Pachet pentru lucrul cu variabile aleatoare continue - Documentatie

Bouruc Petru-Liviu Dumitrescu Delia-Ioana

Ganea Antonio-Alexandru Nanu Robert-Ionut

Februarie 2021

Exercitiul 1

Fiind data o functie f, introdusa de utilizator, determinarea unei constante de normalizare k. In cazul in care o asemenea constanta nu exista, afisarea unui mesaj catre utilizator

Calcularea unei constante de normalizare k, in raport cu o functie introdusa f, se realizeaza determinand inversul valorii integralei functiei f de la $-\infty$ la $-\infty$, adica $\frac{1}{\int_{-\infty}^{\infty} f(x)\,dx}$.

Pentru rezolvarea problemei, vom crea functia *normalizing_constant*, care primeste ca parametru o functie f si returneaza constanta de normalizare, daca aceasta exista, iar daca nu, afiseaza "Can't determine normalizing constant". Inmultind functia cu constanta de normalizare, vom obtine un pdf.

```
normalizing_constant <- function (f)
{
  tryCatch (
    (1/(integrate(f, -Inf, +Inf) $ value)),

    error = function (e) { print("Can't determine normalizing
    ↪  constant") }
  )
}
```

Exercitiul 2

Verificarea daca o functie introdusa de utilizator este densitate de probabilitate.

Pentru a verifica daca o functie este densitate de probabilitate, vom porni de la definitie si vom verifica proprietatile necesare pentru o astfel de functie.

Definitie: O variabila aleatoare X este *continua* daca si numai daca exista o functie $f(x)$ astfel incat oricare ar fi $c <= d$, avem:

$$P(c \leq X \leq d) = \int_c^d f(x)\,dx.$$

Functia f este numita *functia densitate de probabilitate*(**pdf**). Pdf satisface intotdeauna urmatoarele proprietati:

- $f(x) \geq 0$ (f este nenegativa).
- $\int_{-\infty}^{\infty} f(x)\,dx = 1$. Aceasta este echivalenta cu: $P(-\infty < X < \infty) = 1$.

Prin urmare, vom defini in R functia *isPdf*, care primeste 4 parametrii: functia pe care vrem sa o testam, limita inferioara pe care este definita, limita superioara si modul de testare, care va fi explicat mai jos.
Apelare: isPdf(functie, limita_inferioara, limita_superioara, test_with_integrate *sau* test_with_min)

Vom testa intai cea de-a doua proprietate, anume egalitatea integralei cu 1. Integrala nu va da o valoare egala fix cu 1, insa daca eroarea in raport cu 1 este mai mica decat 0.00001, vom considera integrala egala cu 1. Daca valoarea integralei este diferita de 1(cu o eroare de 0.00001) sau a aparut o eroare in timpul integrarii, functia nu este densitate de probabilitate si deci functia *isPdf* se opreste.

Daca am trecut cu bine de prima verificare, ramane sa testam pozitivitatea functiei. Acest lucru poate fi realizat prin una dintre cele 2 functii: *test_with_integrate* sau *test_with_min*.

Prima functie, *test_with_integrate*, va calcula pentru 100 de integrale definite pe capete pseudo-random(primul capat se afla in prima jumatate a intervalului, in timp ce al doilea se afla in a doua jumatate) de integrare daca sunt mai mari ca 0. Daca da, se considera ca functia este pozitiva si deci densitate de probabilitate. Aceasta abordare este utila, spre exemplu, in cazul integralelor cu un capat de integrare egal cu $-\infty$ sau ∞.

A doua functie, *test_with_min*, va calcula minimul functiei fie pe intervalul de definire(daca acesta este cuprins in -100 si 100), fie pe maximul dintre -100 si limita inferioara de definire, respectiv minimul dintr 100 si limita superioara de definire. Daca minimul este mai mare ca 0, se considera ca functia este pozitiva si deci densitate de probabilitate. Aceasta abordare este utila, spre exemplu, in cazul unui interval finit, nu foarte mare.

```
test_with_min <- function(f, lower, upper) {
  lower <- max(lower, -1000)
```

```
  upper <- min(upper, 1000)
  min_val <- 1e9
  for (i in seq(lower, upper, 1e-2))
    min_val <- min(min_val, f(i))
  return (min_val >= 0)
}

test_with_integrate <- function(f, left, right) {
  for (i in 1:100){
    l <- runif (1, left, left + (right-left)/2)
    r <- runif (1, (right-left)/2, right)
    res <- integrate(Vectorize(f), l, r)
    if(res$value < 0){
      print(res$value)
      return (FALSE)
    }
  }
  return (TRUE)
}

isPdf <- function (f, lower, upper, test_positive) {
  tryCatch(
    {
      i <- integrate(Vectorize(f), -Inf, Inf)
      #Daca eroare integrarii e mai mare ca 1e-5, atunci
      ↪  integrala nu este = 1
      if(abs(i$value - 1) > 1e-5)
        return (FALSE)
      else {
        #Daca am ajuns aici, integrala a fost buna si testam
        ↪  pozitivitatea
        left = lower;
        if(lower == -Inf)
          left = -1e9;
        right = upper;
        if(upper == Inf)
          right = 1e9;

        return (test_positive(f, left, right))

      }
    },
    error = function(cond) {
      #Daca am ajuns aici, a existat o problema la integrare si
      ↪  deci integrala nu este = 1
```

```
      message(paste("Error: ", cond))
      return (FALSE)
    }
  )
}

# a e pdf
a <- function(x) {
  if(x > 0)
    return (exp(-x))
  else
    return (0)
}
isPdf(a, -Inf, Inf, test_with_integrate)

# b nu e pdf
b <- function(x) {
  return (exp(x^2))
}
isPdf(b, 0, Inf, test_with_positive)

isPdf(tan, -Inf, Inf, test_with_integrate)
isPdf(exp, -Inf, Inf, test_with_integrate)
```

Exercitiul 5

Calculul mediei, dispersiei si a momentelor initiale si centrate pana la ordinul 4 (daca exista). Atunci cand unul dintre momente nu exista, se va afisa un mesaj corespunzator catre utilizator.

O variabila aleatoare se numeste discreta daca si numai daca ia numai valori isolate. Multimea valorilor unei variabile aleatoare discrete este cel mult numarabila. O variabila aleatoare se numeste continua daca valorile ei umplu un interval.

Pentru a calcula media pentru variabila continua X, vom folosi urmatoarea formula: $E(X) = \int_{\infty}^{-\infty} x f_x(x)\,dx$, unde $f_x(x)$ este pdf-ul. Vom crea functia *get_mean*, care calculeaza media dupa aceasta formula.

Pentru a calcula dispersia pentru variabila continua X, vom folosi formula $Var(X) = E(X^2) - E(X)^2$, unde $E(x)$ a fost calculata mai sus, iar $E(x^2) = \int_{\infty}^{-\infty} x^2 f(x)\,dx$. Vom crea functia *get_dispersion*, care calculeaza dispersia dupa aceasta formula.

Pentru a calcula momentul initial de ordin 'r' pentru variabila continua X, vom folosi formula: $E(X^n) = \int_{\infty}^{-\infty} x^n f_x(x)\,dx$. Vom crea functia *get_initial_moment*, care calculeaza momentul initial dupa aceasta formula.

Pentru a calcula momentul centrat de ordin 'r' pentru variabila continua X, vom folosi formula: $E((X-m)^n) = \int_{\infty}^{-\infty} (x-m)^n f_x(x)\,dx$. Vom crea functia *get_central_moment*, care calculeaza momentul central dupa aceasta formula.

```
get_dispersion <- function (f)
{
  # interior sunt variabilele care urmeaza a fi
  # adaugate in integrale
  # E(x)
  interiorEx <- function (x) { return (x * f(x)) }
  # E(x^2)
  interiorEx2 <- function (x) { return (x ^ 2 * f(x)) }

  part1 <- integrate(interiorEx, -Inf, +Inf) $ value

  part1final <- part1 * part1

  part2 <- integrate(interiorEx2, -Inf, +Inf) $ value

  return (sqrt(part2 - part1final))
}

f <- function (x)
{
  sigma <- 4
  miu <- 2
  1/(sigma * sqrt(2 * pi)) * exp((-1/2) * ((x - miu) / sigma) ^
  ↪  2)
}


get_mean <- function (f)
{
  interior <- function (x) { return (x * f(x)) }

  integrate(interior, -Inf, +Inf) $ value
}

get_central_moment <- function (f, r)
{
  m <- get_mean(f)
```

```
  interior1 <- function (x) { return ((x - m) ^ r) }

  interior2 <- function (x) { return (interior1(x) * f(x)) }

  tryCatch (

    (integrate(interior2, -Inf, +Inf)) $ value,

    error = function (e) { print("Error calculating central
    ↪  moment")}
  )
}

get_initial_moment <- function (f, r)
{
  interior <- function (x) { return (x ^ r * f(x)) }

  tryCatch (

    (integrate(interior, -Inf, +Inf)) $ value,

    error = function (e) { print("Error calculating inital
    ↪  moment") }
  )
}
```

Exercitiul 6

Calculul mediei si dispersiei unei variabile aleatoare $g(X)$, unde X are o repartitie continua cunoscuta iar g este o functie continua precizata de utilizator.

Pentru a rezolva problema, vom defini functia *mean_and_disp*, care primeste ca parametru functia continua g. *Apelare:* mean_and_disp(functie). Pentru a calcula media variabilei aleatoare $g(x)$, unde g este o functie continua, vom folosi definita, anume:

$$E(Y) = E(h(X)) = \int_{\infty}^{-\infty} g(x)pdf(x)\,dx.$$

Cum pdf-ul este cunoscut iar functia g este transmisa ca parametru de catre utilizator, nu ramane decat sa calculam valoarea integralei. Pentru dispersie, avem urmatoarea *definitie:* Fie X o variabila aleatoare continua cu media μ. Dispersia lui X este:

$$Var(X) = E((X - \mu)^2).$$

In practica, vom folosi o teorema a dispersiei pentru a o calcula, anume: $Var(X) = E(X^2) - (E(X))^2 = E(X^2) - \mu^2$. μ reprezinta media calculata anterior, astfel ca nu ne ramane decat sa calculam $E(X^2)$, care va fi egal cu $\int_{\infty}^{-\infty} g(x)^2 pdf(x)\, dx$.

```
mean_and_disp <- function(g){
  pdf_func <- function(x){dnorm(x, 1, 2)}

  #media
  prod <- function(x) { g(x) * pdf_func(x) }
  mean <- integrate(Vectorize(prod), lower = -Inf, upper = Inf)
  print("Mean")
  print(mean)

  #media2 = E(h[x]^2) - media^2
  prod2 <- function(x) { g(x) * g(x) * pdf_func(x) }
  mean2 <- integrate(Vectorize(prod2), lower = -Inf, upper = Inf)
  dispersion <- mean2$value - (mean$value)*(mean$value)
  print("Dispersion")
  print(dispersion)
}

g <- function(x) {2*x+3}
mean_and_disp(g)
```

Exercitiul 7

*Crearea unei funcții P care permite calculul diferitelor tipuri de probabilitatii asociate unei variabile aleatoare continue(similar functiei P din pachetul **discreteRV**)*

Pentru evaluarea condiitilor puse in interiorul functiei **P**, am creat o clasa proprie de obiecte numita "PDF", pentru care am scris o functionalitate specifica operatorilor "<" ">" "|" si %AND%.

Pentru a putea incepe calculul diferitelor probabilitati asociate variabilei aleatoare, este necesar sa trecem functia care descrie densitatea de probabilitate a variabilei prin functia "CV" :

$$f = CV(X)$$

$P((f > 100) \mid (f < 120))$

Astfel, expresia $(f > 100)$ se reduce la un obiect de tip **"CVCond"**.

Aceste obiecte de tip **"CVCond"**, ce reprezinta conditii, pot fi combinate cu operatorii "|" si %AND%.

Rezultatul acestor combinatii este un obiect de tip **"CVResult"**.

In principiu, functia **P** nu calculeaza nimic, ea doar este un mod omogen de a extrage valoarea din obiecte de tip CVCond / CVResult. Logica pentru calcul este parte a operatorilor implementati.

```
# Cerinta 7

#' Converts a function to a package-compatible function
#'
#' @param f A function
#' @return The function bound to the class PDF
#' @examples
#' a <- CV(f)
#' b <- CV(g)
CV <- function( f ){
  res <- f
  class(res) <- "PDF"
  return (res)
}

#' @export
outputCV <- function(number){
  res <- list()
  res $ value <- number
  class(res) <- "CVResult"
  return (res)
}

#' Converts a functiasdasdaon to a package-compatible function
#'
#' @param X A function
#' @param x A function2
#' @return The function basdasdound to the class PDF
#' @examples
#' a <- CV(f)
#' b <- CV(g)
"<.PDF" <- function(X, x){
  probability <- integrate(X, lower=-Inf, upper = x) $ value
  ret = list()
  ret $ value = probability
```

```
  ret $ interval = c(-Inf,x)
  ret $ func = X
  class(ret) <- "CVCond"

  return (ret)
}


#' @export
">.PDF" <- function(X, x){
  probability <- integrate(X, lower=x, upper = +Inf) $ value
  ret = list()
  ret $ value = probability
  ret $ interval = c(x, +Inf)
  ret $ func = X
  class(ret) <- "CVCond"

  return (ret)
}

#' @export
opposing <- function( int1, int2 ){
  if ( int1[1] == -Inf && int2[2] == +Inf )
    return (TRUE)
  if ( int1[2] == +Inf && int2[1] == -Inf )
    return (TRUE)
  return(FALSE)
}

#' @export
intersection <- function ( int1, int2 ){
  if ( int1[2] <= int2[1] )
    return ( c (0,0) )
  if ( int2[2] <= int1[1] )
    return ( c (0,0) )

  if ( int1[1] < int2[1] && int2[2] < int1[2] )
    return ( c ( int2[1], int2[2] ) )

  if ( int2[1] < int1[1] && int1[2] < int2[2] )
    return ( c ( int1[1], int1[2] ) )

  if ( int1[2] >= int2[1] && int2[1] < int1[1] && int1[2] >
↪  int2[2] )
    return ( c ( int1[1], int2[2] ) )
```

```
  if ( int2[2] >= int1[1] && int1[1] < int2[1] && int2[2] >
↪  int1[2] )
    return ( c ( int2[1], int1[2] ) )

  if( int1[2] == Inf && int2[2] == Inf ){
    return ( c ( max(int1[1],int2[1]), Inf ) )
  }
  if (int1[1] == -Inf && int1[1] == -Inf){
    return ( c (-Inf, min (int1[2], int2[2]) ) )
  }
}

#' @export
intersects <- function ( int1, int2 ){
  if ( int1[2] <= int2[1] )
    return (FALSE)
  if ( int2[2] <= int1[1] )
    return (FALSE)

  return (TRUE)
}

#' @export
"|.CVCond" <- function(vec1, vec2) {
  v1 = vec1 $ value
  v2 = vec2 $ value

  if (!(  identical( vec1 $ func, vec2 $ func ) ) ){
    noquote("Different PDF functions used")
  }

  int1 = vec1 $ interval
  int2 = vec2 $ interval

  if ( opposing( int1, int2 ) ){
    # opposing things like ( f > 90 ) | ( f < 100 )
    if ( intersects( int1, int2 ) ){
      newinterval = intersection(int1,int2)
      integration = integrate( vec1 $ func ,
                               lower=newinterval[1],
                               upper = newinterval[2]) $ value
      return ( outputCV( min ( integration / v2, 1 ) ) )
    } else { # things like ( f < 10 ) | ( f > 100 )
      return ( outputCV(0) )
    }
```

```
  }
  else { # things like ( f > 110 ) | (f> 90)

    if ( v2 == 0 ){
      noquote("Cannot divide by 0")
      return (undefined)
    }

    res = v1 / v2

    return ( outputCV ( min(res, 1) ) )
  }
}

#' @export
"%AND%" <- function(vec1, vec2) {
  v1 = vec1 $ value
  v2 = vec2 $ value

  if (!(  identical( vec1 $ func, vec2 $ func ) ) ){
    noquote("Different PDF functions used")
  }

  int1 = vec1 $ interval
  int2 = vec2 $ interval

  if ( intersects( int1, int2 ) ){
    newinterval = intersection(int1,int2)
    integration = integrate( vec1 $ func ,
                             lower = newinterval[1],
                             upper = newinterval[2])
    return (outputCV( integration $ value))
  } else { # things like ( f < 10 ) | ( f > 100 )
    return (outputCV(0))
  }
}

#' @export
P <- function ( cvRes ) {
  print( cvRes $ value )
}
```

Exercitiul 9

Generarea a n valori(unde n este precizat de utilizator) dintr-o repartiție de variabile aleatoare continue.

R contine o functie de generat numere la intamplare, uniform distribuite. Aceasta se numeste $runif$ si o vom folosi pentru a genera numerele la intamplare.

Pentru a rezolva cerinta, am creat o functie *utocdf* care transpune un numar intre 0 si 1 intr-un numar conform unei distributii date. Pentru aceasta operatie, avem numarul y intre 0 si 1, o functie f care reprezinta CDF-ul distributiei si trebuie sa gasim un numar y astfel incat: $f(y) = x$.

Putem sa gasim numarul y gasind radacina ecuatiei: $f(y) - x = 0$

Daca vrem sa desenam 10000 de numere generate in functie de distributia data de f, avem:

a = cdfFromPDF(f, -Inf)

plot(RGCDF(a,10000))

```
# Cerinta 9

#' @export
cdfFromPDF <- function(f, lowerBound){
  res = function(x){
    if ( x < lowerBound ){
      return (0)
    }
    integrate(f, lower=lowerBound, upper = x) $ value
  }
  class(res) <- "CDF"
  return(res)
}

#' Uniform 0-1 number to CDF-based number
#'
#' @param CDF A CDF
#' @param num A number
#' @return The number \code{num} mapped according to the
↪  \code{CDF}
#' @examples
#' utocdf(f, 0.3)
#' uto(f, 0.2)
```

```
utocdf <- function ( CDF, num ){
  uniroot( function(y) {CDF(y) - num}, c(0.5, 1),
↪  extendInt="yes") $ root
}

# Random Generator based on CDF
#' @export
RGCDF <- function (CDF, quantity){
  nums <- runif(quantity)
  sapply(nums, function(x) { utocdf(CDF, x) } )
}
```

Exercitiul 10

Calculul covariantei si coeficientului de corelatie pentru doua variabile aleatoare continue(Atentie:Trebuie sa folositi densitatea comuna a celor doua variabile). aleatoare!

Definitie: Covarianta este o masura a cat de mult 2 variabile aleatoare variaza impreuna.

Avand functia *pdf comuna* $f(x, y)$ pe domeniul $[a, b] \times [c, d]$, pentru a calcula covarianta, ne putem folosi de urmatoarea formula:

$$Cov(X, Y) = (\int_c^d \int_a^b xyf(x, y)\ dxdy) - \mu_X \mu_Y$$

Asadar, primul pas este de a calcula cele 2 medii, asociate pentru variabilele X si Y. Aceste medii sunt salvate in *meanx* si *meany*.
Urmatorul pas este sa calculam integrala dubla: aceasta va fi aplicata pe functia nou creata $InnerFun = xyf(x, y)$.
La final, am pus optiunea de printare a rezultatului.

Definitie: Unitatile de masura ale covariantei Cov(X, Y) sunt "unitatile lui X ori unitatile lui Y". Aceasta face grea compararea covariantelor: daca schimbam scalele, atunci se schimba si covarianta. Corelatia este un mod de a elimina scala din covarianta.

Asadar, pentru a calcula corelatia, folosim urmatoarea formula:

$$Cor(X, Y) = \frac{Cov(X,Y)}{\sigma_X \sigma_Y}$$

In calculul corelatieie, ne vom folosi de functia creata anterior pentru a lua covarianta. Astfel, ne raman de calculat doar deviatiile. In procesul de calcul a deviatiilor, vom afla mai intai dispersiile celor doua variabile, pe care le vom salva *dispx* si *dispy*, urmand ca deviatia sa fie $\sqrt{dispersia}$.

Un exemplu de input este functia:

```
f <- function(x, y) {
    return(x^2+y^2)
}
cov(f, 0, 1, 0, 1, 1)
cor(f, 0, 1, 0, 1)
```

Care produce urmatorul output:

```
> cov(f, 0, 1, 0, 1, 1)
[1] 0.07638889
> cor(f, 0, 1, 0, 1)
[1] 0.9593023
```

```
mean <- function(f, a, b) {
  prod <- function(x) { return (x * f(x)) }
  m <- integrate(prod, a, b)
  return(m$value)
}

marginal_x <- function(f, a, b) {
  return(Vectorize(function (y) {
    integrate(function(x) { f(x, y) }, a, b)$value
  }))
}
marginal_y <- function(f, c, d) {
  return(Vectorize(function (x) {
    integrate(function(y) { f(x, y) }, c, d)$value
  }))
}

common_mean_x <- function(f, a, b, c, d) {
  mcx <- integrate(function(x) {
    return(x*marginal_y(f, c, d)(x))
  }, a, b)
  return(mcx$value)
}
common_mean_y <- function(f, a, b, c, d) {
  mcy <- integrate(function(y) {
    return(y*marginal_x(f, a, b)(y))
  }, c, d)
  return(mcy$value)
}
```

```r
double_integ <- function(f, a, b, c, d) {
  i <- integrate(Vectorize(function(y) {
    integrate(function(x) { f(x, y) }, a, b)$value
  }), c, d)
  return(i$value)
}

varx <- function(f, a, b, c, d, mean) {
  i <- integrate(function(x) {
    return((x-mean)^2 * marginal_y(f, c, d)(x))
  }, a, b)
  return(i$value)
}
vary <- function(f, a, b, c, d, mean) {
  i <- integrate(function(y) {
    return((y-mean)^2 * marginal_x(f, a, b)(y))
  }, c, d)
  return(i$value)
}

Cov <- function(pdfcom, a, b, c, d, print = 1){
  #media in x
  meanx <- common_mean_x(pdfcom, a, b, c, d)
  #media in y
  meany <- common_mean_y(pdfcom, c, d, c, d)

  #covarianta
  InnerFun <- function(x, y) { return (x * y * pdfcom(x, y)) }
  covariance <- double_integ(InnerFun, a, b, c, d) - meanx*meany
  if(print == 1) {
    print(covariance)
  }
  return(covariance)
}

Cor <- function(pdfcom, a, b, c, d) {
  #calculam covarianta
  covariance <- cov(pdfcom, a, b, c, d, 0)
  #media in x
  meanx <- common_mean_x(pdfcom, a, b, c, d)
  #media in y
  meany <- common_mean_y(pdfcom, c, d, c, d)

  #dispersiile
```

```
  dispx <- varx(pdfcom, a, b, c, d, meanx)
  dispy <- vary(pdfcom, a, b, c, d, meany)

  #deviatiile
  devf <- sqrt(dispx)
  devg <- sqrt(dispy)

  #corelatia
  correlation <- covariance / (devf*devg)
  print(correlation)
}

f <- function(x, y) {
  return(x^2+y^2)
}
cov(f, 0, 1, 0, 1, 1)
cor(f, 0, 1, 0, 1)
```

Exercitiul 11

Pornind de la densitatea comuna a doua variabile aleatoare continue, construirea densitatilor marginale si a densitatilor conditionate

Pentru densitatea comuna continua f(x, y) cu domeniul $[a, b] \times [c, d]$, pdf-urile marginale sunt:

$f_X = \int_c^d f(x, y)\, dy,\ f_Y = \int_a^b f(x, y)\, dx$

```
pdfmX <- function(pdfcom, a, b, c, d) {
  function(x) {
    integrate(function(y) { pdfcom(x, y) }, c, d)$value
  }
}

pdfmX <- function(pdfcom, a, b, c, d) {
  function(y) {
    integrate(function(x) { pdfcom(x, y) }, a, b)$value
  }
}
```

Exercitiul 12

Construirea sumei si diferentei a doua variabile aleatoare continue independente(folositi formula de convolutie)

Fie doua variabile aleatoare X si Y. Suma acestora Z = X + Y este data de convolutia celor doua variabile.
Convolutia a doua variabile:

$$f_Z = \int_{-\infty}^{\infty} f(x) * (g(z-x))\, dx$$

Pentru diferenta, schimbam semnul in interiorul functiei:

$$f_Z = \int_{-\infty}^{\infty} f(x) * (g(x-z))\, dx$$

```
sum <- function(f, g) {
  function(h) (integrate(function(x) (f(x) * g(h-x)), -Inf,
  ↪  +Inf)$value)
}
difference <- function(f, g) {
  function(h) (integrate(function(x) (f(x) * g(x-h)), -Inf,
  ↪  +Inf)$value)
}
```