Problemset 2

Nikolai German (12712506)

```r
library(tidyr)
library(dplyr)
```

```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(ggplot2)
```

Exercise 2.1 MCMC

```r
P0 <- matrix(c(1/3, 1/6, 1/6, 1/2, 2/3, 1/2, 1/6, 1/6, 1/3), nrow = 3)
```

a)

rainy today: $\mathbb{P}(R_{t_0}) = 1$

```r
rny_tdy <- c(0, 0, 1)
```

propability of rain on the day after tomorrow, if it's raining today $\mathbb{P}(R_{t_2}|R_{t_0}) = 1$

```r
(rny_tdy %*% P0 %*% P0)[[3]]
```

```
[1] 0.2222222
```

b)

```r
mc <- function(P, state = c("Sunny" = 1, "Cloudy" = 0, "Rainy" = 0), n = 1000) {
  checkmate::assertIntegerish(n, lower = 1, len = 1, any.missing = FALSE)
  checkmate::assertNumeric(state, lower = 0, upper = 1, min.len = 1, any.missing = FALSE)
  checkmate::assertMatrix(P,
                         mode = "numeric",
                         any.missing = FALSE,
                         nrows = length(state),
                         ncols = length(state))

  colnames(P0) <- c("Sunny", "Cloudy", "Rainy")
  rownames(P0) <- c("Sunny", "Cloudy", "Rainy")

  if (Matrix::rankMatrix(P0)[[1]] < length(state)) {
    stop("P has not full rank!")
  }

  E <- eigen(P)
  res <- state %*% E$vectors %*% diag(E$values^n) %*% solve(E$vectors)
  colnames(res) <-c("Sunny", "Cloudy", "Rainy")
  return(res)
}
```

```r
E <- eigen(t(P0))
E$vectors[,1] / sum(E$vectors[,1])
```

```
[1] 0.2 0.6 0.2
```

```r
mc(P0)
```

```
     Sunny Cloudy Rainy
[1,]   0.2    0.6   0.2
```

c)

$$\pi P = \pi \tag{1}$$
$$\iff \pi(P - I) = 0 \tag{2}$$
$$\iff (P - I)^\top \pi^\top = 0 \tag{3}$$

$$(P - I)^\top = \begin{bmatrix} -2/3 & 1/6 & 1/6 \\ 1/2 & -1/3 & 1/2 \\ 1/6 & 1/6 & -2/3 \end{bmatrix} \tag{4}$$

Wir suchen also den Eigevektor zum Eigenwert $\lambda = 1$:

$$\begin{bmatrix} -2/3 & 1/6 & 1/6 \\ 1/2 & -1/3 & 1/2 \\ 1/6 & 1/6 & -2/3 \end{bmatrix} \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{5}$$
$$\implies \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{pmatrix} = \alpha * \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} \tag{6}$$

$$\pi P = \pi$$

Wir schreiben den Ausdruck um, zu:

$$(P - I)^\top \pi^\top = 0$$

$$-2/3\pi_S + 1/6\pi_C + 1/6\pi_R = 0 \tag{7}$$
$$1/2\pi_S - 1/3\pi_C + 1/2\pi_R = 0 \tag{8}$$
$$1/6\pi_S + 1/6\pi_C - 2/3\pi_R = 0 \tag{9}$$

$$-4\pi_S + 1\pi_C + 1\pi_R = 0 \tag{10}$$
$$3\pi_S - 2\pi_C + 3\pi_R = 0 \tag{11}$$
$$1\pi_S + 1\pi_C - 4\pi_R = 0 \tag{12}$$

Exercise 2.2 - MCMC, Gibbs Sampling

a)

Let $X_i \overset{iid}{\sim} Exp(1)$, $S = \sum_{i=1}^{2} X_i$, which leads us to:

$$\mathbb{P}(S \leq s) = \int_0^s f_{X_1}(x) * (F_{X_2}(s-x))dx \tag{13}$$

Taking the complementary propability: $\mathbb{P}(Z > z) = 1 - \mathbb{P}(Z \leq z)$:

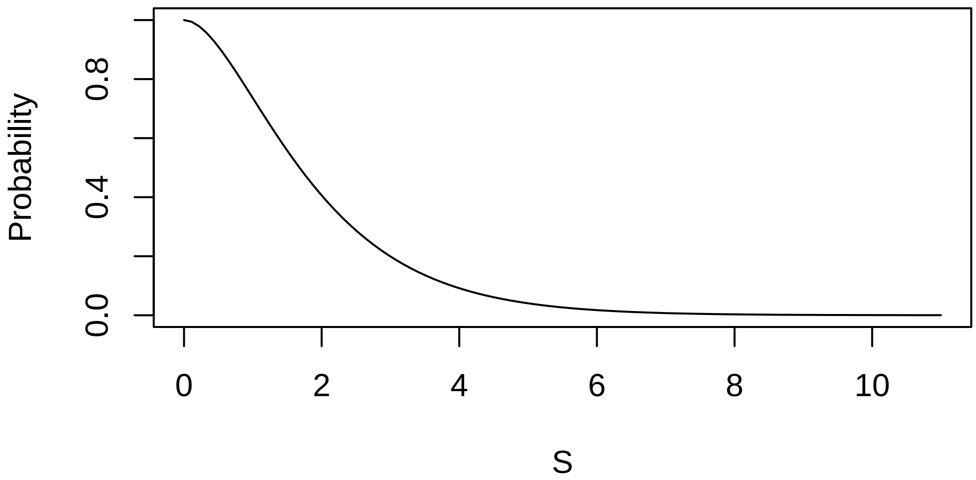
$$\mathbb{P}(S > s) = 1 - \int_0^s f_{X_1}(x) * (F_{X_2}(s-x))dx \tag{14}$$

With $f_{X_i} = exp(-x)$ and $F_{X_i} = 1 - exp(-x)$ we obtain:

$$\begin{aligned}
\mathbb{P}(S > s) &= 1 - \int_0^s exp(-x) * (1 - exp(-(s-x)))dx & (15)\\
&= 1 - \int_0^s exp(-x) + \int_0^s exp(-s)dx & (16)\\
&= 1 - F_{X_i}(s) + exp(-s) * s & (17)\\
&= 1 - (1 - exp(-s)) + s * exp(-s) & (18)\\
&= exp(-s) * (s+1) & (19)
\end{aligned}$$

Let's visaualise this probability:

```r
Proba_S <- function(x) {
  (x + 1) * exp(-x)
}

plot(Proba_S, xlim = c(0, 11), xlab = "S", ylab = "Probability")
```



And compute $\mathbb{P}(S > 10)$:

```r
11 * exp(-10)
```

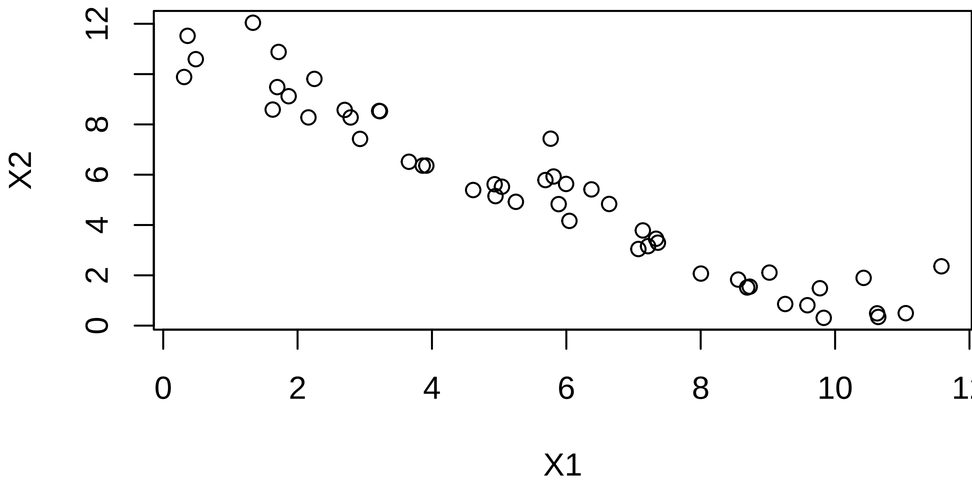
```
[1] 0.0004993992
```

Compare the frequency of the Sum of two random variables:

```r
set.seed(17)
X1 <- rexp(100000)
X2 <- rexp(100000)
S <- X1 + X2

Y <- data.frame(X1, X2, S)[(S > 10), ]

cat(sprintf("Partial of accepted pairs: %.5f", mean(S > 10)))
```

```
Partial of accepted pairs: 0.00050
```

```r
plot(Y[, c(1,2)])
```



One can easyily see, that the probability for $S > 10$ is very low. Just simulating the random variables would result in a turn-down rate of close to 100%. If the threshold would be even larger, we would discard even more candidates.

To successfully draw from Y, we would need to draw approximatly 2000 times from each X.

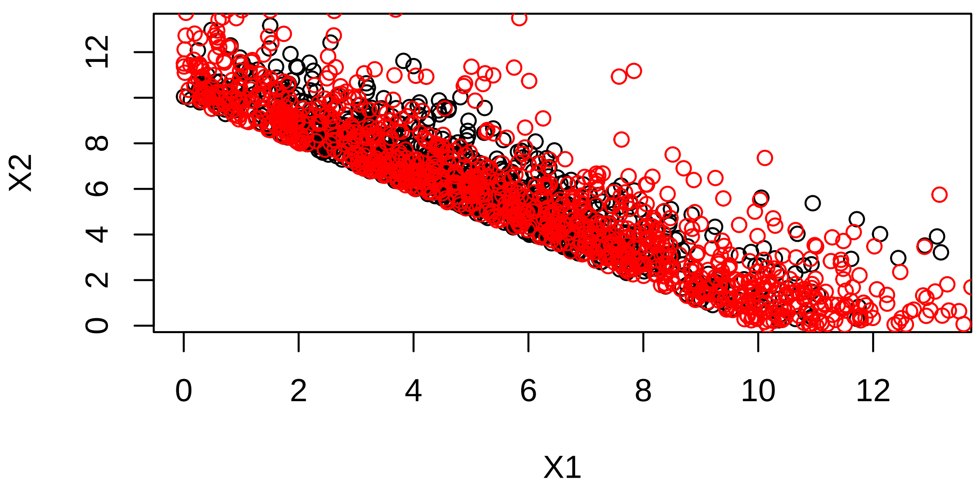
b)

The Metropolis-Hastings Algorithm constructs a Markov-Chain $Y^t, Y^{t+1}, ...Y^T$. Given the current state Y^t, we draw from a proposal $Q(y|y^t)$. The proposal $y*$ gets then either accepted as Y^{t+1} or rejected.

```
mh <- function(start = c(5, 5), sigma = 1, n = 1000) {
  checkmate::assertNumeric(start, len = 2, any.missing = FALSE)
  checkmate::assertNumeric(sigma, len = 1, any.missing = FALSE, lower = 0)
  checkmate::assertIntegerish(n, len = 1, lower = 1, any.missing = FALSE)

  f <- function(y) {
    checkmate::assertNumeric(y, len = 2, any.missing = FALSE)
    if (sum(y) < 10) {
      return(0)
    } else {
      return(dexp(y[[1]]) * dexp(y[[2]]))
    }
  }

  alpha <- function(state, proposal) {
    checkmate::assertNumeric(state, len = 2, any.missing = FALSE, lower = 0)
```

```r
    checkmate::assertNumeric(proposal, len = 2, any.missing = FALSE)
    min(1, f(proposal) / f(state))
  }

  Y <- matrix(nrow = n, ncol = 2)
  colnames(Y) <- c("X1", "X2")
  y_t <- start
  i <- 1

  while (i <= n) {
    proposal <- rnorm(2, mean = y_t, sd = sigma)
    U <- runif(1)
    if (U <= alpha(y_t, proposal)) {
      Y[i, ] <- proposal
      y_t <- proposal
      i <- i + 1
    }
  }

  return(Y)
}
```

```r
y <- mh(sigma =1)
y2 <- mh(sigma = 16)

plot(y)
points(y2, col = "red")
```



c)

d)

Exercise 2.3

a)

The Metropolis-Hasting algorithm states, that Y^* should be accepted, if $U^* < \alpha(Y^*, Y^t)$. With $U^* \sim Unif(0,1)$ and $\alpha(Y^*, Y^t) = min(1, \frac{f^*(Y^*)}{f*(Y^t)}$, one can clearly see, that the propability of the Markov-Chain to move from state i to state j is $a_{i,j} = min(1, \frac{f^*(Y=j)}{f*(Y=i)}$.

```
f <- function(Y) {
  checkmate::assertCharacter(Y, n.chars = 1)
  checkmate::assertChoice(Y, choices = c("H", "R", "W"))
  switch(Y,
         H = .3,
         R = .1,
         W = .6)
}

alpha <- function(state, proposal) {
  min(1, (f(proposal) / f(state)))
}

states <- c("H", "R", "W")
A <- matrix(nrow = 3, ncol = 3, dimnames = list(states, states))

for (i in seq_len(3)) {
  for (j in seq_len(3)) {
    A[i,j] <- alpha(state = states[[i]], proposal = states[[j]])
  }
}



print(A)
```

```
    H         R W
H 1.0 0.3333333 1
R 1.0 1.0000000 1
```

```
W 0.5 0.1666667 1
```

b)

Computing P ist pretty straight forward: First initializing the Matrix Q for $q = 0.2$:

```r
get_Q <- function(q = .2, dim = 3, names = NULL) {
  Q <- matrix(q, nrow = dim, ncol = dim, dimnames = list(names, names))
  diag(Q) <- 1 - (dim - 1) * q
  return(Q)
}

get_Q(names = states)
```

```
    H   R   W
H 0.6 0.2 0.2
R 0.2 0.6 0.2
W 0.2 0.2 0.6
```

After that, we compute the values $a_{i,j}$ of A, differentiating between the cases $i \neq j$ and $i = j$:

```r
Q <- get_Q(names = states)
P <- matrix(nrow = 3, ncol = 3)
for (i in seq_len(3)) {
  for (j in seq_len(3)) {
    if (i != j) {
      P[i,j] <- A[i,j] * Q[i,j]
    } else {
      P[i,i] <- 1 - sum(A[i, seq_len(3) != i] * Q[i, seq_len(3) != i])
    }
  }
}

print(P)
```

```
          [,1]       [,2]      [,3]
[1,] 0.7333333 0.06666667 0.2000000
[2,] 0.2000000 0.60000000 0.2000000
[3,] 0.1000000 0.03333333 0.8666667
```

c)

We want to show, that the distribution $\pi = (0.3, 0.1, 0.6)$ is invariant to P, thus beeing an eigenvector of P to the eigenvalue $\lambda = 1$:

```
c(.3, .1, .6) %*% P
```

```
     [,1] [,2] [,3]
[1,]  0.3  0.1  0.6
```

Since $\pi P = \pi$, it follows directly, that:

$$\begin{aligned}
\pi P^n &= \pi P \cdot P^{n-1} \\
&= \pi P^{n-1} \\
&\vdots \\
&= \pi P \\
&= \pi
\end{aligned}$$

d)

```
mc2 <- function(start = NULL, states = c("H", "R", "W"), fun = f, q = 0.2, K = 100, burn.in =
  checkmate::assertFunction(fun)
  checkmate::assertCharacter(start, len = 1, any.missing = FALSE, null.ok = TRUE)
  checkmate::assertChoice(start, choices = states, null.ok = TRUE)
  checkmate::assertNumeric(q, len = 1, any.missing = FALSE, lower = 0, upper = .5)
  checkmate::assertIntegerish(K, len = 1, any.missing = FALSE, lower = 1)
  checkmate::assertIntegerish(burn.in, len = 1, any.missing = FALSE, lower = 0, upper = K - 1

  if (is.null(start)) {
    start <- sample(states, 1)
  }

  Q <- get_Q(q = q, dim = length(states), names = states)

  alpha <- function(state, proposal) {
  min(1, (fun(proposal) / fun(state)))
  }
```

```r
  y_t <- start
  Y <- character(K)

  tries <- 0
  i <- 1

  while (i <= K) {
    # i.state <- which(states == y_t)
    # probs <- rep(q, length(states))
    # probs[[i.state]] <- 1 - sum(probs[-i.state])
    # proposal <- sample(states, 1, prob = probs)
    proposal <- sample(states, 1, prob = Q[y_t, ])
    U <- runif(1)
    if (U < alpha(y_t, proposal)) {
      y_t <- proposal
      Y[[i]] <- y_t
      i <- i + 1
    }
    tries <- tries + 1
  }

  Y <- Y[(burn.in + 1):K]
  structure(Y,
            distribution = table(Y) / (K - burn.in),
            acceptance_rate = K/tries)
}
```

```r
samp <- mc2(K = 25000)

tibble(
  H = cumsum(samp == "H"),
  R = cumsum(samp == "R"),
  W = cumsum(samp == "W")
) %>% mutate(n = H + R + W) %>%
  filter(n > 50) %>%
  pivot_longer(-n, values_to = "count", names_to = "state") %>%
  mutate(freq = count/n) %>%
  ggplot(aes(n, freq, color = state)) +
  geom_point(size = .6, alpha = .3) +
  geom_line() +
  geom_hline(yintercept = .1, lty = "dashed") +
  geom_hline(yintercept = .3, lty = "dashed") +
```

```
geom_hline(yintercept = .6, lty = "dashed") +
scale_y_continuous(limits = c(0, 1)) +
theme_light()
```