UNICAMP

Análise de Dados

Funções Predefinidas

Prof. Zanoni Dias

2020

Instituto de Computação – Unicamp

Roteiro

Constantes

Funções Predefinidas

Constantes

Constantes

- Em R existem algumas constantes internas predefinidas.
- `letters`: um vetor de caracteres com todas as letras minúsculas.

```
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
[11] "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
[21] "u" "v" "w" "x" "y" "z"
```

- `LETTERS`: um vetor de caracteres com todas as letras maiúsculas.

```
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
[11] "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T"
[21] "U" "V" "W" "X" "Y" "Z"
```

Constantes

- `month.abb`: um vetor de caracteres (strings) com as abreviaturas (com 3 letras) dos nomes dos meses (em inglês).

```
> month.abb
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun"
[7] "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

- `month.name`: um vetor de caracteres (strings) com os nomes dos meses (em inglês).

```
> month.name
 [1] "January" "February" "March" "April"
 [5] "May" "June" "July" "August" "September"
[10] "October" "November" "December"
```

Constantes

- `pi`: razão entre a circunferência e o diâmetro de um círculo (π).

```
> pi
[1] 3.141593
```

- Devemos tomar cuidado com essas "constantes", pois são implementadas como variáveis e seus valores podem ser sobrescritos.

```
> pi
[1] 3.141593
> pi <- 1000
> pi
[1] 1000
```

Funções Predefinidas

Funções Predefinidas

- Funções matemáticas básicas
- Funções trigonométricas
- Funções para sequências, vetores, listas e fatores
- Funções para conjuntos
- Funções estatísticas
- Funções probabilísticas
- Funções para strings
- Funções lógicas

Funções Matemáticas Básicas

- `sqrt()`: raiz quadrada.

```
> sqrt(c(1024, 49, 225))
[1] 32 7 15
```

- `exp()`: exponenciação (base natural).

```
> exp(c(2, 3, 4))
[1] 7.389056 20.085537 54.598150
```

- `factorial()`: fatorial.

```
> factorial(c(3, 5, 4, 6))
[1] 6 120 24 720
```

Funções Matemáticas Básicas

- `log()`: logaritmo.

```
> log(c(1, 1000, 1024))    # base e
[1] 0.000000 6.907755 6.931472
```

```
> log2(c(1, 1000, 1024))   # base 2
[1] 0.000000 9.965784 10.000000
```

```
> log10(c(1, 1000, 1024)) # base 10
[1] 0.0000 3.0000 3.0103
```

```
> log(1000, base = 10)
[1] 3
```

```
> log(1024, 2)
[1] 10
```

Funções Matemáticas Básicas

- `ceiling()`: arredondamento para cima.

```
> ceiling(c(-2.2, -1.8, 2.1, 2.7))
[1] -2 -1  3  3
```

- `floor()`: arredondamento para baixo.

```
> floor(c(-2.2, -1.8, 2.1, 2.7))
[1] -3 -2  2  2
```

- `trunc()`: truncamento.

```
> trunc(c(-2.2, -1.8, 2.1, 2.7))
[1] -2 -1  2  2
```

- `round()`: arredondamento.

```
> round(c(-2.2, -1.8, 2.1, 2.7))
[1] -2 -2  2  3
```

Funções Matemáticas Básicas

- `round()`: arredondamento.

```
> round(0.5:9)
[1] 0 2 2 4 4 6 6 8 8
```

- `signif()`: arrendondamento (com dígitos significativos).

```
> signif(x = pi, digits = 1:5)
[1] 3.0000 3.1000 3.1400 3.1420 3.1416
```

```
> signif(x = pi:10, digits = 3)
[1] 3.14 4.14 5.14 6.14 7.14 8.14 9.14
```

- `abs()`: valor absoluto.

```
> abs(-2.2:3)
[1] 2.2 1.2 0.2 0.8 1.8 2.8
```

Exercício

- Considerando a sequência numérica a seguir:

```
> x <- seq(1, 10, 0.25)
```

- Selecione todos os elementos de `x` que podem ser representados por um número inteiro, ou seja, cuja parte fracionária é igual a zero (exemplo: 1.0, 2.0, etc).

Funções Trigonométricas

- `sin()`: seno.
- `cos()`: cosseno.
- `tan()`: tangente.
- `asin()`: arco seno.
- `acos()`: arco cosseno.
- `atan()`: arco tangente.

Funções Trigonométricas

- Exemplos:

```
> sin(c(0, pi/4, pi/2, pi))
[1] 0.0 0.7071068 1.0 1.224647e-16
```

```
> cos(c(0, pi/4, pi/2, pi))
[1] 1.0 0.7071068 6.123234e-17 -1.0
```

```
> tan(c(0, pi/4, pi/2, pi))
[1] 0.0 1.0 1.633124e+16 -1.224647e-16
```

```
> acos(c(1, 0, -1))
[1] 0 1.570796 3.141593
```

Funções para Sequências, Vetores, Listas e Fatores

- `seq()`: cria uma sequência de números. Principais parâmetros:
  - `from`: número inicial.
  - `to`: número final.
  - `by`: intervalo entre os números da sequência.
  - `lenght.out`: tamanho da sequência desejada.
- `rep()`: replica os valores de um vetor ou de uma lista. Além do vetor ou da lista, os seguintes parâmetros podem ser fornecidos:
  - `times`: um vetor de inteiros indicando o número de vezes que cada elemento será repetido ou número de vezes que o vetor (ou a lista) será repetido, se um único número for fornecido.
  - `lenght.out`: tamanho do vetor (ou da lista) desejado.
  - `each`: número de vezes que cada elemento será repetido.

Funções para Sequências, Vetores, Listas e Fatores

- `gl()`: gera um vetor de fatores, sendo que os dois primeiros parâmetros indicam, respectivamente, o número de fatores e o número de réplicas de cada fator. Outros parâmetros podem ser fornecidos:
  - `lenght`: tamanho do vetor de fatores resultante.
  - `labels`: um vetor com os nomes de cada fator.
  - `ordered`: um valor booleano indicando se os fatores devem ser tratados como ordenados.
- `length()`: determina o número de elementos de um vetor, de uma lista ou de uma matriz. Pode ser usado para redefinir o tamanho de um objeto existente.
- `outer()`: dados dois vetores e o nome de uma função (ou de um operador binário) calcula o "produto" dos dois vetores em relação a função dada.

Funções para Sequências, Vetores, Listas e Fatores

- `sort()`: ordena um vetor de forma crescente. Para ordenar de forma decrescente, usar o argumento "`decreasing = TRUE`".
- `order()`: retorna a posição do menor elemento, do segundo menor elemento, e assim por diante, até do maior elemento de um vetor dado.
- `rank()`: retorna as posições relativas dos elementos de um vetor, em relação ao vetor ordenado.
- `rev()`: reverte a ordem dos elementos de um vetor.
- `unique()`: retorna um vetor sem os elementos repetidos.
- `duplicated()`: dado um vetor, indica quais os elementos estão repetidos.

Funções para Sequências, Vetores, Listas e Fatores

- `sum()`: soma dos valores de um vetor ou de uma estrutura tabular.
  - `rowSums()`: somas das linhas de uma estrutura tabular.
  - `colSums()`: somas das colunas de uma estrutura tabular.
- `prod()`: produto dos valores de um vetor ou de uma estrutura tabular.
- Funções cumulativas:
  - `cummin()`: mínimo cumulativo.
  - `cummax()`: máximo cumulativo.
  - `cumsum()`: soma cumulativa.
  - `cumprod()`: produto cumulativo.

Funções para Sequências, Vetores, Listas e Fatores

- Exemplos:

```
> x <- c(7, 2, 1, 6, 3, 4, 1, 5, 0, 8, 2); x
[1] 7 2 1 6 3 4 1 5 0 8 2
```

```
> length(x)
[1] 11
```

```
> sort(x)
[1] 0 1 1 2 2 3 4 5 6 7 8
```

```
> sum(x)
[1] 39
```

Funções para Sequências, Vetores, Listas e Fatores

- Exemplos:

```
> cummin(x)
[1] 7 2 1 1 1 1 1 1 0 0 0
```

```
> cummax(x)
[1] 7 7 7 7 7 7 7 7 7 8 8
```

```
> cumsum(x)
[1] 7 9 10 16 19 23 24 29 29 37 39
```

```
> cumprod(x)
[1] 7 14 14 84 252 1008 1008 5040 0 0 0
```

Exercícios

- Dado um vetor qualquer, liste os elementos que aparecem repetidos no vetor. Por exemplo, os elementos repetidos no vetor (1 2 1 5 3 4 1 5 2 8 1) são 1, 2 e 5.
- Dada uma permutação x dos n primeiros números naturais, por exemplo, gerada com `x <- sample(n)`, calcule a entropia de x. A entropia de uma permutação é igual a soma das distâncias de cada elemento para sua posição na permutação ordenada. Por exemplo, a entropia da permutação (3 2 4 5 1) é $2 + 0 + 1 + 1 + 4 = 8$.

Funções para Conjuntos

- `union()`: união.
- `intersect()`: intersecção.
- `setdiff()`: diferença de conjuntos.
- `setequal()`: igualdade de conjuntos.
- `is.element()`: verifica se um elemento pertence a um conjunto.

Funções para Conjuntos

- Exemplos:

```
> x <- c(1, 2, 3, 4, 5); x
[1] 1 2 3 4 5
> y <- c(3, 4, 5, 6, 7); y
[1] 3 4 5 6 7
```

```
> union(x, y)
[1] 1 2 3 4 5 6 7
```

```
> intersect(x, y)
[1] 3 4 5
```

Funções para Conjuntos

- Exemplos (continuação):

```
> x <- c(1, 2, 3, 4, 5)
> y <- c(3, 4, 5, 6, 7)
```

```
> setequal(x, y)
[1] FALSE
```

```
> setdiff(x, y)
[1] 1 2
```

```
> setdiff(y, x)
[1] 6 7
```

Funções para Conjuntos

- Exemplos (continuação):

```
> x <- c(1, 2, 3, 4, 5)
> y <- c(3, 4, 5, 6, 7)
```

```
> is.element(x, y)
[1] FALSE FALSE TRUE TRUE TRUE
```

```
> is.element(y, x)
[1] TRUE TRUE TRUE FALSE FALSE
```

```
> setequal(c(1, 2, 3), c(2, 3, 1))
[1] TRUE
```

Exercícios

- Dado um vetor qualquer, liste os elementos que não aparecem repetidos no vetor. Por exemplo, os elementos não repetidos no vetor (1 2 1 5 3 4 1 5 2 8 1) são 3, 4 e 8.

Funções Estatísticas

- `mean()`: média de um vetor.
  - `rowMeans()`: médias das linhas de uma estrutura tabular.
  - `colMeans()`: médias das colunas de uma estrutura tabular.
- `min()`: mínimo de um vetor ou de uma estrutura tabular.
- `max()`: máximo de um vetor ou de uma estrutura tabular.
- `range()`: mínimo e máximo de um vetor ou de uma estrutura tabular.
- `median()`: mediana de um vetor.

Funções Estatísticas

- `quantile()`: mínimo, primeiro quartil, mediana, terceiro quartil e máximo de um vetor.
- `summary()`: sumário dos dados de um vetor ou de uma estrutura tabular.
- `var()`: variância de um vetor.
- `sd()`: desvio padrão de um vetor.
- `cor()`: correlação (dois vetores ou uma estrutura tabular).

Funções Estatísticas

- Exemplos:

```
> x <- 1:5; x
[1] 1 2 3 4 5
```

```
> mean(x)
[1] 3
```

```
> sd(x)
[1] 1.581139
```

```
> cor((1:100)^2, log(1:100))
[1] 0.7785827
```

Funções Probabilísticas

- `runif()`: gera números aleatórios, dado um intervalo, considerando uma distribuição uniforme.
- `rnorm()`: gera números aleatórios, dado um valor médio e um desvio padrão, considerando uma distribuição normal.
- Para outros tipos de distribuições de números aleatórios, consulte `?Distributions`.
- `sample()`: dado um vetor (ou número inteiro `x`, indicando a sequência `1:x`), sorteia um número dado de elementos do mesmo.
  - É possível definir se o sorteio será com ou sem (padrão) reposição.
  - Também é possível especificar a probabilidade de cada elemento do vetor ser sorteado.

Funções Probabilísticas

- Exemplos:

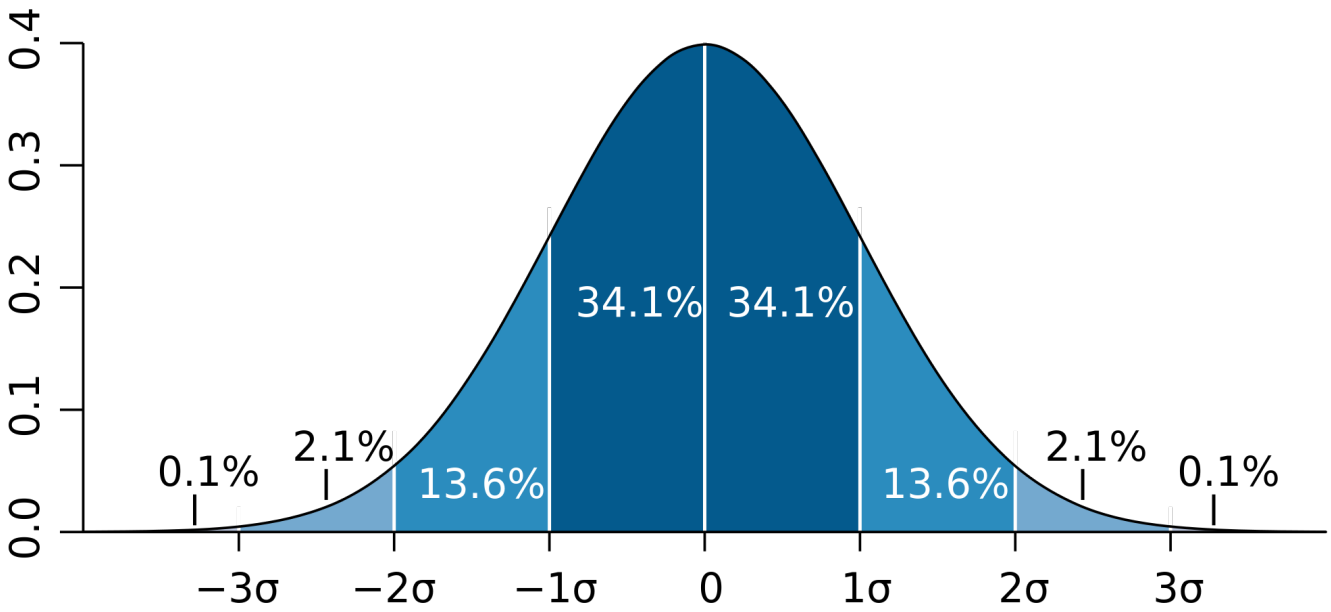
```
> runif(4, min = 0, max = 10)
[1] 4.8076390 4.9269143 6.3839444 0.1048800
```

```
> rnorm(4, mean = 0, sd = 10)
[1] 4.124341 -10.192727 -4.266127 8.168690
```

```
> sample(5)
[1] 4 1 2 3 5
```

```
> sample(1:60, 6)
[1] 44 37 23 51 13 27
```

Exercício



- Estimar a porcentagem de elementos que diferem (em módulo) de mais que um desvio padrão do valor médio em um vetor numérico qualquer que respeite, aproximadamente, uma distribuição normal.

Funções para Strings

- `nchar()`: retorna o número de caracteres de uma string.
- `substr()`: retorna uma substring, dadas uma string e as posições de início e fim. Também permite substituir a substring por outra dada.
- `strtrim()`: dada uma string, retorna um prefixo de tamanho também fornecido.
- `paste()`: concatena as strings fornecidas, usando (como padrão) um espaço em branco (“ ”) para unir as strings.
- `sub()`/`gsub()`: substitui um padrão de caracteres por outro numa string, fornecidos nesta ordem.

Funções para Strings

- `grep()`: dados um padrão de caracteres e um vetor de caracteres (strings), retorna os índices do vetor compatíveis com o padrão.
- `strsplit()`: divide uma string de acordo com padrão de caracteres.
- `toupper()`: transforma todas as letras de uma string em maiúsculas.
- `tolower()`: transforma todas as letras de uma string em minúsculas.
- `chartr()`: troca todas as ocorrências de um caractere por outro caractere numa string, fornecidos nesta ordem.

Funções para Strings

- Exemplos:

```
> x <- "Mineracao de Dados Complexos"
> y <- "Analise de Dados"
```

```
> nchar(c(x, y))
[1] 28 16
```

```
> strsplit(c(x, y), split = " ")
[[1]]
[1] "Mineracao" "de" "Dados" "Complexos"
[[2]]
[1] "Analise" "de" "Dados"
```

Funções para Strings

- Exemplos (continuação):

```
> x <- "Mineracao de Dados Complexos"
> y <- "Analise de Dados"
```

```
> substr(x, start = 11, stop = 18)
[1] "de Dados"
```

```
> substr(x, start = 11, stop = 18) <- y
> x
[1] "Mineracao Analise  Complexos"
```

```
> chartr("zenitpolar", "polarzenit", y)
[1] "Alinaso do Dides"
```

Funções para Strings

- Exemplos (continuação):

```
> x <- "MDC"
> y <- "Analise de Dados"
```

```
> strtrim(c(x, y), 7)
[1] "MDC" "Analise"
```

```
> paste(x, y, sep = " - ")
[1] "MDC - Analise de Dados"
```

```
> toupper(y)
[1] "ANALISE DE DADOS"
```

Funções para Strings

- Exemplos (continuação):

```
> x <- "Mineracao de Dados Complexos"
> y <- "Analise de Dados"
```

```
> sub("a", "*", y)
[1] "An*lise de Dados"
```

```
> gsub("os", "_-_", x)
[1] "Mineracao de Dad_-_ Complex_-_"
```

```
> grep("Dados", c(x, "MDC", y))
[1] 1 3
```

Funções Lógicas

- `any()`: dado um ou mais vetores de objetos lógicos, retorna verdadeiro (`TRUE`) se um dos objetos for verdadeiro (`TRUE`), caso contrário, retorna falso (`FALSE`).

```
> x <- c(F, T, F); x
[1] FALSE TRUE FALSE
```

```
> any(x)
[1] TRUE
```

```
> y <- c(F, F, F); y
[1] FALSE FALSE FALSE
```

```
> any(y)
[1] FALSE
```

Funções Lógicas

- `all()`: dado um ou mais vetores de objetos lógicos, retorna verdadeiro (`TRUE`) se todos os objetos forem verdadeiros (`TRUE`), caso contrário, retorna falso (`FALSE`).

```
> x <- c(T, T, F); x
[1] TRUE TRUE FALSE
```

```
> all(x)
[1] FALSE
```

```
> y <- c(T, T, T); y
[1] TRUE TRUE TRUE
```

```
> all(y)
[1] TRUE
```

A função `do.call()`

- Eventualmente queremos usar uma função que recebe como entrada vários parâmetros, separados por vírgulas, mas temos de fato um único vetor ou lista.
- Neste caso, podemos usar a função `do.call()`, que faz a chamada da função desejada, transformando cada elemento de uma lista de entrada num parâmetro da função.

A função `do.call()`

- Exemplo:

```
> param <- as.list(letters)
> param$sep <- ""
```

```
> param
[[1]]
[1] "a"
...
[[26]]
[1] "z"
$sep
[1] ""
```

```
> do.call(paste, param)
[1] "abcdefghijklmnopqrstuvwxyz"
```