ANA CAROLINA GOMEZ

Aluna do Curso de Ciências Moleculares.

Meus Exercícios

exerc_1.r exerc_2.r exerc_3.r exerc_4.r

Proposta de Trabalho Final

Principal

Pretendo fazer um programa em R de integração usando método de Monte Carlo. Dada uma função R→R, assume-se que um ponto aleatório tem uma chance de estar abaixo da curva que é proporcional à área total sob a mesma curva. A área pode então ser calculada e será mais precisa quanto maior for o número de pontos na simulação.

Comentários

Daniel:

A proposta parece interessante mas gostaria de saber mais detalhes. Qual será a entrada de dados da função, como estes dados serão manipulados (imagino que fará simulações de modelos nulos) e qual será a saída (retorno ao usuário) da sua função?

Ana:

A entrada que o meu programa recebe é uma função matemática, por exemplo log(x^2 + 3*x + 17), o número de pontos da simulação, por exemplo 1000 e o intervalo de integração, por exemplo, [0,10]. Acho que colocarei também uma opção para a visualização da simulação, onde o gráfico da função dada aparece plotado no intervalo pedido, junto com os pontos da simulação à medida que ela acontece.

A saída será a estimativa da área sobre a curva do gráfico no intervalo dado, ou seja, no meu exemplo, a integral de $\log(x^2+3x+17)$ de 0 a 10. O erro da estimativa é tão maior quanto menor for o número de pontos da simulação, ainda vou verificar se é fácil calculá-lo e se for o caso, posso também exibir como saída uma estimativa do erro.

A simulação consiste em gerar pontos aleatórios (x,y) com x dentro do intervalo de integração dado e y de 0 ao máximo da função. A área sobre a curva será proporcional ao número de pontos da simulação que caem sobre ela. Se a função assumir valores negativos, calcula-se a área da porção negativa separadamente e se subtrai da área da porção positiva. Fica mais fácil de entender com figuras, o artigo da wikipédia pode ajudar:

http://en.wikipedia.org/wiki/Monte_Carlo_integration

ps. acabo de ver que o artigo é mais difícil de entender do que a minha explicação, mas tem uma figurinha na direita que eu acho que resume a ideia

Comentários ALe

Como disse rapidamente hoje, não vamos ter como avaliar a parte da integração de monte-carlos, nos falta embasamento matemático. Estamos atentos mais à função e se o algoritmo da função tem coerência. Acho que sua explicação depois do questionamento do Musgo ficou muito legal! Vamos integrar por simulação de números aleatórios… MARAVILHA! Manda ver! Parece um desafio bacana!

Comentários Paulo

Gosto tb, e estou curioso em ver como esta integração numérica vai se comportar com funções que tem valor muito próximos de zero na maior parte do intervalo. Considere se é possível ter mais argumentos de *tunning* alem no numero de pontos a sortear, para resolver este problema e outros. Tb é importante que a função tb retorne alguma informação sobre o erro, como outras funções de integração fazem.

Ana:

Paulo, acho que o método que eu vou usar não tem mais nenhum argumento de tunning possível, a integral pelo método de Monte Carlo depende do número de pontos. Talvez tenha algum erro envolvido com a natureza imperfeita dos cálculos feitos por um computador, mas eu não vou usar números significativos o suficiente para isso ser determinante.

Vou imprimir um erro sim. A minha função retorna a integral calculada e o erro. A integral “real” estará entre calculada-erro e calculada+erro.

ps: fiz a página de help em inglês pq o modelo estava em inglês, mas o meu código está comentado em português para facilitar a correção e compreensão. Está bom assim?

Trabalho Final

Help

```
integrateMC                package:Montecarlo                R Documentation

Monte Carlo integration with graphical exhibition

Description:

    A function that evaluates the integral of a given mathematical function
    on the given closed interval using Monte Carlo integration.

```

```
Usage:

    integrateMC(x)
    #Default
    integrateMC(x, from = 0, to = 1, n = 10000, graphical = FALSE)

Arguments:

    func: function to be integrated, using the usual mathematical operators
    and basic functions in R (*, /, exp, ^, sin, cos, sqrt, etc), constants
(any
    real number) and the variable x.

    from: the lesser endpoint of the integration interval.

    to: the greater endpoint of the integration interval.
    n: the number of random points generated and used in the simulation.
More
    points provide a more precise result.

    graphical: boolean option for displaying the simulation. The function
and the
    points in the simulation are exhibited in a graphic.

Details:
    Evaluates the integral of a one-variable function over a given interval.
Accepts
    any function using the four basic mathematical opperators (*, /, +, -)
and
    basic functions in R, such as sin, cos, tan, sqrt and exp, as well as
compositions
    of those. The function must have x as it's only variable.
    The integral is calculated based on the estimated probability of a
random point
    being between the curve and the x-axis on the given interval. The
function
    exhibits the estimate value of the integral and and error value. The
precise
    value of the integral will be between the calculated + error and
calculated - error.

Warning:

    For a number of points (n) of order of magnitude above 3 (1000 points),
graphical
    is not reccomended, since the simulation will take a large amount of
time.
    The function given must have x as it's only variable.

Author(s):
```

```
    Ana Carolina Ribeiro Gomez

References:

    Kalos, M. H., Whitlock, P. A., (2008) _Monte Carlo Methods_. Wiley-VCH,
77-106
    http://en.wikipedia.org/wiki/Monte_Carlo_integration

Examples:
    #simple calculation
    integrateMC(x^2 * sin(x+exp(2.4*x)))
    #graphical simulation
    integrateMC(3*x^2 + 4*x + 2, graphical = TRUE)
    #more precise estimate, with many points
    integrateMC(sin(x), from = 0, to = 2*pi, n = 1000000, graphical = FALSE)
```

Código

```
integrateMC <- function (func, from = 0, to = 1, n = 10000, graphical =
FALSE)
{
    ##### Verificando consistência de dados #####
    # Na primeira etapa, iremos avaliar se o argumento 'func'
    # realmente corresponde a uma função (Consistência de dados).

    # Primeiro, utilizamos a função 'substitute' para gerar uma
    # árvore de análise sintática da expressão contida em 'func'.

    sfunc <- substitute(func)

    if (sfunc == substitute(x))
            sfunc = substitute(1*x)

    # A etapa anterior é importante para realizar a validação
    # da expressão contida em 'func'. Na condição seguinte,
    # caso 'sfunc' não contenha uma expressão válida ou
    # caso a expressão contida em 'func' contenha uma variável
    # diferente de 'x', nossa função 'Montecarlo' levanta uma
    # mensagem de erro, indicando que a expressão fornecida não
    # é uma expressão válida.

    if (!(is.call(sfunc) && match("x", all.vars(sfunc), nomatch = 0L)))
            stop("invalid function")

    ##### Fim da verificação da consistência de dados #####
```

```
    ##### Cálculo do Máximo e Mínimo da função 'func' #####

    # Nesta etapa, iremos encontrar o máximo e o mínimo
    # da função matemática 'func', dentro do intervalo
    # [from,to].

    max <- -Inf
    min <- Inf
    for(i in c(0:n)) {
    x <- from + i*(to-from)/n
    y <- eval(sfunc)
    if(y>max)
        max<-y
    if(y<min)
        min<-y
    }

    if(min>0)
    min=0
    if(max<0)
    max=0

    ##### Fim do cálculo de Máximo e Mínimo da função 'func' #####

    # O trecho a seguir plota a curva que representa a função 'func'

    if(graphical)
    {
        x<-seq.int(from, to, length.out = 1000)
        y <- eval(sfunc, envir = list(x = x), enclos = parent.frame())
        plot(x, y, type = "l", ylim = c(min,max))
    curve(0*x,add=TRUE)  #Esta linha plota o Eixo x
        invisible(list(x = x, y = y))
    }




    ##### Início do cálculo de probabilidades utilizando o método de Monte
Carlo #####

    pontospos=0
    pontosneg=0

    for(i in c(0:n)) {

    #Gera um par de pontos aleatórios
    x <- runif(1,from,to)
    yrand <- runif(1,min,max)
```

```
    #Avalia a função no x aleatório encontrado
    y <-eval(sfunc)
    #Esta variável irá sinalizar se devemos plotar o ponto ou não
    flag=1


    #Se o ponto está entre a curva da função e o eixo x e é um ponto
positivo
    if(yrand>0 && yrand<y)
    {
        #Aumentamos nosso contador de pontos que compreendem a região
positiva
         pontospos<-pontospos+1

        # O trecho a seguir plota o ponto e indica que ele não precisa mais
ser plottado
        flag=0
        if(graphical)
            points(x,yrand,pch=20,col="red")
    }

    #Se o ponto está entre a curva da função e o eixo x, mas é negativo
    if(y<yrand && yrand < 0)
    {
        #Aumentamos nosso contador de pontos que compreendem a região
negativa
        pontosneg<-pontosneg+1

        # O trecho a seguir plotta o ponto e indica que ele não precisa mais
ser plottado
        flag=0
        if(graphical)
            points(x,yrand,pch=20,col="red")
    }
    # Caso o ponto ainda não tenha sido plottado, plottamos o ponto com
outra cor
    # para indicar que ele é um ponto fora da área da curva.

    if(flag==1 && graphical)
        points(x,yrand,pch=20,col="blue")
    }

    # A probabilidade de um ponto aleatório ser sorteado entre
    # a parte positiva da curva e o eixo x é aproximadamente de:
    ProbPositiva = pontospos/n

    # A probabilidade de um ponto aleatório ser sorteado entre
    # a parte negativa da curva e o eixo x é aproximadamente de:

    ProbNegativa = pontosneg/n
```

```
    # A área total do retângulo que compreende o nosso intervalo
    # de intregação e os extremos da função nesse intervalo é de:

    Areatotal <- (to-from)*(max-min)

    # A integral pode então ser estimada por:

    Integral <- Areatotal*ProbPositiva -Areatotal*ProbNegativa

    # O erro pode ser estimado por:

    Erro <- 1/sqrt(n)

    ##### Fim do cálculo da Integral utilizando o Método de Monte Carlo
#####

    ##### Impressão dos resultados encontrados: #####

    print(c("Estimated value:", Integral))
    print(c("Error:", Erro))

    return(c(Integral,Erro))

}
```

Programa

montecarlo.r