

- [Tutorial](#)
- [Exercícios](#)
- [Apostila](#)

2. Funções Matemáticas e Estatísticas

O R como uma Calculadora Fora do Comum

Operações Aritméticas Básicas

A linha de comando do R funciona como uma calculadora. Todas operações aritméticas e funções matemáticas principais estão disponíveis. Exemplo:

```
> 4 + 9
[1] 13
> 4 - 5
[1] -1
> 4 * 5
[1] 20
> 4 / 5
[1] 0.8
> 4^5
[1] 1024
>
```

A notação básica de operações algébricas, como a aplicação hierárquica de parênteses, também pode ser utilizada:

```
> (4 + 5 ) * 7 - (36/18)^3
[1] 55
> (2 * ( 2 * ( 2 * (3-4))))
[1] -8
>
```

Note que somente os parênteses podem ser utilizados nas expressões matemáticas. As chaves (“{ }”) e os colchetes (“[]”) têm outras funções no R:

```
> (2 * { 2 * [ 2 * (3-4)]})
Error: syntax error in "(2 * { 2 * ["
>
```

Por que o R é uma calculadora **fora do comum** ? Experimente fazer a seguinte operação matemática na sua calculadora:

```
> 1 - (1 + 10^(-15))
```

Funções Matemáticas Comuns

As funções matemáticas comuns também estão disponíveis e podem ser aplicadas diretamente na linha de comando:

```
> sqrt(9)    # Raiz Quadrada
[1] 3
> abs( - 1 )  # Módulo ou valor absoluto
[1] 1
> abs( 1 )
[1] 1
> log( 10 )   # Logaritmo natural ou neperiano
[1] 2.302585
> log( 10, base = 10) # Logaritmo base 10
[1] 1
> log10(10)   # Também logaritmo de base 10
[1] 1
> log( 10, base = 3.4076) # Logaritmo base 3.4076
[1] 1.878116
> exp( 1 )    # Exponencial
[1] 2.718282
>
```

As funções trigonométricas:

```
> sin(0.5*pi)  # Seno
[1] 1
> cos(2*pi)    # Coseno
[1] 1
> tan(pi)      # Tangente
[1] -1.224647e-16
>
> asin(1)      # Arco seno (em radianos)
[1] 1.570796
> asin(1) / pi * 180
[1] 90
>
> acos(0)      # Arco coseno (em radianos)
[1] 1.570796
> acos(0) / pi * 180
[1] 90
> atan(0)      # Arco tangente (em radianos)
[1] 0
> atan(0) / pi * 180
[1] 0
>
```

Funções para arredondamento:

```
> ceiling( 4.3478 )
```

```
[1] 5
> floor( 4.3478 )
[1] 4
> round( 4.3478 )
[1] 4
> round( 4.3478 , digits=3)
[1] 4.348
> round( 4.3478 , digits=2)
[1] 4.35
>
```

Funções matemáticas de especial interesse estatístico:

```
> factorial( 4 )          # Fatorial de 4
[1] 24
> choose(10, 3)           # Coeficientes binomiais: combinação de 10 3-a-3
[1] 120
>
```

Criando Variáveis com Atribuição

Mais do que simples operações aritméticas, o R permite que executemos operações **algébricas** operando sobre variáveis pré-definidas.

Para definir uma variável, basta escolher um nome (*lembre-se das regras de nomes no R*) e atribuir a ela um valor:

```
>
> a = 3.6
> b = sqrt( 35 )
> c = -2.1
> a
[1] 3.6
> b
[1] 5.91608
> c
[1] -2.1
>
> a * b / c
[1] -10.14185
> b^c
[1] 0.02391820
> a + exp(c) - log(b)
[1] 1.944782
>
> a - b * c / d
Error: object "d" not found
>
```

Não esqueça de definir as variáveis previamente!!

Exercícios

Exercício 2.1. Estimador de Pollard

Pollard (1971) propôs o seguinte estimador para estimar a densidade no método de quadrantes:

$$\hat{N} = \frac{4(4n-1)}{\pi \sum_{i=1}^n \sum_{j=1}^4 r_{ij}^2}$$

onde, r_{ij} é a distância de árvore do quadrante j no ponto i ao centro do ponto quadrante e n é o número de pontos quadrantes.

A variância desse estimador é:

$$\text{Var}(\hat{N}_p) = \frac{\hat{N}_p}{4n-2}$$

Imagine que foram amostrados 30 quadrantes, e que o valor da soma do quadrado das distâncias de cada árvore ao centro de seu quadrante foi de:

$$\sum_{i=1}^{30} \sum_{j=1}^4 r_{ij}^2 = 2531,794$$

1. Qual a densidade estimada?
2. Qual a variância?

Exercício 2.2. Área transversal de uma Árvore

A área transversal de uma árvore é calculada assumindo que a secção transversal do tronco à altura do peito (1,3m) é perfeitamente circular. Se o diâmetro à altura do peito (DAP) de uma árvore for 13.5cm, qual a área transversal?

Se uma árvore possui três fustes com DAPs de: 7cm, 9cm e 12cm, qual a sua área transversal?

Exercício 2.3. Área transversal de uma Árvore (Revisitado)

Se uma árvore possui três fustes com DAPs de: 7cm, 9cm e 12cm, qual o diâmetro (único) que é equivalente à sua área transversal?

Exercício 2.4. Cálculo da Biomassa de Árvores do Cerrado

O modelo alométrico de biomassa ajustado para árvores do Cerradão estabelece que a biomassa é dada pela expressão:

$$\hat{b} = e^{-1,7953} d^{2.2974}$$

onde b é a biomassa em kg e d é o DAP em cm .

Já um outro modelo para biomassa das árvores na mesma situação tem a forma:

$$\hat{\ln(b)} = -2.6464 + 1,996\ln(d) + 0,7558\ln(h)$$

onde h é a altura das árvores em m .

Para uma árvore com DAP de $15cm$ e altura de $12m$, os modelos resultarão em estimativas muito distintas?

Mantendo a Coerência Lógica-Matemática

O R também lida com operações matemáticas que envolvem **elementos infinitos** e **elementos indeterminados**:

```
> 1/0
[1] Inf
> -5/0
[1] -Inf
> 5000000000000000000/Inf
[1] 0
> 0/0
[1] NaN
> Inf/Inf
[1] NaN
> log(0)
[1] -Inf
> exp(-Inf)
[1] 0
> sqrt(Inf)
[1] Inf
> sqrt( - 1 )
[1] NaN
Warning message:
NaNs produced in: sqrt(-1)
> 2 * NA
[1] NA
> 2 * NaN
[1] NaN
> NA / 10
[1] NA
> NaN / -1
[1] NaN
>
```

Note que determinadas **palavras** (além do nome das funções) estão reservadas no R, pois são utilizadas com significado especial:

- π - constante $\pi = 3.141593$;
- Inf - infinito;
- NaN - indeterminado (Not a Number), normalmente resultado de uma operação matemática indeterminada;

- NA - indeterminado (Not Available), normalmente caracterizando uma observação perdida (*missing value*).

Na operações matemáticas, NaN e NA atuam sempre como **indeterminado**.

2.5. Exercício Conceitual: Criando Variáveis com Nomes Reservados

O que acontece se você criar uma variável com o nome `pi`? Por exemplo,

```
> pi = 10
```

O que acontece com a constante `pi`?

E se for criada uma constante de nome `sqrt`? O que acontece com a função raiz quadrada (`sqrt()`)?

DICA: O que faz a função `search`, no comando:

```
> search()
```

Exercícios

2.6. Exercício Conceitual: O que é uma Observação Perdida

Como se caracteriza uma **observação perdida**?

Quando o diâmetro de uma árvore deve ter o valor **zero** ou o valor **NA**?

E o peso de um animal? E a biomassa de uma floresta? E a espécie de uma ave?

O R como uma Calculadora Vetorial

Criação de Vetores

O R, e a linguagem S, foram criados para operar não apenas *número-a-número* como uma calculadora convencional.

O R é um ambiente **vetorial**, isto é, quase todas suas operações atuam sobre um *conjunto de valores*, que genericamente chamaremos de vetores¹.

Uma definição mais detalhada dos vetores está [na seção sobre manipulação de dados](#). Aqui fornecemos apenas algumas definições e funções importantes para compreender as operações numéricas com vetores.

Concatenação de Elementos em um Vetor: a Função "c"

Para criar um vetor, podemos usar a função `c` (`c` = colar, concatenar). Essa função simplesmente junta todos os argumentos dados a ela, formando um vetor:

```
> a = c(1, 10, 3.4, pi, pi/4, exp(-1), log( 2.23 ), sin(pi/7) )
> a
[1] 1.0000000 10.0000000 3.4000000 3.1415927 0.7853982 0.3678794
0.8020016 0.4338837
>
```

Criação de Sequências: Operador ":" e Função "seq"

Para criar vetores de números com intervalo fixo unitário (intervalo de 1) se utiliza o *operador seqüencial* (`:`):

```
> b = 1:8
> b
[1] 1 2 3 4 5 6 7 8
> c = 20:32
> c
[1] 20 21 22 23 24 25 26 27 28 29 30 31 32
> d = 2.5:10
> d
[1] 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5
```

Uma forma mais flexível de criar seqüências de números (inteiros ou reais) é usando a função `'seq'`:

```
> seq(10, 30)
[1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
> seq(10, 30, by=2)
[1] 10 12 14 16 18 20 22 24 26 28 30
> seq(1.5, 7.9, length=20)
[1] 1.500000 1.836842 2.173684 2.510526 2.847368 3.184211 3.521053 3.857895
[9] 4.194737 4.531579 4.868421 5.205263 5.542105 5.878947 6.215789 6.552632
[17] 6.889474 7.226316 7.563158 7.900000
```

Vetores de Valores Repetidos: Função "rep"

Também é fácil criar uma seqüência de números repetidos utilizando a função `'rep'`:

```
> rep(5, 3)
[1] 5 5 5
> rep(1:5, 3)
[1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
> rep(1:5, each=3)
[1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
```

>

Exercícios

Exercício 2.7. *Palmeira com Muitos Fustes I*

Uma palmeira perfilhada possui 10 fustes com os seguintes diâmetros: 5, 6, 7, 5, 10, 11, 6, 8, 9 e 7.

Crie um vetor 'dap' com os diâmetros acima e uma sequência que enumera os fustes.

Vetores: Operações Matemáticas

Todas operações matemáticas aplicadas sobre um vetor, serão aplicadas sobre cada elemento desse vetor:

```
> 2 * a
[1] 2.0000000 20.0000000 6.8000000 6.2831853 1.5707963 0.7357589
1.6040032
[8] 0.8677675
> sqrt( a )
[1] 1.0000000 3.1622777 1.8439089 1.7724539 0.8862269 0.6065307 0.8955454
[8] 0.6586985
>
> log( a )
[1] 0.0000000 2.3025851 1.2237754 1.1447299 -0.2415645 -1.0000000
-0.2206447
[8] -0.8349787
>
```

Se as variáveis que trabalhamos são vetores, operações matemáticas entre variáveis serão realizadas pareando os elementos dos vetores:

```
> a* b
[1] 1.0000000 20.0000000 10.2000000 12.566371 3.926991 2.207277 5.614011
[8] 3.471070
> a - b
[1] 0.0000000 8.0000000 0.4000000 -0.8584073 -4.2146018 -5.6321206
-6.1979984
[8] -7.5661163
> a^(1/b)
[1] 1.0000000 3.1622777 1.5036946 1.3313354 0.9528356 0.8464817 0.9689709
[8] 0.9008898
>
> sqrt( a )
```

```
[1] 1.0000000 3.1622777 1.8439089 1.7724539 0.8862269 0.6065307 0.8955454
[8] 0.6586985
> log( b )
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
[8] 2.0794415
>
```

Comprimento de Vetores e a Função "length"

A função `length` retorna o número de elementos de um objeto:

```
> a <- seq(from=0, to=10, by=2)
> a
[1] 0 2 4 6 8 10
> length(a)
[1] 6
> length(1:20)
[1] 20
> length(rep(1:10,each=10))
[1] 100
>
```

A Regra da Ciclagem

O comprimento é muito importante para as operações vetoriais, pois o R permite operações entre dois vetores de comprimentos diferentes, com a seguinte regra:

Ciclagem de Valores

Operações entre vetores de comprimentos diferentes são realizadas pareando-se seus elementos. Os elementos do vetor mais curto são repetidos sequencialmente até que a operação seja aplicada a todos os elementos do vetor mais longo

Quando o comprimento do vetor maior não é múltiplo do comprimento do maior, o R retorna o resultado e um aviso:

```
> b
[1] 0 0 0 0 0 1 1 1 1 1
> c
[1] 1 2 3
> c*b
[1] 0 0 0 0 0 3 1 2 3 1
Warning message:
In c * b : longer object length is not a multiple of shorter object length
> length(b)
[1] 10
```

```
> length(c)
[1] 3
>
```

Mas se o comprimento do vetor maior é um múltiplo do maior, o R retorna apenas o resultado, sem nenhum alerta:

```
> a
[1] 1 2
> b
[1] 0 0 0 0 0 1 1 1 1 1
> a*b
[1] 0 0 0 0 0 2 1 2 1 2
> length(b)/length(a)
[1] 5
>
```

Portanto **muito cuidado com as operações entre vetores de diferentes comprimentos**. A regra da ciclagem é um recurso poderoso da linguagem R ²⁾, mas se você não tiver clareza do que deseja fazer, pode obter resultados indesejados.

Exercícios

Exercício 2.8. *Palmeira com Muitos Fustes II*

Uma palmeira perfilhada possui 10 fustes com os seguintes diâmetros: 5, 6, 7, 5, 10, 11, 6, 8, 9 e 7.

1. Calcule a área transversal de cada fuste dessa palmeira. Guarde este resultado em novo objeto.
2. Calcule a média das áreas transversais, sem usar a função `mean`.
3. Calcule a variância das áreas transversais, sem usar a função `var`

Exercício 2.9. *Bits e Bytes*

Como construir uma sequência que representa o aumento do número de bits por byte de computador, quando se dobra o tamanho dos bytes?

Essa sequência numérica parte do 2 e dobra os valores a cada passo.

Vetores: Operações Estatísticas

As funções matemáticas sobre vetores operam *elemento-a-elemento*. Já as funções estatísticas operam no vetor **como um todo**:

```
> mean( a )  
[1] 2.491344  
> var( b )  
[1] 6  
> max( c )  
[1] 32  
> sd( a )  
[1] 3.259248  
> sum( c )  
[1] 338  
> min( b )  
[1] 1  
> range( c )  
[1] 20 32  
>
```

Algumas funções úteis que não são estatísticas, mas operam no vetor são:

```
> a  
[1] 1.0000000 10.0000000 3.4000000 3.1415927 0.7853982 0.3678794  
0.8020016  
[8] 0.4338837  
> sort(a)  
[1] 0.3678794 0.4338837 0.7853982 0.8020016 1.0000000 3.1415927  
3.4000000  
[8] 10.0000000  
> rev(sort(a))  
[1] 10.0000000 3.4000000 3.1415927 1.0000000 0.8020016 0.7853982  
0.4338837  
[8] 0.3678794  
> cumsum(sort(a))  
[1] 0.3678794 0.8017632 1.5871613 2.3891629 3.3891629 6.5307556  
9.9307556  
[8] 19.9307556  
> cumsum(a)  
[1] 1.000000 11.000000 14.400000 17.54159 18.32699 18.69487 19.49687 19.93076  
> diff(a)  
[1] 9.0000000 -6.6000000 -0.2584073 -2.3561945 -0.4175187 0.4341221  
-0.3681178  
> diff( seq(10, 34, length=15) )  
[1] 1.714286 1.714286 1.714286 1.714286 1.714286 1.714286 1.714286 1.714286  
[9] 1.714286 1.714286 1.714286 1.714286 1.714286 1.714286  
>
```

Exercícios

Exercício 2.10. Conta de Luz

As leituras mensais do medidor de consumo de eletricidade de uma casa foram:

Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
9839	10149	10486	10746	11264	11684	12082	12599	13004	13350	13717	14052

1. Calcule o consumo de cada mês neste período.
2. Qual foi o máximo e mínimo de consumo mensal?
3. Qual a média, mediana e variância dos consumos mensais?

As Funções no R

Já foi visto que ao se digitar o nome de uma função na linha de comando, o R retorna o **código** da função. Veja a diferença de:

```
> ls()
```

para:

```
> ls
```

A maioria das funções precisa de certas **informações** para orientar o seu procedimento, tais informações são chamados de **argumentos**.

Os argumentos de qualquer função são detalhadamente explicados nas páginas de ajuda sobre a função. Mas para uma rápida consulta dos argumentos de uma função podemos usar a função 'args':

```
> args(ls)
function (name, pos = -1, envir = as.environment(pos), all.names = FALSE,
  pattern)
NULL
> args(q)
function (save = "default", status = 0, runLast = TRUE)
NULL
> args(save.image)
function (file = ".RData", version = NULL, ascii = FALSE, compress = !ascii,
  safe = TRUE)
NULL
>
```

Algumas funções, entretanto, são primitivas ou internas e seus argumentos não são apresentados. Geralmente, nesses casos os argumentos são bastante óbvios:

```
> args(sin)
```

```
NULL
> sin
.Primitive("sin")
>
```

Outras funções simplesmente não possuem argumentos:

```
> args(getwd)
function ()
NULL
> getwd
function ()
.Internal(getwd())
<environment: namespace:base>
>
```

Ao observar o resultado da função 'args', você notará que alguns argumentos são seguidos de uma expressão que se inicia com o sinal de igualdade ('='). A expressão após o sinal de igualdade é chamada de **valor default** do argumento. Se o usuário não informar o valor para um dado argumento, a função usa o valor default. Como exemplo veja a função 'save.image':

```
> args(save.image)
function (file = ".RData", version = NULL, ascii = FALSE, compress = !ascii,
  safe = TRUE)
NULL
>
```

Se o usuário simplesmente evocar a função 'save.image()', sem informar o nome do arquivo onde a área de trabalho deve ser gravada, o R gravará as informações num arquivo com nome '.RData'.

Exercícios

Exercício 2.11. Argumentos de Funções Estatísticas

Quais são os argumentos (e seus valores default) das seguintes funções:

- **mean**
- **sd**
- **range**
- **cumsum**

Exercício 2.12. Argumentos de Funções de Uso Comum

Quais são os argumentos (e seus valores default) das funções:

- **sort**
- **log**
- **seq**

O que é o argumento "..."?

Distribuições Estatísticas: Funções no R

Sendo um ambiente para análise de dados, o R dispõe de um grande conjunto de funções para trabalhar com *Distribuições Estatísticas*. Essas funções ajudam não só na análise de dados, como também permitem a *simulação* de dados.

Distribuição Normal

A distribuição Normal é a distribuição central da teoria estatística. Para gerar uma amostra de observações de uma distribuição normal utilizamos a função 'rnorm':

```
> args( rnorm )
function (n, mean = 0, sd = 1)
NULL
> vn1 = rnorm( 1000, mean = 40, sd = 9 )
> mean( vn1 )
[1] 39.47248
> sd( vn1 )
[1] 8.523735
> range( vn1 )
[1] 14.93126 62.11959
>
> vn2 = rnorm( 100000, mean = 40, sd = 9 )
> mean( vn2 )
[1] 40.02547
> sd( vn2 )
[1] 9.025218
> range( vn2 )
[1] 3.40680 78.25496
>
```

Se quisermos saber a *probabilidade acumulada* até um certo valor de uma variável com distribuição normal utilizamos a função 'pnorm':

```
> args(pnorm )
function (q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
NULL
>
> pnorm( 1.96, mean = 0 , sd = 1 )
[1] 0.9750021
> pnorm( 1.96 )
[1] 0.9750021
>
> pnorm( 27, mean = 20, sd = 7 )
[1] 0.8413447
> pnorm( 13, mean = 20, sd = 7 )
[1] 0.1586553
>
```

Se quisermos obter o valor de um *quantil* da distribuição normal utilizamos a função 'qnorm':

```
> args( qnorm )
function (p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
NULL
> qnorm( 0.90 )
[1] 1.281552
> qnorm( 0.30 )
[1] -0.5244005
>
> qnorm( 0.90, 20, 7)
[1] 28.97086
> qnorm( 0.30, 20, 7)
[1] 16.32920
>
```

A função 'dnorm' fornece a *densidade probabilística* para cada valor de uma variável Normal:

```
> args( dnorm )
function (x, mean = 0, sd = 1, log = FALSE)
NULL
> x = seq(-4, 4, length=10000)           # Sequência de -4 a 4 com 10.000
valores
>
> plot(x, dnorm(x))                     # Curva da Dist. Normal com média
0 e desvio padrão 1
> points(x, dnorm(x, sd=2))             # Curva da Dist. Normal com média
0 e desvio padrão 2 (adicionada ao gráfico)
>
```

Exercícios

Exercício 2.13 Amplitude Normal

Tomando uma variável que segue a Distribuição Normal, o que acontece com a *amplitude de variação* dos dados à medida que o tamanho da amostra cresce (por exemplo $n = 100, 1000, 10000$)?

Dica: use as funções `range` e `diff`

Exercício 2.14. Intervalo Normal I

Qual o intervalo da Distribuição Normal Padronizada que têm a média no centro e contem 50% das observações?

Exercício 2.15. Intervalo Normal II

Qual a probabilidade de uma observação da variável Normal Padronizada estar no intervalo $[-1.96, 1.96]$?

As Funções que Operam em Distribuições Estatísticas

O que foi apresentado para Distribuição Normal pode ser generalizado para todas as distribuições que o R trabalha.

Há quatro funções para se trabalhar com distribuições estatísticas:

- **ddistrib** - retorna a *densidade probabilística* para um dado valor da variável;
- **pdistrib** - retorna a *probabilidade acumulada* para um dado valor da variável;
- **qdistrib** - retorna o *quantil* para um dado valor de probabilidade acumulada;
- **rdistrib** - retorna *valores* (números aleatórios) gerados a partir da distribuição;

No caso da Distribuição Normal: *distrib* = norm. Para outras distribuições temos:

DISTRIBUIÇÕES ESTATÍSTICAS NO R		
Distribuição	Nome no R	Parâmetros ³⁾
beta	beta	shape1, shape2, ncp
binomial	binom	size, prob
Cauchy	cauchy	location, scale
qui-quadrado	chisq	df, ncp
exponential	exp	rate
F	f	df1, df2, ncp
gamma	gamma	shape, scale
geométrica	geom	prob
hypergeométrica	hyper	m, n, k
log-normal	lnorm	meanlog, sdlog
logística	logis	location, scale
binomial negativa	nbinom	size, prob
normal	norm	mean, sd
Poisson	pois	lambda
t de Student	t	df, ncp
uniforme	unif	min, max
Weibull	weibull	shape, scale
Wilcoxon	wilcox	m, n

Exercícios

Exercício 2.16. Teste *t*

Você realizou um teste *t* de Student bilateral e obteve o valor $t = 2.2$ com 19 graus de liberdade.

O teste é significativo ao nível de probabilidade de 5%? E se o valor observado fosse $t = 1.9$?

Exercício 2.17. Teste *F*

Você realizou um teste *F* e obteve o valor $F = 2.2$ com 19 graus de liberdade no numerador e 24 graus de liberdade no denominador.

O teste é significativo ao nível de probabilidade de 5%? E se o valor observado fosse $F = 2.5$?

Exercício 2.18. Padrão Espacial I

Gere duas amostras (p.ex.: x e y) de tamanho 1000 ($n=1000$) de números da distribuição Uniforme.

Faça um gráfico plotando uma amostra contra a outra (`plot(x,y)`). Qual o padrão espacial observado?

Você consegue explicá-lo?

Exercício 2.19. Padrão Espacial II

Gere duas amostras (p.ex.: x_p e y_p) de tamanho 10 ($n=10$) de números da distribuição Uniforme, com valor mínimo de zero e máximo de 100.

Gere duas amostras (p.ex.: x_f e y_f) de tamanho 1000 ($n=1000$) de números da distribuição Normal com média zero e desvio padrão 2)

Faça um gráfico plotando a soma das amostras X (x_p+x_f) contra a soma das amostras Y (y_p+y_f) (`plot(xp+xf, yp+yf)`).

Qual o padrão espacial observado? Você consegue explicá-lo?

Exercício 2.20. Gráfico Quantil-Quantil

Construa uma sequência **ordenada** de 1000 números entre 0 e 1:

```
> p = seq(0, 1, length=1000)
```

O vetor ' p ' representa um vetor de probabilidades acumuladas.

Gere 1000 números aleatórios da distribuição Normal com média e desvio-padrão 1 (um) e coloque os números em ordem:

```
> x = sort( rnorm(1000, mean=1) )
```

Faça um gráfico dos quantis da distribuição Normal, tomando o vetor ' p ' de probabilidades, contra os valores de ' x ':

```
> plot( qnorm(p, mean=1), x )
```

Como é o gráfico resultante?

Repita o mesmo processo para a distribuição Exponencial (' $rexp$ '), cujo valor *default* resulta em média = 1 . Como é o gráfico resultante? Por que?

Soluções dos Exercícios

[Aqui](#) você encontra os códigos que solucionam alguns dos exercícios propostos.

Se o seu código for diferente, não quer dizer necessariamente que errou. Compare os dois resultados! Como qualquer linguagem, o R é criativo: em muitos casos há mais de uma maneira de solucionar um problema.

1)

No R, “vetores” são uma classe de objetos definida simplesmente como conjuntos de elementos de um mesmo tipo. Os vetores do R não correspondem a vetores de valores da álgebra matricial, para os quais há outra classe de objetos, que é “matrix”

2)

A vantagem mais óbvia da regra da ciclagem é a possibilidade de multiplicação de um vetor por um valor único. Você compreende por que?

3)

os argumentos de cada função incluem estes parâmetros, entre outras coisas

From:

<http://ecor.ib.usp.br/> - **ecoR**

Permanent link:

http://ecor.ib.usp.br/doku.php?id=03_apostila:03-funcoes&rev=1597223092



Last update: **2020/08/12 06:04**