ContinuousRV

Proiectul 1

Construirea unui pachet R pentru lucru cu variabile aleatoare continue

Autori

Vasilescu Alexandru (234)
Nicolae Tudor (234)
Radulescu Mihai (234)

Introducere

Pachetul creat se numeste ContinuousRV (inspirat dupa discreteRV).

La baza acestuia sta clasa de tip S4 CRV, care este utilizata in majoritatea metodelor si functiilor definite in pachet. Majoritatea exercitiilor au fost implementate utilizand metode generice.

Probleme:

1. Gasirea unei constante de normalizare k pentru orice functie provenita de la utilizator. In caz ca o asemenea constanta nu exista, trebuie afisat un mesaj.

Solutia consta in integrarea functiei date de utilizator, pe un domeniu dat. Daca rezultatul integralei este 0 (constanta rezultata fiind Inf), afisam un mesaj cum ca nu exista constanta de normalizare. Altfel, constanta este 1 / integrala.

```
setGeneric(
  "findNormalizationConstant",
  function(fn, lower, upper) {
    standardGeneric("findNormalizationConstant")
  }
)

setMethod("findNormalizationConstant", signature = c("function",
"numeric", "numeric"), definition = function(fn, lower, upper) {
  tryCatch(
    {
      int <- integrate(fn, lower, upper)$value
```

```
      if (int == 0) {
        message("Nu exista constanta de normalizare")
        return(NaN)
      }

      return(1 / int)
    },
    error=function(cond) {
      message("Nu exista constanta de normalizare")
      return(NaN)
    }
  )
})
```

```
unnormalizedFn <- function(x) {x}

# Returns 2
findNormalizationConstant(unnormalizedFn, 0, 1)

# Returns NaN
findNormalizationConstant(unnormalizedFn, -Inf, Inf)
```

2. O functie care verifica daca o functie data de utilizator este un probability density function. Pentru asta am verificat cele 2 conditii pentru ca o funtie sa fie PDF, si anume:
    a. f(x) > 0, oricare ar fi x in domeniu
    b. Integrala pe domeniul functiei sa fie egala cu 1

Solutia consta in verificarea primei conditii mai intai, daca f(x) > 0. Pentru asta ne folosim de functia optimize din R, pentru gasirea empirica a minimului pe un interval in functia de verificat. Verificam daca acest minim gasit este mai mare ca 0, si daca nu este, returnam FALSE. Daca este, verificam daca integrala pe domeniu este 1, iar daca este, returnam TRUE, altfel FALSE.

```
setGeneric(
  "isPDF",
  function(fn, lower, upper) {
    standardGeneric("isPDF")
  }
)

setMethod("isPDF", signature = c("function"), definition = function(fn,
lower, upper) {
  ifelse(optimize(fn, interval=c(lower, upper)) < 0, return(FALSE),
```

```
return(integrate(fn, lower, upper)$value == 1))
})
```

```
nonPdfFn <- function(x) {x}

# Returns FALSE
isPDF(nonPdfFn, 0, 1)

pdfFn <- function(x) {2 * x}

# Returns TRUE
isPDF(pdfFn, 0, 1)
```

3. Crearea unui obiect generalizat pentru a utiliza variabile aleatoare continue.

Solutia aleasa de noi a fost sa cream o clasa S4 numita CRV (dupa Continuous Random Variable). Constructorul acesteia ia o functie (nu neaparat PDF), si returneaza un obiect care are la baza functia utilizatorului, dar normalizata si domeniul pe care a fost definita aceasta. Totodata in obiectul S3 al domeniului, avem functia seq, care returneaza secventa dintre limita inferioara si superioara, cu pasul step.

Utilizam acest obiect in restul metodelor din proiect pentru simplificarea fracțiilor si a parametrilor.

```
setClass("CRV", representation(
  fn = "function",
  domain = "list",
  originalFn = "function"
))

CRV <- function(fn, lower, upper) {
  normalizationValue <- findNormalizationConstant(fn, lower, upper)

  new(
    "CRV",
    fn = function(x) {normalizationValue * fn(x)},
    originalFn = fn,
    domain = list(
      lower = lower,
      upper = upper,
      seq = function(step = 0.001) {seq(lower, upper, step)}
    )
  )
}
```

```
# Create a new CRV object
crv <- CRV(function(x) {x}, 0, 1)
```

4. Reprezentarea grafica a densitatii si a functiei de repartitie.

La acest exercitiu nu am implementat decat partial cerinta, si anume reprezentarea grafica a functiei de densitate (PDF) si a functiei de repartitie (CDF).

Implementarea functiei CDF consta in transpunerea formulei din curs in cod.

Pentru plotting, si pentru PDF dar si pentru CDF, modul in care procedam este faptul ca cream o lista de probabilitati (cu functia utilitara seq din obiectul S3 “domain” din CRV), si aplicam pe aceasta cu lapply PDF, respectiv CDF. Dupa, apelam functia de plot nativa, cu domeniul cu step de 0.001 si lista pe care a fost mapata respectiva functie.

```
setGeneric(
  "CDF",
  function(crv, x) {
    standardGeneric("CDF")
  }
)

setMethod("CDF", signature = c("CRV", "numeric"), definition =
function(crv, x) {
  integrate(crv@fn, crv@domain$lower, x)$value
})

setMethod("plot", signature = c("CRV"), definition = function(x) {
  # Aplicam PDF-ul variabilei pe domeniul ei si apelam plot-ul nativ
  P <- lapply(x@domain$seq(0.001), x@fn)
  plot(x@domain$seq(0.001), P)

  # Aplicam CDF-ul variabilei pe domeniul ei si apelam plot-ul nativ
  P <- lapply(x@domain$seq(0.001), CDF, crv = x)
  plot(x@domain$seq(0.001), P)
})
```

```
# Create a new CRV object
crv <- CRV(function(x) {x}, 0, 1)

# Returns 0.25
CDF(crv, 0.5)

# Plots both the PDF graph and the CDF graph
```

```
plot(crv)
```

5. Calculul mediei, disperisiei si a momentelor initiale si centrale pana la ordinul 4.

Pentru a le obtine, am aplicat formulele acestora, iar in cazul variatiei am calculat momentul central de ordinul 2.

In cazul mediei, punem calcularea integralei intr-un tryCatch, iar in cazul in care o eroare este aruncata, afisam un mesaj adecvat si returnam NaN.

Totodata, si in functia pentru calculul momentelor centrale, afisam un mesaj adecvat si returnam NaN, daca media returneaza NaN.

```
setMethod("mean", signature = c("CRV"), definition = function(x) {
  tryCatch(
    {
      return(integrate(function(x_) {x_ * x@fn(x_)}, x@domain$lower,
x@domain$upper)$value)
    },
    error=function(cond) {
      message("Nu exista medie pentru varibila data")
      return(NaN)
    }
  )
})

setGeneric(
  "centralMoment",
  function(crv, rank) {
    standardGeneric("centralMoment")
  }
)

setMethod("centralMoment", signature = c("CRV", "numeric"), definition =
function(crv, rank) {
  m <- mean(crv)

  if (is.nan(m)) {
    message(c("Nu exista moment central de ordinul ", rank))
    return(NaN)
  }

  integrate(function(x) {(x - m) ** rank * crv@fn(x)}, crv@domain$lower,
crv@domain$upper)$value
})
```

```
setGeneric(
  "initialMoment",
  function(crv, rank) {
    standardGeneric("initialMoment")
  }
)

setMethod("initialMoment", signature = c("CRV", "numeric"), definition =
function(crv, rank) {
  integrate(function(x) {x ** rank * crv@fn(x)}, crv@domain$lower,
crv@domain$upper)$value
})

setGeneric(
  "variance",
  function(crv) {
    standardGeneric("variance")
  }
)

setMethod("variance", signature = c("CRV"), definition = function(crv) {
  centralMoment(crv, 2)
})
```

```
# Returns 0.666...
mean(crv)

# Returns 0.055...
variance(crv)

# Returns 1
centralMoment(crv, 0)
# Returns approx 0
centralMoment(crv, 1)
# Returns 0.055...
centralMoment(crv, 2)
# Returns -0.007407407...
centralMoment(crv, 3)
# Returns 0.007407407...
centralMoment(crv, 4)

# Returns 1
initialMoment(crv, 0)
# Returns 0.666...
```

```
initialMoment(crv, 1)
# Returns 0.5
initialMoment(crv, 2)
# Returns 0.4
initialMoment(crv, 3)
# Returns 0.333...
initialMoment(crv, 4)
```

6. Aplicarea unei functii g, precizata de utilizator, asupra unei variabile aleatoare continue.

Pentru permiterea si simplificarea crearii unei astfel de functii, solutia gasita a fost punerea la dispozitie a unei functii utilitare, care ia ca parametru o functie cu un singur argument, x, si returneaza o functie apelabila, cu un singur argument, un obiect de tip CRV, asupra caruia va aplica functia originala a utilizatorului (este frumoasa programarea functionala).

```
prepareFunction <- function(fn) {
  function(crv) {
    CRV((function(x) {fn(crv@fn(x))}), crv@domain$lower,
crv@domain$upper)
  }
}
```

```
# Create a new CRV object
crv <- CRV(function(x) {x}, 0, 1)

# Prepare the passed function to be used with CRVs
g <- prepareFunction(function(x) {x^2})

# Returns 0.75
mean(g(crv))
# Returns 0.0375
variance(g(crv))
```

7. Crearea unei functii P similara cu cea prezenta in discreteRV.

Pentru implementarea unei astfel de functie, am creat inca o clasa S4, CrvLogical. Aceasta clasa are ca slot-uri elementele comparate (care pot fi de tip CRV, numeric sau CrvLogical la randul lor), operatorul folosit (<, >, ==, & sau |), si valoarea computata a expresiei.

```
setClassUnion("CrvLogicalComparand", c("CRV", "numeric"))

setClass("CrvLogical", representation(
  comp1 = "CrvLogicalComparand",
```

```
  operand = "character",
  comp2 = "CrvLogicalComparand",
  value = "numeric"
))

setClassUnion("CrvLogicalComparand", c("CRV", "CrvLogical", "numeric"))

CrvLogical <- function(comp1, operand, comp2) {
  value <- 0

  if (class(comp1) == "CRV" && class(comp2) == "numeric") {
    if(operand == "<") {
      value <- CDF(comp1, comp2)
    } else if(operand == ">") {
      value <- 1 - CDF(comp1, comp2)
    }
  } else if (class(comp1) == "CrvLogical" && class(comp2) ==
"CrvLogical") {
    if (operand == "&") {
      value <- (comp1@value + comp2@value) - 1
      if (value < 0) {
        value <- 0
      }
    } else if (operand == "|") {
      value <- comp1@value + comp2@value
      if (value > 1) {
        value <- 1
      }
    }
  }

  new(
    "CrvLogical",
    comp1 = comp1,
    operand = operand,
    comp2 = comp2,
    value = value
  )
}
```

Pentru fiecare operator de interes cream o noua signatura a metodei predefinite de R, cu 2 parametrii, de tip CRV si numeric, respectiv CrvLogical si CrvLogical. La fiecare dintre operatori, tot ce facem este sa returnam un o instanta a clasei CrvLogical, iar constructorul acesteia se ocupa de restul.

Metoda P nu face decat sa extraga slot-ul value din singurul ei argument, un obiect de tip CrvLogical.

```
setMethod("<", signature = c("CRV", "numeric"), definition =
function(e1, e2) {
  CrvLogical(e1, "<", e2)
})

setMethod(">", signature = c("CRV", "numeric"), definition =
function(e1, e2) {
  CrvLogical(e1, ">", e2)
})

setMethod("==", signature = c("CRV", "numeric"), definition =
function(e1, e2) {
  CrvLogical(e1, "==", e2)
})

setGeneric(
  "P",
  function(event) {
    standardGeneric("P")
  }
)

setMethod("P", signature = c("CrvLogical"), definition = function(event)
{
  event@value
})

setMethod("&", signature = c("CrvLogical", "CrvLogical"), definition =
function(e1, e2) {
  CrvLogical(e1, "&", e2)
})

setMethod("|", signature = c("CrvLogical", "CrvLogical"), definition =
function(e1, e2) {
  CrvLogical(e1, "|", e2)
})
```

```
# Returns 0.1
P(X > 0.9)
# Returns 0.7
P(X < 0.7)
```

```
# Returns 0.8
P(X < 0.7 | X > 0.9)
# Returns 0.2
P(X > 0.7 & X < 0.9)

# Returns 0.53
P(X > 0.97 | X < 0.5)
# Returns 1
P(X < 0.97 | X > 0.5)
```

8. Afisarea unei fise de sinteza pentru anumite repartitii.

Reparitiile documentate sunt:

- Uniforma - uniformDistribution
- Exponentiala - exponentialDistribution
- Normala (Gaussiana) - gaussianDistribution sau normalDistribution
- Pareto - paretoDistribution

Pentru accesarea documentatiei acestora, apelam

```
help("uniformDistribution")
help("exponentialDistribution")
help("gaussianDistribution ") # sau help("normalDistribution")
help("normalDistribution")
```

11. Calcularea densitatilor marginale si a celor conditionate, pornind de la un PDF comun.

Acest exercitiu a fost implementat doar partial, mai exact doar densitatile marginale.

```
marginalByX <- function(fn, x, lower, upper) {integrate(fn, lower,
upper, x = x)$value}
marginalByY <- function(fn, y, lower, upper) {integrate(fn, lower,
upper, y = y)$value}
```

```
# Create a joint PDF
pdfComun <- function(x, y) {3/8*(x+2*y)^2}

# Returns 1.625
marginalByX(pdfComun, 1, 0, 1)

# Returns 2.375
marginalByY(pdfComun, 1, 0, 1)
```

Concluzii

Am observat ca variabilele aleatorii continue sunt, d.p.d.v. matematic, concerte si bine definite, dar cand sunt transpuse in cod, isi pierd natura exacta, din cauza necesitatii de aproximare a mai multor formule.

Totodata, am descoperit niste aplicatii surprinzatoare ale reparitiilor continue pentru lumea reala, precum aproximarea medie a marimii unui meteorit, sau aproximarea timpului de descompunere al unui izotop.

Din punct de vedere al limbajului utilizat in proiect, R a fost, in mod surprinzator, foarte placut de utilizat, fiind primul limbaj procedural pe care l-am folosit.