelse med samme middelværdi (kβ) og varians ($k\beta^2$).

En interessant metode er baseret på beta-fordelingen og eksponentialfordelingen. Hvis W er beta-fordelt Be(k,1-k), og C er eksponentialfordelt Ex(1), så vil

$$T = W \cdot C \tag{4.48}$$

være G(k,1). Mere generelt vil det gælde, at hvis W er Be(n,k), og C er G(n+k,1), så er T G(n,1) svarende til n+k = 1 i ovennævnte tilfælde.

Beta-fordelingen blev behandlet i foregående afsnit 4.5.2, og det ses, at vi har en eksakt metode til generering af gamma-fordelte observationer. Metoden er kun effektiv for små parameterværdier (jf. beta-fordelingen). Men vi kan spalte problemet op i to dele: generering af en Erlang-k-fordelt størrelse (afsnit 4.6.3) og generering af en G(α,1)-fordelt størrelse, hvor k er heltallig, og $0 \leq \alpha < 1$ (ovennævnte metode). I denne form må denne metode anbefales til generering af observationer fra gamma-fordelingen.

Der er fremsat forslag til flere forskellige metoder til generering af gamma-fordelte observationer ved direkte anvendelse af afvisningsmetoden (Atkinson 1977, Tadikamalla 1978). De adskiller sig fra hinanden ved valg af indhyllingsfordelingen h(t,β). Vi nævner de vigtigste metoder:

1. h(t,β) = Erlang-fordeling (Tadikamalla 1978). Denne metode kan anvendes for k > 0. Erlangfordelingen er gennemgået i afsnit 4.6.3, hvoraf det ses, at det antal tilfældige tal, der skal anvendes, er proportionalt med k.

2. h(t,β) = Cauchy-fordelingen (Ahrens & Dieter 1972). Cauchy-fordelingen kan anvendes ved invers transformation (afsnit 4.2). Metodens hurtighed er stort set uafhængig af k's størrelse og således velegnet for store parameterværdier. Der kan imidlertid opstå

numeriske problemer ved beregning af tangens i nærheden af $\pm\frac{\pi}{2}$ (dette indgår i den inverse Cauchy-fordeling) (afsnit 4.2.4).

3. h(t,β) består dels af en Normalfordeling, der dækker gammafordelingens top, dels af en eksponentialfordeling, der dækker halen. Metoden kan kun anvendes for k > 2,53. Beregningstiden aftager for voksende k, og metoden er den hurtigste for større k. Til gengæld er den mere kompleks end de to andre.

4.5.4 Eksponentialfordelingen

Vi slutter dette afsnit af med en afvisningsmetode til generering af eksponentialfordelte observationer. Metoden, der er udledt af Von Neumann (se e.g. Tocher 1963, p.35), er illustreret i Fig. 4.5 og kræver ud over en tilfældigtals generator kun logiske funktioner.

Metoden kan effektiviseres ved oprettelse af en tabel (Marsaglia 1961). Metoden er eksakt, og man undgår logaritmeberegningen ved den inverse transformation. Normalt vil man dog anvende den inverse transformation.

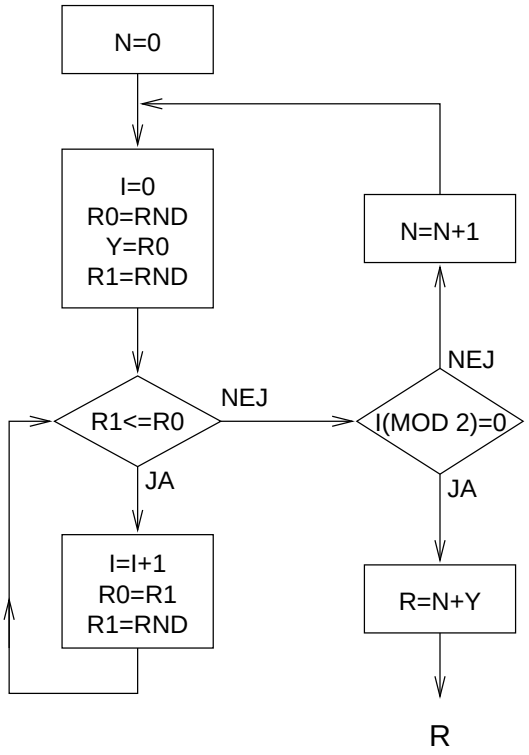


Figure 4.5: *Neumann's afvisningsmetode til generering af eksponentialfordelte observationer ud fra ligeligt fordelte observationer RND.*

4.6 Kompositionsmetoder = blandingsmetoder

Disse metoder anvendes i tilfælde, hvor en ønsket fordeling ikke kan behandles på simpel måde, men kan omskrives til en vægtet sum af fordelinger, som er lette at behandle. Vi har allerede anvendt denne metode i form af en vægtet sum af rektangulære fordelinger som en approksimation til en kontinuert fordeling (se e.g. Marsaglia 1961).

Den ønskede fordeling udtrykkes som følger:

$$F(t) = \sum_{i=1}^{n} P_i \cdot F_i(t) \tag{4.49}$$

hvor vægtfaktorerne er ikke-negative og har summen én:

$$\sum_{i=1}^{n} P_i = 1 \quad , \quad P_i \geq 0 \quad \text{for alle i} \tag{4.50}$$

Ofte vil summen reelt være spaltet op i 2 led:

$$F(t) = \sum_{i=1}^{n-1} P_i \cdot F_i(t) + P_n \cdot F_n(t) \tag{4.49 a}$$

hvor det første led består af fordelinger, der er simple at behandle (e.g. rektangulære), medens den sidste fordeling F_n(t) er vanskelig at behandle, men til gengæld optræder meget sjældent (P_n er lille).

Kompositionsmetoder virker ikke umiddelbart særlig elegant, men de kan være meget effektive, især hvis $P_i \simeq 1$ for en enkelt værdi af i.

Vægtfaktorerne (4.50) svarer til en diskret sandsynlighedsfordeling og kan naturligvis erstattes af en generel fordeling:

$$F(t) = \int_{-\infty}^{+\infty} g(t|y) \cdot dH(y) \tag{4.51}$$

(4.50) fås som specialtilfælde.

Metoden anvendes ved først at generere et tilfældigt tal til bestemmelse af, hvilken fordeling vi skal anvende, og derefter generere en observation fra denne.

I andre anvendelser udtrykkes den ønskede fordeling som et produkt af fordelinger, som vi hver for sig kan generere observationer fra. Endvidere kan kompositionsmetoder kombineres med afvisningsmetoden.

Eksempel 6.1

Hvis T_1 har fordelingen $F_1(t)$, og T_2 har fordelingen $F_2(t)$, så har $\max(T_1, T_2)$ fordelingen $F_1(t) \cdot F_2(t)$. Således (jf. eksempel 5.1) vil en stokastisk variabel med fordelingen

$$F(t) = t^2 = t \cdot t \qquad 0 \leq t \leq 1 \tag{4.52}$$

derfor kunne genereres ved at vælge det største af to ligeligt fordelte tilfældige tal.

4.6.1 Normalfordelingen N(0,1)

(jf. afsnit 4.7.1)

En simpel og udbredt metode til generering af Normalfordelte tal er baseret på den centrale grænseværdisætning.

Adderer man n rektangulært fordelte U(0,1) observationer, fås approksimativt en Normalfordelt størrelse med middelværdi n/2 og standardafvigelse $(n/12)^{1/2}$. Ved addition af 12 tilfældige tal og subtraktion med 6 fås derfor en størrelse, der tilnærmet er N(0,1)-fordelt:

$$T = \sum_{i=1}^{12} U_i - 6 \sim N(0,1) \tag{4.53}$$

Denne metode, der f.eks. anvendes i Fortran-SSP (program GAUSS), vil umiddelbart generere de ekstreme værdier for sjældent. I praksis kan man jo aldrig få værdier uden for intervallet (-6,6).

Med dette program bliver f.eks.

$$\begin{array}{ll} \text{P\{T<-4\}} = \begin{array}{l} 0.000009\ , \\ 0.000032 \end{array} & \text{hvor den burde være (teoretisk)} \end{array}$$

Der findes formler (bl.a. Teichroew's), der numerisk korrigerer for disse afvigelser og dermed resulterer i en næsten eksakt metode.

Metoden kræver minimal lagerplads og kan anvendes til små prøvekørsler, men det må i almindelighed tilrådes at bruge den eksakte metode, der er angivet i afsnit 4.7.1.

4.6.2 Hypereksponentialfordeling med n faser H_n

Denne fordelingsklasse har udstrakt anvendelse til beskrivelse af tidsintervaller, specielt i tele- og datatrafik (ref. Iversen 1985, kapitel 3).

Fordelingsfunktionen udtrykkes som en vægtet sum af eksponentialfordelinger:

$$\begin{aligned} F(t) &= \sum_{i=1}^{n} P_i \left(1 - e^{-\lambda_i t}\right) \\ &= 1 - \sum_{i=1}^{n} P_i \cdot e^{-\lambda_i t} \end{aligned} \qquad t \geq 0, \lambda_i \geq 0 \tag{4.54}$$

En observation genereres ved hjælp af to tilfældige tal. Det ene bruges til at bestemme klasse i med, og det andet bruges til generering af en eksponentialfordelt observation $\mathrm{Ex}(\lambda_i)$.

4.6.3 Erlang-k-fordelingen $E_k(\lambda) = G(k, \frac{1}{\lambda})$

Medens en hypereksponentialfordeling er en række eksponentialled i parallel, er Erlang-k-fordelingen k eksponentialled i serie. De k led har normalt samme parameter λ. Erlang-k-fordelingen er en speciel Γ-fordeling (afsnit 4.5.3), hvorfor alle metoder til generering af gammafordelte variable også kan bruges til generering af Erlang-k-fordelte variable.

Tæthedsfunktionen er givet ved

$$f(t) = \frac{k\lambda t^{k-1}}{(k-1)!} e^{-k\lambda t} \cdot \lambda \quad , \qquad \lambda > 0, k = 1, 2, \cdots \tag{4.55}$$

Middelværdien er normeret til $1/\lambda$ ved at erstatte t med kt.

Den kumulerede fordelingsfunktion kan udtrykkes som en sum af led fra Poissonfordelingen:

$$F(t) = 1 - e^{-\lambda t} \sum_{i=0}^{k-1} \frac{k\lambda t^i}{i!} \quad , \qquad k = 1, 2, \cdots \tag{4.56}$$

For k = 1 fås naturligvis eksponentialfordelingen som en sum af k eksponentialfordelinger kan vi imidlertid generere Erlang-k-fordelte observationer ved at addere k eksponentielle observationer (og eventuelt normere med k til middelværdien $1/\lambda$):

$$\begin{aligned} T &= \tfrac{1}{k} \textstyle\sum_{i=1}^{k} \left[-\tfrac{1}{\lambda} ln(U_i)\right] \\ &= \tfrac{1}{k\lambda} ln \textstyle\prod_{i=1} kU_i \end{aligned} \tag{4.57}$$

Vi skal således blot multiplicere k tilfældige tal og tage logaritmen til produktet.

4.6.4 Poissonfordelingen P(λ)

I en Poissonproces med intensiteten λ, er afstanden mellem hændelser eksponentialfordelt med samme intensitet. Tiden fra et vilkårligt valgt tidspunkt til næste hændelse er også eksponentialfordelt med samme intensitet, da eksponentialfordelingen er uden hukommelse.

Tiden, indtil der er indtruffet k hændelser, er følgelig Erlang-k-fordelt, og antal hændelser i et interval af fast længde t er Poissonfordelt med parameter λt :

$$P(n) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \quad , \qquad n = 0, 1, 2, \cdots \tag{4.58}$$

Man kan derfor generere en Poissonfordelt observation N med parameter λt ved at tælle, hvor mange tilfældige tal, vi skal generere, før produktet af disse er mindre end $e^{-\lambda t}$:

$$\prod_{i=1}^{N} U_i > e^{-\lambda t} \geq \prod_{i=1}^{N+1} U_i \tag{4.59}$$

4.6.5 Binomialprocessens fordelinger

Binomialprocessen er matematisk den diskrete analogi til den kontinuerte Poissonproces (Teletrafikteori (Iversen 1997)). Vi har allerede set slægtskabet mellem den geometriske fordeling og eksponentialfordelingen (afsnit 4.3.3).

Den geometriske fordeling NB(1,p) beskrev fordelingen af det antal forsøg, der (f.eks. med møntkast eller terningkast) skal foretages indtil første succes (inkl. dette forsøg).

Antal forsøg indtil netop k succeser er derfor netop en sum af k geometriske fordelinger og kaldes en Pascalfordeling eller en Negativ Binomialfordeling NB(k,p).

$$T = \sum_{i=1}^{k} Int\left[\frac{ln(U_i)}{ln(1-p)}\right] + k \tag{4.60}$$

er derfor NB(k,p)-fordelt.

På samme måde svarer Binomialfordelingen, der beskriver antal succeser i netop n forsøg, til Poissonfordelingen.

Hvis

$$N + \sum_{i=1}^{N} Int\left[\frac{ln(U_i)}{ln(1-p)}\right] \leq n < \sum_{i=1}^{N+1} Int\left[\frac{ln(U_i)}{ln(1-p)}\right] + N + 1 \tag{4.61}$$

vil N være Binomialfordelt B(n,p).

En Binomialfordelt observation kan også genereres ved at generere n tilfældige tal og tælle op, hvor mange der er mindre end p. Denne metode er velegnet for n lille.

For store værdier (n > 20 og np > 10) kan vi som en god approksimation bruge en Normalfordeling med middelværdi np og varians np(1-p). Men normalt bør man foretrække (4.61).

I afsnit 4.8 vises en nyere metode, der er meget effektiv for diskrete fordelinger, der antager et stort antal værdier.

Bemærk, at de enkelte geometriske fordelinger i ovenstående starter i klasse 1. I visse tilfælde anvender man geometriske fordelinger, der starter i klasse 0 ("antal forsøg før man får succes").

4.7 Matematiske metoder

Ved hjælp af matematiske omskrivninger kan visse komplekse fordelingsfunktioner skrives på en enklere form, for hvilken man kan generere tilfældige tal. Vi skal her gennemgå den vigtigste metode til generering af tilfældige observationer fra Normalfordelingen og dermed fra fordelinger der er beslægtet med denne.

4.7.1 Normalfordelingen $\mathbf{N}(\mu, \sigma^2)$

Den kumulerede fordelingsfunktion for en normeret Normalfordelt stokastisk variabel T er givet ved:

$$F(t) = \int_{-\infty}^{t} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx \qquad -\infty < x < +\infty \tag{4.62}$$

Denne funktion kan ikke udregnes analytisk, og det er derfor umuligt at finde den inverse fordelingsfunktion.

Box og Müller har (Box & Muller, 1958 [6]) vist, at man ud fra to uafhængige rektangulært fordelte tilfældige tal U_1 og U_2, kan generere to uafhængige N(0,1) - fordelte observationer T_1 og T_2.

Den simultane frekvensfunktion for to uafhængige N(0,1)-fordelte stokastiske variable er givet ved

$$\begin{aligned} f(t_1, t_2)dt_1dt_2 &= f(t_1)dt_1 \cdot f(t_2)dt_2 \\ &= \tfrac{1}{2\pi} e^{1/2(t_1^2+t_2^2)} dt_1 dt_2 \end{aligned} \tag{4.63}$$

Transformationerne

$$\begin{aligned} t_1 &= r \cdot cos\theta \\ t_2 &= r \cdot sin\theta \end{aligned} \tag{4.64}$$

til et polært koordinatsystem giver frekvensfunktionen

$$f(r,\theta)drd\theta = \frac{1}{2\pi}e^{-r^2/2}drd\theta \tag{4.65}$$

således at θ er rektangulært fordelt U(0,2π) og $\frac{r^2}{2}$ er eksponentialfordelt Ex(1).

Idet vi går ud fra to uafhængige tilfældige tal U_1 og U_2, får vi med de inverse transformationer for henholdsvis den rektangulære fordeling og eksponentialfordelingen:

$$\begin{aligned} r &= [-2ln(U_1)]^{1/2} \\ \theta &= 2\pi U_2 \end{aligned} \tag{4.66}$$

og dermed fra (4.64) to uafhængige Normalfordelte N(0,1) observationer:

$$\begin{aligned} T_1 &= [-2 \cdot ln(U_1)]^{1/2} \cdot cos(2\pi U_2) \\ T_2 &= [-2 \cdot ln(U_1)]^{1/2} \cdot sin(2\pi U_2) \end{aligned} \tag{4.67}$$

Det er tilstrækkeligt at kunne generere N(0,1)-fordelte observationer, idet er Normalfordeling med middelværdi μ og varians σ^2 N(μ, σ^2) fås fra N(0,1) ved den simple transformation:

$$x = \sigma \cdot t + \mu \tag{4.68}$$

Denne metode bør normalt anvendes i praksis, da den er eksakt. Ud fra Normalfordelingen kan vi umiddelbart opnå en række andre fordelinger.

4.7.2 Den logaritmiske Normalfordeling LN(α, β^2)

For at generere en observation L, der følger en logaritmisk Normalfordeling behøver vi blot at generere en Normalfordelt observation $T \in N(\alpha, \beta^2)$ og foretage transformationen

$$L = e^T$$

4.7.3 Chi-i-anden fordelingen χ^2(n)

En observation fra denne fordeling kan som tidligere nævnt genereres som en observation fra en speciel Γ-fordeling G($\frac{n}{2}$,2). (afsnit 4.5.3)

Fra statistikken ved vi imidlertid også, at den kan genereres ved at kvadrere og addere n normerede Normalfordelte observationer, hvor n er antallet af frihedsgrader i χ-fordelingen.

Hvis n er et lige tal, er $\frac{n}{2}$ et helt tal, og vi får da af (4.67) :

$$T = -2\sum_{i=1}^{\frac{n}{2}} ln(U_i) \tag{4.69}$$

idet de trigonometriske led parvis går ud, når vi kvadrerer og adderer. Dette svarer netop til, at χ^2-fordelingen bliver en sum af $\frac{n}{2}$ eksponentielle led, en Erlang- $\frac{n}{2}$ fordeling.

Er n ulige fås

$$T = -2\sum_{i=1}^{(n-1)/2} ln(U_i) + (T_1)^2 \tag{4.70}$$

hvor T_1 er N(0,1)-fordelt.

For n=2 fås en eksponentialfordelt observation med middelværdi 2. Vi har således en direkte sammenhæng mellem en eksponentialfordelt observation og summen af kvadraterne af to Normalfordelte observationer, som kan være genereret med en vilkårlig metode.

4.7.4 Student's t-fordeling t(n,μ)

Hvis T_1 er en observation fra en N(μ,1)-fordeling, og T_2 er en observation fra en χ-fordeling, vil

$$T = \frac{T_1}{\sqrt{T_2/n}} \tag{4.71}$$

være t-fordelt med n frihedsgrader og skævhedsparameter μ.

Generering af et større antal observationer fra t-fordelingen kan dog ske ved mere direkte metoder. I (Chen 1978) er der angivet nøjagtige approksimationer til den inverse fordelingsfunktion.

4.7.5 F-fordelingen

En observation fra denne fordeling fås ved at tage forholdet mellem to χ^2-fordelte observationer. Der henvises i øvrigt til (Conradsen 1976).

4.8 Alias-metoden for diskrete fordelinger

I afsnit 4.4.3 så vi, at ved numerisk invers transformation af en diskret fordeling skal vi, i de tilfælde hvor den inverse funktion ikke eksisterer, søge efter det interval, hvor $f_i \leq U_i < f_{i+1}$

(4.33). Dette vil i de tilfælde, hvor der er mange klasser, selv med effektive søgeprocedurer, kræve en del ressourcer pr. realisation.

Walker (1977) har anvist en ny effektiv metode til generering af pseudo-tilfældige variable fra en vilkårlig diskret sandsynlighedsfordeling med et endeligt udfaldsrum. Metoden er beslægtet med afvisningsmetoden, men er langt mere effektiv, fordi de "afviste" tilfældige tal ikke kasseres, men erstattes med "alias'er". For hvert ligeligt fordelte tilfældige tal fås netop én observation med den ønskede fordeling.

4.8.1 Beskrivelse af alias-metoden

Vi betragter en diskret sandsynlighedsfordeling, der med sandsynligheden p(i) antager værdien i ($0 \leq i \leq n$), hvor $\sum_i p(i) = 1$. Vi ønsker at generere tilfældige observationer med denne fordeling.

Den generelle alias-algoritme er som følger:

1. Generer et tilfældigt tal $U \in U(0,n+1)$
2. I := Int(U)
   U_1 := U - I (=Frac(U))
3. T := $\begin{cases} I & \text{hvis} \quad U_1 < F(I) \\ L(I) & \text{ellers} \end{cases}$

F(i) og L(i) (i=0,1,...,n) er tabeller med konstanter, som afhænger af den betragtede fordeling. Vi vender senere tilbage til konstruktionen af disse.

Vi genererer ét tilfældigt tal, som spaltes op i to tilfældige tal, heltalsdelen I og brøkdelen U_i, der er et tilfældigt tal U(0,1). (Er tilfældighedsgeneratoren ikke god, må det anbefales at bruge 2 tilfældige tal til dette). U_1 sammenlignes med tabelværdien F(I). Er $U_1 \leq F(I)$, er I den ønskede observation; hvis ikke, er det I's alias L(I).

Det fremragende ved alias-metoden er, at det kun er nødvendigt med én sammenligning, uanset hvor mange klasser fordelingen har.

I ord kan alias-metoden forklares på følgende måde. Vi starter med at generere et tilfældigt tal fra en ligefordeling U(0,n+1). Sammenholdt med ligefordelingen vil den fordeling, vi betragter, have klasser (f.eks. klasse j), hvor sandsynlighedsmassen er mindre end i ligefordelingen, og klasser (f.eks. klasse k), som har større sandsynlighedsmasse end ligefordelingen.

Ved i første omgang at fordele sandsynlighedsmassen efter ligefordelingen vil klasse j således få for meget (være rig), og klasse k vil få for lidt (være fattig). Med alias-metoden giver

den rige til den fattige. Tabellen F(j) angiver hvor meget, den rige selv skal bruge (p(j)), og L(j) angiver adressen på en fattig (alias'en), som får hele overskuddet. Hvis den fattige derved ikke får nok, kan den også få fra andre rige. Den fattige får således overskuddet fra et bestemt antal rige. Får den derved for meget, vil den altid kunne opnå balance ved at forære en del af det, den oprindeligt selv fik, videre til en anden fattig (delen angives af dens F(k), og adressen på den anden fattige bestemmes af L(k)). Der vil netop være mindst én fattig, som ikke forærer noget videre, d.v.s. én værdi i tabellen L(i) vil være ubestemt.

Vi skal ikke gå ind på beviset for, at metoden er teoretisk korrekt (se Kronmal & Peterson, 1979), men blot angive en algoritme for, hvorledes tabellerne kan konstrueres. Metoden er baseret på den kendsgerning, at enhver diskret fordeling med n klasser kan skrives som en vægtet sum af n-1 to-punktsfordelinger, hvor alle fordelinger optræder med samme vægt (Dieter 1982).

4.8.2 Generering af tabellerne F(i) og L(i)

Den følgende algoritme, der er angivet i (Kronmal & Peterson, 1979), konstruerer ovennævnte tabeller i et endeligt antal trin (maksimalt = antal klasser - 1).

1. Opret de to tabeller $F(i)$ og $L(i)$, $i = 0, 1, \ldots, n$.
2. Initialiser $F(i)$: $F(i) = (n+1)\,p(i), i = 0, 1, \ldots$,n.
   ($n+1$ er antallet af udfald af den stokastiske variable)
3. Definér to mængder G og S. Lad G(g̃reater) bestå af mængden af i'er, for hvilke F(i) ≥ 1. Lad S(s̃maller) bestå af mængden af i'er, for hvilke F(i) ≤ 1. (Mængden S er kun tom, hvis den ønskede fordeling er en ligefordeling; i det tilfælde er algoritmen færdig).
4. Udfør trin (5) - (8) indtil mængden S er tom. Sløjfen gennemløber højst n gange. (Bemærk at mindst én indgang i tabellen L(i) forbliver udefineret).
5. Vælg et sæt indices: Vælg et element i G, og betegn det med k. Vælg et element i S, og betegn det med j. (Hermed vil F(j) (j's eget behov) være fastlagt).
6. Vælg j's alias: L(j) := k (Hermed er L(j) bestemt).
7. Redefinér F(k): F(k) := F(k) - (1-F(j)).
   ((1-F(j)) angiver den mængde, som j forærer til k, F(k) er nu ikke nødvendigvis større end én mere).
8. Redefinér mængderne G og S
   (a) hvis F(k) < 1, så skal k flyttes fra G til S:
   G := G - {k} , S := S + {k}
   (hvis F(k) < 1, har klasse k fået mere, end den skal bruge)

(b) S := S - {j}
(klasse j er færdigbehandlet, idet den netop har foræret det, den kan undvære, væk).

Eksempel 4.8.1:
Vi betragter den diskrete fordeling

$$P(1) = \frac{1}{6}, P(2) = \frac{1}{12}, P(3) = \frac{7}{12}, P(4) = \frac{1}{6}$$

Antal klasser er således 4

(1)	$F(1) = \frac{8}{12}, F(2) = \frac{4}{12}, F(3) = \frac{28}{12}, F(4) = \frac{8}{12}$
(2)	G = {3} , S = {1,2,4}
(4)	k = 3 , j = 1
(5)	L(1) = 3
(6)	$F(3) = \frac{28}{12} - (1 - \frac{8}{12}) = \frac{24}{12}$
(7b)	S = {2,4}
(4)	k = 3 , j = 2
(5)	L(2) = 3
(6)	$F(3) = \frac{24}{12} - (1 - \frac{4}{12}) = \frac{16}{12}$
(7b)	S = {4}
(4)	k = 3 , j = 4
(5)	L(4) = 3
(6)	$F(3) = \frac{16}{12} - (1 - \frac{8}{12}) = 1$
(7b)	S = Ø (tom)

Resultatet bliver således:

$F(1) = \frac{8}{12}$	$F(2) = \frac{4}{12}$	$F(3) = 1$	$F(4) = \frac{8}{12}$
$L(1) = 3$	$L(2) = 3$	$L(3) = ?$	$L(4) = 3$

Klasse 3 får overskud fra de 3 øvrige klasser og har dermed netop fået dækket sit behov. (En dårlig algoritme kunne resultere i, at f.eks. klasse 2 forærede endnu mere af sit eget til klasse 3, som så til gengæld forærede en tilsvarende mængde af sit eget til klasse 2). Se Fig. 4.6. □

Af hensyn til afrundingsfejl bør man ved implementeringen af algoritmen sikre sig, at disse ikke får indflydelse på det antal trin, der er nødvendigt for oprettelse af tabellerne.

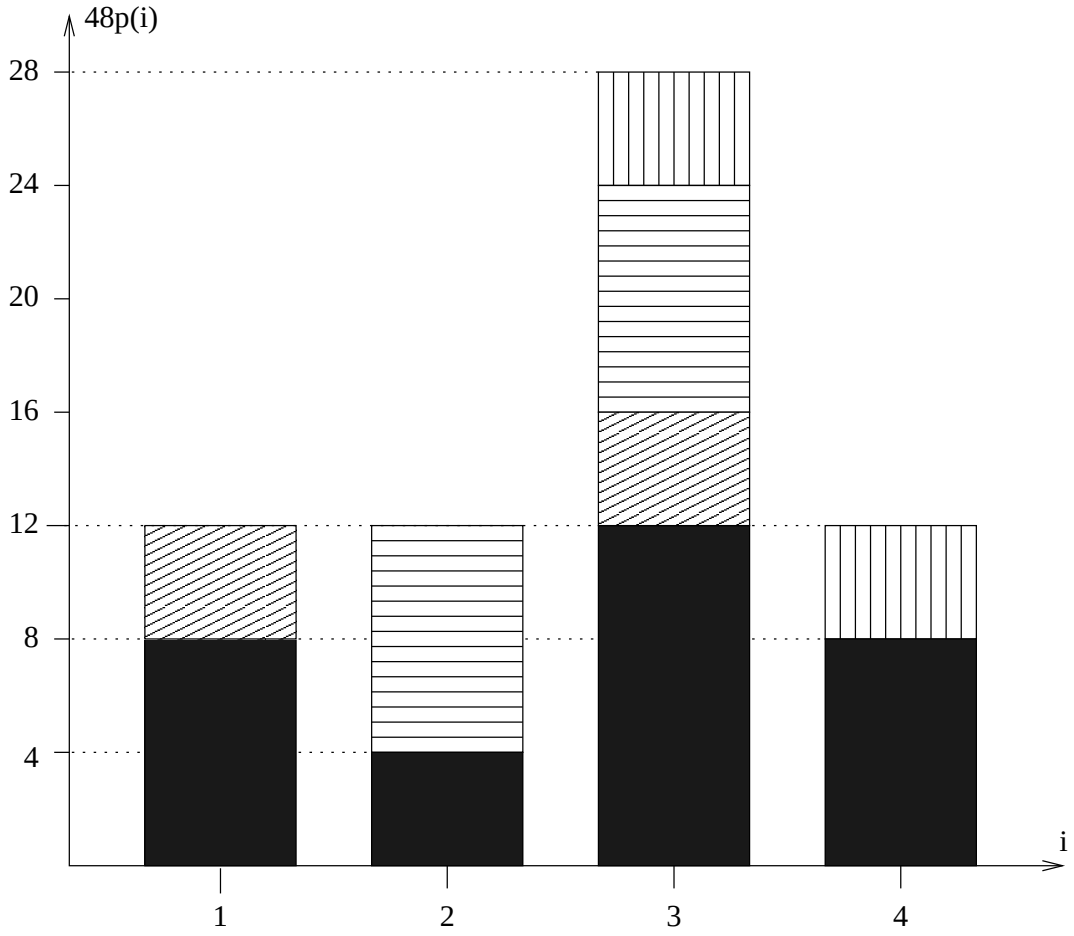


Figure 4.6: *Illustration af hvorledes alias–metoden omfordeler ligefordelte sandsynlighsmasser, så at den ønskede fordeling opnås (Eksempel* **??***).*

Eksempel 4.8.2: Cox-fordelinger
Cox-fordelte stokastiske variable (Iversen, 1985, kap. 3) genereres ved at bestemme antal faser med alias-metoden. De enkelte faser genereres som eksponentialfordelte stokastiske variable (Neuts 6 Pagano, 1981). □

4.8.3 Afsluttende bemærkninger

Algoritmen karakteriseres ved, at den kræver en lagerplads på $2x$ ord til tabeller, hvor x er antallet af klasser i fordelingen. Tabellerne konstrueres en gang for alle og kræver højst x operationer.

Til generering af en tilfældig observation kræves kun et tilfældigt tal, som opdeles i en heltalsdel og en brøkdel. Med det tilfældige tal foretages én sammenligning, der enten resulterer i accept, eller, i tilfælde af forkastelse, i et tabelopslag.

Principielt kan metoden kun anvendes til diskrete fordelinger med et endeligt udfaldsrum. I praksis er der dog ingen problemer med et ubegrænset antal klasser, idet (f.eks. i Poissonfordelingen) et endeligt antal klasser vil omfatte så godt som hele sandsynlighedsmassen. Man kan evt. betragte halen som en enkelt klasse og bagefter give denne en specialbehandling. Metoden kan også udvides til visse kontinuerte fordelinger (Kronmal & Peterson, 1979).

Metoden kan effektiviseres ved at forøge antallet af klasser i tabellerne, idet vi da kan introducere klasser, der ikke har en alias (degenererede to-punktsfordelinger). I disse klasser skal vi kun bruge ét tilfældigt tal.
Foranstående eksempel 4.8.1 kan således ved anvendelse af 12 klasser (12 degenererede topunktsfordelinger) samples ved anvendelse af kun ét tilfældigt tal (Klein 1989). Dette svarer til Urne-metoden, idet vi trækker fra en urne, hvor hver klasse er repræsenteret med et antal kugler (i ovenstående eksempel henholdsvis 2, 1, 7 og 2).

4.9 Stokastiske afhængige observationer

Flerdimensionale uafhængige variable kan naturligvis genereres ud fra de éndimensionale fordelinger.

Er kovariansen mellem de stokastiske variable positiv, kan de flerdimensionale stokastiske variable også genereres succesivt ud fra de éndimensionale fordelinger efter de foranstående metoder, hvis alle de betingede fordelinger er kendte.

Multivariable Normalfordelinger med kendte kovariansmatricer kan genereres ved transformation af éndimensionale Normalfordelinger. Man kan også generere autokorrelerede tidsrækker

af Normalfordelte variable, og enhver stationær Gaussisk proces kan approksimeres vilkårligt godt med digital filtreret hvid støj og simuleres.

Vi skal ikke omtale dette emne nærmere her, men f.eks. henvise til (Andréasson 1973).

4.10 Afsluttende bemærkninger

Der findes en omfattende litteratur om generering af tilfældige observationer med bestemte statistiske fordelinger. For mange fordelinger er der publiceret specielle effektive og/eller nøjagtige metoder. Nye artikler fremkommer jo netop, fordi de publicerede metoder har fordele frem for kendte metoder. Der foregår derfor en konkurrence på effektivitet, og mange effektive metoder er ret komplekse at forstå og at implementere.

Disse effektive metoder bør tages i betragtning, hvis man skal udvikle i standardprogrammer til et datamaskinsystem. De kan f.eks. søges i (Sowey 1972 og 1978, Knuth 1969), hvor man også får indtryk af hvilke tidsskrifter, der behandler emnet.

Den almindelige bruger af simulation som værktøj står sig derimod bedst ved at anvende *simple* og *nøjagtige* metoder baseret på en god tilfældigtals generator. Eksempelvis kan man for eksponentialfordelingen således anvende logaritmetransformationen (4.10) og for Normalfordelingen Box & Müllers metode (4.67). Kun hvis de nødvendige matematiske funktioner, der skal anvendes i en metode, ikke eksisterer (f.eks. mikro-processorer) bør man overveje andre metoder. Ulempen ved at bruge mange tilfældige tal pr. observation ophæves i et vist omfang at en forbedring i de statistiske egenskaber af observationerne. Disse metoder er ganske vist ikke de mest effektive, men forskellen er normalt uden betydning i praksis, sammenholdt med de ressourcer en opgave i øvrigt kræver. Under programudviklingen undgår man endvidere mistanke om, at besynderlige resultater skyldes fejl ved de genererede tilfældige observationer, og man kan helt koncentrere sig om modelopbygningen, hvor der ofte er store besparelser i regnetid at hente. I mange simulationer kræver udvikling og afprøvning af programmer lige så meget regnetid som selve produktionskørslen.

Generering af tilfældige permutationer planlægges behandlet i forbindelse med simulation af køsystemer.

Bibliography

[1] Ahrens, J.H. & Dieter, U. (1972): Computer Methods for Sampling from the Exponential and Normal Distributions. Communications of the ACM, Vol. 15 (1972) : 10, pp. 873–882.

[2] Andersson, S.L. (1990): Random Number Generators on Vector Supercomputers and other Advanced Architecture SIAM Review, Vol. 32 (1990) : 2, 221–251.

[3] Andréasson, I. (1973): *Simulationsmetodik*. KTH, Stockholm 1973. 107 pp.

[4] Atkinson, A.C. (1977): An Easily Programmed Algorithm for Generating Gamma Random Variables. J. R. Statist. Soc. A, Vol. 140 (1977), Part 2, 232–234.

[5] Blum, L. & Blum, M. & Shub, M. (1986): A Simple Unpredictable Pseudo-Random Number Generator. SIAM J. Comput., Vol. 15 (1986) : 2, 364–383.

[6] Box, G.E.P. & Muller, Mervin E. (1958): A Note on the Generation of Random Normal Deviates. The Annals of Mathematical Statistics, Vol. 29 (1958) 610–611.

[7] glossaryBratley, P. Bratley, P. & Bennett, L.F. & Schrage, L.E. (1983): *A Guide to Simulation*. Springer–Verlag 1983. 383 pp.

[8] Christiansen, H. Dalgas (1975): Random Number Generators in Several Dimensions. Theory, Tests and Examples. IMSOR, Technical University of Denmark 1975. 66 pp.

[9] Coates, R.F.W. & Janacek, G.J. & Lever, K.V. (1988): Monte Carlo Simulation and Random Number Generation. IEEE Journal of Selected Areas in Communications, Vol. 6 (1988) : 1, 58–66.

[10] Collings, B.J. (1988): Compound Random Number Generators. JASA, Vol. 82 (1988) : 398, 525–527.

[11] Conradsen, K. (1976): *En Introduktion til Statistik*. IMSOR, Technical University of Denmark 1976. 484 pp.

[12] Chen, H.J. (1978): On approximations to the Inverse Student's-t Distribution Function. Journal of Statistical Computation and Simulation, Vol. 7 (1978):3-4. pp. 167 - 180.

[13] Devroye, L. (1981): Recent Results in Non-Uniform Random Variate Generation. 1981 Winter Simulation Conference Proceedings, pp. 517 - 521.

[14] Dieter, U. (1982): An Alternate Proof for the Representation of Discrete Distributions by Equiprobable Mixtures. Journal of Applied Probability, Vol. 19 (1982) 869 - 872.

[15] Dodge, Y. (1996): A Natural Random Number Generator. International Statistical Review, Vol. (64) : 3, 329–344.

[16] Eichenauer-Herrmann, J. (1992): Inversive Congruential Pseudorandom Numbers: a Tutorial. International Statistical Institute, Vol. 60 (1992):2, 167-176.

[17] Van Es, A.J., R.D. Gill and C. Van Putten (1983): Random Number Generators for a Pocket Calculator. Statistica Neerlandica, Vol. 37 (1983): 3, pp. 95 - 102.

[18] Glasserman, P. (1992): Some guidelines and Guarantees for Common Random Numbers. Management Science, Vol. 38 (1992):6, 884-908.

[19] Haas, A. (1987) The Multiple Prime Random Number Generator. ACM Transactions on Mathematical software, Vol. 13 (1992):4, 368-381.

[20] Hammersley, I.M. and D.C. Handscomb (1964): Monte Carlo Methods, London 1964. 178 pp.

[21] Iversen, V.B. (1985): *Data- og Teletrafikteori.* Den Private Ingeniørfond 1985. Del I 313 pp. Del II pp.

[22] Isaksson, H. (1958): Generator for Tilfældige Tal. Teleteknisk nr. 4, 1958, pp. 175 - 186.

[23] Jansson, B. (1966): *Random Number Generators.* Stockholm 1966. Kapitel 8: Pseudorandom numbers from various statistical distributions. pp. 170–191.

[24] Jennergren, L.P. (1984): Discrete-events simulations models in PASCAL/MT+ on a microcomputer. Studentlitteratur 1984. 135 pp.

[25] Jensen, Arne (1967): Trækning af Kreditforeningers Kasseobligationer. IMSOR 1967. 61 pp.

[26] Jørgensen, F.G., A. Fischer Madsen og H. Marqvardsen (1974): Documentation and Test of Exor-Generator. IMSOR 1974. 70 pp.

[27] Kao, Chiang (1989): A Random-number Generator for Microcomputers. J. Opl. Res. Soc., Vol. 40 (1989):7, pp. 687 - 691.

[28] Kleijnen, J.P.C., and B. Annink (1992): Pseudorandom Number Generators for Supercomputers and Classical Computers: A Practical Introduction. European Journal of Operations Research, Vol. 63 (1992), 76-85.

[29] Klein, Henrik (1989): Alias-Urne metoden. Rapport opgave i Simulering. IMSOR 1989. 15 pp.

[30] Knuth, D.E. (1981): *The Art of Computer Programming.* Vol. 2: Seminumerical Algorithms. Chapter 3: Random Numbers. pp. 1–177. Second edition. Addison–Wesley Publ. Co. 1981.

[31] Kobayashi, H. (1978): Modeling and Analysis: An Introduction to System Performance Evaluation Methodology. Addison-Wesley Publ. Co. 1978. Chapter 4: The Simulation Method. pp. 221 - 314.

[32] Kronmal, R.A. and A.V. Peterson, Jr. (1979): On the Alias Method for Generating Random Variables From a Discrete Distribution. The American Statistician, Vol. 33 (1979): 4, pp. 214 - 218.

[33] Kronmal, R.A. and Arthur V. Peterson (1982): On Mixture Methods for the Computer Generation of Random Variables. The American Statistician, Vol. 36 (1982): 3, part 1, pp. 184 - 191.

[34] Lecuyer, Pierre (1988): Efficient and Portable Combined Random Number Generators. Communications of the ACM, Vol. 31 (1988): 6, pp. 742 - 749 + 770.

[35] Lewis, P.A.W. (1980): Chapter G of the IMSL Library - Generation and Testing of random deviates: Simualtion. Proceedings of the 1980 Winter Simulation Conference, pp. 357 - 360.

[36] Maclaren, N.M. (1989): The Generation of Multiple Independent Sequences of Pseudo-random Numbers. Appl. Statistics, Vol. 38 (1989):2, 351-359.

[37] Malmberg, A.C. (1971): Matematiske Tabeller: Erlang T. København 1971. 40 pp.

[38] Marqvardsen, H.M. (1974): Generering af Tilfældige Tal. Kapitel 5 i "Skemalægning ved Numerisk Simulation". IMSOR 1974, pp. 91 - 136.

[39] Marsaglia, G. (1961): Expressing a Random Variable in Terms of Uniform Random Variables, Ann. Math. Stat., Vol. 32 (1961), pp. 894 - 898.

[40] Marsaglia, G. (1961): Generating Exponential Random Variables. Ann. Math. Stat., Vol. 32 (1961), pp. 899 - 902.

[41] Marsaglia, G., and A. Zaman (1990): Towards a Universal Random Number Generator. Statistics & Probability Letters, Vol. 8 (1990) 35-39.

[42] Marsaglia, G., and A. Zaman (1991): A New Class of Random Number Generators. The Annals of Applied Probability, Vol. 1 (1991):3, 462-480.

[43] Neuts, M.F. and M.E.Pagano (1981): Generating Random Variates from a Distribution of Phase Type. 1981 Winter Simulation Conference Proceedings, pp. 381 - 387.

[44] Newman, T.G. and P.L. Odell: *The Generation of Random Variates.* London 1971. 88 pp.

[45] Niederreiter, H. (1991): Recent Trends in Random Numbers and Random Vector Generation. Annals of Operations Research, Vol. 31 (1991) 323-346.

[46] Park, Stephen K. and Keith W. Miller (1988): Random Number Generators: Good Ones are hard to Find. Communications of the ACM, Vol. 31 (1988): 10, pp. 1192 - 1201.

[47] Pohl, p. (1975): MCV - A Fast Pseudo-random Number Generator with Extremely Good Statistical Properties. Afhandling, KTH, Stockholm 1975. 17 pp.

[48] Rand Corporation (1955): A Million Random Digits with 100.000 Normal Deviates. Glencoe, Illinois 1955.

[49] Ripley, B.D. (1983): Computer Generation of Random Variables: A Tutorial. International Statistical Rewiew, Vol. 51 (1983), pp. 301 - 319.

[50] Schmeiser, B.W. (1980): Random Variate Generation: A Survey. Simulation with Discrete Models: A State-of-the-Art Survey. IEEE 1980. pp. 79 - 104.

[51] Schmeiser, B.W. (1981): Random Variate Generation. 1981 Winter Simulation Conference Proceedings, pp. 227 - 242.

[52] Sowey, E.R. (1972): A Chronologocal and Classified Bibliography on Random Number Generation and Testing. Int. Stat. Rev., Vol. 40 (1972):3, pp. 355 - 371.

[53] Sowey, E.R. (1978): A Second Classified Bibliography on Random Number Generation and Testing. Int. Stat. Rev., Vol. 46 (1978):1, pp. 89 - 102.

[54] Tadikamalla, P.R. (1978): Computer Generation of Gamma Random Variables. Communications of the ACM, Vol. 21 (1978): 5, pp. 419 - 422.

[55] Tippett, L.H.C. (1959): Random Sampling Numbers. Tracts for Computers No. XV. Cambridge 1959.

[56] Tocher, K.D. (1963): *The Art of Simulation*. London 1963. 184 pp.

[57] Walker, A.J. (1977): An Efficient Method for Generation Discrete Random Variables with General Distributions. ACM Transactions on Mathematical Software, Vol. 3 (1977), pp. 253 - 256.

[58] Yakowitz, S.J. (1977): Computational Probability and Simulation. Reading, Massachusetts 1977. Kapitel 2, pp. 39 - 76.

4.11 Fordelingsindeks

Alfabetisk oversigt over fordelinger, der er omtalt i teksten. Tallene henviser til afsnit, og understregning henviser til metoder, der normalt må anbefales.

Chapter 5

Simulation af Markovprocesser

I dette afsnit ser vi på simulering af Markov-processer, der hyppigt anvendes til modellering af stokastiske systemer. Markovprocesser er karakteriseret ved, at processens fremtidige udvikling kun afhænger af den aktuelle tilstand, ikke af hvorledes processen kom til denne tilstand. Dette er ækvivalent med at sige, at alle tidsintervaller er eksponential-fordelte stokastiske variable i kontinuert tid, geometrisk fordelte i diskret tid.

Vi vil først se på generering af punktprocesser (f.eks. kundeankomster, tidspunkter for ulykker, etc.). Blandt disse er den stationære Poissonproces helt fundamental. Vi skal derefter se, hvorledes vi kan simulere inhomogene Poissonprocesser, Markov-modulerede Poissonprocesser mv. En vigtig Monte-Carlo teknik er den såkaldte Roulette-metode, hvor vi simulerer den indlejrede Markovkæde (diskret tid) i stedet for den kontinuerte Markovproces.

5.1 Stationære Poissonprocesser

En stationær Poissonproces er entydigt karakteriseret ved processens intensitet $\lambda > 0$, der er det gennemsnitlige antal hændelser per tidsenheden. En af de følgende to egenskaber er tilstrækkelig til at karakterisere processen:

A. Antal hændelser i et interval af fast længde t er **Poissonfordelt** (deraf navnet Poissonproces):

$$P(i,t) = \frac{(\lambda t)^i}{i!} e^{-\lambda t}, \quad i = 0, 1, \cdots .$$

Middelværdi og varians er begge lig med $\lambda\, t$.

B. Afstanden mellem to på hinanden følgende hændelser er **eksponentialfordelt** med

Tæthedsfunktion:	$f(t) = \lambda e^{-\lambda t} i\,,$	$t > 0\,.$
Fordelingsfunktion:	$F(t) = 1 - e^{-\lambda t}$	$t > 0\,.$

Eksponentialfordelingen har middelværdien $(1/\lambda)$ og variansen $(1/\lambda)^2$.

Vi kan derfor simulere en stationær Poissonproces ved at generere Poissonfordelte eller eksponentialfordelte tilfældige tal med intensiteten λ . Simulerer vi f.eks. ankomster i et servicesystem, genereres næste ankomsttidspunkt T_{i+1} ud fra det foregående ankomsttidspunkt ved (fig. 1):

$$T_{i+1} = T_i - (1/\lambda) \cdot \ln(U)$$

hvor U er et tilfældigt tal.

Eksponentialfordelingens egenskaber

En god forståelse af eksponentialfordelingens egenskaber er fundamental for forståelsen af grundideen i Markovkæde simulation.

A. Eksponentialfordelingens mangel på hukommelse

Betragter vi eksponentialfordelte levetider (f.eks. levetiden af en elektrisk pære), vil den resterende levetid fra et givet tidspunkt være uafhængig af den aktuelle alder, dvs. der er ingen alder (hukommelse) indbygget i eksponentialfordelingen:

$$\begin{aligned} f(t+x|x) &= \tfrac{f(t+x)}{1-F(x)} \\ &= \tfrac{\lambda e^{-\lambda(t+x)}}{e^{-\lambda x}} \\ &= \lambda e^{-\lambda t} \\ &= f(t) \end{aligned}$$

B. Den mindste af k eksponentialfordelte størrelser er selv eksponentialfordelt.

Det gælder generelt, at

$$P\{X_i \leq t\} = F_i(t)\,, \quad i = 1, 2, \cdots, k\,.$$

$$F_{min}(t) = P\{min\{X_i\} \leq t\}$$

$$= 1 - \prod_{i=1}^{k}(1 - F_i(t))$$

Indsætter vi eksponentialfordelingen, finder vi

$$F_i(t) = 1 - e^{-\lambda_i t}$$

$$F_{min}(t) = 1 - e^{-\lambda_s t}$$

hvor

$$\lambda_s = \lambda_1 + \lambda_2 + \cdots + \lambda_k$$

Eksempel: Er k kunder under betjening i et køsystem med eksponentialfordelt betjeningstid med intensitet λ , er tiden indtil den første bliver færdig eksponentialfordelt med intensiteten $k\lambda$.

C. Sandsynligheden for at den første hændelse (den mindste observation) er fra eksponentialfordeling nummer i, er:

$$P\{\text{i er den mindste}\} = \frac{\lambda_i}{\lambda_s}$$

5.2 Inhomogene Poissonprocesser

Afhænger intensiteten i Poissonprocessen af tiden, $\lambda = \lambda(t)$ (f.eks. døgnvariationer), kan man for et bestemt tidsrum stadig bruge samme formler som ovenfor, idet λ erstattes med $\overline{\lambda(t)}$:

$$\overline{\lambda} = \overline{\lambda(t)} = \frac{1}{t}\int_0^t \lambda(u)du$$

$$F(t) = 1 - e^{-\overline{\lambda(t)}}$$

$$P(i,t) = \frac{(\overline{\lambda} \cdot t)^i}{i!}e^{-\overline{\lambda} \cdot t}$$

Sandsynligheden for at få en hændelse inden tidspunktet t:

$$F(t) = 1 - P(0,t)\,,$$

eller for at få i hændelser i intervallet $(0,t)$ afhænger således kun af det totale areal under $\lambda(u)$, ikke af hvorledes dette er fordelt i intervallet $(0,t)$ (jf. fig. 3).

Der er flere måder, hvorpå man kan simulere en inhomogen Poissonproces.

PS: Der mangler nogle figurer her

- **A. Stykvis konstant intensitet (fig. 2)**

  Er intensiteten stykvis konstant, kan vi i hvert interval med konstant intensitet bruge metoden for en stationær Poissonproces. Overgangen mellem de enkelte intervaller klares let ved udnyttelse af eksponentialfordelingens mangel på hukommelse. Lad intensiteten i det første interval være λ_1 , og lad den skifte til λ_2 til tidspunktet t_1. Vi genererer da hændelser med intensiteten λ_1 , indtil vi første gang får en hændelse efter tidspunktet t_1. Denne sidstnævnte hændelse kasseres, tiden skrues tilbage til t_1. Med dette tidspunkt som starttid simuleres derefter hændelser med intensiteten λ_2. Tidspunktet t_1 er ikke nogen hændelse. Ovennævnte fremgangsmåde fungerer på grund af Poissonprocessens mangel på hukommelse. Vi kan også fortolke fremgangsmåden som følger. Vi genererer hele tiden hændelser fra alle Poissonprocesserne, men i et givet interval medtager vi kun hændelse fra den proces, der har den betragtede intensitet. Da alle processerne er uden hukommelse, behøver vi ikke at starte dem til tidspunkt $-\infty$, men kan vente med at starte dem til det tidspunkt, hvor vi begynder at observere dem.

- **B. Integrabel intensitetsfunktion**

  Da det i et givet interval kun er det totale areal under intensitetsfunktionen, der betyder noget, kan vi også generere hændelser fra en stationær Poissonproces, og derefter transformere tidsaksen så vi får den inhomogene Poissonproces. Lad den homogene Poissonproces med intensiteten λ realisere en hændelse til tidspunktet t. Dette svarer til et areal $\lambda \cdot t$. Vi skal derfor finde den værdi af t, for hvilken

  $$\overline{\lambda(t)} \cdot t = \lambda \cdot t$$

  Dette er muligt hvis integralet af intensitetsfunktionen er let at udregne, f.eks. for stykvis lineære intensitetsfunktioner.

- **C. Afvisningsmetoden**

  Denne metode kan anvendes for alle intensitetsfunktioner, hvor det for en given værdi t er let at udregne $\lambda(t)$. Vi starte igen med at generere en hændelse fra en Poissonproces med konstant intensitet $\lambda > max\{\lambda(t)\}$. Finder vi en hændelse til tidspunktet τ, afgør vi med et tilfældigt tal U, om vi skal acceptere denne hændelse eller ej. Har vi

  $$U < \lambda(\tau)/\lambda$$

accepteres hændelsen fra den homogene Poissonproces som en hændelse i den inhomogene Poissonproces. Ellers forkastes den. Vi omformer således en homogen Poissonproces til en inhomogen Poissonproces, ved at tilskære den homogene. På grund af Poissonprocessens mangel på hukommelse, er det kun intensiteten til tidspunktet τ, der betyder noget, ikke arealet fra den foregående hændelse (spaltningssætningen).

I stedet for at bruge en stationær Poissonproces som indhylningskurve, kunne ved også bruge en stykvis lineær indhylningskurve (ref. pkt. B), og dermed kassere færre hændelse i den oprindelige proces.

Den fremgangsmåde er generel og simpel at programmere.

5.3 Markovmodulerede Poissonprocesser (MMPP)

En Markov Moduleret Poisson Proces MMPP er en Poissonproces, hvor intensiteten kan antage en række forskellige værdier

$$\lambda_1, \lambda_2, \cdots, \lambda_k$$

En Markovkæde afgør, hvor længe processen er i en given tilstand, og hvilken tilstand, der skiftes til.

I Fig. 5.1 betragter vi en MMPP med to tilstande. I tilstand i er Poissonprocessens ankomstintensitet λ_i.

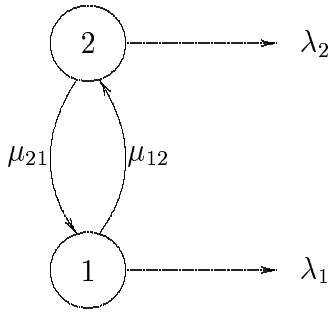


Figure 5.1: En Markov Moduleret Poisson Proces med to tilstande

Processen veksler mellem disse to tilstande, og opholdstiden i en tilstand er eksponentialfordelt med intensiteten μ_{12}, henholdsvis μ_{21}. Er vi i tilstand 1, er næste hændelse enten en hændelse i Poissonprocessen med intensitet λ_1 eller eller et tilstandsskift med intensiteten μ_{12} til tilstand 2. Begge tidsintervaller er eksponentialfordelte, og tiden indtil den første af disse to hændelser er derfor også eksponentialfordelt. Vi kan derfor simulere processen som følger:

Tilstand 1: Når vi er i denne tilstand indtræffer næste hændelse efter et eksponentialfordelt tidsrum med intensiteten $(\lambda_1 + \mu_{12})$. Sandsynligheden for at det er en ankomst, henholdsvis et skift er:

$$P(ankomst) = \frac{\lambda_1}{\lambda_1 + \mu_{12}}$$

$$P(skift) = \frac{\mu_{12}}{\lambda_1 + \mu_{12}}$$

Tilstand 2:

$$P(ankomst) = \frac{\lambda_2}{\lambda_2 + \mu_{21}}$$

$$P(skift) = \frac{\mu_{21}}{\lambda_2 + \mu_{21}}$$

For at simulere processen bruger vi således to tilfældige tal per gang: et til at generere afstanden til næste hændelse, og et til at afgøre, hvad det er for en hændelse.

Er vi kun interesseret i sekvensen af hændelser, og nummererer disse $1, 2, 3, \cdots$ finder vi den indlejrede Markovkæde, hvor tiden er diskret. Vi simulerer da ikke afstanden mellem to hændelser - den er altid een - men kun hvad slags hændelse, vi får næste gang.

Denne teknik skal vi se på i et kommende afsnit.

Chapter 6

Simulering af kø–systemer

En simuleringsmodel beskriver den tidsmæssige udvikling i et køsystem. Samspillet mellem struktur, strategi og trafik er indbygget i modellen, og datamaskinen registrerer den aggregerede opførsel som en dynamisk proces.

Ved tidstro simulering styres dette samspil af en *klokke*, der angiver det aktuelle tidspunkt i simulationen. Klokken deler tidsaksen op i en fortid og en fremtid. I en simulation antager denne en række værdier, der er ordnet kronologisk i en *kalender* (time-manager).

En af de væsentligste opgaver i en tidstro simulation er netop kronologisk at ordne de fremtidige hændelser i kalenderen, efterhånden som man får kendskab til dem.

Klokken kan opdateres på to måder.

- Ved *synkron simulation* (time-unit advance) forøges klokken med én tidsenhed ad gangen. Tids-skalaen er diskret (heltallige værdier), og alle tidspunkter indgår i kalenderen. Denne metode anvendes i kontinuerte simulations-sprog, hvor man beskriver forløbet af løsningen til et sæt sammenhørende differentialligninger. Metoden vil normalt introducere unøjagtigheder, og i de fleste tilfælde er den uegnet til diskret simulation. Med en fin inddeling af tidsaksen vil vi få mange tidspunkter uden hændelser, og med en grov inddeling kan der optræde multiple hændelser.

- Ved *asynkron simulation* (event-by event advance) består kalenderen kun af de tidspunkter, hvor der finder hændelser sted. Klokken springer derfor fra tidspunktet for én hændelse til tidspunktet for den næste hændelse, uanset hvor stor afstanden mellem disse er. Tids-skalaen er normalt kontinuert, og tidspunkter repræsenteres med reelle tal. I sammenligning med synkron simulation har denne metode flere fordele, og den anvendes i alle højere sprog til diskret simulation. Man anvender således en diskret tids-akse i kontinuerte simulationer og en kontinuert tids-akse i diskrete simulationer.

6.1 Simulation af GI/G/1 FCFS

Som et meget simpelt eksempel på en tidstro simulation vil vi simulere et køsystem med ét betjeningssted, en vilkårlig opkaldsproces, en vilkårlig betjeningstidsfordeling og en ordnet, ubegrænset kø. Vi er interesserede i de enkelte kunders ventetid. Dette system er vanskeligt at behandle analytisk, men ved at betragte Fig. 6.1 ser vi, at hvis den i'te kunde har ventetiden w_i og betjeningstiden s_i, og den $(i+1)$'te kundes interopkaldstid efter kunde i er a_i, så vil den $(i+1)$'te kunde få ventetiden:

$$w_{i+1} = \max\{0, w_i + s_i - a_i\}\,. \tag{6.1}$$

Vi kan derfor simulere dette system ved at generere tilfældige observationer fra opkaldstidsfordelingen (GI) og belægningstidsfordelingen (G) og beregne ventetiden for enhver kunde ud fra den foregående kundes ventetid. Simulationen starter eventuelt med et tomt system og afsluttes, når vi har det ønskede antal observationer.

Selv om det er en tidstro simulation, holder vi ikke regnskab med klokken, og vi har ingen kalender. Opkaldstidspunktet for den i'te kunde er:

$$T_i = \sum_{j=1}^{i} a_j\,.$$

Da vi allerede på opkaldstidspunktet kan beregne en kundes ventetid, er det unødvendigt at holde regnskab med antal kunder i systemet etc. Ønsker vi oplysninger om kølængde, eller har vi en vilkårlig anden kødisciplin (*LCFS*, *SIRO*, etc.), bliver simulationen væsentlig mere kompleks.

6.2 Simulation af GI/G/n–afvisningssystem

Vi skal nu se, hvorledes klokken anvendes til simulation af et fuldt tilgængeligt afvisningssystem med en vilkårlig opkaldsproces og en vilkårlig betjeningstidsfordeling. Dette er en generalisering af Erlang's klassiske afvisningssystem.

Ved simulation af dette system har vi dels brug for at vide, hvilke kanaler, der på et givet tidspunkt (klokkeslet) er optaget, dels hvornår optagne kanaler bliver ledige. Begge disse oplysninger kan kombineres til én enkelt, idet vi opretter en *statusbeskriver* for ledningsbundtet.

Statusbeskriveren er her en vektor $S(n)$ med n pladser. På den i'te plads noterer vi, hvornår den i'te ledning bliver ledig. Ved at sammenligne dette tidspunkt med klokken, kan vi altid afgøre, om en bestemt kanal er ledig eller optaget. Er klokken mindre end ophørstidspunktet, er kanalen optaget, ellers er den ledig.

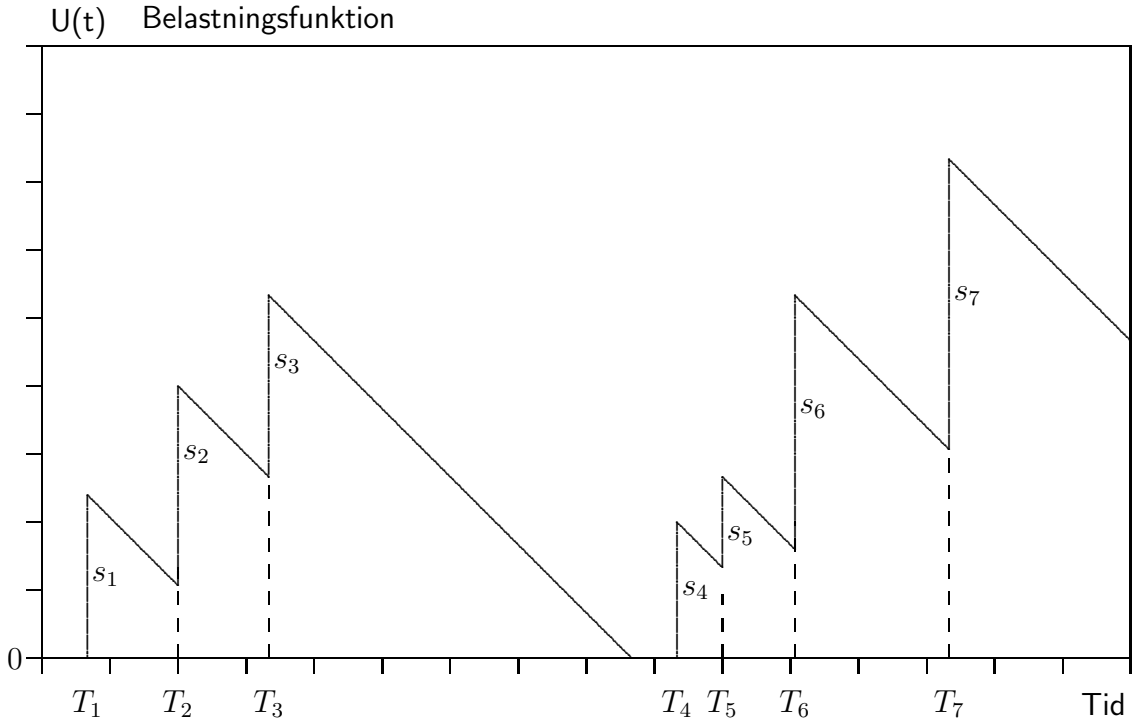


Figure 6.1: *Belastningsfunktionen for GI/G/1. Kaldes opkaldstiden* $a_i = T_{i+1} - T_i$*, fås at næste kundes ventetid* w_{i+1} *kan udtrykkes ved den foregående kundes ventetid* $w_{i+1} = \max\{0, w_i + s_i - a_1\}$*, hvor* s_i *er betjeningstiden af den i'te kunde.*

Ved simulationens start sættes klokken og alle pladser i $S(n)$ lig med nul. Et opkaldstidspunkt genereres ved at addere en opkaldsafstand til det foregående opkaldstidspunkt (bootstrapping). Klokken springer fra ét opkaldstidspunkt til det næste. Er der et ledigt betjeningssted, belægges det ved at notere ophørstidspunktet (= klokken + betjeningstiden) i den tilsvarende plads i statusbeskriveren. Er alle kanaler optagne, afvises opkaldsforsøget.

Vi foretager os ikke noget til ophørstidspunktet, og oplysningerne om de enkelte opkald opbevares kun så længe, de er nødvendige. Kalenderen består udelukkende af opkaldstidspunkter, som genereres i kronologisk rækkefølge. Vi undgår derfor sortering af tidspunkter i kalenderen.

6.3 Tidstro simulation af ventetidssystemer

Principielt er der ikke forskel på simulation af afvisningssystemer og ventetidssystemer. De sidstnævnte er imidlertid mere komplekse, og programmeringsmæssigt er det derfor nødvendigt med en systematisk struktur.

Denne strukturering sker automatisk i de fleste simulationssprog. Vi skal i dette afsnit il-

Figure 6.2: *Køsystem med vilkårlige trafikprocesser, n betjeningssteder, $k-n$ ventepladser og vilkårlig kødisciplin. En kunde får enten straksekspedition, ventetid eller bliver afvist.*

Figure 6.3: *Listeopbygningen ved tidstro simulation. Listen starter med hoved (H) og slutter med hale (T). Til højre er der indlagt en ekstra hjaendelse; Herved skifter to pegere værdi.*

lustrere de grundlæggende opgaver, som et simulations-sprog udfører, idet dette lettest gøres med et konkret eksempel.

Lad os betragte det køsystem, der er vist i Fig. 6.2 (*GI/G/n,k/*). Både opkaldstids- og betjeningstidsfordelingen er vilkårlige. Der er n betjeningssteder og $(k-n)$ ventepladser. Også kødisciplinen kan vælges vilkårligt. Dette system er meget generelt og kan ikke behandles analytisk.

Det aktuelle tidspunkt i simulationen er som sædvanlig givet ved en *klokke* og det totale antal kunder x i systemet (ventende eller under betjening) til dette tidspunkt er givet ved *systemets tilstand*.

Statusbeksriveren består nu af en række oplysninger om alle kunder, som ikke har forladt systemet. Det er ikke som i afsnit6.2 tilstrækkeligt blot at kende ophørstidspunktet. For hver kunde, der betjenes i systemet, har vi typisk brug for følgende Oplysninger:

- ankomsttidspunkt
- betjeningstid
- afgangstidspunkt
- prioritet
- peger

Når ankomsttidspunktet for en kunde kendes, reserveres der plads til disse oplysninger, der registreres efterhånden som de foreligger. Når en kunde forlader systemet, udnyttes de indsamlede oplysninger til statistikker (måleresultater), hvorefter de reserverede lagerpladser frigøres til en kunde, der ankommer engang i fremtiden.

Er k begrænset, er der behov for oplysninger om $k+1$ kunder, idet vi må kende mindst én fremtidig ankomst. Er k ubegrænset (rent ventetidssystem), vil der alligevel i realistiske systemer kun være behov for et endeligt antal kunder.

6.3.1 Listeopbygningen

I *kalenderen* organiseres de fremtidige hændelser i kronologisk orden. Dette kan gøres i en enkelt liste, men normalt er det naturligt at opdele kalenderen i flere lister, f.eks. e liste for fremtidige kunder, en liste for hver kø og en liste for hver gruppe af betjeningssteder. I vores tilfælde oprettes der en ankomstliste, en venteliste og en afgangsliste, svarende til, at en kunde i statusbeskriveren enten ankommer engang i fremtiden, venter i køen eller er under betjening. En kunde i statusbeskriveren vil i dette tilfælde således altid være medlem af netop én liste.

Inden for hver liste er kunderne sorteret i rækkefølge efter et bestemt kriterium. I *ankomstlisten* er dette ankomsttidspunktet. Nye kunder, der lægges ind i ankomstlisten, placeres derfor bagest i listen, medens kunder, der ankommer, tages ud forrest i listen.

I *afgangslisten* er kunderne tilsvarende sorteret efter afgangstidspunktet. Ankomst- og afgangsliste er *dynamiske* lister, idet de er ordnede efter tidspunkter, og indgår aktivt i kalenderen. *Ventelisten* er derimod en *statisk* liste. Her er kunderne ordnet i den rækkefølge, i hvilken de skal udvælges til ekspedition. Dette efhænger af den aktuelle kødisciplin. Det er simpelt at implementere alle de almindelige kødiscipliner, idet opkaldstidspunkt, betjeningstid og/eller prioritet umiddelbart kan indgå i kriteriefunktionen. Denne liste indgår passivt i kalenderen, idet der kun sker ændringer i forbindelse med hændelser i en af de andre lister.

Listerne består således af en række kunder, der er elementer i statusbeskriveren. Elementerne er ordnet i rækkefølge efter et bestemt kriterium. Det forreste element klades hovedet (H) og det sidste hale (T)(tail). Listen hægtes sammen ved hjælp af en peger (hægte, pointer), der peger på det næste element i listen. Halen markeres på en speciel måde, og rn variabel oplyser om hovedets adresses. Der må naturligvis tages højde for, at en liste kan være tom.

I vores tilfælde er en kunde kun medlem af én liste ad gangen, og vi behøver derfor kun at reservere plads til én peger i statusbeskriveren. Når et element skal indlægges i en liste, sker det ved at starte med hovedet og derefter løbe gennem listen, indtil man finder det nye elements plads. De relevante pegeres værdi ændres som illustreret i figur6.3. Detfremgår umiddelbart af samme figur, hvorledes et element fjernes fra en liste. I komplekse modeller kan en kunde være medlem af flere lister, og listerne kan eventuelt være dobbeltrettede.

6.4 Simulationens dynamiske forløb

En simulation består blot i manipulation af listerne. I vores køsystem er næste hændelse enten en ankomst eller en afgang.

6.4.1 Ankomst

Er afgangslisten tom, eller er tidspunktet i afgangslistens hovedelelement større end tidspunktet i ankomstlistens hovedelement, er næste hændelse en ankomst. Klokken opdateres til ankomst-tidspunktet, og systemets tilstand i undersøges og ajourføres:

- $i < n$: *Straksbetjening*. Der er et ledigt betjeningssted, og kunden fjernes fra ankomstlisten og indlægges i afgangslisten. $i := i + 1$.
- $n \leq i < k$: *Ventetid*. Der er en ledig venteplads. Kunden fjernes fra ankomstlisten og indlægges i ventelisten. $i := i + 1$.
- $i = k$: *Afvisning*. Alle systemets pladser er optagne. Kunden fjernes fra ankomstlisten og indlægges bagest i ankomstlisten, og i er uændret.

6.4.2 Afgang

I dette tilfælde opdateres klokke og systemtilstand. Oplysninger om det afgående opkald anvendes til statistik-indsamling, og kunden fjernes fra afgangslisten (og indlægges bagest i ankomstlisten, hvis det er et lukket køsystem).

Simulation af kø-netværk kan umiddelbart gennemføres, idet man opretter en overordnet liste, hvor alle dynamiske lister optræder med hovedelementet.

Ovenfor har vi angivet de grundlæggende principper. Den aktuelle implementering vil naturligvis afhænge af det anvendte programmeringssprogs muligheder.

Author index

Index