Introduksjon (5) til R

Halvor Aarnes

Innhold

Slumptallsgenerator

Slumptall er tilfeldige tall generert fra en statistisk fordeling. En ekte slumptallsgenerator er terningkast, men i datamaskiner er det uekte slumptallsgeneratorer (PRNG- «pseudorandom number generator») som lager pseudoslumptall (pseudorandome tall) basert på en algoritme. Slumptall benyttes i Monte Carlo simuleringer og innen kryptografi. At tallene virkelig er tilfeldige, og ikke følger et mønster er derfor av avgjørende betydning, noe som John von Neumann påpekte. I R kan man velge pseudoslumptall fra forskjellige statistiske fordelinger for eksempel **`rnorm()`** fra normalfordelingen, **`runif()`** fra uniform fordeling, rchisq() fra kjikvadratfordelingen osv. Generering av slumptallene blir styrt av en funksjon **`.Random.seed`**. Ved R-kommandoen **`set.seed()`** kan man få en slumptallsgenerator til å starte fra samme sted, noe som gjør at man får samme resultat hver gang, men genererer slumptall. Som utgangsmetode i R benyttes Mersenne twister, utviklet av Makota Matsumoto og Tauji Nishimura i 1998, hvor perioden for gjentakelse er 2^{19937}-1 iterasjoner, tilnærmet lik $4.32 \cdot 10^{6001,}$ 32-bits tall jevnt fordelt i 623 dimensjoner basert seg på en lineær kongruent generator kombinert med primtallene. jfr. Mersenne-primtall 2^n-1. Innen kryptografi må man ha kryptografisikre pseudoslumptallsgeneratorer.
Målet for en simulering er å gjenta en enkel prosedyre tusenvis av ganger, som erstatning for mer komplekse beregninger, og som det i mange tilfeller også er umulig å utføre.
Pseudorandome tall vil i praktisk bruk oppføre seg omtrent som tilfeldige tall.

```
?.Random.seed
```

Vi plukker ut 5 tilfeldige store bokstaver. Gjenta kommandoen og se at det blir forskjellige bokstavkombinasjoner:

```
sample(LETTERS, 5)
```

```
[1] "X" "E" "D" "J" "T"
```

Vi definerer startpunkt med **`set.seed()`**, og ved å gjenta **`sample()`** ser man at resultatet blir det samme hver gang, men bare når du har brukt **`set.seed()`** hver gang:

```
set.seed(20)
```

```
sample(LETTERS, 5)
```

```
set.seed(25)
```

```
sample(1:100, 8)
```

```
[1] 42 69 15 88 12 94 59 32
```

```
set.seed(25)
```

```
sample(1:100, 8)
```

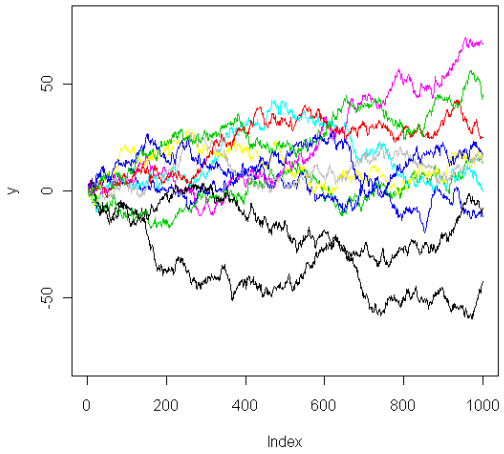
```
[1] 42 69 15 88 12 94 59 32
```

```
sample(1:100, 8)
```

Brownske bevegelser og virrevandring (random walk)

Hva blir summen av tallene 1 til 10, eller 1 til 100 ? Ifølge myten fikk Gauss dette spørsmålet fra sin lærer, som trodde han skulle holde sin elev opptatt en stund, men svaret kom sporenstreks 5050. Gauss tenkte slik ved å legge sammen fra begge ender: 99+1, 98+2, 97+3 osv.

```
cumsum(1:10) #kumulativ sum
[1]  1  3  6 10 15 21 28 36 45 55
cumsum(1:100)

#Virrevandring
set.seed(30)
n <- 1000
x<- rnorm(n) # n tall fra standard normalfordeling
y <- cumsum(x)
plot(y, type="l", ylim =c(-80, 80))
for (i in 1:10){
     x <- rnorm(n)
     y <- cumsum(x)
     lines(y, type="l", col = 2 + i)
     }
```

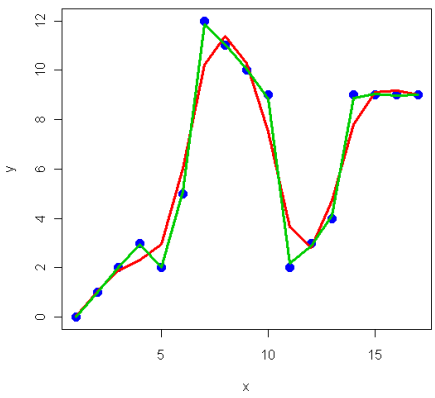


Kurveglatting og splines

Splines er n-te gradspolynomer brukt til stykkevis kurveglatting. Der hvor polynomene møtes er det knuter, og flere knuter gir økt tilpasning. En spline er et bøybart og fleksibelt metallstykke som brukes av håndverkere til å trekke kurvelinjer. Kubiske spline er tredjegradapolynomer for eksempel kubisk B-spline, kubisk Bézier spline

Vi lager noen tilfeldige y-verdier og bruker deretter **`smooth.spline()`** til kurveglatting:

```
y <- c(0:3, 2, 5, 12:9, 2:4, rep(9,4))
x <- seq(1,length(y))
plot(x, y, col = 4, pch = 16, cex= 1.5)
s1 <- smooth.spline(y); s1
s2 <- smooth.spline(y, spar = 0.1);s2
lines(s1, lwd = 3, col= 2)
lines(predict(s2, x), col = 3, lwd= 3)
```



Taylorpolynomer og feilanalyse

Mange ganger finner man ikke eksakte løsninger fra funksjoner, men må klare seg med tilpassete løsninger som blir riktige innen toleransegrenser for feil. Tangenten til en funksjon y = f(x) ved punktet x = a viser hvordan grafen oppfører seg nær punktet (a, f(a)).

I numerisk analyse kan man bruke enkle funksjoner, vanligvis polynomer (P_n), som tilpasning (approksimasjon) til en gitt funksjon *f(x)* ved punktet x = a.

I sin enkleste form kan vi lage en lineær tilpasning med et polynom av første grad (P_1), som blir lik tangentlinjen:

$$P_1(x) = f(a) + f'(a)(x - a)$$

hvor vi har samme funksjon og derivert:

$$P_1(a) = f(a) \qquad P_1'(a) = f'(a)$$

Errorfunksjonen *E(x)* til denne tilpasningen er:

$$E(x) = f(x) - P_1(x) = f(x) - \ f(a) + f'(a)(x - a)$$

og feilen blir minst når *x* er i nærheten av *a*.

Minner om definisjonen av den deriverte, som er lik tangenten og sier noe som stigningen til kurven i tangeringspunktet:

$$f'(a) = \lim_{x \to a} \frac{f(x) - f(a)}{x - a}$$

noe som gir tilnæringen hvor x ligger nær a:

$$f'(a) \approx \frac{f(x) - f(a)}{x - a}$$
$$f(x) \approx f(a) + f'(a)(x - a)$$

En bedre tilpasning får vi med et polynom av andre grad:

$$P_2(x) = f(a) + f'(a)(x - a) + \frac{f''(a)}{2}(x - a)^2$$

Generelt har vi et Taylorpolynom med grad *n* for den n-deriverbare funksjonen *f(x)* ved x=a gitt som

$$P_n(x) = f(a) + \frac{f'(a)}{1!}(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \frac{f'''(a)}{3!}(x - a)^3 + \cdots \frac{f^n(a)}{n!}(x - a)^n$$

Som nevnt finner man ofte ikke eksakte løsninger, men nedenfor er det brukt eksempler med kjente funksjoner som viser prinsippet for Taylorpolynomer. Navn etter den engelske matematiker Brook Taylor (1685-1731).

Vi skal sammenligne Taylorpolynom av grad grad 3 for logaritmefunksjonen ved x=1:

$$f(x) = \ln(x)$$

x = 1:

```
f <- function (x){log(x)}
curve(f, 0.01, 4, col = 4, lwd = 3)
```

Finner den førstederiverte og lager en funksjon av den:

```
df1 <- D(expression(log(x)), "x"); df1
1/x
df1dx <- function(x) {eval(df1)}
```

Finner den andrederiverte ved å derivere den førstederiverte, og lager en funksjon av den andrederiverte.

```
df2 <- D(expression(1/x), "x"); df2
-(1/x^2)
df2dx <- function(x) {eval(df2)}
```

Finner den tredjederiverte ved å derivere den andrederiverte og lager en funksjon av den tredjederiverte:

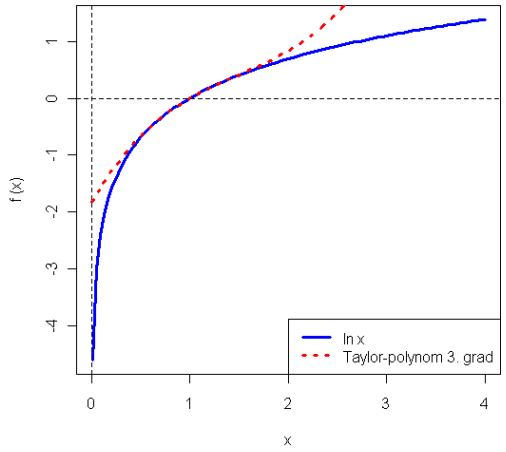
```
df3 <- D(expression(-(1/x^2)), "x"); df3
2 * x/(x^2)^2
df3dx <- function(x) {eval(df3)}
```

Setter inn verdien hvor vi skal beregne Taylorpolynom:

```
a <- 1
```

Regner ut Taylorpolynomet tredje grad , og trekker en linje for denne, og ser at polynomet følger deler av funksjonen.

```
taylor <- function(x) f(a)+df1dx(a)*
     (x-a)+(df2dx(a)/factorial(2))*(x-a)^2+
     (df3dx(a)/factorial(3))*(x-a)^3

curve(taylor, 0, 4, col = 2, lty = 3, lwd = 3, add = TRUE)
abline (v=0, h=0, lty = 2)
legend("bottomright", c("ln x", "Taylor-polynom 3. grad"),
     col=c(4,2), lty=c(1,3), lwd= 3)
```



McLaurin rekke

Hvis a = 0 har vi en egen utgave av Taylor-rekken som kalles MacLaurin-rekke (Colin Mac Laurin (1698-1746):

$$P_n(x) = f(0) + \frac{f'(0)}{1!}(x) + \frac{f''(0)}{2!}(x)^2 + \frac{f'''(0)}{3!}(x)^3 + \cdots \frac{f^n(0)}{n!}(x)^n$$

Hvis vi har funksjonen:

$$f(x) = sinx$$

så blir McLaurin-rekken:

$$sin(x) = f(0) + \frac{f'(0)}{1!}(x) + \frac{f''(0)}{2!}(x)^2 + \frac{f'''(0)}{3!}(x)^3 + \cdots \frac{f^n(0)}{n!}(x)^n$$

Vi definerer funksjonen f og finner deretter førstederiverte (vær oppmerksom på at når man bruker samme objektnavn som brukt tidligere forsvinner det forrige inneholdet og erstattes med det nye)

```
f <- function(x) sin(x)

df1 <- D(expression(sin(x)), "x"); df1
cos(x)
df1dx <- function(x) {eval(df1)}
```

Deretter lager vi en funksjon som inneholder den førstederiverte og finner deretter den andrederiverte:

```
df2 <- D(expression(cos(x)),"x"); df2
-sin(x)
df2dx <- function(x) {eval(df2)}
```

Deretter finner vi den tredederiverte:

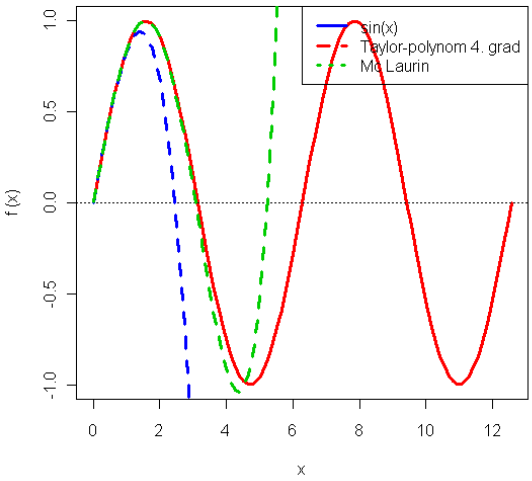
```
df3 <- D(expression(-sin(x)), "x"); df3
-cos(x)
df3dx <- function(x) {eval(df3)}
```

Og den fjerdederiverte:

```
df4 <- D(expression(-cos(x)),"x"); df4
sin(x)
df4dx <- function(x) {eval(df4)}
```

Deretter finner vi verdiene for de enkelte leddene I rekken og bruker **`factorial()`** for fakultetsfunksjonen n!:

```
a <- 0

taylor2 <- function(x) f(a)+df1dx(a)*
     (x-a)+(df2dx(a)/factorial(2))*(x-a)^2+
     (df3dx(a)/factorial(3))*(x-a)^3 +
     (df4dx(a)/factorial(4))*(x-a)^4
```



Vi har altså for sin(x):

$$sin(x) = f(0) + \frac{f'(0)}{1!}(x) + \frac{f''(0)}{2!}(x)^2 + \frac{f'''(0)}{3!}(x)^3 + \cdots \frac{f^n(0)}{n!}(x)^n$$

Vi får da den velkjente rekken for sin(x) når vi setter inn for verdiene:

$$sin(x) = x - \frac{x^3}{3!} + \cdots$$

Lineær kongruent generator

Slumptall kan bli laget via en lineær kongruent generator:

$$x_{i+1} = ax_i + b(mod\ d) \qquad a > 0, b \geq 0, d > 0, i = 1, 2, 3, \ldots$$

hvor a er multiplikator, b er økningen og d er modulus.

Vi kan lage n = 50 slumptall fra en slik slumptallsgenerator (RNG) hvor man velger fra tallene fra 0-100

Setter inn tilfeldig verdi for multiplikator, økning og modulus (periode). Legg merke til at tallene starter ved frøet (seed) s = 17.

```
a <- 20
b <- 3
d <- 101
s <- 17
n <- 50
```

Lager et objekt X for lagring av slumptallene, og setter startverdien lik s:

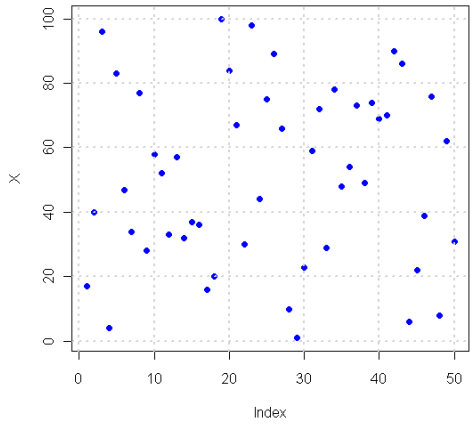
```
X <- numeric(n)

X[1] <- s
```

Lager en for-løkke som lager neste tall ifølge formelen for lineær kongruent generator:

```
for (i in 1:(n-1)) X[i+1] <- (a*X[i]+b)%%d
```

Vi kan lage et plot av de 50 slumptallene:
Det er ikke alltid lett å bedømme om disse tallene forekommer tilfeldig eller ikke.

```
plot(X, pch = 16, col = 4)
grid(lwd = 2)
```

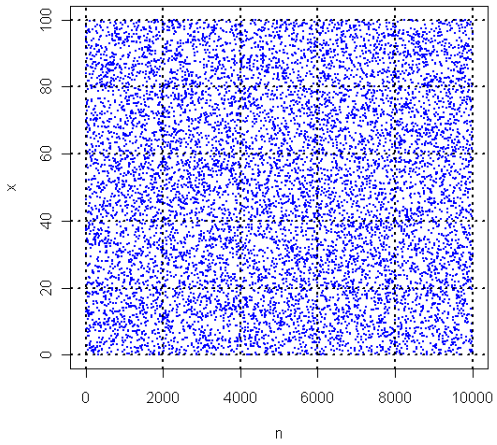



```
table(X)
x
  1   4   6   8  10  16  17  20  22  23  28  29  30  31  32  33  34  36  37  39
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
 40  44  47  48  49  52  54  57  58  59  62  66  67  69  70  72  73  74  75  76
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
 77  78  83  84  86  89  90  96  98 100
  1   1   1   1   1   1   1   1   1   1
```

Forsøk å endre parameterverdien i denne enkle generatoren og se hva som skjer. Verdien av d bør være mye større enn n.

Vi endrer parameterverdiene og plukker ut n = 10000 tilfeldige tall mellom 0-100 med denne generatoren. Lager et plot og et histogram

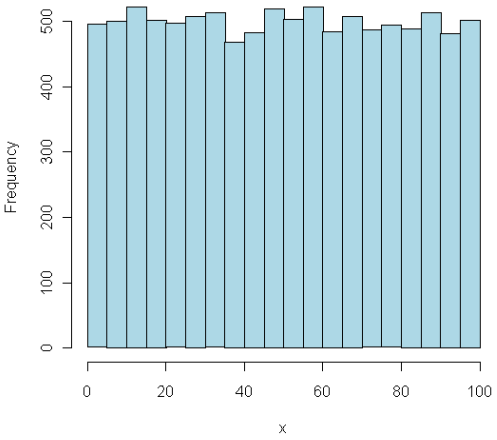
```
a <- 789
b <- 16784
d <- 65347
s <- 17
n <- 10000
X <- numeric(n)
X[1] <- s
for (i in 1:(n-1)) X[i+1] <- (a*X[i]+b)%%d
plot(X/d*100, pch = 16, cex = 0.1, col = 4,
     xlab = "n", ylab = "x")
grid(lwd = 2, col = 1)
```



Et histogram viser fordeling av tilfeldige tall 0-100:

```
hist(X/d*100, xlab = "x", main = "",col="lightblue")
```

Det ser ut til at tallene 0-100 trekkes ut like ofte, men allikevel kan det finnes et mønster i tallene, det vil si de er ikke ordentlige slumptall.



HAa/2015