Bonus

a. Construiţi o funcţie frepcomgen care primeşte ca parametri m şi n şi care generează un tabel cu repartiţia comună a v.a. X şi Y incompletă, dar într-o formă în care poate fi completată ulterior.

Pentru inceput generez aleator valorile $x_1, x_2, \dots, x_m$, si $y_1, y_2, \dots, y_n$ si ordonez crescator cele doua siruri. Creez o matrice cu n linii si m coloane si pastrez valorile pe pozitia denumirilor pentru linii, respectiv pentru coloane.

```
52 # Generez doua variabile si repartitia comuna incompleta a acestora
53 frepcomgen <- function(m, n) {
54   MAX = max(10, n, m)
55 
56   x <- sample((-MAX):MAX, m)
57   y <- sample((-MAX):MAX, n)
58   x <- x[order(x)]
59   y <- y[order(y)]
60 
61   mtx <- matrix(nrow = n + 1, ncol = m + 1,
62                 dimnames = list(c(y, " "), c(x, " ")))
63 
64   mtx <- generate_matrix(mtx, m, n)
65   mtx <- generate_q(mtx, m, n)
66 }
```

Pentru a construi matricea generez aleator valorile pentru fiecare π_{ij} si calculez suma acestora. Pe parcurs calculez si p_i respectiv q_j potrivite iar la final impart toate aceste valori la suma. S-a observat empiric ca daca incepem prin a calcula p_i si q_j pot aparea valori negative in interiorul tabelului, ceea ce il face, in fapt, imposibil de completat. Pentru usurinta citirii tabelului am decis sa aproximam valorile la 3 zecimale.

```
 4 # Generez valorile repartitiei comune
 5 generate_matrix <- function(mtx, m, n) {
 6   sum_y <- array(0, m)
 7   for (i in 1:n) {
 8     sum_x <- 0
 9     for (j in 1:m) {
10         aux <- sample(1:100, 1)
11         mtx[i, j] <- aux
12         sum_x <- sum_x + aux
13         sum_y[j] <- sum_y[j] + aux
14     }
15     mtx[i, m + 1] <- sum_x
16   }
17 
18   sum <- 0
19   for (i in 1:m) {
20     mtx[n + 1, i] <- sum_y[i]
21     sum <- sum + sum_y[i]
22   }
23 
24   mtx[n + 1, m + 1] <- sum
25   for (i in 1: (n + 1)){
26     for (j in 1: (m + 1))
27       mtx[i, j] <- ceiling(as.numeric(mtx[i, j]) / sum * 1000) / 1000;
28   }
29 
30   return (mtx)
31 }
```

Pentru a ne complica munca am decis sa lasam libere casutele de pe prima coloana a tabelului, de pe diagonala care incepe pe pozitia 1,1 si, daca m > n, de pe prima linie incepand cu coloana n + 1. Acestea se observa in functia *generate_q*.

b. Construiţi o funcţie fcomplrepcom care completează repartiţia comună generată la punctul anterior(pentru cazul particular sau pentru cazul general).

Functia cu nume prea complicat de rescris cauta o linie sau coloana pe care se afla un singur element necompletat, il completeaza, apoi se reapeleaza recursiv pana nu mai exista astfel de linii sau coloane existente. In acel moment am completat tabelul.

```
112 # Completez repartitia comuna din mtx
113 fcomplrepcom <- function(mtx, m, n) {
114   for (i in 1:(n + 1))
115     if (count_q(mtx, i, m, 0) == 1) {
116       mtx <- fix(mtx, i, m, 0)
117       mtx <- fcomplrepcom(mtx, m, n)
118       return (mtx)
119     }
120 
121   for (j in 1:(m + 1))
122     if (count_q(mtx, j, n, 1) == 1) {
123       mtx <- fix(mtx, j, n, 1)
124       mtx <- fcomplrepcom(mtx, m, n)
125       return (mtx)
126     }
127 
128   return (mtx)
129 }
```

Casutele necompletate de pe o linie / coloana sunt numarate cu ajutorul functiei *count_q*.

Pentru a completa o linie / coloana cautam pozitia elementului necunoscut.

```
91    h <- m + 1
92    sum <- 0
93    for (l in 1:m) {
94      if (pp == 0)  j = l
95      else  i = l
96 
97      if (mtx[i, j] == "?") h <- l
98      else  sum <- sum + as.numeric(mtx[i, j])
99    }
```

Acum avem 2 variante:

1. Elementul se afla la capat de linie / coloana (este un p_i sau un q_j), caz in care valoarea acestuia este egala cu suma celorlalte elemente de pe linie / coloana

```
101   if (h == m + 1) {
102     if (pp == 0)  mtx[k, h] = sum
103     else  mtx[h, k] = sum
```

2. Elementul se afla in interiorul liniei / coloanei, caz in care valoarea acestuia este egala cu diferenta dintre elementul de la capatul liniei / coloanei si suma celorlalte elemente cunoscute

```
104   } else {
105     if (pp == 0)  mtx[k, h] = as.numeric(mtx[k, m + 1]) - sum
106     else  mtx[h, k] = as.numeric(mtx[m + 1, k]) - sum
107   }
```

c. Având la dispoziţie repartiţia comună a v.a. X şi Y de la punctul b) calculaţi:

1. <u>Cov(5X,-3Y)</u>

Calculam covalenta folosind formula cov(X, Y) = E(XY) - E(X) * E(Y)

```
153 covalent <- function(mtx, m, n) {
154   val_x <- colnames(mtx)
155   val_y <- rownames(mtx)
156
157   Ex <- E(mtx, val_x, n, m, 0)
158   Ey <- E(mtx, val_y, m, n, 1)
159
160   Exy <- 0
161   for (i in 1:n){
162     for (j in 1:m){
163       Exy <- Exy + as.numeric(mtx[i,j]) *
164         as.numeric(val_x[j]) * as.numeric(val_y[i])
165     }
166   }
167
168   return (Exy - Ex * Ey)
169 }
```

E(X) si E(Y) sunt calculate separat cu ajutorul functiei E(), aflata la randul 137.

2. P(0 < X < 3 | Y > 2)

Calculez initial probabilitatea pentru variabile marginite: P(a < X < b), P(a < X), P((a < X < b) ∩ (Y > C)). Pentru a face acest lucru, adun π_{ij} pentru fiecare p_i si q_j in intervalul dat. Predefinim lowerbound-ul la valoarea minima posibila si upperbound-ul la valoarea maxima posibila pentru cazul in care nu sunt marginite variabilele in ambele capete.

```
171 # Calculez probabilitatea marginita pentru doua variabile aleatoare
172 probability <- function(mtx, m, n, lower_x = MINIM, upper_x = MAXIM,
173                         lower_y = MINIM, upper_y = MAXIM) {
174   val_x <- as.numeric(colnames(mtx))
175   val_y <- as.numeric(rownames(mtx))
176 
177   p <- 0
178   for (i in  1:m){
179     for (j in 1:n){
180       if (val_x[i] > lower_x && val_x[i] < upper_x &&
181           val_y[j] > lower_y && val_y[j] < upper_y)
182         p <- p + as.numeric(mtx[i,j])
183     }
184   }
185   return (as.numeric(p))
186 }
```

Pentru probabilitatea conditionata calculez probabilitatea intersectiei probabilitatii de sus cu a celei de jos, probabilitatea celei de jos si apoi le impartim.

```
171 # Calculez probabilitatea marginita pentru doua variabile aleatoare
172 probability <- function(mtx, m, n, lower_x = MINIM, upper_x = MAXIM,
173                         lower_y = MINIM, upper_y = MAXIM) {
174   val_x <- as.numeric(colnames(mtx))
175   val_y <- as.numeric(rownames(mtx))
176 
177   p <- 0
178   for (i in  1:m){
179     for (j in 1:n){
180       if (val_x[i] > lower_x && val_x[i] < upper_x &&
181           val_y[j] > lower_y && val_y[j] < upper_y)
182         p <- p + as.numeric(mtx[i,j])
183     }
184   }
185   return (as.numeric(p))
186 }
```

Problema efectiva se rezolva cu apelarea functiei *cond_probability* cu datele potrivite

```
209 # Subpunctul b
210 prob_b <- function(mtx, m, n) {
211   return (cond_probability(mtx, m, n, lower_x = 0, upper_x = 3, lower_cond_y = 2))
212 }
```

3. P(X > 6, Y < 7)

Consideram ca P(X > 6, Y < 7) = P(X > 6 ∩ Y < 7). Rezolvam problema apeland functia *probability* descrisa anterior.

```
214 # Subpunctul c
215 prob_c <- function(mtx, m, n) {
216   return (probability(mtx, m, n, lower_x = 6, upper_y = 7))
217 }
```

d. Pentru exemplul obţinut la punctul b) construiţi două funcţii fverind şi respectiv fvernecor cu ajutorul cărora să verificaţi dacă variabilele X şi Y sunt:

1. Independente

Variabilele aleatoare X si Y sunt independente daca $\pi_{ij} = p_i * q_j$ pentru oricare $i \leq n$ si oricare $j \leq m$. Asadar, functia fverind nu trebuie decat sa verifice daca se respecta aceasta proprietate.

```
223 # Verific daca variabilele din mtx sunt independente
224 fverind <- function(mtx, m, n) {
225   for (i in 1:n)
226     for (j in 1:m)
227       if (EPSILON < as.numeric(mtx[i, j]) -
228           as.numeric(mtx[i, m + 1]) * as.numeric(mtx[n + 1, j]))
229         return (0)
230 
231   return (1)
232 }
```

2. Necorelate

Variabilele aleatoare X si Y sunt necorelate daca cov(X, Y) = 0. Functia fernecor verifica aceasta proprietate.

```
234 # Verific daca variabilele din mtx sunt necorelate
235 fvernecor <- function(mtx, m, n) {
236   return (abs(covalent(mtx, m, n)) < EPSILON)
237 }
```

Intrucat calculele in R sunt inexacte, pentru a verifica cele doua proprietati folosim un epsilon suficient de mic.

In ultima faza programul apeleaza, pe rand, functiile descrise anterior si afiseaza rezultatele.

```
mtx <- frepcomgen(m, n)
print("Repartitia comuna incompleta a variabilelor generate:")
print(mtx)

mtx <- fcomplrepcom(mtx, m, n)
print("Repartitia comuna completa a variabilelor generate:")
print(mtx)

print("Cov(5X,-3Y)")
print(cov_a(mtx, m, n))

print("P(0<X<3/Y>2)")
print(prob_b(mtx, m, n))

print("P(X>6,Y<7)")
print(prob_c(mtx, m, n))

if (fverind(mtx, m, n)) {
  message(cat("Variabilele X si Y sunt independente"))
} else {
  message(cat("Variabilele X si Y nu sunt independente"))
}

if (fvernecor(mtx, m, n)) {
  message(cat("Variabilele X si Y sunt necorelate"))
} else {
  message(cat("Variabilele X si Y sunt corelate"))
}
```