Exam Help

Henrik Karlsson

3/2/2018

Contents

Math - Derivation rules

$\frac{d}{dX}\left(ln(x)\right) = \frac{1}{x}$; $\frac{d}{dX}\left(ln(x^2)\right) = \frac{2}{x}$

$\frac{d}{dX}\left(\frac{1}{x}\right) = -\frac{1}{x^2}$; $\frac{d}{dX}\left(\frac{1}{x^2}\right) = -\frac{2}{x^3}$

$\frac{d}{dX}\left((x-y)^2\right) = 2(x-y)$

$\frac{d}{dX}\left(e^x\right) = e^x$

$(x^2)^2 = x^4$

If distribution contains absolute values around x, split up the distribution.

Math - Integration rules

$\int_a^b \frac{1}{2}e^t = \frac{1}{2}\left[e^t\right]_a^b$; $\int_a^b \frac{1}{2}e{-}t = \frac{1}{2}\left[-e^{-t}\right]_a^b$

$e^{-\infty} = 0$

Math - Conditional Marginal Distribution

Discrete RV: $Pr(X = x) = \sum_y Pr(X = x, Y = y) = \sum_y Pr(X = x|Y = y)Pr(Y = y)$

Continuous RV: $p_X(x) = \int_y p_{X,Y}(x, y)dy = \int_y p_{X|Y}(x|y)p_Y(y)dy$. If I want to find marginal dist for X, solve joint PDF with respect to Y.

Lab1 - Question 1 - Over- and underflow

Question: Be careful when comparing

Overflow = Number larger than can be represented. **Underflow** = Loss of significant digits. When suming fractions, start with larger fractions.

```
x1 <- 1/3
x2 <- 1/4
if (x1-x2 == 1/12){
  print("Subtraction is correct")
} else {
  print("Subtraction is wrong")
}
```

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

```
x1 <- 1
x2 <- 1/2
if (x1-x2==1/2){
  print("Subtraction is correct")
} else {
  print("Subtraction is wrong")
}
```

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

L1-Q1-1.

Question: Check the results of the snippets. Comment what is going on.

When comparing rational numbers, R rounds with floating point precision, at best we obtain

$$x_1 - x_2 = 0.33333333333333333148296 - 0.25 = 0.083333333333333331482962$$

because of rounding precision errors, whereas $\frac{1}{12} = 0.083333333333333287074$. So subtracting $\frac{1}{3}$ with 0.25 yields rounding errors - since $\frac{1}{3}$ has infinite amount of decimals.

This is happing because of the **underflow** phenomenom, which implies that we don't have enough numeric precision to represent the numbers. To represent $\frac{1}{3}$ in floating point precision would require infinitely amount of space, therefore we round the fraction to fit in one bit.

L1-Q1-2.

Question: If there are any problems, suggest improvements.

Using the function **all.equals()** we obtain the true result of the comparison of subtraction. The function all.equals() return the TRUE or the difference if not TRUE, so it's useful to wrap it in the function **isTRUE(all.equals())**.

```r
x1 <- 1/3
x2 <- 1/4
options(digits = 22)
if (isTRUE(all.equal(x1-x2,1/12))){
  print("Subtraction is correct")
} else {
  print("Subtraction is wrong")
}
```

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

```r
x1 <- 1
x2 <- 1/2
if (isTRUE(all.equal(x1-x2, 1/2))){
  print("Subtraction is correct")
} else {
  print("Subtraction is wrong")
}
```

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

Lab1-Question 2: Derivative

From the defintion of a derivative a popular way of computing it at a point x is to use a small ϵ and the formula.

$$f'(x) = \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

L1-Q2-1. Define derivative funtion

Question: Write your own R function to calculate the derivative of $f(x) = x$ in this way with $\epsilon = 10^{-15}$.

```r
# Version 1 - General solution
f <- function(x) {
  return(x)
}

fp <- function(x,eps) {
  eps <- 10^(-15)

  return((f(x+eps) - f(x))/eps)
}

# Version 2
derivative <- function(x, e = 10^-15){
    res <- ((x+e) - x)/e
    return(res)
}
```

L1-Q2-2. Underflow

Question: Evaluate your derivative function at x = 1 and x = 100000.

```
fp(1)
```

```
## [1] 1.110223024625156540424
```

```
fp(100000)
```

```
## [1] 0
```

```
fp(16)
```

```
## [1] 0
```

Since we have higher density in computer representation between $[-2^0, 2^0]$ comparing larger numbers will yield a less density in computer representation, hence the value $10^5 + 10^{-15} - 10^5 = 0$ in computer arithmetics whereas $1 + 10^{-15} - 1 = 10^{-15}$. This is due to the fact that the exponential fills the space so the mantissa gets less space and the higher our x gets, above 15 the mantissa gets no space.

L1-Q2-3. Underflow

Question: What values did you obtain? What are the true values? Explain the reasons behind the discovered differences.

With $f'(1) = 1.110223024625156540424$ and with $f'(10^5) = 0$ due to the previous explanation whereas the true values should be $f'(x) = 1$

Lab1-Question 3: Variance

A known formula for estimating the variance based on a vector of n observations is

$$Var(\vec{x}) = \frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right)$$

L1-Q3-1. Estimate Variance

Question: Write your own R function, *myvar*, to estimate the variance in this way.

```
myvar <- function(x){
  n = length(x)
  res <- 1/(n-1)*(sum(x^2) - (1/n)*(sum(x)^2))
  return(res)
}
```

L1-Q3-2.

Question: Generate a vector $x = (x_1, ..., x_{10000})$ with 10000 random numbers with mean 10^8 and variance 1.

```r
set.seed(1337)
x <- rnorm(n = 10000, mean = 10^8, sd = sqrt(1))

myvar(x)
```

```
## [1] 0
```

```r
var(x)
```

```
## [1] 1.0017664372910224379 61
```

We assume normality in the vector.

L1-Q3-3. Floating point percision

Question: For each subset $X_i = \{x_1, ..., x_i\}$, i = 1 ,..., 10000 compute the difference $Y_i = myvar(X_i) - var(X_i)$, where $var(X_i)$ is the standard variance estimation function in R. Plot the dependence Y_i on i. Draw conclusions from this plot. How well does your function work? Can you explain the behaviour?

```r
Y <- c()
for(i in 1:length(x)){
    Y[i] <- myvar(x[1:i]) - var(x[1:i])
}
Y <- Y[-1]
library(ggplot2)
# ggplot() +
#     geom_point(aes(x = 2:10000, y = Y, alpha = 0.01))
```

First observation is removed, because variance of 1 number is NaN.

When subtracting **myvar() - var()** for certain intervals the difference is -1, this is because **myvar()** is suffering from floating point precision due to performing operations on large numbers which will results in **myvar()** = 0. The **var()** scales the numbers before performing the operations, and therefore computes a precise result, 1, and subtraction **myvar() - var()** = -1. The other values not equal or close to 0 are noise.

The function does not perform well with unscaled numbers.

L1-Q3-4. Floating point percision improvements

Question: How can you better implement a variance estimator? Find and implement a formula that will give the same results as var()?

```r
myvar2 <- function(x) {
  n <- length(x)
  x_c <- x - mean(x)
  res <- 1/(n-1)*(sum(x_c^2) - (1/n)*(sum(x_c)^2))
  return(res)
}

options(digits = 22)
myvar2(x) - var(x)
```

```
## [1] 0
```

```r
Y2 <- c()
for(i in 1:length(x)){
```

```
    Y2[i] <- myvar2(x[1:i]) - var(x[1:i])
}
Y2 <- Y2[-1]
# ggplot() +
#     geom_point(aes(x = 2:10000, y = Y2)) +
#     geom_abline(slope = 0, intercept = 0, color = "red")
```

If we center the x vector we remove the precision problems since we have number representation near 0 for floating point values.

Lab1-Question 4: Linear Algebra

L1-Q4-1.

Question: Import the data set to R

```
df <- read.csv("~/Google Drive/Statistics and Data Mining/732A90 Computational Statistics/CS_Lab1/tecat
Y <- as.numeric(df$Protein)
X <- as.matrix(df[, c(-1,-103)])
```

L1-Q4-2. Singluarity issues in matrix multiplication

Question: Optimal regression coefficients can be found by solving a system of the type $\mathbf{A}\vec{\beta} = \vec{b}$ where $\mathbf{A} = \mathbf{X^T X}$ and $\vec{b} = \mathbf{X^T}\vec{y}$. Compute $\mathbf{A}$ and $\vec{b}$ for the given data set. The matrix $\mathbf{X}$ are the observations of the absorbance records, levels of moisture and fat, while $\vec{y}$ are the protein levels.

```
A <- t(X)%*%X
b <- t(X)%*%Y
# B <- solve(A) %*% b # This can't be run due to singularity
```

L1-Q4-3. Cholesky decomposition

Question: Try to solve $\mathbf{A}\vec{\beta} = \vec{b}$ with default solver solve(). What kind of result did you get? How can this result be explained?

```
library(Matrix)
rankMatrix(A)[1]
```

```
## [1] 70
```

```
ch_A <- chol(A)
rankMatrix(ch_A)[1]
```

```
## [1] 102
```

```
B <- solve(ch_A) %*% b
```

Matrix A is singular (not of full rank) and therefore not invertible. Either, you can scale matrix X and vector Y or apply Cholesky Decomposition, matrix ch_A becomes full rank and we are able to perform the inverse. It's a good practice to scale variables before conducting statistical analysis, especially if the numeric values differentiate in scales.

L1-Q4-4. Condition number - Kappa

Question: Check the condition number of the matrix **A** (function kappa()) and consider how it is related to your conclusion in step 3.

```
kappa(A)
```

```
## [1] 1157834236871692.25
```

```
kappa(ch_A)
```

```
## [1] 45492443.62946256250143
```

After applying Cholesky decomposition, we can see that the $\kappa(\mathbf{A}) = ||\mathbf{A}^{-1}||\,||\mathbf{A}||$ value, the conditional number of the matrix, is greatly reduced which is good since it's indicating less ill-conditioning of the matrix.

Where as we have a large value of $\kappa(\mathbf{A})$ doesn't imply we have an ill-conditioned matrix but it's a bad sign and warrants further investigation.

The implications of an ill-conditioned matrix, operations of the matrix magnifies the values greatly instead of proportionately. Meaning that the function is sensitive to change and noise in the input, which results in greater errors in the output.

L1-Q4-5. Scaling and comparing results

Question: Scale the data set and repeat steps 2–4. How has the result changed and why?

```
Y_s <- scale(Y)
X_s <- scale(X)

A_s <- t(X_s)%*%X_s
b_s <- t(X_s)%*%Y_s
B_s <- solve(A_s) %*% b_s

kappa(A_s)
```

```
## [1] 490471520662.0501098633
```

After scaling the input data, matrix A is of full rank (thus now invertible) and the condition number (κ) has lowered. Still performing the Cholesky decomposition, yields an even lower condition number (κ). Scaling the matrix removes linear dependency thus makes it invertible. Also scaling squishes the values to the scale in each variable, around zero, which increases computing precision (since the floating point numbers have higher density around zero).

Lab2-Question1 Optimizing a Model Parameter

L2-Q1-1. Import data

Question: Import this file to R and add one more variable LMR to the data which is the natural logarithm of Rate.

```
options(digits = 6)
data <- read.csv2("~/Google Drive/Statistics and Data Mining/732A90 Computational Statistics/CS_Lab2/mo
data$LMR <- log(data$Rate)

n=dim(data)[1]
```

```r
set.seed(123456)

id=sample(1:n, floor(n*0.5))
train=data[id ,]
test=data[-id ,]
```

The data is being read in split into two partitions, train and test.

L2-Q1-2. Build MSE function

Write your own function myMSE() that for given parameters λ and list pars containing vectors X, Y, Xtest, Ytest fits a LOESS model with response Y and predictor X using loess() function with penalty λ (parameter enp.target in loess()) and then predicts the model for Xtest. The function should compute the predictive MSE, print it and return as a result. The predictive MSE is the mean square error of the prediction on the testing data.

```r
myMSE <- function(lambda, pars){
    res <- loess(Y ~ X, data = pars, enp.target = lambda)
    MSE <- 1/length(pars$Xtest) * sum((pars$Ytest - predict(res, newdata = pars$Xtest))^2)
    counter <<- counter + 1
    return(MSE)
}
```

A function to compute *Mean Square Error* is defined.

L2-Q1-3. Brute-force Optimization

Question: Use a simple approach: use function myMSE(), training and test sets with response LMR and predictor Day and the following λ values to estimate the predictive MSE values: $\lambda = 0.1, 0.2, ..., 40$

```r
l <- list(Y = train$LMR,
          X = train$Day,
          Ytest = test$LMR,
          Xtest = test$Day)

lambdas <- seq(0.1, 40, 0.1)

MSE <- c()
counter <- 0
MSE <- sapply(X = 1:length(lambdas), FUN = function(X){
    MSE[X] <- myMSE(lambdas[X], l)
})
counter_MSE <- counter
```

The defined MSE-function is used to compute the mean square error of Y from test data set for different pentalties. MSE is computed for each $\lambda = 0.1, 0.2, ..., 40$.

L2-Q1-4. Brute force optimization continuous

Question: Create a plot of the MSE values versus λ and comment on which λ value is optimal. How many evaluations of myMSE() were required (read ?optimize) to find this value?

```r
# plot(x = lambdas, y = MSE)

output <- data.frame(f = "MSE_full_search",
                     minMSE = min(MSE),
                     lambda = lambdas[which.min(MSE)],
                     iteration = counter_MSE,
                     stringsAsFactors = FALSE)
# knitr::kable(output)
```

The graph above shows the MSE for different values of λ. The minimum MSE, 0.1310469648318724755764, is acquired at $\lambda = 11.7$. To find the minimum value we have to use brute-force so we require all the necessary iterations done (400).

L2-Q1-5. Optimize()

Question: Use optimize() function for the same purpose, specify range for search [0.1,40] and the accuracy 0.01. Have the function managed to find the optimal MSE value? How many myMSE() function evaluations were required? Compare to step 4.

```r
counter <- 0
opt <- optimize(f = myMSE, interval = c(0.1,40), tol = 0.01, pars = l)
counter_opt <- counter

output[2,] <- list(f = as.character("optimize()"),
                   minMSE = opt$objective,
                   lambda = opt$minimum,
                   iteration = counter_opt)
# knitr::kable(output)
```

The **optimize()** function finds a slightly different minimal MSE and λ, already after 18 iterations, compared to the loop that requires to compute the MSE for all 400 inserted pentalty values. The optimize function, also have a higher precision, since we are using a lower accuracy (0.01).

The **optimize()** uses a combination of the *Golden Section Search* algorithm and *Successive Parabolic Interpolation* algorithm. The Golden Section Search algorithm selects a interval and searches for the minimum within that interval, by narrowing down the serach interval with a constant reduction factor. Here, the optimize function searches in the interval of 0.1 to 40.

L2-Q1-6. optim()

Question: Use optim() function and BFGS method with starting point $\lambda = 35$ to find the optimal λ value. How many myMSE() function evaluations were required (read ?optim)? Compare the results you obtained with the results from step 5 and make conclusions.

```r
counter <- 0
opt2 <- optim(par = 35, fn = myMSE, method = "BFGS", pars = l)
counter_opt2 <- counter

output[3,] <- list(f = as.character("optim()"),
                   minMSE = opt2$value,
                   lambda = opt2$par,
                   iteration = opt2$counts[1])

# knitr::kable(output)
```

The **optim()** function checks for if it can descend in any direction and goes in the direction with strongest slope. In this case λ is specified at 35, which is a local minimum (which also can be seen in previous graph), therefore the **optim()** is stuck and stops after first iteration.

optim() with method *BFGS* is a quasi-Newton method, which means that the optimization algorithm approximate the Hessian matrix. When the algorithm uses approximations, it usually needs more iterations to find the optimal threshold, but each iteration require less computations which in the end usually generates a shorter overall computating time.

Lab2-Question 2: Maximizing Likelihood

L2-Q2-1. Data import

Question: Load the data to R environment.

```
load(file = "~/Google Drive/Statistics and Data Mining/732A90 Computational Statistics/CS_Lab2/data.RDa
```

L2-Q2-2. Derive Maximum likelihood for Normal

Question: Write down the log-likelihood function for 100 observations and derive maximum likelihood estimators for μ, σ analytically by setting partial derivatives to zero. Use the derived formulae to obtain parameter estimates for the loaded data.

The log likelihood of the *Gaussian Distribution* is:

$$l(\mu, \sigma^2; x_1, ..., x_n) = -\frac{n}{2} ln(2\pi) - \frac{n}{2} ln(\sigma^2) - \frac{1}{2\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2$$

The partial derivative of μ:

$$\frac{\partial L(\mu)}{\partial \mu} = -\frac{1}{2\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2 = \frac{1}{\sigma^2} \sum_{j=1}^{n} x_j - n\mu$$

We set the derivative equal to zero and then:

$$\frac{1}{\sigma^2} \sum_{j=1}^{n} x_j - n\mu = 0 \Leftrightarrow \hat{\mu} = \frac{1}{n} \sum_{j=1}^{n} x_j$$

The partial derivative of σ:

$$\frac{\partial L(\sigma)}{\partial \sigma} = -\frac{n}{2} ln(\sigma^2) - \frac{1}{2\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2 = -\frac{n}{\sigma} + \frac{1}{\sigma^3} \sum_{j=1}^{n} (x_j - \mu)^2$$

We set the derivative equal to zero and then:

$$-\frac{n}{\sigma} + \frac{1}{\sigma^3} \sum_{j=1}^{n} (x_j - \mu)^2 = 0 \Leftrightarrow \frac{1}{n} \sum_{j=1}^{n} (x_j - \mu)^2 = \sigma^2$$

```r
ll_gaussian <- function(data = data, mu, sigma){
    scaling <- data - mu
    res <-
      -(length(data) / 2) * log(2 * pi) -
      (length(data) / 2) * log(sigma ^ 2) -
      1 / (2 * sigma ^ 2) * sum((scaling) ^ 2)
    return(res)
}
```

```r
options(digits= 6)
n <- length(data)
mu <- 1/n * sum(data)
sigma <- 1/n * sum((data - mu)^2)

ml <- ll_gaussian(data, mu, sigma)
```

The obtained estimates for $\hat{\mu} = 1.27553$ and $\sigma^2 = 2.95878$ (this number is weird), computing the log likelihood for our parameter estimates gives us a log likelihood of -243.546.

L2-Q2-3. Conjugate Gradient Method and BFGS with and without gradient

Question: Optimize the minus log–likelihood function with initial parameters $\mu = 0$, $\sigma = 1$. Try both Conjugate Gradient method (described in the presentation handout) and BFGS (discussed in the lecture) algorithm with gradient specified and without. Why it is a bad idea to maximize likelihood rather than maximizing log–likelihood?

```r
ll_gaussian <- function(par,pars){
  mu <- par[1]
  sigma <- par[2]
  scaling <- pars - mu
  res <-
    -(length(pars) / 2) * log(2 * pi) -
    (length(pars) / 2) * log(sigma ^ 2) -
    1 / (2 * sigma ^ 2) * sum((scaling) ^ 2)
  return(-res)
}

gr_gaussian <- function(par,pars) {
  mu <- par[1]
  sigma <- par[2]
  n <- length(pars)
  scaling <- pars - mu

  res <- c(- sum(scaling) / sigma^2, n / sigma - sum(scaling^2) / sigma^3)
  return(res)
}

time.taken <- c()
start.time <- Sys.time()
cgGR <- optim(c(0,1), fn=ll_gaussian, gr=gr_gaussian, method="CG", pars = as.vector(data))
end.time <- Sys.time()
time.taken[1] <- end.time - start.time

start.time <- Sys.time()
```

```r
cg <- optim(c(0,1),ll_gaussian, method="CG", pars = as.vector(data))
end.time <- Sys.time()
time.taken[2] <- end.time - start.time

start.time <- Sys.time()
bfgsGR <- optim(c(0,1),ll_gaussian, gr_gaussian, method="BFGS", pars = as.vector(data))
end.time <- Sys.time()
time.taken[3] <- end.time - start.time

start.time <- Sys.time()
bfgs <- optim(c(0,1),ll_gaussian, method="BFGS", pars = as.vector(data))
end.time <- Sys.time()
time.taken[4] <- end.time - start.time
```

Since we are working with a floating point precision system we prefer values closer to 0 rather than large numbers, which the log-likelihood will yield. Whereas the likelihood function values will be larger. Since the log-function is monotone the maxima and minima will be at the same x-value as the same function. Also taking and using derivative of likelihood function is difficult analytically.

Since we are taking the gradient of the minus log-likelihood, the sign of the gradient changes from the analytically obtained gradient.

The gradient function is the partial derivatives of the likelihood (before we set them to 0).

L2-Q2-4. Comparision Conjugate Gradient Method and BFGS

Question: Did the algorithms converge in all cases? What were the optimal values of parameters and how many function and gradient evaluations were required for algorithms to converge? Which settings would you recommend?

```r
funs <- c("C. Gr. Desc. w. gr", "C. Gr. Desc.", "BFGS w. gr", "BFGS")
convs <- c(cgGR$convergence, cg$convergence, bfgsGR$convergence, bfgs$convergence)
mus <- c(cgGR$par[1], cg$par[1], bfgsGR$par[1], bfgs$par[1])
sigmas <- c(cgGR$par[2], cg$par[2], bfgsGR$par[2], bfgs$par[2])
llik <- c(cgGR$value, cg$value, bfgsGR$value, bfgs$value)
iters <- c(cgGR$counts[1], cg$counts[1], bfgsGR$counts[1], bfgs$counts[1])
giters <- c(cgGR$counts[2], cg$counts[2], bfgsGR$counts[2], bfgs$counts[2])
res <- data.frame(
  Functions = funs,
  Convergence = convs,
  Mu = mus,
  Sigma = sigmas,
  LogLike. = llik,
  F.Iter. = iters,
  G.Iter. = giters,
  Time = time.taken)
```

Functions	Convergence	Mu	Sigma	LogLike.	F.Iter.	G.Iter.	Time
C. Gr. Desc. w. gr	0	1.27553	2.00598	211.507	53	17	0.029437
C. Gr. Desc.	0	1.27553	2.00598	211.507	180	33	0.004702
BFGS w. gr	0	1.27553	2.00598	211.507	38	15	0.001949
BFGS	0	1.27553	2.00598	211.507	37	15	0.003825

All the algorithms converged, with 0 being convergence in the table. For the optimal values and number of function and gradient evaluations we refer to the table above.

Since BFGS has more iterations than Newton's method, but each iterations is quicker - then it is more preferable. Though comparing CGD and BFGS we note that CGD is slower than BFGS, but uses significantly less memory (since it does not have to store a large inverted matrix) and CGD has local minima convergence which makes it more reliable than the BFGS (in situations where there exists weak local minima).

Concludingly, since we have plenty of memory available and the BFGS (without gradient information) obtains the min log-likelihood value after fewest iterations and lowest execution time, it is the recommended model in this setting.

Lab3-Question 1: Cluster Sampling

L3-Q1-1. Import data

```
df <- read.csv2("~/Google Drive/Statistics and Data Mining/732A90 Computational Statistics/CS_Lab3/popul
df$Municipality <- as.character(df$Municipality)
```

L3-Q1-2. Select by proportional probability

Question: Use a uniform random number generator to create a function that selects 1 city from the whole list by the probability scheme offered above

```
set.seed(1991)
prop_random_city <- function(prop_var){
    # Randomizer
    rnd <- runif(1)
    # Cumulative sum
    csum <- cumsum(prop_var)/sum(prop_var)
    # Find interval for randomizer
    selected_city <- findInterval(rnd, csum) + 1
    return(selected_city)
}

prop_random_city(df$Population)
```

```
## [1] 16
```

L3-Q1-3. Building select and remove selected function

Question: Use the function you have created in step 2 as follows:

- Apply it to the list of all cities and select one city
- Remove this city from the list
- Apply this function again to the updated list of the cities
- Remove this city from the list
- ... and so on until you get exactly 20 cities.

```
select_cities <- function(data, nr_cities){
    tmp <- data
    output <- data.frame(Municipality = NA,
```

```
                          Population = NA)
    for(i in 1:nr_cities){
        selected <- prop_random_city(tmp$Population)
        output[i,] <- tmp[selected,]
        tmp <- tmp[-selected,]
    }
    return(output)
}
```

L3-Q1-4. Evaluate selected cities

Question: Run the program. Which cities were selected? What can you say about the size of the selected cities?

```
cities <- select_cities(df, 20)
# knitr::kable(cities)
```

The above cities were randomly selected. We can see that the cities above is overrepresented by larger cities. The average city in Sweden have 32209 inhabitants meanwhile the average of our selected cities have 85344 inhabitants. This is explained by the fact that each city have the probability of being selected proportional to their population size, which yields that a larger city is more likely to be selected in our sample.

L3-Q1-5. Plotting results

Question: Plot one histogram showing the size of all cities of the country. Plot another histogram showing the size of the 20 selected cities. Conclusions?

```
# par(mfrow =c(1,2))
# hist(df$Population, breaks = 50, freq = FALSE, main = "City Population", xlab = "Population")
# hist(cities$Population, breaks = 50, freq = FALSE, main = "City Sample Population", xlab = "Populatio
```

We notice the same pattern when comparing the histograms, the relative proportion of large cities is much higher in our sample of randomly selected cities compare to the entire population.

Lab3-Question 2: Different distributions

L3-Q2-1. Inverse CDF method

From slides: Let X be a random variable with CDF $X \sim F_X$ (F is strictly increasing). Consider $Y = F_X^{-1}(U),\ where\ U \sim Unif(0,1)$

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

Y has same probability distribution as X.

To compute the inverse of a function, compute the CDF and set the CDF = Y, where Y is uniform distribution from 0 to 1. Then solve for X and we find how to draw Y to find the inverse of X, the inverse of the CDF.

Question: The double exponential (Laplace) distribution is given by formula: $DE(\mu, \alpha) = \frac{\alpha}{2} exp(-\alpha|x - \mu|)$. Write a code generating double exponential distribution DE(0, 1) from Unif(0, 1) by using the inverse CDF

method. Explain how you obtained that code step by step. Generate 10000 random numbers from this distribution, plot the histogram and comment whether the result looks reasonable.

In order to utilize the *inverse CDF method* starting with a *Probability Density Function* (PDF), you should:

1. Integrate the PDF to obtain the CDF.
2. Inverse the CDF by solving for Y, note that not all distributions can be inverted.

The probability density function of the double exponential (Laplace) distribution is given by formula:

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

Since the double exponential distribution is **strictly increasing,** we can utilize the inverse CDF method.

$$f(x) = \begin{cases} \frac{\alpha}{2} exp(-\alpha(x - \mu)), & \text{if } x \geq \mu. \\ \frac{\alpha}{2} exp(\alpha(x - \mu)), & \text{if } x < \mu. \end{cases}$$

When $x \geq \mu$:

$$F_{x \geq \mu}(x) = \int_{-\infty}^{x} f(x)dx = \int_{\mu}^{x} f(x)dx + \int_{-\infty}^{\mu} f(x)dx =$$

$$\int_{\mu}^{x} \frac{\alpha}{2} e^{-\alpha(x-\mu)}dx + \int_{-\infty}^{\mu} \frac{\alpha}{2} e^{\alpha(x-\mu)}dx = \frac{\alpha}{2} \left(\left[-\frac{1}{\alpha} e^{-\alpha(x-\mu)} \right]_{\mu}^{x} + \left[\frac{1}{\alpha} e^{\alpha(x-\mu)} \right]_{-\infty}^{\mu} \right) =$$

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

explain why we use the intervals we do over the second integral

When $x < \mu$:

$$F_{x<\mu}(x) = \int_{-\infty}^{x} \frac{\alpha}{2} e^{\alpha(x-\mu)}dx = \frac{\alpha}{2} \left[\frac{1}{\alpha} e^{\alpha(x-\mu)} \right]_{-\infty}^{x} = \frac{1}{2} e^{\alpha(x-\mu)}$$

So:

$$\therefore F(x) = \begin{cases} 1 - \frac{1}{2} e^{-\alpha(x-\mu)}, & \text{if } x \geq \mu. \\ \frac{1}{2} e^{\alpha(x-\mu)}, & \text{if } x < \mu. \end{cases}$$

The inverse of the CDF is:

$$x \geq \mu : u = 1 - \frac{1}{2} e^{-\alpha(x-\mu)} \Leftrightarrow 2 - 2u = e^{-\alpha(x-\mu)} \Leftrightarrow \ln(2 - 2u) = -\alpha(x - \mu) \Leftrightarrow x = \mu - \frac{1}{\alpha} \ln(2 - 2u)$$

$$x < \mu : u = \frac{1}{2} e^{\alpha(x-\mu)} \Leftrightarrow \ln(2u) = \alpha(x - \mu) \Leftrightarrow x = \mu + \frac{1}{\alpha} \ln(2u)$$

$$\therefore F_x^{-1}(u) = \mu - \frac{1}{\alpha} sgn(u - \frac{1}{2}) \ln(1 - 2|u - \frac{1}{2}|)$$

Simpler solution: Since we know DE(0,1)

$$DE(0, 1) = \frac{1}{2} exp(-|x|)$$

$$f(x) = \begin{cases} \frac{1}{2}exp(-x), & \text{if } x \geq \mu. \\ \frac{1}{2}exp(x), & \text{if } x < \mu. \end{cases}$$

$$F_{x<\mu}(x) = \int_{-\infty}^{x} f(u)du = \int_{-\infty}^{x} \frac{1}{2}e^u du = \frac{1}{2}\left[e^u\right]_{-\infty}^{x} = \frac{1}{2}(e^x - e^{-\infty}) = \frac{1}{2}e^x$$

$$F_{x\geq\mu}(x) = \int_{-\infty}^{x} f(u)du = \int_{-\infty}^{0} f(u)du + \int_{0}^{x} f(u)du = \int_{-\infty}^{0} \frac{1}{2}e^u du + \int_{0}^{x} \frac{1}{2}e^{-u} du$$

$$\frac{1}{2}\left[e^u\right]_{-\infty}^{0} + \frac{1}{2}\left[-e^{-u}\right]_{0}^{x} = \frac{1}{2}\left(e^0 - e^{-\infty}\right) + \frac{1}{2}\left(-e^{-x} + e^0\right) = \frac{1}{2}(1-0) + \frac{1}{2}(-e^{-x}+1) = 1 - \frac{1}{2}e^{-x}$$

$$F(x) = \begin{cases} \frac{1}{2}exp(x), & \text{if } x < 0. \\ 1 - frac12exp(-x), & \text{if } x \geq 0. \end{cases}$$

So inverse when x < 0

$$y = \frac{1}{2}e^x \Rightarrow 2y = e^x \Rightarrow ln(2y) = x$$

So inverse when $x \geq 0$

$$y = 1 - \frac{1}{2}e^{-x} \Rightarrow y + \frac{1}{2}e^{-x} = 1 \Rightarrow \frac{1}{2}e^{-x} = 1 - y \Rightarrow e^{-x} = 2 - 2y \Rightarrow -x = ln(2-2y) \Rightarrow x = -ln(2-2y)$$

```
invDE <- function(mu, alpha, U){
    x <- mu - 1/alpha * sign(U - 0.5) * log(1 - 2 *abs(U - 0.5))
    return(x)
}

nr <- sort(runif(10000))
res <- sapply(nr, FUN = function(i){invDE(0, 1, i)})
# plot(res)
# hist(res, breaks = 100, freq = FALSE, main = "Histogram of generated Double Exponential (Laplace) Dis
```

L3-Q2-2. Acceptance / Rejection methods

Question: Use the Acceptance/rejection method with DE(0,1) as a majorizing density to generate N(0,1) variables. Explain step by step how this was done. How did you choose constant c in this method? Generate 2000 random numbers N(0,1) using your code and plot the histogram. Compute the average rejection rate R in the acceptance/rejection procedure. What is the expected rejection rate ER and how close is it to R? Generate 2000 numbers from N(0, 1) using standard rnorm() procedure, plot the histogram and compare the obtained two histograms.

The purpose of using the *Acceptance /Rejection method*, is that we have our unknown distribution (*target density*, $p_X()$) given from our data which we want to generate new numbers from.

To build a random number generator out of this distribution, we utilize an known similar PDF-distribution (*majorizing densiity*, $g_Y()$) that is as similar as possible to the target density, with the only constrain that the majorizing density can be scaled by a constant, giving the *majorizing density or proposal density function*, $cg_Y()$.

Once we have the majorizing density and scaled it to cover the entire target density, we can:

1. Generate y from our majorizing density function $g_Y()$

2. Generate u from a unif(0,1)

If $u \leq \frac{p_X(y)}{cg_Y(y)}$ then y is within the target density, if condition doesn't hold, start over generating a new number.

One of the strengths with acceptance / rejection methods is that is scales to higher dimensions (if the majorizing distribution also is multivariate), however as the dimensions increases it suffers from the curse of dimensionality.

Below, the double exponential distribution (Laplace distribution) and the Gaussian distribution is defined.

```
DE <- function(mu, alpha, x){
    res <- alpha/2 * exp(-alpha * abs(x - mu))
    return(res)
}

gaussian <- function(mu, sigma, x){
    res <- (1/sqrt(2*pi*sigma^2))*exp(-(x-mu)^2/2*sigma^2)
    return(res)
}

# gaussian(0,1,-5:5) / DE(0,1, -5:5)
```

```
comp_c <- function(x) {
    return(sqrt(2/pi) * exp(abs(x) - 0.5 * x^2))
}
```

```
library(tidyverse)
library(ggplot2)

gg_data <- data.frame(interval = seq(-5,5, 0.005)[2:2001],
                      DE01 = NA,
                      normal01 = NA,
                      cDE01 = NA)
c <- comp_c(1)
gg_data <- gg_data %>%
    dplyr::mutate(DE01 = DE(0,1, gg_data$interval),
           normal01 =gaussian(0,1, gg_data$interval),
           cDE01 = DE01 * c) %>%
    gather(variable, value, -interval)

# ggplot(data = gg_data) +
#     geom_line(aes(x = interval, y = value, color = variable)) +
#     labs(title = paste("Double exponential PDF as majorizing with c =",
#                        c, "for standard Normal PDF", sep = " "), x = "Range") +
#     guides(color = guide_legend(title = "PDF")) +
#     theme_minimal()
```

In the above graph, we can see a plot of the DE(0,1), $\mathcal{N}(0,1)$ and the scaled cDE(0,1), which is scaled by a factor of 1.315489. We chose c such that our majorizing distribution completely covers the theoretical $\mathcal{N}(0,1)$ distribution, our target distribution. We obtain c by deriving the expression

$$c \geq \frac{f_X(x)}{f_Y(x)} = \frac{\mathcal{N}(0,1)}{DE(0,1)} = \frac{\sqrt{2\pi}^{-1} e^{-0.5x^2}}{0.5e^{-|x|}} = \sqrt{\frac{2}{\pi}} e^{|x|-0.5x^2} = g(x)$$

Since $x \in (-1,1)$ we obtain c by taking the limiting case $c \geq g(1) = 1.315489$. Note, $f_X(x)$ is target density and $f_Y(x)$ is majorizing density we're drawing from.

```r
fgennorm<-function(c){
    # browser()
  x <- NA
  num_reject <- 0
  while (is.na(x)){
    y <- invDE(0,1, runif(1))
    u <- runif(1)
      if (u <= dnorm(y, 0, 1) / (c/2*exp(-abs(y)))){
        x <- y
    }   else{
      num_reject <- num_reject + 1
    }
  }
  c(x, num_reject)
}

y <- dnorm(seq(-5,5, 0.005)[2:2001], 0,1)

mnorm1<-sapply(rep(c,2000),fgennorm)

# hist(mnorm1[2,],col="black",breaks=100,xlab="",freq=TRUE,main="")
rej_rate <- mean(mnorm1[2,])

p <- length(mnorm1[2,][mnorm1[2,] == 0]) / length(mnorm1[2,])

ER <- (1 - p) / p
```

Our average rejection rate $\bar{R} = 0.3345$. Since R is distributed $R \sim Geo(k, p)$ where k is the number of trials and p is the success rate, such that

$$p = \frac{\text{Number of times we don't reject}}{\text{Total amount of trials}}$$

. This gives $E[R] = \frac{1-p}{p}$ this calculation yields $E[R] \simeq 0.345895$, where $p \simeq 0.743$. To note, we conjecture that R is assumed geometrically distributed, since it's a distribution of trials and errors.

We note that there is a slight difference between the average and the expected value of $|\bar{R} - E[R]| = 0.011395$.

```r
# par(mfrow = c(1,2))
# hist(mnorm1[1,],col="black",breaks=100,xlab="",freq=FALSE,main="Acceptance / Rejection method", ylim=
# points(seq(-5,5, 0.005)[2:2001],y,pch=19,cex=0.3,col="red")
#
# hist(rnorm(seq(-5,5,0.05)[2:2001],0,1), breaks = 100, freq = FALSE, main ="r-norm", xlab = "", col =
# points(seq(-5,5, 0.005)[2:2001],y,pch=19,cex=0.3,col="red")
```

When comparing the two histograms of random numbers generated from our acceptance/rejection method and the *rnorm()* function we note that they share similar distribution, with slight differences. The mean and variance looks about the same for both distributions.

From seminar: inverse CDF is a better method if we have the possibility of choosing one, since we have an exact formula providing the distribution.

Lab4-Question 1: Computations with Metropolis-Hastings

We are given:

$$f(x) \propto x^5 e^{-x}, \text{ if } x > 0$$

L4-Q1-1. Metropolis-Hastings log-normal and Burn-in period

Question: Use Metropolis-Hastings algorithm to generate samples from this distribution by using proposal distribution as log-normal $LN(X_t, 1)$, 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?

```
set.seed(123456)
f <- function(x) {
  return(x^5 * exp(-x))
}

metro_hast_lnorm <- function(tmax, start) {
  t <- 1
  X <- c()
  X[t] <- start

  while(t < tmax) {
    Y <- rlnorm(1,X[t],1)
    U <- runif(1,0,1)

    alpha <- min(1, f(Y)*dlnorm(X[t], Y, 1) / (f(X[t]) * dlnorm(Y,X[t],1)))
    #browser()
    if(U < alpha) {
      X[t+1] <- Y
    } else {
      X[t+1] <- X[t]
    }
    t <- t + 1
  }
  return(X)
}

mh_lnorm <- metro_hast_lnorm(tmax = 1000,1)
# plot.ts(mh_lnorm, main = "log-Normal", ylab = "")
```

The acceptance probability is:

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

where $\pi(x)$ is the PDF we want to sample from, $q(\cdot|X_t)$ is a *proposal distribution* that has a regular from with respect to the PDF. It consists of two fractions, where the first is the ratio for the density for proposed target value vs the current value $\frac{\pi(Y)}{\pi(X_t)}$ and ratio of proposal densities value, where the denominator is the probability of drawing the value we actually did given the current state and the nominator is how likely would the current value have been if we were at the proposed value $\frac{q(X_t|Y)}{q(Y|X_t)}$.

We have implemented the Metropolis-Hastings algorithm with start at point 1.

Based from visual inspection, the chain seem to converge quick. After approximately 550 iterations, the chain seem to have found the range it oscillates inbetween.

The chain has a short burn-in period. We've would consider the burn-in period to end after approximately 10 iterations. This could be explained by the fact that the probability density function is non-complex.

L4-Q1-2. Metropolis-Hastings chi-square

Question: Perform Step 1 by using the chi–square distribution $\chi^2(\lfloor X_t + 1 \rfloor)$ as a proposal distribution, where $\lfloor x \rfloor$ is the floor function, meaning the integer part of x for positive x, i.e. $\lfloor 2.95 \rfloor = 2$

```r
set.seed(123456)
metro_hast_chi <- function(tmax, start=1) {
  t <- 1
  X <- c()
  X[t] <- start

  while(t < tmax) {
    Y <- rchisq(1,1)
    U <- runif(1,0,1)

    alpha <- min(1, f(Y)*dchisq(floor(X[t]+1), 1) / (f(X[t]) * dchisq(floor(Y+1), 1)))
    #browser()
    if(U < alpha) {
      X[t+1] <- Y
    } else {
      X[t+1] <- X[t]
    }
    t <- t + 1
  }
  return(X)
}
ml_chi <- metro_hast_chi(tmax = 1000)

# par(mfrow = c(1,2))
# plot.ts(mh_lnorm, main = "log-Norm", ylab ="", ylim = c(0, 11))
# plot.ts(ml_chi, main = "Chi-square", ylab ="", ylim = c(0, 11))
```

L4-Q1-3. Comparing distributions Metropolis-Hastings

Question: Compare the results of Steps 1 and 2 and make conclusions.

Both chains seem to be stationary by visual inspection, since the chains don't have slope and continually oscillates within the same ranges (for each proposal density).

The log-normal proposal distribution have a lower mean than the Chi-sqare chain, where they should be the same if the function would approximate $f(x)$ equally well. The log-Norm also have longer horizontal stretches, which is something unwanted. The interpretation of that is that the number that is being drawn is worse than the one we already got, and therefore we prefer the value we have from the earlier iteration. This implies that we have dependencies in our chain which also unwanted.

The χ^2 chain generates more unique values, which is prefered when using the generated data for further analysis.

L4-Q1-4. Analyze convergence - Gelman-Rubin

Generate 10 MCMC sequences using the generator from Step 2 and starting points 1, 2, . . . , or 10. Use the Gelman–Rubin method to analyze convergence of these sequences.

```
n <- 1000
chains <- matrix(nrow=1000,ncol=10)

for(k in 1:10) {
  chains[,k] <- metro_hast_chi(tmax = 1000, k)
}

v <- colMeans(chains)
B <- n / 9 * sum((v - mean(v))^2)

s <- c()
for(k in 1:10) {
  s[k] <- sum((chains[,k] - v[k])^2) / (n - 1)
}

W <- sum(s)/10
Varv <- (n-1)/n * W + (1/n) * B
Rsqrt <- sqrt(Varv / W)
# Rsqrt
```

A value close to 1 implies convergence, our Gelman-Rubin value is 1.025316, which indicates convergence.

L4-Q1-5. Expected value

Question: Estimate $\int_0^\infty xf(x)dx$ using samples from steps 1 and 2.

To estimate the integral, we decompose it into: $f(x) = g(x)p(x)$ where $g(x) = x$ and $f(x) = p(x)$ and $\int_0^\infty p(x)dx = 1$. Then

$$E[g(x)] = \int_0^\infty g(x)p(x)dx$$

which gives us

$$E[g(x)] \simeq \frac{1}{n}\sum_{i=1}^{n} g(x_i)$$

```
ch_mean <- mean(metro_hast_chi(tmax=1000,1))
```

```
lnorm_mean <- mean(metro_hast_lnorm(tmax=1000,1))
```

The mean for the log-normal is 3.286928 and the mean for the χ^2 is 5.707418.

L4-Q1-6. The distribution generated is in fact a gamma distribution. Look in the literature and define the actual value of the integral. Compare it with the one you obtained.

The Gamma probability density function is:

$$\Gamma(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$$

When comparing it to $f(x) \propto x^5 e^{-x}$, we can see that $\alpha = 6$ and $\beta = 1$. The expected value from a Γ-distribution is

$$E(x) = \frac{\alpha}{\beta} = \frac{6}{1} = 6,$$

which is approximately the same value we receieved from our chi-squared MCMC. The chi-squared MCMC performs better, because it better samples $f(x)$.

Lab4-2. Gibbs Sampling

A concentration of a certain chemical was measured in a water sample, and the result was stored in the data **chemical.RData** having the following variables:

- X: day of the measurement
- Y: measured concentration of the chemical

The instrument used to measure the concentration had certain accuracy; this is why the measurements can be treated as noisy. Your purpose is to restore the expected concentration values.

L4-Q2-1. Import data

Question: Import the data to **R** and plot the dependence of Y on X. What kind of model is reasonable to use here?

```
require(ggplot2)
load("~/Google Drive/Statistics and Data Mining/732A90 Computational Statistics/CS_Lab4/chemical.RData")
# plot(X, Y)
```

The data seem to have a linear trend the first 30 observations, then it seems like the values stabilize and the linear increase stops. It might be reasonable to believe that the first 30 observations was needed to calibrate the instrument, and first 30 observations should be considered as a burn-in period for a Gibbs sampler method.

L4-Q2-2. Bayesian Random Walk

Question: A researcher has decided to use the following (random-walk) Bayesian model (n=number of observations, $\vec{\mu} = (\mu_1, ..., \mu_n)$ are unknown parameters):

$$Y_i \sim \mathcal{N}(\mu, \text{variance} = 0.2),\ i = 1, ..., n$$

where the prior is

$$p(\mu_1) = 1$$
$$p(\mu_{i+1}|\mu_i) = \mathcal{N}(\mu_i, 0.2),\ i = 1, ..., n$$

Present the formulae showing the likelihood $p(\vec{Y}|\vec{\mu})$ and the prior. **Hint:** a chain rule can be used here $p(\vec{\mu}) = p(\mu_1)p(\mu_2|\mu_1)p(\mu_3|\mu_2)...p(\mu_n|\mu_{n-1})$

Normal log-likelihood of the function is:

$$p(\vec{y}, |\vec{\mu}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{1}{2\sigma^2}(y_i - \mu_i)^2\right) = (2\pi\sigma^2)^{-n/2} exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu_i)^2\right)$$

The prior of $\vec{\mu}$ is:

$$p(\vec{\mu}) = p(\mu_1)p(\mu_2|\mu_1)p(\mu_3|\mu_2)...p(\mu_n|\mu_{n-1}) = p(\mu_1)\prod_{i=2}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{1}{2\sigma^2}(\mu_i - \mu_{i-1})^2\right)$$

L4-Q2-3. Bayesian mega equation

Question: Use Bayes' Theorem to get the posterior up to a constant proportionality, and then find out the distributions of $(\mu_i|\vec{\mu_{-i}}, \vec{Y})$, where $\vec{\mu_{-i}}$ is a vector containing all μ values except of μ_i.

Hint A: consider for separate formulae for $(\mu_1|\vec{\mu_{-1}}, \vec{Y})$, $(\mu_n|\vec{\mu_{-n}}, \vec{Y})$ and then a formula for all remaining $(\mu_i|\vec{\mu_{-i}}, \vec{Y})$

Hint B:

$$exp\left(-\frac{1}{d}\left((x-a)^2+(x-b)^2\right)\right) \propto exp\left(-\frac{(x-(a+b)/2)^2}{d/2}\right)$$

Hint C:

$$exp\left(-\frac{1}{d}\left((x-a)^2+(x-b)^2+(x-c)^2\right)\right) \propto exp\left(\frac{(x-(a+b+c)/3)^2}{d/3}\right)$$

Start of proof:

$$p(\mu_i|\vec{\mu}_{-i}, \vec{Y}) = \frac{p(\vec{\mu}_{-i}, \vec{Y}|\mu_i)p(\mu_i)}{p(\vec{\mu}_{-i}, \vec{Y})} = \frac{p(\vec{\mu}_{-i}|\mu_i)p(\vec{Y}|\mu_i)p(\mu_i)}{p(\vec{\mu}_{-i}, \vec{Y})} = (*)$$

where

$$p(\vec{\mu}_{-i}|\mu_i) = /\text{Extended Markov property}/ = p(\mu_n, ..., \mu_{i+1}|\mu_i)$$
$$= p(\mu_n|\mu_i)p(\mu_{n-1}|\mu_i)\dots p(\mu_{i+1}|\mu_i) = p(\mu_n|\mu_{n-1}, ..., \mu_i)p(\mu_{n-1}|\mu_{n-2}, ..., \mu_i)\dots p(\mu_{i+1}|\mu_i)$$
$$= /\text{Markov Property}/ = p(\mu_n|\mu_{n-1})\dots p(\mu_{i+1}|\mu_i) = \prod_{j=i+1}^{n} p(\mu_j|\mu_{j-1})$$

and

$$p(\mu_i) = p(\mu_i|\mu_{i-1})p(\mu_{i-1}) = \left(\prod_{j=2}^{i} p(\mu_j|\mu_{j-1})\right)p(\mu_1) = p(\vec{\mu})$$

Concluding

$$(*) = \frac{p(\vec{Y}|\mu_i)\left(\prod_{j=i+1}^{n} p(\mu_j|\mu_{j-1})\right)\left(\prod_{j=2}^{i} p(\mu_j|\mu_{j-1})\right)p(\mu_1)}{p(\vec{\mu}_{-i}, \vec{Y})}$$

$$= \prod_{j=i+1}^{n} p(\mu_j|\mu_{j-1}) \prod_{j=2}^{i} p(\mu_j|\mu_{j-1})p(\mu_1) = \frac{p(\vec{Y}|\mu_i)p(\vec{\mu})}{p(\vec{\mu}_{-i}, \vec{Y})} = \frac{p(Y_1)\dots p(Y_i|\mu_i)\dots p(Y_n)p(\vec{\mu})}{p(\vec{\mu}_{-i}, \vec{Y})}$$

$$= \frac{p(Y_1)\dots p(Y_i|\mu_i)\dots p(Y_n)p(\mu_1)\dots p(\mu_i|\mu_{i-1})p(\mu_{i+1}|\mu_i)\dots p(\mu_n|\mu_{n-1})}{p(\vec{\mu}_{-i}, \vec{Y})}$$

$$\propto p(Y_i|\mu_i)p(\mu_i|\mu_{i-1})p(\mu_{i+1}|\mu_i) = exp\left[-\frac{1}{2\sigma^2}\left((y_i-\mu_i)^2+(\mu_i-\mu_{i-1})^2+(\mu_{i+1}-\mu_i)^2\right)\right]$$

$$= exp\left[-\frac{1}{2\sigma^2}((\mu_i-y_i)^2+(\mu_i-\mu_{i-1})^2+(\mu_i-\mu_{i+1})^2)\right] \propto exp\left[-\frac{3}{2\sigma^2}\left(\mu_i-\frac{(y_i+\mu_{i+1}+\mu_{i-1})}{3}\right)^2\right]$$

$$\therefore p(\mu_i|\vec{\mu}_{-i}, \vec{Y}) \simeq \mathcal{N}\left(\frac{y_i+\mu_{i+1}+\mu_{i-1}}{3}, \frac{\sigma^2}{3}\right)$$

L4-Q2-4. Gibbs sampler

Question: Use the distributions derived in Step 3 to implement a Gibbs sampler that uses $\vec{\mu^0} = (0, ..., 0$ as a starting point. Run the Gibbs sampler to obtain 1000 values of $\vec{\mu}$ and then compute the expected value of $\vec{\mu}$ by using a Monte Carlo approach. Plot the expected value of $\vec{\mu}$ versus X and Y versus X in the same graph. Does it seem that you have managed to remove the noise? Does it seem that the expected value of $\vec{\mu}$ can catch the true underlying dependence between Y and X?

```
gibbz<-function(Y){
  n <- length(Y)
  mu0 <- rep(0,n)
  t <- 1
  tmax <- 1000
  mu <- matrix(NA,ncol=1001,nrow=n)
  mu[,1] <- mu0
  sigma2 <- 0.2

  while(t <= tmax){
    mu[1,t+1] <- rnorm(1, mean=((Y[1]+mu[2,t])/2), sd=sqrt(sigma2/2))
    for(i in 2:(n-1)){
      mu[i,t+1] <- rnorm(1, mean=((Y[i]+mu[i+1,t]+mu[i-1,t+1])/3), sd=sqrt(sigma2/3))
    }
    mu[n,t+1] <- rnorm(1, mean=((Y[n]+mu[n-1,t+1])/2), sd=sqrt(sigma2/2))
    t <- t+1
  }

  return(mu)
}

res <- gibbz(Y)
# ggplot() +
#   geom_point(aes(X, rowMeans(res), col="red")) +
#   geom_point(aes(X,Y, col="black")) +
#   xlab("X") + ylab("Y") + ggtitle("Gibbs Sampler vs. Collected Data") +
#   theme_minimal() +
#   scale_color_manual(name="Source:", values=c("black", "red"),
#                      labels = c("Original","Gibbs"))
```

Yes, looking at the plot above, the Gibbs generated data points have reduced noise compared to the original data. The conclusion is that we've successfully generated data using Gibbs sampling which seems to follow the underlying dependence between X and Y.

L4-Q2-5. Trace plot

Question: Make a trace plot for μ_n and comment on the burn-in period and convergence.

```
# plot.ts(res[50,])
```

Looking the above plot, the burn-in period is super short. After aprroximately 5 observations, the Gibbs sampler seems to have ended it's burn-in period and oscillates around the mean value. The conclusion is that the Gibbs sampler is very effective producing a random sample that replicates the distribution of the original data.

The trace plot consists of all the columns from sampled μ_n.

Lab5-Question 1: Hypothesis testing

L5-Q1-1.Import data - Check randomness

Question: Make a scatterplot of Y versus X and conclude whether the lottery looks random.

```
df <- read.csv2("~/Google Drive/Statistics and Data Mining/732A90 Computational Statistics/CS_Lab5/lotte


library(ggplot2)
# ggplot(data = df) +
#     geom_point(aes(x = Day_of_year, y = Draft_No)) +
#     labs(title = "Day of Year vs Draft Number", x = "Day of Year", y = "Draft Number") +
#     theme_minimal()
```

While visually inspecting the plot we can see that the data points seem equally distributed all over the sample space, which is an indication that the lottery is random.

L5-Q1-2. Apply loess smoother

Question: Compute an estimate $\hat{Y}$ of the expected response as a function of X by using a loess smoother (use loess()), put the curve $\hat{Y}$ versus X in the previous graph and state again whether the lottery looks random.

```
mod <- loess(Draft_No ~ Day_of_year, df)

df$y_hat <- predict(mod, newdata = df$Day_of_year)

# ggplot(data = df) +
#     geom_point(aes(x = Day_of_year, y = Draft_No)) +
#     geom_line(aes(x = Day_of_year, y = y_hat, color = "red"), size = 2, show.legend = FALSE) +
#     labs(title = "Day of Year vs Draft Number smoothed by Loess", x = "Day of Year", y = "Draft Numbe
#     theme_minimal()
```

After applying the loess smoother to Y, we can by visual inspection see that there seems to be an underlying trend in the data. The smoothed draft number peaks after approximately 90 days (end of march), and then decreases throughout the rest of the year. Since the curve is not univariate, it seems like there is a trend in selecting people born in the earlier part of the year.

L5-Q1-3. Test randomness - Non-parametric Bootstrap

Question: To check whether the lottery is random, it is reasonable to use test statistics

$$T = \frac{\hat{Y}(X_b) - \hat{Y}(X_a)}{X_b - X_a},$$

where $X_b = argmax_X Y(X)$, $X_a = argmin_X Y(X)$. If this value is significantly greater than zero, then there should be a trend in the data and the lottery is not random. Estimate the distribution of T by using a non-parametric bootstrap with B = 2000 and comment whether the lottery is random or not. What is the p-value of the test?

```
T_test <- function(y_hat, samp){
    ind_a <- which.min(samp$Draft_No)
    ind_b <- which.max(samp$Draft_No)
    Xa <- samp$Day_of_year[ind_a]
```

```
    Xb <- samp$Day_of_year[ind_b]
    Y_Xa <- y_hat[ind_a]
    Y_Xb <- y_hat[ind_b]
    # browser()
    T_test <- (Y_Xb-Y_Xa)/(Xb-Xa)
    return(T_test)
}
# T_test(df$y_hat, df)
```

```
set.seed(123456)
booty <- function(X, B) {
  D <- 1:nrow(X)
  n = length(D)
  D_star <- matrix(rep(NA, B*n), nrow=n, ncol = B)
  for(i in 1:B) {
    D_star[,i] <- sample(D, n, replace=TRUE)
  }

  res <<- apply(D_star, 2, function(i) {
      mod <- loess(Draft_No ~ Day_of_year, X[i,])
      T_test(predict(mod, newdata = X[i,]),
             X[i,])
      })
  # browser()
  # hist(res, breaks=20, main="Bootstrap test-statistics", xlab="Test-statistic")
}
booty(df, 2000)
T_test_pvalue <- mean(res <= 0)
```

The p-value of the test statistica is computed by:

$$\hat{p} = \frac{\#[T(X_{gb*}) \geq T(X)]}{B} = 0.9305$$

Since $\hat{p} = 0.9305 > 0.05$ we fail to reject H_0 and conclude that the lottery is random on 5%-significance.

L5-Q1-4. Permutation test

Question: Implement a function depending on data and B that tests the hypothesis H_0 : Lottery is random versus H_1 : Lottery is non-random by using a permutation test with statistics T. The function is to return the p-value of this test. Test this function on our data with B = 2000.

```
hyptest <- function(dat, B) {
  stat <- numeric(B)
  n = nrow(dat)
  T_x <- T_test(dat$y_hat, dat)
  for(b in 1:B) {
    #browser()
    Gb <- sample(1:n, n, replace = FALSE)
    new_data <- dat
    new_data$Day_of_year <- dat$Day_of_year[Gb]
    mod <- loess(Draft_No ~ Day_of_year, new_data)

    stat[b] <- T_test(mod$fitted, new_data)
```

```
  }
  res <- list(p = mean(abs(stat) >= abs(T_x)), stat=stat)
  return(res)
}

res <- hyptest(df,2000)
res_pval <- res$p
# res_pval

qu2_5<-quantile(res$stat,probs = 0.025)
qu97_5<-quantile(res$stat,probs = 0.975)

# hist(res$stat, breaks = 35, main="Bootstrap Hypothesis Test", xlab = "Test-statistic")
# abline(v=qu2_5, col = "red", lty = 2)
# abline(v=qu97_5, col = "red", lty = 2)
```

When using permutation test, Since $\hat{p} = 0.093 > 0.05$ we do not reject H_0 in favor of H_1 and conclude that the lottery is random on 5% significance. We fail to reduce H_0 in favor for H_1

If you add abs() around a test statistic, it means it's two-sided.

L5-Q1-5. Power of test

Question: Make a crude estimate of the power of the test constructed in Step 4:

5a.

Question: Generate (an obviously non–random) dataset with n = 366 observations by using same X as in the original data set and $Y(x) = max(0, min(\alpha x + \beta, 366))$, where $\alpha = 0.1$ and $\beta \sim N(183, sd = 10)$.

```
df2 <- data.frame(Day_of_year = df$Day_of_year)

df2$Draft_No <- sapply(df2$Day_of_year, function(x) {
  beta <- rnorm(n = 1, 183, 10)
  return(max(0, min(0.1 * x + beta, 366)))
})
```

5b.

Question: Plug these data into the permutation test with B = 200 and note whether it was rejected.

```
df2$y_hat = predict(mod, newdata=df2)
hyptest(df2, 200)$p
```

```
## [1] 0
```

We have noted that H_0 was rejected once again. This is as expected since we know that the data we've generated is not random.

5c.

Question: Repeat Steps 5a–5b for $\alpha = 0.2, 0.3, ..., 10$. What can you say about the quality of your test statistics considering the value of the power?

```r
df3 <- data.frame(Day_of_year = df$Day_of_year)
pvals <- c()
for(alpha in seq(0.2,10,by=0.1)) {

  df3$Draft_No <- sapply(df3$Day_of_year, function(x) {
    beta <- rnorm(n = 1, 183, 10)
    return(max(0, min(alpha * x + beta, 366)))
  })

  df3$y_hat = predict(mod, newdata=df3)
  pvals <- c(pvals,hyptest(df3, 200)$p)

}
out <- data.frame(X= seq(0.2,10,by=0.1), Y=1-pvals)

# ggplot(data = out) +
#   geom_point(aes(out$X,out$Y)) +
#   geom_smooth(method="loess", aes(X,Y)) +
#   labs(x="Alpha", y="Power", title="Alpha vs. Power") +
#   theme_minimal()
```

All the P-values from the test-statistics are significant, they produce $P_i << 0.05$ which implies that we reject the Null Hypothesis, the data is not random, on each case.

The power of a statistical test is the probability that the test correctly rejects the null hypothesis (Power = 1 − Type-II-error. The Type-II error is when H_0 is false, but we fail to reject it. Since we know that the data produced is not random, we should produce no Type-II errors, which means that we have Power = 1. Thus every H_0 of the bootstrap samples are rejected.

The power of our test implies that we have a high probability of rejecting a false negative, however a consequence of that is that we have a high probability of rejecting a true H_0 (Type-I-error).

Lab5-Question 2: Bootstrap, jackknife and confidence intervals

L5-Q2-1. Import data estimate mean

Question: Plot the histogram of Price. Does it remind any conventional distribution? Compute the mean price.

```r
dat <- read.csv2("~/Google Drive/Statistics and Data Mining/732A90 Computational Statistics/CS_Lab5/pri

# ggplot(data = dat) +
#     geom_histogram(aes(Price), bins = 35) +
#     labs(title = "Histogram of Price", y = "Frequency", x = "Price") +
#     theme_minimal()
# mean(dat$Price)
```

The histogram of prices is not similar to any known distribution. The mean of price is 1080.472727.

L5-Q2-2. Bootstrap - Bias correction and variance

Question: 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 (Hint: use boot(),boot.ci(),plot.boot(),print.bootci())

```
stat_mean <- function(data,vn){
    data<-as.data.frame(data[vn,])
    return(mean(data$Price))
}

library(boot)
set.seed(123456)
res_mean <- boot(dat, stat_mean, R=1000)
# print(res_mean)
# plot(res_mean)

var_T <- 1/(res_mean$R - 1) * sum((res_mean$t - mean(res_mean$t))^2)
bias_T <- (2 * mean(dat$Price)) - mean(res_mean$t)

ci <- boot.ci(res_mean, conf = 0.95,
              type=c("norm", "perc", "bca"))
# print(ci)
```

By visually inspecting the histogram, t, where t* is each bootstraps mean value, the distribution of the bootstraps looks normal distributed. This is supported by the QQ-plot, where the residuals are approximately linear.

The bias corrected estimator and variance estimates is computed by:

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

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

The bias corrected estimator is 1078.776536 and the variance is 1307.33025.

The intervals for each bootstrap method is shown in the table below.

The three different methods for non-parametric bootstrapping is:

1. *Percentile* which removes 2.5 % of each tail and computes the bootstrap interval.
2. *BCa* interval calculated using adjusted percentile method.
3. *Normal* which uses an normal approximation when computing the bootstrap interval, normal also utilizes bias correction.

Non-parametric bootstraps work badly for small samples (n<40)

L5-Q2-3. Jackknife - Estimate variance

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

```
n <- min(res_mean$R, length(dat$Price))
X <- dat$Price
jacks <- sapply(1:n, function(i) {
  mean(X[-i])
})
```

```
jack_T <- n*mean(dat$Price) - (n-1)*jacks
jack_var <- 1/(n*(n - 1)) * sum((jack_T - mean(jacks))^2)
```

Jackknife overestimates variance and as we can see the variance for jackknife is 1320.911044 higher compared to the bootstrap variance which is 1307.33025.

You should not compute confidence intervals with jackknife method, since each jackknife bootstrap is too dependent.

L5-Q2-4. Compare Confidence intervals

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

```
lframe <- data.frame(
  Normal = c(ci$normal[2],ci$normal[3], ci$normal[3] - ci$normal[2], mean(dat$Price), var_T, bias_T),
  BCa = c(ci$bca[4], ci$bca[5], ci$bca[5] - ci$bca[4], mean(dat$Price), var_T, bias_T),
  Percentile = c(ci$percent[4], ci$percent[5], ci$percent[5] - ci$percent[4], mean(dat$Price), var_T, bi
)

rownames(lframe) <- c("CI_Min", "CI_Max","CI_Length", "Mean", "Variance_T", "Bias_Corr_Est")
# knitr::kable(t(lframe))
```

When comparing the intervals, we can see that the normal approximation have the narrowest interval and BCa have the widest. The normal interval has the lowest min and max values (that are within the confidence region), BCa have the highest confidence max value.

We utilize bootstrapping to increase the percision in our estimation of the variance and the confidence intervals. Since the bootstrap sample is the same for all three methods, the variance and bias corrected estimates is the same. When compunting the mean, the calculation only utilize the original data and therefore, the mean is also same for each method. # Lab6-Question 1: Genetic algorithm

L6-Q1-1. Define distribution function

Question: Define the function: $f(x) := \frac{x^2}{e^x} - 2exp(-(9sinx)/(x^2 + x + 1))$

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

L6-Q1-2. Define crossover function

Question: Define the function crossover(): for two scalars x and y it returns their "kid" as $(x + y)/2$

```
crossover <- function(x,y){
    res <- (x+y)/2
    return(res)
}
```

L6-Q1-3. Define mutate function

Define the function mutate() that for a scalar x returns the result fo the integer division $x^2 \text{mod} 30$.

```
mutate <- function(x, mod = 30){
    res <- x^2 %% mod
    return(res)
}
```

L6-Q1-4. Build genetic_algorithm

Question: Write a function that depends on the parameters maxiter and mutprob and:

1. Plots function f in the range from 0 to 30. Do you see any maximum value?
2. Defines an initial population for the genetic algorithm as $X = (0, 5, 10, 15, ..., 30)$
3. Computes vector Values that contains the function values for each population point.
4. Performs maxiter iterations where at each iteration

- i. Two indexes are randomly sampled from the current population, they are further used as parents (use sample()).
- ii. One index with the smallest objective function is selected from the current pop- ulation, the point is referred to as victim (use order()).
- iii. Parents are used to produce a new kid by crossover. Mutate this kid with prob- ability mutprob (use crossover(), mutate()).
- iv. The victim is replaced by the kid in the population and the vector Values is updated.
- v. The current maximal value of the objective function is saved.

5. Add the final observations to the current plot in another colour.

```
library(ggplot2)
genetic_algorithm <- function(maxiter, mutprob) {
    p <- ggplot()
    p <- p + geom_line(aes(seq(0,30,by=0.1), f(seq(0,30,by=0.1)), color="black"), color="black")

    X_init <- seq(0,30,by=5)
    X <- matrix(NA, length(X_init)*maxiter, ncol=maxiter, nrow=length(X_init))
    X[,1] <- X_init

    Values <- matrix(NA, length(X_init)*maxiter, ncol=maxiter, nrow=length(X_init))
    Values[,1] <- f(X_init)

    for(i in 2:maxiter) {
        X[,i] <- X[,i-1]
        Values[,i] <- f(X[,i])

        pars <- X[sample(1:length(X[,i]), 2),i]
        vict <- order(Values[,i])[1]
        kid <- crossover(pars[1], pars[2])

        if(rbinom(1, 1, mutprob) == TRUE) {
            X[vict,i] <- mutate(kid)
        } else {
            X[vict,i] <- kid
```

```
        }

        Values[,i] <- f(X[,i])
    }

    p <- p +
        geom_point(aes(X[,1], Values[,1], color="green"), color="green") +
        geom_point(aes(X[,maxiter],Values[,maxiter], color="red"), color="red") +
        labs(title = paste("Genetic Algorithm with maxiter =", maxiter, "and mutprob =", mutprob),
              x = "Index", y = "Value") +
        theme_minimal()

    return(list(plot = p,
                data = Values))
}

asd <- genetic_algorithm(100,0.5)
```

L6-Q1-5. Testing different parameters

Question: Run your code with different combinations of $maxiter = 10, 100$ and $mutprob = 0.1, 0.5, 0.9$. Observe the initial population and final population. Conclusions?

```
iter <- c(10, 100)
mutprobs <- c(.1, .5, .9)
l <- list()
k <- 1
for(i in iter){
    for(j in mutprobs){
        l[[k]] <- genetic_algorithm(maxiter = i, mutprob = j)$plot
        k <- k + 1
    }
}
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
# grid.arrange(grobs = l[1:3])
# grid.arrange(grobs = l[4:6])
```

Initially (black dotted line), we seem to have a small peak after 5 iterations.

The function replaces the lowest value in the population, by replacing the lowest value with the new kid in each iteration. The kid is generated by two randomize parents with crossover function and a probability of mutating.

As the probability of mutation increases we notice that the population in the final iteration tends to have higher values, a more optimized result. The same happens as the number of iterations increases, we notice that final values is higher compared to fewer iterations, which is intuitive since the algorithm have more iterations to optimize the population.

Lab6-Question 2: EM Algorithm

The data file physical.csv describes a behavior of two related physical processes Y = Y(X) and Z = Z(X).

```
df_em <- read.csv("~/Google Drive/Statistics and Data Mining/732A90 Computational Statistics/CS_Lab6/phy
```

L6-Q2-1. Make a time series plot describing dependence of Z and Y versus X. Does it seem that two processes are related to each other? What can you say about the variation of the response values with respect to X?

```
library(tidyr)
df_em$index <- 1:100
df_plot <- df_em %>%
    select(-X) %>%
    gather(variable, value, -index)

# ggplot(data = df_plot) +
#     geom_line(aes(x = index, y = value, color = variable)) +
#     theme_minimal()
```

While visually inspecting Y and Z, we notice that some peaks coincide in the plot, which would indicate that they are dependent. However, there is also peaks which does not coincide which would imply that they are not completely independent. Concludingly, there seems to be some dependence.

L6-Q2-2. Estimiate log-likelihood of exponential distribution

Question: Note that there are some missing values of Z in the data which implies problems in estimating models by maximum likelihood. Use the following model

$$Y_i \sim exp(X_i/\lambda),\ Z_i \sim exp(X_i/(2\lambda))$$

where λ is some unknown parameter. **The goal is to derive an EM algorithm that estimates** λ.

$$L(\lambda|Y,Z) = p(\vec{Y},\vec{Z}|\lambda) = p(\vec{Y}|\lambda)p(\vec{Z}|\lambda) = \prod_{i=1}^{n} p\left(Y_i|\frac{x_i}{\lambda}\right) p\left(Z_i|\frac{x_i}{2\lambda}\right) =$$

$$\prod_{i=1}^{n}\left(\frac{x_i}{\lambda}e^{-\frac{x_i}{\lambda}y_i}\frac{x_i}{2\lambda}e^{-\frac{x_i}{2\lambda}z_i}\right) = \prod_{i=1}^{n}\left(\frac{x_i^2}{2\lambda^2}e^{\frac{x_i}{\lambda}(-y_i+\frac{z_i}{2})}\right) = \frac{1}{(2\lambda^2)^n}\left(\prod_{i=1}^{n}x_i^2\right) exp\left[-\frac{1}{\lambda}\sum_{i=1}^{n}x_i\left(y_i+\frac{z}{2}\right)\right]$$

$$ln(L(\lambda)) = -n\,ln(2\lambda^2) + ln\left(\prod_{i=1}^{n}x_i^2\right) - \frac{1}{\lambda}\sum_{i=1}^{n}x_i\left(y_i+\frac{z_i}{2}\right)$$

$$E_{z^*}[ln\,L(\lambda)] = E_{z^*}\left[-n\,ln(2\lambda^2) + ln\left(\prod_{i=1}^{n}x_i^2\right) - \frac{1}{\lambda}\sum_{i=1}^{n}x_i\left(y_i+\frac{z_i}{2}\right)\right] =$$

$$-n\,ln(2\lambda^2) + ln\left(\prod_{i=1}^{n}x_i^2\right) - \frac{1}{\lambda}E_{z^*}\left[\sum_{i=1}^{n}x_i\left(y_i+\frac{z_i}{2}\right)\right] =$$

$$-n\,ln(2\lambda^2) + ln\left(\prod_{i=1}^{n} x_i^2\right) - \frac{1}{\lambda} E_{z^*}\left[\sum_{i=1}^{n} x_i y_i + \sum_{i=1}^{n} \frac{x_i z_i}{2}\right] =$$

$$-n\,ln(2\lambda^2) + ln\left(\prod_{i=1}^{n} x_i^2\right) - \frac{1}{\lambda}\left(\sum_{i=1}^{n} x_i y_i + \sum_{i\in obs.} \frac{x_i z_i}{2} + E_{z^*}\left[\sum_{i\in unobs.} \frac{x_i z_i}{2}\right]\right)$$

$$-n\,ln(2\lambda^2) + ln\left(\prod_{i=1}^{n} x_i^2\right) - \frac{1}{\lambda}\left(\sum_{i=1}^{n} x_i y_i + \sum_{i\in obs.} \frac{x_i z_i}{2} + \sum_{i\in unobs.} x_i E_{z^*}\left[\frac{z_i}{2}\right]\right) =$$

$$= -n\,ln(2\lambda^2) + ln\left(\prod_{i=1}^{n} x_i^2\right) - \frac{1}{\lambda}\left(\sum_{i=1}^{n} x_i y_i + \sum_{i\in obs} \frac{x_i z_i}{2}\right) - \frac{2(n-r)\lambda}{2\lambda}$$

since $\sum_{i\in unobs.} E_{z^*}\left[\frac{z_i}{2}\right] = 2(n-r)\lambda$.

Thus,

$$\frac{df}{d\lambda} = -\frac{4n\lambda}{2\lambda^2} + \frac{1}{\lambda^2}\left(\sum_{i=1}^{n} x_i y_i + \sum_{i\in obs} \frac{x_i z_i}{2}\right) + \frac{\lambda^k}{\lambda^2} = 0 \Rightarrow \lambda = \frac{\left(\sum_{i=1}^{n} x_i y_i + \sum_{i\in obs} \frac{x_i z_i}{2}\right) + (n-r)\lambda^k}{2n}$$

L6-Q2-3. Implement EM-exponential distribution

Question: Implement this algorithm in R, use $\lambda_0 = 100$ and convergence criterion "stop if the change in λ is less than 0.001". What is the optimal λ and how many iterations were required to compute it?

```
em_exp <- function(X, Y, Z, eps, kmax){
    Zobs <- Z[!is.na(Z)]

    n <- length(X)
    r <- length(Zobs)

    k <- 1
    lambdas <- c()
    lambdak <- 100
    lambdaprev <- 0

    while ((abs(lambdaprev - lambdak) > eps) && (k < (kmax+1))) {
        lambdaprev <- lambdak
        ## E-step
        EZ <- (sum(X*Y) + sum(X*Z, na.rm = TRUE)/2) + (n-r)*lambdak

        ## M-step
        lambdak <- EZ/(2*n)
        lambdas <- c(lambdas, lambdak)
        k <- k+1
    }
    return(list(lambda = lambdas,
                iterations = 1:(k-1)))
}

lambda <- em_exp(df_em$X, df_em$Y, df_em$Z, 10^(-3), 1000)
```

Iterations	Lambda
1	14.2678
2	10.8385
3	10.7014
4	10.6959
5	10.6957

The EM-algorithm finds the optimal lambda($\lambda = 10.695655$) after 5 iterations.

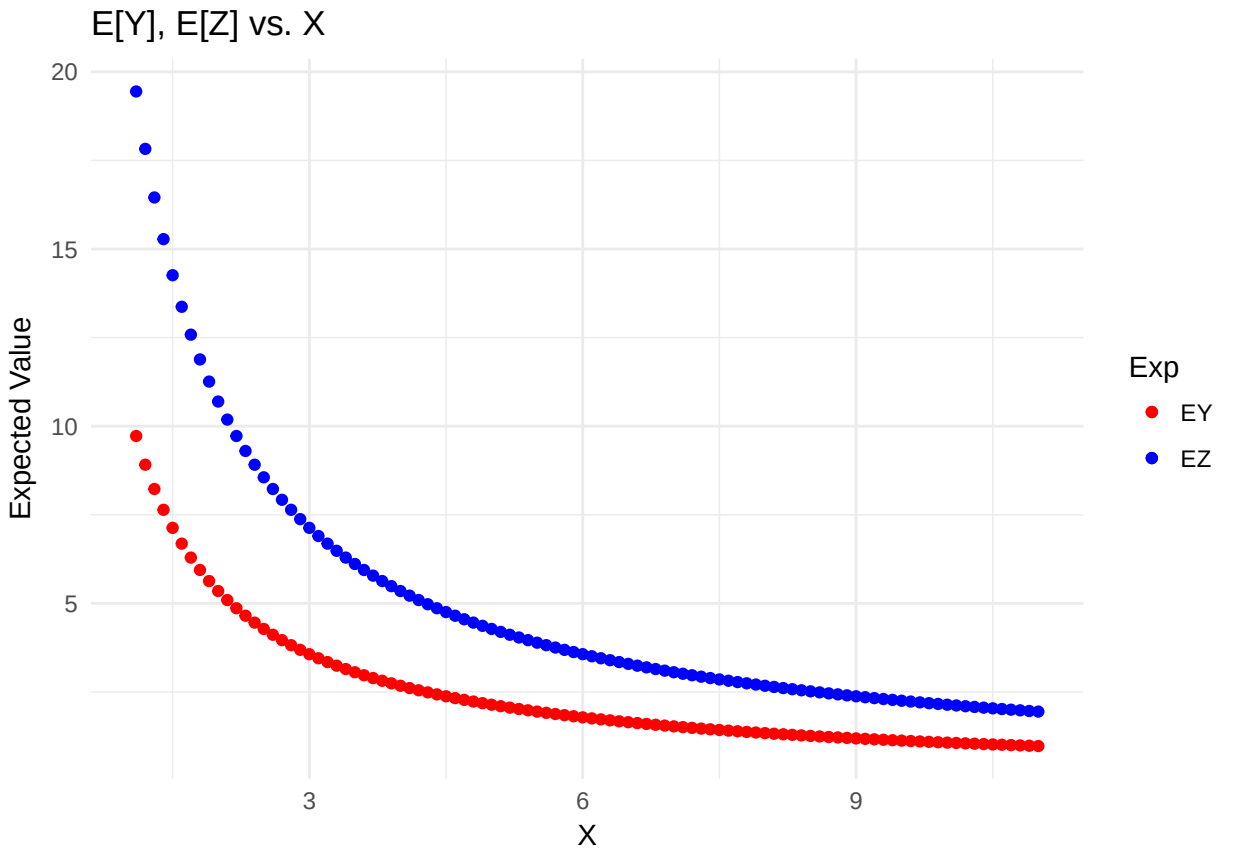
L6-Q2-4. Plot results

Question: Plot E[Y] and E[Z] versus X in the same plot as Y and Z versus X. Comment whether the computed λ seems to be reasonable.

$E[Y_i] = \frac{\lambda}{X_i}$ and $E[Z] = \frac{2\lambda}{X_i}$

```
EY <- lambda$lambda[length(lambda$lambda)] / df_em$X
EZ <- 2*lambda$lambda[length(lambda$lambda)] / df_em$X

ggplot() +
    geom_point(aes(x = df_em$X, y = EY, color="EY")) +
    geom_point(aes(x = df_em$X, y = EZ, color="EZ")) +
    labs(x="X", y="Expected Value", title="E[Y], E[Z] vs. X")+
    scale_colour_manual(name="Exp", values=c("red","blue")) +
    theme_minimal()
```

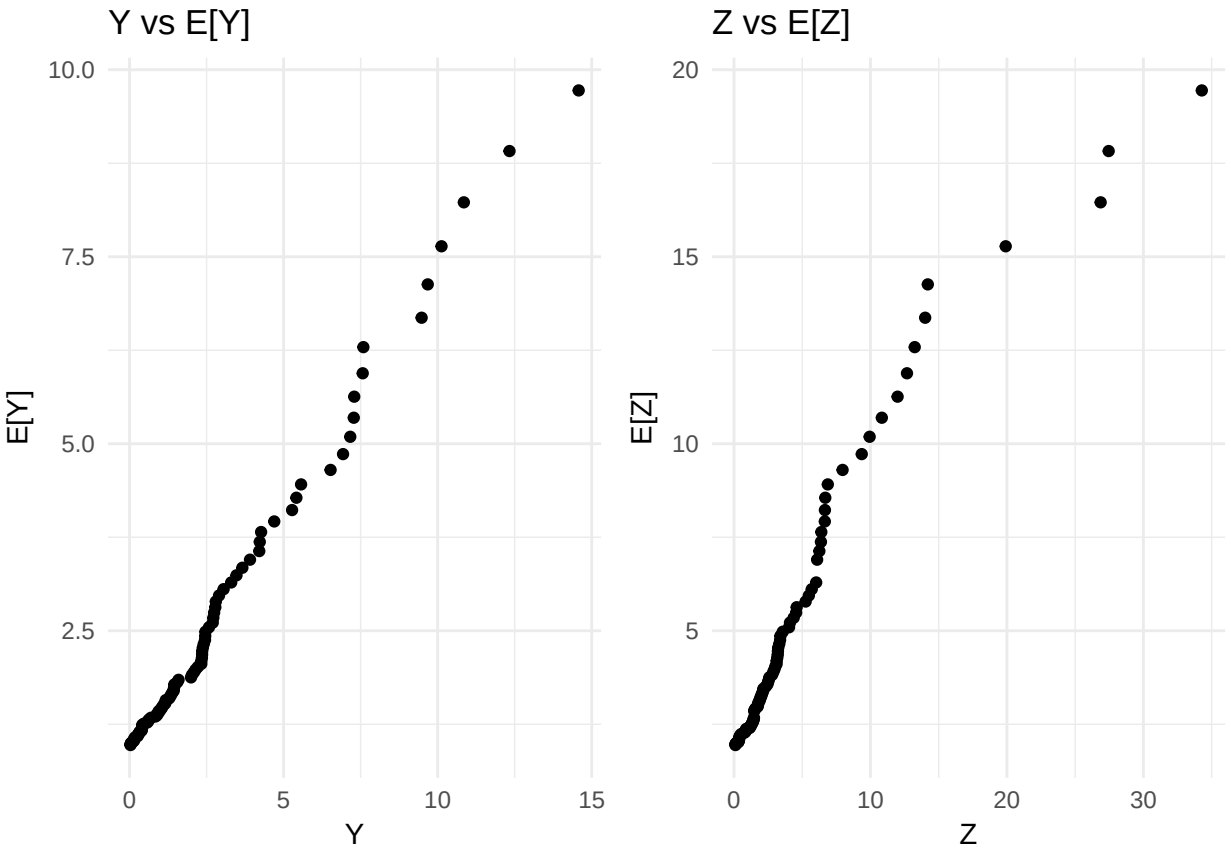


The above plot follows the exponential distribution nicely, which as we suspected. If we continoue to plot Y and Z towards each repective expected value, we notice that there seems to be an indication of linear dependence, which we interpret as the EM-algorithm as successful.

```
Y_EY <- ggplot() +
    geom_point(aes(sort(df_em$Y), sort(EY))) +
    labs(x = "Y", y = "E[Y]", title = "Y vs E[Y]") +
    theme_minimal()

ind <- which(!is.na(df_em$Z))
Z_EZ <- ggplot() +
    geom_point(aes(sort(df_em$Z[ind]), sort(EZ[ind]))) +
    labs(x = "Z", y = "E[Z]", title = "Z vs E[Z]") +
    theme_minimal()

mat <- rbind(c(1,2))
grid.arrange(Y_EY, Z_EZ, layout_matrix = mat)
```



KB - Generate Normal dist from Unif

Top row of output is $X_1 = \sqrt{-2\,ln\,D}\cos\theta$ and second row is $X_2 = \sqrt{-2\,ln\,D}\sin\theta$. I believe all values are Normal distribution, they are at least uncorrelated

```
fgenNorm <- function(N){
    mTD <- matrix(runif(2*N), ncol=N, nrow=2)
    mTD[1,] <- mTD[1,] * 2 * pi
```

```
    output <- apply(mTD, 2, function(TD){
        Th <- TD[1]
        D <- TD[2]
        a <- sqrt(-2*log(D))
        c(a*sin(Th), a*cos(Th))
    })
    return(as.vector(output)[1:N])
}
# fgenNorm(1)
```

KB - Generate Beta - Acceptance/Rejection method

First row of matrix is Beta distributed values. Second row is number of rejections. c-parameter is the constant the tells how much to scale in function I'm drawing numbers from to fit the beta distribution. It's important that C cover our target distribution, but the bigger C, the harder to generate significant values, so we want it small to keep it efficient.

```
fgenbeta<-function(c, alpha, beta){
    x<-NA
    num.reject <- 0
    while(is.na(x)){
    y <- runif(1)
    u <- runif(1)
    if(u <= dbeta(y, alpha, beta)/c){x <- y}
    else{num.reject <- num.reject + 1}
    }
    c(x, num.reject)
}
# c <- 2
# N <- 10000
# alpha <- 2
# beta <- 5
# hist(sapply(rep(c, N), fgenbeta, alpha, beta)[1,], breaks = 100)
```

KB/HK - Generate Exponential dist - Inverse CDF

$X \sim exp(\lambda)$ with pdf $f_X(x) = \lambda e^{-\lambda x}$ when $x \geq 0$

$F_X(x) = \int_0^x f_X(s)ds = 1 - e^{\lambda x}$, $x \geq 0$ integral goes from 0 to x and not $-\infty$ since exponential is not defined when x is smaller than 0.

F_X^{-1} is comupted by:

$$y = 1 - e^{-\lambda x}$$

$$e^{-\lambda x} = 1 - y$$

$$x = -\frac{1}{\lambda} ln(1 - y)$$

$$F_X^{-1}(y) = -\frac{1}{\lambda} ln(1 - y)$$

Hence, if $U \sim U(0, 1)$, then $-\frac{1}{\lambda} ln(1 - U) = X \sim exp(\lambda)$

```r
fgenexp <- function(lambda){
    res <- -1/lambda * log(1- runif(1, 0, 1))
    return(res)
}
# lambda <- 1
# N <- 1000
# hist(sapply(rep(lambda, N), fgenexp))
```

KB - Generate unif(0,n) from unif(0,1)

```r
fgenUnifn <- function(a, b){(a + runif(1) * (b-a))}
# fgenUnifn(0,10)
```

KB - Generate discrete Uniform from unif(0,1)

```r
fgenDescUnif <- function(n){floor(n * runif(1)) + 1}
#sapply(rep(6,1000), fgenDescUnif)
```

KB - Golden section search - optimize()

```r
f <- function(x, a) (x - a)^2
xmin <- optimize(f, c(0, 1), tol = 0.0001, a = 1/3)
# print(xmin)
# plot(seq(0,1,0.01), f(seq(0,1,0.01),(1/3)))

## See where the function is evaluated:
# optimize(function(x) x^2*(print(x)-1), lower = 0, upper = 10)
```

HK - Generate Geometric using sample

```r
fgenGeo <- function(p){
    trials <- 1
    success <- 0
    # browser()
    while(success == 0){
        if(sample(c(0, 1), 1, replace = FALSE, prob = c(1 - p, p)) == 1){
            success <- 1
        } else{
            trials <- trials + 1
        }
    }
    return(trials)
}
# sapply(rep((1/3), 500), fgenGeo)
```

Template - Acceptance/Rejection region

```r
# Template for the acceptance rejection method:
dtarget <- function(x) {dbeta(x, 1, 2)}
dproposal <- function(x) {dunif(x)}
rproposal <- function(){runif(1)}

maxiter <- 2000
c <- 2

acceptance_rejection <- function(maxiter, c, dt, dp, rp) {
  # Draws from the target distribution by applying acceptance rejection model
  # Args:
  #   maxiter   No. of elements drawn
  #   c         majorizing constant
  #   dt        target density distribution
  #   dp        proposal density distribution
  #   rp        proposal random draw
  X <- vector("numeric", length= maxiter)
  i <- 0
  counter <- 0

  repeat {
    Y <- rproposal()
    U <- runif(1)

    if (U <= (dtarget(Y) / (c * dproposal(Y)))) {
      X[i] <- Y
      i = i + 1
    }

    counter = counter +1

    if (i == maxiter) {break}
  }

  return(structure(list(
    expected_rejection = (c-1)/c,
    real_rejection = 1 - maxiter/counter,
    X = X
  )))
}
```

Template - Gibbs sampling

```r
calculate_conditional_probability <- function(i, X, Y, ...) {
  # Returns the conditional probability of an element
  # Args: (might change with the given task)
  #   X   Value we are looking for
  #   Y   Data
  #   i   position
```

```r
  # 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)
}
```

Template - Metropolis Hasting

```r
dt <- function(x) {
  if (x <= 0) {
    stop("x has to be greater than 0.")
  }
  return(x^5 * exp(-x))
}

dp <- function(x, xt) {
  dchisq(x, floor(xt + 1))
}

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

tmax <- 2000

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(1, 10000, dt, dp, rp)
# hist(test, breaks = 25, freq = FALSE,
#      col = "grey",
#      xlim = c(0, 20), ylim = c(0, 0.2),
#      xlab = "x", ylab = "Density",
#      main = "Test of metropolis hastings")
# legend("topright",
#        legend = c("Sampled", "True values"),
#        col = c("grey", "red"),
#        pch = c(16, 16))
# lines(x, y / 120, col = "red", lwd = 2)
```

KB - Random Walk Monte Carlo - Choice of proposal dist

```
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
    abline(h=0)
    abline(h=1.96)
    abline(h=-1.96)
}
```

Albin - Estimating mean of dist with Monte Carlo by integral

```
# Estimate the mean from a integral
my_fun2<-function(first=0,sec=10,f_x=dnorm,true_mean_fun=rnorm,...){

  #Extract "almost" the true mean
  true_mean<-mean(true_mean_fun(10000,...))

  #Select the boundery for intergral from first to sec=end
  #my_x<-runif(10000,first,sec)

  #If boundery is -inf and inf, we use
  my_x<-true_mean_fun(10000,...)

  # Set h_x(x)*f(x)
  f_x__h_x<-function(x) x*f_x(x,...)
```

```r
#Extract the Y values for the integral
my_y<-f_x__h_x(my_x)

#Plot the integral
my_plot<-ggplot(mapping = aes(x=my_x))+
  geom_line(mapping = aes(y=f_x(my_x,...),col="Dist"),size=1)+
  geom_line(mapping = aes(y=my_y,col="Mean dist"),size=1)+
  theme_bw()

#Extract the box boundery of the inegral
max_y<-max(my_y)
min_y<-min(my_y)
max_x<-max(my_x)
min_x<-min(my_x)


#Sample data in the box
my_data<-data.frame(x=runif(2000,min_x,max_x),y=runif(2000,min_y,max_y))

#Get the f(x) from the sampled x in the box
my_data$f_x__h_x<-f_x__h_x(my_data$x)

#Exctact the values thats inside the f(x)
my_data$in_or_out1<-(my_data$y)<(my_data$f_x__h_x) & (my_data$y)>0
my_data$in_or_out2<- my_data$y>my_data$f_x__h_x & my_data$y<0
my_data$in_or_out3<- my_data$in_or_out1 |my_data$in_or_out2


my_plot2<-my_plot+geom_point(aes(x=my_data$x,y=my_data$y,col=my_data$in_or_out3))+
  scale_colour_manual(breaks = c("Dist",
                                 "Mean dist",
                                 "TRUE",
                                 "FALSE"),
                      labels = c("Dist"="f(x) distribution",
                                 "Mean dist"="x*f(x) distribution",
                                 "TRUE"="Inside",
                                 "FALSE"="Outside"),
                      values = c("Dist"="red",
                                 "Mean dist"="blue",
                                 "TRUE"="gray23",
                                 "FALSE"="gray79"))

my_plot2<-my_plot2+labs(y="Density",x="X",title="f(x) and x*f(x)")+
  theme(plot.title = element_text(hjust = 0.5))


#p1=numer of grey values in the graph over 0
#p2=numer of grey values in the graph under 0 (Often this is 0)
#N=Total number of

#sum(p1+p2)/N
first<-mean(my_data$in_or_out1)-mean(my_data$in_or_out2)
```

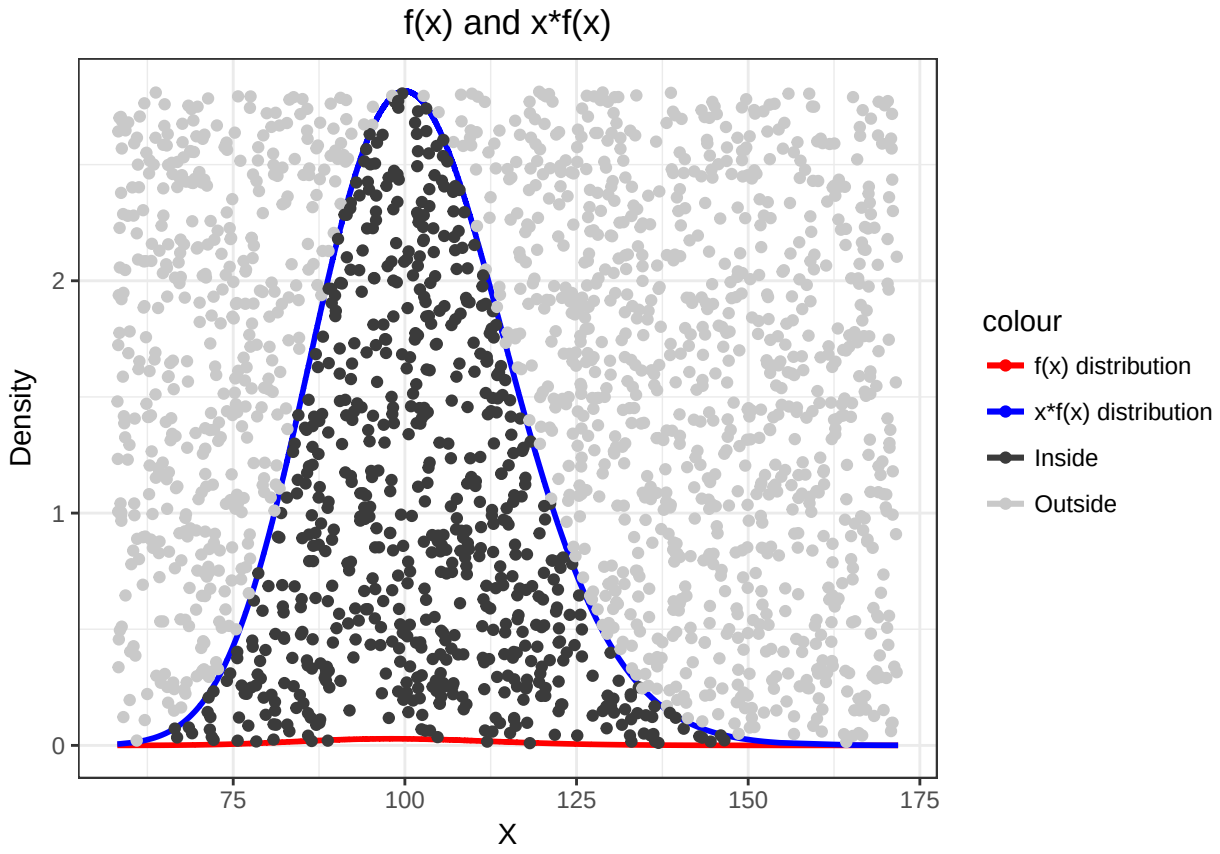
```
  #range(y)*range(x) from the box
  the_range<-(range(my_y)[2]-range(my_y)[1])*(range(my_x)[2]-range(my_x)[1])


  # (sum(p1+p2)/N) *range(y)*range(x)  The estimated mean
  estimated_mean<-first*the_range

  #Return the plot, estimated mean and "almost" the true mean
  list(my_plot2,c(Esimated=estimated_mean,Almost_true_mean=true_mean))
}

# Chi-Squared Distribution ####
my_fun2(first=0,sec = 100,f_x=dchisq,true_mean_fun =rchisq ,df=100)
```

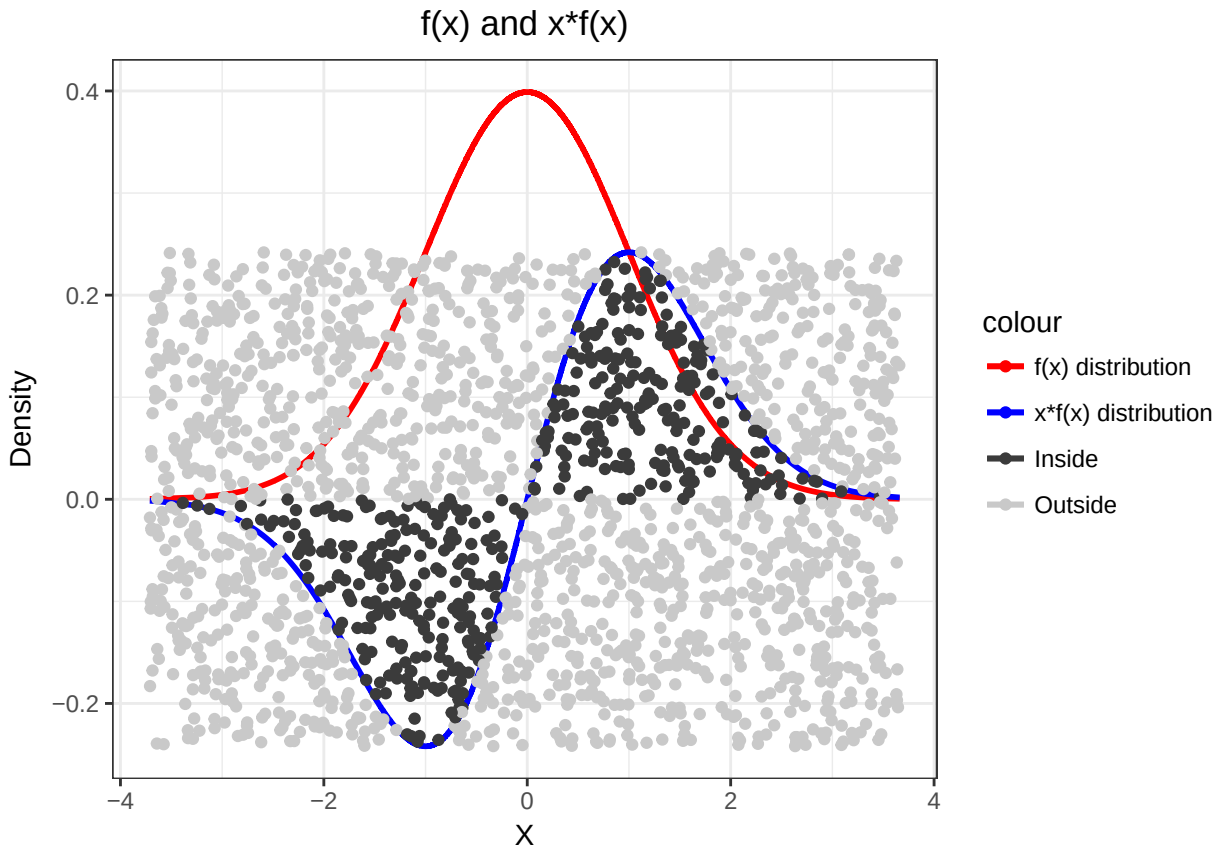
```
## [[1]]
```




```
##
## [[2]]
##         Esimated Almost_true_mean
##         103.1020          99.7212
```

```
# Normal Distribution ####
my_fun2(first=0,sec = 100,f_x=dnorm,true_mean_fun =rnorm ,mean=0,sd=1)
```

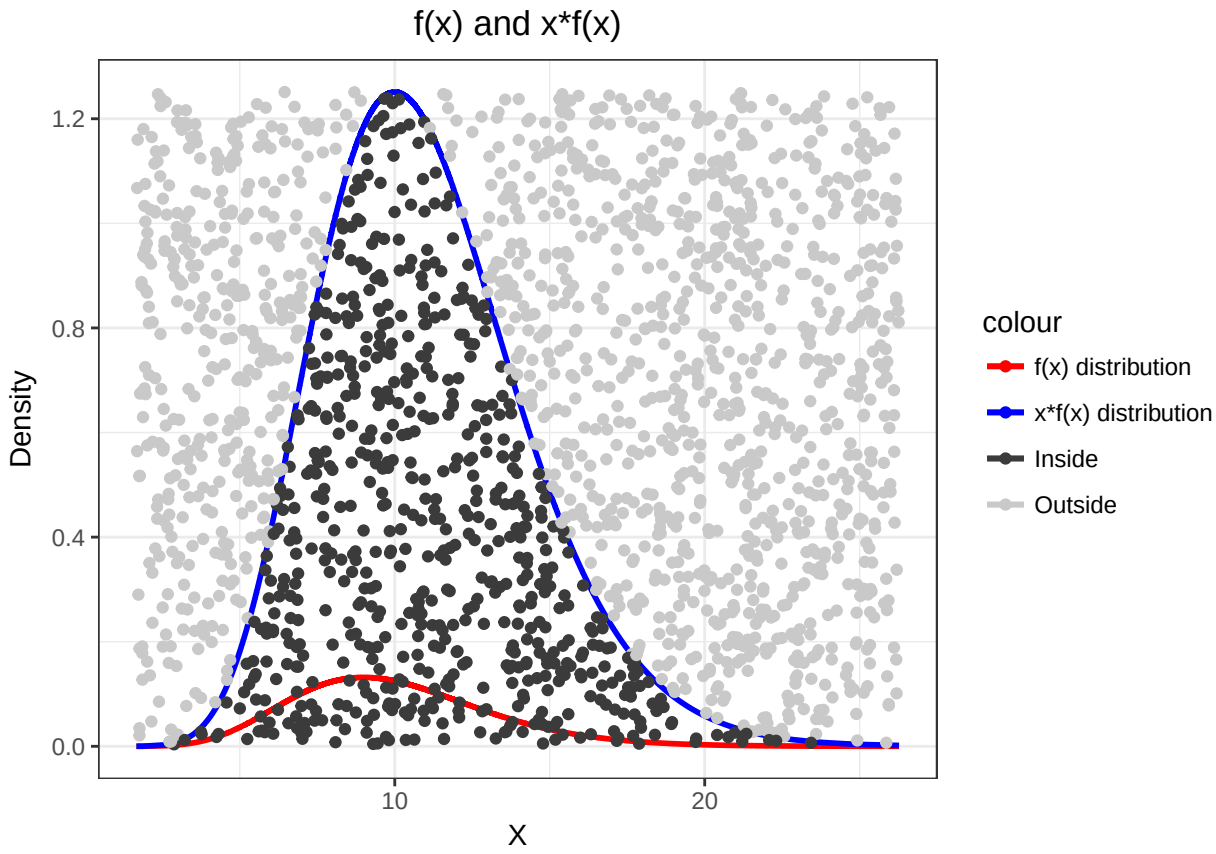
```
## [[1]]
```




```
##
## [[2]]
##         Esimated Almost_true_mean
##       -0.0142721        0.0100441
```

```
# Gamma Distribution ####
my_fun2(first=0,sec = 100,f_x=dgamma,true_mean_fun =rgamma ,shape=10)
```

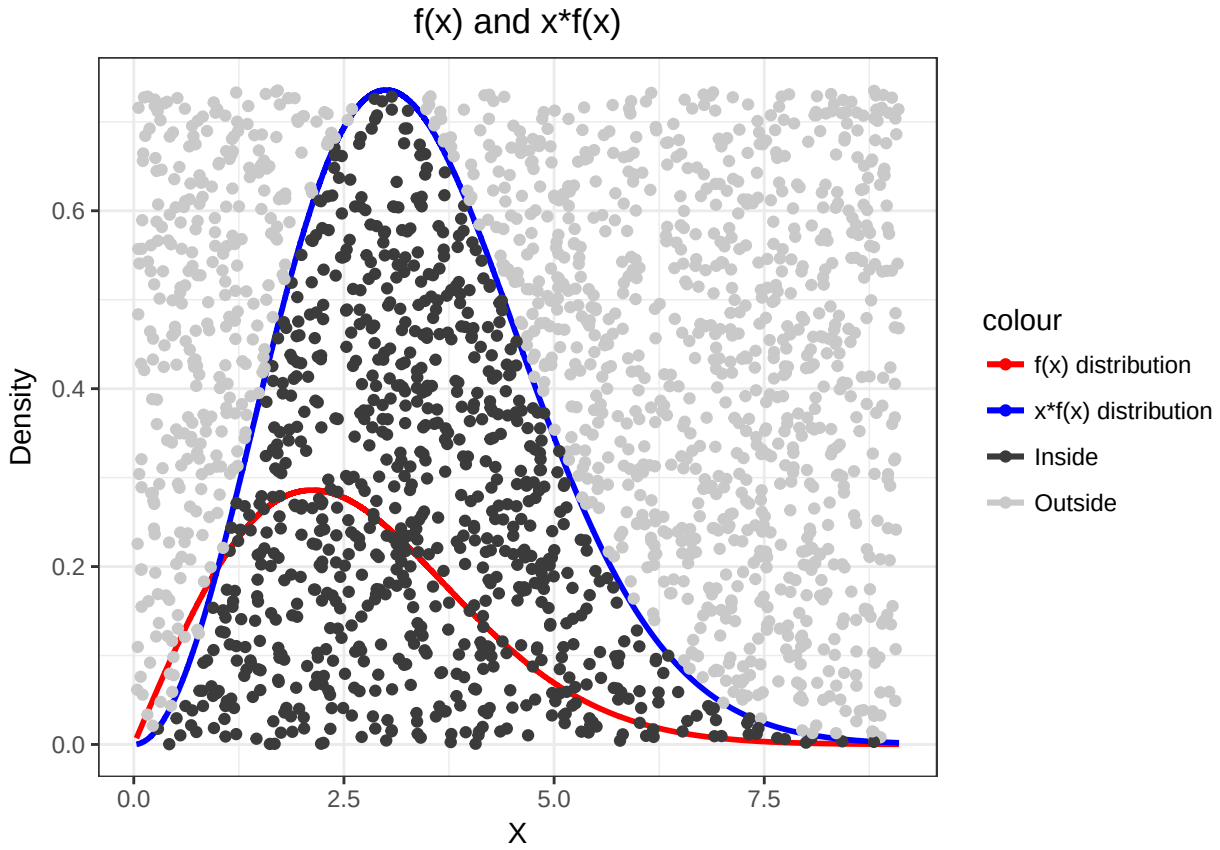
```
## [[1]]
```




```
##
## [[2]]
##         Esimated Almost_true_mean
##          9.86456         10.00782
```

```
# Weibull Distribution ####
my_fun2(first=0,sec = 20,f_x=dweibull,true_mean_fun =rweibull ,shape=2,scale=3)
```

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




```
##
## [[2]]
##         Esimated Almost_true_mean
##          2.63902          2.65363
```

Exam 171109

Exam 171109 - Q1.1 Generate Normal and Geometic RV

Question: Write two separate simulators. One that simulates from the standard normal, N(0,1), distribution (using only R's runif()) and another that simulates from the geometric distribution with a user provided parameter p (using only R's sample(c(0,1),1,replace=FALSE,prob=c(1-p,p))). Point(s) will be deducted for using other generators, in particular (but not limited to) rnorm() or rgeom().

Tip: Recall that if $\theta \sim Unif[0, 2\pi]$ and $D \sim Unif[0, 1]$ then the pair $X_1 = \sqrt{-2\,ln\,D}\,cos\,\theta$, $X_2 = \sqrt{-2\,ln\,D}\,sin\,\theta$ is a pair of independent and N(0,1) distributed random variables.

Tip: The geometric distribution can be interpreted as the number of tries to observing the first success.

See code for generating different distributions above.

```
fgenNorm(10)
```

```
##  [1]  0.0484832 -0.6766794  1.3441302  0.9399011 -1.3764731 -0.5781240
##  [7] -1.0781678  0.1361788 -0.7696082  0.1644549
```

```
# sapply(rep((1/3), 10), fgenGeo)
```

Exam 171109 - Q1.2 - optim() and Frechet mean

Question: Using your implemented simulators generate a bivariate (U, V) sample of 10 and 50 observations, where U and V are independent $U \sim N(0,1)$ and $V \sim$ geometric with parameter p = 1/3 distributions. Using optim() find the Frechet means of all 3 samples separately for $d(\vec{x},\vec{y}) := |x_1 - y_1| + |x_2 - y_2|$ and $d(x,y) := (x_1 - y_1)^2 + (x_2 - y_2)^2$. You may take $w_i = 1$ everywhere. You are free to choose the optimization method.

The idea is to build bivariate sample pairs $sample10 = (U_1, V_1), ..., (U_{10}, V_{10})$, and then compute distance function $d(a,b) = \sum^{10} i = 1(a - U_i)^2 + (b - V_i)^2$, where a is the start for par in optim() for Normal and b is start for par in optim() for geometric. By naming parameters in distance function to par and pars (as in optim()), the optim() passes along the parameters in the distance function.

The optim()$par returns the mean of the Normal and Geometric distribution if squared distance is used and the median if absolute distance is used.

```
# Dist functions
dist_sq <- function(par, pars){
    total <- c()
    # browser()
    for(i in 1:length(pars)){
        dist <- (par[1] - pars[[i]][1])^2 + (par[2] - pars[[i]][2])^2
        total <- c(total, dist)
    }
    return(sum(total))
}
# dist_sq(par = c(1,1), pars = bivar10)

dist_abs <- function(par, pars){
    total <- c()
    # browser()
    for(i in 1:length(pars)){
        dist <- abs(par[1] - pars[[i]][1]) + abs(par[2] - pars[[i]][2])
        total <- c(total, dist)
    }
    return(sum(total))
}

# Bivar10
bivar10 <- list()
for(i in 1:10){
    bivar10[[i]] <- c(fgenNorm(1), sapply(rep((1/3), 1), fgenGeo))
}

vect_N10 <- c()
for(i in 1:length(bivar10)){
    vect_N10 <- c(vect_N10, bivar10[[i]][1])
}

vect_geo10 <- c()
for(i in 1:length(bivar10)){
    vect_geo10 <- c(vect_geo10, bivar10[[i]][2])
}

c(mean(vect_N10), mean(vect_geo10))
```

```
## [1] -0.330976  3.300000
```

```
c(median(vect_N10), median(vect_geo10))
```

```
## [1] -0.621884  3.500000
```

```
opt_sq_10 <- optim(par = c(0,0), fn = dist_sq, method = "SANN", pars = bivar10)
opt_abs_10 <- optim(par = c(0,0), fn = dist_abs, method = "SANN", pars = bivar10)

# bivar50
bivar50 <- list()
for(j in 1:50){
    bivar50[[j]] <- c(fgenNorm(1), sapply(rep((1/3), 1), fgenGeo))
}

vect_N50 <- c()
for(i in 1:length(bivar50)){
    vect_N50 <- c(vect_N50, bivar50[[i]][1])
}

vect_geo50 <- c()
for(i in 1:length(bivar50)){
    vect_geo50 <- c(vect_geo50, bivar50[[i]][2])
}

c(mean(vect_N50), mean(vect_geo50))
```

```
## [1] 0.101196 3.280000
```

```
c(median(vect_N50), median(vect_geo50))
```

```
## [1] -0.0146955  2.0000000
```

```
opt_sq_50 <- optim(par = c(0,0), fn = dist_sq, method = "SANN", pars = bivar50)
opt_abs_50 <- optim(par = c(0,0), fn = dist_abs, method = "SANN", pars = bivar50)
```

Exam 171109 - Q1.3 - Plotting optim() results

Question: Make a scatterplot of your samples. On the plot indicate the estimated Frechet mean, the sample average and sample median (calculate the median separately in both dimensions). Can you draw any conclusions? If you find it difficult to make conclusions try generating samples of size 30 and 100, calculate the Frechet means (using optim()) and do the same plots.

```
library(ggplot2)
# ggplot() +
#     geom_point(aes(x = vect_N50, y = vect_geo50)) +
#     geom_point(aes(x = mean(vect_N50)), y = mean(vect_geo50),  color = "red") +
#     geom_point(aes(x = opt_sq_50$par[1], y = opt_sq_50$par[2]), color = "firebrick4", position = "jit
#     geom_point(aes(x = median(vect_N50)), y = median(vect_geo50), color = "green") +
#     geom_point(aes(x = opt_abs_50$par[1], y = opt_abs_50$par[2]), color = "forestgreen") +
#     labs(x = "N(0,1)", y = "Geometric(p = 1/3)") +
#     theme_minimal()
```

Exam 171109 - Q2.1 - Generate Unif(0,n)

Question: Assume you have a routine for generating uniform numbers Unif[0,1]. Implement an algorithm for sampling a uniform integer between 1 and n, i.e. $X \sim Unif\{1, ..., n\}$. Point(s) will be deducted for using other generators, in particular (but not limited to) for using sample().

```
# fgenDescUnif(100)
```

Exam 171109 - Q2.2 Robot Walk - Monte Carlo

Question: Consider a regular lattice of size m x m. A robot should walk from vertex (1, 1) to (m, m). It is only allowed to walk horizontal or vertical edges. When the robot reaches a vertex, it takes a random direction, including the edge where it entered. For an internal vertex this means 4 equally likely directions. At the outermost vertices and corners there are less options (2 or 3), but these (2 or 3 directions) are still equally likely. The horizontal and vertical edges have distance 1. Figure 1 illustrates this for m = 4. Let T be the distance walked by the robot before reaching vertex (m, m). What is the probability of the event T = 4m, for m = 4 and m = 8. The distribution of T may be explored by Monte Carlo sampling. It can also be found analytically but this is outside the scope of the course. Save your simulations for Question 2.3!

```
# Albins solution
robot_walk <- function(m=4){
  start<-c(1,1)
  end<-c(m,m)
  path <- matrix(c(1,1), ncol = 2)

  now <- start
  update_count <- 0
  while(any(now!=end)){

    temp_now <- c(0,0)
    i=1
    while(any(temp_now == (m+1), temp_now == 0)){
      temp_now <- now

      dirr<-sample(1:4,size = 1)
      dirr1<-c(0,0,0,0)
      dirr1[dirr] <- 1

      temp_now[1] <- temp_now[1]+dirr1[1]
      temp_now[1] <- temp_now[1]-dirr1[2]

      temp_now[2] <- temp_now[2]+dirr1[3]
      temp_now[2] <- temp_now[2]-dirr1[4]
    }
    update_count <- update_count+1
    now <- temp_now
    path <- rbind(path, now)
  }
  return(update_count)
  #return(path)
}

walking4 <- sapply(rep(4,1000), robot_walk)
walking8 <- sapply(rep(8,1000), robot_walk)
```

```
# hist(walking4 ,breaks = 50,freq = FALSE)
# hist(walking8 ,breaks = 50,freq = FALSE)
```

Exam 171109 - Q2.3

Question: What is the sample mean of the length your simulated walk from (1, 1) to (m, m)? Use R's boot(), boot.ci(), in library(boot), function to obtain 95% bootstrap confidence intervals for the mean. Report all the calculated bootstrap confidence intervals. Provide a short (maximum four sentences) intuitive description how the bootstrap and bootstrap confidence interval work (using formulae is strongly discouraged). **TIP**: Take about 1000 bootstrap replicates.

```
library(boot)
stat_mean <- function(data, lines){return(mean(data[lines]))}
stat_mean(walking4)
```

```
## [1] 44.498
```

```
boots <- boot(data = walking4, statistic = stat_mean, R = 1000)
ci <- boot.ci(boots, conf = .95, type = "bca")
```

The bootstrap function takes our sample and generate new subsamples out of the initial sample. Since the observed sample contains all the available information of the underlying population, the observed sample can be considered to be the "population", hence the distribution of any relevant test statistic can be simulated by using random samples from the "population" consisting of the original sample.

The simplest approach to find the confidence intervals is to use the generated subsamples distribution as an approximate to the sampling distribution of the population.

Exam 170509

Recall that the binomial coefficient "n choose k" is defined as ... where n and k are an arbitrary pair of integers satisfying $0 \leq k \leq n$. Consider the three below R expressions for computing the binomial coefficient. They all use the prod() function, which computes the product of all the elements of the vector passed to it.

- A) prod(1:n) / (prod(1:k) * prod(1:(n-k)))
- B) prod((k+1):n) / prod(1:(n-k)) -C) prod(((k+1):n) / (1:(n-k)))

Exam 170509 - Q1.1

Even if overflow and underflow would not occur these expressions will not work correctly for all values of n and k. Explain what is the problem in A, B and C respectively.

```
n <- 10
k <- 10
prod(1:n) / (prod(1:k) * prod(1:(n-k))) # A
```

```
## [1] Inf
```

```
prod((k+1):n) / prod(1:(n-k)) # B
```

```
## [1] Inf
```

```r
prod(((k+1):n) / (1:(n-k))) # C
```

```
## [1] Inf
```

Option A evaluate each part of the function on it's own, makes each faculty computation, and then uses the result from each component to compute the final vaule. As a result, we are working with extremely large numbers where multiplication is performed in the denominator and in the division. Working with numbers of this magnitude have a high risk of becoming bigger than we can handle and can't be represented properly, which is overflow.

Option B differs from option A, since the product of denominator is computed differently from A. But we are still performing a division with potentially very large numbers and risk of suffering from overflow.

Option C performs the production operation last, which is preferably, since we simplify as much as possible before making the product calculation, which implies that we're working with a maximum percision for as long as possible, since it's the product operation that makes the exponent explode in size.

However, non of the above option can compute the all situations correctly. For instance, A does not work when k = 0, since our denominator becomes 0. Non of the options manage to compute the situation when n = k.

Exam 170509 - Q1.2

Introduction
Computer Arithmetics

732A90
Computational Statistics

Krzysztof Bartoszek
(krzysztof.bartoszek@liu.se)

17 I 2018
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 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
- 2016 lecture slides (732A38)
- Handouts, R code
- Various suggested www pages or articles
- **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, handouts)
- Random Number Generation (JG pages 305–312, 325–328, handouts)
- Monte Carlo Methods (JG pages 312–318, 328 417–429, handouts)
- Numerical Model Selection and Hypothesis Testing (JG pages 52–56, 424, 435–467, handouts)
- Expectation Maximization Algorithm and Stochastic Optimization (JG pages 275–284, 296–298, 480–483, handout)

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

Floating–point system

- Rationals rounded towards the nearest computer float

```
options(digits=22)  #max possible
0.1
[1] 0.1000000000000000055511
```

- **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, 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}
```

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

- Distribution of computer floats

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

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

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

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

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] 2.0000000000000004440 89
(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] 14.39272672286572252176
[1] 14.39272672286572429812
```

- **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] 2.0611536224385575 7869942e-09
fTaylor(-20,100)
[1] -3.8538772173524193 93137e-10
fTaylor(-20,200)
[1] -3.8538772173524193 93137e-10
```

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 ...

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)
```

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)

23 I 2018
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

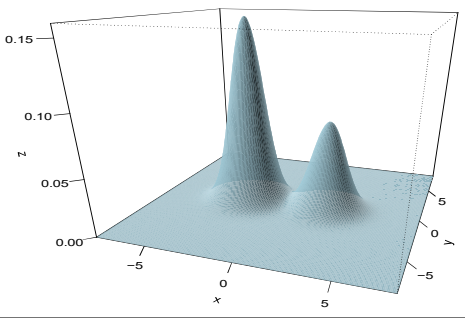
Economics/Logistics

- 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

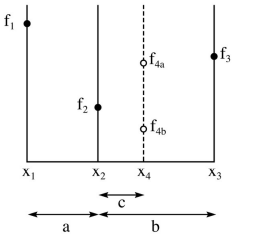
- Function of a single parameter, find minimum
- *What algorithm would you suggest?*
- **Golden–section search**
  local minimum on $[A, B]$ interval (constraint)
- Works by narrowing down the search interval with a constant reduction factor

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

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

732A90_ComputationalStatisticsVT2018_Lecture02codeSlide16.R

Multi–dimensional optimization

Find

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

Using (known, or numerically evaluated)

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

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

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

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

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

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)

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

BFGS Algorithm

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

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 $\vec{r}_k$)

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

`732A90_ComputationalStatisticsVT2018_Lecture02codeSlide31.R`

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

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

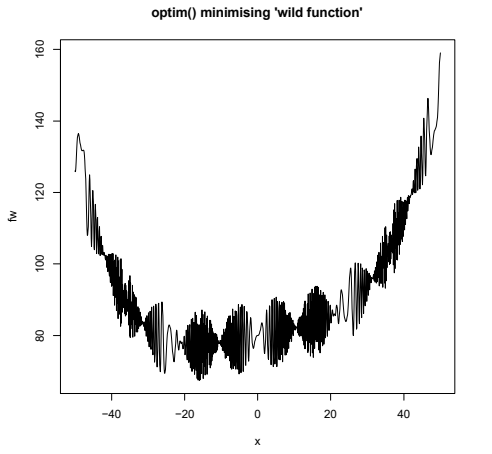
IDEA: $\vec{p}$ and $\vec{q}$ are orthogonal w.r.t. to an inner product associated with $\mathbf{A}$. Use this to find a basis that will allow for easy finding of $\vec{x}$.

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**

kD Optimization: Example



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)

31 I 2018
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. ...

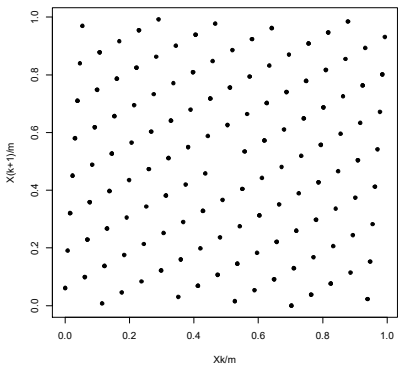
First step: Generating Unif$[0,1]$

```
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  1  0  1
[1] 73  0  1  0  0  1
[1] 253  1  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$[0,1]$



`732A90_ComputationalStatisticsVT2018_Lecture03codeSlide06.R`

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$[a,b]$

- $U \sim$ Unif$[0,1]$ can be transformed into $X \sim$ 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

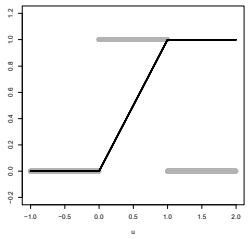
1. Why $+1$?
2. How can U be transformed into Y, where Y is discrete uniform on integers $(50, 55, 60)$?

Second step: Generating nonuniform random numbers

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

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

- 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: Example

Find F_X^{-1}

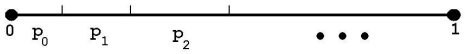
$$\begin{aligned} y &= 1 - e^{-\lambda x} \\ e^{-\lambda x} &= 1 - y \\ x &= -\tfrac{1}{\lambda}\ln(1-y) \\ F_X^{-1}(y) &= -\tfrac{1}{\lambda}\ln(1-y) \end{aligned}$$

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

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

Generating discrete RVs

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



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 $U \sim \text{Unif}(0,1)$

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

Y has same probability distribution as X

Inverse CDF method: Example

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

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

implying (**SHOW THIS**)

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

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

Inverse CDF method

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

   time–consuming

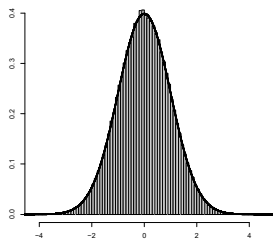
   numerical errors ?

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

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

Assume

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



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

$$X_1 = \sqrt{-2\ln D}\cos\theta$$
$$X_2 = \sqrt{-2\ln D}\sin\theta$$

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**

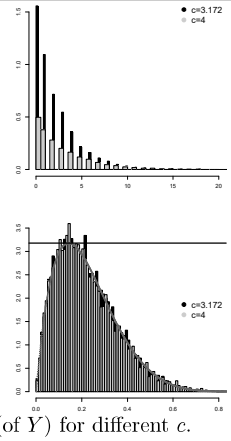
- $X \sim f_X$ **CHECK THIS**
- Larger c: larger rejection rates—c as small as possible
- Can work in higher dimensions—**but** high rejection rate

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

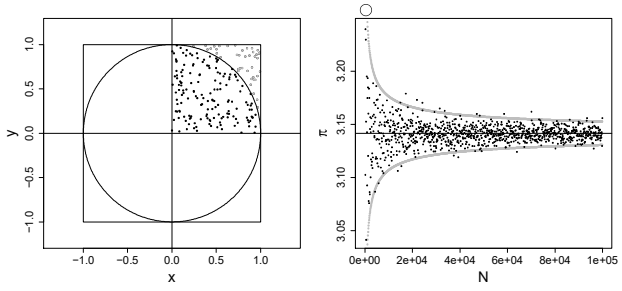
Krzysztof Bartoszek
(krzysztof.bartoszek@liu.se)

6 II 2018
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

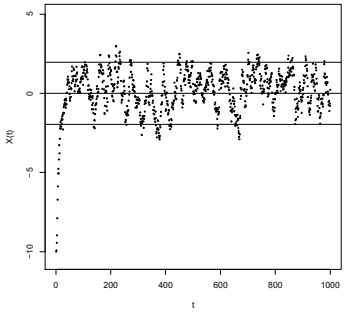
1. MCMC solves this
2. Not needed (given D it is constant)

Markov Chains: Recap

- A Markov chain is a sequence $X_0, X_1, \ldots$ 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

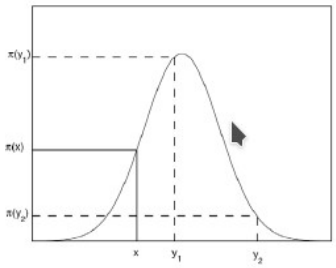
$$\alpha(X_t, Y) = \min\left\{1, \frac{\pi(Y)q(X_t|Y)}{\pi(X_t)q(Y|X_t)}\right\}$$

1: Initialize chain to X_0, $t = 0$
2: **while** $t < t_{\max}$ **do**
3: Generate a candidate point $Y \sim q(\cdot|X_t)$
4: Generate $U \sim Unif(0,1)$
5: **if** $U < \alpha(X_t, Y)$ **then**
6: $X_{t+1} = Y$
7: **else**
8: $X_{t+1} = X_t$
9: **end if**
10: $t = t + 1$
11: **end while**

Choice of proposal distribution

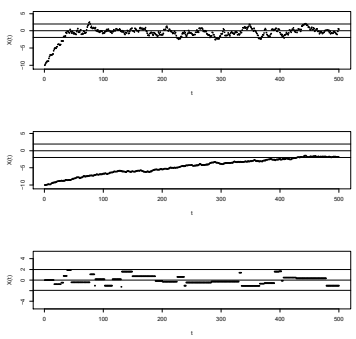
- In Random–Walk Monte Carlo

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



Choice of proposal distribution

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



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

`732A90_ComputationalStatisticsVT2018_Lecture04codeSlide14.R`

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).

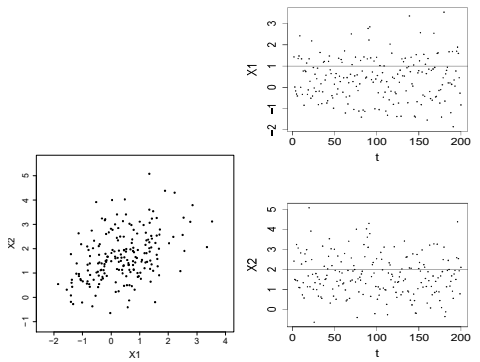
Gibbs sampler: target: d–dim $\mathcal{N}(\mu, \Sigma)$

`732A90_ComputationalStatisticsVT2018_Lecture04codeSlide18.R`

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

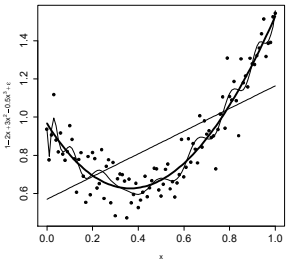
Model Selection and Hypothesis Testing

732A90
Computational Statistics

Krzysztof Bartoszek
(krzysztof.bartoszek@liu.se)

14 II 2018
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

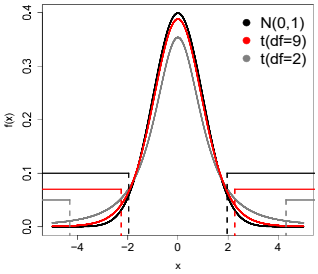
```
x<-rnorm(10,mean=4,sd=1)
```

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

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

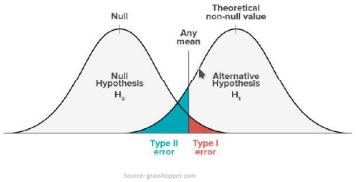



```
tx<-(mean(x)-4)/(sqrt(var(x)/length(x)))
t0<-qt(0.975,df=length(x)-1)
(tx>t0)||(tx<(-t0)) ## reject if TRUE
```

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.

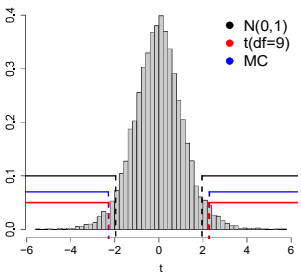
$H_0 : \mu = 4, X \sim \mathcal{N}(\mu, \sigma^2)$
$H_1 : \mu \neq 4, X \sim \mathcal{N}(\mu, \sigma^2)$

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

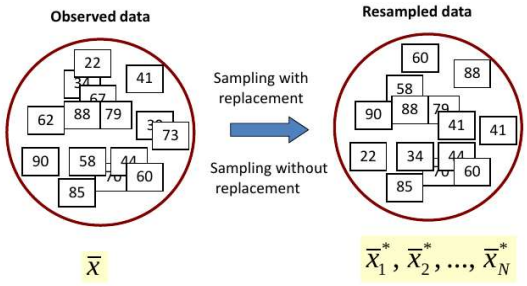
1: $T(X)$ value of statistic from observed data

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: $\hat{p} = \#\{|T(X_{g_b^*})| \geq |T(X)|\}/B$

Resampling methods



Uncertainty estimation: confidence intervals

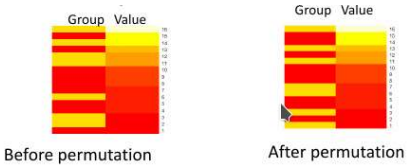
Estimate $100(1-\alpha)\%$ percentile confidence interval for w
$se(\cdot)$ is the square root of estimated variance (computationally heavy)
NOT by jackknife **TOO DEPENDENT!!**

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)

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)

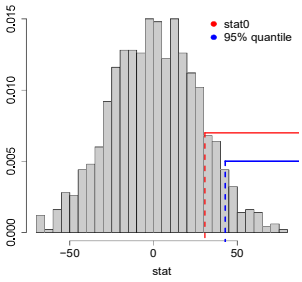


Test statistic:

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

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
```

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 $\hat{w}$ 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: variance of estimator

Bootstrap

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

Jackknife ($n = B$)

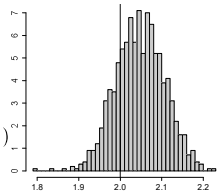
$$\widehat{\text{Var}}[T(\cdot)] = \frac{1}{n(n-1)} \sum_{i=1}^{n} (T_i^*) - J(T))^2 ,$$

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 2016 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)

15 II 2018
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	
	$\rightarrow x_i^{(k+1)}$ 11011010
$x_j^{(k)}$ 00111010	
Inversion	
$x_i^{(k)}$ 11101011	$\rightarrow x_i^{(k+1)}$ 11010111
Mutation	
$x_i^{(k)}$ 11101011	$\rightarrow x_i^{(k+1)}$ 10111011
Clone	
$x_i^{(k)}$ 11101011	$\rightarrow x_i^{(k+1)}$ 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** $\mathbf{Y}$'s and $\mathbf{Z}$'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

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

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

732A90_ComputationalStatisticsVT2018_Lecture06codeSlide15.R

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 2016 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.

Source: Pattern recognition by Bishop

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