732A90_ExamMaterial

Me

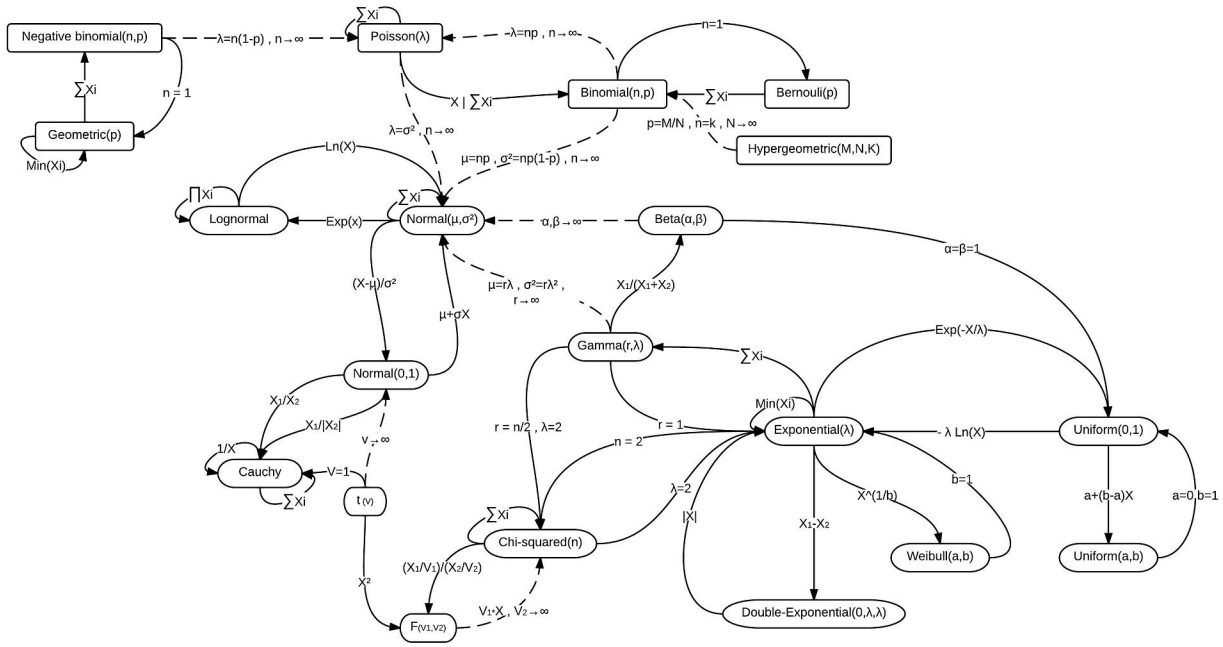
12 March, 2019

Libraries

```
library("ggplot2")
library("dplyr")
library("boot")
library("knitr")
```

Distributions

Relationship between distributions



Bernoulli Distribution

- PDF of Bernoulli

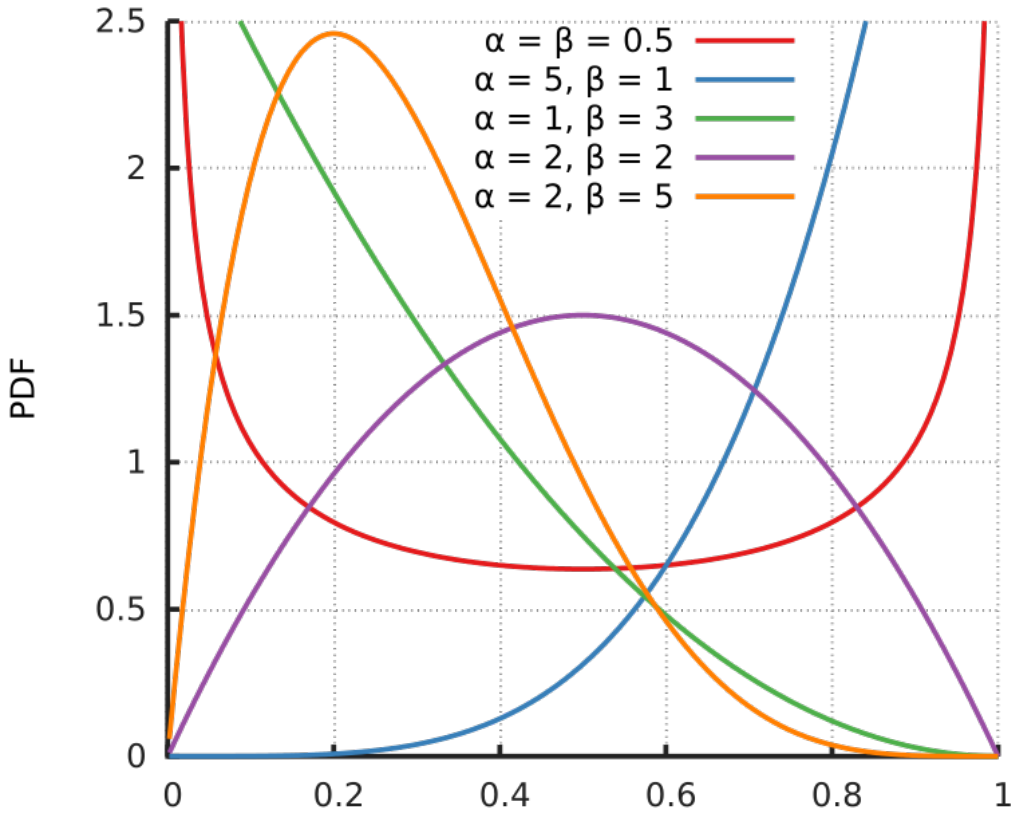
$$q = 1 - p \quad if \ k = 0$$
$$q = p \quad if \ k = 1$$

+ p is the probability of success + q is the probability of failure, q=1-p

- CDF of Bernoulli

$$X(m,n) = \left\{ \begin{array}{ll} 0, & \text{if } k < 1 \\ 1-p, & \text{if } 0 \leq k < 1 \\ 1, & \text{if } k \geq 1 \end{array} \right\}$$

Beta Distribution

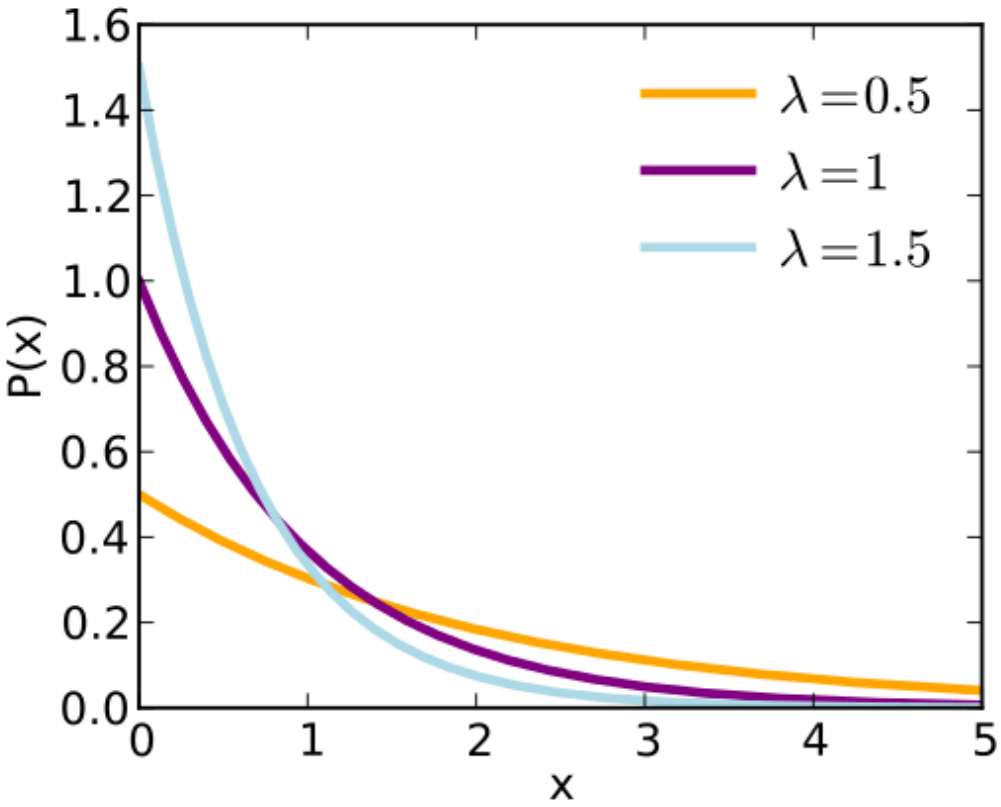


- PDF of Beta

$$\Gamma(n) = (n-1)!$$

$$f(x;\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$$

Exponential distribution



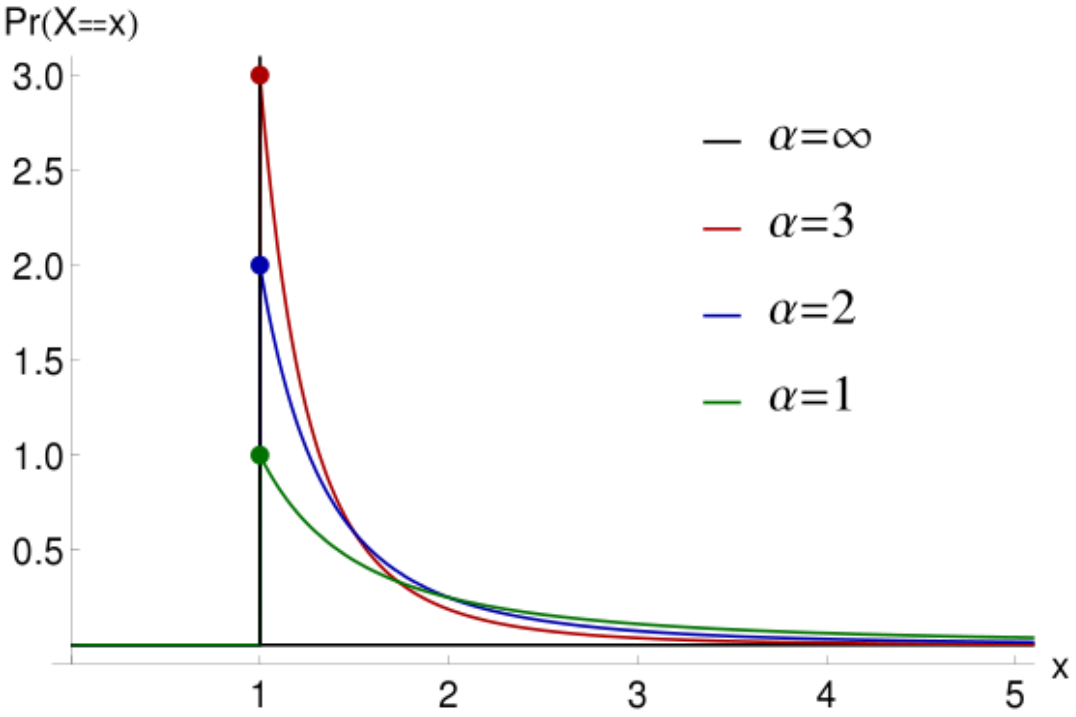
- PDF of Exponential
  - lambda is the rate parameter

$$f(x;\lambda) = \left\{ \begin{array}{ll} \lambda e^{-\lambda \cdot x}, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{array} \right\}$$

- CDF of Exponential

$$F(x;\lambda) = 1 - e^{-\lambda \cdot x}$$

Pareto Distribution



- PDF of Pareto
  - xm is the (necessarily positive) minimum possible value of X
  - alpha is a positive parameter

$$f(x) = \left\{ \begin{array}{ll} \frac{\alpha x_m^\alpha}{x^{\alpha+1}}, & \text{if } x \geq x_m \\ 0, & \text{if } x < x_m \end{array} \right\}$$

- CDF of Pareto

$$F(x) = 1 - (\frac{x_m}{x})^\alpha$$

Uniform Distribution

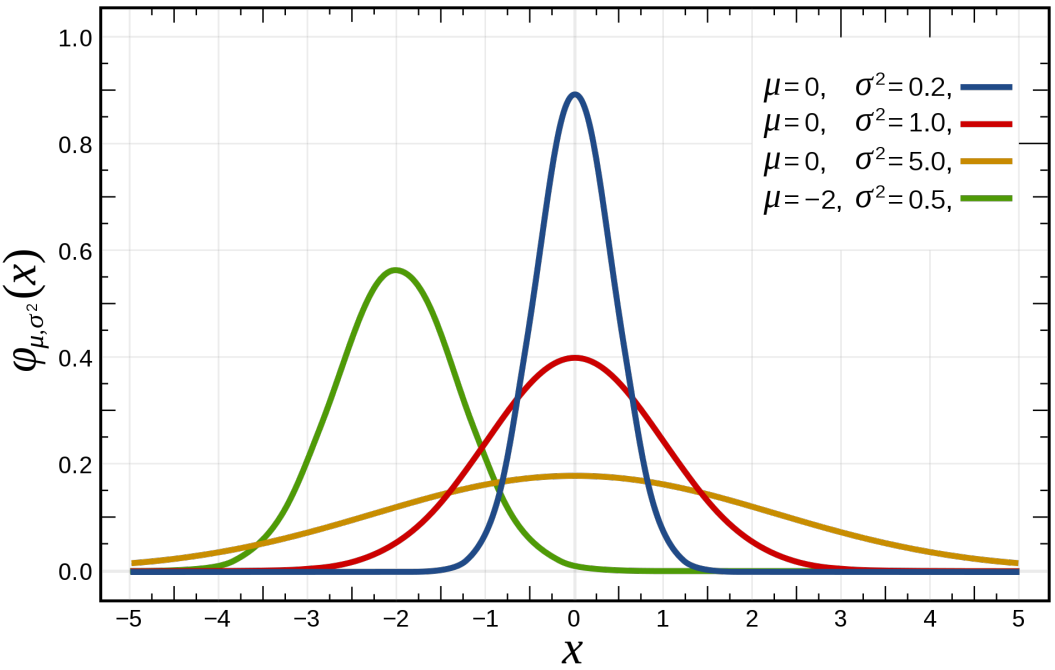
- PDF of Uniform

$$f(x) = \left\{ \begin{array}{ll} \frac{1}{b-a}, & \text{if } a \leq x \leq b, \\ 0 & \text{otherwise} \end{array} \right\}$$

- CDF of Uniform

$$F(x) = \left\{ \begin{array}{ll} 0, & \text{if } x < a \\ \frac{x-a}{b-a}, & \text{if } a \leq x \leq b \\ 1 & \text{otherwise} \end{array} \right\}$$

Normal Distribution



- PDF of normal
  - mu is the mean or expectation of the distribution (and also its median and mode)
  - sigma is the standard deviation
  - sigma squares is the variance

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Extra suggestions

Run the following to get different distributions and their formula

```
help("Distributions")
```

Random Sampling from Uniform distribution

Sampling based probabilities proportional to the number of inhabitants of the city

```
data <- read.csv2("population.csv", header = TRUE)
```

```
get_city <- function(data){

data$Municipality <- as.character(data$Municipality)
data$prob <- data$Population/sum(data$Population)
#sorting the dataset
```

```
data <- data %>% arrange(prob)

data$cum_prob <- cumsum(data$prob)
data$lead_cum_prob <- lead(data$cum_prob, n=1)

# filling NA
data$lag_cum_prob[1] <- 0
data$lead_cum_prob[NROW(data)] <- 1

set.seed(12345)
num <- runif(1,0,1)

X <- ifelse(((num >= data$cum_prob) & (num <=  data$lead_cum_prob)), row.names(data), NA)
X <- na.omit(X)

data_name <- data[row.names(data) %in% X,]
return(data_name$Municipality)
}
```

```
data$Municipality <- as.character(data$Municipality)
# Select one city
get_city(data = data)
```

```
## [1] "Umeå"
```

```
# Remove one city
df <- data


df <- df[!df$Municipality %in% get_city(data=df),]

# apply function again
get_city(data = df)
```

```
## [1] "Lund"
```

```
# rempve this city
df <- df[!df$Municipality %in% get_city(data=df),]

# do this till 20 cities left
while(NROW(df) >20){
  df <- df[!df$Municipality %in% get_city(data=df),]
}
```

Numeric Precision

Using a very small number making the numeric precision upto required digits

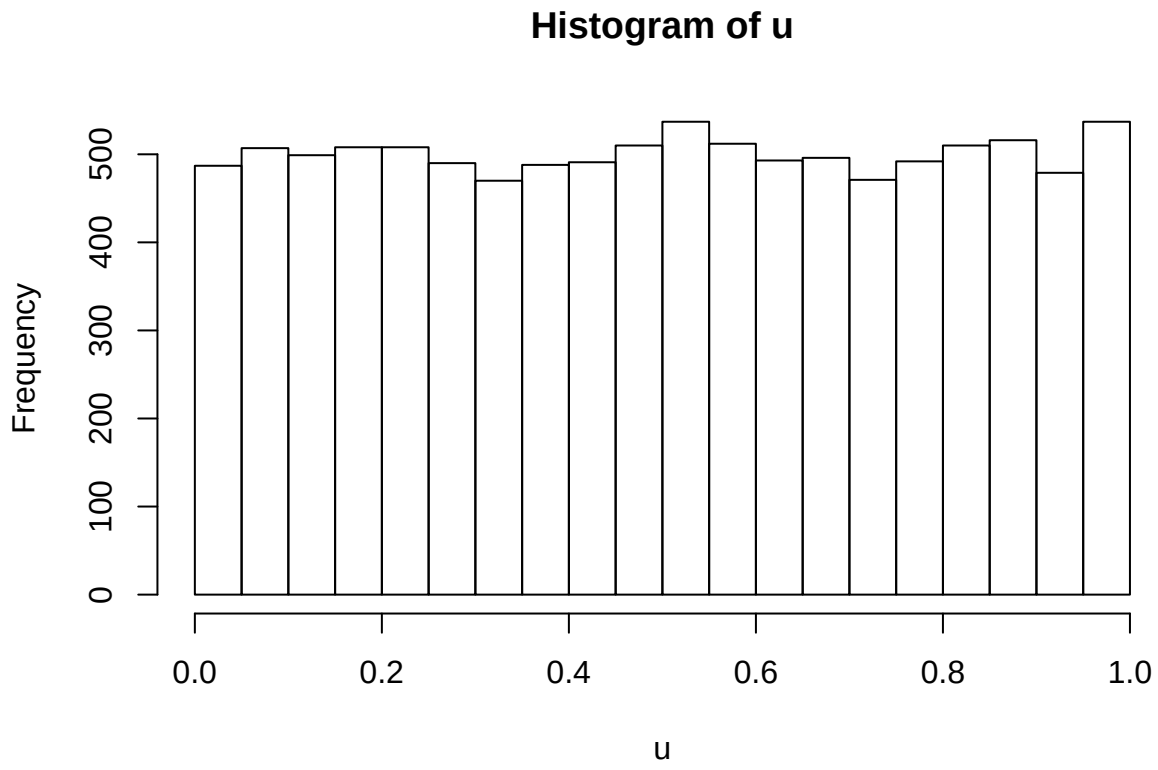
```
options(digits = 22)
tol <- 1e-9
x1 <- 1/3
x2 <- 1/4

if(abs(x1-x2-1/12) <= tol){print("Subtraction is correct")
```

```
}else{print("Subtraction is wrong")}
```

```
## [1] "Subtraction is correct"
```

Random Number Generation

```
x <- rep(0,10000)
x[1] <- 1

a <- 7^5
c <- 0
m <- 2^31-1

for(i in 1:length(x)){
  x[i+1] <- (a*x[i] + c) %% m
}

u <- x/m

hist(u)
```

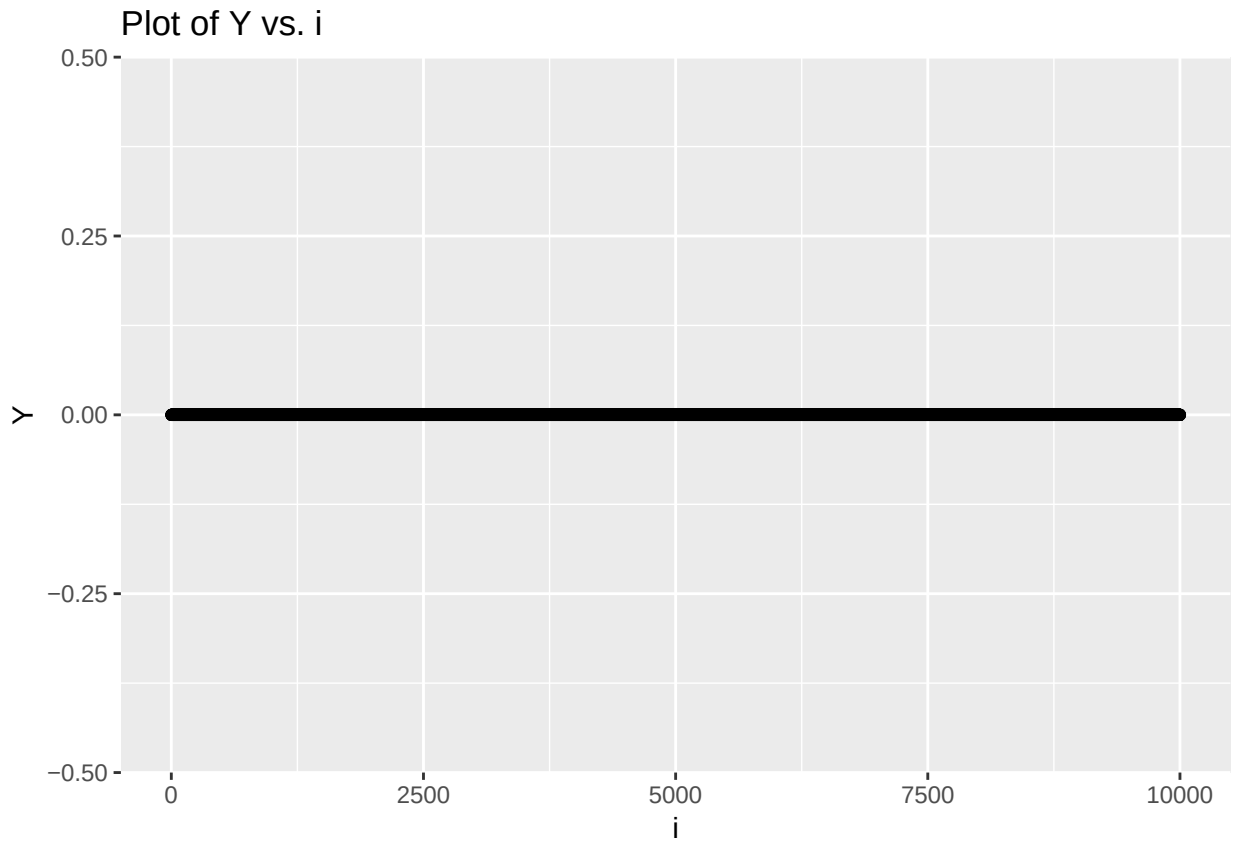


Implementing the Varience

```
my_rand_num <- rnorm(n=10000, mean=10^8, sd=1)

myvar_better <- function(x){
  n <- length(x)
  m <- mean(x)
answer <- (1/(n - 1)) * sum((x - m)^2)
return(answer)
}

Y <- vector(mode = "numeric", length = length(my_rand_num))
my_mat2 <- vector(mode = "numeric", length = length(my_rand_num)+1)

for(i in 2:length(my_rand_num)){
  options(digits = 22)
  X_i = my_rand_num[1:i]
  X_i = na.omit(X_i)
  if((myvar_better(X_i) - var(X_i)) <= 1e-15){
    Y = 0
  }else{
      Y = myvar_better(X_i) - var(X_i)
      }
  temp <- cbind(i, Y)
  my_mat2 <- rbind(temp, my_mat2)
}
```

```
## Warning in rbind(temp, my_mat2): number of columns of result is not a
## multiple of vector length (arg 2)
```

```
my_mat2 <- as.data.frame(my_mat2)

ggplot(my_mat2, aes(x=i, y=Y)) + geom_point() + ggtitle("Plot of Y vs. i")
```



Scaling to get better results

The numeric precision of storage is highest between '0' and '1'

```
kappa(A)
```

Analysis: Following the ?kappa function in R, we get a very large number for the condition number. A large value for the condition number indicates that this matrix is close to being singular.

```
X_scale <- scale(X)
Y_scale <- scale(Y)

A_scale <- t(X_scale) %*% X_scale
b_scale <- t(X_scale) %*% Y_scale

B_hat_scale <- solve(A_scale, b_scale)

kappa(A_scale)
```

Split the data into train and test

```
data <- read.csv2("mortality_rate.csv")
data$LMR <- log(data$Rate)
```

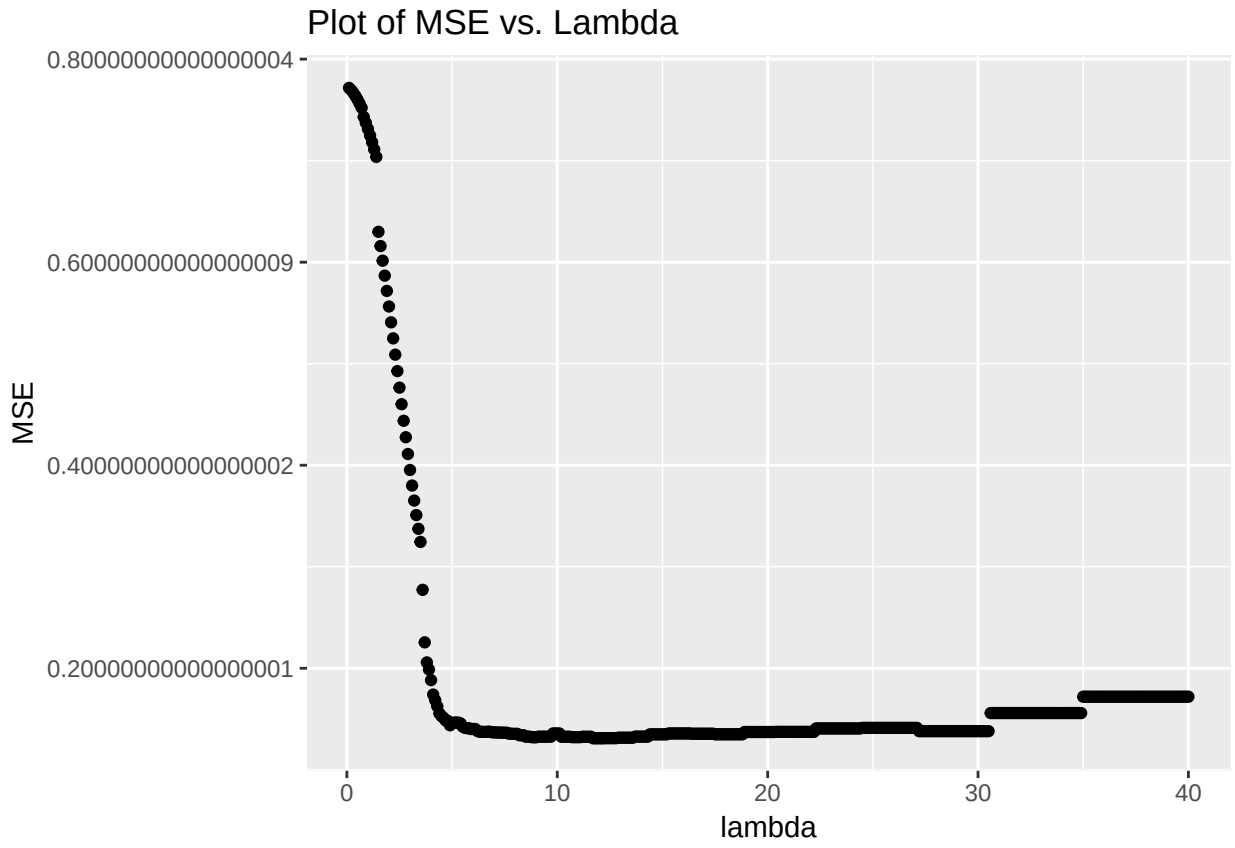
```
n=NROW(data)
set.seed(123456)
id=sample(1:n, floor(n*0.5))
train = data[id,]
test = data[-id,]
```

Loess Model with brute force of finding in minimizing of a function

```
myMSE <- function(pars, lambda){

  X <- pars$X
  Y <- pars$Y
  Xtest <- pars$Xtest
  Ytest <- pars$Ytest

model <- loess(Y ~ X, enp.target=lambda)
predicted <- predict(model, Xtest)
n <- length(Ytest)

exp <- c()
for(i in 1:n){
  exp[i] <- (Ytest[i] - predicted[i])^2
}

answer_mse <- (1/n) * sum(exp)
     count <<- count + 1
return(answer_mse)
}
```

```
library(dplyr)

X <- train %>% select(c(Day)) %>% as.matrix()
Y <- train %>% select(c(LMR)) %>% as.matrix()

Xtest <- test %>% select(c(Day)) %>% as.matrix()
Ytest <- test %>% select(c(LMR)) %>% as.matrix()

pars <- list(X=X, Y=Y , Xtest=Xtest, Ytest=Ytest)

final <- NULL
count <- 0
for(lambda in seq(from = 0.1, to = 40, by = 0.1)){
temp <- myMSE(pars, lambda=lambda)
temp <- cbind(temp, lambda)
final <- rbind(temp, final)
}

colnames(final) <- c("MSE", "lambda")
```

```
ggplot(data=data.frame(final), aes(x = lambda, y=MSE)) + geom_point() +
  ggtitle("Plot of MSE vs. Lambda")
```



Optimize function to find the minimum

```
count <- 0
optimize(interval = c(0.1, 40), f = myMSE, pars=pars, tol=0.01, maximum = FALSE)
```

```
## $minimum
## [1] 10.693610651201773
##
## $objective
## [1] 0.13214414192037774
```

```
cat("iternations: ", count)
```

```
## iternations:  18
```

Optim function to find the minimum

BFGS

```
count <- 0
optim(35, myMSE, pars=pars, method = c("BFGS"))
```

```
## $par
## [1] 35
##
## $value
## [1] 0.17199961445847106
##
## $counts
## function gradient
##        1        1
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
cat("iternations: ", count)
```

```
## iternations:  3
```

CG

```
load("data.RData")
my_log_likehood <- function(pars){
  x<-data
  mu <- pars[1]
  sigma <- pars[2]
  n <- length(x)
answer <- n*0.5*log(2*pi*sigma^2) + (0.5/sigma^2)* sum((x-mu)^2)

return(answer)
}

gradient <- function(pars){
    x<-data
  mu <- pars[1]
  sigma <- pars[2]
  n <- length(x)
  grad_mu <- - (1/sigma^2)* sum(x-mu)
  grad_sig <- (n/sigma) - (1/sigma^3) * sum((x-mu)^2)
  return(c(grad_mu, grad_sig))
}
```

BFGS with and without gradient

```
run1 <- optim(c(0,1), fn = my_log_likehood, gr=NULL, method = c("BFGS"))
run2 <- optim(c(0,1), fn = my_log_likehood, gr=gradient, method = c("BFGS"))
```

CG with and without gradient

```
run3 <- optim(c(0,1), fn = my_log_likehood, gr=NULL, method = c("CG"))
run4 <- optim(c(0,1), fn = my_log_likehood, gr=gradient, method = c("CG"))
```

Showing all results in table

```
final <- NULL
final$algorithm <- c("BFGS", "CG", "BFGS+gradient", "CG+gradient")
final$parameters <- rbind(run1$par, run2$par, run3$par, run4$par)
final$counts <- rbind(run1$counts, run2$counts, run3$counts, run4$counts)
final$convergence <- rbind(run1$convergence, run2$convergence, run3$convergence, run4$convergence)
final$value <- rbind(run1$value, run2$value, run3$value, run4$value)


knitr::kable(as.data.frame(final), caption = "Table showing the summary from various optimization techni
```

Table 1: Table showing the summary from various optimization techniques

algorithm	parameters.1	parameters.2	counts.function	counts.gradient	convergence	
BFGS	1.2755275244738007	2.0059769626279471	37	15	0	211.5069
CG	1.2755275504025778	2.0059765494524084	39	15	0	211.5069
BFGS+gradient	1.2755277195455159	2.0059765019515114	297	45	0	211.5069
CG+gradient	1.2755275911253063	2.0059764724938893	53	17	0	211.5069

Parabolic Interpolation using optim

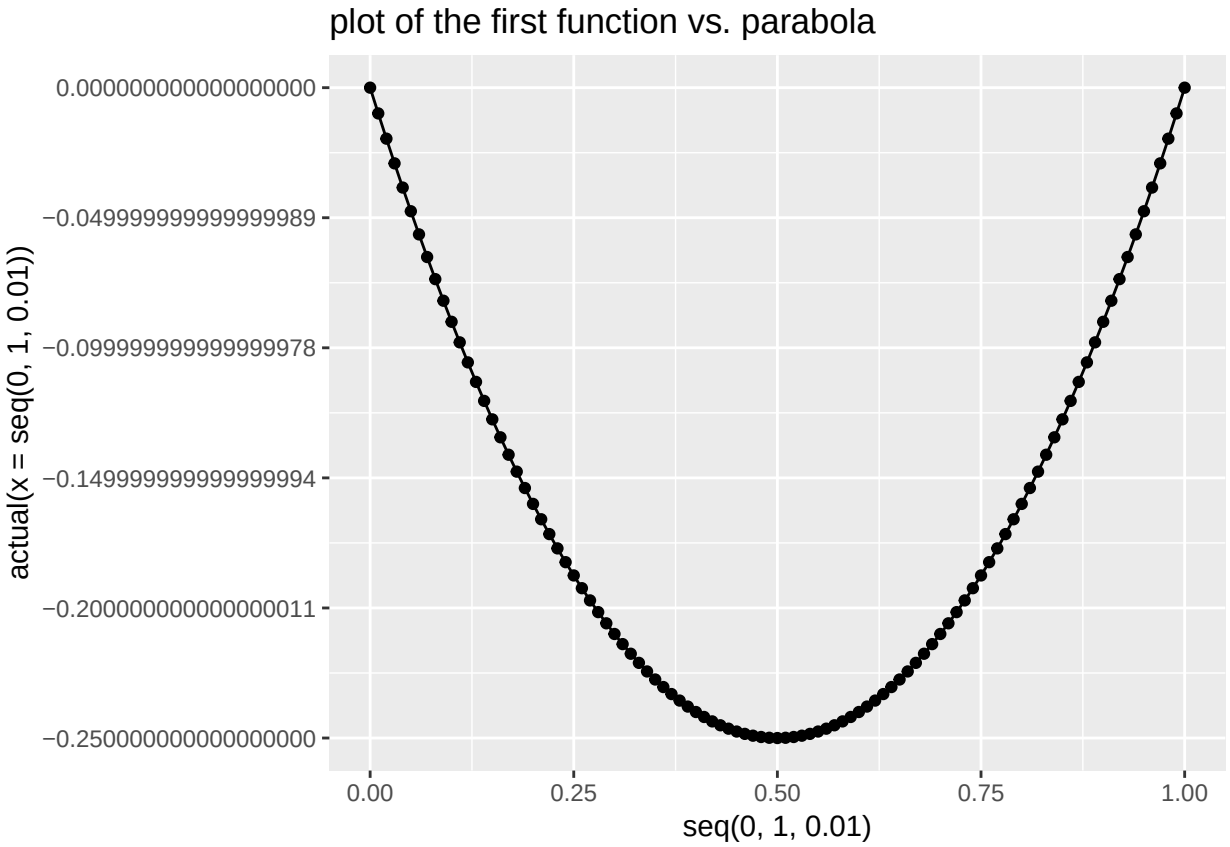
Write a function that uses optim() and find values of (a0, a1, a2) for which g interpolates f at user provided points x0,x1,x2. Interpolate means f(x0) = ~g(x0), f(x1) = g(x1) and f(x2) = g(x2). optim() should minimize the squared error, i.e. find (a0,a1,a2) that make (f(x0)-g(x0))^2 + (f(x1)-g(x1))^2 + (f(x2)-g(x2))^2 minimum

```
# Actual function to estimate the minimum value of
actual <- function(x){
  result <- -x *(1-x)
return(result)
}

# The estimation function whose parameters a0,a1,a2 are unknown
```

```r
parabola <- function(par,x){
  a0 <- par[1]
  a1 <- par[2]
  a2 <- par[3]
  result <- a0+a1*x+a2*x^2
return(result)
}

# finding the difference between the functions for three given values (x0,x1,x2)
difference_function <- function(par,x){
  x0 <- par[4]
  x1 <- par[5]
  x2 <- par[6]
  result <- sum((actual(x0)-parabola(par,x0))^2,(actual(x1)-parabola(par,x1))^2, (actual(x2)-parabola(pa
  return(result)
}

find_parameters <- function(){
temp <- optim(par=c(0,-1,1,0.1,0.8,0.9), fn=difference_function)
a0 <- temp$par[1]
a1 <- temp$par[2]
a2 <- temp$par[3]
return(list=c(a0=a0,a1=a1,a2=a2))
}

find_parameters()
```

```
## a0 a1 a2
##  0 -1  1
```

```r
# plotting to show the values
ggplot() +
  geom_point(aes(x=seq(0,1,0.01), y=actual(x=seq(0,1,0.01)))) +
   geom_line(aes(x=seq(0,1,0.01), y=parabola(par=c(0,-1,1),x=seq(0,1,0.01)))) +
  ggtitle("plot of the first function vs. parabola")
```



Inverse CDF

The inverse transform method is a simple algorithm for generating random variables x from a *continuous* target distribution $f(x)$ using random samples from a $Unif(0,1)$ distribution.

Algorithm:

1. For target probability density function (pdf) $f(X)$, calculate the CDF, $F(X)$
2. Set the CDF equal to U, $F(X) = U$, then solving for X, obtaining $F^{-1}(U) = X$
3. Generate n random variables from $u \sim Unif(0,1)$
4. Plug in u observed values in $F^{-1}(U = u)$ to obtain n values for which $x \sim f(X)$

Example: Pareto Distribution

For information on the Pareto distribution.

The $Pareto(a,b)$ distribution has CDF $F(X \leq x) = 1 - (\frac{b}{x})^a$ for $x \geq b > 0,\ a > 0$

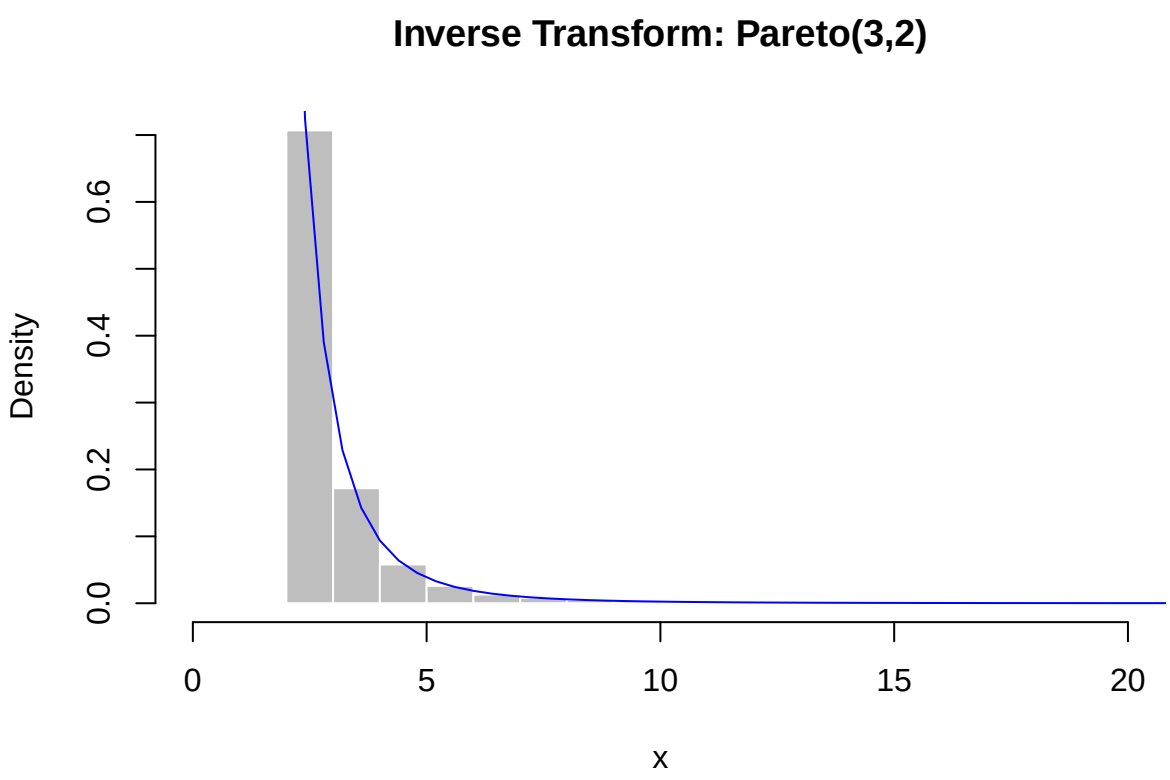
1. First set $F(x) = U$, where $U \sim Unif(0,1)$, then solve for X

$$\begin{aligned} 1 - \left(\frac{b}{x}\right)^2 &= U \\ \left(\frac{b}{x}\right)^a &= 1 - U \\ \frac{b}{x} &= (1-U)^{1/a} \\ x &= b \times (1-U)^{-1/a} \\ &= F_X^{-1}(U) \end{aligned}$$

```
set.seed(123)
n = 1000
U =runif(n)
a = 3
b = 2
X = b*(1-U)^(-1/a)
pareto = function(x){(a*(b^a)/x^(a+1))}

summary(X)
```

```
##                Min.             1st Qu.              Median
##  2.0003103289867905  2.2048580148641430  2.5031823508377102
##                Mean             3rd Qu.                Max.
##  2.9690832866834609  3.1613937424093539 23.7725736025729120
```

```
hist(X, probability = TRUE, breaks = 25, xlim =c(0, 20),
     col = "gray", border = "white",
     main = "Inverse Transform: Pareto(3,2)", xlab = "x")
curve(pareto(x), from = 0, to = 40, add = TRUE, col = "blue")
```



The double exponential (Laplace) distribution is given by formula:

$$DE(\mu, \alpha) = \frac{\alpha}{2} e^{(-\alpha|x-\mu|)}$$

CDF:

$$F(x) = \int_{-\infty}^{x} f(x)dx$$

$$F(x) = \int_{-\infty}^{x} \frac{\alpha}{2} e^{-\alpha(x-\mu)} dx, \qquad (if \quad x > \mu)$$

$$= 1 - \int_{x}^{\infty} \frac{\alpha}{2} e^{-\alpha(x-\mu)} dx$$

$$= 1 - \frac{1}{2} e^{-\alpha(x-\mu)}$$

$$F(x) = \int_{-\infty}^{x} \frac{\alpha}{2} e^{\alpha(x-\mu)} dx, \qquad (if \quad x \leq \mu)$$

$$= \frac{1}{2} e^{\alpha(x-\mu)}$$

Inverse of CDF

$$For \quad x > \mu, \ we \ got \ F(x) = 1 - \frac{1}{2}e^{-\alpha(x-\mu)}$$

$$y = 1 - \frac{1}{2}e^{-\alpha(x-\mu)}$$

$$\frac{ln(2-2y) - \alpha\mu}{-\alpha} = x$$

$$For \ U \sim U(0,1), \qquad \frac{ln(2-2U) - \alpha\mu}{-\alpha} = X$$

$$For \quad x \leq \mu, \ we \ got \ F(x) = \frac{1}{2}e^{\alpha(x-\mu)}$$

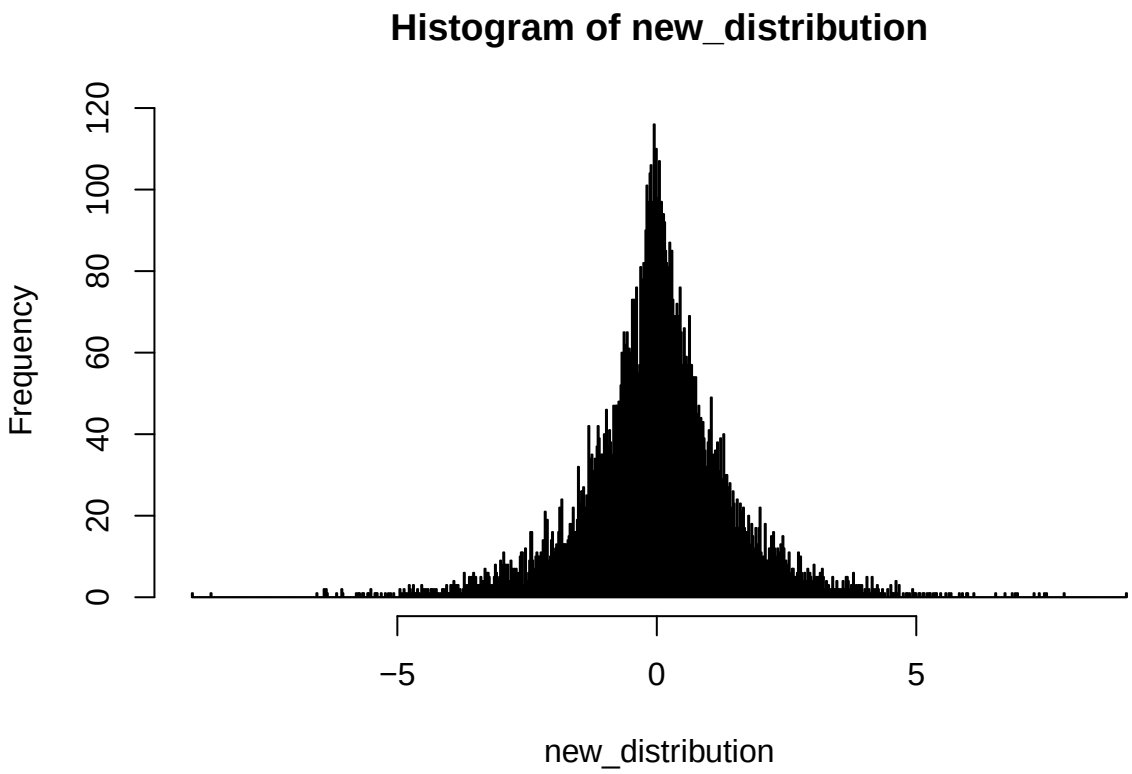
$$y = \frac{1}{2}e^{\alpha(x-\mu)}$$

$$\frac{ln(2y)}{\alpha} + \mu = x$$

$$For \ U \sim U(0,1), \qquad \frac{ln(2U)}{\alpha} + \mu = X$$

```
de_dist <- function(u, mu, a){
  de_distribution <- c()

    for (i in 1:length(u)){
      if (u[i] > 0.5){
        de_distribution[i] <- (log(2-2*u[i])-a*mu)/(-a)
      }
      else {
        de_distribution[i] <- (log(2*u[i])/a)+mu
      }
    }
  return(de_distribution)
}
```

```
new_distribution <- de_dist(runif(10000, 0, 1), mu=0, a=1)
hist(new_distribution, breaks = 1000)
```



Generate Normal distribution using uniform

```
box_muller <- function(n = 1, mean = 0, sd = 1)
{
  x <- vector("numeric", n)

  i <- 1
  while(i <= n)
  {
    u1 <- runif(1, 0, 1)
    u2 <- runif(1, 0, 1)

    x[i] <- sqrt(-2 * log(u1)) * cos(2 * pi * u2)

    if ((i + 1) <= n)
    {
      x[i + 1] <- sqrt(-2 * log(u1)) * sin(2 * pi * u2)
      i <- i + 1
    }

    i <- i + 1
  }
```

```
  x * sd + mean
}

hist(box_muller(1000))
```

Accpetance/Rejection Method

To generate n samples, `for(i in 1:n)`

1. Generate $Y \sim g_Y(t)$ and $U \sim Unif(0,1)$
2. If $U \leq \frac{f(Y)}{M \times g(Y)}$ then we accept Y, such that $Y = X$
3. Repeat until you have sufficient samples

In order for the algorithm to work we require the following constrains:

1. f and g have to have compatible supports (i.e. $g(x) > 0$ when $f(x) > 0$)
2. There is a constant M such that $\frac{f(t)}{g(t)} \leq M$

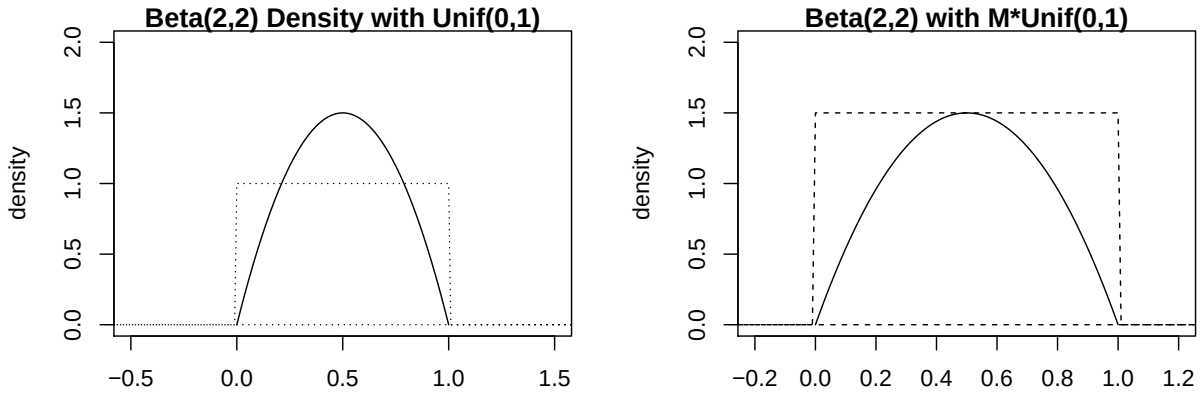
Example: Beta(2,2)

Suppose we'd like to generate samples from $Beta(2,2)$ distribution. The density function for $Beta(2,2)$ is simply $f(x) = 6x(1-x)$ for $0 < x < 1$. Since our domain is between 0 and 1, we can use a simple $Unif(0,1)$

density as our instrumental density, g. Then, by the accept-reject algorithm we can simulate a random variable $Y \sim g$, and a random variable $U \sim Unif(0,1)$. Then, if

$$M \times U \leq \frac{f(Y)}{g(Y)}$$

we accept the candidate variable $Y \sim g$ as X, $X = Y$. Otherwise, we reject Y and simulate again until we get an appropriate sample size.



Generating N points and finding maximizing value

Note that the target density f has a maximum of 1.5, so we can set M = 1.5 (using calculus or use optimize function as shown)

```
# find M using optimize
f <- function(x){ 6*x*(1 - x)} ## pdf of Beta(2,2)
max_find <- optimize(f=f, lower = 0, upper = 10, maximum = TRUE)
max_find
```

```
## $maximum
## [1] 0.49999999999999956
##
## $objective
## [1] 1.5
```

```
M <- max_find$objective



## Accept-Reject
M = 1.5
X = rep(x = NA, 5) ## create a vector of length 5 of NAs
set.seed(123)
f <- function(x){ 6*x*(1 - x)} ## pdf of Beta(2,2)
g <- function(x){ 1 } ## pdf of Unif(0,1) is just 1

n = 10000
```

Now, say we needed $n = 10,000$ samples from $Beta(2,2)$, then a better implementation would be

```
X = rep(NA, n); M = 1.5
i = 0 ## index set to start at 0
```

```r
while(sum(is.na(X))){
  U = runif(1); Y = runif(1)
  accept <- U <= f(Y)/(M*g(Y))
  if(accept){
    i = i+1 ## update the index
    X[i] <- Y
  }
}

round(summary(X), 4)
```

```
##                  Min.               1st Qu.                Median
## 0.0037000000000000002 0.3250000000000000111 0.4961999999999999744
##                  Mean               3rd Qu.                  Max.
## 0.5000999999999999890 0.6766999999999999682 0.9929999999999999938
```

```r
round(qbeta(p = c(0, 0.25, 0.5, 0.75, 1), 2, 2), 4)
```

```
## [1] 0.0000000000000000 0.3264000000000000002 0.5000000000000000000
## [4] 0.6735999999999998 1.0000000000000000000
```

Example

$$\frac{f(y)}{g(y)} \leq c$$

Thus if we maximize the ratio f(y)/g(y) then that would be the value of c. This can be done by using parital derivate of the fraction.

$$Majorizing\ density\ F_Y(y) \sim DE(0,1) = \frac{1}{2}e^{-|x|}$$

$$Target\ density\ F_X(y) \sim N(0,1) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$$

$$C \geq \frac{\sqrt{2}}{\sqrt{\pi}}e^{-\frac{x^2}{2}+|y|}$$

Differenitating with respect to x we get that the expression is maximum at x=1, thus C is:

$$\sqrt{\frac{2}{\pi}}e^{\frac{-1}{2}}$$

$$= \frac{\sqrt{2}}{C\sqrt{\pi}} \cdot e^{-\frac{y^2}{2}+|y|}$$

$$= e^{-\frac{y^2}{2}+|y|+\frac{1}{2}}$$

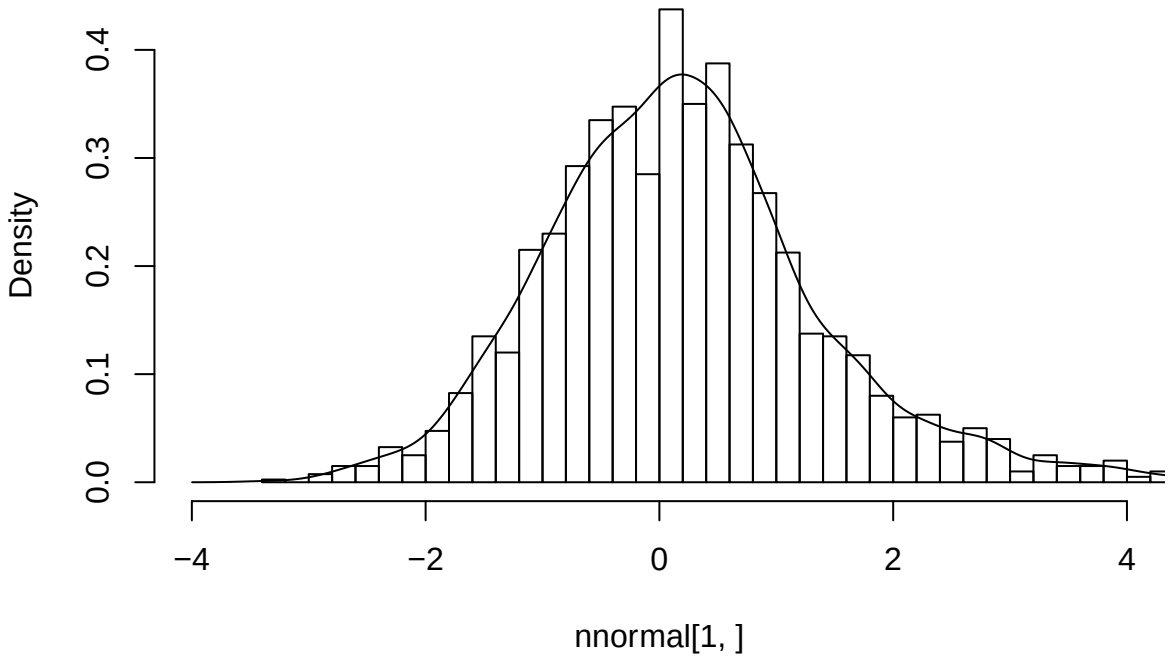
Thus C is

$$C = \sqrt{(\frac{2 * e^1}{\pi})}$$

```r
generate_n <- function(c){
  x <- NA
  num.reject <- 0
    while (is.na(x)){
      u <- runif(1, 0, 1)
      y <- de_dist(u, 0, 1)
      U <- runif(1)
        if (y>0 && U <= sqrt(2/pi)/c*exp(-(y^2)/2)+y){
          x <- y
        } else if (y<=0 && U<=sqrt(2/pi)/c*exp(-(y^2)/2-y)){
            x <- y
              } else {
                num.reject <- num.reject + 1
            }
    }
  c(x,num.reject)
}


c <- sqrt(2*exp(1)/pi)
```

```r
set.seed(12345)
nnormal <- sapply(rep(c,2000), generate_n)
```

```r
hist(nnormal[1,], breaks = 50, freq = FALSE, xlim = c(-4,4),
     main = "Normal distribution generated by Accept/Reject method")
lines(density(nnormal[1,]))
```




```
average_rejection <- sum(nnormal[2,])/(ncol(nnormal)+sum(nnormal[2,]))
average_rejection
```

```
## [1] 0.15433403805496829
```

```
expected_rejection <- 1-(1/c)
expected_rejection
```

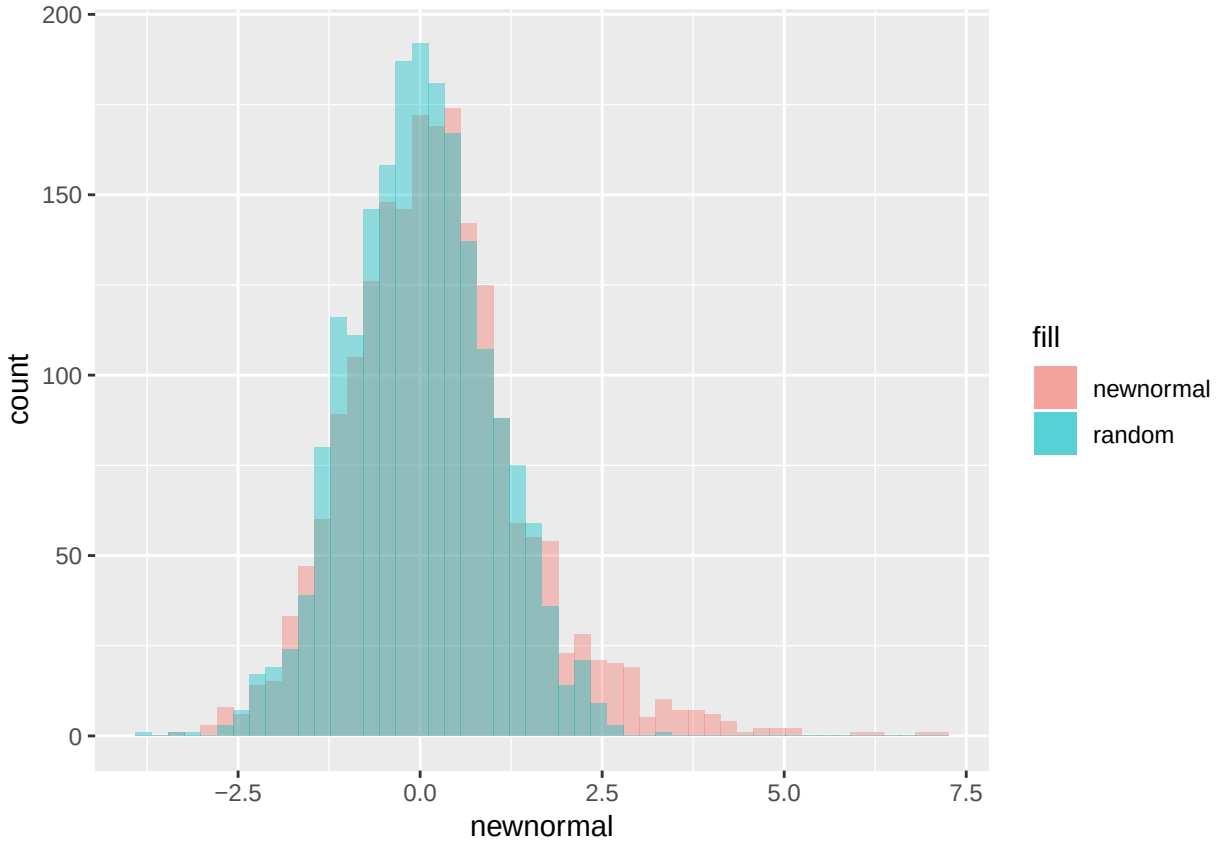
```
## [1] 0.23982654946685966
```

```
rejection_difference <- expected_rejection - average_rejection
rejection_difference
```

```
## [1] 0.085492511411891375
```

```
newnormal <- nnormal[1,]
random <- rnorm(2000, 0, 1)
df <- as.data.frame(cbind(newnormal, random))
```

```
ggplot(df, aes(newnormal, fill = "newnormal")) + geom_histogram(alpha = 0.4, bins = 50) +
  geom_histogram(aes(random, fill = "random"), alpha = 0.4, bins = 50)
```



Monte Carlo Integration

Now, to generalize the method used. Given a function $h(x)$ whose integral is well defined, where we wish to evaluate at interval a to b. Then

$$\begin{aligned}\theta &= \int_a^b h(x)dx \\ &= (b-a)\int_a^b h(x)\frac{1}{b-a}dx \\ &= (b-a)\int_a^b h(x)f(x)dx\end{aligned}$$

where $f(x) = \frac{1}{b-a}$ is $Unif(a,b)$, and $x \sim Unif(a,b)$.

The algorithm to calculate $\hat{\theta}$ is as follows:

1. Find a density $f(x)$ from which we can sample x
2. Generate $x_1, ..., x_n \sim f(x)$
3. Compute $(b-a) \times \bar{g}_n$, where $\bar{g}_n = \frac{1}{n}\sum_{i=1}^n h(x_i)$

Example

Suppose we have a function $h(x) = 3x^2$ for which we wish to integrate over the interval $[0, 2]$.

- Can apply deterministic numerical approximation methods (see R's `integrate`)
- or we could treat x as a random variable, $X = x$, from a $Unif(0, 2)$ whose pdf is simply $f(x) = \frac{1}{2-0} = \frac{1}{2}$.

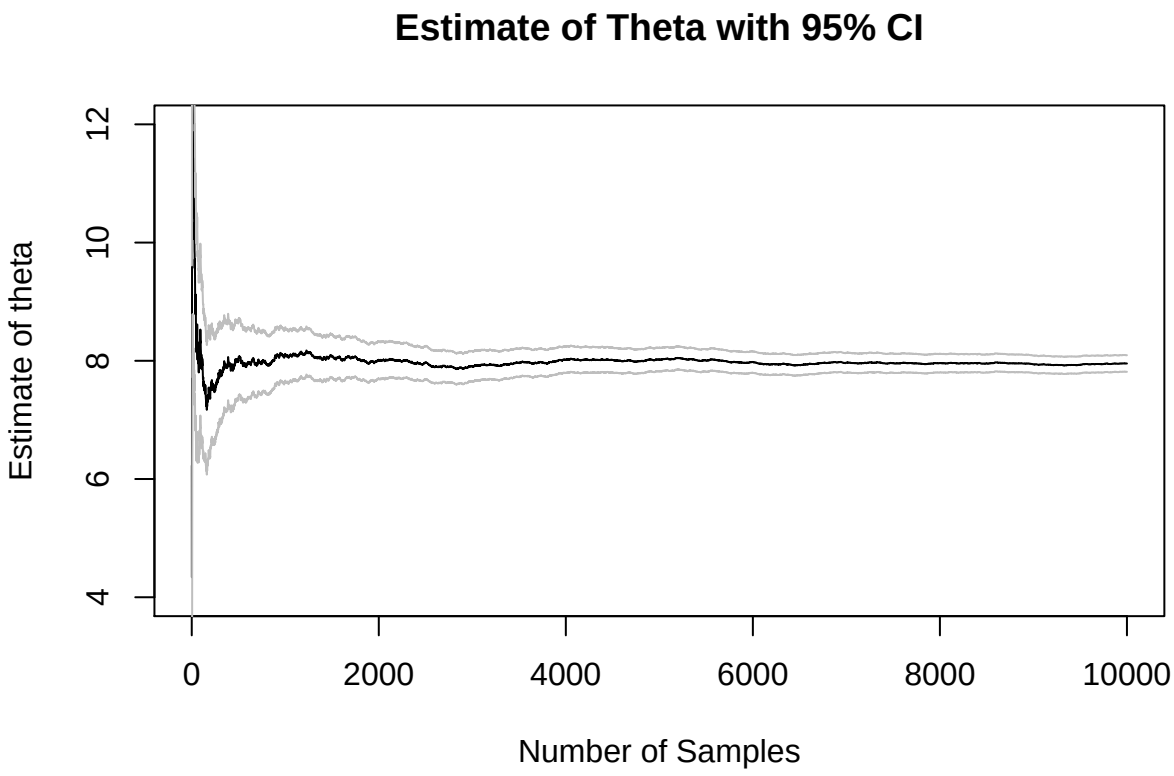
If we now generate some n random values from $f(x)$ and evaluate them at $h(x)$, then take the mean, we'd be calculating the expected value of $h(x)$,

$$\begin{aligned}
\theta &= \int_0^2 h(x)dx \\
&= (\frac{2-0}{2-0}) \times \int_0^2 h(x)dx = 2 \times \int_0^2 h(x)\frac{1}{2}dx \\
&= 2 \times E[h(X)] = 2 \times \int_{-\infty}^{\infty} h(x)f(x)dx \\
&\approx 2 \times \frac{1}{n}\sum_{i=1}^{n} h(x_i) \\
&= \hat{\theta} \approx \theta
\end{aligned}$$

Monte Carlo Integration, Variance Estimation

We can now calculate the standard error for the former example.

```
N = 10000 ## sample size
h <- function(x) { 3*x^2 } ## function of interest, h(x)
X <- runif(n = N, min = 0, max = 2)  ## samples from f(x)
h_values <- 2 * h(X)

cumMean <- function(x, n){
  num = cumsum(x)  ## numerator
  denom = 1:n  ## denominator
  result = num/denom
  return(result)
}


cumSE <- function(x, n){
  m = mean(x)
  num = sqrt(cumsum((x - m)^2)) ## numerator
  denom = 1:n ## denominator
  result = num/denom ## cummulative mean of (x_i - theta)^2
  return(result)
}

thetas = cumMean(h_values, N)
SE = cumSE(h_values, N)


plot(x = 1:N, y = thetas, type = "l", ylim = c(4, 12), xlab = "Number of Samples",
     ylab = "Estimate of theta",
     main = "Estimate of Theta with 95% CI")
lines(x = 1:N, y = thetas + 1.96*SE, col = "gray") ## CI
lines(x = 1:N, y = thetas - 1.96*SE, col = "gray") ## CI
```




```
## final estimate
thetaHat = mean(h_values)
se <- sd(x = h_values)/sqrt(N)
ci <- thetaHat + 1.96*c(-1,1) * se

print(thetaHat) ## theta estimate
```

```
## [1] 7.9538675271615578
```

```
print(ci) ## 95% CI
```

```
## [1] 7.8134362147371919 8.0942988395859228
```

Metropolis Hastings

The **Independent Metropolis-Hastings algorithm** as described Robert & Casella goes as follows

Given $x^{(t)}$

1. Generate $Y_t \sim g(y)$
2. Take

$$X_{t+1} = \begin{cases} Y_t & \text{with probability } \rho(x^{(t)}, Y_t) \\ x^{(t)} & \text{with probability } 1 - \rho(x^{(t)}, Y_t) \end{cases}$$

where

$$\rho(x^{(t)}, Y_t) = \min\left\{\frac{f(Y_t)}{f(x^{(t)})}\frac{g(x^{(t)})}{g(Y_t)}, 1\right\}$$

In simpler terms, as we want to generate $X \sim f$, we first take an initial value $x^{(0)}$ (which can almost be any arbitrary value in the support of f).

1. We generate a value $Y_0 \sim q(y|x^{(0)})$.
2. We calculate $\rho(x^{(t)}, Y_t)$
3. Generate a random value $U \sim Unif(0, 1)$
4. If $U < \rho(x^{(t)}, Y_t)$, then we accept $X^{(1)} = Y_t$; else we take $X^{(1)} = X^{(0)}$
5. Repeate steps 1-4 until you've satisfied the number of samples needed

Example

Use Metropolis-Hastings algorithm to generate samples from the below distribution by using proposal distribution as chi^2(floor(X(t)+1)), take some starting point. Plot the chain you obtained as a time series plot. What can you guess about the convergence of the chain? If there is a burn-in period, what can be the size of this period?

$$f(X) \sim x^5 e^{-x}, x > 0$$

```
#density function from which we want to sample.
dt <- function(x) {
  if (x <= 0) {
    stop("x has to be greater than 0.")
  }
  return(x^5 * exp(-x))
}

#proposal density function, should accept 2 arguments:
  #   x:  value at which to compute density.
  #   x_t: value on which the rq is conditioned.
dp <- function(x, xt) {
  dchisq(x, floor(xt + 1))
}

rp <- function(x) {
  rchisq(1, floor(x+1))
}

metropolis_hastings <- function(x_0, t_max, dt, dp, rp) {
  # Perform Metropolis-Hastings sampling of the specified density.
  #
  # Args:
  #   x_0   starting value.
  #   t_max maximum numer of iterations.
  #   dt    density function from which we want to sample.
  #   dp    proposal density function, should accept 2 arguments:
  #           x:  value at which to compute density.
  #           x_t: value on which the rq is conditioned.
  #   rp    proposal random number generator, should accept 1 argument:
  #           x_t: value on which the rq is conditioned.
```

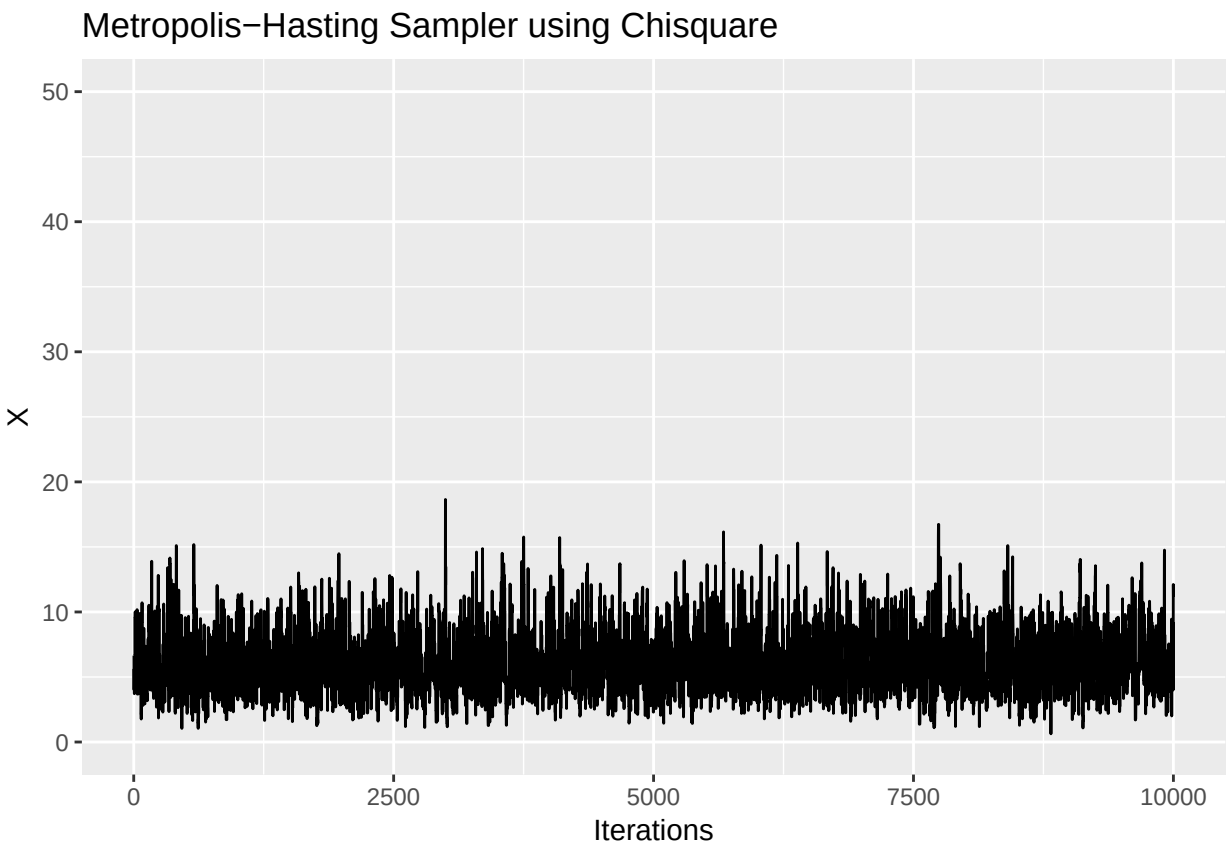
```
  #
  # Returns:
  #   Vector of sampled points of length t_max.

  x_t <- x_0
  x <- vector("numeric", t_max) # pre-allocate

  # Metropolis-Hastings
  for (t in 1:t_max) {
    # Generate a candidate
    y <- rp(x_t)
    # Generate U
    u <- runif(1, 0, 1)
    # Compute alpha
    alpha <- min(1, (dt(y) * dp(x_t, y)) / (dt(x_t) * dp(y, x_t)))
    # Accept the jump or stay in x_t
    if (u < alpha) {
      x_t <- y
    }
    # Sace x_t in vector
    x[t] <- x_t
  }

  return(x)
}


test <- metropolis_hastings(x_0=1, t_max = 10000, dt, dp, rp)

# compare the distribution
x=seq(from=0.01, to=20, by=0.002)
actual <- numeric(length(x))
for(i in seq_along(x)){
  actual[i] <- dt(x=x[i])
}

quantile(test)
```

```
##                   0%                  25%                  50%
##  0.63207349214489117  4.20543485481044410  5.71968566388519051
##                  75%                 100%
##  7.57026419107211890 18.64709586611740377
```

```
quantile(temp)
```

```
##                   0%                  25%                  50%
##  0.17199961445847106 10.12899971084385342 20.08599980722923561
##                  75%                 100%
## 30.04299990361461781 40.00000000000000000
```

```
ggplot() +
geom_line(aes(x=1:10000, y=test)) +
labs(x="Iterations", y="X") +
ylim(0, 50) +
ggtitle("Metropolis-Hasting Sampler using Chisquare")
```



Gelman Rubin

```
all_series <- NULL
for(i in 1:10) {
temp <- metropolis_hastings(x_0=i, t_max = 10000, dt, dp, rp)
temp <- coda::as.mcmc(temp)
all_series[[i]] <- temp
}

#convergence analysis
coda::gelman.diag(all_series)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]          1          1
```

Gibbs Sampling

```
calculate_conditional_probability <- function(i, X, Y, ...) {
  # Returns the conditional probability of an element
  # Args: (might change with the given task)
```

```r
  #   X   Value we are looking for
  #   Y   Data
  #   i   position
  # In the lab we had to consider three cases
  # This will change in the exam!
  d <- length(X)
  if (i == 1) {
    # i = 1
    mean <- (Y[1] + X[2]) / 2
    variance <- sigma_squared / 2
  } else if (i == d) {
    # i = d
    mean <- (X[d - 1] + Y[d]) / 2
    variance <- sigma_squared / 2
  } else {
    # i = 2 ... d-1
    mean <- (X[i - 1]+ Y[i] + X[i + 1]) / 3
    variance <- sigma_squared / 3
  }

  # Generate a random variables
  X_i <- rnorm(1, mean = mean, sd = sqrt(variance))

  return(X_i)
}

gibbs_sampling <- function(Y, n, X0, ...) {
  # Gibbs sampling
  # Args:
  #   Y   Data
  #   n   Number of generated samples
  #   X0  Initialization points
  # Returns:
  #   X   Matrix with samples (colmeans = value we are looking for)
  d <- length(Y)
  X <- matrix(NA, ncol = d, nrow = n)
  X[1, ] <- X0

  for (row in 2:n) {
    for (col in 1:d) {
      X[row, col] <- calculate_conditional_probability(
        i = col,
        X = c(X[row, 1:(col - 1)], X[row - 1, col:d]),
        Y = Y,
        sigma_squared = sigma_squared
      )
    }
  }
  return(X)
}
```

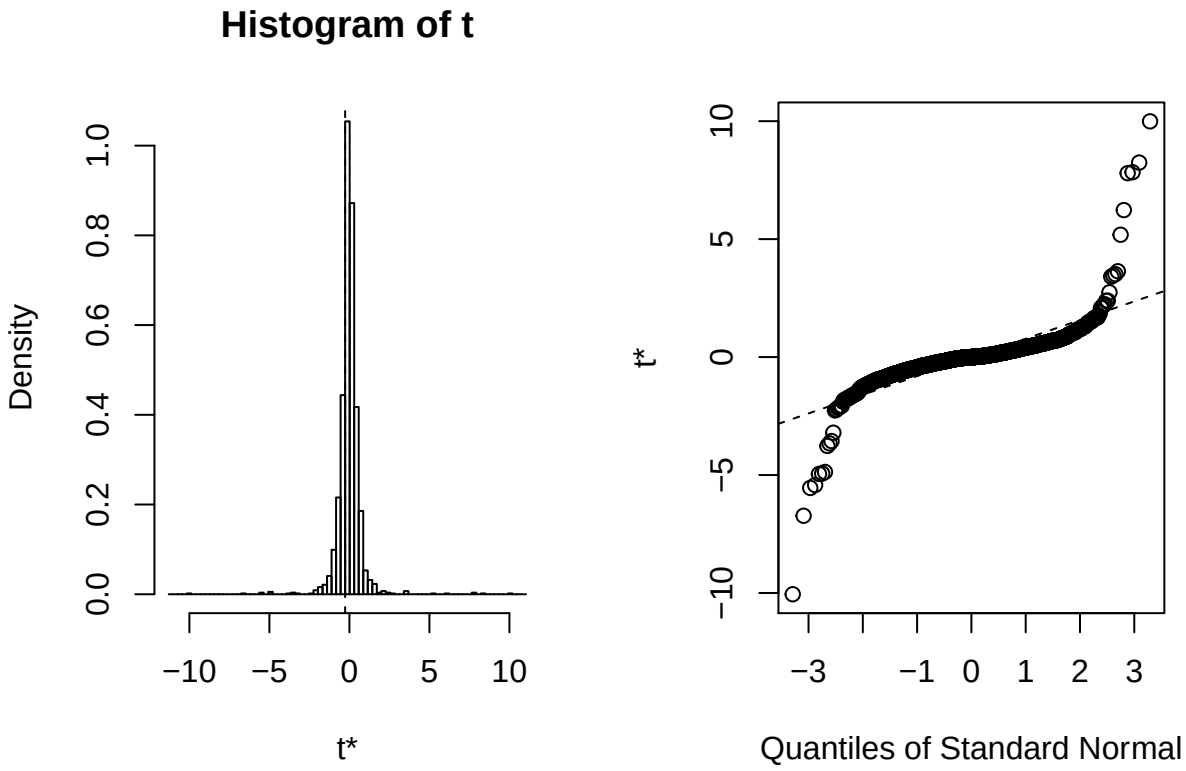
Bootstrap estimation

Estimating the distribution of T by using a non-parametric bootstrap with B = 2000

Using Boot library

$T = \frac{\hat{Y}(X_b) - \hat{Y}(X_a)}{X_b - X_a}$ Where $X_b = argmax_x Y(X)$ and $X_a = argmin_x Y(X)$

```
data <- read.csv2('lottery.csv')

stat1 <- function(data,vn){
  data <- as.data.frame(data[vn,])
  X_b <- data[which.max(data$Draft_No),]$Day_of_year
  X_a <- data[which.min(data$Draft_No),]$Day_of_year
  model <- loess(formula = Draft_No~Day_of_year
                 , data = data)
  Y_hat <- predict(object = model, newdata = data)
  Y_hat_a <- Y_hat[X_a]
  Y_hat_b <- Y_hat[X_b]
  (Y_hat_b - Y_hat_a)/(X_b - X_a)
}

res <- boot(data = data, statistic = stat1
                          # Ordinary nonparametric bootstrap (default)
            , R = 2000, sim = "ordinary")


plot(res)
```



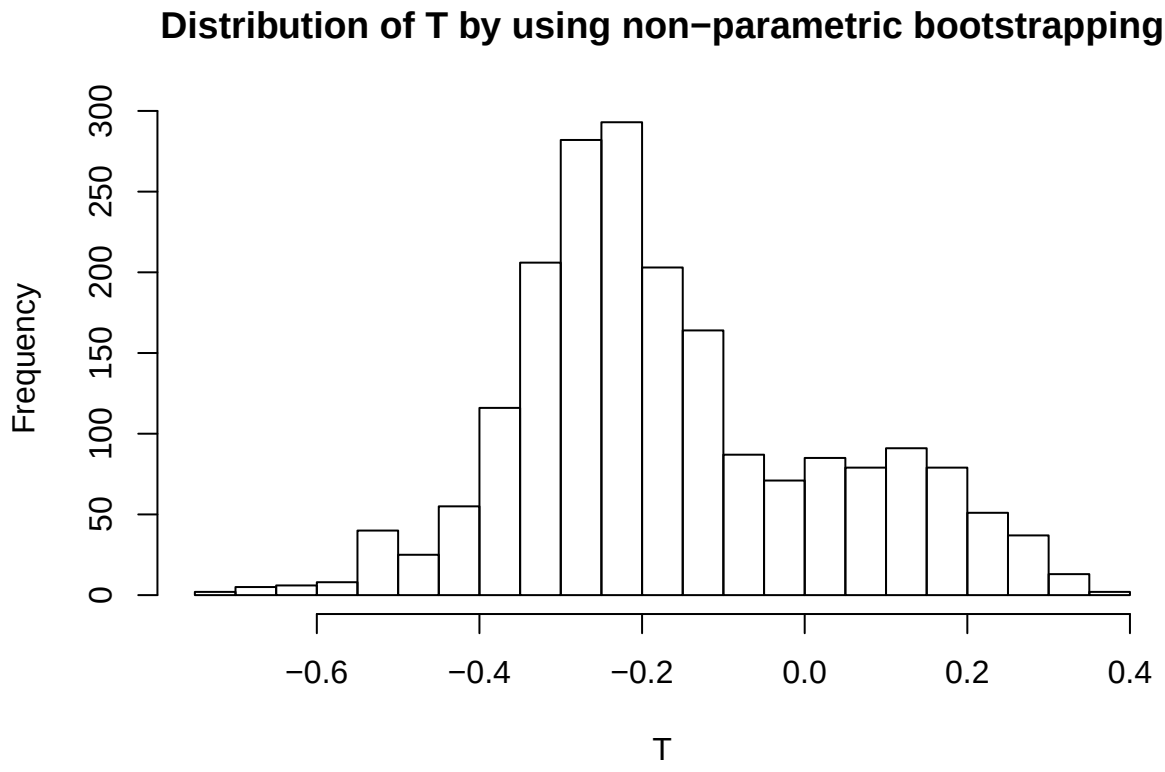

```
# P-Value
mean(abs(res$t0) < abs(res$t-mean(res$t)))
```

```
## [1] 0.47599999999999998
```

Without Boot library

```
set.seed(12345)

X <- data$Day_of_year
Y <- data$Draft_No

T <- numeric()

for (i in 1:2000) {
        boot <-sample(length(X), replace=TRUE)
        data1 <- cbind(X,boot)

        X_a <- data1[which.min(data1[,2])]
        X_b <- data1[which.max(data1[,2])]

        model2 <- loess(Y~X, data=as.data.frame(data1), method="loess")

        fitted_X_a <- model2$fitted[X_a]
        fitted_X_b <- model2$fitted[X_b]
```

```
        test <- (fitted_X_b - fitted_X_a)/(X_b - X_a)
        T[i] <- test
        }


pval <- length(which(T>=0))


hist(T, breaks=30,
     main="Distribution of T by using non-parametric bootstrapping")
```




```
print("Estimated p-value:")
```

```
## [1] "Estimated p-value:"
```

```
print(pval/2000)
```

```
## [1] 0.2185
```

Varience Estimate

Estimate the distribution of the mean price of the house using bootstrap. Determine the bootstrap bias-correction and the variance of the mean price. Compute a 95% confidence interval for the mean price using bootstrap percentile, bootstrap BCa, and first-order normal approximation.

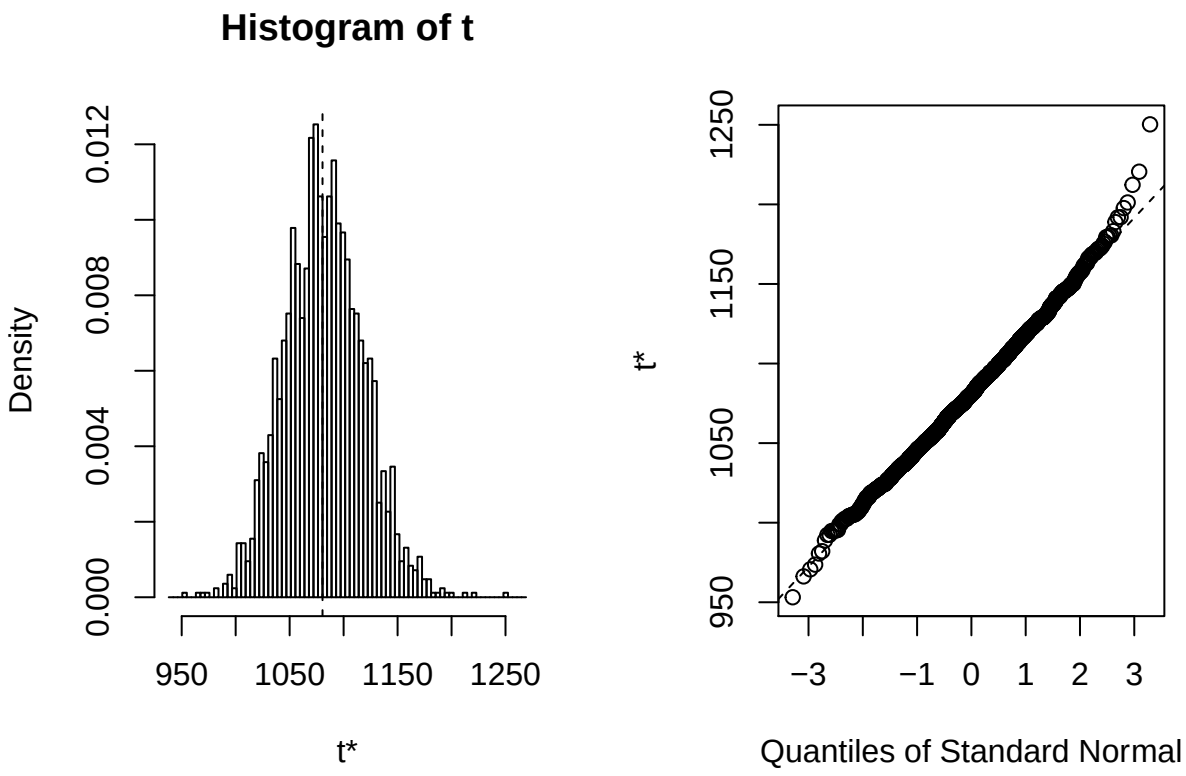
Bias correction

$$T1 = 2.T(D) - \frac{1}{D}\sum_{i=1}^{B} T_i^*$$

```
price_data <- read.csv("prices1.csv", sep=";")

# Estimation of mean of Price
stat_mean <- function(data, index){
    data <- data[index,]
    answer <- mean(data$Price)
    return(answer)
}

res <- boot::boot(data=price_data, statistic = stat_mean, R=2000)
res
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot::boot(data = price_data, statistic = stat_mean, R = 2000)
##
##
## Bootstrap Statistics :
##               original             bias           std. error
## t1* 1080.4727272727273 1.4798636363636888 36.546015841440351
```

```
plot(res,index = 1)
```




```
#95% CI for mean using percentile
boot.ci(res, index=1, type=c('perc'))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = res, type = c("perc"), index = 1)
##
## Intervals :
## Level     Percentile
## 95%   (1014, 1156 )
## Calculations and Intervals on Original Scale
```

```
#95% CI for mean using bca
boot.ci(res, index=1, type=c('bca'))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = res, type = c("bca"), index = 1)
##
## Intervals :
## Level       BCa
## 95%   (1016, 1158 )
## Calculations and Intervals on Original Scale
```

```r
#95% CI for mean using first order normal
boot.ci(res, index=1, type=c('norm'))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = res, type = c("norm"), index = 1)
##
## Intervals :
## Level      Normal
## 95%   (1007, 1151 )
## Calculations and Intervals on Original Scale
```

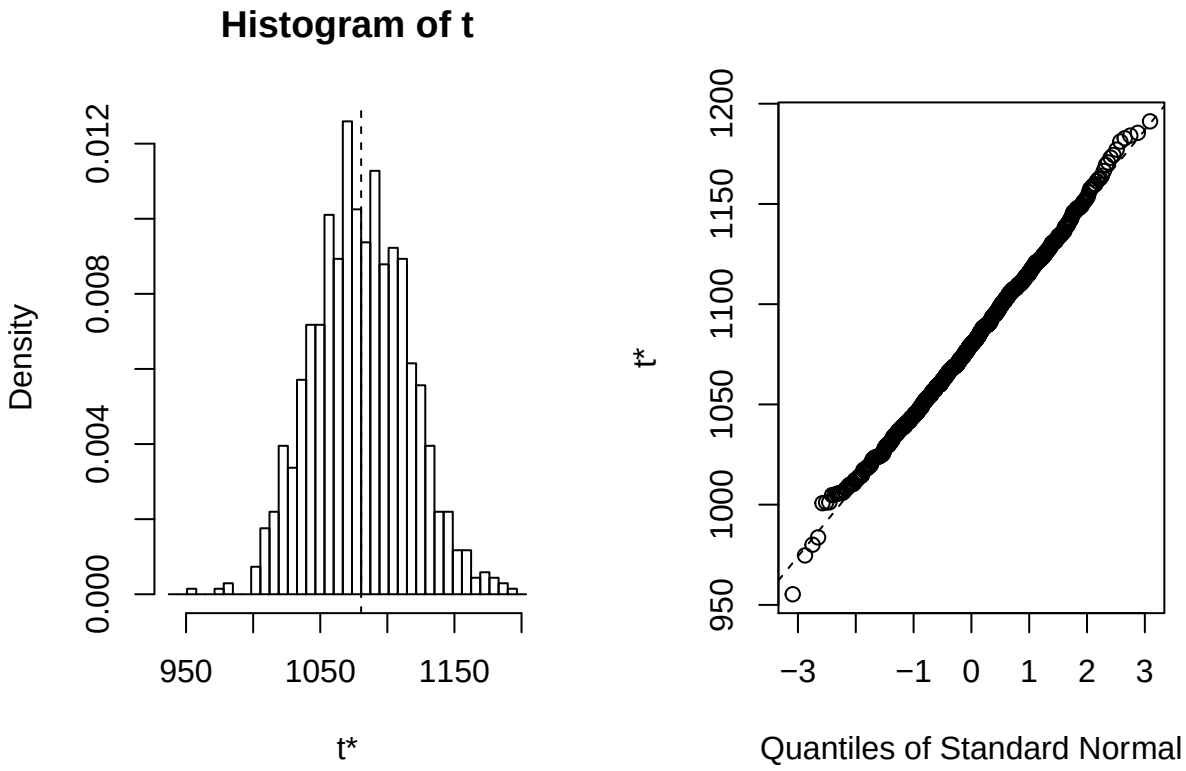
```r
# Bias-correction and Varience of Price
boot.fn <- function(data,index){
        d <- data[index]
        res <- mean(d)
}

boot.result <- boot(data=price_data$Price, statistic=boot.fn, R=1000)
bias_cor <- 2*mean(price_data$Price)-mean(boot.result$t)


list("bias_correction"=bias_cor, "variance_of_the_mean_price"=35.93^2)
```

```
## $bias_correction
## [1] 1080.4343636363637
##
## $variance_of_the_mean_price
## [1] 1290.9648999999999
```

```r
boot.result
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = price_data$Price, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##               original               bias           std. error
## t1* 1080.4727272727273 0.038363636363555997 35.455872075252685
```

```r
plot(boot.result)
```




```
boot.ci(boot.result)
```

```
## Warning in boot.ci(boot.result): bootstrap variances needed for studentized
## intervals
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.result)
##
## Intervals :
## Level      Normal              Basic
## 95%   (1011, 1150 )   (1009, 1148 )
##
## Level     Percentile            BCa
## 95%   (1013, 1151 )   (1018, 1159 )
## Calculations and Intervals on Original Scale
```

Estimate the variance of the mean price using the jackknife and compare it with the bootstrap estimate

Jacknife(n=B):

$$\widehat{Var[T(\cdot)]} = \frac{1}{n(n-1)} \sum_{i=1}^{n} (T_i^* - J(T))^2$$

where,

$$T_i^* = nT(D) - (n-1)T(D_i^*) \ , \qquad J(T) = \frac{1}{n}\sum_{i=1}^{n} T_i^*$$

When you compute the equation given aboce, you got

$$\frac{n-1}{n}\sum_{i=1}^{n}(T_i^* - J(T))^2$$

Reference:The Jackknife Estimation Method, Avery I. McIntosh (http://people.bu.edu/aimcinto/jackknife.pdf)

```
result <- numeric()
n <- NROW(price_data)

for (i in 1:n){
        updated_price <- price_data$Price[-i]
        result[i] <- mean(updated_price)
}

var_T <- (n-1)/n*sum((result-mean(result))^2)
mean_T <- mean(result)

cat("The variance from jacknife method is:", var_T)
```

```
## The variance from jacknife method is: 1320.9110440518652
```

Analysis: The obtained variance using Jackknife method is 1320.911 while using bootstrapping the obtained value was 1290.965. Considering the fact that Jackknife overestimate variance, the answer seems reasonable.

Compare the confidence intervals obtained with respect to their length and the location of the estimated mean in these intervals.

```
confidence_interval_jackknife <- c((mean_T - 1.96*var_T), (mean_T + 1.96*var_T))

confidence_interval_jackknife
```

```
## [1] -1508.5129190689286  3669.4583736143832
```

```
intervals <- c("(1150-1011)","(1148-1011)","(1149-1013)","(1146-1007)")
length <- c(1150-1011, 1148-1011,1149-1013,1146-1007)
Center_of_interval <- c((1150+1011)/2,(1148+1011)/2,(1149+1013)/2,(1146+1007)/2)

dt <- data.frame(intervals, length, Center_of_interval,row.names = c("Normal", "Basic", "Percentile", "

dt %>% kable(col.names = c("Confidence interval", "Length of interval","Center of interval"))
```

	Confidence interval	Length of interval	Center of interval
Normal	(1150-1011)	139	1080.5
Basic	(1148-1011)	137	1079.5
Percentile	(1149-1013)	136	1081.0
BCa	(1146-1007)	139	1076.5

Genetic Algorithm

Find the max of one-dimensional function

$$f(x) := \frac{x^2}{e^x} - 2e^{-}\frac{(9sinx)}{(x^2+x+1)}$$

```
function_f <- function(x){
answer <- x^2/exp(x) - 2*exp(-(9*sin(x))/(x^2+x+1))
return(answer)
}
```

Crossover function

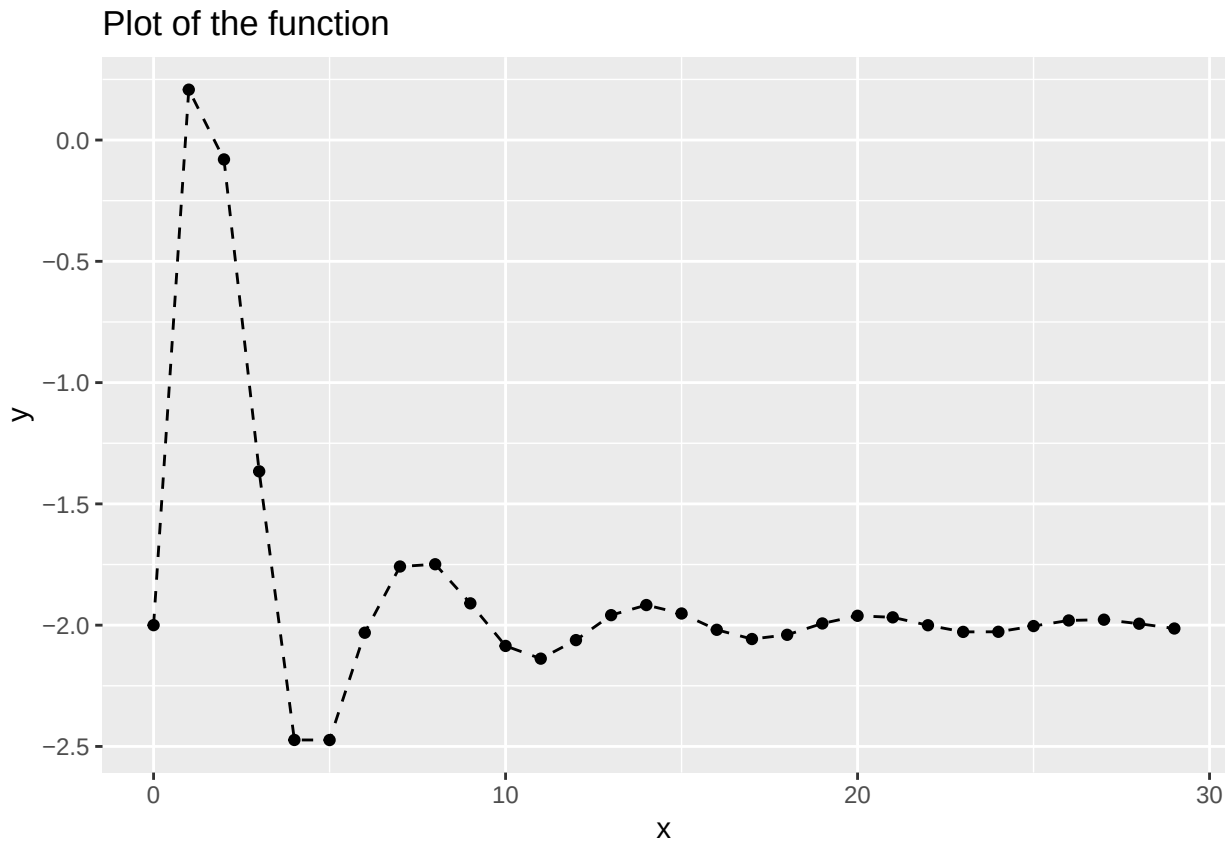
```
crossover <- function(x,y){
answer <- (x+y) * 0.5
return(answer)
}
```

Mutate Function

```
mutate <- function(x){
answer <- (x^2)%%30
return(answer)
}
```

Plot of function

```
y <- vector("double", length = 30)
x <- seq(from=0,to=29,by=1)
for(i in 0:29){y[i+1] <- function_f(x=i)}

data <- data.frame(cbind(x,y))
data$flag <- "original_values"
data$x <- as.numeric(as.character(data$x))
data$y <- as.numeric(as.character(data$y))

ggplot(data, aes(x=x, y=y, group=1)) +
geom_point() +
geom_line(linetype = "dashed") +
ggtitle("Plot of the function")
```




```
cat("The maximum values of the function is at x =",which.max(data$y), "and the value is = ",max(data$y))
```

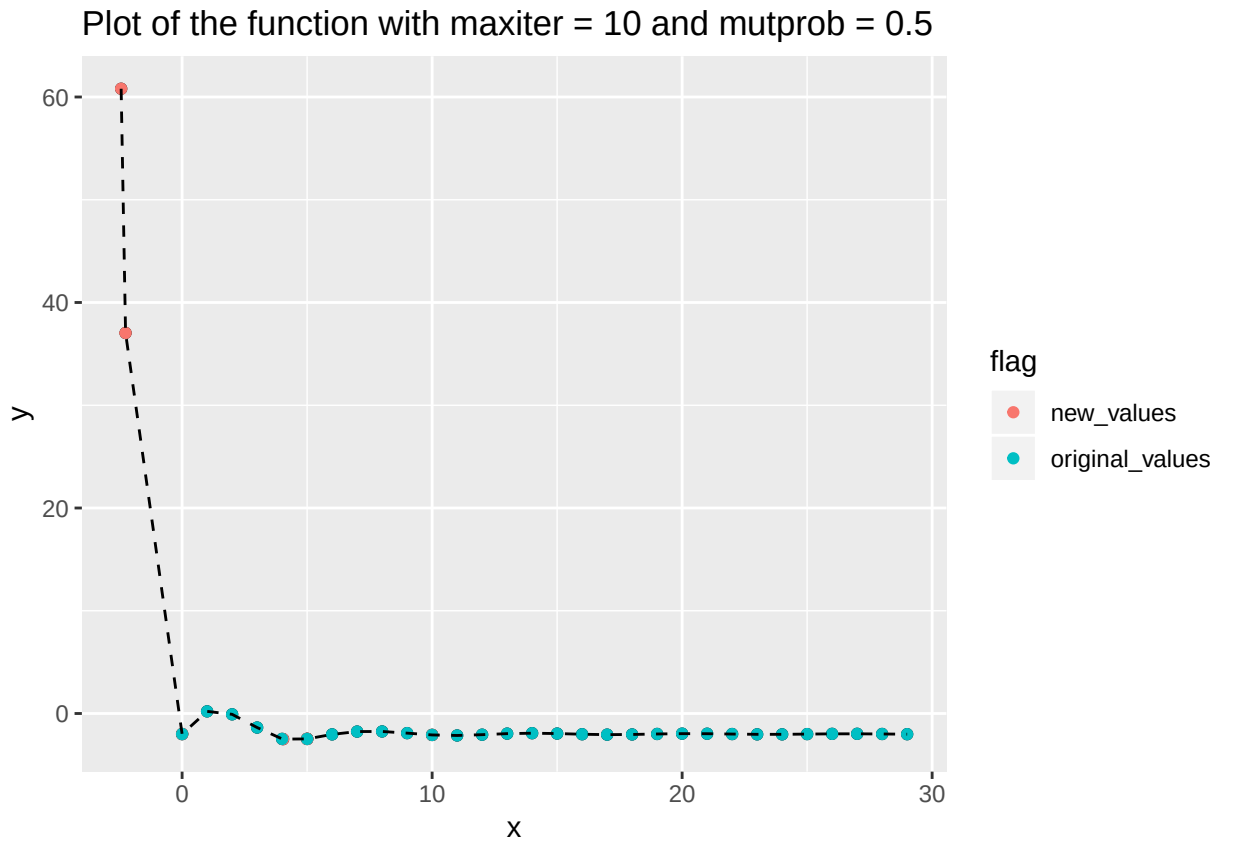
```
## The maximum values of the function is at x = 2 and the value is =  0.20766879224596799
```

```
inital_population <- seq(from=0, to=30, by=5)
Values <- vector("double", length = 7)
for(i in seq_along(inital_population)){Values[i] <- function_f(x=inital_population[i])}
```

Main function

```
myfunction <- function(maxiter, mutprob){

# maxiter = 10
# mutprob = 0.5


for(i in seq_along(1:maxiter)){
# sampling and getting parent and victim position
parent_index <- sample(x = seq(from = 1, to=7, by=1), size = 2, replace = TRUE)
parent_1 <- Values[parent_index[[1]]]
parent_2 <- Values[parent_index[[2]]]
victim <- which.min(order(Values))

# performing crossover and mutate based on prob
kid <- crossover(x = parent_1, y = parent_2)
```

```
if(as.numeric(rbinom(1,1,mutprob))==1){kid <- mutate(x=kid)}
inital_population[victim] <- kid

# generating new values
for(j in seq_along(inital_population)){Values[j] <- function_f(x=inital_population[j])}

}
new_max <- max(Values)
index_max_val <- inital_population[which.max(Values)]

data2 <- data.frame(cbind(x=inital_population,y=Values))
data2$flag <- "new_values"
data2$x <- as.numeric(as.character(data2$x))
data2$y <- as.numeric(as.character(data2$y))
data3 <- rbind(data2,data)

answer <- ggplot(data3, aes(x=x, y=y, group=1)) +
geom_point() +
geom_point(aes(colour=flag)) +
geom_line(linetype = "dashed") +
ggtitle(paste0("Plot of the function with maxiter = ",maxiter," and mutprob = ",mutprob))
return(answer)
}

myfunction(maxiter=10, mutprob=0.5)
```



The EM Algorithm

Given a random sample of size n, with observed sample $\mathbf{X} = (X_1, ..., X_m)$ and *missing* random sample $\mathbf{Z} = Z_{m+1}, ..., Z_n$ we seek to compute

$$\hat{\theta} = \text{ arg max } L(\theta|\mathbf{X}, \mathbf{Z})$$

Although $\mathbf{Z}$ is unobservable, we assume that $(\mathbf{X}, \mathbf{Z}) \sim \mathbf{f}(\mathbf{x}, \mathbf{z}|\theta)$.

We place a conditional distribion on $\mathbf{Z}$ given the observed data $\mathbf{x}$,

$$k(\mathbf{z}|\theta, \mathbf{x}) = f(\mathbf{x}, \mathbf{z}|\theta)/g(\mathbf{x}|\theta)$$

Here we assume that that $\mathbf{X} \sim g(\mathbf{x}|\theta)$, where

$$g(\mathbf{x}|\theta) = \int \mathbf{f}(\mathbf{x}, \mathbf{z}|\theta)\mathbf{dz}$$

Denote the complete-data likelihood as $L^c(\theta|\mathbf{x}, \mathbf{z})$ and the observed-data likelihood as $L(\theta|\mathbf{x})$. Then, for any value of θ, θ_i

$$logL(\theta|\mathbf{x}) = E[logL^c(\theta|\mathbf{x}, \mathbf{z})] - E[logk(\mathbf{Z}|\theta_\mathbf{i}, \mathbf{x})]$$

where the expectation is with respect to $k(\mathbf{z}|\theta_i, \mathbf{x})$. We can rewrite this as

$$E[logL^c(\theta|\mathbf{x}, \mathbf{z})] = logL(\theta|\mathbf{x}) + E[logk(\mathbf{Z}|\theta_i, \mathbf{x})]$$

where our focus is concerned with maximizing $E[logL^c(\theta|\mathbf{x}, \mathbf{z})]$.

Denoting $E[logL^c(\theta|\mathbf{x}, \mathbf{z})] = Q(\theta|\theta_i, \mathbf{x})$, the EM algorith iterates through values of θ_i by maximizing $Q(\theta|\theta_i, \mathbf{x})$.

The EM Algorithm

`Pick a starting value` $\hat{\theta_0}$

`Then for i in 1:n do`

`1. Compute` (E-step)

$$Q(\theta|\theta_{i-1}, \mathbf{x}) = E[logL^c(\theta|\mathbf{x}, \mathbf{z})]$$

`where the expectation is with respect to` $k(\mathbf{Z}|\theta_\mathbf{i}, \mathbf{x})$

`2. Maximize` $Q(\theta|\theta_{i-1}, \mathbf{x})$ `in` θ `and take`

$$\hat{\theta_i} = \text{ arg max } Q(\theta|\theta_{i-1}, \mathbf{x})$$

`repeat until convergence criteria is met`

Normal Distribution with missing values using EM

Suppose $X = (x_1, ..., x_n)^T$ is a random sample from $N(\mu, 1)$. Let the observations be in order such that $x_1 < x_2 < ... < x_n$. Suppose that after time c, values are censored or missing, such that only $x_1, ..., x_m$ are observed, and $x_{m+1}, ..., x_n$ are unobserved. Then, $r = (n - m)$ would be the quantity missing. We will use the EM and MCEM algorithms to find approximations for μ. Let $Z = (x_{m+1}, ..., x_n)^T$.

First, construct the likelihood function.

$$
\begin{aligned}
L(\mu|x) &= \prod^{m} f(x_i|\mu,1) \times \prod^{r} f(z_i|\mu,1) \\
&= (2\pi)^{-n/2} exp(-\frac{1}{2}\sum_{i=1}^{m}(x_i-\mu)^2) \times exp(-\frac{1}{2}\sum_{i=1}^{m}(z_i-\mu)^2) \\
&\propto exp(-\frac{1}{2}\sum_{i=1}^{m}(x_i-\mu)^2) \times exp(-\frac{1}{2}\sum_{i=1}^{m}(z_i-\mu)^2)
\end{aligned}
$$

The log-likelihood is then

$$
ln(L(\mu|X)) = -\frac{1}{2}\sum_{i=1}^{m}(x_i-\mu)^2) - \frac{1}{2}\sum_{i=1}^{m}(z_i-\mu)^2
$$

We now find the conditional expectation $E[z_i|X]$

$$
\begin{aligned}
E[z_i|X] = E[z_i|x>c] &= \int_c^{\infty} \frac{P(x_i>x|x_i>c)}{P(x_i>c)} \\
&= \mu + \sigma\frac{\phi(c-\mu)}{1-\Phi(c-\mu)}
\end{aligned}
$$

$$
\begin{aligned}
Q(\mu|\mu_t) &= -\frac{1}{2}\sum_{i=1}^{m}(x_i-\mu)^2) - \sum E[z_i|X] \\
&= -\frac{1}{2}\sum_{i=1}^{m}(x_i-\mu)^2) - \sum E[z|X] \\
&= -\frac{1}{2}\sum_{i=1}^{m}(x_i-\mu)^2) - (n-m)E[z|X]
\end{aligned}
$$

The MLE for μ is then,

$$
\begin{aligned}
\mu_{t+1} &= \frac{m\bar{x}}{n} + \frac{(n-m)E[z|X]}{n} \\
&= \frac{m\bar{x}}{n} + \frac{(n-m)(\mu_t)}{n} + \frac{(n-m)\phi(c-\mu_t)}{n\Phi(c-\mu_t)}
\end{aligned}
$$

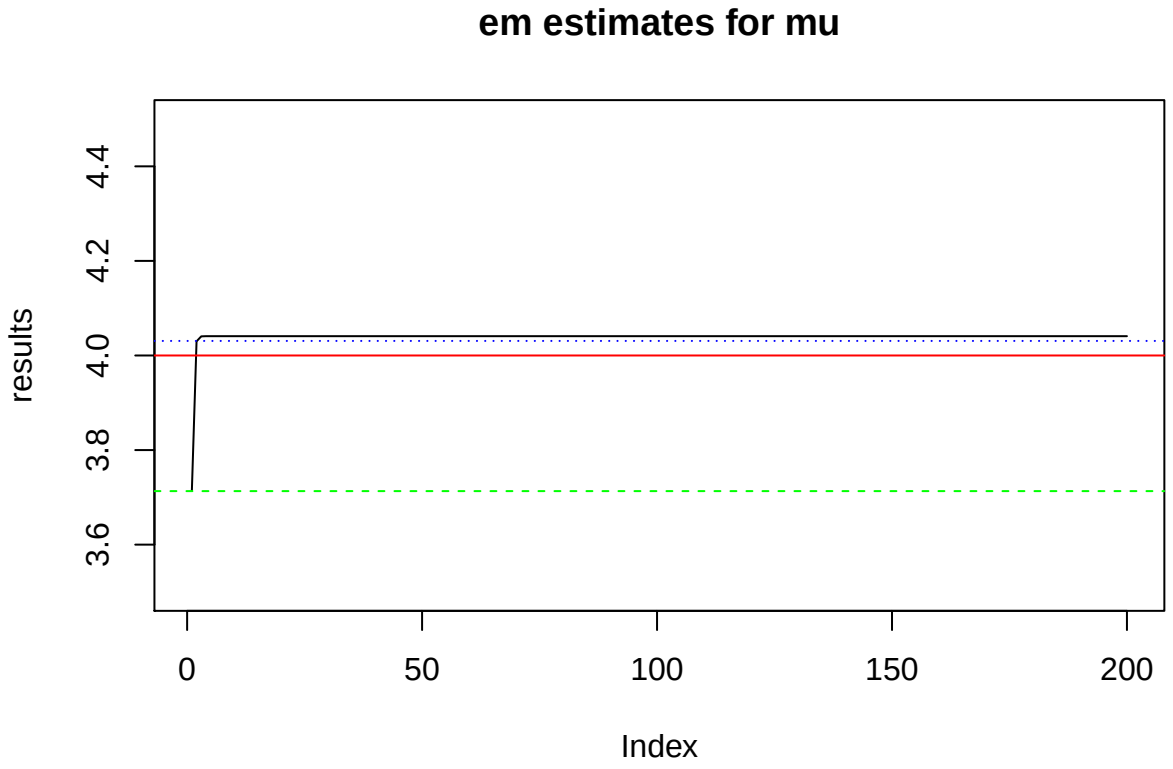
```
set.seed(2345)
n = 100
mu = 4
sd = 1
x = rnorm(n, mu, sd) ## generate some data
c = 5 ## time cut off
w = x[x < c] ## obtain samples before time cut off
m = sum(x < c) ## number of observed samples
wbar = mean(w) ## observed mean
r = n - m ## difference in sample size
```

```
N = 200
mu_new = wbar
results = numeric(N)
for(i in 1:N){
    results[i] = mu_new
```

```
    mu_old = mu_new
    mu_new = m*wbar/n + (r*mu_old/n) +
      (r/n)*sd*(dnorm(c - mu_old))/(1 - pnorm(c - mu_old))  ## r/n instead of 1/n
    #print(mu_new)
}

print(tail(results))
```

```
## [1] 4.0408209179195476 4.0408209179195476 4.0408209179195476
## [4] 4.0408209179195476 4.0408209179195476 4.0408209179195476
```

```
plot(results, type = "l", main = "em estimates for mu", ylim = c(3.5, 4.5))
abline(h = mu, col = "red")
abline(h = wbar, col = "green", lty = 2)
abline(h = mean(x), col = "blue", lty = 3)
```



EM using Monte Carlo

A MC flavor of the EM algorithm

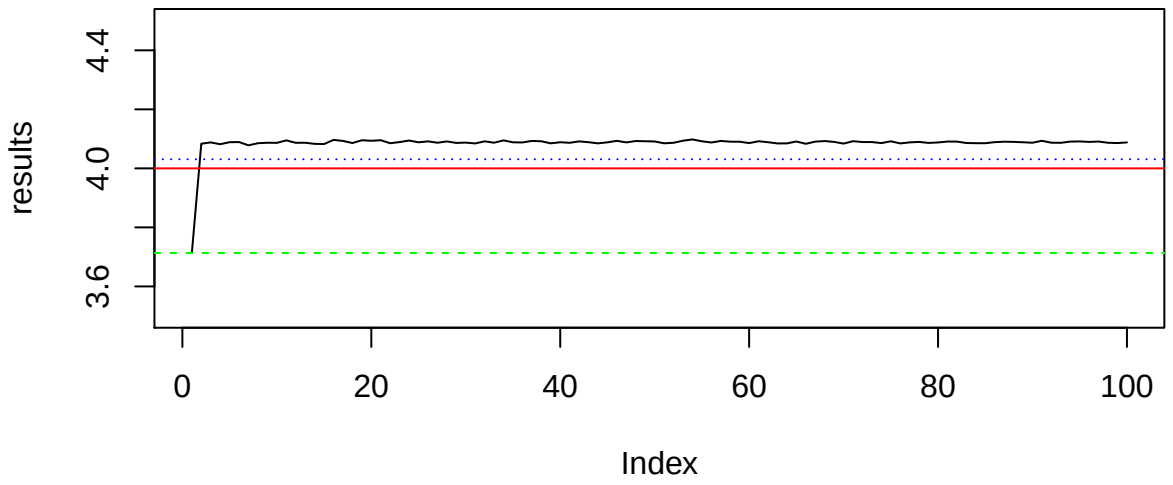
1. Draw missing data sets $\mathbf{Z_1}, \mathbf{Z_2}, ..., \mathbf{Z_m} \sim f_{Z|X}(z|x, \theta_i)$ where each $\mathbf{Z_i}$ is a vector of all missing values needed to complete the observed data set $(\mathbf{X}, \mathbf{Z})$.
2. Calculate $\bar{Q}(\theta|\theta_{i-1}, X, \mathbf{Z_1}, ..., \mathbf{Z_m}) = \frac{1}{m}\sum_{i=1}^{m} Q(\theta|\theta_{i-1}, X, \mathbf{Z_i})$

```
set.seed(2345)
n = 100
```

```r
mu = 4
sd = 1
x = rnorm(n, mu, sd)
c = 5
w = x[x < c]
m = sum(x < c)
wbar = mean(w)
r = n - m
```

```r
M = 10
N = 100
mu_new = wbar
results = numeric(N)
for(i in 1:N){
    results[i] = mu_new
    mu_old = mu_new
    ## abs(N(0,1)) + mu_old + (c - mu_old) to *approximate*
    ## the truncated samples we need
    Z = matrix(data = (c - mu_old) + (mu_old +  abs(rnorm(n = r*M, mean = 0, sd = 1))),
        nrow = r, ncol = M)
    mu_new = (m*wbar/n) + mean(colMeans(Z))*r/n
    M = M + 1
}
```

```r
plot(results, type = "l", ylim = c(3.5, 4.5))
abline(h = mu, col = "red")
abline(h = wbar, col = "green", lty = 2)
abline(h = mean(x), col = "blue", lty = 3)
```



Integration Help

Table of Integrals*

Basic Forms

$$\int x^n dx = \frac{1}{n+1}x^{n+1} \quad (1)$$

$$\int \frac{1}{x}dx = \ln|x| \quad (2)$$

$$\int udv = uv - \int vdu \quad (3)$$

$$\int \frac{1}{ax+b}dx = \frac{1}{a}\ln|ax+b| \quad (4)$$

Integrals of Rational Functions

$$\int \frac{1}{(x+a)^2}dx = -\frac{1}{x+a} \quad (5)$$

$$\int (x+a)^n dx = \frac{(x+a)^{n+1}}{n+1}, n \neq -1 \quad (6)$$

$$\int x(x+a)^n dx = \frac{(x+a)^{n+1}((n+1)x-a)}{(n+1)(n+2)} \quad (7)$$

$$\int \frac{1}{1+x^2}dx = \tan^{-1}x \quad (8)$$

$$\int \frac{1}{a^2+x^2}dx = \frac{1}{a}\tan^{-1}\frac{x}{a} \quad (9)$$

$$\int \frac{x}{a^2+x^2}dx = \frac{1}{2}\ln|a^2+x^2| \quad (10)$$

$$\int \frac{x^2}{a^2+x^2}dx = x - a\tan^{-1}\frac{x}{a} \quad (11)$$

$$\int \frac{x^3}{a^2+x^2}dx = \frac{1}{2}x^2 - \frac{1}{2}a^2\ln|a^2+x^2| \quad (12)$$

$$\int \frac{1}{ax^2+bx+c}dx = \frac{2}{\sqrt{4ac-b^2}}\tan^{-1}\frac{2ax+b}{\sqrt{4ac-b^2}} \quad (13)$$

$$\int \frac{1}{(x+a)(x+b)}dx = \frac{1}{b-a}\ln\frac{a+x}{b+x},\ a \neq b \quad (14)$$

$$\int \frac{x}{(x+a)^2}dx = \frac{a}{a+x} + \ln|a+x| \quad (15)$$

$$\int \frac{x}{ax^2+bx+c}dx = \frac{1}{2a}\ln|ax^2+bx+c| - \frac{b}{a\sqrt{4ac-b^2}}\tan^{-1}\frac{2ax+b}{\sqrt{4ac-b^2}} \quad (16)$$

Integrals with Roots

$$\int \sqrt{x-a}dx = \frac{2}{3}(x-a)^{3/2} \quad (17)$$

$$\int \frac{1}{\sqrt{x\pm a}}dx = 2\sqrt{x\pm a} \quad (18)$$

$$\int \frac{1}{\sqrt{a-x}}dx = -2\sqrt{a-x} \quad (19)$$

$$\int x\sqrt{x-a}dx = \frac{2}{3}a(x-a)^{3/2} + \frac{2}{5}(x-a)^{5/2} \quad (20)$$

$$\int \sqrt{ax+b}dx = \left(\frac{2b}{3a} + \frac{2x}{3}\right)\sqrt{ax+b} \quad (21)$$

$$\int (ax+b)^{3/2}dx = \frac{2}{5a}(ax+b)^{5/2} \quad (22)$$

$$\int \frac{x}{\sqrt{x\pm a}}dx = \frac{2}{3}(x\mp 2a)\sqrt{x\pm a} \quad (23)$$

$$\int \sqrt{\frac{x}{a-x}}dx = -\sqrt{x(a-x)} - a\tan^{-1}\frac{\sqrt{x(a-x)}}{x-a} \quad (24)$$

$$\int \sqrt{\frac{x}{a+x}}dx = \sqrt{x(a+x)} - a\ln\left[\sqrt{x}+\sqrt{x+a}\right] \quad (25)$$

$$\int x\sqrt{ax+b}dx = \frac{2}{15a^2}(-2b^2+abx+3a^2x^2)\sqrt{ax+b} \quad (26)$$

$$\int \sqrt{x(ax+b)}dx = \frac{1}{4a^{3/2}}\left[(2ax+b)\sqrt{ax(ax+b)} - b^2\ln\left|a\sqrt{x}+\sqrt{a(ax+b)}\right|\right] \quad (27)$$

$$\int \sqrt{x^3(ax+b)}dx = \left[\frac{b}{12a} - \frac{b^2}{8a^2x} + \frac{x}{3}\right]\sqrt{x^3(ax+b)} + \frac{b^3}{8a^{5/2}}\ln\left|a\sqrt{x}+\sqrt{a(ax+b)}\right| \quad (28)$$

$$\int \sqrt{x^2\pm a^2}dx = \frac{1}{2}x\sqrt{x^2\pm a^2} \pm \frac{1}{2}a^2\ln\left|x+\sqrt{x^2\pm a^2}\right| \quad (29)$$

$$\int \sqrt{a^2-x^2}dx = \frac{1}{2}x\sqrt{a^2-x^2} + \frac{1}{2}a^2\tan^{-1}\frac{x}{\sqrt{a^2-x^2}} \quad (30)$$

$$\int x\sqrt{x^2\pm a^2}dx = \frac{1}{3}\left(x^2\pm a^2\right)^{3/2} \quad (31)$$

$$\int \frac{1}{\sqrt{x^2\pm a^2}}dx = \ln\left|x+\sqrt{x^2\pm a^2}\right| \quad (32)$$

$$\int \frac{1}{\sqrt{a^2-x^2}}dx = \sin^{-1}\frac{x}{a} \quad (33)$$

$$\int \frac{x}{\sqrt{x^2\pm a^2}}dx = \sqrt{x^2\pm a^2} \quad (34)$$

$$\int \frac{x}{\sqrt{a^2-x^2}}dx = -\sqrt{a^2-x^2} \quad (35)$$

$$\int \frac{x^2}{\sqrt{x^2\pm a^2}}dx = \frac{1}{2}x\sqrt{x^2\pm a^2} \mp \frac{1}{2}a^2\ln\left|x+\sqrt{x^2\pm a^2}\right| \quad (36)$$

$$\int \sqrt{ax^2+bx+c}dx = \frac{b+2ax}{4a}\sqrt{ax^2+bx+c} + \frac{4ac-b^2}{8a^{3/2}}\ln\left|2ax+b+2\sqrt{a(ax^2+bx+c)}\right| \quad (37)$$

$$\int x\sqrt{ax^2+bx+c} = \frac{1}{48a^{5/2}}\left(2\sqrt{a}\sqrt{ax^2+bx+c} \times\left(-3b^2+2abx+8a(c+ax^2)\right) + 3(b^3-4abc)\ln\left|b+2ax+2\sqrt{a}\sqrt{ax^2+bx+c}\right|\right) \quad (38)$$

$$\int \frac{1}{\sqrt{ax^2+bx+c}}dx = \frac{1}{\sqrt{a}}\ln\left|2ax+b+2\sqrt{a(ax^2+bx+c)}\right| \quad (39)$$

$$\int \frac{x}{\sqrt{ax^2+bx+c}}dx = \frac{1}{a}\sqrt{ax^2+bx+c} - \frac{b}{2a^{3/2}}\ln\left|2ax+b+2\sqrt{a(ax^2+bx+c)}\right| \quad (40)$$

$$\int \frac{dx}{(a^2+x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2+x^2}} \quad (41)$$

Integrals with Logarithms

$$\int \ln ax dx = x\ln ax - x \quad (42)$$

$$\int \frac{\ln ax}{x}dx = \frac{1}{2}(\ln ax)^2 \quad (43)$$

$$\int \ln(ax+b)dx = \left(x+\frac{b}{a}\right)\ln(ax+b) - x, a \neq 0 \quad (44)$$

$$\int \ln(x^2+a^2)\,\mathrm{dx} = x\ln(x^2+a^2) + 2a\tan^{-1}\frac{x}{a} - 2x \quad (45)$$

$$\int \ln(x^2-a^2)\,\mathrm{dx} = x\ln(x^2-a^2) + a\ln\frac{x+a}{x-a} - 2x \quad (46)$$

$$\int \ln\left(ax^2+bx+c\right)dx = \frac{1}{a}\sqrt{4ac-b^2}\tan^{-1}\frac{2ax+b}{\sqrt{4ac-b^2}} - 2x + \left(\frac{b}{2a}+x\right)\ln\left(ax^2+bx+c\right) \quad (47)$$

$$\int x\ln(ax+b)dx = \frac{bx}{2a} - \frac{1}{4}x^2 + \frac{1}{2}\left(x^2-\frac{b^2}{a^2}\right)\ln(ax+b) \quad (48)$$

$$\int x\ln\left(a^2-b^2x^2\right)dx = -\frac{1}{2}x^2 + \frac{1}{2}\left(x^2-\frac{a^2}{b^2}\right)\ln\left(a^2-b^2x^2\right) \quad (49)$$

Integrals with Exponentials

$$\int e^{ax}dx = \frac{1}{a}e^{ax} \quad (50)$$

$$\int \sqrt{x}e^{ax}dx = \frac{1}{a}\sqrt{x}e^{ax} + \frac{i\sqrt{\pi}}{2a^{3/2}}\mathrm{erf}\left(i\sqrt{ax}\right), \text{ where } \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt \quad (51)$$

$$\int xe^x dx = (x-1)e^x \quad (52)$$

$$\int xe^{ax}dx = \left(\frac{x}{a} - \frac{1}{a^2}\right)e^{ax} \quad (53)$$

$$\int x^2e^x dx = \left(x^2-2x+2\right)e^x \quad (54)$$

$$\int x^2e^{ax}dx = \left(\frac{x^2}{a} - \frac{2x}{a^2} + \frac{2}{a^3}\right)e^{ax} \quad (55)$$

$$\int x^3e^x dx = \left(x^3-3x^2+6x-6\right)e^x \quad (56)$$

$$\int x^n e^{ax}\,\mathrm{dx} = \frac{x^n e^{ax}}{a} - \frac{n}{a}\int x^{n-1}e^{ax}\,\mathrm{dx} \quad (57)$$

$$\int x^n e^{ax}\,\mathrm{dx} = \frac{(-1)^n}{a^{n+1}}\Gamma[1+n,-ax], \text{ where } \Gamma(a,x) = \int_x^\infty t^{a-1}e^{-t}\,\mathrm{dt} \quad (58)$$

$$\int e^{ax^2}\,\mathrm{dx} = -\frac{i\sqrt{\pi}}{2\sqrt{a}}\mathrm{erf}\left(ix\sqrt{a}\right) \quad (59)$$

$$\int e^{-ax^2}\,\mathrm{dx} = \frac{\sqrt{\pi}}{2\sqrt{a}}\mathrm{erf}\left(x\sqrt{a}\right) \quad (60)$$

$$\int xe^{-ax^2}\,\mathrm{dx} = -\frac{1}{2a}e^{-ax^2} \quad (61)$$

$$\int x^2e^{-ax^2}\,\mathrm{dx} = \frac{1}{4}\sqrt{\frac{\pi}{a^3}}\mathrm{erf}(x\sqrt{a}) - \frac{x}{2a}e^{-ax^2} \quad (62)$$

Integrals with Trigonometric Functions

$$\int \sin ax dx = -\frac{1}{a}\cos ax \tag{63}$$

$$\int \sin^2 ax dx = \frac{x}{2} - \frac{\sin 2ax}{4a} \tag{64}$$

$$\int \sin^n ax dx = -\frac{1}{a}\cos ax \ {}_2F_1\left[\frac{1}{2}, \frac{1-n}{2}, \frac{3}{2}, \cos^2 ax\right] \tag{65}$$

$$\int \sin^3 ax dx = -\frac{3\cos ax}{4a} + \frac{\cos 3ax}{12a} \tag{66}$$

$$\int \cos ax dx = \frac{1}{a}\sin ax \tag{67}$$

$$\int \cos^2 ax dx = \frac{x}{2} + \frac{\sin 2ax}{4a} \tag{68}$$

$$\int \cos^p ax dx = -\frac{1}{a(1+p)}\cos^{1+p} ax \times {}_2F_1\left[\frac{1+p}{2}, \frac{1}{2}, \frac{3+p}{2}, \cos^2 ax\right] \tag{69}$$

$$\int \cos^3 ax dx = \frac{3\sin ax}{4a} + \frac{\sin 3ax}{12a} \tag{70}$$

$$\int \cos ax \sin bx dx = \frac{\cos[(a-b)x]}{2(a-b)} - \frac{\cos[(a+b)x]}{2(a+b)}, a \neq b \tag{71}$$

$$\int \sin^2 ax \cos bx dx = -\frac{\sin[(2a-b)x]}{4(2a-b)} + \frac{\sin bx}{2b} - \frac{\sin[(2a+b)x]}{4(2a+b)} \tag{72}$$

$$\int \sin^2 x \cos x dx = \frac{1}{3}\sin^3 x \tag{73}$$

$$\int \cos^2 ax \sin bx dx = \frac{\cos[(2a-b)x]}{4(2a-b)} - \frac{\cos bx}{2b} - \frac{\cos[(2a+b)x]}{4(2a+b)} \tag{74}$$

$$\int \cos^2 ax \sin ax dx = -\frac{1}{3a}\cos^3 ax \tag{75}$$

$$\int \sin^2 ax \cos^2 bx dx = \frac{x}{4} - \frac{\sin 2ax}{8a} - \frac{\sin[2(a-b)x]}{16(a-b)} + \frac{\sin 2bx}{8b} - \frac{\sin[2(a+b)x]}{16(a+b)} \tag{76}$$

$$\int \sin^2 ax \cos^2 ax dx = \frac{x}{8} - \frac{\sin 4ax}{32a} \tag{77}$$

$$\int \tan ax dx = -\frac{1}{a}\ln \cos ax \tag{78}$$

$$\int \tan^2 ax dx = -x + \frac{1}{a}\tan ax \tag{79}$$

$$\int \tan^n ax dx = \frac{\tan^{n+1} ax}{a(1+n)} \times {}_2F_1\left(\frac{n+1}{2}, 1, \frac{n+3}{2}, -\tan^2 ax\right) \tag{80}$$

$$\int \tan^3 ax dx = \frac{1}{a}\ln \cos ax + \frac{1}{2a}\sec^2 ax \tag{81}$$

$$\int \sec x dx = \ln|\sec x + \tan x| = 2\tanh^{-1}\left(\tan\frac{x}{2}\right) \tag{82}$$

$$\int \sec^2 ax dx = \frac{1}{a}\tan ax \tag{83}$$

$$\int \sec^3 x \, \mathrm{dx} = \frac{1}{2}\sec x \tan x + \frac{1}{2}\ln|\sec x + \tan x| \tag{84}$$

$$\int \sec x \tan x dx = \sec x \tag{85}$$

$$\int \sec^2 x \tan x dx = \frac{1}{2}\sec^2 x \tag{86}$$

$$\int \sec^n x \tan x dx = \frac{1}{n}\sec^n x, n \neq 0 \tag{87}$$

$$\int \csc x dx = \ln\left|\tan\frac{x}{2}\right| = \ln|\csc x - \cot x| + C \tag{88}$$

$$\int \csc^2 ax dx = -\frac{1}{a}\cot ax \tag{89}$$

$$\int \csc^3 x dx = -\frac{1}{2}\cot x \csc x + \frac{1}{2}\ln|\csc x - \cot x| \tag{90}$$

$$\int \csc^n x \cot x dx = -\frac{1}{n}\csc^n x, n \neq 0 \tag{91}$$

$$\int \sec x \csc x dx = \ln|\tan x| \tag{92}$$

Products of Trigonometric Functions and Monomials

$$\int x\cos x dx = \cos x + x\sin x \tag{93}$$

$$\int x\cos ax dx = \frac{1}{a^2}\cos ax + \frac{x}{a}\sin ax \tag{94}$$

$$\int x^2\cos x dx = 2x\cos x + \left(x^2 - 2\right)\sin x \tag{95}$$

$$\int x^2\cos ax dx = \frac{2x\cos ax}{a^2} + \frac{a^2x^2 - 2}{a^3}\sin ax \tag{96}$$

$$\int x^n cosx dx = -\frac{1}{2}(i)^{n+1}\left[\Gamma(n+1, -ix) + (-1)^n\Gamma(n+1, ix)\right] \tag{97}$$

$$\int x^n cosax dx = \frac{1}{2}(ia)^{1-n}\left[(-1)^n\Gamma(n+1, -iax) - \Gamma(n+1, ixa)\right] \tag{98}$$

$$\int x\sin x dx = -x\cos x + \sin x \tag{99}$$

$$\int x\sin ax dx = -\frac{x\cos ax}{a} + \frac{\sin ax}{a^2} \tag{100}$$

$$\int x^2\sin x dx = \left(2 - x^2\right)\cos x + 2x\sin x \tag{101}$$

$$\int x^2\sin ax dx = \frac{2 - a^2x^2}{a^3}\cos ax + \frac{2x\sin ax}{a^2} \tag{102}$$

$$\int x^n\sin x dx = -\frac{1}{2}(i)^n\left[\Gamma(n+1, -ix) - (-1)^n\Gamma(n+1, -ix)\right] \tag{103}$$

Products of Trigonometric Functions and Exponentials

$$\int e^x\sin x dx = \frac{1}{2}e^x(\sin x - \cos x) \tag{104}$$

$$\int e^{bx}\sin ax dx = \frac{1}{a^2+b^2}e^{bx}(b\sin ax - a\cos ax) \tag{105}$$

$$\int e^x\cos x dx = \frac{1}{2}e^x(\sin x + \cos x) \tag{106}$$

$$\int e^{bx}\cos ax dx = \frac{1}{a^2+b^2}e^{bx}(a\sin ax + b\cos ax) \tag{107}$$

$$\int xe^x\sin x dx = \frac{1}{2}e^x(\cos x - x\cos x + x\sin x) \tag{108}$$

$$\int xe^x\cos x dx = \frac{1}{2}e^x(x\cos x - \sin x + x\sin x) \tag{109}$$

Integrals of Hyperbolic Functions

$$\int \cosh ax dx = \frac{1}{a}\sinh ax \tag{110}$$

$$\int e^{ax}\cosh bx dx = \begin{cases} \frac{e^{ax}}{a^2-b^2}[a\cosh bx - b\sinh bx] & a \neq b \\ \frac{e^{2ax}}{4a} + \frac{x}{2} & a = b \end{cases} \tag{111}$$

$$\int \sinh ax dx = \frac{1}{a}\cosh ax \tag{112}$$

$$\int e^{ax}\sinh bx dx = \begin{cases} \frac{e^{ax}}{a^2-b^2}[-b\cosh bx + a\sinh bx] & a \neq b \\ \frac{e^{2ax}}{4a} - \frac{x}{2} & a = b \end{cases} \tag{113}$$

$$\int e^{ax}\tanh bx dx = \begin{cases} \frac{e^{(a+2b)x}}{(a+2b)}{}_2F_1\left[1 + \frac{a}{2b}, 1, 2 + \frac{a}{2b}, -e^{2bx}\right] \\ \quad -\frac{1}{a}e^{ax}{}_2F_1\left[\frac{a}{2b}, 1, 1E, -e^{2bx}\right] & a \neq b \\ \frac{e^{ax} - 2\tan^{-1}[e^{ax}]}{a} & a = b \end{cases} \tag{114}$$

$$\int \tanh ax \, dx = \frac{1}{a}\ln\cosh ax \tag{115}$$

$$\int \cos ax\cosh bx dx = \frac{1}{a^2+b^2}\left[a\sin ax\cosh bx + b\cos ax\sinh bx\right] \tag{116}$$

$$\int \cos ax\sinh bx dx = \frac{1}{a^2+b^2}\left[b\cos ax\cosh bx + a\sin ax\sinh bx\right] \tag{117}$$

$$\int \sin ax\cosh bx dx = \frac{1}{a^2+b^2}\left[-a\cos ax\cosh bx + b\sin ax\sinh bx\right] \tag{118}$$

$$\int \sin ax\sinh bx dx = \frac{1}{a^2+b^2}\left[b\cosh bx\sin ax - a\cos ax\sinh bx\right] \tag{119}$$

$$\int \sinh ax\cosh ax dx = \frac{1}{4a}\left[-2ax + \sinh 2ax\right] \tag{120}$$

$$\int \sinh ax\cosh bx dx = \frac{1}{b^2-a^2}\left[b\cosh bx\sinh ax - a\cosh ax\sinh bx\right] \tag{121}$$

Lecture slides

Introduction
Computer Arithmetics

732A90
Computational Statistics

Krzysztof Bartoszek
(krzysztof.bartoszek@liu.se)

24 I 2017
Department of Computer and Information Science
Linköping University

Computational Statistics — In brief

- Even simple data analysis (mean, variance) by hand is tedious
- Today: huge datasets, models capturing system complexity, interactions (between variables **and** observations)
- We will discuss:
  - Being careful with calculations—overflow
  - Generation of random variables including correlated ones
  - Numerically optimizing functions, esp. maximum likelihood
  - Computing confidence (credible) intervals for distributions when analytical ones are unobtainable

Lesson structure

- Lectures
- Computer Labs
- Seminars
- Examination: Reports, seminars, final exam
- Final exam: computer based
- Answer in Swedish or English, as you prefer. **Presenting group has to answer in English!**
- Electronic reports as **.PDF**.
- Disclose **ALL** collaborations and sources.
- Provide source code (if used).
- E–mail contact: krzysztof.bartoszek@liu.se

Course materials, software

- Lecture slides
- Previous year's lecture slides (732A38)
- Handouts
- Various suggested www pages
- **Googling**
- James E. Gentle "Computational Statistics", Springer, 2009
- Geof H. Givens, Jennifer A. Hoeting "Computational Statistics", Wiley, 2013

- R

Course contents

- Recap: R
- Recap: Basic Statistics
- Computer Arithmetics (JG pages 85–105)
- Optimization (JG pages 241–272)
- Random Number Generation (JG pages 305–312, 325–328)
- Monte Carlo Methods (JG pages 312–318, 328 417–429, handout)
- Numerical Model Selection and Hypothesis Testing (JG pages 52–56, 424, 435–467, handout)
- Expectation Maximization Algorithm and Stochastic Optimization (JG pages 275–284, 296–298, 480–483)

Pages are recommended reading for each lecture, **NOT** exact lecture content. The lectures will build up on this material.

Computer Arithmetics

SHOULD YOU CARE?

Computer Arithmetics: Examples

Computations can be affected by magnitudes of numbers.

```
x<-0.5^10000;y<-0.4^10000;x/(x+y)+y/(x+y)
x<-0.5^1000;y<-0.4^1000;x/(x+y)+y/(x+y)
x<-0.1^1000;y<-0.2^1000;x/(x+y)+y/(x+y)
```

```
t<-rnorm(5,10^18,1);t[3]-t[4];t[1]-t[2]
```

```
x<-10^800;sd<-10^400;y<-x/sd;y
```

And your are doing estimation under a nice, fancy model ...

Data presentation

- Computers store information in binary form
  0 1 0 1 1 0 0 1
- 1Byte=8bits (typical counting unit)
- 1Word=32 or 64bits (depending on architecture)
- 1KB=1024bytes
- 1MB=1024KB
- and so on

QUESTION: Why binary form?

Character encoding

- ASCII (American Standard Code for Information Interchange)
  - 1 byte per character, 7 bits coding, 1 parity check or 0
  - $2^7 = 128$ characters can be encoded
  - "Usual" English letters, Arabic numerals, punctuation, i.e. "standard" keyboard
  - 1–31 control characters, 0: NULL **WHY?**
  - Design influenced by contemporary (1960) hardware
  - Extended ASCII: all 8 bits, 256 characters
- Unicode
  - 8, 16 or 32 bits encoding
  - "of more than 128,000 characters covering 135 modern and historic scripts, as well as multiple symbol sets" (Wikipedia)
- `read.csv()`, `read.table()` have `fileEncoding` argument

Arithmetic operations

R operations on binary

- Addition, multiplication: base–2 instead of base–10
- Subtraction: $A - B = A + (-B)$ (*twos–complement*)
- Division: tedious, rounded towards 0 `as.integer(17/3)`
- **Overflow:** adding two large numbers the sign bit can be treated as a high order bit and on some architectures results in a negative number

Fixed–point system (integers)

- We use the base–10 (decimal) system, e.g. $1234 = 1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0$
- We could use base–m system for any m
- Computers: base–2 (binary) system
- Each integer represented as: $A = a_0 \cdot 2^0 + a_1 \cdot 2^1 + a_2 \cdot 2^2 + \ldots$

EXERCISE: 5, 16, 17, 31, 32, 33, 255, 256 in binary
QUESTION: What is the range of a byte, word, double word?

- Negative numbers
  - **Leading bit:** first bit 0 if positive, 1 if negative
  - **Twos–complement:** sign bit 1, remaining bits to **opposite** value and then +1
    e.g. $5 = 00000101$, $-5 = 11111011$
    QUESTIONS $5 + (-5) =$?, try 12 and -12
  - Range: $[-2^{k-1}, 2^{k-1} - 1]$ on k bits **WHY?**

Floating–point system (rational, "real")

- How can we represent fractions (rational numbers)?
- Sign
- Exponent (**signed**, read standards if interested)
- Mantissa or Significand
- on 64bits:

sign (1bit)	Exponent (11bits)	Mantissa (52 bits)

$$\pm 0.d_1 d_2 \ldots d_p \cdot b^e \qquad b = 2, \quad p = 52$$

- **Range:** $\approx [-10^{300}, 10^{300}] \approx [-b^{e_{max}}, b^{e_{max}}]$

Floating–point system

- Rationals rounded towards the nearest computer float

```
options(digits=22) #max possible
0.1
[1] 0.10000000000000000055511
```

- **EXAMPLE:** Assume base $b = 10$ and mantissa has 5 digits $p = 5$:
  $1.2345 = +0.12345 \cdot 10^1$
  $4.0000567 = +0.40000 \cdot 10^1$
- Problem remains whatever base (b) is chosen

- **EXERCISE:** Try to convert some numbers

Floating–point system

- Distribution of computer floats

- Dense from -1 to 1
- Density decreases
- same number of points for each exponent:

$$\ldots,\ \cdot 10^{-3},\ \cdot 10^{-2},\ \cdot 10^{-1},\ \cdot 10^{1},\ \cdot 10^{2},\ \cdot 10^{3},\ \ldots$$

- What about integers?
  $5 = +0.50000 \cdot 10^1$

```
options(digits=22)
9007199254740992
9007199254740993
9007199254740994
```

Floating–point system, special "numbers"

- We do not discuss how the exponent is actually coded.
- Usually the maximum allowed number in the exponent is one unit less than possible.
- $\pm$Inf: exponent is $\exp_{\max} + 1$, mantissa is 0
- NaN: exponent is $\exp_{\max} + 1$, mantissa is $\neq 0$
- 0 **WHY?**

Overflow: number larger than can be represented
Underflow: loss of significant digits

```
10^200*10^200 =  Inf
10^400/10^400 =  NaN
10^-200/10^200 =  0
10^-200*10^200 =
0*10^400 =
x<-10^300;while(1){x<-x+1}
```

Arithmetic operations

- Floats are rounded so usual mathematical laws do not hold — floating point arithmetic
- **Examples**

```
1/3+1/3 =  0.6666667
options(digits=22)
1/3+1/3 =  0.66666666666666296592
10^(-200)/(10^(-200)+10^(-200)) =  0.5
10^(-200)/(10^(-200)+20^(-200)) = 1
```

- Software is designed to make operations as correct as possible

- **Do we need to work with such extreme numbers?**

Arithmetic operations

- $X + Y$, $X \cdot Y$ can display overflow, underflow
- $A \neq B$ but $X + A = B + X$
- $A + X = X$ but $A + Y \neq Y$
- $A + X = X$ but $X - X \neq A$
- **COMPARING FLOATS IS TRICKY!**

```
options(digits=22)
x<-sqrt(2)
x*x
[1] 2.0000000000000000444089
(x*x)==2
[1] FALSE
all.equal(x*x,2)
[1] TRUE
```

Summation

Underflow problems can occur with any summation (`x<-x+1`)

```
options(digits=22)
x<-1:1000000; sum(1/x); sum(1/rev(x))
[1] 14.392726722865722521 76
[1] 14.392726722865724298 12
```

- **WHICH ONE IS CORRECT?**
- **WHICH ONE IS MORE ACCURATE?**

Potential solutions

Solution A:

1. Sort the numbers ascending **CAN BE EXPENSIVE**
2. Sum in this order

Solution B:

1. Sum numbers pairwise, from n obtain $n/2$ numbers
   HOW TO CHOOSE PAIRS?
2. Continue until 1 number left

More on summing

Example

- Computing exponent using Taylor series

$$e^x = 1 + x + x^2/2 + x^3/6 + \ldots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

The exponential

```
options(digits=22)
fTaylor<-function(x,N){1+sum(sapply(1:N,
    function(i,x){x^i/prod(1:i)},x=x,simplify=
    TRUE))}
exp(20) #fine
[1] 485165195.4097902774811
fTaylor(20,100)
[1] 485165195.4097902774811
fTaylor(20,100)-exp(20)
[1] 0
exp(-20) #problem
[1] 2.061153622438557869942e-09
fTaylor(-20,100)
[1] -3.853877217352419393137e-10
fTaylor(-20,200)
[1] -3.853877217352419393137e-10
```

Can you explain why?

```
## Example due to Thomas Ericsson in his
    Numerical Analysis course at Chalmers
f1<-function(x){(x-1)^6}
f2<-function(x){1-6*x+15*x^2-20*x^3+15*x^4-6*x
    ^5+x^6}
x<-seq(from=0.995,to=1.005,by=0.0001)
y1<-f1(x)
y2<-f2(x)
plot(x,y1,pch=19,cex=0.5,ylim=c(-5*10^(-15),20
    *10^(-15)),main="Two ways to calculate (x
    -1)^6",xlab="x",ylab="y")
points(x,y2,pch=18,cex=0.8)
```

More on summing

Example

- Computing exponent using Taylor series

$$e^x = 1 + x + x^2/2 + x^3/6 + \ldots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

- **WHY?**

- Varying sign of terms
- **CANCELLATION** adding two numbers of almost equal magnitude but of opposite sign
- Effects of cancellations accumulate
- **SOLUTION:** Different algorithm ...

Matrix Calculations

- Many problems (in Statistics, Numerical methods, e.t.c) can be reduced to solving

$$\mathbf{A}\vec{x} = \vec{b}$$

$\mathbf{A}$ (design) matrix
$\vec{x}$ vector of unknowns
$\vec{b}$ vector of scalars (data)

- Algorithm should be **numerically stable**

(small changes in $\mathbf{A}$ or $\vec{b}$ imply in small changes in $\vec{x}$)

Example: Linear regression models

Minimize

$$RSS(\beta_0, \beta_1, \ldots, \beta_m) = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \ldots - \beta_m x_{im})^2$$

The system of equations

$$\frac{\partial RSS}{\beta_0} = \frac{\partial RSS}{\beta_1} = \ldots \frac{\partial RSS}{\beta_m} = 0$$

can be written as

$$\mathbf{X}^T\mathbf{X}\vec{\beta} = \mathbf{X}\vec{y}$$

where

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & \ldots & x_{1m} \\ 1 & x_{21} & \ldots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \ldots & x_{nm} \end{bmatrix}$$

Solving a linear system

- $\mathbf{A}\vec{x} = \vec{b}$ needs a stable numerical solution
- **Computer arithmetics**

- - Original system: $\mathbf{A}\vec{x} = \vec{b}$
  - Perturbed system: $\mathbf{A}\vec{x'} = \vec{b'}$, $\vec{x'} = \vec{x} + \delta\vec{x}$, $\vec{b'} = \vec{b} + \delta\vec{b}$
- Stable: small perturbation in $\vec{b}$, small perturbations in $\vec{x}$
- $\|\vec{b}\| = \|\mathbf{A}\vec{x}\| \leq \|\mathbf{A}\|\|\vec{x}\|$ implies $\|\vec{x}\|^{-1} \leq \|\mathbf{A}\|\|\vec{b}\|^{-1}$
  $\|\delta\vec{x}\| = \|\mathbf{A}^{-1}(\delta\vec{b})\| \leq \|\mathbf{A}^{-1}\|\|\delta\vec{b}\|$
  together

$$\frac{\|\delta\vec{x}\|}{\|\vec{x}\|} \leq \|\mathbf{A}^{-1}\|\|\mathbf{A}\|\frac{\|\delta\vec{b}\|}{\|\vec{b}\|}$$

Solving a linear system

Condition number of a matrix

$$\kappa(\mathbf{A}) = \|\mathbf{A}^{-1}\|\|\mathbf{A}\|$$

- Large $\kappa(\mathbf{A})$ is a bad sign, but does not imply ill–conditioning
- L_2 norm: $\kappa(\mathbf{A})$ is the ratio of the maximum and minimum eigenvalues of $\mathbf{A}$
- Under L_2 norm

$$\kappa(\mathbf{A}^T\mathbf{A}) \geq \kappa(\mathbf{A})^2 \geq \kappa(\mathbf{A})$$

(regression setting)

Solving a linear system

Dealing with ill–conditioning

- Rescale the variables (columns)
- Use a different algorithm for solving
  e.g. QR, Cholesky, SVD

Cholesky: $\mathbf{A}\vec{x} = \vec{b}$ is equivalent to $\mathbf{L}\mathbf{L}^T\vec{x} = \vec{b}$ **WHY?**

1. Solve $\mathbf{L}\vec{y} = \vec{b}$
2. Solve $\mathbf{L}^T\vec{x} = \vec{y}$

Summary

- Computations can behave "differently" at different numerical ranges.
- Floating point system.
- Computer arithmetics is not the same as "usual" arithmetic.
- Summing series, solving linear systems (inversion?)

Optimization

732A90
Computational Statistics

Krzysztof Bartoszek
(krzysztof.bartoszek@liu.se)

31 I 2017
Department of Computer and Information Science
Linköping University

Plan for today

- Introduction
- Mathematical definition of problem
- 1D optimization
- kD optimization
- `R` code examples

Optimization

Nearly everything is optimization **!**

- Chemistry
- Physics
- Economics, **Industry**
- Engineering

BUT EVEN

- Your mobile price plan
- Course scheduling
- Your lunch choice

STATISTICS

- Fit parameters to data
- Propose optimal decision

Optimization: Example

ANY BIOLOGICAL

ORGANISM

YOU

Optimization: Example

Industry

How to produce a cylindrical (**WHY?**) $0.5L$ beer can so it requires minimum material?

Given a certain product minimize e.g. material usage, production effort while still meeting consumer requirements.

Optimization: Example

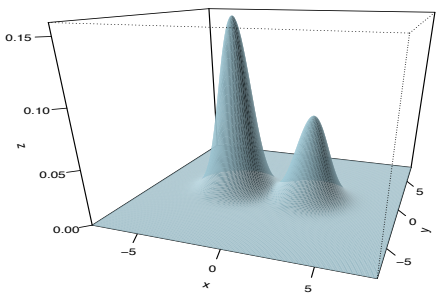
Economics/Logistics

- Travelling Salesman Problem
- Windmills
- Flight schedule (especially "cheap" airlines)

Optimization: Example

Statistics

Maximize likelihood, model fitting



Maximal likelihood

An i.i.d. sample $(X_1, \ldots, X_n)$ is drawn from a probability distribution $P(X|\Theta)$, where Θ is an unknown parameter set.

The joint probability of all the observations is

$$P(X_1, \ldots, X_n|\Theta) = \prod_{i=1}^{n} P(X_i|\Theta).$$

Find Θ that maximizes $P(X_1, \ldots, X_n|\Theta)$.

Mathematical formulation

The goal is to minimize (maximize)

Objective function: $f(\theta)$
(reproduction, chances of survival, quality of life, cost, profit, likelihood, fit to data)

depending on

Parameters or **Unknowns** θ
(reproduction strategy, resource utilization, consumer choices, height & diameter, production, raw material choice, service times, route, flight routes/times ,parameters)

Mathematical formulation

$$\min_{\theta \in \Theta} f(\theta) \quad \text{subject to} \quad \begin{array}{ll} c_i(\theta) = 0, & i \in E \\ c_i(\theta) \geq 0, & i \in I \end{array}$$

QUESTION: What should we do if we are interested in maximization instead of minimization?

QUESTION: What should we do if the constraints are $c_i(x) \leq 0,\ i \in I$?

Constraints examples

- Available environment
- Volume: 0.5l of can
- Production: Factories (F_1, F_2), retail outlets (R_1, R_2, R_3), cost of shipping $i \to j$: c_{ij}, production a_i per week, requirement b_j per week **to optimize:** x_{ij} amount shipped $i \to j$ per week

  $$\begin{array}{ll} \min_{x \in \mathrm{R}^3} \sum_{ij} c_{ij} x_{ij} & \text{minimize shipping costs} \\ \sum_{j=1}^{3} x_{ij} \leq a_i, i = 1, 2 & \text{production capacity} \\ \sum_{i=1}^{3} x_{ij} \geq b_j, j = 1, 2, 3 & \text{demand} \\ \forall_{i,j} x_{ij} \geq 0 & \end{array}$$

  Question: What would happen if we drop demand constraint?
- ML: often no constraints

Exercise

- Split into pairs/triplets/quadruples
- Think of some human anatomy part/organ:
  - What is its function?
  - What could it have been optimized for over the course of time?
  - Is it still under selection?
  - What constraints was and is it under?
- Think of a situation where optimization is needed in your own student/professional/personal/financial situation.
- State the problem in terms of
  - Objective function
  - Parameters
  - Constraints
  - Does it have a trivial solution?
- **10** minutes

Optimization approaches

- Constrained optimization
  - Lagrange multipliers, linear programming
  - E.g. LASSO
  - **Not this lecture!**
- Unconstrained optimization
  - Steepest descent
  - Newton method
  - Quasi–Newton–Methods
  - Conjugate gradients

Why are there different methods?

1D Optimization

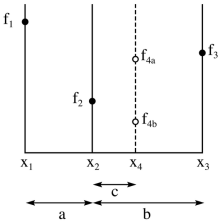
- Function of a single parameter, find minimum
- *What algorithm would you suggest?*
- **Golden–section search**
  local minimum on $[A, B]$ interval (constraint)
- Works by narrowing down the search interval with a constant reduction factor

$$1 - \alpha = \frac{\sqrt{5} - 1}{2} \approx 0.62$$

Question: Does α remind you of something?

Golden section (minimization)

1: $x_1 = A$, $x_3 = B$,
2: **while** $x_1 - x_3 > \epsilon$ **do**
3: $a = \alpha(x_3 - x_1)$
4: $x_2 = x_1 + a$, $x_4 = x_3 - a$
5: **if** $f(x_4) > f(x_2)$ **then**
6: $x_1 = x_1$, $x_3 = x_4$
7: **else**
8: $x_1 = x_2$, $x_3 = x_3$
9: **end if**{We know the value at **3** points!}
10: **end while**



Wikipedia, Golden–section search

f has to be **UNIMODAL**

1D Optimization: Example

```
f <- function (x, a) (x - a)^2
xmin <- optimize(f, c(0, 1), tol = 0.0001, a =
    1/3)
## See where the function is evaluated:
optimize(function(x) x^2*(print(x)-1), lower =
    0, upper = 10)

## "wrong" solution with unlucky interval and
    piecewise constant f():
f  <- function(x) ifelse(x > -1, ifelse(x < 4,
    exp(-1/abs(x - 1)), 10), 10)
fp <- function(x) { print(x); f(x) }
xmin1<-optimize(fp, c(-4, 20))    # doesn't see
    the minimum
xmin2<-optimize(fp, c(-7, 20))    # ok
```

Multi–dimensional optimization

Find

$$\boxed{\min_{\vec{x}\in\mathbb{R}^n} f(\vec{x})}$$

Using (known, or numerically evaluated)

Gradient $\bigtriangledown f(\vec{x}) = \left(\frac{\partial f(\vec{x})}{\partial x_1}, \ldots, \frac{\partial f(\vec{x})}{\partial x_n}\right)^T$

Hessian $\bigtriangledown^2 f(\vec{x}) = \left[\frac{\partial^2 f(\vec{x})}{\partial x_i \partial x_j}\right]_{i,j=1}^{n}$

General strategy

1. Provide a **(good)** starting point $\vec{x}_0$, $\vec{x} = \vec{x}_0$
2. Choose a direction $\vec{p}$ ($\|p\| = 1$) and step size a
3. Move to $\vec{x} := \vec{x} + \alpha\vec{p}$
4. Repeat step 2 until convergence

How to choose the direction?

Taylor's theorem

$$f(\vec{x} + a\vec{p}) = f(\vec{x}) + \boxed{\alpha\vec{p}^T \cdot \bigtriangledown f(\vec{x})} + o(\alpha^2)$$

$\vec{p}$ s.t. $\vec{p}^T \cdot \bigtriangledown f(\vec{x}) < 0$ is a *descent* direction.

Steepest descent is

$$\vec{p} = -(\bigtriangledown f(\vec{x}))/\| \bigtriangledown f(\vec{x})\|$$

How to choose the step size?

- **Expensive way**: find the global minimum in direction $\vec{p}$
- **Trade–off way**: find a decrease which is *sufficient*

BACKTRACKING

1: Choose (large) $\alpha_0 > 0$, $\rho \in (0,1)$, $c \in (0,1)$,
2: $\alpha = \alpha_0$
3: **repeat**
4: $\quad \alpha = \rho\alpha$
5: **until** $f(\vec{x} + \alpha\vec{p}) \leq f(\vec{x}) + c\alpha\vec{p}^T \bigtriangledown f(\vec{x})$

Newton's method

- Newton–Raphson method
- Hessian ignored in steepest descent
- If f is quadratic

$$f(\vec{p}) = \frac{1}{2}\vec{p}^T \mathbf{A}\vec{p} + \vec{b}^T\vec{p} + c,$$

then minimum

$$\vec{p}^* = \mathbf{A}^{-1}\vec{b}.$$

- Taylor expansion of f

$$f(\vec{x} + a\vec{p}) = f(\vec{x}) + \alpha\vec{p}^T \cdot \bigtriangledown f(\vec{x}) + \frac{\alpha^2}{2}\vec{p}^T \bigtriangledown^2 f(\vec{x})\vec{p} + o(\alpha^3)$$

- $x := x + \alpha\vec{p}$ where

$$\boxed{\vec{p} = -\left(\bigtriangledown^2 f(\vec{x})\right)^{-1} \bigtriangledown f(\vec{x})}$$

Quasi–Newton methods

- k iteration number
- Compute an approximation to the Hessian, $\mathbf{B}$, that will allow for efficient choice of $\vec{p}$.
- **SECANT CONDITION:** (quasi–Newton condition)

$$\mathbf{B}_{k+1}\left(\vec{x}_{k+1} - \vec{x}_k\right) = \bigtriangledown f(\vec{x}_{k+1}) - \bigtriangledown f(\vec{x}_k)$$

BFGS Algorithm

1: Choose $\mathbf{B}_0 > 0$, $\vec{x}_0$, $k = 0$
2: **repeat**
3: $\vec{p}_k$ is solution of $\mathbf{B}_k\vec{p}_k = \bigtriangledown f(\vec{x}_k)$
4: find suitable α_k
5: $\vec{x}_{k+1} = \vec{x}_k + \alpha_k\vec{p}_k$
6: calculate $\mathbf{B}_{k+1}$ {next slide}
7: $k = k + 1$
8: **until** convergence of $\vec{x}_k$ at minimum

Newton's method

- $\left(\bigtriangledown^2 f(\vec{x})\right)^{-1}$ is expensive to compute, there are quicker approaches, e.g. Cholesky decomposition
- Hessian should be **positive definite** for $\vec{p}$ to be a descent direction (if not see book)
- Memory expensive — need to store $O(n^2)$ elements

BUT

- Method converges quickly esp. near optimum

How to compute $\mathbf{B}_{k+1}$?

- We want $\mathbf{B}_{k+1}$ and $\mathbf{B}_k$ to be close to each other
- $$\min_{\mathbf{B}} \|\mathbf{B} - \mathbf{B}_k\|$$
  $$s.t.\ \mathbf{B} = \mathbf{B}^T, \text{ secant condition}$$
- $\vec{y}_k = \bigtriangledown f(\vec{x}_{k+1}) - \bigtriangledown f(\vec{x}_k)$, $\vec{s}_k = \vec{x}_{k+1} - \vec{x}_k$
- $$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k\vec{y}_k\vec{y}_k^T\mathbf{B}_k}{\vec{y}_k^T\mathbf{B}_k\vec{y}_k} + \frac{\vec{s}_k\vec{s}_k^T}{\vec{y}_k\vec{s}_k^T}$$
- Closed form Sherman–Morrison formula for $\mathbf{B}_{k+1}^{-1}$
- We have to store $\mathbf{B}_k^{-1}$

BFGS

- BGFS: Broyden–Fletcher–Goldfarb–Shanno
- More iterations than Newton's method (uses approximation)
- Each iteration quicker, no numeric inversion
- Good for large scale problems
- Choice of $\mathbf{B}_0$?

Conjugate Gradient method

- $\vec{p}_0 = \vec{r}_0$
- $\vec{p}_{k+1} = -\vec{r}_k + \beta_{k+1}\vec{p}_k$
- Conjugate condition has to be satisfied so

$$\beta_{k+1} = \frac{\vec{r}_k^T \mathbf{A}\vec{p}_{k-1}}{\vec{p}_k^T \mathbf{A}\vec{p}_k}$$

Exercise: check this

- Convergence in $\dim(\mathbf{A})$ steps (or unless cutoff for $\vec{r}_k$)

Conjugate Gradient method—quadratic case

Minimize

$$f(\vec{x}) = \frac{1}{2}\vec{x}^T \mathbf{A}\vec{x} - \vec{b}^T\vec{x}$$

for $\mathbf{A}$ symmetric positive definite.
Gradient:

$$\bigtriangledown f(\vec{x}) = \mathbf{A}\vec{x} - \vec{b} = r(\vec{x})$$

Two vectors $\vec{p}$ and $\vec{q}$ are **conjugate** with respect to $\mathbf{A}$ if

$$\vec{p}^T \mathbf{A}\vec{q} = 0.$$

IDEA: $\vec{p}$ and $\vec{q}$ are orthogonal w.r.t. to an inner product associated with $\mathbf{A}$. Use this to find a basis that will allow for easy finding of $\vec{x}$.

Nonlinear CG method

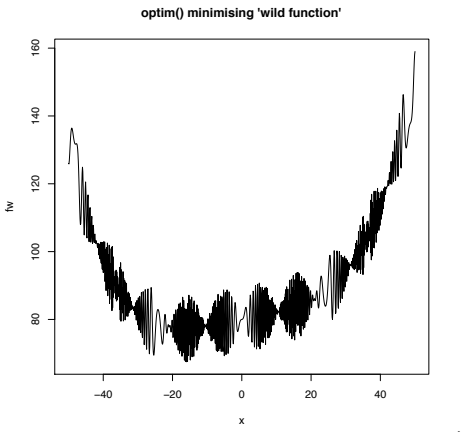
- If $f(\cdot)$ general, use $\bigtriangledown f(\cdot)$ instead of $r(\cdot)$

1: Choose $\vec{x}_0$, $\vec{p}_0 = -\bigtriangledown f(\vec{x}_0)$, $k = 0$
2: **while** $\bigtriangledown f(x_k) \neq \vec{0}$ **do**
3: find suitable α_k
4: $\vec{x}_{k+1} = \vec{x}_k + \alpha_k\vec{p}_k$ {and now update step}
5: $\beta_{k+1} = \left(\bigtriangledown^T f(\vec{x}_{k+1}) \bigtriangledown f(\vec{x}_{k+1})\right) / \left(\bigtriangledown^T f(\vec{x}_k) \bigtriangledown f(\vec{x}_k)\right)$ {Fletcher–Reeves update, other possible}
6: $\vec{p}_{k+1} = -\bigtriangledown f(\vec{x}_{k+1}) + \beta_{k+1}\vec{p}_k$
7: $k = k + 1$
8: **end while**

Nonlinear CG method

- Local minimum convergence
- But this is true of all methods that cannot "jump out" of descent path
- Faster than steepest descent
- Slower than Newton and Quasi–Newton but significantly less memory

kD Optimization: Example



kD Optimization: Example

```
## "wild" function , global minimum at about
    -15.81515
fw <- function (x)
          10*sin(0.3*x)*sin(1.3*x^2) + 0.00001*
              x^4 + 0.2*x+80
plot(fw, -50, 50, n = 1000, main = "optim() 
    minimising 'wild function'")
# method = c("Nelder-Mead", "BFGS", "CG", "L-
    BFGS-B", "SANN","Brent")
res <- optim(50, fw, method = "SANN",lower = -
    Inf, upper = Inf, control = list(maxit =
    20000, temp = 20, parscale = 20), hessian =
     FALSE)
# res$par= -15.8144, res$value=67.47249, res$$
    counts["function"]=20000,res$$counts["
    gradient"]=NA, res$convergence=0 (did!),
    res$message=NULL
```

Summary

- Optimization is everywhere
- Numerical methods for finding minimum
- 1D: Golden section (unimodal), `optimize()`
- kD: choose step size and direction (gradient), `optim()`

Random Number Generation

732A90
Computational Statistics

Krzysztof Bartoszek
(krzysztof.bartoszek@liu.se)

7 II 2017
Department of Computer and Information Science
Linköping University

Pseudorandom numbers

- **A computer is a deterministic machine**
- *Congruential generators*
- Functions of time
- Be careful with respect to application

First step: Generating Unif$[0,1]$

Linear congruential generator

Define a sequence of integers according to

$$x_{k+1} = (a \cdot x_k + c) \mod m, \quad k \geq 0$$

x_0 is **seed**, e.g. based on time

mod m: remainder after division by m

- $x_k \in \{0, \ldots, m-1\}$ and integer
- $x_k/m \sim$ Unif$[0,1]$
- $a, c \in [0, m)$ need to be carefully selected

First step: Generating Unif$[0,1]$

Generated numbers will get into a loop with a certain **period**

$$x_{k+1} = (a \cdot x_k + c) \mod m, \quad k \geq 0$$

$x_0 = a = c = 7$, $m = 10$

1. $x_1 = (7 \cdot 7 + 7) \mod 10 = 56 \mod 10 = 6$
2. $x_1 = (7 \cdot 6 + 7) \mod 10 = 49 \mod 10 = 9$
3. $x_1 = (7 \cdot 9 + 7) \mod 10 = 70 \mod 10 = 0$
4. $x_1 = (7 \cdot 0 + 7) \mod 10 = 7 \mod 10 = 7$
5. $x_1 = (7 \cdot 7 + 7) \mod 10 = 56 \mod 10 = 6$
6. ...

First step: Generating Unif$[0,1]$

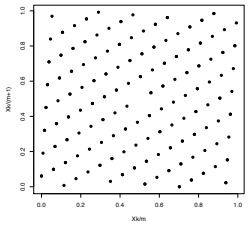
```
fthreebits<-function(k,s,L,N){
    X0<-4*s+1;a<-8*k+5;m<-2^L;X<-X0
    for (i in 1:N){
        print(c(X,rev(intToBits(X)[1:5])))
        X<-(a*X)%%m ##c=0
    }
}
```

```
> source("CongGen.R");fthreebits(k=2,s=3,L=8,N=10)
[1] 13  0  1  1  0  1
[1] 17  1  0  0  0  1
[1] 101   0   0   1   0   1
[1] 73  0  1  0  0  1
[1] 253   1   1   1   0   1
[1] 193   0   0   0   0   1
[1] 213   1   0   1   0   1
[1] 121   1   1   0   0   1
[1] 237   0   1   1   0   1
[1] 113   1   0   0   0   1
```

Last three bits change between 001 and 101
Discard less significant bits

First step: Generating Unif$[0,1]$

```
fCongGenGrid<-function(a,m,c,X0,N){
    X<-X0;vU<-rep(X0,N)
    for (i in 1:N){vU[i]<-X;X<-(a*X+c)%%m}
    vU<-vU/m;plot(vU[1:(N-1)],vU[2:N],pch=19,
        cex=0.8,main="",xlab="Xk/m",ylab="Xk/(m
        +1)")
};fCongGenGrid(a=17,m=131,c=8,X0=4,N=200)
```



First step: Generating Unif$[0,1]$

- Period has to be smaller than m
- a, c, m (**large**) have to be chosen carefully
  1. c and m have to be relatively prime (no common divisors bar 1)
  2. $a = 1 \mod p$ for every prime divisor p of m
  3. $a = 1 \mod 4$ if 4 divides m
  4. Then full period m reached
- Seed defines the random sequence — same seed, same sequence
  Be careful when re–opening an R workspace
- Other methods (not in this course)

Second step: Generating Unif$[a,b]$

- $U \sim \text{Unif}[0,1]$ can be transformed into $X \sim \text{Unif}[a,b]$ as

$$X = a + U \cdot (b - a)$$

- U can also be transformed into **discrete** uniform distribution on integers $\in \{1, \ldots, n\}$ as ($[\cdot]$, integer part)

$$X = [nU] + 1$$

Questions

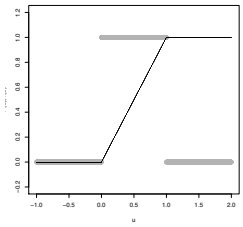
1. Why $+1$?
2. How can U be transformed into Y, where Y is discrete uniform on integers $(50, 55, 60)$?

Second step: Generating nonuniform random numbers

- $U \sim \text{Unif}(0,1)$
- Let F_U be the *cumulative distribution function* (CDF) of U

$$F_U(u) = P(U \leq u) = \begin{cases} 0 & u \leq 0 \\ u & 0 < u \leq 1 \\ 1 & 1 < u \end{cases}$$

- The *probability distribution function* (PDF) of U



$$f_U(u) = \begin{cases} 1 & 0 < u < 1 \\ 0 & u \notin (0,1) \end{cases}$$

Inverse CDF method

If we can generate $U \sim \text{Unif}(0,1)$, then

we can generate $X \sim F_X$ as

$$X = F_X^{-1}(U)$$

Provided we can calculate $F_X^{-1} \ldots$

Inverse CDF method

Let X be a random variable with CDF $X \sim F_X$ (F_X strictly increasing)

Consider $Y = F_X^{-1}(U)$, where $U \sim \text{Unif}(0,1)$

$$\begin{aligned} F_Y(y) &= P(Y \leq y) = P(F_X^{-1}(U) \leq y) \\ &= P(F_X(F_X^{-1}(U)) \leq F_X(y)) \\ &= P(U \leq F_X(y)) = F_U(F_X(y)) = F_X(y) \end{aligned}$$

Y has same probability distribution as X

Inverse CDF method: Example

Let $X \sim \exp(\lambda)$, i.e. with pdf

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

implying (**SHOW THIS**)

$$F_X(x) = \int_{-\infty}^{x} f_X(s)\mathrm{d}s = 1 - e^{-\lambda x}, \quad x \geq 0$$

QUESTIONS:
What is $F_X(x)$ for $x < 0$?
What is $\mathrm{E}[X]$?

Inverse CDF method: Example

Find F_X^{-1}

$$
\begin{aligned}
y &= 1 - e^{-\lambda x} \\
e^{-\lambda x} &= 1 - y \\
x &= -\tfrac{1}{\lambda}\ln(1-y) \\
F_X^{-1}(y) &= -\tfrac{1}{\lambda}\ln(1-y)
\end{aligned}
$$

Hence, if $U \sim \mathrm{U}(0,1)$, then

$$-\frac{1}{\lambda}\ln(1-U) = X \sim \exp(\lambda)$$

Inverse CDF method

1. When F_X^{-1} can be derived: **EASY**
2. When **NOT**: numerical solution

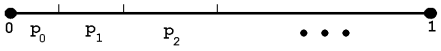
   time–consuming

   numerical errors ?

Situation 2 is common ... e.g. $\mathcal{N}(0,1)$

Generating discrete RVs

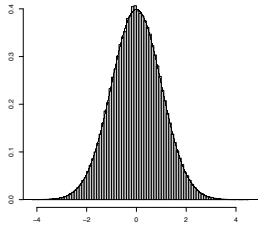
1. Define distribution $P(X = x_i) = p_i$
2. Generate $U \sim \mathrm{Unif}(0,1)$
3. If $U \leq p_0$, set $X = x_0$
4. Else if $U \leq p_0 + p_1$, set $X = x_1$
5. ...



Generating $\mathcal{N}(0,1)$

Assume

- $\theta \in \mathrm{Unif}(0, 2\pi)$
- $D \in \mathrm{Unif}(0,1)$



1: Generate θ, D
2: Generate X_1 and X_2 as

$$
\begin{aligned}
X_1 &= \sqrt{-2\ln D}\cos\theta \\
X_2 &= \sqrt{-2\ln D}\sin\theta
\end{aligned}
$$

X_1 and X_2 are independent and normally distributed

But finding such transformations is not easy

Acceptance/rejection methods

- IDEA: generate $Y \sim f_Y$ similar to some known PDF f_X
- IDEA: f_Y is easy to generate from
- REQUIREMENT: there exists a constant c

$$\forall_x c f_Y(x) \geq f_X(x)$$

- f_Y: majorizing density, proposal density
- f_X: target density
- c: majorizing constant

Acceptance/rejection methods

1: **while** X not generated **do**
2: Generate $Y \sim f_Y$
3: Generate $U \sim \text{Unif}(0,1)$
4: **if** $U \leq f_X(Y)/(cf_Y(Y))$ **then**
5: $X = Y$
6: Set X is generated
7: **end if**
8: **end while**

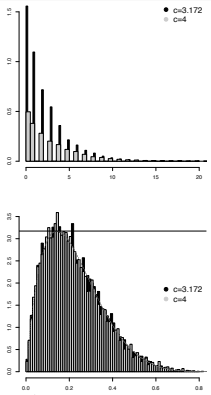
- $X \sim f_X$ **CHECK THIS**
- Larger c: larger rejection rates—c as small as possible
- Can work in higher dimensions—**but** high rejection rate

Acceptance/rejection methods: Example

Generate beta(2,7)

```
y<-dbeta(seq(0,2,0.0001),2,7)
c<-max(y);c
[1] 3.172554
```

1: **while** X not generated **do**
2: Generate $Y \sim \text{Unif}(0,1)$
3: Generate $U \sim \text{Unif}(0,1)$
4: **if** $U \leq dbeta(Y,2,7)/(c \cdot 1)$ **then**
5: $X = Y$
6: Set X is generated
7: **end if**
8: **end while**



QUESTION:
Compare acceptance and rejection regions (of Y) for different c.

Acceptance/rejection methods:

- Acceptance/rejection is difficult to apply
- Difficult to find majorizing density
  - can always take $\sup(f_X) \cdot \text{Unif}(0,1)$
  - but what is the problem?

Generating multivariate normal

Generate $\mathcal{N}(\vec{\mu}, \mathbf{\Sigma}) \in \mathbb{R}^n$

1: Generate n i.i.d. $\mathcal{N}(0,1)$ r.vs. $\vec{X} = (X_1, \ldots, X_n)$
{We know how to do this, see slide 16}
2: Compute Cholesky decomposition (a.k.a. matrix square root) of $\mathbf{\Sigma}$, i.e. find $\mathbf{A}$, lower triangular s.t. $\mathbf{A}\mathbf{A}^T = \mathbf{\Sigma}$,
{in `R`: `chol()` }
3: $\vec{Y} = \mu + \mathbf{A}\vec{X}$

QUESTION:
what is the expectation and variance–covariance of $\vec{Y}$?

Random numbers in R

1. `d`distribution name`()`: density of distribution
2. `p`distribution name`()`: CDF of distribution
3. `q`distribution name`()`: quantiles of distribution
4. `r`distribution name`()`: simulate from distribution

Summary

- Computers generate pseudo–random numbers
- We draw from pseudo–uniform and transform to desired distribution
- Analytical methods for transforming exist but are distribution specific

Monte Carlo Methods

732A90
Computational Statistics

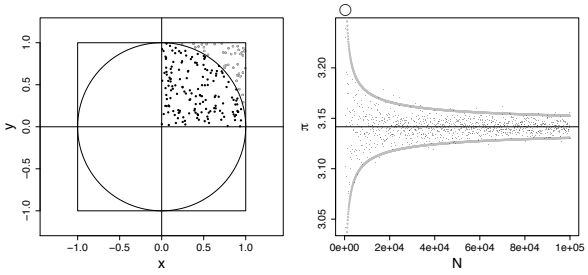
Krzysztof Bartoszek
(krzysztof.bartoszek@liu.se)

8 II 2017
Department of Computer and Information Science
Linköping University

What is the area of the unit circle?

```
f.circArea<-function(N){
    m.xy<-cbind(runif(N),runif(N))
    4*sum(apply(m.xy,1,function(xy){xy[1]^2+xy
        [2]^2<1}))/N
}
```

$$3.141 \approx \pi = \int 1 \mathrm{d}x$$



Monte Carlo methods: outline

- **Monte Carlo methods** are a class of computational algorithms that use repeated random sampling to compute their results.
- Monte Carlo methods for random number generation
  - Metropolis–Hastings algorithm
  - Gibbs sampler
- Monte Carlo methods for statistical inference
  - Estimate integrals (we already did!)
  - Variance estimation
  - Variance reduction: importance sampling, control variates

Markov Chain Monte Carlo

Previous lecture: Generate

- univariate distributions (inverse CDF, acceptance/rejection)
- multivariate normal

but general multivariate distribution?

MCMC

Bayesian inference: Recap

A dataset D is obtained by sampling from a distribution $f(\cdot|\theta)$. How to estimate θ?

- *Frequentists*: θ is an unknown but fixed parameter, compose likelihood $\mathcal{L}(D|\theta)$ and find θ that maximizes it.
- *Bayesians*: θ is a random variable with **prior** probability law $p(\theta)$ before observing D
- After observing D, Bayes' theorem gives

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\int p(D|\theta)p(\theta)\mathrm{d}\theta}$$

Bayesian inference: Recap

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\int p(D|\theta)p(\theta)\mathrm{d}\theta}$$

We know: $p(D|\theta)$ (the model), $p(\theta)$ (the prior)
We need: simulate from $p(\theta|D)$ (the posterior)

1. General (multivariate) type distribution
2. Integral can be impossible to compute

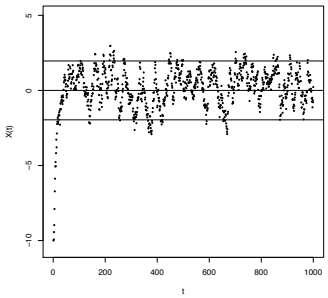
1. MCMC solves this
2. Not needed (given D it is constant)

Markov Chains: Recap

- A Markov chain is a sequence X_0, X_1, ... of random variables such that the distribution of the next value depends only on the current one (and parameters).
- $P(X_{t+1}|X_t)$ is called a **transition kernel**. Assume it does not depend on t (**time homogeneous**).
- A Markov chain is **stationary**, with stationary distribution Φ, if $\forall_k\ X_k \sim \Phi$
- One shows (not trivial in general) that under *certain* conditions a Markov chain will converge to the stationary distribution in the limit.

Markov Chains: Example

$$X(t+1) = e^{-1}X(t) + \epsilon\ , \epsilon \sim \mathcal{N}(0, \tfrac{5}{2} \cdot (1 - e^{-2}))$$



Discard first $K - 1$ samples: **burn–in period**

MCMC: Example

Linear regression with residual normally/student/etc. distributed

$$Y = \beta X + \epsilon$$

How to find credible interval for β if we know $\text{Var}[\epsilon] = \sigma^2$?

1. $$P(Y|X,\beta) = \prod_{i=1}^{N} f(Y_i|\text{mean} = \beta X_i, \text{var} = \sigma^2)$$
2. Obtain $P(\beta|Y,X)$ by drawing from $P(Y|X,\beta)P(\beta)$ **in a clever way**.
3. The prior ?
4. Use the MCMC sample to obtain quantiles.

Normal residual: analytical solution

Metropolis–Hastings Sampler

$$\alpha(X_t, Y) = \min\left\{1, \frac{\pi(Y)q(X_t|Y)}{\pi(X_t)q(Y|X_t)}\right\}$$

Initialize chain to X_0, $t = 0$
while $t < t_{\max}$ **do**
  Generate a candidate point $Y \sim q(\cdot|X_t)$
  Generate $U \sim Unif(0,1)$
  if $U < \alpha(X_t, Y)$ **then**
    $X_{t+1} = Y$
  else
    $X_{t+1} = X_t$
  end if
  $t = t + 1$
end while

Metropolis–Hastings algorithm

We have

- A PDF $\pi(x)$ that we want to sample from.
- A **proposal distribution** $q(\cdot|X_t)$ that has a **regular** form w.r.t. to $\pi(\cdot)$
  E.g. $q(\cdot|X_t)$ is normal with mean X_t and given variance
- *Regular* form: suffices that the proposal has the same support as π.

Metropolis–Hastings Sampler: Properties

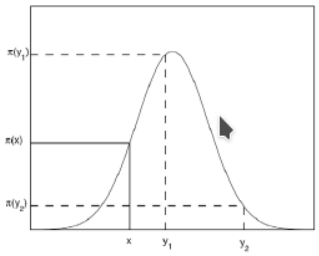
- Informally: "The chain $(X_t)_{t=0}^{\infty}$ will converge to $\pi(\cdot)$."
- The chain might not move sometimes.
- The values of the chain are dependent.
- If $q(X_t|Y) = q(Y|X_t)$ (i.e. symmetric proposal) we get **Random–walk Monte Carlo**:

$$\alpha(X_t, Y) = \min\left\{1, \frac{\pi(Y)}{\pi(X_t)}\right\}$$

Choice of proposal distribution

- In Random–Walk Monte Carlo

  If $\pi(Y) \geq \pi(X)$, the chain moves to the next point, otherwise only with some probability.

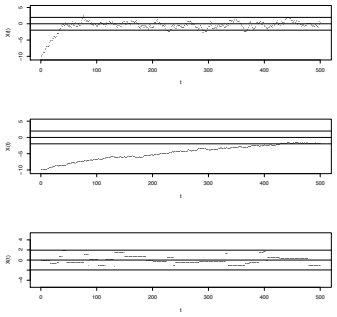


Choice of proposal dist.: **target**: $\pi(\cdot) = \mathcal{N}(0,1)$

```
f.MCMC.MH<-function(nstep,X0,props){
    vN<-1:nstep
    vX<-rep(X0,nstep);
    for (i in 2:nstep){
        X<-vX[i-1]
        Y<-rnorm(1,mean=X,sd=props)
        u<-runif(1)
        a<-min(c(1,(dnorm(Y)*dnorm(X,mean=Y,sd=
            props))/(dnorm(X)*dnorm(Y,mean=X,sd=
            props))))
        if (u <=a){vX[i]<-Y}else{vX[i]<-X}
    }
    plot(vN,vX,pch=19,cex=0.3,col="black",xlab="t",
        ylab="X(t)",main="",ylim=c(min(X0-0.5,-5),
        max(5,X0+0.5)))
    abline(h=0)
    abline(h=1.96)
    abline(h=-1.96)
}
```

Choice of proposal distribution

q normal with sd: `props=` 0.5, 0.1 and 20



Gibbs sampler: alternative to Metropolis–Hastings

We want to generate from a distribution on $\mathbb{R}^d$.

1: Initialize chain to $X_0 = (X_{0,1}, \ldots, X_{0,d})$, $t = 0$
2: **while** $t < t_{\max}$ **do**
3: **for** $i = 1, \ldots, d$ **do**
4: Generate

$$X_{t+1,i} \sim f(\cdot | X_{t+1,1}, \ldots, \mathbf{X_{t+1,i-1}}, \mathbf{X_{t,i+1}}, \ldots, X_{t,d})$$

5: **end for**
6: $t = t + 1$
7: **end while**

Gibbs sampler

- At each iteration inside the `for` loop univariate random numbers are generated.
- Only one element is updated.
- **WE NEED TO KNOW THE CONDITIONAL MARGINAL DISTRIBUTIONS**.
- Convergence may be slow.
- Can be useful in high dimensions (i.e. proposal density may be difficult to find in another way).

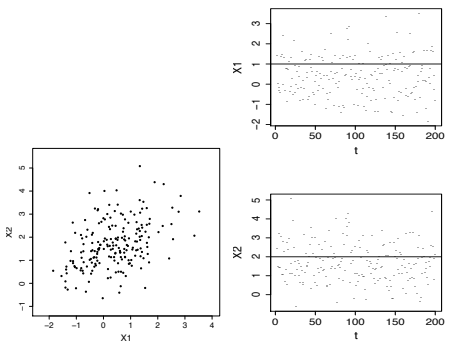
Gibbs sampler: target: d–dim $\mathcal{N}(\mu, \Sigma)$

```
f.MCMC.Gibbs<-function(nstep,X0,vmean,mVar){
    d<-length(vmean); mX<-matrix(0,nrow=nstep,ncol=
        d); mX[1,]<-X0
    for (i in 2:nstep){
        X<-mX[i-1,];Y<-rep(0,d)
        Y[1]<-rnorm(1,mean=vmean[1]+(mVar[1,-1]%*%
            solve(mVar[-1,-1]))%*%(X[2:d]-vmean
            [-1]),sd=sqrt(mVar[1,1]-mVar[1,-1]%*%
            solve(mVar[-1,-1])%*%mVar[-1,1]))
        for (j in 2:(d-1)){Y[j]<-rnorm(1,mean=vmean
            [j]+(mVar[j,-j]%*%solve(mVar[-j,-j]))%*
            %(c(Y[1:(j-1)],X[(j+1):d])-vmean[-j]),
            sd=sqrt(mVar[j,j]-mVar[j,-j]%*%solve(
            mVar[-j,-j])%*%mVar[-j,j]))}
        Y[d]<-rnorm(1,mean=vmean[d]+(mVar[d,-d]%*%
            solve(mVar[-d,-d]))%*%(Y[1:(d-1)]-vmean
            [-d]),sd=sqrt(mVar[d,d]-mVar[d,-d]%*%
            solve(mVar[-d,-d])%*%mVar[-d,d]))
        mX[i,]<-Y
    };mX}
```

Gibbs sampler: Example (code: see R scripts)

Generate from

$$\mathcal{N}([1\ 2]^T, \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix})$$



Convergence monitoring

- When should we stop the chain? When are we (nearly) at the stationary distribution?
- Typically such a sample is generated to make further inference.

Convergence monitoring: Gelman–Rubin method

We want to estimate $v(\theta)$.

1. Generate k sequences of length n with different starting points.
2. Compute between– and within– sequence variances:

$$B = \frac{n}{k-1}\sum_{i=1}^{k}(\overline{v}_{i\cdot} - \overline{v}_{\cdot\cdot})^2 \quad W = \sum_{i=1}^{k}\frac{s_i^2}{k} \quad s_i^2 = \sum_{j=1}^{n}\frac{(\overline{v}_{ij} - \overline{v}_{i\cdot})^2}{n-1}$$

3. Overall variance estimate: $\hat{\mathrm{Var}}[v] = \frac{n-1}{n}W + \frac{1}{n}B$
4. Gelman–Rubin factor:

$$\sqrt{R} = \sqrt{\frac{\hat{\mathrm{Var}}[v]}{W}}$$

5. Values much larger than 1 indicate lack of convergence
6. See `?coda::gelman.diag`

Gibbs sampler

```
library(coda)
f1<-mcmc.list();f2<-mcmc.list();n<-100;k<-20
X1<-matrix(rnorm(n*k),ncol=k,nrow=n)
X2<-X1+(apply(X1,2,cumsum)*(matrix(rep(1:n,k),ncol=
    k)^2))
for (i in 1:k){f1[[i]]<-as.mcmc(X1[,i]);f2[[i]]<-as
    .mcmc(X2[,i])}
print(gelman.diag(f1))
# Potential scale reduction factors:
#      Point est. Upper C.I.
#[1,]       0.999       1.01

print(gelman.diag(f2))
# Potential scale reduction factors:
#      Point est. Upper C.I.
#[1,]       1.82       2.38
```

MC for inference

- Estimation of a definite integral

$$\theta = \int_D f(x)\mathrm{d}x \quad \left(\text{recall } \pi = \int_{\bigcirc} 1\mathrm{d}x\right)$$

- Decompose into:

$$f(x) = g(x)p(x) \quad \text{where} \quad \int_D p(x)\mathrm{d}x = 1$$

- Then, if $X \sim p(\cdot)$

$$\theta = \mathrm{E}[g(X)] = \int_D g(x)p(x)\mathrm{d}x$$

-

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} g(x_i), \quad \forall_i x_i \sim p(\cdot)$$

MC for inference

- Decomposition is not unique, some will be better (lower variance) others worse. $p(x) \propto |f(x)|$: minimal
- Can we easily generate from $p(\cdot)$?
- Bayesian inference: use MCMC samples from $p(\theta|D)$ to obtain a point estimator

$$\theta^* = \int \theta p(\theta|D) \approx \frac{1}{n}\sum_{i=1}^{m}\theta_i$$

- $\hat{\theta}$ depends on n and $g(X)$, how variable will it be?

$$\widehat{\mathrm{Var}\left[\hat{\theta}\right]} = \frac{1}{n(n-1)}\sum_{i=1}^{n}\left(g(x_i) - \overline{g(x)}\right)^2$$

- MCMC: estimator biases as chain correlated, use longer chain and batch mean instead of x_i.

Summary

1. Generating data from a general multivariate distribution
2. Markov Chain Monte Carlo: Metropolis–Hastings algorithm, Gibbs sampling
3. Convergence: Gelman–Rubin method
4. Estimation of integral

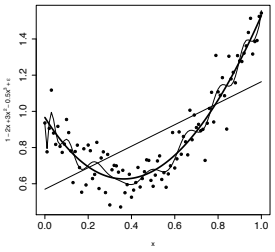
Model Selection and Hypothesis Testing

732A90
Computational Statistics

Krzysztof Bartoszek
(krzysztof.bartoszek@liu.se)

28 II 2017
Department of Computer and Information Science
Linköping University

Model selection



Tools for model selection

- **Comparing different models**
- Information criteria (not this course)
- Cross–validation
- Hypothesis testing

- **Uncertainty estimation**
- Confidence intervals

Hypothesis testing: **Recap**

1. Assume a probabilistic model
   State a null hypothesis (H_0 e.g. no difference) and alternative (H_1 difference)
2. Observe data X
3. Calculate a test statistic e.g. $T(X) = (\overline{X})/(\widehat{\mathrm{sd}(X)})$
   (different statistics will have different **efficiency** (power, ability to distinguish between hypotheses) associated with them)
4. Under H_0 $T(X)$ has "known" distribution
5. Decision: Is the value of $T(X)$ *surprising* (in the **critical region**)? If so reject H_0 in favour of H_1.

Hypothesis testing: Example

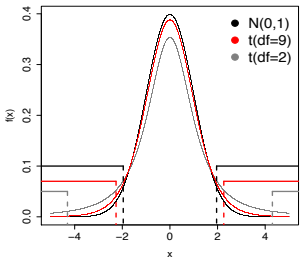
```
x<-rnorm(10,mean=4,sd=1)
```

Hypotheses:

$H_0 : \mu = 4, X \sim \mathcal{N}(\mu, \sigma^2)$
$H_1 : \mu \neq 4, X \sim \mathcal{N}(\mu, \sigma^2)$

Test statistic

$T(x) = \frac{\overline{x}-\mu}{s/\sqrt{n}} \sim t(n-1)$

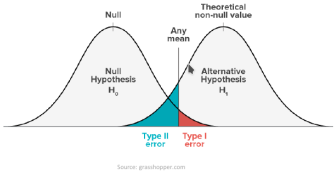



```
tx<-(mean(x)-4)/(sqrt(var(x)/length(x)))
t0<-qt(0.975,df=length(x)-1)
(tx>t0)||(tx<(-t0)) ## reject if TRUE
```

Hypothesis testing: Power

How does one compares different statistics?
POWER
Power = 1 − Type II error

Ability to correctly identify *surprise*,
i.e. indicate H_1.



How to compute power?

- Analytically (?)
- Generate data samples that satisfy H_1
  Compute percent of correct rejections

Monte Carlo Hypothesis testing

We may use "any" test statistic.
We do **not** need to know its distribution.

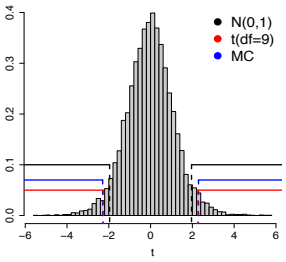
$H_0 : \mu = 4, X \sim \mathcal{N}(\mu, \sigma^2)$
$H_1 : \mu \neq 4, X \sim \mathcal{N}(\mu, \sigma^2)$

Test statistic

$T(x) = \frac{\overline{x}-\mu}{s/\sqrt{n}} \sim t(n-1)$

1: **for** $i = 1$ to B **do**
2: Generate $Y_1, \ldots, Y_n$ i.i.d. from H_0, i.e. $\mathcal{N}(\mathbf{4}, \sigma^2)$
3: Compute t_i from $Y_1, \ldots, Y_n$
4: **end for**
5: Use $t_1, \ldots, t_B$ to construct a histogram
6: Use the histogram as the distribution of $T(x)$ under H_0

Monte Carlo Hypothesis testing




```
x<-rnorm(10,4,1)
s<-var(x)
B<-10000
n<-length(x)
tsamp<-rep(NA,B)
for (i in 1:B){
    Y<-rnorm(n,4,s)
    tsamp[i]<-(mean(Y)-4)/(sd(Y)/sqrt(length(Y)))
}
hist(tsamp,breaks=50,col=gray(0.8),main="",xlab="t"
    ,ylab="",freq=FALSE,cex.axis=1.5,cex.lab=1.5)
```

Permutation tests

- A. k. a. randomization tests
- One solution if we do not know the distribution under H_0
- Computationally expensive
- Any sample size
- Two sample problem:
  - Population 1 distributed as F
  - Population 2 distributed as G
  - $H_0 : F = G$
    $H_1 : F \neq G$

Permutation tests: mouse data

- Control group
- Treatment group
- Group variable g
- Values variable v

```
> t(mouse)
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]  [,10] [,11] [,12]
Group "y"   "z"   "z"   "y"   "y"   "z"   "y"   "y"   "y"   "y"   "z"   "z"
Value " 10" " 16" " 23" " 27" " 31" " 38" " 40" " 46" " 50" " 52" " 94" " 99"
      [,13] [,14] [,15] [,16]
Group "y"   "z"   "y"   "z"
Value "104" "141" "146" "197"
```

Do the values differ significantly between control and treatment groups?

Permutation test: scheme

1: $T(X)$ value of statistic from observed data

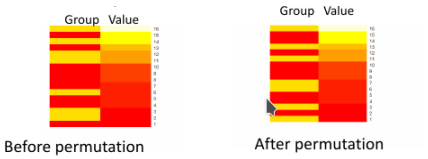
2: Create permutations $g_1^*, \ldots, g_B^*$ of group variable
{If the number of permutations is too large, sample B randomly **without** replacement. E.g. generate random permutations and keep only unique ones.}

3: Evaluate test statistic on each permutation

4: Estimate p–value: $\hat{p} = \#\{T(X_{g_b^*}) \geq T(X)\}/B$
5: If test is two–sided: $\hat{p} = \#\{|T(X_{g_b^*})| \geq |T(X)|\}/B$

Permutation tests

IDEA: If $F = G$ then **group label does not matter**
We may permute labels and still have a sample from F (or G)

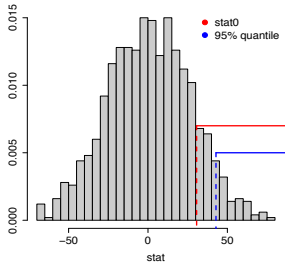


Test statistic:

$$T(X) = \text{mean}(\text{values}|\text{group} = z) - \text{mean}(\text{values}|\text{group} = y)$$

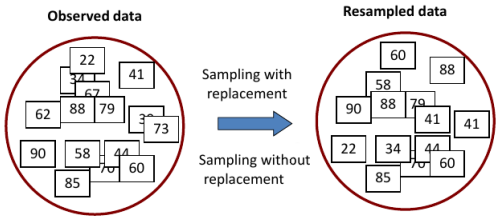
Permutation tests

Do we reject the null?




```
B=1000
stat=numeric(B)
n=dim(mouse)[1]
for(b in 1:B){
  Gb=sample(mouse$Group, n)
  stat[b]=mean(mouse$Value[Gb=='z'])-mean(mouse$
      Value[Gb=='y'])
}
stat0=mean(mouse$Value[mouse$Group=='z'])-mean(
    mouse$Value[mouse$Group=='y'])
print(c(stat0,mean(stat>stat0)))
## [1] 30.63492  0.12700
```

Resampling methods



Jackknife and bootstrap

Theory **different**, coding **similar**
Data (**i.i.d.**) $X \sim F(\cdot, w)$

1: Observed data: $D = (X_1, \ldots, X_n)$, estimator $\widehat{w} = T(D)$
2: **for** $i = 1, \ldots, B$ { Jackknife $B \leq n$} **do**
3: Generate

$D_i^* = (X_1^*, \ldots, X_n^*)$ by sampling with replacement
{**Nonparametric Bootstrap**, F unknown}

$D_i^* = X[-i]$ {**Jackknife**, F unknown}

$D_i^* = (X_1^*, \ldots, X_n^*)$ by generating from $F(\cdot, \hat{w})$
{**Parametric Bootstrap**, F known}
4: **end for**
5: Distribution of $\widehat{w}$ is estimated by $T(D_1^*), \ldots, T(D_B^*)$
{The histogram based on resampled values is used in place of the true density.}

Uncertainty estimation: confidence intervals

Estimate $100(1-\alpha)\%$ percentile confidence interval for w
$se(\cdot)$ is the square root of estimated variance (computationally heavy)
NOT by jackknife **TOO DEPENDENT!!**

1: Compute $T(D_1^*), \ldots, T(D_B^*)$
2: Sort in ascending order, obtaining $y_1, \ldots, y_B$
{**percentile** method} **OR**
Compute $y_i = (T(D_i^*) - T(D))/(se(T(D_i^*)))$ $i = 1, \ldots, B$
{**t** method}
3: Define $A_1 = \lceil (B\alpha/2) \rceil$, $A_2 = \lfloor (B - B\alpha/2) \rfloor$
4: Confidence interval is given by
(y_{A_1}, y_{A_2}) {**percentile** method} **OR**
$(T(D) - se(T(D^*)) \cdot y_{A_1}, T(D) + se(T(D^*)) \cdot y_{A_2})$
{**t** method}

Hypothesis testing: does statistic from observed data fall into CI (H_0) or not (H_1)

Uncertainty estimation: variance of estimator

Bootstrap

$$\widehat{\mathrm{Var}}[T(\cdot)] = \frac{1}{B-1} \sum_{i=1}^{B} \left(T(D_i^*) - \overline{T(D^*)} \right)^2$$

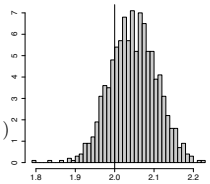
Jackknife ($n = B$)

$$\widehat{\mathrm{Var}}[T(\cdot)] = \frac{1}{n(n-1)} \sum_{i=1}^{n} (T_i^*) - J(T))^2,$$

where

$$T_i^* = nT(D) - (n-1)T(D_i^*) \qquad J(T) = \frac{1}{n} \sum_{i=1}^{n} T_i^*$$

Bootstrap in R



```
library("boot")
stat1<-function(data,vn){
    data<-as.data.frame(data[vn,])
    res<-lm(Response~Predictor,data)
    res$coefficients[2]
}
x<-rnorm(100);data<-cbind(Predictor=x,Response=3+2*
    x+rnorm(length(x),sd=0.5))
res<-boot(data,stat1,R=1000)
print(boot.ci(res))
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
##Based on 1000 bootstrap replicates
#Intervals :
#Level      Normal              Basic
#95%   ( 1.933,  2.164 )   ( 1.935,  2.162 )
# Level      Percentile            BCa
#95%   ( 1.934,  2.161 )   ( 1.936,  2.166 )
```

Bootstrap bias correction

1: Observed data: $D = (X_1, \ldots, X_n)$, estimator $\widehat{w} = T(D)$
2: **for** $i = 1, \ldots, B$ **do**
3: Generate

$D_i^* = (X_1^*, \ldots, X_n^*)$ by sampling with replacement.
4: Calculate $T_i^* = T(D_i^*)$.
5: **end for**
6: Bias corrected estimator is

$$T_1 := 2T(D) - \frac{1}{B}\sum_{i=1}^{B} T_i^*.$$

Jackknife also has a bias correction method (see last year's slides).

Comments

- Jackknife overestimate variance
- Bootstrap–t method is more accurate than percentile
- Permutations: sampling **without** replacement, bootstrap **with**
- Permutation p–value exact if all permutations used, bootstrap always approximate
- Bootstrap may be used for a wider class of problems
- Nonparametric bootstrap works badly for small samples ($n < 40$)
- Parametric bootstrap can work for small samples
- Bias corrections
- Methods do not require distributional assumptions

Permutation tests for model selection

Data predictors: `X[,c(V1,V2)]`, response: `Y`
Model M relating Y and X

Competing models
H_0 variables `V1` should not be in M (smaller model)
H_1 all variables are significant
Test statistic: $T(M)$

Permutation test
1: **for** $i = 1 \ldots B$ **do**
2: Obtain `V1`* by permuting order of columns in `V1`, fit model `Y=M(X[,c(V1*,V2)])`
3: Compute test statistic T_i for this model
4: **end for**
5: Compute p–value using above distribution of T

Summary

- Why are some models better than others?
- Hypothesis testing
- Monte Carlo hypothesis testing
- Resampling methods (permutations, jackknife, bootstrap)
- Simulation methods (parametric bootstrap)

EM Algorithm, Stochastic Optimization

732A90
Computational Statistics

Krzysztof Bartoszek
(krzysztof.bartoszek@liu.se)

7 III 2017
Department of Computer and Information Science
Linköping University

Stochastic and combinatorial optimization

- So far: Unconstrained optimization
  - Predictor variables are continuous
  - Response function is differentiable
- We discussed Steepest descent, Newton, BFGS, CG
- But: predictors can be discrete
  (scheduling problems, travelling salesman)
- But: outcome can be discrete, noisy or multi–modal

Stochastic and combinatorial optimization

Given a (large) set of states S, find

$$\min_{s \in S} f(s)$$

- Exhaustive search (shortest path algorithm)
- Often exhaustive search is NP–hard (TSP)
- Alternative: stochastic methods
  random search

Simulated annealing

Motivation from physics: cooling of metal

- Parameters:
  Energy of metal
  (decreasing, but not strictly monotonic)
  Temperature (decreasing)
- Aim: find global minimum energy

Simulated annealing

0. Set $k = 1$ and initialize state s.
1. Compute the temperature $T(k)$.
2. Set $i = 0$ and $j = 0$.
3. Generate a new state r and compute $\delta f = f(r) - f(s)$.
4. Based on δf, decide whether to move from state s to state r.
   If $\delta f \leq 0$,
   accept state r;
   otherwise,
   accept state r with a probability $P(\delta f, T(k))$.
   If state r is accepted, set $s = r$ and $i = i + 1$.
5. If i is equal to the limit for the number of successes at a given temperature, go to step 1.
6. Set $j = j + 1$. If j is less than the limit for the number of iterations at given temperature, go to step 3.
7. If $i = 0$,
   deliver s as the optimum; otherwise,
   if $k < k_{\max}$,
   set $k = k + 1$ and go to step 1;
   otherwise,
   issue message that
   'algorithm did not converge in $k_{\max}$ iterations'.

Simulated annealing

- `https://www.youtube.com/watch?v=iaq_Fpr4KZc`
- Generating new state:
  - Continuous: choose a new point a (random) distance from the current one
  - Discrete: similar or some rearrangement
- Selection probability: e.g $\exp(-\delta f(x)/T)$: decreasing with $f(x)$, increasing with T
- Temperature function: constant, proportional to k, or

$$T(k+1) = b(k)T(k), \quad b(k) = (\log(k))^{-1}$$

Remember: A smaller value is better than one on the path to the global minimum! Always keep track of smallest found.

Simulated annealing: TSP example

Assume constant temperature

1: Choose initial configuration $(Town_1, \ldots, Town_n)$
2: $k = 1$
3: **while** $k < k_{max} + 1$ **do**
4: Generate new configuration by rearrangement,

$$(1,2,3,4,5,6,7,8,9) \rightarrow (1,6,5,4,3,2,7,8,9)$$
$$(1,2,3,4,5,6,7,8,9) \rightarrow (1,7,8,2,3,4,5,6,9)$$

5: Measure difference in path length (δf) between old and new configuration
6: **if** shorter path found **then**
7: accept it
8: **else**
9: accept it with probability $P(\delta f)$
10: **end if**
11: $k++$
12: **end while**

Genetic algorithm

- Inspiration from evolutionary theory: survival of the fittest
- Variables=genotypes
- Observation=organism, characterized by genetic code
- State space=population of organisms
- Objective function=fitness of organism

New points are obtained from old points by crossover and mutation, the population only retains the fittest organisms (with better objective function).

`https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications`

Genetic algorithm

Encoding points

1. Enumerate each element of the state space, S
2. Code for observation i is binary representation of i (or something else)

Mutation and recombination rules

Generation k	Generation $k+1$
Crossover	
$x_i^{(k)}$ 11001001	
	$\rightarrow x_s^{(k+1)}$ 11011010
$x_j^{(k)}$ 00111010	
Inversion	
$x_i^{(k)}$ 11101011	$\rightarrow x_s^{(k+1)}$ 11010111
Mutation	
$x_i^{(k)}$ 11101011	$\rightarrow x_s^{(k+1)}$ 10111011
Clone	
$x_i^{(k)}$ 11101011	$\rightarrow x_s^{(k+1)}$ 11101011

Genetic algorithm

0. Determine a representation of the problem, and define an initial population, $x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}$. Set $k = 0$.
1. Compute the objective function (the "fitness") for each member of the population, $f(x_i^{(k)})$ and assign probabilities p_i to each item in the population, perhaps proportional to its fitness.
2. Choose (with replacement) a probability sample of size $m \leq n$. This is the reproducing population.
3. Randomly form a new population $x_1^{(k+1)}, x_2^{(k+1)}, \ldots, x_n^{(k+1)}$ from the reproducing population, using various mutation and recombination rules (see Table 6.2). This may be done using random selection of the rule for each individual of pair of individuals.
4. If convergence criteria are met, stop, and deliver $\arg\min_{x_i^{(k+1)}} f(x_i^{(k+1)})$ as the optimum; otherwise, set $k = k + 1$ and go to step 1.

Genetic algorithm: TSP example

Encoding and crossover

- Encode tours as $A_1, \ldots, A_n$ but

  Parent 1: FAB|ECGD Parent 2: DEA|CGBF
  Child: FAB|CGBF Child: DEA|ECGD

Instead

1. Remove FAB from DEACGBF $\longrightarrow$ DECG.
   Child becomes FABDECG.
2. Second child will be by taking prefix from Parent 2:
   DEAFBCG

Genetic algorithm: Mutations

- If a population is small and only crossover: the input domain becomes limited and may converge to a local minimum.
- Large initial populations are computationally heavy.
- Mutations allow one to explore more of S: jump out of local minimum.
- In TSP: mutation move a city in the tour to another position.
- Reproduction: Among m tours selected at step 2, two best are selected for reproduction, two worst replaced by children.
- If m is large, some tours might never be parents, global solution may be missed. Random chance of reproduction?
- Mutation probability is usually small (unless you want to jump wildly)

EM algorithm

Fundamental algorithm of computational statistics!

Model depends on the data which are observed (known) $\mathbf{Y}$ and **latent** (unobserved) data $\mathbf{Z}$.

The data's (**both Y**'s and **Z**'s) distribution depends on some parameters θ.

AIM: Find MLE of θ.

- All data is known: Apply unconstrained optimization (discussed in Lecture 2)
- Unobserved data
  - **Sometimes** it is possible to look at the marginal distribution of the observed data.
  - Otherwise: **EM algorithm**

EM algorithm

Let

$$Q(\theta, \theta^k) = \int \log p(\mathbf{Y}, \mathbf{z}|\theta) p(\mathbf{Y}|\mathbf{z}, \theta^k) \mathrm{d}\mathbf{z} = \mathrm{E}\left[\mathrm{loglik}(\theta|\mathbf{Y}, \mathbf{Z})|\theta^k, \mathbf{Y}\right]$$

1: $k = 0$, $\theta^0 = \theta^0$
2: **while** Convergence not attained **and** $k < k_{max} + 1$ **do**
3: **E–step**: Derive $Q(\theta, \theta^k)$
4: **M–step**: $\theta^{k+1} = \mathrm{argmax}_\theta \;\; Q(\theta, \theta^k)$
5: $k++$
6: **end while**

Example: Normal data with missing values (but here analytical approach is also possible)

EM algorithm: R

```
floglik<-function(y, mu, sigma2, n){ -0.5*n*log(2*pi*sigma2)-0.5*sum
    ((y-mu)^2)/sigma2}

EM.Norm<-function(Y,eps,kmax){
    Yobs <- Y[!is.na(Y)]; Ymiss <- Y[is.na(Y)]
    n <- length(c(Yobs, Ymiss)); r <- length(Yobs)

    k<-1;muk<-1;sigma2k<-0.1

    llvalprev<-floglik(Yobs,muk,sigma2k,r);
    llvalcurr<-llvalprev+10+100*eps
    print(c(muk,sigma2k,llvalcurr))

    while ((abs(llvalprev-llvalcurr)>eps) && (k<(kmax+1))){
        llvalprev<-llvalcurr
        ## E-step
        EY<-sum(Yobs)+(n-r)*muk
        EY2<-sum(Yobs^2)+(n-r)*(muk^2+sigma2k)

        ## M-step
        muk<-EY/n
        sigma2k<-EY2/n-muk^2

        ## Compute log-likelihood
        llvalcurr<-floglik(Yobs,muk,sigma2k,r)
        k<-k+1

        print(c(muk,sigma2k,llvalcurr))
    }
}
```

EM algorithm: R

```
> Y<-rnorm(100)
> Y[sample(1:length(Y),20,replace=FALSE)]<-NA
> EM.Norm(Y,0.0001,100)
[1]    1.0000    0.1000 -997.5705
[1]    0.1341894    1.3227095 -128.2789837
[1]   -0.03897274    1.38734070 -126.86036252
[1]   -0.07360517    1.39307050 -126.80801589
[1]   -0.08053165    1.39392861 -126.80593837
[1]   -0.08191695    1.39408871 -126.80585537
> mean(Y,na.rm=TRUE)
[1] -0.08226328
> var(Y,na.rm=TRUE)
[1] 1.411775
```

Notice: can be done by studying marginal distribution of observed data.

EM algorithm: Applications

Mixture models Z is a latent variable, $P(Z = k) = \pi_k$

- Mixed data comes from different sources (e.g. for regression, classification)
- Clustering
  1. Density in each cluster is normally distributed.
  2. Cluster label is latent (we do not know what are the chances an observation is from the given cluster)

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\vec{\mu}_k, \mathbf{\Sigma}_k) \quad \text{(informally)}$$

Direct MLE leads to numerical problems.
Introduce latent class variables and use EM.

Summary

Random walk over the state space in search of minimum

1. Follow decreasing path
2. **BUT** with a certain probability go to higher values, to avoid local minima traps.
3. **Never forget** best found conformation!

4. Simulated annealing, Genetic algorithm, **EM algorithm**,
   Stochastic gradient descent (see last year's slides)

EM algorithm: Gaussian mixtures

1. Initialize the means $\boldsymbol{\mu}_k$, covariances $\mathbf{\Sigma}_k$ and mixing coefficients π_k, and evaluate the initial value of the log likelihood.
2. **E step**. Evaluate the responsibilities using the current parameter values

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \mathbf{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \mathbf{\Sigma}_j)}. \tag{9.23}$$

3. **M step**. Re-estimate the parameters using the current responsibilities

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \tag{9.24}$$

$$\mathbf{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})^{\mathrm{T}} \tag{9.25}$$

$$\pi_k^{\text{new}} = \frac{N_k}{N} \tag{9.26}$$

where

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}). \tag{9.27}$$

4. Evaluate the log likelihood

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \mathbf{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \mathbf{\Sigma}_k) \right\} \tag{9.28}$$

and check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied return to step 2.

$$Ez_{nk} = \gamma(z_{nk})$$

Source: Pattern recognition by Bishop