1.Functia frepcomgen(n, m):

```
frepcomgen <- function(n, m)
{
  joint_distribution = generate_random_joint_distribution(n, m)
  print("Repartitia comuna completa")
  print(joint_distribution)
  partial_joint_distribution = erase_some_values(joint_distribution = joint_distribution)
  print("Repartitia comuna incompleta")
  print(partial_joint_distribution)
  return(partial_joint_distribution)
}
```

a) generate_random_joint_distribution

```
x = c(sample(1:(10 * n), n, rep=FALSE))
y = c(sample(1:(10 * m), m, rep=FALSE))
x = sort(x)
y = sort(y)
aux = matrix(sample(1:(n*m), n * m, replace = TRUE), nrow = n, ncol = m)
joint_distribution = matrix(0, nrow = n + 2, ncol = m + 2)
```

-imi generez in x si y valorile pentru cele doua variabile discrete
-in aux voi avea o matrice de NxM elemente pe care le voi insuma si apoi le voi normaliza si trece in joint_distribution astfel incat ele sa poata reprezenta o repartitie comuna(impart la suma lor initiala)

```
joint_distribution[i, j] = aux[i - 1,j - 1]
joint_distribution[i, j] = joint_distribution[i, j] / sum(aux)
```

-in joint_distribution voi avea pe prima linie si pe prima coloana, valorile lui y, respectiv valorile lui x; pe ultima linie si ultima coloana voi avea probabilitatile valorilor lui y, respectiv x

```
joint_distribution[i + 1, 1] = x[i]
joint_distribution[i + 1, m + 2] = rowSums(aux)[i]
joint_distribution[1, i + 1] = y[i]
joint_distribution[n + 2, i + 1] = colSums(aux)[i]
```

b) erase_elements
- parcurg elementetele matricei in spirala in sensul ceasului iar de fiecare data cand am un viraj, elementul din viraj il sterg

```
if (erase[posx, posy + 1] == TRUE){ # daca e liber in dreapta merg
  erase[posx, posy] = FALSE
  posy = posy + 1
} else if (erase[posx + 1, posy] == TRUE) { # daca nu e liber in dreapta merg in jos
  joint_distribution[posx, posy] = -1
  d = 2
} else {
  joint_distribution[posx, posy] = -1 # daca nu pot nici in jos atunci ma opresc
  ok = FALSE
}
```

- la fel si pentru celelalte directii doar ca se schimba a doua directie ca optiune :
  - in jos → in stanga
  - in stanga → in sus

- in sus → in dreapta

2. Functia fcomplrepcom
-cat timp la pasul anterior au fost gasite noi elemente in matrice se face o noua iteratie prin matrice si se incearca aflarea unui nou element al matricei

- se gaseste un nou element al matricei atunci cand este singurul necunoscut de pe aceeasi linie sau coloana – functia solve-element
-pentru elementele care nu reprezinta probabilitatile varibilei X sau Y:

```
for (col in 2:(C - 1)) {
  if (A[r, col] != -1) {
    no_elem_line = 1 + no_elem_line
    sum_line = sum_line + A[r, col]
  }
}
if (no_elem_line == C - 3) {
  ans = A[r, C] - sum_line
  return(ans)
}
for (row in 2:(R - 1)) {
  if (A[row, c] != -1) {
    no_elem_col = 1 + no_elem_col
    sum_col = sum_col + A[row, c]
  }
}
if (no_elem_col == R - 3) {
  ans = A[R, c] - sum_col
  return(ans)
}
```

-pentru elementele care reprezinta probabilitatile ale valorilor lui X:
- ma uit pe aceeasi linie, daca el este singurul lipsa atunci este egal cu 1 - suma elementelor de pe linie
- ma uit pe aceeasi coloana, daca el este singurul lipsa atunci este egal cu suma elementelor de pe aceeasi coloana
-la fel si pentru Y

3. Functia care calculeaza Cov(5x, -3y)
- mai intai calculez covarianta variabilelor:

```
mean_xy = 0
for (row in 2:(nrow(joint_distribution) - 1)) {
  for (col in 2:(ncol(joint_distribution) - 1)) {
    mean_xy = (mean_xy + (joint_distribution[row, col] * joint_distribution[1, col] * joint_distribution[row, 1]))
  }
}
mean_x = 0
mean_y = 0
for (row in 2:(nrow(joint_distribution) - 1)) {
  mean_x = (joint_distribution[row, 1] * joint_distribution[row, ncol(joint_distribution)] + mean_x)
}
for (col in 2:(ncol(joint_distribution) - 1)) {
  mean_y = (joint_distribution[1, col] * joint_distribution[nrow(joint_distribution), col] + mean_y)
}
res = (mean_xy - (mean_x * mean_y) )
return(res)
```

- apoi in functie inmultesc pe res cu (-15)

4.Functia care calculeaza P(0<X<3/Y>2) si P(X>6,Y<7)

- P(0<X<3/Y>2) = P(0<X<3 ∩ Y>2) / P(Y > 2)

```
for (row in 2:(nrow(joint_distribution) - 1)){
  for (col in 2:(ncol(joint_distribution) - 1)) {
    if (joint_distribution[1, col] > 2) {
      s2 = s2 + joint_distribution[row, col]
      if (joint_distribution[row, 1] > 0 && joint_distribution[row, 1] < 3) {
        s1 = s1 + joint_distribution[row, col]
      }
    }
  }
}
res1 = s1 / s2
```

-P(X>6,Y<7) = P(X>6 ∩ Y<7)

```
for (row in 2:(nrow(joint_distribution) - 1)) {
  for (col in 2:(ncol(joint_distribution) - 1)) {
    if (joint_distribution[row, 1] > 6 && joint_distribution[1, col] < 7) {
      res2 = res2 + joint_distribution[row, col]
    }
  }
}
```

5. Functia fverind:

- pentru fiecare element al matricei ex: a_{ij} verific sa fie egal cu produsul dintre probabilitatea lui x_i si probabilitatea lui y_j, daca gasesc un element care nu indeplineste aceasta conditie atunci cele doua variabilele sunt dependente

```
for (row in 2:(R - 1)){
  for (col in 2:(C - 1)) {
    if (joint_distribution[row, col] %!=% (joint_distribution[row, C] * joint_distribution[R, col])) {
        print("Variabilele sunt dependente")
        return()
      }
    }
}
print("Variabilele sunt independente")
```

6. Functia fvernecor:

- -calculez indicele φ = cov(x,y)/sqrt(var(x) * var(y))
- -covarianta o calculez folosind functia definita si la subpunctul anterior
- -pentru varianta folosesc functia:

```r
variance_discrete_variable <- function(discrete_var) {
  mean_var = 0
  mean_var2 = 0
  for (col in ncol(discrete_var)) {
    mean_var2 = (mean_var2 + (discrete_var[1, col] ^ 2) * discrete_var[2, col])
    mean_var = (mean_var + discrete_var[1, col] * discrete_var[2, col])
  }
  res = mean_var2 - (mean_var ^ 2)
  return(res)
}
```