Något om slumptalsgenerering

Jan Enger och Gunnar Englund
Matematisk statistik
KTH

Ht 2001

1 Inledning

Ordet simulera betyder ju i vardagsspråket "låtsas", "efterhärma" eller "fuska" medan simulering i tekniska och matematiska sammanhang innebär att man ersätter verkligheten med en matematisk modell och utför experiment i denna. Monte Carlo-simulering innebär att lotta fram värden på stokastiska variabler för att ersätta komplicerade analytiska beräkningar av t.ex. fördelningen eller väntevärdet av en funktion av dessa stokastiska variabler.

Vi kommer att visa hur man kan konstruera slumptal från ett antal fördelningar och ge exempel på matlab-filer som genererar sådana slumptal. Bland annat kommer då följande matlabkommanden att användas,

`u=(X<p)`
där `X` är en vektor och `p` ett tal, ger en vektor av 0:or och 1:or. $\texttt{u}_\texttt{i}$ är 1 om $\texttt{X}_\texttt{i} < \texttt{p}$ och 0 annars.

`I=find(X)`
där `X` vektor, ger en vektor av de index för vilka $\texttt{X}_\texttt{i}$ är skild från 0. Till exempel ger `find(X<p)` som resultat en vektor med de index för vilka $\texttt{X}_\texttt{i} < \texttt{p}$

`ceil(x)` ger det minsta heltal större eller lika med `x`.

Hur `sort`, `sum` och `cumsum` fungerar på vektorer och matriser, se Matlabs `help`-funktion.

Matlabkoderna gör inga kontroller av att värden på parametrarna är korrekta, t.ex. att de är positiva om det är ett krav o.s.v. Matlabs egna slumpgeneratorer i modulen `Stats` gör ett flertal sådana test.

2 Slumptal

Vi önskar generera tal som uppför sig som oberoende utfall av stokastiska variabler med specificerad fördelning. Man skulle kunna tänka sig att som slumpmekanism ha något fysikaliskt fenomen som t ex radioaktivt sönderfall som enligt modern fysik är exempel på genuin slump, men vi kommer i stället

att använda oss av s k pseudo-slumptal dvs sekvenser som uppträder med "tillräcklig slumpmässighet".

En möjlighet är att använda decimalbråksutvecklingen i t.ex. $\pi = 3.1415926....$ Talen i decimalbråksutvecklingen 1, 4, 1, 5, 9 o.s.v. uppträder, enligt alla undersökningar man gjort, fullständigt slumpmässigt. Slumpmässigheten skall t.ex. innebära att talet 1 förekommer i långa loppet i en tiondel av fallen och samma för talen, 2,3,...,9 och 0. Inte nog med det. Tar vi talen parvis, finns 100 talpar $(0,0),(0,1),...,(9,9)$ och dessa talpar skall då uppträda var och en en gång av hundra i det långa loppet. I decimalbråksdelen av π börjar parserien med $(1,4)$, $(1,5)$, $(9,2)$. På liknande sätt om vi betraktar tripplar eller allmänt n-tupler.

Decimalbråksutveckling av ett tal görs i basen 10. Vi kan mycket väl använda en annan bas, t.e.x. 2. Ett tal kallas *normalt* om i bråkutvecklingen av talet, i vilken bas man än betraktar, varje tal i basen i långa loppet förekommer lika ofta. Utvecklingen av ett tal (som vi antar ligger mellan 0 och 1) i basen n är $0.a_1a_2... = \sum_{k=1}^{\infty} a_k/n^k$ där $0 \leq a_k < n,\ k = 1, 2, \ldots$. Normala tal skulle vara idealiska för slumptalsgenerering. Men nu har man följande något märkliga faktum.

Man kan visa, att om ett tal tas slumpmässigt i intervallet $(0,1)$ (rektangelfördelning R(0,1)), så är sannolikheten lika med ett, att talet är normalt, d.v.s. "nästan alla" tal är normala. Dock har man inte för ett enda givet tal kunnat visa att det faktiskt är normalt! Alla undersökningar av t.ex. π tyder på att det är normalt, men något bevis har man inte funnit. Däremot är det hur enkelt som helst att finna icke-normala tal! Ett rationellt tal kan inte vara normalt. Om talet är $\frac{m}{n}$ där $m \leq n$ är utvecklingen i basen n ändlig, om $m < n$ skrivs talet i basen n som $0.m$ och ett sådant tal kan inte vara normalt, det slutar med idel 0:or i utvecklingen. Å andra sidan är de rationella tal uppräkneliga, och av det följer att $P(X \text{ rationellt}) = \sum_{x_k \text{ rationellt}} P(X = x_k) = \sum_{k=1}^{\infty} 0 = 0$ om $X \in R(0,1)$. Ingen motsägelse finns mot det tidigare resultatet.

Om man nu utgår från t.ex. π för att konstruera slumptal, har man att problemet beräkna decimalbråkstalen. Man skall alltså finna en algoritm som genererar dessa på lämpligt sätt. Men en lämplig algoritm behöver ju inte nödvändigtvis vara en som genererar ett förmodat normalt tal. Vad man behöver är en algoritm som så snabbt som möjligt beräknar tal som uppträder till synes helt slumpmässigt. Nästan uteslutande används i praktiken kongruensalgoritmer av typen

$$x_{n+1} = (ax_n + b) \mod c$$

(där a, b och c är givna heltal) dvs den rest man får då man dividerar $ax_n + b$ med c. Man dividerar de erhållna x_n:en med c för att erhålla slumptal på intervallet $(0,1)$. Dessa algoritmer genomförs oerhört snabbt i en dator. Algoritmen kan utföras genom en skiftning av ettor och nollor i ett register eller minne. I en tidigare version av Matlab användes algoritmen $x_{n+1} = (7^7 x_n) \mod (2^{31} - 1)$ men algoritmen har förändrats mellan olika versioner.

Matlab levererar sådana pseudoslumptal med funktionen `rand` och enligt manualen för version 5 gäller att
"This generator can generate all the floating point numbers in the closed interval $[2^{-53}, 1-2^{-53}]$. Theoretically, it can generate over 2^{1492} values before repeating itself." Det anges dock inte vilken algoritm som egentligen används.

Genom att anropa funktionen med `rand(n,m)` får man en $n \times m$-matris av sådana pseudoslumptal.

Man kan även styra sekvensen av pseudoslumptal och kan därigenom upprepa exakt samma sekvens. Detta kan vara användbart om man önskar upprepa exakt samma simulering. Läsaren hänvisas till Matlabs manualer eller till online-hjälpen i Matlab.

Dessa pseudo-slumptal uppträder alltså som om de vore oberoende likformigt fördelade på intervallet $(0,1)$. Vi kommer inte att vara särskilt bekymrade över att de "egentligen" är deterministiska utan kommer att lita på att de har tillräcklig slumpmässighet vad gäller fördelning och oberoende för att tjäna våra syften.

3 Inversmetoden

Om man har tillgång till oberoende likformigt fördelade stokastiska variabler (dvs $R(0,1)$-fördelade) så kan man i princip generera vilken en-dimensionell fördelning som helst i enlighet med följande sats.

Sats Låt $F(x)$ vara en fördelningsfunktion och definiera "inversen"

$$F^{-1}(y) = \inf\{x : F(x) \geq y\},\ 0 \leq y \leq 1.$$

Om U är $R(0,1)$ och vi låter $X = F^{-1}(U)$ så gäller att $P(X \leq x) = F(x)$, dvs X har F till fördelningsfunktion. □

Inversen F^{-1} överensstämmer med den vanliga inversen om F är kontinuerlig strikt växande, men fungerar också om F har språng (dvs punktmassor i sannolikhetsfördelningen) eller är konstant på ett intervall (dvs det saknas massa på motsvarande intervall). Se figur 1 för en illustration.

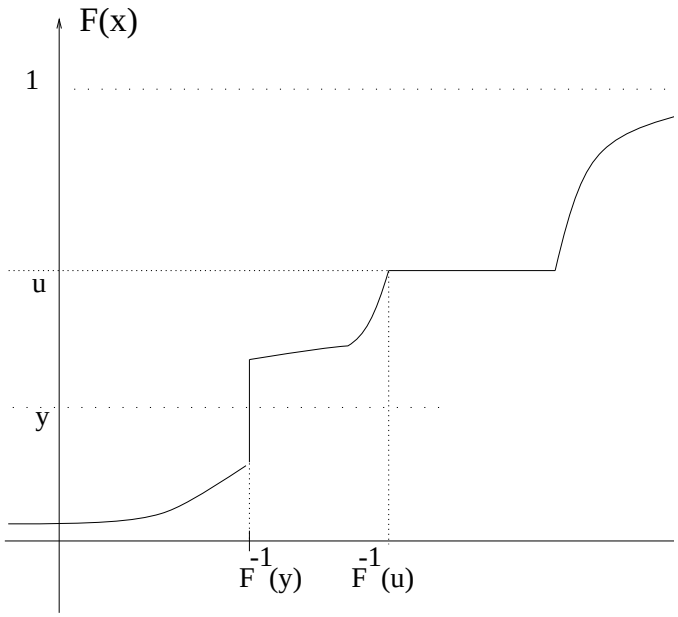
Bevis: Av definitionen framgår att $\{F^{-1}(y) > x\} = \{y > F(x)\}$ vilket ger

$$P(X > x) = P\left(F^{-1}(U) > x\right) = P(U > F(x)) = 1 - F(x)$$

dvs $P(X \leq x) = F(x)$ □

Exempel 3.1 Exponentialfördelningen $\text{Exp}(\theta)$ har fördelningsfunktionen

$$F(x) = 1 - e^{-x/\theta} \text{ för } x \geq 0.$$



Figur 1: Inversmetoden

Denna har inversen $F^{-1}(y) = -\theta \ln(1-y)$ och alltså kan vi se att

$$X = -\theta \ln(1-U) \text{ är Exp}(\theta) \text{ om } U \text{ är } R(0,1).$$

Eftersom även $1-U$ är $R(0,1)$ så gäller också att $X = -\theta \ln(U)$ är Exp(θ).□

Inte alla fördelningar har lättinverterade fördelningsfunktioner. Normalfördelningens $\Phi(x)$ är t ex inte så lätt att invertera, men då finns det andra knep som de som visas i avsnitt 5.8 på sidan 9. I Matlab:s standardmodul finns funktionen `randn` som genererar N(0,1)-fördelade slumptal.

4 Diskreta fördelningar

För diskreta fördelningar F som alltså lägger massan $P(X = x_k)$ i x_k för $k = 1, 2, \cdots$ kan man använda sig av att ta

$$F^{-1}(u) = x_j \text{ om } \sum_{k=1}^{j-1} P(X = x_k) \leq u < \sum_{k=1}^{j} P(X = x_k),\ j = 1, 2, \cdots$$

Exempel 4.1 Om $P(X = 1) = 0.4, P(X = 7) = 0.2$ och $P(X = 10) = 0.4$ kan man låta

$$X = F^{-1}(U) = \begin{cases} 1 & \text{om } 0 \leq U < 0.4 \\ 7 & \text{om } 0.4 \leq U < 0.6 \\ 10 & \text{om } 0.6 \leq U < 1 \end{cases}$$

□

Följande matlabkod ger ett slumptal från en godtycklig diskret fördelning. Parametern `p` skall vara en vektor med sannolikhetsfunktionens värden. Om fördelningen kan anta oändligt många värden, får man i praktiken trunkera den på lämpligt sätt. Vi antar att vektorn innehåller sannolikheterna för värdena 0,1,2,...

```
function y=randdiskr1(p)
% y=randdiskr(p) ger ett slumptal fran sannol.fkn p

P=cumsum(p); % fordelningsfunktionen
x=rand(1);
y=min(find(x<P))-1;
```

Om man på en gång vill simulera n stycken slumptal från en diskret fördelning kan man först generera n standardslumptal och sortera in dessa i fördelningsfunktionen. Följande matlabkod gör detta. Läsaren får själv fundera på hur koden fungerar.

```
function y=randdiskr2(n,p)
% y=diskret(n,p) ger n slumptal fran sann.fkn p

p=p(:)'; % omvandlar p till radvektor
P=cumsum(p); % fordelningsfunktionen
x=rand(1,n);
[xs Is]=sort(x);
z=[xs P];
[z I]=sort(z);
[z I]=sort(I); % inverterar I
z=I(1:n)-(1:n); % ger sorterade observationer
y=z(Is); % ger slumpmassig sortering
```

Exempel 4.2 Om X är Poissonfördelad Po(m), är $p(k) = P(X = k) = e^{-m}\frac{m^k}{k!}$. Väntevärde och varians är båda m. Värden större än $\max(20, m + 10\sqrt{m})$ antas med sannolikhet mindre än 10^{-10}. Vi ser att $p(k) = p(k-1)m/k$ vilket ger det enkelt att generera sannolikhetsfunktionen. En möjlig matlab kod är

```
function p=dpo(m)
% p=dpo(m) genererar sann.fkn for Po(m)

N=round(max(20,m+10*sqrt(m))); % ovre grans
z=log(m./(1:N));
lp=cumsum([-m z]); % ger logaritmerade sannolikheter
p=exp(lp);
```

Notera att k:te koordinaten i `p` motsvarar sannolikheten $p(k-1)$, eftersom den första sannolikheten skall vara $P(X = 0)$. Att vi först beräknat logaritmerade

sannolikheter beror på att e^{-m} för stora m är ett litet tal som sedan skall multipliceras med ett stort m^k, vilket kan ge numeriska problem. Stabilare då att logaritmera först. □

5 Slumptal från standardfördelningar

5.1 Rektangelfördelning

Om U är likformigt fördelad $R(0,1)$ så är $X = (b-a)\cdot U + a$ likformigt fördelad, $X \in R(a,b)$. En linjärtranslation av vanliga slumptal ger alltså slumpal från en rektangelfördelning. Följande matlabkod generar en radvektor av n sådana slumptal

```
function y=randr(n,a,b)
% randr(n,a,b) ger n R(a,b) slumptal

y=(b-a)*rand(1,n)+a;
```

5.2 Exponentialfördelning

Exponentialfördelningen betraktade vi i exempel 3.1. Inversmetoden ger på ett enkelt sätt exponentialfördelade slumptal från standardslumptalen. En matlabkod som ger en vektor av n slumptal från exponentialfördelning med väntevärde m:

```
function y=randexp(n,m)
% y=randexp(n,m) ger n Exp(m)-slumptal

u=rand(1,n);
y=-m*log(u);
```

5.3 Weibullfördelning

Weibullfördelningen är mycket viktig fördelning inom tillförlitlighetsteorin. Den kan ofta användas för att beskriva livslängder av komponenter. Den stokastiska variabeln X är Weibullfördelad med skalparameter a och formparameter c om

$$P(X > x) = \begin{cases} e^{-(x/a)^c} \text{ för } x \geq 0 \\ 0 \text{ annars} \end{cases} \tag{1}$$

Fördelningsfunktionen är således $P(X \leq x) = 1 - e^{-(x/a)^c}$. Denna kan lätt inverteras,

$$F^{-1}(x) = a(-\ln(1-x))^{1/c} \tag{2}$$

Det innebär att $a(-\ln(U))^{1/c}$ är Weibullfördelad om U är $R(0,1)$ (notera att om U är $R(0,1)$ så är $1-U$ också $R(0,1)$).
Matlabkod som ger vektor av n Weibullfördelade slumptal.

```
function y=randwei(n,a,c)
% y=randwei(n,a,c) ger n weibull-slumptal, skalpar a, formpar c

u=rand(1,n);
y=a*(-log(u)).^(1/c);
```

5.4 För första gången-fördelning

En för första gången fördelad stokastisk variabel X kan ses som antal gånger ett försök får upprepas till en viss händelse inträffar. Om sannolikheten att händelsen inträffar är p, så är $x \in \text{ffg}(p)$. Om U är $R(0,1)$ så inträffar händelsen $U < p$ med sannolikhet p. Vi kan alltså generera en följd av slumptal tills vi får ett som är mindre än p. Antalet genererade slumptal är då ffg(p). Matlabkod som generar ett sådant slumptal.

```
function y=randffg1(p)
% y=randffg1(p) ger ett ffg-fordelat slumptal

y=1;
x=rand(1);

while(x>p)
 x=rand(1);
 y=y+1;
end
```

Om p är litet kan man på detta sätt behöva generera många slumptal innan ett ffg-slumptal erhålles. En alternativ metod är följande.
Antag X är Exp(m). Låt $Y = [X] + 1$, där $[X]$ står för heltalsdelen av X. Då är för $k = 1, 2, \ldots$

$$P(Y=k) = P([X]+1=k) = P(k \le X+1 < k+1) = P(k-1 \le X < k) =$$
$$\int_{k-1}^{k} \frac{1}{m} e^{-x/m} dx = e^{-(k-1)/m} - e^{-k/m} = e^{-(k-1)/m}(1-e^{-1/m}) = pq^{k-1} \quad (3)$$

där $p = 1 - e^{-1/m}$ och $q = 1 - p = e^{-1/m}$. Med det innebär att Y är ffg(p). Utgår vi från p-värdet, skall m vara $-1/\ln(1-p)$. Av detta inses att följande matlabprogram genererar en vektor av n för första gången-fördelade slumptal.

```
function y=randffg2(n,p)
% y=randffg2(n,p) ger n ffg(p)-slumptal

m=-1/log(1-p);
x=randexp(n,m);
y=ceil(x);
```

5.5 Binomialfördelning

Om $U_1, U_2, \ldots, U_n$ är oberoende stokastiska 0–1 variabler som antar värdet 1 med sannolikhet p och värdet 0 med sannolikhet $q = 1 - p$ är $X = U_1 + U_2 + \cdots + U_n$ binomialfördelad Bin(n, p). Följande matlabkod utnyttjar detta för att ge en vektor av N binomialfördelade slumptal.

```
function y=randbin(N,n,p)
% y=randbin(N,n,p) ger N st Bin(n,p) slumptal

U=rand(n,N); % matris av slumptal
V=(U<p); % 0-1 matris
y=sum(V);
```

5.6 Gamma-fördelning

Gammafördelningen $\Gamma(n, a)$, där n är heltal kan ses som summan av n oberoende exponentialfördelade Exp$(1/a)$ slumpvariabler. Summerar man n exponentialfördelade slumptal, får man alltså ett gamma-fördelat slumptal. Detta åstadkommer matlabkoden

```
function y=randgamma(N,n,a)
% y=randgamma(N,n,a) ger N gamma(n,a) slumptal

x=-log(rand(n,N))/a; % ger nxN matris av exponentialfordelade slumptal
y=sum(x);
```

Resultatet blir en vektor av N gammafördelade slumptal.

5.7 Poissonfördelning

I avsnitt 4 visades hur poissonfördelade slumptal kunde åstadkommas. Här en annan metod. Antalet händelser i en Poissonprocess i tidsintervallet $(0, t)$ är poissonfördelat, Po(λt) om intensiteten är λ. Tiderna mellan händelserna är oberoende och $Exp(1/\lambda)$, se till exempel markovkompendiet. Låt nu $t = m$ och $\lambda = 1$. Då är antalet händelser i tidsintervallet Po(m). Därför genererar följande matlabkod Poissonfördelade slumptal.

```
function y=randpoiss(m)
% y=randpoiss(m) ger ett Po(m) slumptal

x=randexp(1,1);
y=0;
while x< m
  x=x+randexp(1,1);
  y=y+1;
end
```

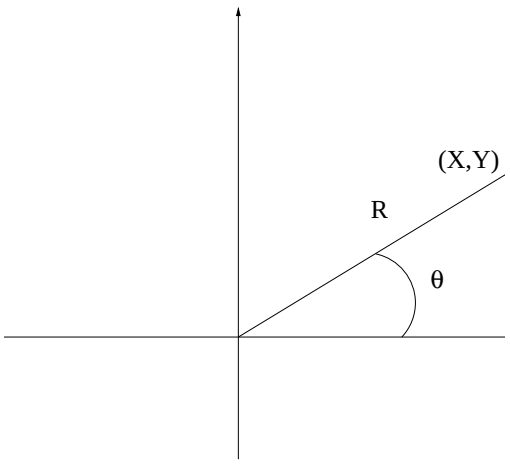
Koden simulerar en Poissonprocess och y räknar antalet händelser fram till tidpunkten m.

5.8 Normalfördelning

Som angavs ovan är normalfördelningen lite trasslig att simulera ifrån med hjälp av Inversmetoden, men det finns flera metoder att kringgå detta.

5.8.1 Box-Mullers metod

En vanlig metod att generera oberoende N$(0,1)$-fördelade slumptal är den s k Box-Mullers metod.



Figur 2: Box-Mullers metod

Antag att X och Y är oberoende N(0,1) och sätt

$$R = \sqrt{X^2 + Y^2}$$
$$\Theta = \arg(X, Y)$$

Vi har då

$$X = R\cos(\Theta),\ Y = R\sin(\Theta)$$

enligt figur 2.
Den tvådimensionella fördelningsfunktionen för (R, Θ) blir då för $u \geq 0$ och $0 \leq v \leq 2\pi$

$$F_{(R,\Theta)}(u,v) = P(R \leq u, \Theta \leq v) = P(\sqrt{X^2+Y^2} \leq u, \arg(X,Y) \leq v) =$$
$$\iint\limits_{\substack{\sqrt{x^2+y^2}\leq u \\ \arg(x,y)\leq v}} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \frac{1}{\sqrt{2\pi}} e^{-y^2/2} dxdy = \iint\limits_{\substack{\sqrt{x^2+y^2}\leq u \\ \arg(x,y)\leq v}} \frac{1}{2\pi} e^{-(x^2+y^2)/2} dxdy = (\text{polära}$$
$$\text{koordinater}) = \iint\limits_{\substack{0\leq r\leq u \\ 0\leq\theta\leq v}} \frac{1}{2\pi} r e^{-r^2/2} dr d\theta = \int_0^u r e^{-r/2} dr \int_0^v \frac{1}{2\pi} d\theta = (1 - e^{-u^2/2}) \frac{v}{2\pi}.$$

Nu blir det lätt att beräkna fördelningen för R och Θ.

$$F_R(u) = P(R \leq u) = P(R \leq u, \Theta \leq 2\pi) = 1 - e^{-u^2/2}$$
$$F_\Theta(v) = P(\Theta \leq v) = P(R < \infty, \Theta \leq v) = \frac{v}{2\pi}.$$

Fördelningen för Θ är således rektangelfördelad, R$(0, 2\pi)$, och R:s fördelning kallas Rayleigh-fördelningen. Vi ser dessutom att den tvådimensionella fördelningsfunktionen är produkten av de endimensionella, varför R och Θ är oberoende. Fördelningsfunktionen för R är lätt inverterbar. Vi får

$$F_R^{-1}(y) = \sqrt{-2\ln(1-y)}.$$

Rayleighfördelningen är för övrigt identisk med en Weibullfördelning med formparameter 2 och skalparameter $\sqrt{2}$.
Alltså kan vi generera två stycken oberoende N(0,1)-fördelade variabler X och Y ur två stycken $R(0,1)$-fördelade variabler U_1 och U_2 genom

$$X = \cos(2\pi U_1)\sqrt{-2\ln(1-U_2)}$$

$$Y = \sin(2\pi U_1)\sqrt{-2\ln(1-U_2)}$$

Här kan vi ersätta $1 - U_2$ med U_2 om vi så vill!

Om vi vill generera slumptal från N(m, σ) skaffar vi oss N$(0, 1)$-fördelade slumptal Z metod och bildar sen $X = m + \sigma Z$ som ju får avsedd fördelning.
Matlabkod:

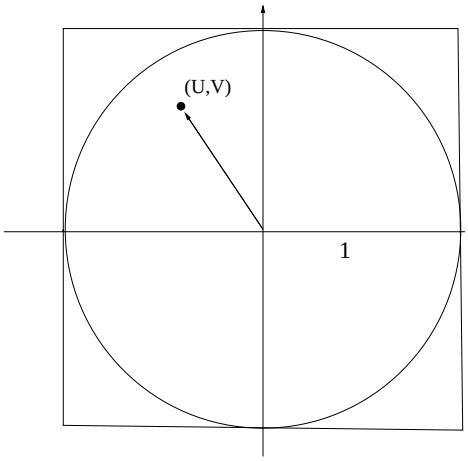
```
function y=randnorm1(m,s)
% y=randnorm1(m,s) ger 2 N(m,s) slumptal

u=rand(1,2);
z(1)=cos(2*pi*u(1))*sqrt(-2*log(u(2)));
z(2)=sin(2*pi*u(1))*sqrt(-2*log(u(2)));
y=m+s*z;
```

Koden ger två N(m, s)-fördelade slumptal.

5.8.2 Alternativ metod

En alternativ metod att generera normalfördelade slumptal är att först välja U och V som oberoende likformigt fördelade på intervallet $[-1, 1]$. Om de hamnar inom den i kvadraten inskrivna cirkeln enligt figur 3 har man därigenom genererat en likformigt fördelad slumpmässig riktning.
Skulle (U, V) hamna utanför cirkeln, vilket innebär att $U^2 + V^2 > 1$ får man lotta fram nya värden på U och V tills man fått en punkt inne i cirkeln. Man väljer sedan en radie R i enlighet med Rayleigh-fördelningen och flyttar sig på strålen genom $(0, 0)$ och (U, V) så att punktens avstånd blir R. Den sålunda genererade punkten (X, Y) har oberoende N$(0, 1)$-fördelade koordinater X och



Figur 3: Metod för att få likformig slumpmässig riktning.

Y. Eftersom cirkelns area är π och kvadratens är 4 måste man generera ett ffg$(\pi/4) \approx$ ffg(0.7854)-fördelat antal (U, V)-par. Fördelen med detta förfarande är att man inte behöver beräkna cosinus och sinus för vinkeln Θ som i Box-Mullers metod vilket kan snabba upp simuleringen.

Följande matlabkod ger två slumptal, N(m, s).

```
function y=randnorm2(m,s)
% y=randnorm2(m,s) ger 2 N(m,s) slumptal

U=2*rand(1)-1;
V=2*rand(1)-1;
while U^2+V^2>1
  U=2*rand(1)-1;
  V=2*rand(1)-1;
end

R=sqrt(-2*log(rand(1)));
x=[U V]*R/sqrt(U^2+V^2);
y=m+s*x;
```

5.8.3 Centrala gränsvärdessatsen-metod

Enligt centrala gränsvärdessatsen är $Y = U_1 + U_2 + \cdots + U_{12}$ approximativt normalfördelad om $U_1, U_2, \ldots, U_{12}$ är oberoende och alla $R(0, 1)$. Eftersom $E(U_i) = 0.5$ och $V(U_i) = 1/12$ är $X = Y - 6$ approximativt normalfördelad, N$(0, 1)$. Denna approximation är ganska god och ger oss möjlighet att åstadkomma n stycken approximativt normalfördelade, N(m, s), slumptal med följande matlabkod.

```
function y=randnorm3(n,m,s)
% y=randnorm3(n,m,s) ger n approx N(m,s) slumptal

u=rand(12,n);
x=sum(u)-6;
y=m+s*x;
```

6 Urval ur ändliga populationer

I många sammanhang har man anledning att simulera dragning ur ändliga populationer, t.ex. då man vill simulera urvalsundersökningar eller kortspelsproblem. Antag att populationen utgörs av elementen i vektorn $a_1, a_2, \ldots, a_N$ och att n skall tas ut utan återläggning. Det kan göras genom att lotta ut indexen $i_1, i_2, \ldots, i_n$ för elementen som skall dras. Generera därför en vektor om N slumptal och ta indexen för de n minsta talen i denna slumpvektor. I matlab görs detta enkelt genom anropet `[y I]=sort(x)`. Det ger som resultat en vektor `y` som är `x` sorterad i storleksordning och en indexvektor `I` som anger vilken ordning de sorterade elementen utsprungligen kom i `x`.
Kod för att plocka ut n tal (index) slumpmässigt ur talen (indexen) $1, 2, \ldots, N$.

```
function y=randurval(n,N)
% y=randurval(n,N) ger n slumpmassiga tal ur 1,2,...,N

x=rand(1,N);
[xs I]=sort(x);
y=I(1:n);
```

Idén kan användas för att erhålla slumptal från en hypergeometrisk fördelning. Om vi låter X vara antalet element som är mindre eller lika med v vid dragning av n element ur talen 1 t.o.m N är X Hyp$(N, n, v/N)$. Följande matlabkod simulerar X.

```
function y=randhyp(N,n,p)
% y=randhyp(N,n,p) ger ett slumptal fran Hyp(N,n,p)

v=round(N*p); % v skall vara heltal
x=rand(1,N);
[xs I]=sort(x);
y=sum(I(1:n)<=v);
```

Exempel 6.1 I kortspelet bridge räknar man honnörspoäng på så sätt att de fyra essen erhåller 4 poäng var, de fyra kungarna 3 poäng var, damerna 2 poäng och knektarna 1 poäng var. De övriga trettiosex korten erhåller 0 poäng. Var och en av spelarna erhåller 13 kort. Låt oss simulera totala honnörspoängen hos en spelare. Vi skall alltså plocka ut tretton element slumpmässigt ur följden $0, 0, \ldots, 0, 1, 1, 1, 1, 2, \ldots, 4, 4$ och summera poängen. Matlabkoden nedan åstadkommer detta.

```
kort=[4 4 4 4 3 3 3 3 2 2 2 2 1 1 1 1 zeros(1,36)];
I=randurval(13,52);
hp=sum(kort(I))
```

□

7 Matlabs slumptal från standardfördelningar

Standardmodulen i Matlab har funktionerna `rand` och `randn` för $R(0,1)$ respektive den standardiserade normalfördelningen.

Matlabs `Stats`-modul (finns ej i KTH-licensen) har en hel uppsättning slumptalsgeneratorer för olika fördelningar (både diskreta och kontinuerliga):

```
Random Number Generators.
betarnd     - Beta random numbers.
binornd     - Binomial random numbers.
chi2rnd     - Chi square random numbers.
exprnd      - Exponential random numbers.
frnd        - F random numbers.
gamrnd      - Gamma random numbers.
geornd      - Geometric random numbers.
hygernd     - Hypergeometric random numbers.
lognrnd     - Lognormal random numbers.
mvnrnd      - Multivariate normal random numbers.
nbinrnd     - Negative binomial random numbers.
ncfrnd      - Noncentral F random numbers.
nctrnd      - Noncentral t random numbers.
ncx2rnd     - Noncentral Chi-square random numbers.
normrnd     - Normal (Gaussian) random numbers.
poissrnd    - Poisson random numbers.
random      - Random numbers from specified distribution.
raylrnd     - Rayleigh random numbers.
trnd        - T random numbers.
unidrnd     - Discrete uniform random numbers.
unifrnd     - Uniform random numbers.
weibrnd     - Weibull random numbers.
```