Trabalho final: Modelos de Linguagens de Programação

Estudo sobre a linguagem R

Izadora Dourado Berti, 275606, Letícia dos Santos, 275604

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brazil

{idberti,lsantos}@inf.ufrgs.br

Vídeo da apresentação: https://youtu.be/hXzRhsSxeaI

Resumo – *Esse trabalho busca apresentar e avaliar a linguagem R utilizando os conceitos estudados durante o semestre.*

1. Introdução

A linguagem R, nomeada com a inicial de seus criadores, busca ser uma linguagem de programação de fácil uso para pesquisadores de áreas que não são diretamente relacionadas a computação.

Esses pesquisadores precisam realizar experimentos que, geralmente, envolvem estatística e visualizar os resultados por meio de gráficos diversos de forma prática e intuitiva. Então, o aprendizado da linguagem precisa ser rápido, pois o foco da pesquisa não é a computação em si.

O aspecto da linguagem ser voltada para o usuário sem familiaridade com programação é cativante e esse foi um dos motivos de ser escolhida como assunto desse estudo.

Outro motivo é que a linguagem foi utilizada em outras cadeiras. Logo, foi necessário aprender sobre o seu funcionamento.

O estudo apresentado a seguir destaca os diferenciais da linguagem R e está dividido em histórico, funcionalidades da linguagem, estrutura de dados, avaliação de expressões, comandos de controle de fluxo, exemplos de uso e conclusões.

2. Histórico

Foi criado originalmente por Ross Ihaka e por Robert Gentleman no departamento de Estatística da Universidade de Auckland, Nova Zelândia. Atualmente é mantido por uma comunidade de colaboradores voluntários que contribuem com código fonte da linguagem e com a expansão de funcionalidades por bibliotecas. A linguagem R é largamente usada entre estatísticos e analistas de dados para desenvolver software de estatística e análise de dados. Pesquisas e levantamentos com profissionais da área mostram que a popularidade do R aumentou substancialmente nos últimos anos e a comunidade R é bem ativa, o que facilita no seu uso e compreensão.

3. Funcionalidades da linguagem

R é uma linguagem de programação multi-paradigma (com ênfase em programação funcional), dinâmica, fracamente tipada, voltada à manipulação, análise e visualização de dados. Além disso, é interpretada tipicamente através de um Interpretador de comandos.

A sintaxe possuiu uma similaridade superficial com C, porém utiliza a semântica de uma linguagem de programação funcional com grande afinidade com Lisp e APL. Esse ambiente permite computação na linguagem, ou seja, é possível escrever funções que utilizam expressões como entrada, o que é útil para modelos estatísticos e gráficos.

O R é fortemente extensível através do uso de pacotes enviados pelo utilizador para funções específicas ou áreas específicas de estudo. Devido à sua herança do S, o R possui fortes recursos de programação orientada por objetos, mais que a maioria das linguagens de computação estatística. Ampliar o R também é facilitado pelas suas regras de contexto lexical. Outra força do R são os gráficos estáticos, que podem produzir imagens com qualidade para publicação, incluindo símbolos matemáticos. Gráficos dinâmicos e interativos estão disponíveis através de pacotes adicionais. O R tem a sua própria documentação em formato LaTeX, a qual é usada para fornecer documentação de fácil compreensão, simultaneamente on-line em diversos formatos e em papel.

4. Estruturas de dados

Como o R é uma linguagem interpretativa, tudo executa dentro de um ambiente(*environment*) que consiste em *frame*, conjunto de pares símbolo-valor, e *enclosure*, ponteiro para outro ambiente incluso.

A linguagem trabalha com dados que podem ser referenciados com símbolos ou variareis. Os tipos principais são:

- NULL
- *Symbol*: nome da variável.
- *Closure*: função.
- *Logical*: valores lógicos.
- *Integer*: valores inteiros.
- *Double*: valores reais.
- *Complex*: valores complexos.
- *Character*: valores de caracteres.
- *Expression*: uma expressão.
- *List*: lista.

Os dados podem ser organizados em:

4.1 Vectors (vetores)

Podem ser representados como células contiguas de memória acessadas com um índice. Números individuais (exemplo: 4) e *strings*(“quatro”) são vetores de comprimento 1.

4.2 Matrix (matrizes)

São vetores de duas dimensões.

4.3 Lists(listas)

São vetores que podem conter elementos de tipos diferentes.

4.4 Objetos

Podem ser do tipo *symbo*l (nome do objeto) *expression* (expressão) ou *function* (função). Um aspecto interessante é que é utilizada avaliação tardia sobre as expressões. A funções possuem lista de argumentos, um corpo e ambiente. O código 1 apresenta um exemplo.

```
f <- function(x) {
y <- 10
return(x y)
}
```

Código 1 – Exemplo de função

A função f possui o argumento x, o corpo está entre chaves e o ambiente possui a variável local y. O retorno é x+10.

Existem outros tipos de objetos mais avançados que não serão tratados nesse estudo.

5. Avaliação de expressões

Quando o usuário digita um comando no prompt ou quando uma expressão é lida de um arquivo, a informação é transformada para uma representação interna. O *evaluator* avalia e retorna um valor. Esse é o cerne da linguagem.

```
> 3
[1] 3
> y <- 4
> y
[1] 4
```

Código 2 – Exemplo de avaliação no *prompt*

De acordo com o código 2, a constante 3 obtém um vetor de tamanho 1 contendo 3 e a variável y é criada e inicializada com 4. O nome da variável em si é um *symbol*. Então, o *evaluator* retorna o valor associado.

Quando são chamadas funções, cada uma possui seu valor de retorno específico, como pode ser observado no código 1.

Além de operadores comuns, a linguagem suporta os seguintes operadores:

Tabela 1 – Operadores diferenciados

Operador	Descrição	Tipo
%%	Modulo	Binário
%/%	Divisão inteira	Binário
%*%	Produto de matrizes	Binário
%o%	Produto externo	Binário
%x%	Produto de Kronecker	Binário

Por conter características de uma linguagem funcional, a expressão x y é equivalente à chamada de função '+' (x, y), ou seja, produzem o mesmo resultado.

6. Comandos de controle de fluxo

Como a maioria das linguagens de programação o R possui estruturas de if, else, switch, for, while. Seus comandos peculiares são repeat e a família apply.

6.1 if

Quando observado o if, a linguagem R só irá executar o bloco entre chaves se a condição entre parênteses for verdadeira. Caso seja falsa, será executado o bloco seguinte ao else, caso exista else, ou contínua o fluxo do programa.

```
if(x < 0){
    sinal <- "negativo"
}else if(x == 0){
    sinal<-"neutro"
}else if(x > 0){
    sinal <- "positivo"
}
```

Código 3 – Exemplo if e else

6.2 switch

O comando condicional switch() permite desviar o código conforme o valor da chave. Assim, conforme o valor da chave, o programa executa o código associado à mesma, retornando ao código do script. Trata-se de um comando extremamente importante em programação, pois evita uma sequência de comandos condicionais if(), além de deixar o código mais claro e limpo.

Um exemplo do comando switch() é mostrado no código 4, onde a variável recebe o valor igual a um vetor de caracteres e o comando switch() retorna com um nome que corresponda exatamente valor que é avaliado. Se não houver correspondência, um único argumento sem nome será usado como padrão. E se nenhum padrão é especificado, NULL é retornado.

```
> y <- "fruit"
> switch(y, fruit =
"banana", vegetable =
"broccoli", "Neither")
[1] "banana"
> y <- "meat"
> switch(y, fruit =
"banana", vegetable =
"broccoli", "Neither")
[1] "Neither"
```

Código 4 – Switch executado no *prompt*

6.3 Loops: for, while e repeat

Já o comando for, o bloco é executado para todo valor da variável dentro do intervalo. Observando o código 5, i receberá o valor de 1 na primeira execução, 2 na segunda e assim sucessivamente até atingir n.

Para o comando while, o bloco será executado até a condição ser falsa. No código 6, o laço será interrompido quando $i > n$.

O comando repeat repete o bloco incondicionalmente. Para sair do laço, é utilizado break, como pode ser visto no código 7.

Então, considerando o seguinte problema: dado um valor de n gerar amostrar de tamanho 1, 2,…,n e calcule a média de cada amostra, com 3 casas decimais.

O problema foi resolvido de formas diferentes nos códigos 5, 6, 7 e 8.

```
f1 <- function(n) {
   medias <- numeric(n)
   #vetor com n números
   for (i in 1:n) {
     am <- rnorm(i)#retorna
    #amostra randômica
    #segundo a distribuição
    #normal de n valores
      medias[i] <-
      round(mean(am),dig=3)
      #arredondamento de 3
      #dígitos
   }
   return(medias)
}
```

Execução no prompt(mesmo resultado utilizando f2 e f3):

```
> set.seed(283)
> f1(10)
[1]  1.007 -0.063 -
0.392  1.546  0.341 -
0.514 -0.086 -
0.224  0.137  0.138
```

Código 5 – Solução com for

```
    f2 <- function(n) {
   medias <- numeric(n)
   i <- 1
   while (i <= n) {
            am <- rnorm(i)
      medias[i] <-

round(mean(am), dig = 3)
      i <- i           + 1
   }
   return(medias)
}
```

Código 6 – Solução com while

```
    f3 <- function(n) {
   medias <- numeric(n)
   i <- 1
   repeat {
            am <- rnorm(i)
      medias[i] <-

round(mean(am), dig = 3)
      if (i == n)     break
      i <- i +           1
   }
   return(medias)
}
```

Código 7 – Solução com repeat

6.4 Família apply

O R é uma linguagem vetorial e loops podem ser substituídos por outras formas de cálculo sempre que possível. Usualmente são utilizadas as seguintes funções para implementar cálculos de forma mais eficiente:

- apply() para uso em matrizes, arrays ou data-frames.
- tapply() para uso em vetores, sempre retornando uma lista.
- sapply() para uso em vetores, simplificando a estrutura de dados do resultado se possível (para vetor ou matriz).
- mapply() para uso em vetores, versão multivariada de sapply().
- lapply() para ser aplicado em listas.

7. Exemplos

Como a capacidade de construir gráficos facilmente é uma característica marcante da linguagem, os exemplos que serão apresentados envolvem gráficos da biblioteca *graphics*.

7.1 Gráfico de dispersão e histograma

```
(1) exemplo1<- function(){
(2)  vetor <- rnorm(1000,
(3)          mean=4, sd=2)
(4)  dev.new()
(5)  png(file="vetor.png")
(6)  plot(vetor)
(7)  dev.off()
(8)  hist(vetor)
(9) }
```

Código 8 – Função exemplo 1

Observando o Código 8, pode-se notar na linha 2 que vetor é inicializado com 1000 valores seguindo uma distribuição normal de média 4 e desvio padrão 2.

A função dev.new() inicia novo espaço de desenvolvimento na linha 4 e dev.off() termina na linha 7. O gráfico produzido na linha 6 com valores de vetor no eixo y e índice no eixo x será salvo em vetor.png como é definhado na linha 5.

O histograma de vetor criado na linha 8 será mostrado em uma nova janela.

As figuras 1 e 2 são o resultado de uma chamada exemplo1().

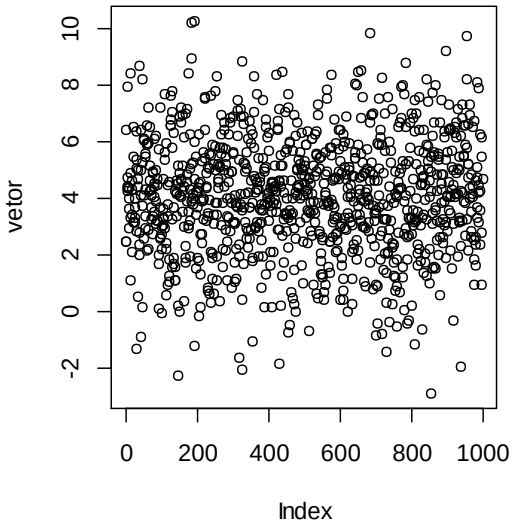

Figura 1 – Gráfico de dispersão de vetor, arquivo vetor.png

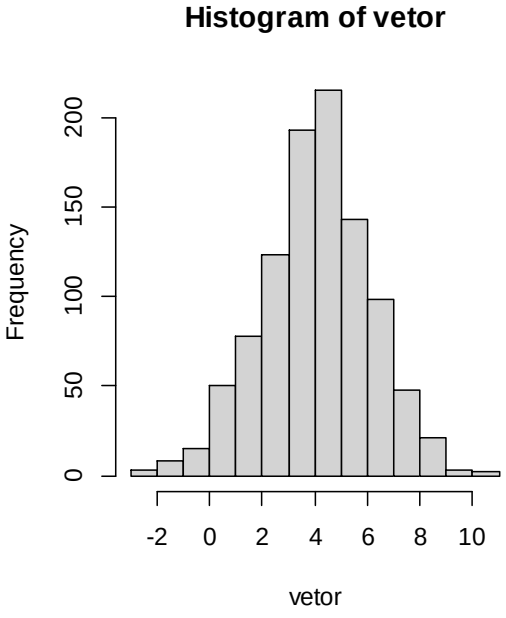

Figura 2 – Histograma de vetor

Com poucas linhas foi possível produzir gráficos com a formatação padrão de trabalhos acadêmicos.

7.2 Gráfico de setores

```
exemplo2<- function(){

valores<-c(21, 42, 53, 30)
rotulos<-("Calabresa",
"Milho", "Frango","Brócolis")
cores<-c("red", "yellow",
"orange", "green")

dev.new()
pie(valores, rotulos,
 main = "PIZZA", col = cores)
}
```

Código 9 – Função exemplo 2

A função c concatena as informações para formar os vetores valores, rolulos e cores no Código 9. Então, a função pie cria um gráfico com o título PIZZA e dados dos vetores.

A figuras 3 é o resultado de uma chamada exemplo2(). A facilidade de produzir e formatar gráficos é fascinante.

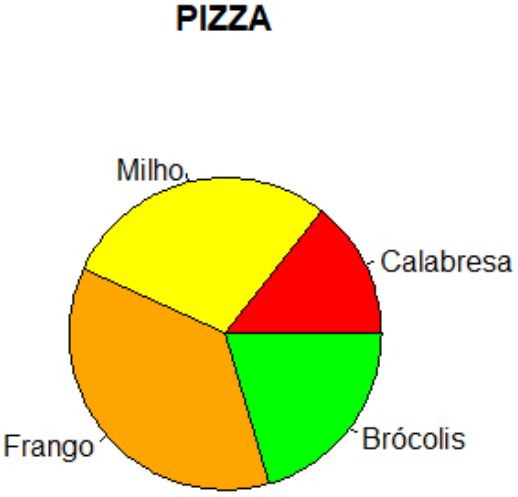

Figura 3 – Gráfico de setores

7.3 Fractal de Mandelbrot

Esse exemplo produz o fractal de Mandelbrot, que é descrito pela seguinte equação em que Z e c são matrizes:

$$\begin{cases} Z_0 = 0 \\ Z_n = (Z_{n-1})^2 + c, \quad n \geq 1;\ c \in \mathbb{C} \end{cases}$$

O resultado do exemplo será um gif com $1 \leq n \leq 20$. A imagem possui 600x600 pixels.

O gráfico gerado retrata a magnitude ou valor absoluto ou módulo de Z a cada interação. Uma função exponencial é utilizada para maximizar o Z, assim o fractal pode ser mais bem observado com poucas interações. As cores escalam de vermelho escuro, valor muito baixo, até azul escuro, valor muito alto.

O cálculo de Z utiliza a constante complexa c com a parte real variando de -1.8 até 0.6 e a parte imaginária de -1.2 até 1.2, amplitude de 2.4 em ambos os eixos. A matriz C seguirá o modelo:

$$C = \begin{bmatrix} -1{,}8 - 1{,}2i & \cdots & -1{,}8 + 1{,}2i \\ \vdots & \ddots & \vdots \\ -0{,}6 - 1{,}2i & \cdots & -0{,}6 + 1{,}2i \\ \vdots & \ddots & \vdots \\ 0{,}6 - 1{,}2i & \ldots & 0{,}6 + 1{,}2i \end{bmatrix}_{600\ x\ 600}$$

É importante notar que a parte real varia a cada linha e a parte imaginária a cada coluna, ou seja, o módulo de C (resultado da interação n=1) é uma matriz com um gradiente de valores altos nas extremidades até zero no elemento com o valor $0 + 0i$. Isso pode ser observado na primeira imagem do gif.

Esse conceito matemático complexo pode ser expresso em poucas linhas no código 10 que está na próxima página.

O exemplo 3 resulta no fractal disponível em:

https://pt.wikipedia.org/wiki/Ficheiro:Mandelbrot_Creation_Animation.gif.

```
(1)    exemplo3 <- function(){
(2)        library(caTools) #adiciona a biblioteca necessária
(3)        jet.colors = colorRampPalette(c("#00007F", "blue",
   "#007FFF", "cyan", "#7FFF7F", "yellow", "#FF7F00", "red",
   "#7F0000")) #configurações de cor, azul até vermelho
(4)
(5)        m=600#referência para tamanho do gif:600x600 pixels
(6)
(7)        C=complex(
(8)              real=rep(seq(-1.8,0.6,length.out=m),each=m),
(9)              imag=rep(seq(-1.2,1.2, length.out=m), m ) )
(10)         #define C
(11)
(12)       C=matrix(C,m,m) #coloca em forma de matriz
(13)       Z=0  #Z inicial
(14)       X=array(0, c(m,m,20))#onde será gravado o resultado
(15)
(16)       for (k in 1:20) { #20 interações
(17)           Z = Z^2+C
(18)           X[,,k] = exp(-abs(Z))
(19)       }
(20)       write.gif(X,"Mandelbrot.gif",
(21)                 col=jet.colors, delay=100)
(22)       #grava em gif com intervalo de 100ms
(23)   }
```

Código 10 – Função exemplo3

8. Conclusões

A linguagem R possui muitas qualidades, entre elas se destacam a capacidade de criar diferentes gráficos, expressar fórmulas matemáticas complexas de forma clara e compacta e facilidade de manuseio do interpretador.

Olhando de uma forma mais técnica, muitas linguagens funcionais, como o Prolog, possuem uma sintaxe de difícil entendimento, o que não acontece com o R. Pois a linguagem combina elementos do paradigma funcional, que facilitam o uso da matemática, com a sintaxe semelhante a C, que utiliza o paradigma procedural. A sintaxe do C foi criada a muitos anos e o fato de ser a base de várias outras linguagens comprova que é fácil entendê-la.

Então, o R possui várias aplicações e características únicas. Foi muito agradável fazer um estudo sobre a linguagem.

9. Referências

https://www.tutorialspoint.com/r/index.htm

https://pt.wikipedia.org/wiki/Ficheiro:Mandelbrot_Creation_Animation.gif

https://www.ufrgs.br/soft-livre-edu/software-educacional-livre-na-wikipedia/r-linguagem-de-programacao/

https://material.curso-r.com/rbase/

https://en.wikipedia.org/wiki/R_(programming_language)

http://www.leg.ufpr.br/~paulojus/embrapa/Rembrapa/Rembrapase29.html

Manual do R: https://cran.r-project.org/doc/manuals/