LINKÖPINGS UNIVERSITET
Department of information and computer science
Statistics and Machine learning
Måns Magnusson

Exam

Exam solutions (to 732G36 and 732A50)

Time:	8-12, 2015-10-14
Material:	The extra material is included in the zip-file **exam_material.zip**.
Grades:	A = 19-20 points. B = 17-18 points. C = 12-16 points. D = 10-11 points. E = 8-9 points. F = 0-7 points.

Instructions

Write your code in an R script file named **Main.R**. The R code should be complete and readable code, possible to run by copying directly into a script. Comment directly in the code whenever something needs to be explained or discussed. Follow the instructions carefully.

Problem 1 (5 p)

a) Create the following mathematical function as a function in R (called `f`)

$$f(\mathbf{x}) = \frac{\sum_i^n |x_i - \bar{x}|}{n}$$

where $\bar{x}$ is $\frac{1}{n}\sum_i x_i$ an n is the length of the vector $\mathbf{x}$. You are not allowed to use any vectorized functions such as `sum()` or `mean()`.

```
f(1:5)

[1] 1.2

f(c(7,2,2,1,-4))

[1] 2.48
```

Suggested solution:

```
function(x){
  n <- length(x)
  mean_x <- 0
  for(i in seq_along(x)){
    mean_x <- mean_x + x[i]
  }
  mean_x <- mean_x / n
  res <- 0
  for(i in seq_along(x)){
    res <- res + abs(x[i] - mean_x)
  }
  res / n
}
```

---

b) What is the computational complexity of this algorithm based on the input length?

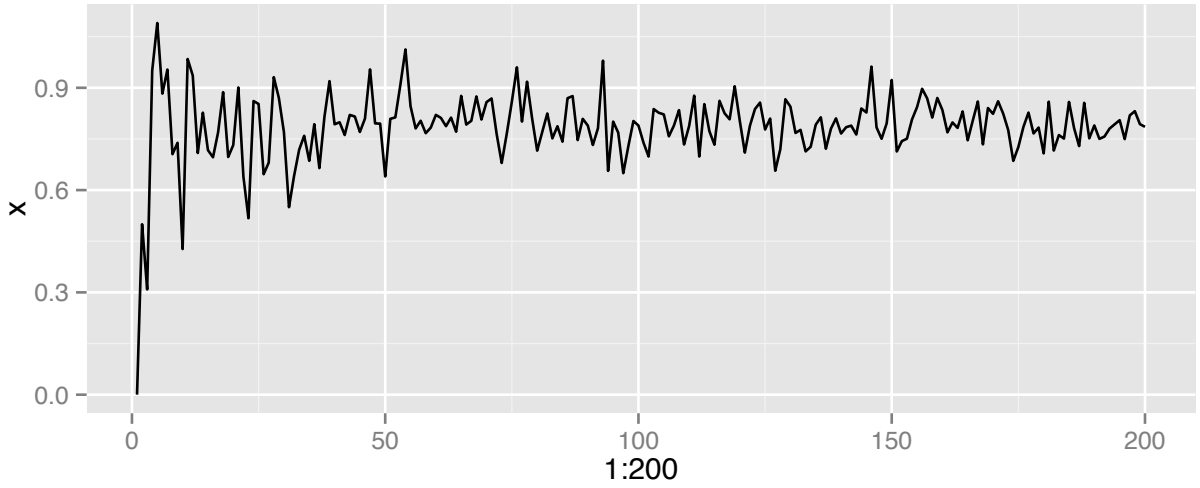
Suggested solution:
The complexity is linear, $O(x)$, in the input size.

---

c) Visualize the value of $f(\mathbf{x})$ as $\mathbf{x}$ is growing by drawing 1,2,3,...,200 draws from a $\mathcal{N}(0,1)$ distribution and visualize the value of $f(\mathbf{x})$ using a linegraph in `ggplot2`.

Suggested solution:

```
x <- vapply(X = 1:200, FUN=function(X) f(rnorm(X)), numeric(1))
library(ggplot2)
qplot(x=1:200, y=x, geom = "line")
```



Problem 2 (5 p)

a) Create a function you call `hilbert(n,m)` that creates a Hilbert matrix. The element (i, j) of the Hilbert matrix is defined as follows

$$H_{ij} = \frac{1}{i + j - 1}$$

where i is the row and j is the column in the matrix. The size of the matrix should be `n` $\times$ `m`.

```
hilbert(1,4)

     [,1] [,2]     [,3] [,4]
[1,]    1  0.5 0.333333 0.25

hilbert(2,2)

     [,1]     [,2]
[1,]  1.0 0.500000
[2,]  0.5 0.333333
```

Suggested solution:

```
function(n, m){
  res_mat <- matrix(0, n, m)
  for(i in 1:n){
    for(j in 1:m){
      res_mat[i,j] <- 1 / (i + j - 1)
    }
  }
  res_mat
}
```

---

b) Calculate $\det\left(\mathbf{H}^T\mathbf{H}\right)$ where $\mathbf{H}$ is a 5×5 Hilbert matrix.

Suggested solution:

```
det(t(hilbert(5,5))%*%hilbert(5,5))

[1] 1.40572e-23
```

---

c) Create a test suite for this function (using `testthat`) that checks that (1) the result of the function is a matrix and (2) that one of the examples above will be returned using the function.

Suggested solution:

```
library(testthat)
test_that("hilbert() is working", {
  H <- hilbert(1,4)
  expect_is(H, "matrix")
  expect_equal(H, matrix(c(1,1/2,1/3, 1/4), nrow=1))
})
```

Problem 3 (5 p)

a) Create a function you call `my_tidy_titanic_data()` that should return a tidy dataset from the Titanic dataset in R. You should use `tidyr` and `dplyr` in the function.

The `Titanic` dataset is a three dimensional table, below is how you should convert it to a `data.frame` (and then make it tidy).

```
# Load and convert the titanic dataset
data("Titanic")
x <- as.data.frame(as.matrix(ftable(Titanic)))
# The resulting function
head(my_tidy_titanic_data())

  class    sex   age survived counts
1   1st   Male Child       No      0
2   1st   Male Adult       No    118
3   1st Female Child       No      0
4   1st Female Adult       No      4
5   2nd   Male Child       No      0
6   2nd   Male Adult       No    154
```

Suggested solution:

```
function(){
  data("Titanic")
  tit <- as.data.frame(as.matrix(ftable(Titanic)))
  tit <- dplyr::mutate(tit, tmp = rownames(tit))
  tit <- tidyr::gather(tit, survived, counts, -tmp)
  tit_tidy <- tidyr::separate(tit, "tmp", c("class", "sex", "age"), "_")
  tit_tidy
}
```

---

b) Create a new function you call `aggregate_away_sex(x)` that takes a tidy titanic dataset as variable `x` and returns a dataset where the variable `sex` has been aggregated together in each group. See the example below. You should use `dplyr` functions.

```
tita<- my_tidy_titanic_data()
head(aggregate_away_sex(tita))

  class   age survived counts
1   1st Adult       No    122
2   1st Adult      Yes    197
3   1st Child       No      0
4   1st Child      Yes      6
5   2nd Adult       No    167
6   2nd Adult      Yes     94
```

Suggested solution:

```
function(x){
  grp_data <- dplyr::group_by(x, class, age, survived)
  agg_data <- dplyr::summarise(grp_data, counts=sum(counts))
  as.data.frame(agg_data)
}
```

Problem 4 (5 p)

a) Create a function you call `counter_factory()` that generates counter functions/clojures. The factory function should take two arguments, `start` and `max`. The `start` argument should set the starting value of the counter and `max` should be the maximum value of the counter (so when called afterwards, the maximum value will be return each time).

```
cnt <- counter_factory(start = 2, max = 4)
cnt()

[1] 3

cnt()

[1] 4

cnt()

[1] 4
```

Suggested solution:

```
function(start, max){
  i <- start
  function(){
    if(i < max) i <<- i + 1
    i
  }
}
```

---

b) Make the counter a counter object and implement a `summary()` method that returns the current value of the counter and the `max` value of the counter. You can choose to use either a S3 or RC type classes.

```
cnt <- counter_factory(start = 2, max = 4)
summary(cnt())

Count object
i: 3
max:  4

summary(cnt())

Count object
i: 4
max:  4
```

Suggested solution:

```
counter_factory

function(start, max){
  i <- start
  function(){
    if(i < max) i <<- i + 1
    res <- c(i, max)
    class(res) <- "counter"
    res
  }
}

summary.counter

function(x, ...){
   cat("Count object\ni:", x[1], "\nmax: ",x[2])
}
```

---

c) Document the function using roxygen type documentation. The documentation should include a title, a short description, the input arguments and the return value of the function.

Suggested solution:

```
#' @title A counter factory
#' @param start Starting iteration number
#' @param max Maximum iterations
#' @description Creates a counter object that is a clojure.
#' @value Returns a function/clojure with environment containing i and max
```

Good luck!