# Bonus PS

*Livia Măgureanu, Radu Andreica, Diana-Michesa Roșu - 235*

# Cerință:

Fie două variabile aleatoare discrete $X$ şi $Y$ cu repartiţiile:
$X : \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ p_1 & p_2 & \dots & p_n \end{pmatrix}$ și, respectiv, $Y : \begin{pmatrix} y_1 & y_2 & \dots & y_m \\ q_1 & q_2 & \dots & q_m \end{pmatrix}$

1. Construiţi o funcţie `frepcomgen` care primeşte ca parametri $n$ și $m$ 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.
   ***Observaţie***: Se cere la a) să generaţi valorile lui $X$, valorile lui $Y$ şi suficient de multe valori pentru $p_i$, $qj$ şi, respectiv, $\pi_{ij}$ astfel încât să poată fi determinată repartiţia comună a celor doua v.a.
2. Construiţi o funcţie `fcomplrepcom` care completează repartiţia comună generată la punctul anterior.
3. Având la dispoziţie repartiţia comună a v.a. $X$ şi $Y$ de la punctul b. calculaţi:
   a. $Cov(5X, -3Y)$
   b. $P(0 < X < 3 \mid Y > 2)$
   c. $P(X > 6, Y < 7)$
4. Pentru exemplul obţinut la punctul b. construiţi două funcţii `fverind` şi respectiv `fvernecor` cu ajutorul cărora să verificaţi dacă variabilele $X$ şi $Y$ sunt:
   a. independente
   b. necorelate

# Rezolvare:

## Cerința 1.

Funcția

```
frepcomgen <- function(n, m) {
```

setează valorile lui $X$ ca fiind $1, 2, \dots n$ și pe ale lui $Y$ ca fiind $1, 2, \dots m$

```
  xv <- seq(1, n, by=1)
  yv <- seq(1, m, by=1)
```

generează numere cu din intervalul $(1, n + m)$ pentru valorile $\pi_{ij}$, și pe restul le completează cu $0$.

```
  xycomp <- round(runif(m, 1, n + m), 0)
  xycomp <- append(xycomp, 0)

  for (i in 2:n) {
    aux <- round(runif(m, 1, n + m), 0)
    xycomp <- append(xycomp, aux)
    xycomp <- append(xycomp, 0)
  }
  xycomp <- append(xycomp, replicate(m + 1, 0))
```

Ulterior, scalează valorile generate, împărțindu-le la suma lor

```
  sum <- sum(xycomp)
  xycomp <- xycomp / sum
```

și transfirmă repartiția comună în matrice.

```
  xycomp[n + 1, m + 1] = 1
  rownames(xycomp) <- append(xv, "q")
  colnames(xycomp) <- append(yv, "p")
```

Pentru ca repartiția să fie incompletă, sterge $\pi_{nm}$ (atribuindu-i valoarea $NaN$)

```
  xycomp[n, m] = NaN
```

și returnează repartiția astfel obținută.

```
  return(xycomp)
}
```

Pentru n = 7 și m = 9 generează:

```
> frepcomgen(7, 9)
      1     2     3     4     5     6     7     8     9     p
1 0.032 0.016 0.016 0.008 0.020 0.014 0.028 0.008 0.018 0.160
2 0.026 0.010 0.022 0.012 0.010 0.026 0.014 0.016 0.008 0.144
3 0.006 0.016 0.032 0.028 0.026 0.006 0.024 0.030 0.026 0.194
4 0.012 0.020 0.010 0.006 0.010 0.010 0.010 0.010 0.028 0.116
5 0.012 0.012 0.008 0.012 0.008 0.012 0.012 0.024 0.008 0.108
6 0.018 0.008 0.026 0.020 0.002 0.028 0.004 0.026 0.026 0.158
7 0.016 0.016 0.004 0.030 0.026 0.006 0.010 0.004   NaN 0.120
q 0.122 0.098 0.118 0.116 0.102 0.102 0.102 0.118 0.122 1.000
>
```

## Cerința 2.

Funcția

```
fcomplrepcom <- function(xycomp) {
```

inițial extrage dimensiunile celor două varibile aleatoare.

```
  n <- dim(xycomp)[1] - 1
  m <- dim(xycomp)[2] - 1
```

Cât timp există poziții completabile în tabel

```
  poz <- next_to_complete(xycomp)
  while(!is.nan(poz)) {
```

completează poziția respectivă în funcție de cum i se zice ca poate fi completată

```
    if (is.nan(poz[3])) {
      if (poz[1] == n + 1) {
        xycomp[poz[1], poz[2]] = sum(xycomp[poz[1], 1:m])
      } else if (poz[2] == m + 1) {
        xycomp[poz[1], poz[2]] = sum(xycomp[1:n, poz[2]])
      }
    } else if (identical(poz[3], "l")) {
      xycomp[poz[1], poz[2]] = xycomp[poz[1], m + 1]
                               - sum(xycomp[poz[1], 1:m], na.r
```

```
m=TRUE)
    } else {
      xycomp[poz[1], poz[2]] = xycomp[n + 1, poz[2]]
                               - sum(xycomp[1:n, poz[2]], na.r
m=TRUE)
    }
```

și cere următoarea poziție de completat.

```
   poz <- next_to_complete(xycomp)
 }
```

Când toate pozițiile au fost completate sau niciuna dintre cele completate nu poate fi calculata, se oprește și returnează repartiția comună obținută.

```
  return (xycomp)
}
```

Se poate observa că funcția `fcomplrepcom` se folosește de o altă funcție `next_to_complete` care găsește o poziție goală și completabilă

```
next_to_complete <- function(xycomp) {
  # Deminsiunile celor doua variabile
  n <- dim(xycomp)[1] - 1
  m <- dim(xycomp)[2] - 1

  # Cauta o pozitie completabila prin sume pe linii
  for (i in 1:(n + 1)) {
    row <- xycomp[i, ]
    first_NaN = find(is.nan(row), TRUE)
    last_NaN = find(is.nan(row), TRUE, last=TRUE)
    if (first_NaN == last_NaN && !is.nan(first_NaN)) {
      if (first_NaN != m + 1) {
        return(c(i, first_NaN, "l"))
      } else {
        return(c(i, first_NaN, NaN))
```

```
      }
    }
  }

  # Cauta o pozitie completabila prin sume pe coloane
  for (i in 1:(m + 1)) {
    col <- xycomp[, i]
    first_NaN = find(is.nan(col), TRUE)
    last_NaN = find(is.nan(col), TRUE, last=TRUE)
    if (first_NaN == last_NaN && !is.nan(first_NaN)) {
      if (first_NaN != n + 1) {
        return(c(first_NaN, i, "c"))
      } else {
        return(c(n + 1, i, NaN))
      }
    }
  }

  return(NaN)
}
```

și care, la rândul ei, se folosește de o altă funcție `find`.

```
find <- function(v, value, last=FALSE) {
  n <- length(v)
  if (last) {
    for (i in n:1) {
      if (v[i] == value) {
        return(i)
      }
    }
  } else {
```

```
    for (i in 1:n) {
      if (v[i] == value) {
        return(i)
      }
    }
  }
  return(NaN)
}
```

Pentru repartiția $xy$:

```
> xy
          1           2           3           4           5           6           7           8           9           p
1 0.019264448 0.010507881 0.024518389 0.024518389 0.019264448 0.01576182 0.028021016 0.02451839 0.021015762 0.18739054
2 0.021015762 0.021015762 0.007005254 0.005253940 0.015761821 0.02626970 0.017513135 0.01751313 0.026269702 0.15761821
3 0.014010508 0.026269702 0.008756567 0.008756567 0.008756567 0.02276708 0.022767075 0.02626970 0.015761821 0.15411559
4 0.007005254 0.024518389 0.005253940 0.001751313 0.007005254 0.01401051 0.017513135 0.01225919 0.007005254 0.09632224
5 0.010507881 0.028021016 0.008756567 0.005253940 0.014010508 0.01401051 0.005253940 0.00525394 0.026269702 0.11733800
6 0.015761821 0.014010508 0.021015762 0.007005254 0.028021016 0.02451839 0.026269702 0.02101576 0.005253940 0.16287215
7 0.003502627 0.007005254 0.012259194 0.019264448 0.024518389 0.01926445 0.003502627 0.02276708         NaN 0.12434326
q 0.091068301 0.131348511 0.087565674 0.071803853 0.117338004 0.13660245 0.120840630 0.12959720 0.113835377 1.00000000
```

va completa:

```
> xy <- fcomplrepcom(xy)
```

```
> xy
          1           2           3           4           5           6           7           8           9           p
1 0.019264448 0.010507881 0.024518389 0.024518389 0.019264448 0.01576182 0.028021016 0.02451839 0.021015762 0.18739054
2 0.021015762 0.021015762 0.007005254 0.005253940 0.015761821 0.02626970 0.017513135 0.01751313 0.026269702 0.15761821
3 0.014010508 0.026269702 0.008756567 0.008756567 0.008756567 0.02276708 0.022767075 0.02626970 0.015761821 0.15411559
4 0.007005254 0.024518389 0.005253940 0.001751313 0.007005254 0.01401051 0.017513135 0.01225919 0.007005254 0.09632224
5 0.010507881 0.028021016 0.008756567 0.005253940 0.014010508 0.01401051 0.005253940 0.00525394 0.026269702 0.11733800
6 0.015761821 0.014010508 0.021015762 0.007005254 0.028021016 0.02451839 0.026269702 0.02101576 0.005253940 0.16287215
7 0.003502627 0.007005254 0.012259194 0.019264448 0.024518389 0.01926445 0.003502627 0.02276708 0.012259194 0.12434326
q 0.091068301 0.131348511 0.087565674 0.071803853 0.117338004 0.13660245 0.120840630 0.12959720 0.113835377 1.00000000
```

# Cerința 3.

Funcția auxiliară `E` calculează media unei varibile aleatoare X dată sub formă de matrice de 2 linii.

```
E <- function(x) {
  return(sum(x[2, ] * x[1, ]))
}
```

Funcția `Cov` calculează $Cov(aX, bY)$. În cazul în care $a$ și $b$ sunt $1$, va calcula

direct, folosind formula: $Cov(X, Y) = E((X - \mu_X)(Y - \mu_Y))$.
Altfel, apelează recursiv și returnează $a * b * Cov(X, Y)$.

```
Cov <- function(xycomp, a = 1, b = 1) {
  if (a != 1 || b != 1) {
    return(a * b * Cov(xycomp))
  }

  n <- dim(xycomp)[1] - 1
  m <- dim(xycomp)[2] - 1

  # Calculam variabila aleatoare obtinuta daca din X scadem E(X)
  x <- append(round(as.numeric(rownames(xycomp[1:n, ])), 0), xycomp[1:n, m + 1])
  x <- matrix(x, nrow=2, byrow=TRUE)
  x[1, ] <- x[1, ] - E(x)

  # Calculam variabila aleatoare obtinuta daca din Y scadem E(Y)
  y <- append(round(as.numeric(colnames(xycomp[, 1:m])), 0), xycomp[n + 1, 1:m])
  y <- matrix(y, nrow=2, byrow=TRUE)
  y[1, ] <- y[1, ] - E(y)

  answer <- 0
  for (i in 1:n) {
    answer <- answer + sum(xycomp[i, 1:m] * y[1,]) * x[1, i]
  }

  return(answer)
}
```

Spre exemplu, pentru repartiția comuna $xy$ de la b. $Cov(5X, -3Y)$ va fi:

```
> Cov(xy, 5, -3)
[1] 0.6643336
> 
```

Funcția `p1` calculează $P(0 < X < 3 | Y > 2) = \frac{P((0<X<3) \cap (Y>2))}{P(Y>2)}$.

```r
p1 <- function(xycomp) {
  n <- dim(xycomp)[1] - 1
  m <- dim(xycomp)[2] - 1

  a <- sum(xycomp[1:2, 3:m])
  b <- sum(xycomp[n + 1, 3:m])
  return(a / b)
}
```

Pentru repartiția comuna $xy$ de la b., va returna:

```
> p1(xy)
[1] 0.3513514
> 
```

Funcția `p2` calculează $P(X > 6, Y < 7) = P((X > 6) \cap (Y < 7))$.

```r
p2 <- function(xycomp) {
  n <- dim(xycomp)[1] - 1
  m <- dim(xycomp)[2] - 1

  return(sum(xycomp[7:n, 1:6]))
}
```

Pentru repartiția comuna $xy$ de la b., va returna:

```
> p2(xy)
[1] 0.08581436
> 
```

## Cerința 4.

Funcția `fverind` verifică dacă variabilele aleatoare $X$ și $Y$ sunt independente. Aceasta proprietate este echivalentă cu $\pi_{ij} = p_i * q_j,\ \forall\ i = \overline{1,n},\ j = \overline{1,m}$.

```r
fverind <- function(xycomp) {
  n <- dim(xycomp)[1] - 1
  m <- dim(xycomp)[2] - 1

  for (i in 1:n) {
    for (j in 1:m) {
      if (xycomp[i, j] != xycomp[i, m + 1] * xycomp[n + 1, j])
{
        return(FALSE)
      }
    }
  }
  return(TRUE)
}
```

Funcția `fvernecor` verifică dacă variabilele aleatoare $X$ și $Y$ sunt necorelate. Aceasta proprietate este echivalentă cu $Cov(X,Y) = 0$.

```r
fvernecor <- function(xycomp) {
  return(Cov(xycomp) == 0)
}
```

Pentru repartiția comuna $xy$ de la b., vor returna:

```
> fverind(xy)
[1] FALSE
> fvernecor(xy)
[1] FALSE
>
```