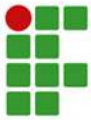


INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Bahia

CNPq
Conselho Nacional de Desenvolvimento
Científico e Tecnológico

GRAFOS E REDES COMPLEXAS: FUNDAMENTOS E MODELAGEM COMPUTACIONAL

PIBIC 2020

Dirceu Melo (Orientador) e Chel Freitas (Bolsista)

TUTORIAL R

O objetivo desse tutorial é ...

1ª Ed., Novembro de 2021

Sumário

I FUNDAMENTOS DE LINGUAGEM R

1 R e RStudio: primeiros passos 11

1.1 Como baixar e instalar 11

1.2 Área de Trabalho do RStudio 12

1.3 Pacote igraph 18

2 Objetos e comandos fundamentais do R 21

2.1 Como criar objetos na linguagem R 21

2.2 Vetores 22

2.3 Fator 24

2.4 Matrizes 25

2.5 Array 29

2.6 Data Frame 32

2.7 Lista 35

2.8 Função Plot 38

2.9 Função Hist 41

3 Estruturas de controle da linguagem R 43

3.1 Estrutura de Desvio Condicional 43

3.2 Estruturas de repetição 44

3.3 Função 46

II REDES: FUNDAMENTOS E MODELAGEM COMPUTACIONAL

4 Grafos e Redes 51

4.1 O que são? 51

4.2 De onde surgiu a ideia de redes? 52

4.3 Implementando grafos com o R 54

4.4 Grafos Notáveis 63

5 Características de Redes Complexas 69

5.1 Grau 69

5.2 Coeficiente de Aglomeração (transitividade) 71

5.3 Densidade 75

5.4 Caminhos e distâncias 78

5.5 Modularidade 82

5.6 Distribuição de graus 90

6 Tipologia de Redes Complexas 93

6.1 Grafos Aleatórios 93

6.2 Redes Small World 96

6.3 Redes Livres de Escala 100

III APLICAÇÕES

7 Aplicações 105

IV APÊNDICE

8 Apêndice 109

8.1 Distribuição de Frequências 109

8.2 Tipos de distribuições 111

8.3 Distribuição Normal 117
8.4 Distribuição Binomial 117
8.5 Distribuição de Poisson 119
8.6 Lei de potência 121

Bibliografia 125
Artigos 125
Livros 126
Manuais 126
Sites 126

Índice 129

FUNDAMENTOS DE LINGUAGEM R

1 R e RStudio: primeiros passos 11

1.1 Como baixar e instalar

1.2 Área de Trabalho do RStudio

1.3 Pacote igraph

2 Objetos e comandos fundamentais do R 21

2.1 Como criar objetos na linguagem R

2.2 Vetores

2.3 Fator

2.4 Matrizes

2.5 Array

2.6 Data Frame

2.7 Lista

2.8 Função Plot

2.9 Função Hist

3 Estruturas de controle da linguagem R 43

3.1 Estrutura de Desvio Condicional

3.2 Estruturas de repetição

3.3 Função

1. R e RStudio: primeiros passos

O R é uma linguagem de programação livre, gratuita, de código aberto, cuja principal funcionalidade é a realização de análise de dados e construção de gráficos.

O RStudio é um ambiente de desenvolvimento integrado para a linguagem R, e foi idealizado para trazer recursos facilitadores ao usuário na utilização das potencialidades do R. Maiores detalhes podem ser encontrados em [17].

1.1 Como baixar e instalar

O software do R pode ser baixado acessando o site `https://www.r-project.org/about.html`. Nesse site também encontra-se o repositório CRAN, que possui várias bibliotecas e pacotes fundamentais.

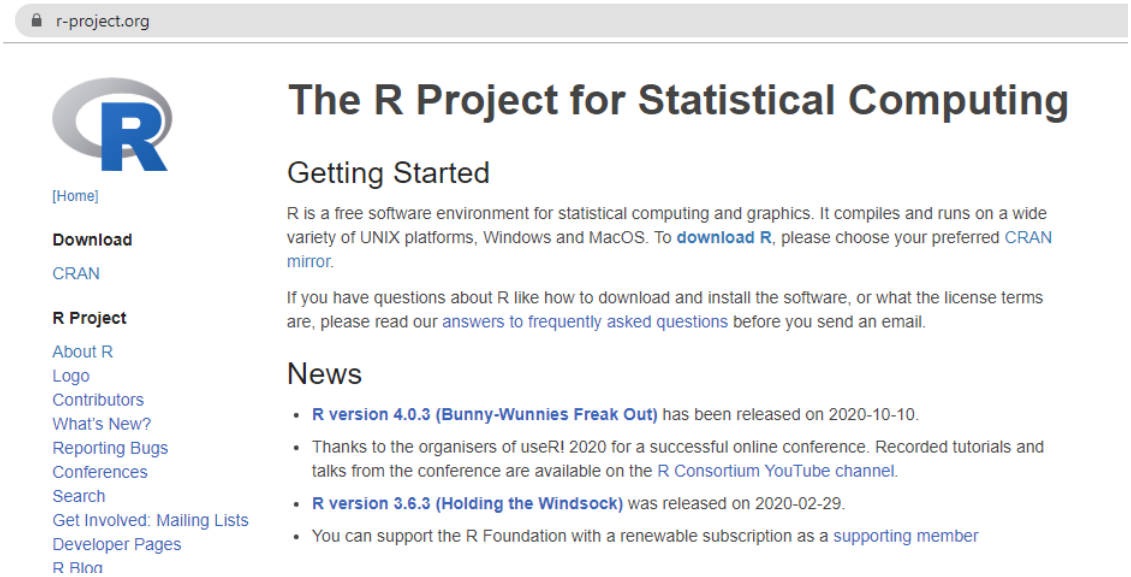


Figura 1.1: Página web do R Project

Para baixar RStudio acesse o site `https://rstudio.com/products/rstudio/download/`. Antes de instalar o RStudio é necessário instalar primeiramente o R.

A instalação do R e do RStudio é simples: basta clicar no instalador e seguir as instruções.

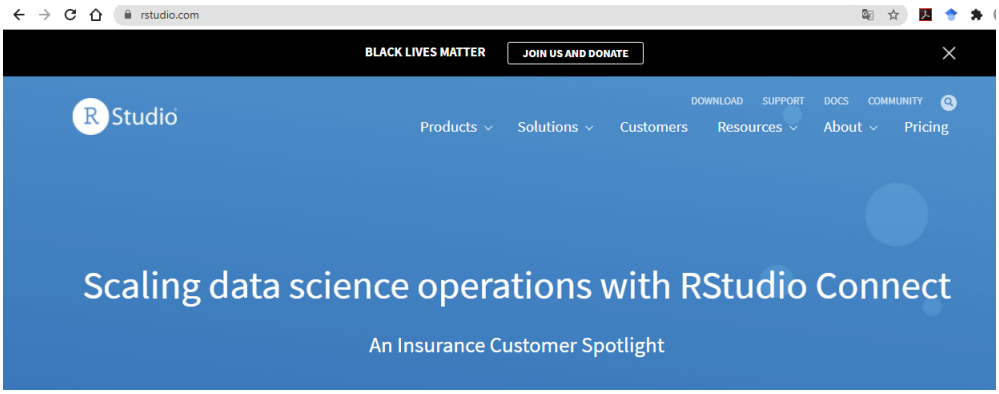


Figura 1.2: Página web do RStudio

Você pode acessar várias recomendações a respeito da instalação do RStudio em `https://support.rstudio.com/hc/en-us/articles/115002344588-Configuration-and-sizing-recommendations`. Outro site bastante útil é o `https://www.rdocumentation.org/`, onde é possível acessar uma vasta documentação sobre o R.

1.2 Área de Trabalho do RStudio

A área de trabalho do RStudio é formado por quatro campos como pode ser visto na Figura 1.6

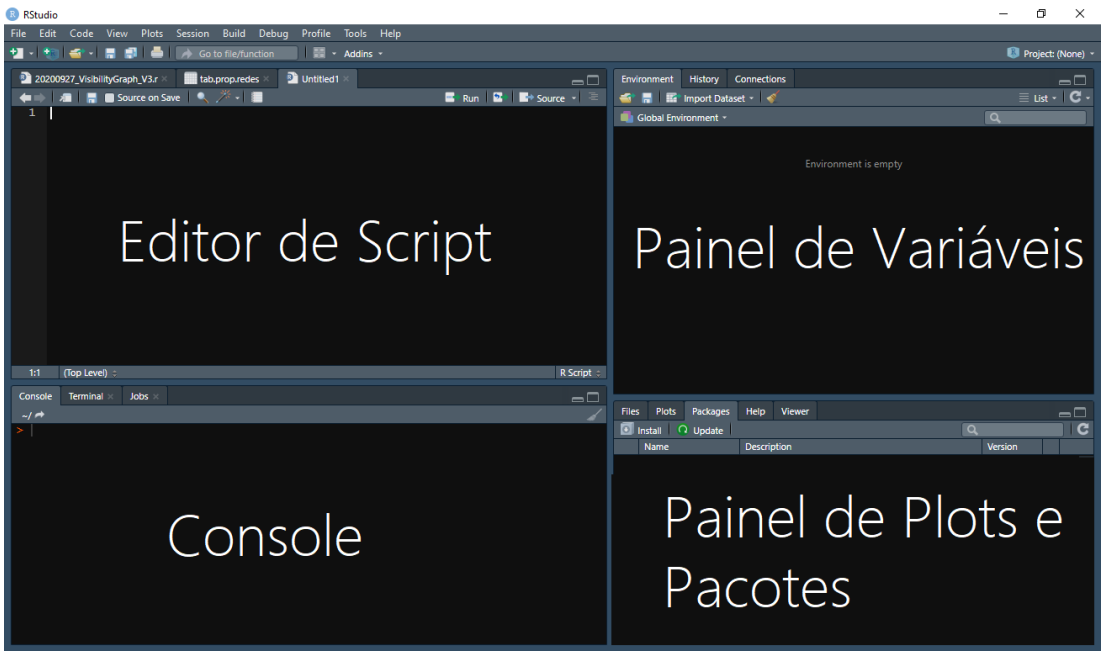


Figura 1.3: Área de trabalho do RStudio

- Editor de Script - local onde se escreve o código que será executado. O script produzido pode ser salvo para ser usado posteriormente.

A Figura 1.4 mostra a expressão $2+3\cdot5-12^2$ digitada no editor de escript. Após executá-la usando o *run* (1), a mesma expressão aparece no console (2), juntamente com o respectivo resultado.

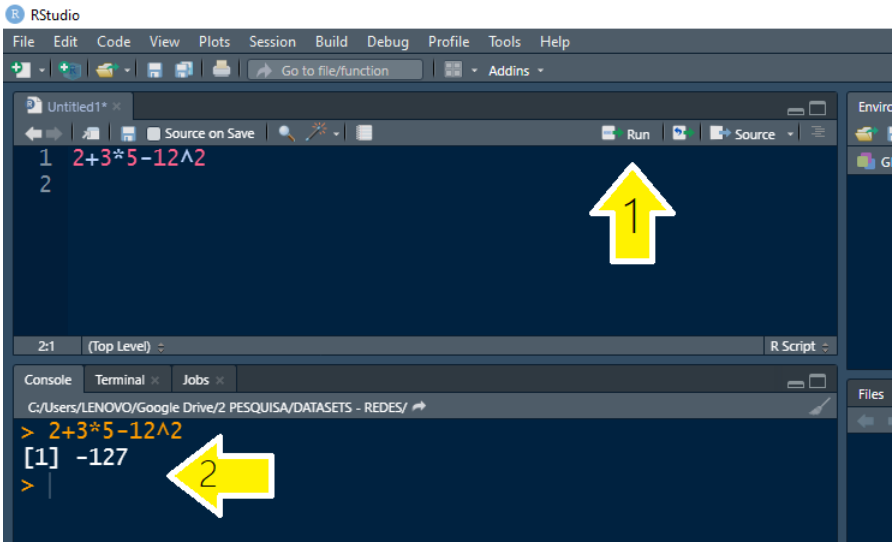


Figura 1.4: Exemplo para Editor de Scipts

Uma linha de código também pode ser executada colocando o cursor na referida linha e fazendo Ctrl + Enter. Para salvar o script basta usar o Ctrl+S.

- Painel do Console R - apresenta o prompt do R, local onde vemos os comandos executados no editor de script. É possivel também executar código diretamente no console, contudo os comandos não serão registrados no history, nem poderão ser salvos.

  Após digitar a linha de comando no console basta teclar Enter. Por exemplo, executando o comando $3\cdot4+1$, temos como resultado o que mostra a Figura 1.5.

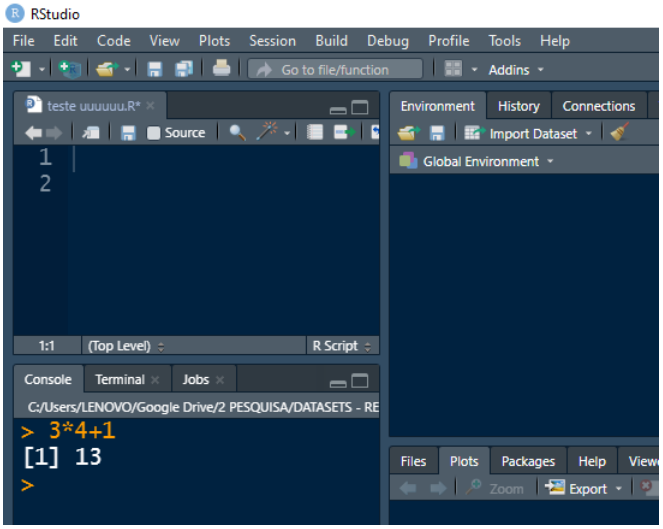


Figura 1.5: Exemplo para comando no console

- Painel de Variáveis - Na aba *Global Environment* ficam salvas as variáveis executadas no editor no script ou no console. No *History*, que exibe o histórico dos comandos executados no RStudio.

  No exemplo a seguir, atribuimos o número 8 à variável x, a e3xpressão $2 \cdot 11 + 3$ á variável y, o quociente $24 : x$ à variável a, e x+y+a á variável SOMA.

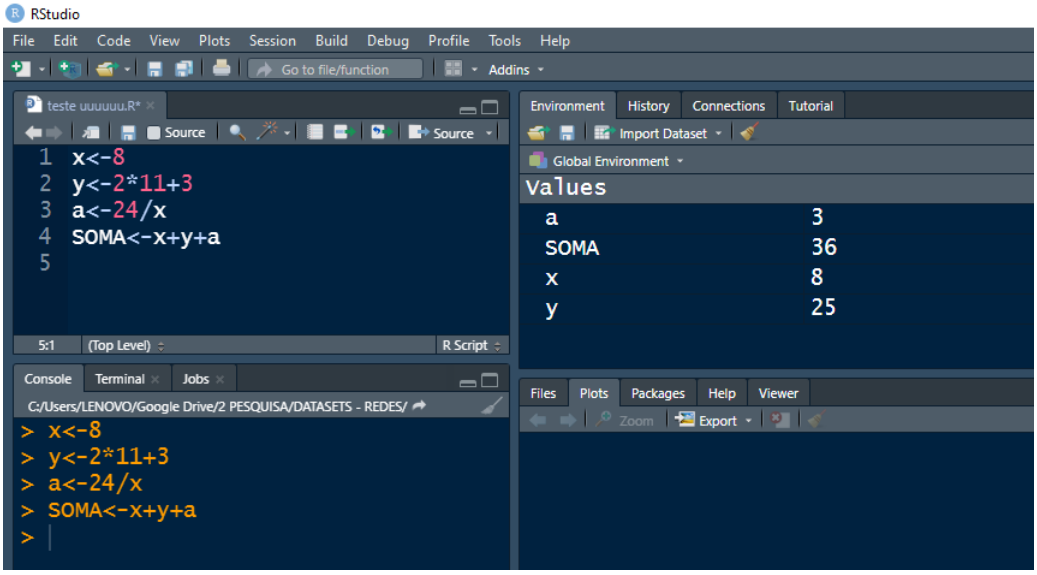


Figura 1.6: Exemplo para Painel de Variáveis

Na aba History ficam registrados todos os comandos executados no script.

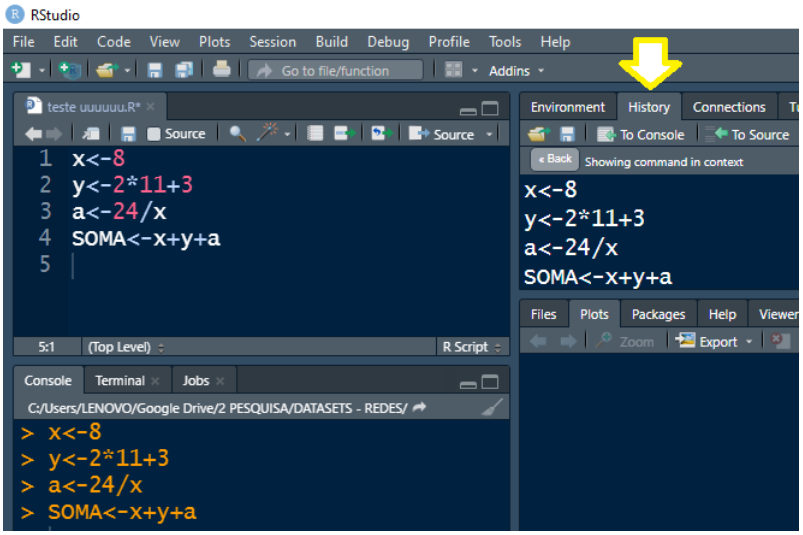


Figura 1.7: Exemplo para a aba history

- Painel de Arquivos e Pacotes
  - *Files*: Nessa aba é possível acessar os arquivos no diretório de trabalho;
  - *Plots*: Aqui é possível visualizar gráfico e figuras diversas;
  - *Packages*: Exibe os pacotes instalados e carregados pelo software, também é possível instalar

novos pacotes;

- *Help*: Nessa seção é possível adquirir informações sobre determinado comando, funcões e pacotes ou mesmo descobrir um comando para determinada situação;
- *Viewer*: Exibe resultados dinâmicos e interativos.

Diretório de Trabalho

Uma informação muito útil é saber qual o seu diretório de trabalho. Para saber essa informação use a função getwd(). Daí, para mudar de diretório você usa getwd("endereco do novo diretorio").

Outro modo: Use esses três passos para mudar o diretório de trabalho:

PRIMEIRO PASSO: vá no painel de plots e pacotes (Fig.1.8)

SEGUNDO PASSO: Escolha a sua pasta para ser a área de trabalho

TERCEIRO PASSO: Selecione o novo diretório (Fig.1.10) (Fig.1.9).

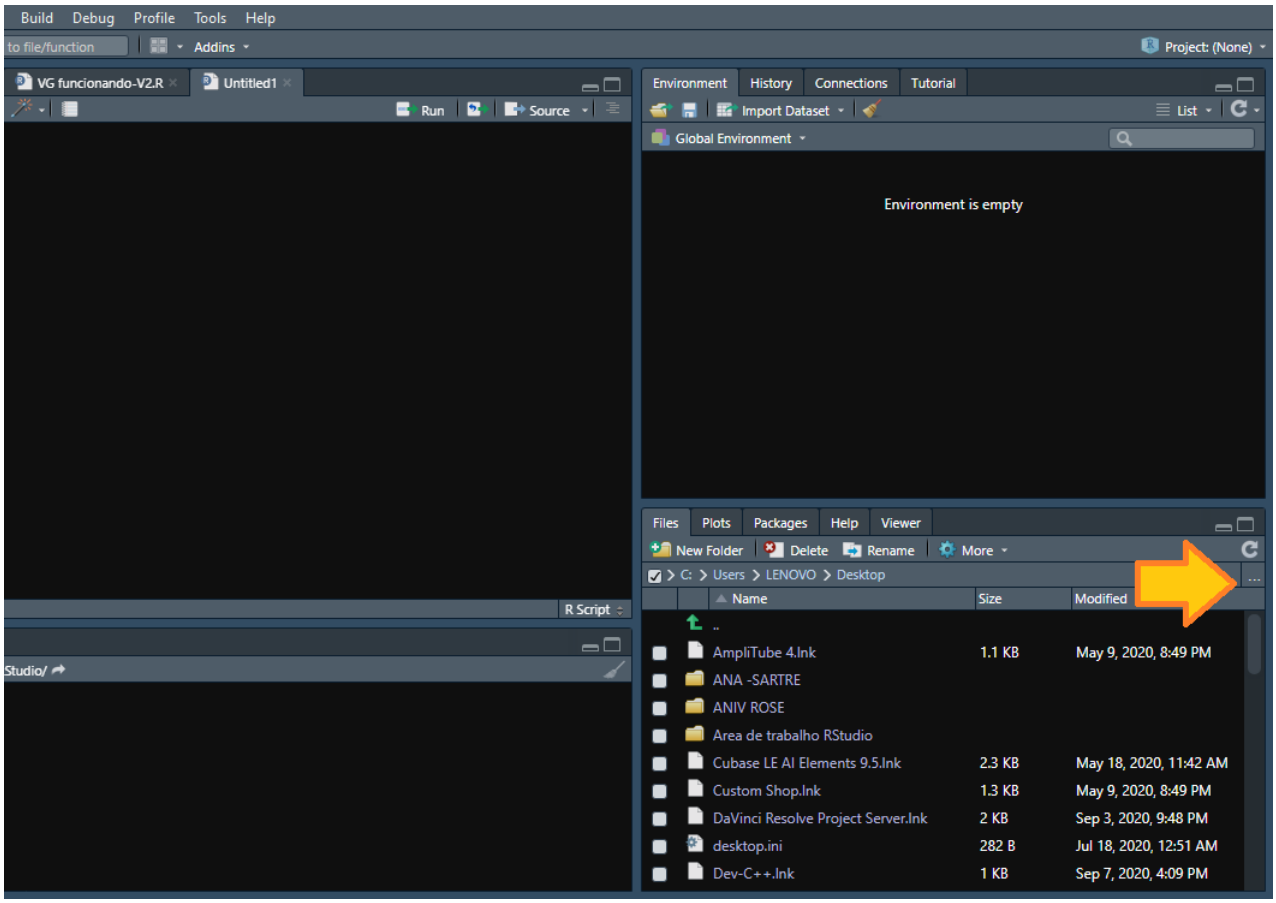


Figura 1.8: Mudando o diretório: passo 1

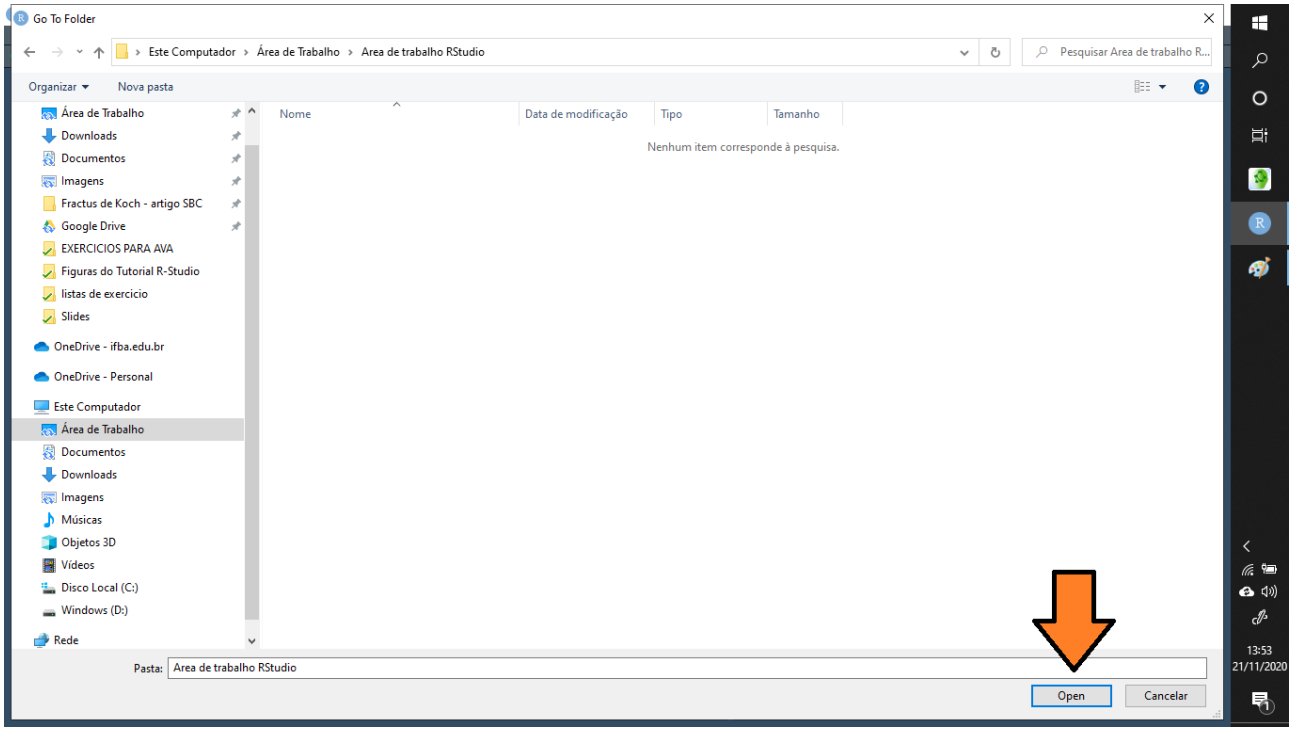


Figura 1.9: Mudando o diretório: passo 2

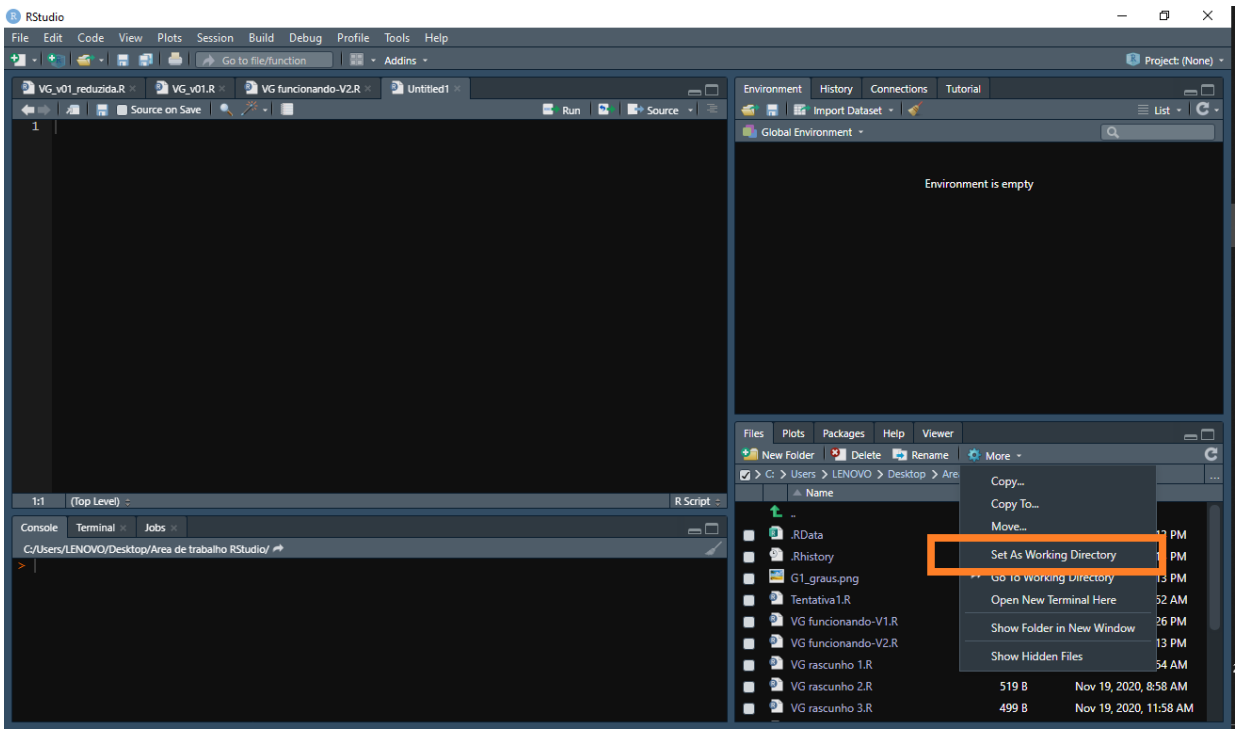


Figura 1.10: Mudando o diretório: passo 3

Outra maneira é seguir o caminho: Tools >> Global Options >> General >> Basic >> Default working directory e mude o diretório para algum local de sua preferência (Fig:1.11)

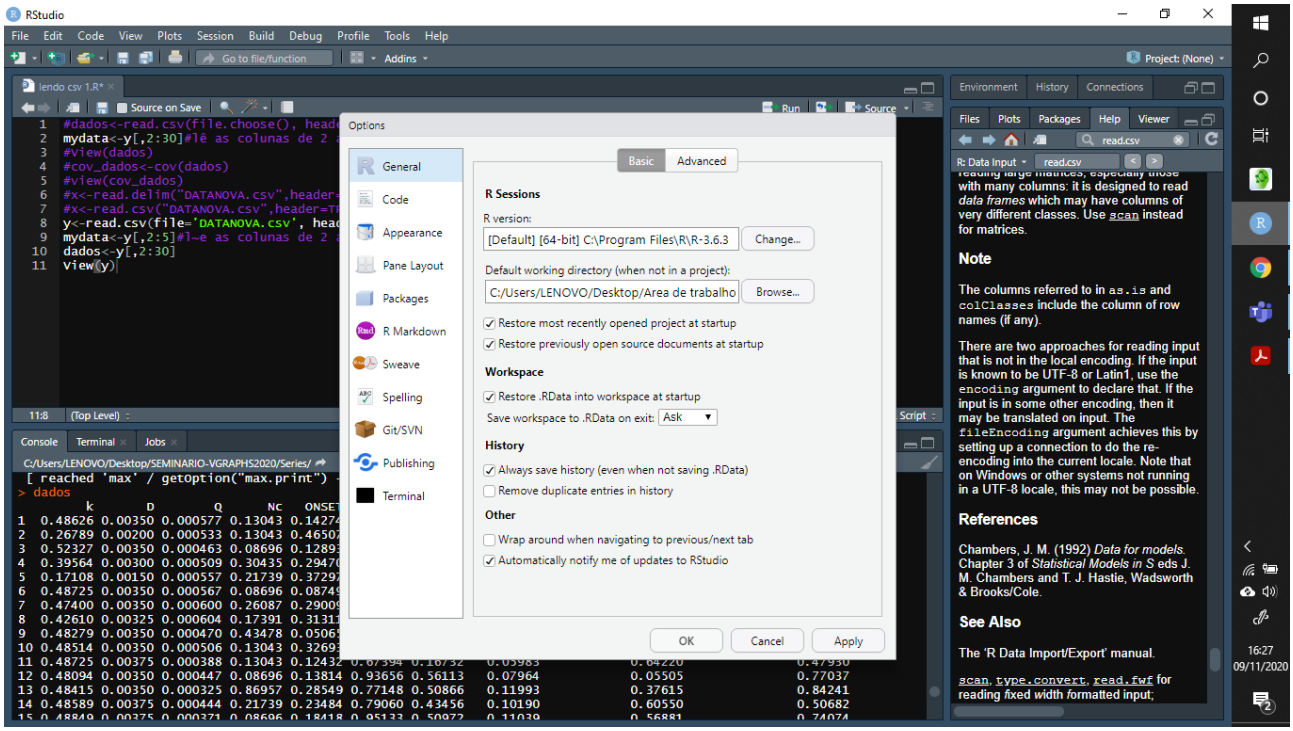


Figura 1.11: Caption

Comentário no R

Para incluir um comentário no R (ou seja, uma sequência de caracteres que deve ser ignorada pelo compilador da linguagem R), basta digitar o símbolo e, em seguida, o comentário. Tudo que aparece depois do sinal é ignorado pelo R.

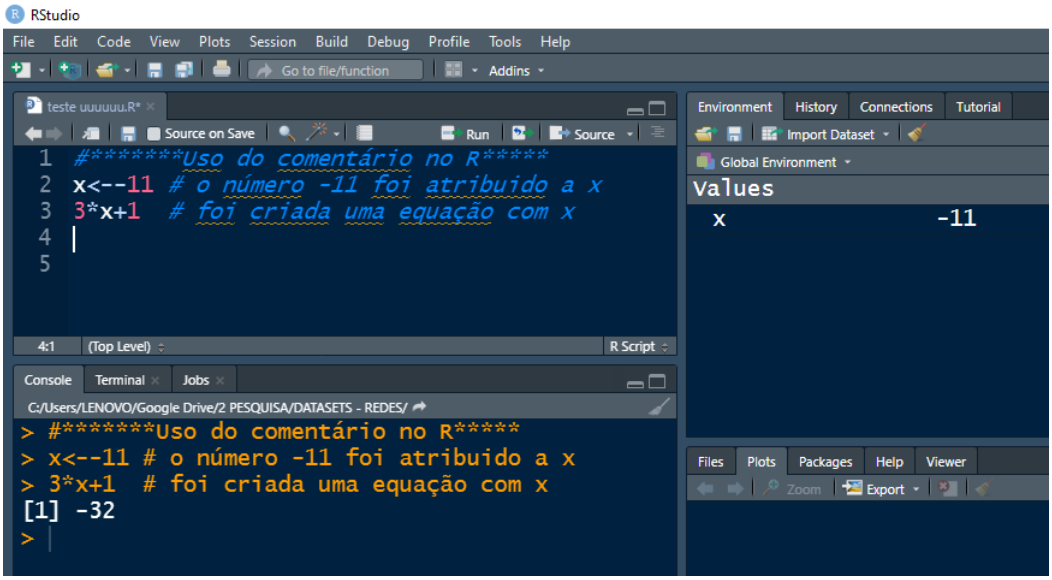


Figura 1.12: Exemplo para comentário

1.3 Pacote igraph

O *igraph* é uma coleção de ferramentas de análise de redes com ênfase na eficiência, portabilidade e facilidade de uso (`https://igraph.org/`). É um software livre e de código aberto escrito em R.

Figura 1.13: Página do igraph

Dentre os objetivos principais dessa biblioteca estão: "proporcionar rápido manuseamento de grandes grafos, com milhões de vértices e arestas; permitir prototipagem rápida por meio de linguagens de alto nível como o R"[5].

*Instalação do igraph Para instalar o pacote igraph, vá em "Painel de Plots e Pacotes"na área de trabalho e na aba "pacotes"(1) clique em "install"(2) (Figura 1.14).

Depois digite "igraph"na janela "Packages"e depois clique em "install".

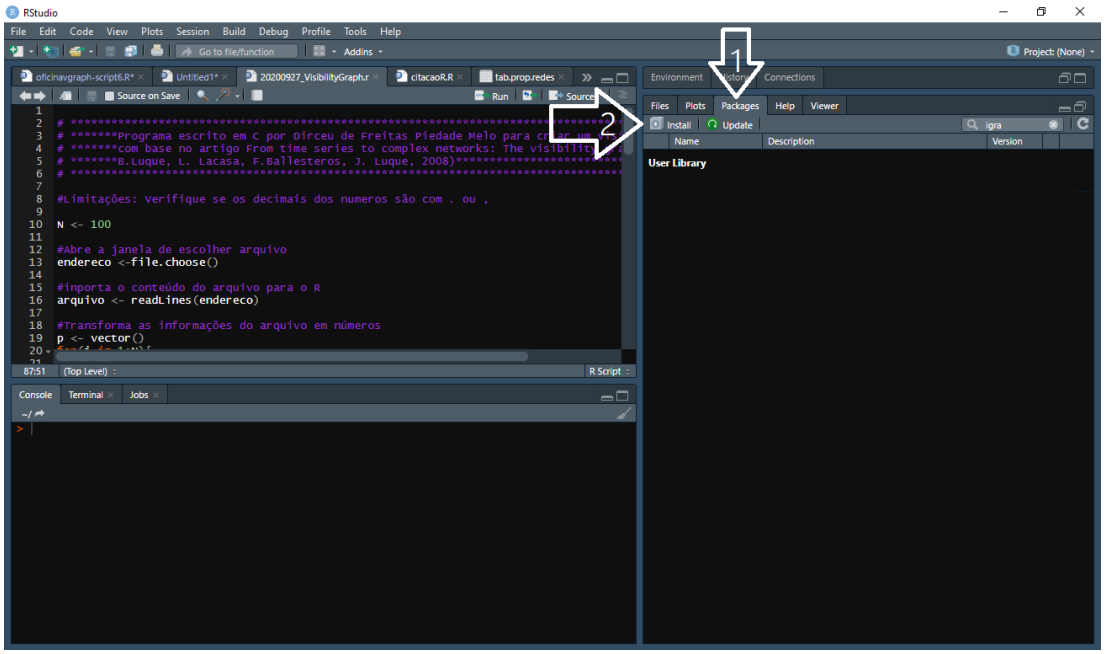


Figura 1.14: instalação do pacote igraph - passos 1 e 2

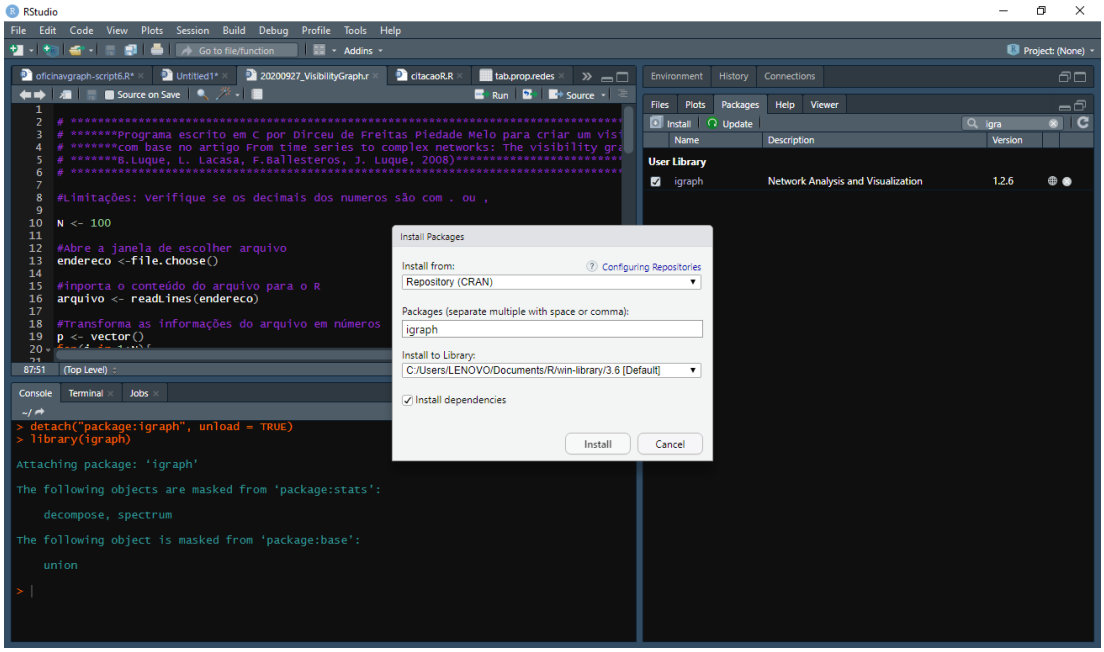


Figura 1.15: instalação do pacote igraph - passo 3

2. Objetos e comandos fundamentais do R

2.1 Como criar objetos na linguagem R

No R tudo é um objeto. Para atribuir e manipular um objeto, usamos o sinal:

```
<-
```

Por exemplo, se quisermos atribuir o número 5 à variável *a*, indicamos:

```
a<-5
```

Para visualizar o objeto salvo, basta digitar o nome do objeto no console e apertar Enter, ou usar a função print(). Podemos também compilar o nome do objeto no editor de script e ele vai exibir o valor no console.

```
>a
[1]5
>print(a)
[1]5
```

Além dados numéricos e lógicos, podemos guardar outros tipos de dados como texto (caracter) ou fator (factor). Abaixo um exemplo de criação de uma variável com dado em texto.

```
b<-"Maria"
```

```
>print(b)
[1]Maria
```

Para substituir o valor da variável basta executar novamente a variável com o novo valor ou texto. No caso para remover uma variável, iremos usar a função rm(). Abaixo no exemplo vamos substituir o valor de por por 5 e deppis remover a variável.

```
a<-105
rm(a)
```

Valores Booleanos

Ao comparar duas sentenças usando os operadores < (menor que), > (maior que), <= (menor ou igual), >= (maior ou igual), == (igual), != (desigual), temos como retorno um valor lógico TRUE (verdadeiro) ou FALSE (falso).

Exemplos

```
> 2==2
[1] TRUE

> 7<=2
[1] FALSE

> 2>7
[1] FALSE

> 4*5+3-3^2 == 14
[1] TRUE

> 4*5+3-3^2 =! 14
[1] FALSE

> x<-2
> y<-0
> x != y
[1] TRUE
```

Observações

- atualmente, o R também aceita o sinal de igual no lugar do operador de atribuição "<-",produzindo o mesmo resultado. Neste texto, usaremos preferencialmente o sinal "<-".
- letras maiúsculas e minúsculas são consideradas diferentes para o R, ou seja o R é *case sensitive*.

2.2 Vetores

Os vetores são estruturas de dados largamente usados em diferentes contextos da linguagem R. Um vetor é um conjunto de valores, todos de mesma classe, separados por vírgula. O vetor pode funcionar como uma variável mesmo tendo vários elementos.

Como criar

Para criar um vetor usamos a função “c()”. Por exemplo:

```
a<-c(5,10,15) # vetor a com 3 elementos
b<-c("janeiro","fevereiro","abril","maio") # vetor de elementos formados por
    caracteres.
```

```
>a # mostrar o vetor a
[1]5 10 15
>b # mostrar o vetor b
[1]janeiro fevereiro mar o abril
```

Como acessar

Os elementos do vetor podem ser acessados através do operador colchetes. sintaxe: vetor[índice].

Exemplo

```
a<-c(5,10,15)
a[1] # acessando o primeiro elemento do vetor a
a[3] # acessando o terceiro elemento do vetor a
```

```
[1]5 # primeiro elemento do vetor a
[1]15 # terceiro elemento do vetor a
```

É possível também acessar múltiplos valores de uma única vez, combinando com a função c().

Exemplo

```
a[c(2,3)] # acessando o segundo e terceiro elemento do vetor a
```

```
[1]5 15
```

Como modificar

Para modificar um ou mais elementos do vetor será preciso acessar os elementos que serão modificados e atribuir os novos valores.

```
a[1]<-35
a[c(1,2)]<-c(40,55)
```

```
[1]35 10 15
[1]35 40 55
```

Para adicionar novos elementos ao vetor, basta acessar o vetor e adicionar as posições dentro dos colchetes.

```
a[4]<-90
a[c(5,6,7)]<-c(95,65,25)
```

```
[1]35  10  15  90
[1]35  10  15  90  95  65  25
```

Para excluir um ou mais elementos do vetor usamos o índice negativo com a posição do elemento que se quer excluir.

```
a[-2]
a[c(-5,-6)]
```

```
[1]35  15  90  95  65  25
[1]35  15  90  95
```

Operadores lógicos ou booleanos

Podemos usar operadores lógicos com múltiplas situações conectadas a mesma expressão, e retornaram vetores lógicos com resposta TRUE (verdadeiro) e False (falso).

Como exemplo podemos imaginar um grupo de pessoas e verificar quantos tem mais de 40 anos, neste caso usando o operador de comparação.

```
v<-c(52,63,30,25,42,12,16,57)
v>40
```

```
[1]52  63  30  25  42  12  16  57
[1]TRUE  TRUE  FALSE  FALSE  TRUE  FALSE  FALSE  TRUE
```

2.3 Fator

O Fator é um objeto do R usado para armazenar variáveis categóricas. Os exemplos a seguir mostram o caso de dados armazenados como vetor e como fator. No caso do vetor esses dados não são vistos como variáveis categóricas.

Ao dar o comando para exibir o fator, o R retorna as categorias no "Levels".

Exemplo 1

```
animal.tipo.v <- c("mamifero", "reptil", "mamifero", "passaro", "passaro", "passaro") # vetor
animal.tipo.f <- factor(c("mamifero", "reptil", "mamifero", "passaro", "passaro", "passaro")) # fator

animal.tipo.v # mostra os dados do vetor
## [1] "mamifero", "reptil", "mamifero", "passaro", "passaro", "passaro"

animal.tipo.f  # mostra os dados e as categorias do fator
## [1] mamifero reptil mamifero passaro passaro passaro  # dados
## Levels: mamifero passaro reptil   # categorias
```

Exemplo 2

```
Nesse tipo de objeto, o R vai identificar diferentes categorias (ou n veis) do fator e cada uma dessas categorias ser  armazenada com um inteiro correspondente  sua posi  o\cite{ognyanova2016network}. No caso de nosso exemplo as categorias s o colocadas em ordem alfab tica e depois numeradas.

levels(animal.tipo.f) # mostra as categorias do fator
## [1] "mamifero" "passaro"  "reptil"  # mamifero  o 1, passaro o 2, e reptil o 3.

as.numeric(animal.tipo.f) # mostra os dados como n meros inteiros atribuidos a cada categoria, na sua ordem de registro.
## [1] 1 3 1 2 2 2
```

2.4 Matrizes

Na linguagem R, a matriz é uma estrutura de dados homogênea, isso quer dizer que só pode contar um tipo de dado (númerico, caracter ou lógicos) de duas dimensões. Sua sintaxe básica é:

```
matrix()
```

Como criar Matrizes

Para se criar uma matriz é preciso determinar os elementos, na forma de vetor, e o número de linhas e colunas. Por exemplo

```
m<-matrix(data=1:12,nrow=4,ncol=3)
```

```
>m
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

Por padrão os elementos são organizados por colunas, mas é possível organizar por linha usando um comando complentar byrow=TRUE.

```
n<-matrix(data=1:12,nrow=4,ncol=3,byrow=TRUE)
```

```
>n
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

A matrizes podem ser construidas a partir de outros vetores atraves dos comandos cbind() e rbind().

Rbind monta uma matriz, empilhando os argumentos linha a linha na ordem em que aparecem na lista.

```
x<-3:5
y<-13:15
z<-23:25
rbind(x, y)
```

```
  [,1] [,2] [,3]
x    3    4    5
y   13   14   15
z   23   24   25
```

Cbind() monta a matriz colocando os argumentos lado a lado na ordem em que aparecem na lista.

```
x<-3:5
y<-13:15
z<-23:25
rbind(x,z,y)
```

```
     x  z  y
[3,] 3 23 13
[4,] 4 24 14
```

```
[5,] 4 25 15
```

Como acessar

Para acessar os valores da matriz, é necessário informar 2 coordenadas: uma para a linha e outra para a coluna, entre colchetes. Por exemplo, queremos identificar qual o valor se encontra na segunda linha da terceira coluna da matriz n.

```
n[2,3]
```

```
>[1]6
```

É possível também visualisar somente os valores da linha ou da coluna, idenficando a linha ou a coluna e deixando a outra coordenada vazia. Por exemplo, queremos identificar os elementos da linha 2 da matriz n.

```
n[2,]
```

```
>[1]4 5 6
```

Como modificar

Para modificar um elemento em uma matriz, usaremos a mesma lógica que nos vetores, lembrando que a matriz precisa de duas coordenadas para ser acessada.

```
n[2,3]<-25
```

```
[1]25
```

Para adicionar novos elementos a matriz, basta acessar a matriz e adicionar a linha ou a coluna,além de seu elementos.

```
n[5,]<-c(13,14,15)
```

```
[1]13 14 15
```

Para excluir uma linha ou uma coluna usando o índice negativo com a posição da linha ou da coluna.

```
n[2,]<-n[-2,]
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    7    8    9
[3,]   10   11   12
[4,]   13   14   15
```

Operações com matrizes

As operações com matrizes numéricas seguem a mesma lógica dos vetores. A operação é feita para todos os elementos da matriz. Por exemplo.

```
n+5
n*5
```

```
     [,1] [,2] [,3]
[1,]    6    7    8
[2,]   12   13   14
[3,]   15   16   17
[4,]   18   19   20

     [,1] [,2] [,3]
[1,]    5   10   15
[2,]   35   40   45
[3,]   50   55   60
[4,]   65   70   75
```

Os comandos mean, sum e sd irão funcionar consideração todos os elementos da matriz. Por exemplo.

```
mean(n)
sum(n)
```

```
>[1]8.75
>[1]105
```

As operações entre matrizes também são admissíveis no R. Mas, é preciso ressaltar que é preciso seguir as regras para calcular matrizes.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} B = \begin{bmatrix} 1 & 2 & 1 \\ 4 & 1 & 6 \end{bmatrix} C = \begin{bmatrix} 1 & 2 \\ 4 & 1 \\ 2 & 1 \end{bmatrix} D = \begin{bmatrix} 1 & 2 \\ 4 & 1 \end{bmatrix}$$

```
A<-matrix(c(1,2,3,4,5,6),nrow=2,ncol=3)
B<-matrix(c(1,2,1,4,1,6),nrow=2,ncol=3)
```

Na soma e substração

```
A+B
A-B
```

```
     [,1] [,2] [,3]
[1,]    2    4    4
[2,]    8    6   12

     [,1] [,2] [,3]
[1,]    0    0    2
[2,]    0    4    0
```

Multiplicação matricial

```
A%*%C
```

```
     [,1] [,2]
[1,]   15    7
[2,]   36   19
```

Transposta de uma matriz,a transposta de uma matriz é calculada pela pela função t()

```
t(A)
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Determinante de uma matriz, o determinante de uma matriz é calculado pela função det()

```
det(D)
```

```
>[1]-7
```

2.5 Array

Na linguagem R, o conceito de array é semelhante ao de matriz, enquanto em uma matriz possui duas dimensões, no array possui três dimensões, sendo a terceira dimensão a quantidade de matrizes. Sua sintaxe básica é:

```
array()
```

Como criar Array

Podemos criar array usando como elementos matrizes ou vetores.

```
array(elementos,dimensao=(linhas,colunas,quantidades_de_matrizes),nomes_das_
    dimensoes)
```

Usando vetores

```
vendas<-c(12,24,30)
produtos<-c("A","B","C")
A<-array(c(produtos,vendas),dim=c(3,2,2))
```

```
, , 1

     [,1]          [,2]
[1,] "TV"          "12"
[2,] "Geladeira"   "24"
[3,] "Cama"        "30"

, , 2

     [,1]          [,2]
[1,] "TV"          "12"
[2,] "Geladeira"   "24"
[3,] "Cama"        "30"
```

Usando matriz

```
B<-matrix(1:20,4,5)
array(B,dim=c(4,5,1))
```

```
, , 1

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20

, , 2

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

Podemos também fazer array de forma direta.

```
X<-array(c("D","C","B"),dim = c(2,3,3))
```

```
, , 1
     [,1] [,2] [,3]
[1,] "D"  "B"  "C"
[2,] "C"  "D"  "B"

, , 2
     [,1] [,2] [,3]
[1,] "D"  "B"  "C"
[2,] "C"  "D"  "B"

, , 3
     [,1] [,2] [,3]
[1,] "D"  "B"  "C"
[2,] "C"  "D"  "B"
```

Como acessar

Para acessar os valores do array, é necessário informar 3 dimensões: uma para a linha, uma para a coluna e outra para matrizes entre colchetes. Por exemplo, queremos identificar qual o valor se encontra na segunda linha da terceira coluna da matriz 2 do array X.

```
X[2,3,2]
```

```
[1] "B"
```

Da mesma forma que matrizes, é possível idenficar a linha ou a coluna ou a matriz, deixando as outras dimensões vazias. Por exemplo, queremos identificar os elementos da linha 2 do array X.

```
X[2,,]
```

Neste caso, será exibido a linha 2 de todas as matrizes do array.

```
> X[2,,]
     [,1] [,2] [,3]
[1,] "C"  "C"  "C"
[2,] "D"  "D"  "D"
[3,] "B"  "B"  "B"
```

Como modificar

Para modificar um elemento de um array, usaremos a mesma lógica que nas matrizes, lembrando que a array precisa de três dimensões para ser acessada.

```
X[2,3,1]<-"F"
X[1,,1]<-c("F","F","F")
```

```
[1]"F"
[1]"F"  "F"  "F"
```

Como nomear os elementos do array

Para isso iremos criar o array com os nomes das matrizes.

```
AAA
AAA
```

2.6 Data Frame

Na linguagem R, o conceito de data frame é uma estrutura semelhante à uma matriz, mas que aceita vetores de tipos diferentes para cada coluna. Isso permite ter colunas com caracteres e outras com valores númericos. Sua sintaxe básica é:

```
data.frame()
```

Como criar data frame

Existem três formas de criar data frames

```
vendedores<-c("A","B","C")
Adidas<-c(20,30,10)
Nike<-c(22,60,6)
Olk<-c(52,40,10)
vendas<-cbind(vendedores,Adidas,Nike,Olk)
```

Transformando matriz em data frame

```
vendas_df<-as.data.frame(vendas)
vendas_df
```

Transformando vetores em um data frame

```
vendas_v<-data.frame(vendedores,Adidas,Nike,Olk)
vendas_v
```

```
> vendas_v
  vendedores Adidas Nike Olk
1          A     20   22  52
```

```
2          B      30     60    40
3          C      10      6    10
```

Criando o data frame fornecendo os elementos e os dados, neste caso é necessário fornecer também os nomes dos elementos.

```
vendas_x<-data.frame(Vendedores=c("A","B","C"),Adidas=c(20,30,10),Nike=c(22,60,6),
    Olk=c(52,40,10))
```

```
> vendas_x
  Vendedores Adidas Nike Olk
1          A     20   22  52
2          B     30   60  40
3          C     10    6  10
```

Como pesquisar ou acessar elementos do data frame

O acesso dos elementos de uma data frame é semelhante a de matrizes.

```
vendas_x[2,]
vendas_x[1:3,]
```

```
> vendas_x[2,]
  vendedores Adidas Nike Olk
2          B     30   60  40
> vendas_x[2:4,]
  vendedores Adidas Nike  Olk
2          B     30   60   40
3          C     10    6   10
NA      <NA>   <NA> <NA> <NA>
```

É possível também acessar os elementos usando o símbolo de sifão.

```
vendas_x$Adidas
vendas_x$Nike[1]
```

```
> vendas_x$Adidas
[1] 20 30 10
> vendas_x$Nike[1]
[1] 22
```

Como modificar os elementos do data frame

A modificação do data frame usa a mesma lógica aplicada as matrizes. Basta acessar o elemento e atribuir com um novo elemento.

```
vendas_x[2,2]<-99
vendas_x[2,2]
vendas_x$Adidas<-c(70,80,90)
vendas_x$Adidas
vendas_x$Adidas[3]<-130
vendas_x$Adidas[3]
```

```
> vendas_x[2,2]
[1] 99
> vendas_x$Adidas
[1] 70 80 90
> vendas_x$Adidas[3]
[1] 130
```

Também é possível adicionar ou remover variáveis ao data frame.

```
vendas_x$Rainha<-c(10,15,20)
vendas_x
vendas_x<-vendas_x[,-5]
vendas_x
```

```
> vendas_x
  vendedores Adidas Nike Olk Rainha
1          A     20   22  52      10
2          B     30   60  40      15
3          C     10    6  10      20
> vendas_x
  vendedores Adidas Nike Olk
1          A     20   22  52
2          B     30   60  40
3          C     10    6  10
```

Outra situação é adicionar ou remover casos.

```
vendas_x[4,]<-c("D",60,70,90)
vendas_x
vendas_x<-vendas_x[-4,]
vendas_x
```

```
> vendas_x
  vendedores Adidas Nike Olk
1          A     20   22  52
2          B     30   60  40
3          C     10    6  10
4          D     60   70  90
> vendas_x
  vendedores Adidas Nike Olk
1          A     20   22  52
2          B     30   60  40
3          C     10    6  10
```

2.7 Lista

O conceito de lista na linguagem R, é um estrutura que permite armazenar diferentes tipos de dados. Um mesma lista pode armazenar um vetor, uma matriz e uma data.frame, que não precisam ter o mesmo comprimento. Portanto, uma lista é dito como uma coleção de objetos. Sua sintaxe básica é:

```
list()
```

Como criar lista

Podemos criar lista usando vetores e matrizes ao mesmo tempo.

```
nomes<-c("Ana","Carlo","Edu","Gil")
idade<-c(21,11,14,17)
novo<-c(TRUE,FALSE,FALSE,TRUE)
clientes<-list(nomes,idade,novo,matrix(0,4,5))
```

```
> clientes
[[1]]
[1] "Ana"   "Carlo" "Edu"   "Gil"

[[2]]
[1] 21 11 14 17

[[3]]
[1]  TRUE FALSE FALSE  TRUE

[[4]]
     [,1] [,2] [,3] [,4] [,5]
```

```
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
```

Como nomear os elementos da lista

Para adicionar o nome do elementos da lista basta adicionar o nome antes de cada elemento da lista mais o igual.

```
clientes<-list(Nomes=nomes,Idade=idade,Novo= novo,Hist rico=matrix(0,4,5))
```

```
> clientes
$Nomes
[1] "Ana"   "Carlo" "Edu"   "Gil"

$Idade
[1] 21 11 14 17

$Novo
[1]  TRUE FALSE FALSE  TRUE

$Hist rico
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
```

Como pesquisar informações da lista

Para acessar informação da lista é necessário identificar a lista mais o simbolo de sifão mais o nome do vetor ou matriz ou data.frame que deseja acessar. Caso deseje acessar um elemento acrescenta o indice entre colchetes. Caso queira acessar o elemento completo da lista usamos um duplo colchetes.

```
clientes$Hist rico
clientes$Nomes[2]
clientes$'Idade'[2]
clientes[[1]]
```

```
> clientes$Hist rico
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
> clientes$Nomes[2]
[1] "Carlo"
> clientes$'Idade'[2]
[1] 11
> clientes[[1]]
[1] "Ana"   "Carlo" "Edu"   "Gil"
```

Como adicionar e remover informações da lista

```
clientes$Nomes[5]<-"Laila"
clientes$Nomes[6:8]<-c("Rita","Tati","Lia")
clientes
clientes$Nomes<-clientes$Nomes[-c(5,6,7,8)]
clientes
```

```
> clientes$Nomes
[1] "Ana"   "Carlo" "Edu"   "Gil"   "Laila" "Rita"  "Tati"  "Lia"
> clientes$Nomes
[1] "Ana"   "Carlo" "Edu"   "Gil"
```

Operação - Combinação de listas

```
info1<-list(Data="12-01-2020",Gestor="Sr. Lucas",Cargo="Diretor")
info2<-list(Data="22-01-2020",Gestor="Sr. Lucas",Cargo="Diretor")
info_total<-c(info1,info2)
info_total
```

```
> info_total
$Data
[1] "12-01-2020"

$Gestor
[1] "Sr. Lucas"

```

```
$Cargo
[1] "Diretor"

$Data
[1] "22-01-2020"

$Gestor
[1] "Sr. Lucas"

$Cargo
[1] "Diretor"
```

Transformar lista em vetor

```
infonovo<-unlist(info_total)
infonovo
```

```
> infonovo
        Data       Gestor        Cargo         Data       Gestor
"12-01-2020"  "Sr. Lucas"   "Diretor" "22-01-2020"  "Sr. Lucas"
       Cargo
   "Diretor"
```

2.8 Função Plot

Para construir gráfico usamos o comando plot(), que serve para produzir gráficos de modo automático dependendo dos dados e classe do objeto. De forma simples, os eixos, nomes dos eixos e títulos serão gerados de forma automaticamente, mas também é possível excluir todos estes dados no gráfico gerado.

Sintaxe básica do comando plot()

```
plot(x,y)
```

Os dados dos argumentos x e y são vetores numéricos de mesma dimensão. Por exemplo:

```
plot(1:10,11:20)
```

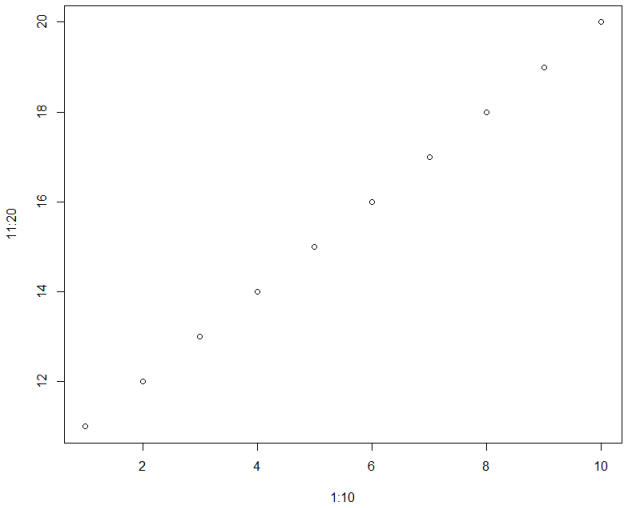


Figura 2.1: Gráfico simples com pontos

Para gerar um gráfico com representação diferente usamos o argumento type.

```
plot(x,y,type="l") #gera um grafico com linhas
```

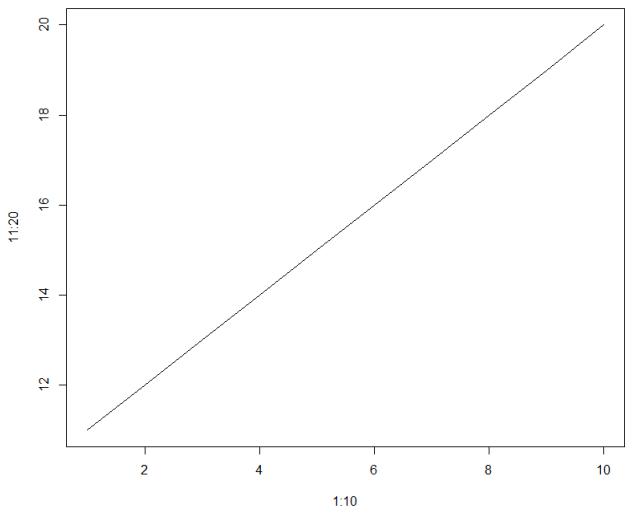


Figura 2.2: Gráfico simples com linhas

Também é possível criar limites nas abicissas e ordenadas.

```
plot(1:10,11:20,xlim=c(0,12),ylim=c(10,22))
```

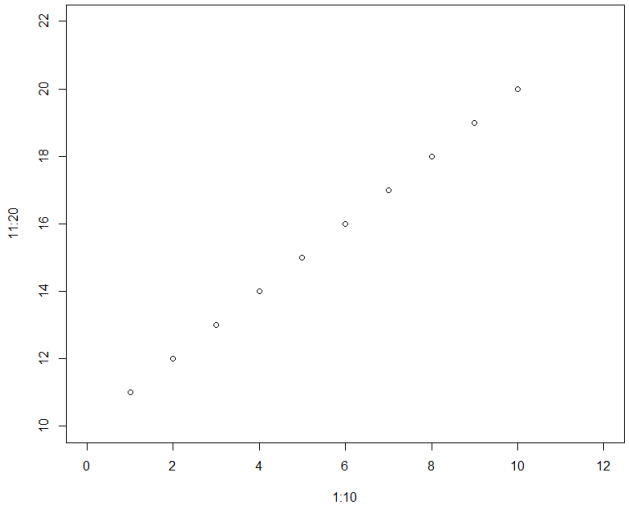


Figura 2.3: Gráfico com limites nas abicissas e ordenadas.

Podemos adicionar nomes aos eixos e um título ao gráfico, adicionando os argumentos xlab, ylab e main, respectivamente.

```
plot(1:10,11:20,xlim=c(0,15),ylim=c(11,25),xlab="valores x",ylab="valores y",main="
    Teste 1")
```

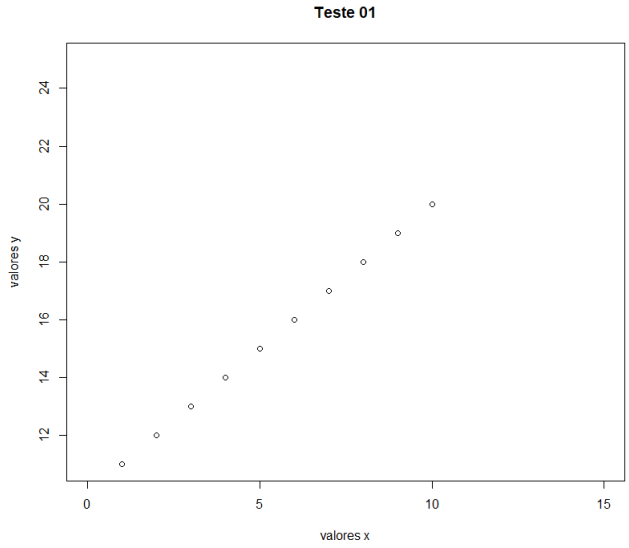


Figura 2.4: Gráfico com nomes nos eixos e título

2.9 Função Hist

Para construir um histograma na linguagem de programação R usamos o comando hist(), que serve para construir gráfico de frequência de uma determinada amostra. Sintaxe básica do comando plot().

```
hist(x)
```

Exemplo de um histograma a partir de um vetor.

```
x<-c(6,3,4,3,2,3,4,5,3,1,2,3,4,1)
hist(x)
```

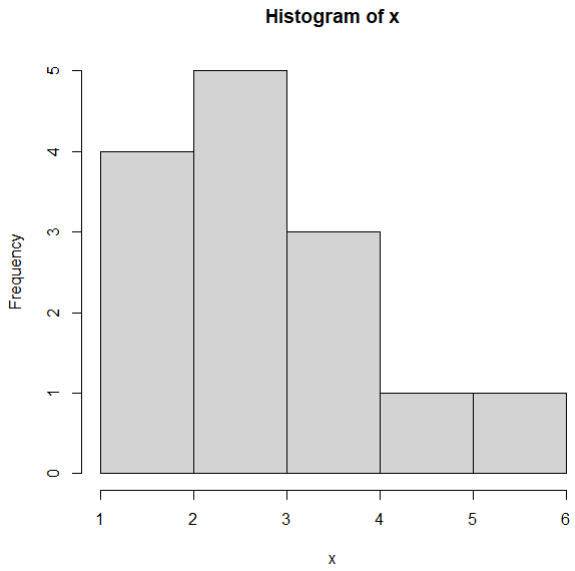


Figura 2.5: Histograma simples

Também é possivel adicionar outros parâmetros para melhorar a representação do histograma.

main	-	define um título ao histograma
xlab ou ylab	-	define a legenda do eixo x ou y
xlim e ylim	-	define os limites dos eixos
col	-	define cor das barras do histograma
border	-	define a cor das bordas do histograma ou se tem borda
breaks	-	define a quantidade de classes

obs: freq - por padrão a frequência do histograma é absoluta, mas é possível também apresentar frequência relativa, desativando o parâmetro freq.

No exemplo abaixo, apresentamos o histograma com os parâmetros adicionais.

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

```
2 hist(x,main="Histograma Simples",xlab="Classe",ylab="Frequencia",xlim=c(1,8),ylim=c
    (0,0.5),col="white",border="blue",breaks=c(1,2,3,4,5,6),freq=F)
```

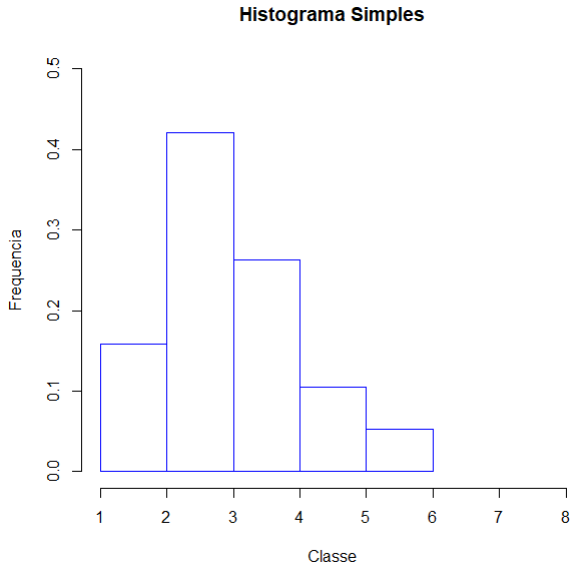


Figura 2.6: Histograma com parâmetros adicionais

3. Estruturas de controle da linguagem R

As estruturas de controle se refere aos desvios condicionais, as estruturas repetições e as funções.

3.1 Estrutura de Desvio Condicional

Estruturas de desvio condicionais são funções que verificam se um resultado de uma condição é satisfeita, e então executa um bloco de código que dependerá do resultado. A estrutura de desvio condicional pode ser simples, composta ou encadeada.

Desvio condicional simples - comando if() Permite executar apenas um bloco de código, por exemplo:

```
a<-10
if(a==20){
    res<-"Valor igual a 20."
    print(res)
}
a<-20
if(a==20){
    res<-"Valor igual a 20."
    print(res)
}
```

```
[1] "Valor igual a 20."
```

O primeiro script não gera resposta, pois o comando tem valor lógica falso.

Desvio condicional composto – comandos if() e else() Permite criar dois blocos de código depois de fazer o teste lógico, por exemplo:

```
a<-10
if(a==20){
    res<-"Valor igual a 20."
    print(res)
}else{
```

```
    res<-"Valor diferente de 20."
    print(res)
}
```

Lembrando que o comando else deve ser inserida na mesma linha da chave de fechamento do comando if, para o programa entender que pertence ao mesmo script.

```
[1] "Valor diferente de 20."
```

Desvio condicional encadeado – comandos if(), else if() e else()

Permite conferir condições de teste sucessivas, onde uma ação será executada, assim que uma condição for satisfeita.

```
a<-10
if(a==20){
    res<-"Valor igual a 20."
    print(res)
}else if(a>20){
    res<-"Valor maior que 20."
    print(res)
}else{
    res<-"Valor menor que 20."
    print(res)
}
```

```
[1] "Valor menor que 20."
```

3.2 Estruturas de repetição

Quando precisamos repetir um conjunto de bloco diversas vezes no script, podemos usar uma estrutura de repetição, mas conhecido como estrutura de laço (loop), que permite repetir trechos de código quantas vezes for necessária. As três principais estruturas de repetição são: repetição com variável de controle, repetição pré-testada e repetição pós-testada.

Repetição com variável de controle – comando for()

```
##Sintaxe:
for(variavel in vetor){
 comandos
}
```

Permite criar loops sabendo o número de repetições que devem ser realizadas, por exemplo:

```
for(i in 1:5){
```

```
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Repetição pré-testada – comando while()

```
##Sintaxe:
while(teste){
  comandos
}
```

Permite realizar um teste logico, e cada vez que o teste retorna verdadeira, o trecho associado ao loop será executado, por exemplo:

```
a<-2
while(a<=12){
    print(a)
    a<-a+2
}
```

```
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
[1] 12
```

Repetição pós-testada – repeat()

```
##Sintaxe:
repeat {
    Comandos a serem repetidos
if (Condi  o para que a repeti  o pare) break()
}
```

Permite repete o código sem condições. Sendo necessário mais uma comando para o programa parar de repetir o código, por exemplo:

```
a=1
repeat {
```

```
    print (a)
    a=a+1
if (a>6)break()
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
```

3.3 Função

Para criar uma função, usamos o comando function(), seguida de uma lista de argumentos e separados por virgulas, após executada o usuário pode usar como qualquer outro comando.

```
##Sintaxe:
nome<-function(argumento.1,...,argumento.n){
    Comandos da funcao
}
```

Lembrando que é preciso dar um nome a nova função seguindo às regras para nomeação de variáveis. Assim, a nova função será executada usando o nome criado.
Os argumentos da função são as informações que precisamos inserir para a criação do como comando. Por exemplo, se quisermos criar uma função para calcular o delta de uma equação quadrática, vamos precisar das constantes da equação. Esses argumentos serão separados por vírgula. Os comandos da função são os códigos que estarão sujeitos ao objetivo da função, sendo necessário em muitos dos casos o uso de estruturas condicionais ou repetições. Lembrando que todos esses comandos devem ser delimitados por chaves.
O comando return não é obrigatório, sua função é retornar o resultado a partir do código que deverá ser escrito dentro dos parênteses do comando. Para executar o novo comando basta chamá-la pelo nome dado a função e escrever os argumentos e executar. Por exemplo:
Com return()

```
maior<-function(x,y){
if(x>y){
return(x)
}else{
return(y)
}
}
maior(5/2,9/7)
```

```
[1] 2.5
```

Sem return()

```
delta.quad<-function(a,b,c){
    b^2-4*a*c
}
delta.quad(5,8,3)
```

```
[1] 4
```

II REDES: FUNDAMENTOS E MODELAGEM COMPUTACIONAL

4 Grafos e Redes 51

4.1 O que são?

4.2 De onde surgiu a ideia de redes?

4.3 Implementando grafos com o R

4.4 Grafos Notáveis

5 Características de Redes Complexas . 69

5.1 Grau

5.2 Coeficiente de Aglomeração (transitividade)

5.3 Densidade

5.4 Caminhos e distâncias

5.5 Modularidade

5.6 Distribuição de graus

6 Tipologia de Redes Complexas 93

6.1 Grafos Aleatórios

6.2 Redes Small World

6.3 Redes Livres de Escala

4. Grafos e Redes

Nesse capítulo apresentaremos informações fundamentais sobre a linguagem R, ao mesmo tempo que mostraremos exemplos de modelagem computacional usando o pacote Igraph e pequenas rotinas computacionais para exemplificar cada um dos aspectos abordados.

4.1 O que são?

Figura 4.1: Ilustração de uma rede

O termo rede tem origem do latim *rete*, usado para designar aquelas conhecidas redes de caça ou pesca. Mas as redes que queremos definir têm muito a ver com outro termo latino - *connectare* - que é a composição de *con* (junto), mais *nectare* (ligar,atar) [15]. De um modo mais amplo, podemos definir uma rede como um conjunto de vértices conectados por um conjunto de arestas, onde existe uma relação entre os vértices representado pelas arestas. A representação de uma rede recebe o nome de grafo.

As redes são concepções com as quais é permitido agrupar quaisquer tipos de relações, desde

relações entre indíviduos (amizade, trabalho, família, escola), passando relações entre cidades (aeroportos, rodovias, ferrovias), relações entre palavras de um texto (redes semânticas), e a relação entre computadores (internet).

No mundo, são incalculáveis os sistemas que podem ser explicados em um sistema de rede. Muitos desses sistemas exibem propriedades coletivas emergentes e são chamados de sistemas complexos. As redes complexas são redes usadas para modelar esses tipo de sistema. Os exemplos mais conhecidos de redes complexas são:

- **Redes Sociais** - de um modo geral é um conjunto de pessoas ou grupos de pessoas, ligados entre si por relações que podem ser de trabalho, profissionais, familiares, etc. Existem redes sociais que pertencem a grandes corporações. Dentre as mais conhecidas podemos destacar: Linkedin, Facebook, Watsapp, Instagram, e Twetter;
- **Redes Biológicas** - redes aplicadas a sistemas biológicos. Exemplos: sistema de organização do funcionamento das células do corpo humano, rede de interação das proteínas; rede funcional do sistema nervoso central;
- **Rede de informação** - WWW (World Wide Web) - rede onde os nós são os documentos disponíveis e os links fazem as ligações entre documentos (Albert et al., 1999). Outro exemplo de uma rede de informação é a rede das citações entre artigos científicos (Jaffe e Trajtenberg, 2002).

4.2 De onde surgiu a ideia de redes?

Muitos aceitam que o estudo sistemático das redes teve início com o estabelecimento de sua representação através dos grafos, implementada pelo matemático suíço Leonhard Euler, ao resolver o enigma das pontes de *Königsberg*. Euler utilizou um sistema constituído de nós e conexões através de um diagrama que representava o mapa da cidade, como pode ser visto na Figura 4.2. Esse enigma consistia em responder a seguinte pergunta: alguém pode atravessar todas as sete pontes e nunca atravessar a mesma duas vezes?

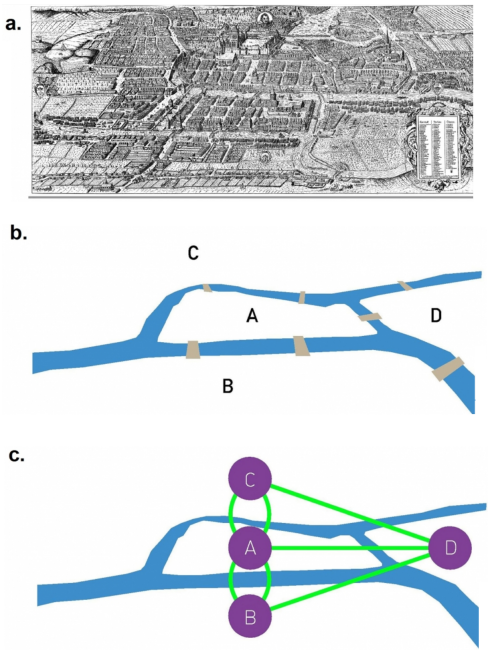


Figura 4.2: [18] (a) Mapa contemporâneo de Königsberg (agora Kaliningrado, Rússia). Durante o tempo de Euler era capital da Prússia Oriental. (b) Esquema ilustrativo dos quatro terrenos de Königsberg e das sete pontes sobre eles. (c) Grafo construido por Euler. Ele pensou em uma configuração com quatro vértices (A, B, C, D), cada um correspondendo a um pedaço de terra, e sete links representando as sete pontes. Ele então mostrou que não há caminho contínuo que cruzasse as sete pontes e nunca cruzasse a mesma ponte duas vezes.

O problema permaneceu por muito tempo, quando Leonard Euler, ofereceu uma prova matemática rigorosa de que esse caminho não existe. Euler resolveu o problema mostrando que não há caminho contínuo que cruzasse as sete pontes e nunca passasse pela mesma ponte duas vezes. A demonstração matemática desta solução foi apresentada por Euler para os membros da Academia Petersburgo em 26 de Agosto, 1735, e publicado em 1736 no artigo entitulado *Solutio Problematis ad Geometriam Situs Pertinentis* (A solução a um problema relacionado com a geometria da posição) [1].

A Figura 4.3 traz uma animação interessante que o Livro "Network Science"[2] apresenta sobre o problema das sete pontes e sua solução.

Figura 4.3: Simulação do Problema das Pontes de Königsberg - Clique aqui para ver o vídeo

Matematicamente um grafo é o conjunto $G = (V,E)$ onde V são os vértices, $V = v_1, v_2, ..., v_N$, e E arestas, $E = e_1, e_2, ..., e_M$. As conexões de uma rede podem ser dirigidas ou não dirigidas 4.4. Nas redes dirigidas, o sentido das conexões é levado em consideração, o que não acontece nas redes não dirigidas.

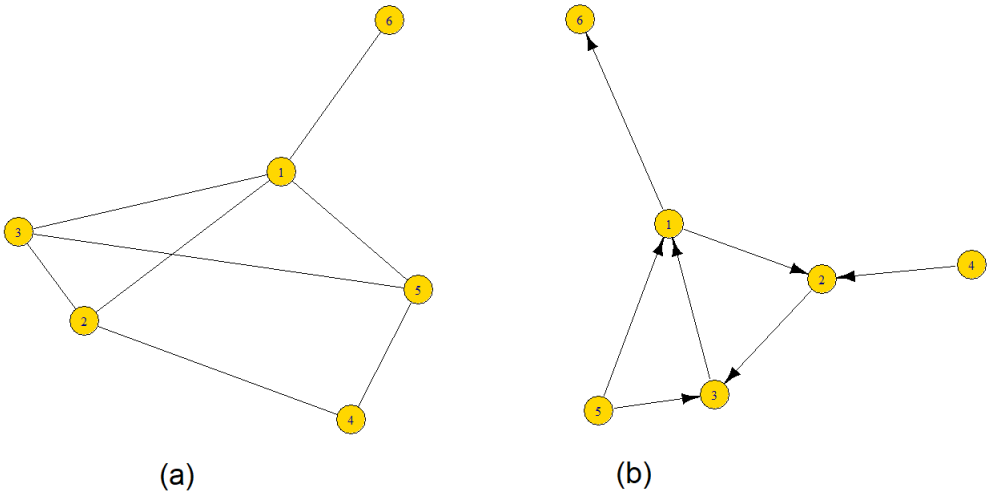


Figura 4.4: (a) Grafo não direcionado e (b) Grafo direcionado

4.3 Implementando grafos com o R

Para implementar computacionalmente um grafo podemos usar uma lista de arestas ou uma matriz de adjacências.

Grafos usando lista de arestas

Para criar um grafo a partir de uma lista de arestas, devemos escrever uma sequência onde cada par de caracteres representa a ligação de dois vértices. Por exemplo, a sequência: 1,2,2,3,3,1,4,2 representa a

ligação do vértice 1 ao 2, do 2 ao 3, do 3 ao 1, e do 4 ao 2.

No R podemos usar o comando make_graph(). Para trabalhar com as rotinas desse tutorial devemos carregar o pacote Igraph (veja no capítulo 1.3). A rotina abaixo cria e faz a plotagem do grafo a partir da lista mencionada anteriormente em nosso exemplo:

```
library(igraph)
g1<-make_graph(edges=c(1,2,2,3,3,1,4,2), directed=FALSE)
plot(g1)
```

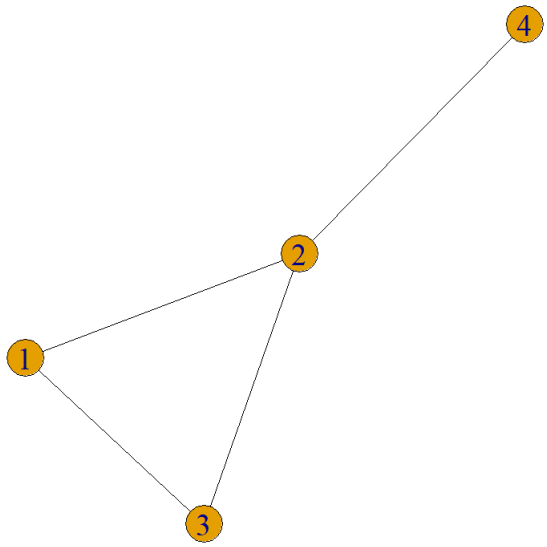


Figura 4.5: Grafo não direcionado gerado pelo make_graph

Observe que no comando make_graph() usamos os argumentos *edges* e *directed*:

edges: É um vetor que define as arestas. A primeira aresta aponta do primeiro elemento para o segundo, a segunda aresta do terceiro para o quarto, etc. Para um vetor numérico, estes são interpretados como ids de vértices internos. Para vetores de caracteres, eles são interpretados como nomes de vértices.

directed: Para criar um grafo direcionado faça directed = "TRUE" ou "T", caso contrário faça directed="FALSE" ou "F". Você pode usar dir no lugar de directed.

Você pode encontrar mais detalhes sobres os outros argumentos do em make_graph() `https://igraph.org/r/doc/make_graph.html`

OUTROS EXEMPLOS:

Grafo direcionado:

```
library(igraph)
g2<-make_graph(edges=c(1,2,2,3,3,1,4,2), directed=TRUE)
plot(g2)
```

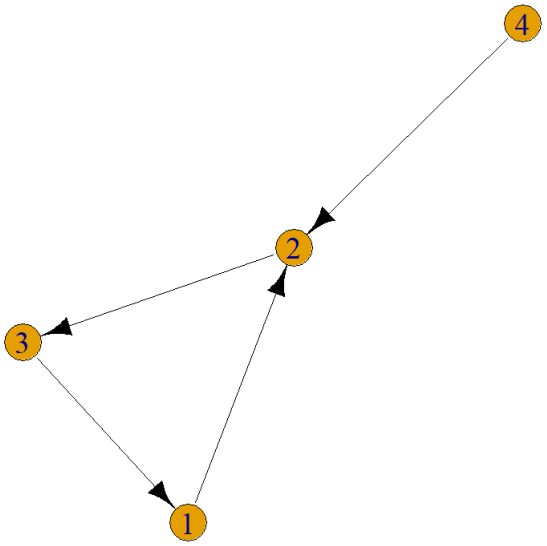


Figura 4.6: Grafo direcionado gerado pelo make_graph

Grafos com rótulos não numéricos:

```
library(igraph)
g3<-make_graph(edges=c("Bahia","Recife","Recife","Aracaju","Sao Paulo","Aracaju","
    Sao Paulo","Bahia"), directed=TRUE)
plot(g3)
```

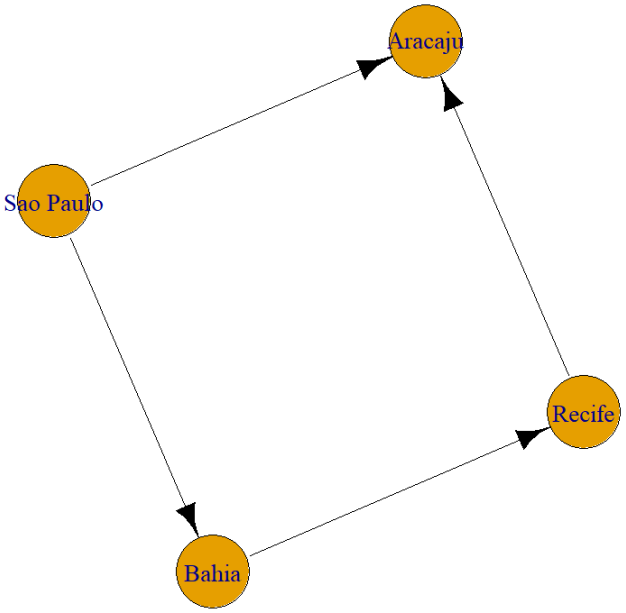


Figura 4.7: Grafos com rótulos gerado pelo make_graph

```
g4<-make_graph(edges=c("A","B","B","C","C","D","A","D","B","D"),directed=FALSE)
plot(g4)
```

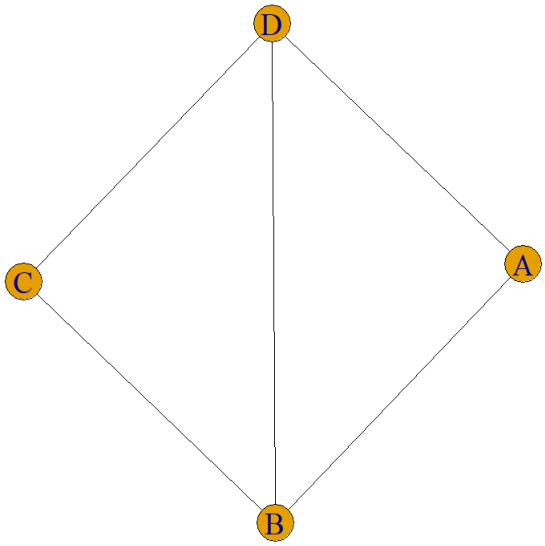


Figura 4.8: Grafos com rótulos não numéricos gerado pelo make_graph

EXERCÍCIOS

1. Crie no RStudio um grafo direcionado e um grafo não direcionado a partir de uma lista de arestas. Use os comando make_graph() e plot().
2. Use lista de arestas para representar no R uma rede de envio e recebimento de mensagens em um aplicativo de celular (semelhante ao WatsApp ou Telegram), de modo que, se a pessoa A manda uma mensagem para a pessoa B, cria-se uma aresta direcionada de A para B.

Lista de arestas usando arquivo no formato pajek

O RStudio é capaz de ler vários tipos de formatos de arquivos, incluindo o pajek.

O formato pajek é um formato para dados de rede criado originalmente para o programa de manipulação de grafos denominado pajek `http://mrvar.fdv.uni-lj.si/pajek/`.

Para criar uma rede usando lista de arestas o arquivo pajek utiliza o seguinte formato:

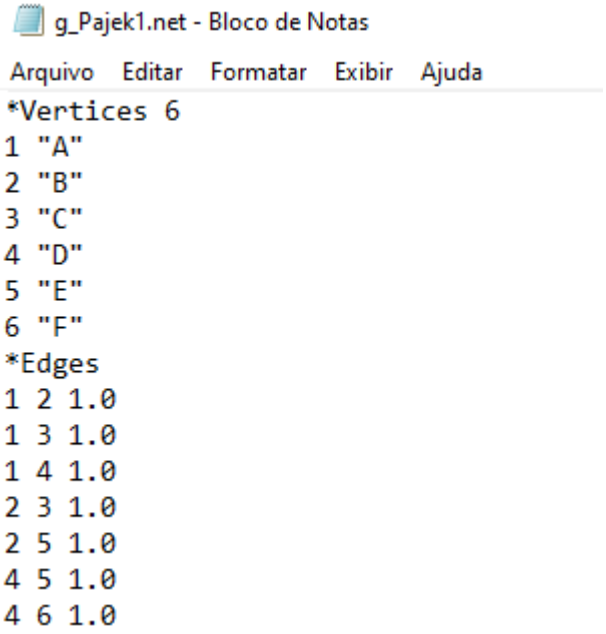


Figura 4.9: Arquivo .net criado a partir de um txt

Esse arquivo é criado abrindo-se um txt, e após inserir o conteúdo, salva-se o com a extensão .net.

O arquivo começa com a coluna *Vertices 6 declarando o label dos vértices e o número de vértices (6). Na coluna *Edges a primeira linha é o vértice de label “A”, representado pelo número 1, que está ligado ao vértice “B” (número 2). O terceiro elemento da primeira linha é o peso do vértice 1—2, que em todas as linhas foi atribuído como 1. Esse arquivo é criado abrindo-se um txt, e após inserir o conteúdo, em seguida, salvando com a extensão do pajek, que é .net.

Para abrir, ler e plotar esse arquivo .net no RStudio, usamos:

```
1 library(igraph) # Carrega o pacote igraph
2 setwd("D:/Grafo em Pajek") # muda o diretorio para o diretorio onde esta o .net
3 g=read.graph("g_Pajek1.net", format="pajek")
4 plot(g)
```

Como resultado obtemos

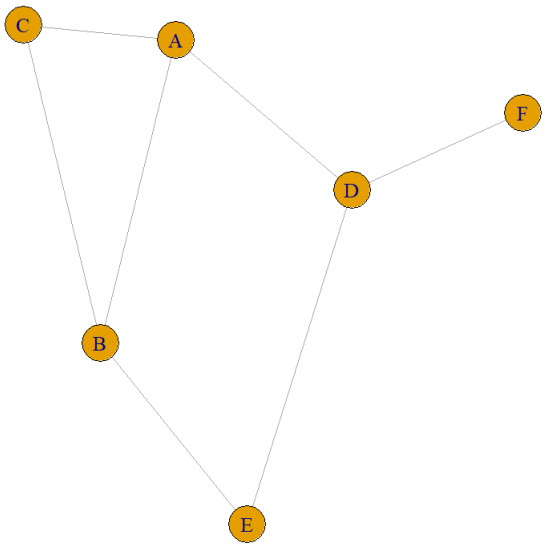


Figura 4.10: Grafo usando arquivo pajek

Nesse próximo exemplo está ilustrado um arquivo .net onde os vértices não têm label:

g_Pajek2.net - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

```
*Vertices 6
1
2
3
4
5
6
*Edges
1 2 1.0
1 3 1.0
1 4 1.0
2 3 1.0
2 5 1.0
4 5 1.0
4 6 1.0
```

Figura 4.11: Arquivo pajek

```
1 library(igraph)
2 g=read.graph("g_Pajek2.net", format="pajek")
3 plot(g)
```

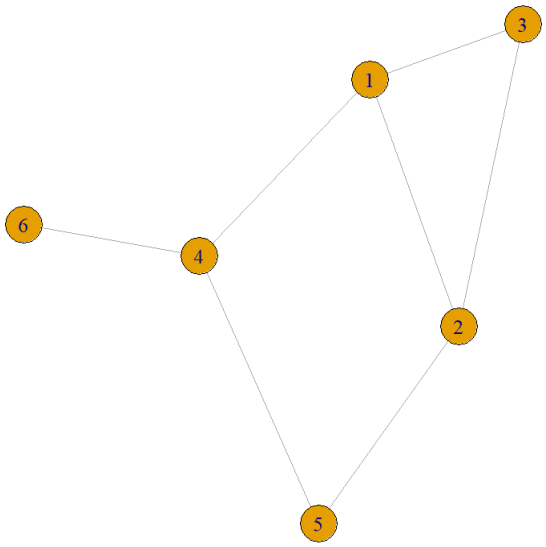


Figura 4.12: Grafo usando arquivo pajek

Grafos usando matriz de adjacências

Em uma matriz de adjacência, se dois vértices i e j estão ligados, o elemento a_{ij} na matriz será igual a 1, caso contrário será 0.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} \end{bmatrix}$$

A Figura 4.13 (a) mostra a representação matricial de uma rede não-dirigida, e a Figura 4.13 (b) apresenta o exemplo para uma rede dirigida.

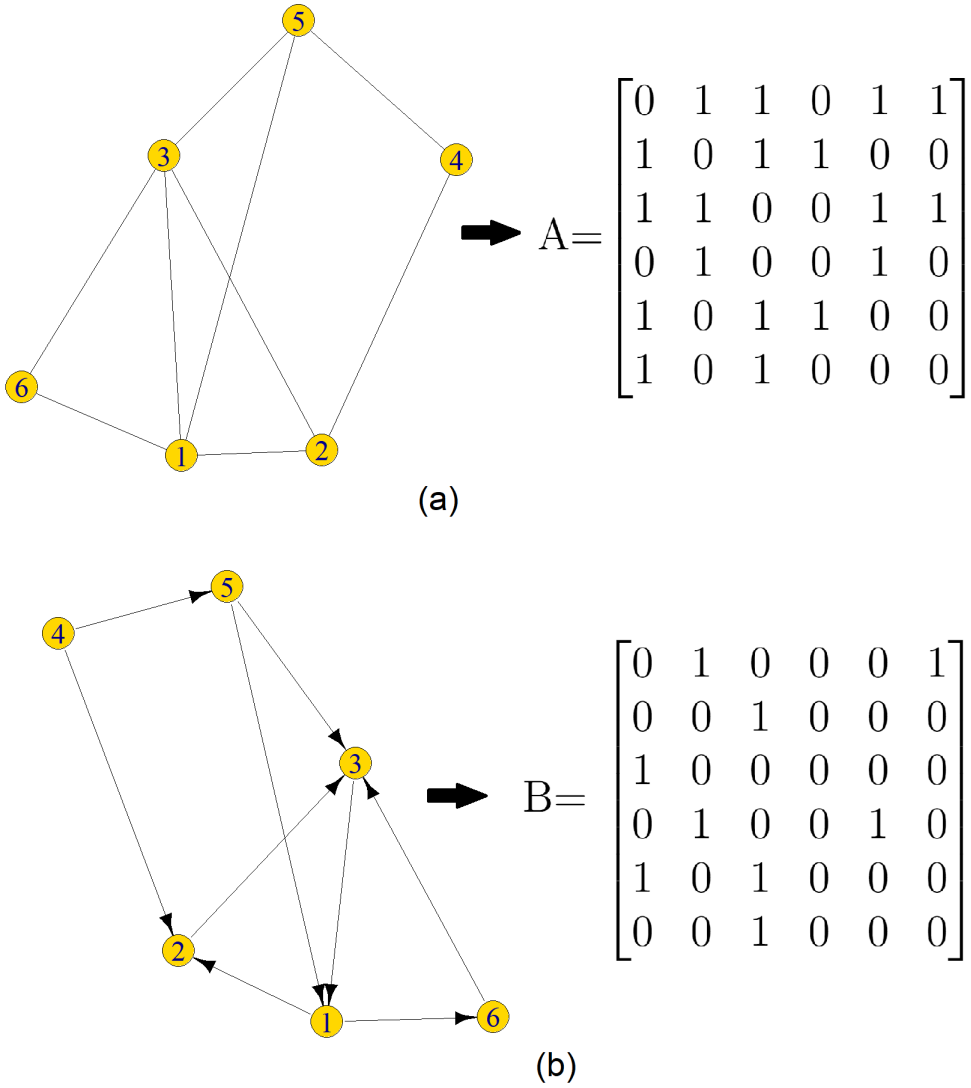


Figura 4.13: [18]Matriz de adjacência

É possível também criar matrizes adjacências de grafos no R, usando a função as_adj().

Exemplo 1

```
library(igraph)
a<-make_graph(edges=c(1,2,1,6,2,3,3,1,4,2,4,5,5,1,5,3,6,3),dir=F)
as_adj(a)
```

```
> as_adj(a)
6 x 6 sparse Matrix of class "dgCMatrix"
[1,] . 1 1 . 1 1
[2,] 1 . 1 1 . .
[3,] 1 1 . . 1 1
```

```
[4,] . 1 . . 1 .
[5,] 1 . 1 1 . .
[6,] 1 . 1 . . .
```

Exemplo 2

```
library(igraph)
b<-make_graph(edges=c(1,2,1,6,2,3,3,1,4,2,4,5,5,1,5,3,6,3),dir=T)
as_adj(b)
```

```
> as_adj(b)
6 x 6 sparse Matrix of class "dgCMatrix"
[1,] . 1 . . . 1
[2,] . . 1 . . .
[3,] 1 . . . . .
[4,] . 1 . . 1 .
[5,] 1 . 1 . . .
[6,] . . 1 . . .
```

Podemos notar no grafo não dirigido o vértice 1 está ligado aos vértices 2, 3, 5, 6, logo, na sua matriz adjacência, os elementos a_{12}, a_{13}, a_{15}, a_{16} terão valor 1, enquanto, os elementos a_{11},a_{14} terão valor 0. Uma vez que não existe loop, um vértice ligado a ele próprio, todo elementos da diagonal principal (a_{11}, a_{22}, a_{33}, a_{44}, a_{55}, a_{66}) terão valor 0. Percebemos também que cada linha representada o status de ligação dos demais vértices i com os vértices j.

Podemos encontrar a matriz de adjacências no R utilizando o comando $as_adjacency_matrix(wtc)$. A rotina implementa computacionalmente o grafo da figura 4.13a a partir de sua lista de arestas e gera a sua respectiva matriz de adjacências:

```
library(igraph)
g<-make_graph(edges=c(1,2,2,3,3,4,4,5,5,1,1,3,2,5,4),directed=FALSE)
as_adjacency_matrix(g)
```

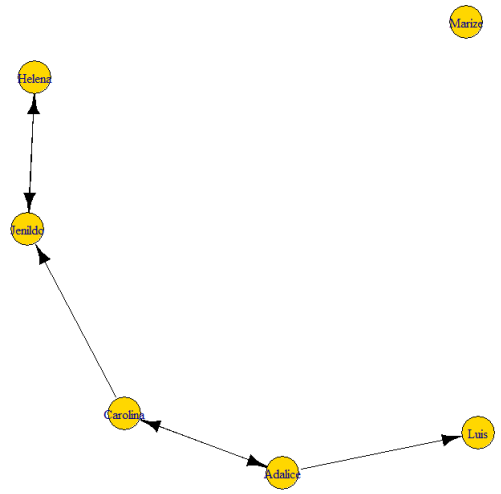


Figura 4.14: Grafo a partir de uma matriz

A rotina a seguir

```
g=read.graph("g_Pajek3.net", format="pajek")
plot(g)
```

O formato .net para expressar um grafo a partir de uma matriz de adjacência é mostrado através do exemplo abaixo:

*g_Pajek3.net - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

```
*Vertices 6
1 "Adalice"
2 "Carolina"
3 "Luis"
4 "Jenildo"
5 "Helena"
6 "Marize"
*Matrix
0.0 1.0 1.0 0.0 0.0 0.0
1.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0
```

Figura 4.15: Arquivo pajek

4.4 Grafos Notáveis

Existem algumas funções que geram alguns tipos especiais de grafos. Vejamos alguns exemplos:

Grafo Bull

O grafo Bull possui 5 vértices, 5 arestas, e lembra a cabeça de um touro se desenhado corretamente.

```
g5<-make_graph("Bull")
plot(g5)
```

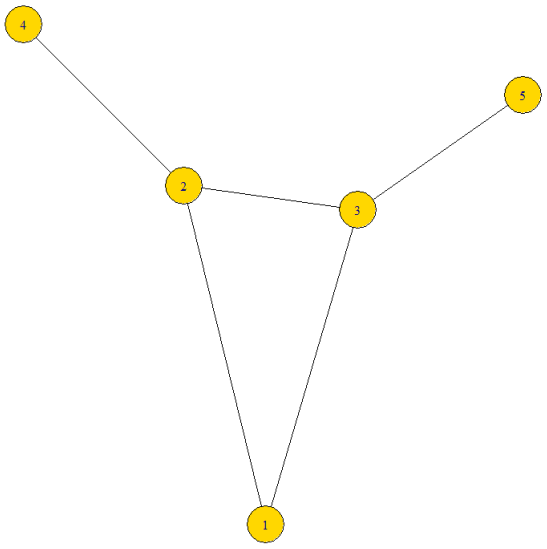


Figura 4.16: Grafo Bull

Grafo Icosahedral

O grafo icosahedral é um sólido platônico com 12 vértices e 30 arestas.

```
g6<-make_graph("icosahedral")
plot(g6)
```

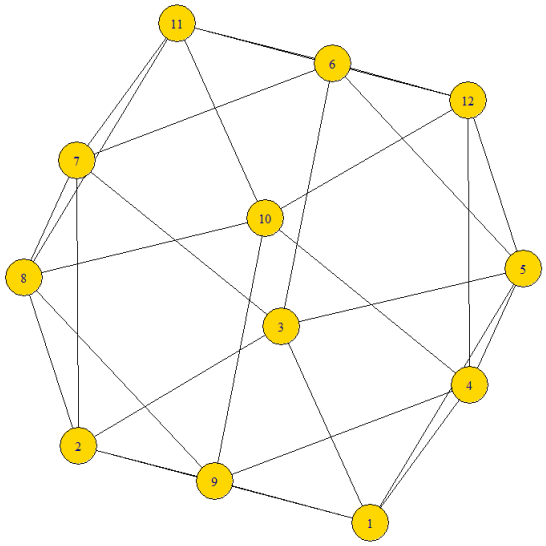


Figura 4.17: Grafo Icosahedral

Grafo Zachary

O grafo Zachary é a representação de uma rede social de amizades entre 34 membros de um clube de caratê em uma universidade dos EUA na década de 1970. [24]

```
g7<-make_graph("Zachary")
plot(g7)
```

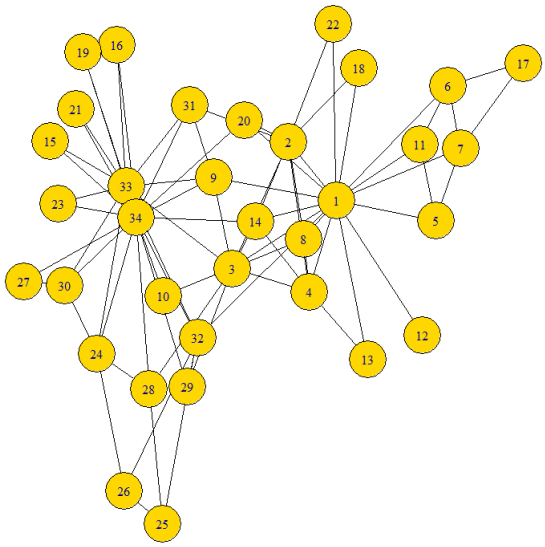


Figura 4.18: Grafo Zachary

Tree graph

```
tr <- make_tree(40, children = 3, mode = "undirected")
```

```
plot(tr, vertex.size=10, vertex.label=NA)
```

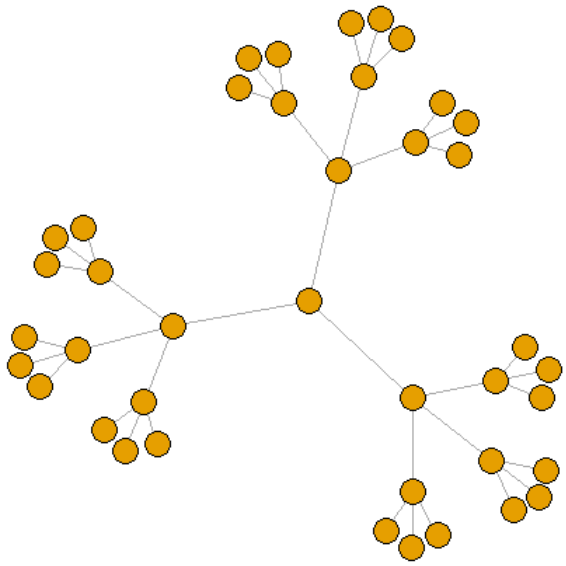


Figura 4.19: Grafo Icosahedral

Simple star graph

```
tr <- make_tree(40, children = 40, mode = "undirected")
plot(tr, vertex.size=10, vertex.label=NA)
```

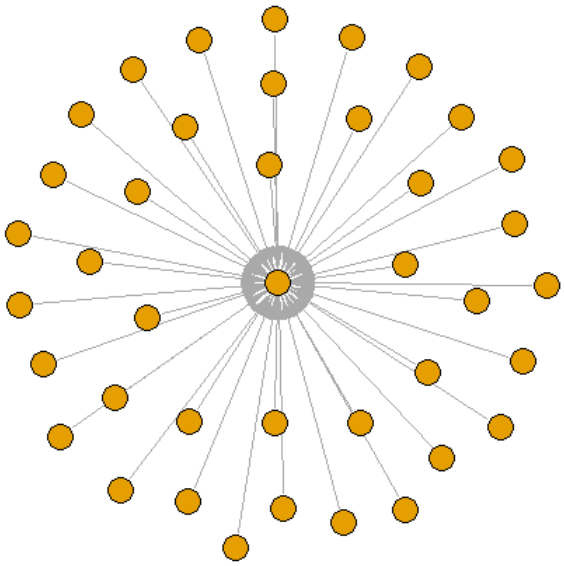


Figura 4.20: Simple star graph

5. Características de Redes Complexas

5.1 Grau

É o número de arestas de um vértice.

Denotamos com k_i o grau do i-ésimo vértice na rede. Por exemplo, para o grafo da Figura 5.1, temos que o grau do vértice 1 é três ($k_1 = 3$), do vértice 2 é quatro ($k_2 = 4$), do vértice 3 é dois ($k_3 = 2$), do vértice 4 é hum ($k_4 = 1$), do vértice 5 é dois ($k_5 = 2$).

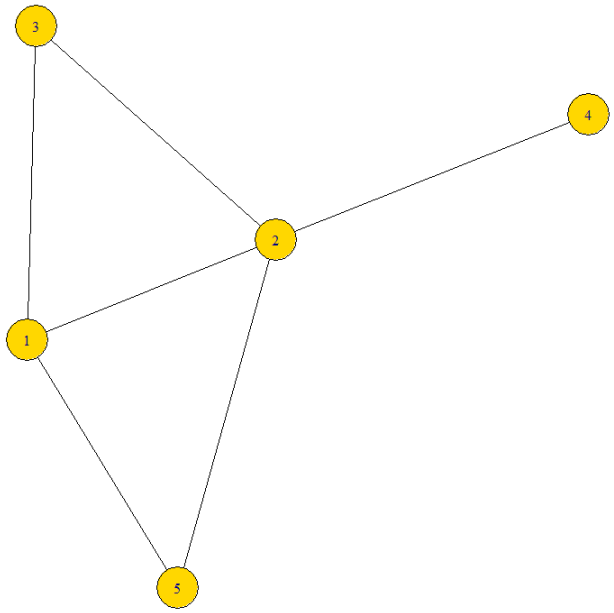


Figura 5.1: Grafo não direcionado

Em uma rede não direcionada, o número total de arestas, L, pode ser expresso como a soma dos graus dos vértices:

$$L = \frac{1}{2}\sum_{i=1}^{N} k_i \tag{5.1}$$

Aqui, o fator 1/2 corrige o fato de que na soma (2.1) cada aresta é contado duas vezes. Por exemplo, a

aresta que conecta os vértices 2 e 4 na Figura 4.19 será contado uma vez no grau de vértice 1 e uma vez no grau de vértice 4. No exemplo da Figura 5.1, o número total de arestas é:

L= $\frac{1}{2}(3+4+2+1+2)$

L= 6 Em grafos direcionados ...

No R calculamos o número de arestas usando o comando gsize().

A rotina a seguir calcula o número de arestas do grafo da Figura 5.1

```
g<-make_graph(edges=c(1,2,1,3,1,5,2,3,2,5,2,4),directed=FALSE)
gsize(g)
```

```
[1] 6
```

Para calcular o grau de cada vértice usamos o comando degree(). A função grau () tem um modo de entrada para grau, saída para grau de saída e total para grau total.

```
g<-make_graph(edges=c(1,2,1,3,1,5,2,3,2,5,2,4),directed=FALSE)
degree(g)
```

```
[1] 3 4 2 1 2
```

Grau Médio

Uma medida fundamental da estrutura de uma rede é a sua conectividade ou grau (k). O grau médio é representado por $< k >$. Essa medida informa o número médio de conexões entre os vértices. O grau de um dado vértice *i* para uma rede não-dirigida é dado por

$$k_i = \sum_{j=1}^{n} A_{ij} \tag{5.2}$$

Daí o grau médio é dado pela equação a seguir, onde N éo número total de vértices:

$$\langle k \rangle = \frac{1}{N} \sum_{j=1}^{n} k_i \tag{5.3}$$

No exemplo da Figura 5.1, o grau médio é:

k= $\frac{1}{5}(3+4+2+1+2)$

k= 2.4

No R calculamos o grau médio usando o comando mean(degree()).
A rotina a seguir calcula o grau médio do grafo da Figura 5.1

```
g<-make_graph(edges=c(1,2,1,3,1,5,2,3,2,5,2,4),directed=FALSE)
mean(degree(g))
```

No console:

```
[1] 2.4
```

5.2 Coeficiente de Aglomeração (transitividade)

O coeficiente de aglomeração (*Clustering Coefficient*) C_i é um número real $0 \leq C_i \leq 1$, que captura o nível de conexão entre os k vizinhos de um determinado vértice i. O coeficiente de aglomeração de um determinado vértice i (coeficiente de aglomeração local) é calculado pela razão entre a quantidade de arestas existentes entre os k vizinhos de i (E_k), e o número total de arestas possíveis entre vizinhos $\frac{k(k-1)}{2}$. O C_i mede a densidade das conexões locais da rede: quanto mais densamente interconectada a vizinhança do vértice i, maior é seu coeficiente de aglomeração local. O coeficiente de aglomertação também é conhecido como transitividade.

$$\frac{\text{Número de arestas entre os k vizinhos de um dado vértice i}}{\text{Número máximo possível de arestas entre os k vizinhos de i}} = \frac{E_k}{\frac{k(k-1)}{2}} \tag{5.4}$$

o coeficiente de aglomeração médio da rede é:

$$\langle C \rangle = \frac{1}{N} \sum_{i=1}^{N} C_k \tag{5.5}$$

Na Figura 5.2 temos três grafos e os respectivos coeficientes de aglomeração do vértice i.

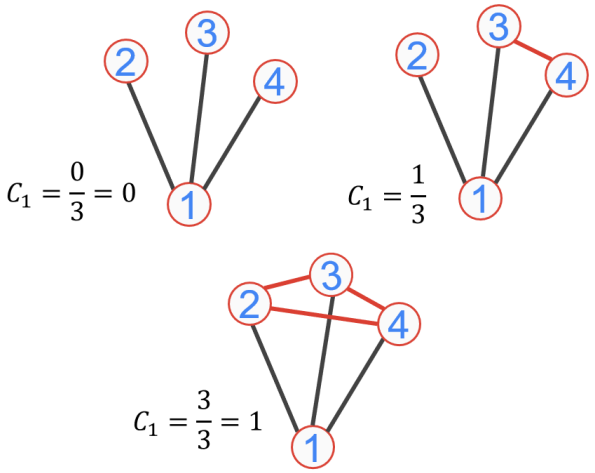


Figura 5.2:

- C_1=0, significa que nenhum dos vizinhos do vértice 1 (os vértices 2, 3 e 4) se conectam.
- $C_1 = \frac{1}{3}$, indica a razão entre a aresta que existe entre os vértices 3 e 4, e todas as possíveis arestas que podem ser formadas entre 2, 3 e 4.
- $C_1 = 1$ quer dizer que todos os vizinhos do vértice 1, se conectam, ou seja, formama um grafo completo.

C_i pode ser interpretado como a probabilidade de dois vértices vizinhos a i se conectarem. Na Figura 5.3, os coeficientes C_1 e C_3 indicam que existe 100% de chance dos vizinhos dos vértices 1 e 3 estarem conectados; os resultados de C_4, C_5 e C_7, mostram que as chance dos vizinhos dos vértices 4, 5 e 7, estarem conectados são, respectivamente, 30%, 66,7%, 50%.

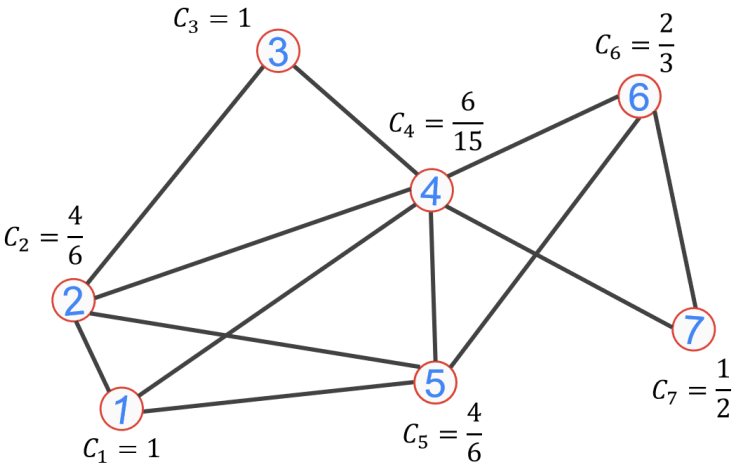


Figura 5.3:

De forma mais detalhada temos:

- $C_1 = \frac{1}{\frac{2(2-1)}{2}} = 1$
- $C_2 = \frac{4}{\frac{4(4-1)}{2}} = \frac{4}{6}$
- $C_3 = \frac{1}{\frac{2(2-1)}{2}} = 1$
- $C_4 = \frac{6}{\frac{6(6-1)}{2}} = \frac{6}{15}$
- $C_5 = \frac{4}{\frac{4(4-1)}{2}} = \frac{4}{6}$
- $C_6 = \frac{2}{\frac{3(3-1)}{2}} = \frac{2}{3}$
- $C_7 = \frac{1}{\frac{2(2-1)}{2}} = \frac{1}{2}$

Calculando o coeficiente de aglomeração médio da rede temos:

$$\langle C \rangle = \frac{1}{7} \sum_{i=1}^{7} C_k = 0,605 \tag{5.6}$$

É muito comum encontrar em redes reais, a ocorrência de subgrafos formados por três vértices totalmente conectados. O coeficiente de aglomeração em uma dada rede social, por exemplo, indica a probabilidade de dois amigos quaisquer, terem um terceiro amigo em comum.

O comando R para calcular o coeficiente de aglomeração é o transitivity().

transitivity(graph, type = c("global", "local", "average"), vids = NULL, isolates = c("NaN", "zero"))

Argumentos

graph	grafo a ser calculado a transitividade
type	O tipo de transitividade a ser calculada. Valores possíveis:"global", "local"
vids	id do vértice que se pretende calcular a transitividade local
isolates	Valor para ser atribuído a vértices isolados. Melhor colocar "zero"no caso de transitividade local

Exemplo

Calculando no R o coeficiente de aglomeração médio do grafo:

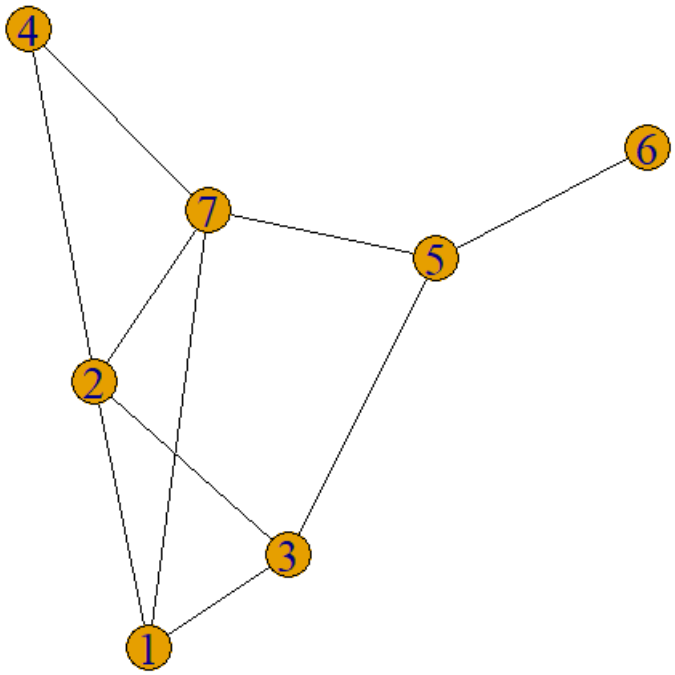


Figura 5.4:

```
library(igraph)
g<-make_graph(edges=c(1,2,1,3,2,3,3,5,5,6,5,7,7,1,7,2,7,4,2,4),directed=FALSE)
transitivity(g, type="average")
```

Visualizando o resultado no console:

```
> transitivity(g, type="average")
[1] 0.4722222
```

Para calcular o coeficiente de aglomeração local em cada vértice:

```
library(igraph)
g<-make_graph(edges=c(1,2,1,3,2,3,3,5,5,6,5,7,7,1,7,2,7,4,2,4),directed=FALSE)
C1<-transitivity(g, type="local",vids=1,isolates = "zero")
C2<-transitivity(g, type="local",vids=2,isolates = "zero")
C3<-transitivity(g, type="local",vids=3,isolates = "zero")
C4<-transitivity(g, type="local",vids=4,isolates = "zero")
C5<-transitivity(g, type="local",vids=5,isolates = "zero")
C6<-transitivity(g, type="local",vids=6,isolates = "zero")
C7<-transitivity(g, type="local",vids=7,isolates = "zero")
```

Visualizando a resposta no console:

```
> C1
[1] 0.6666667
> C2
```

```
[1] 0.5
> C3
[1] 0.3333333
> C4
[1] 1
> C5
[1] 0
> C6
[1] 0
> C7
[1] 0.3333333
```

Na literatura sobre redes, além do coeficiente de aglomeração local, encontramos o coeficiente de aglomeração global. A transitividade global é a proporção dos triângulos e dos triplos conectados no grafo. Para grafo direcionado, a direção das arestas é ignorada. O coeficiente de aglomeração global é dado por:

$$C_{\Delta} = \frac{\text{3 X Número de Triângulos}}{\text{Número de Triplas Conectadas}} \tag{5.7}$$

Para calcular a transitividade global no R:

```
library(igraph)
g<-make_graph(edges=c(1,2,2,3,3,1,4,2),directed=FALSE)
transitivity(g, type="global")
```

No console:

```
> transitivity(g, type="global")
[1] 0.4090909
```

Comparando os valores do coeficiente de aglomeração médio e o coeficiente de aglomeração global do grafo do nosso exemplo, podemos observar que $< C >$ e C_{Δ} não iguais. Isso ocorre pois o coeficiente de aglomeração médio e o global não são equivalentes.

5.3 Densidade

Seja n o número de vértices de uma rede. A densidade é a razão entre o número total de arestas (E) e o número de arestas possíveis de uma rede.

$$\Delta = \frac{E}{\frac{n(n-1)}{2}} \tag{5.8}$$

Para calcular a densidade do grafo da Figura 5.5 fazemos $\Delta = \frac{10}{\frac{7(7-1)}{2}} = 0{,}476$. Podemos dizer que o grafo possui uma densidade de $47{,}6\%$.

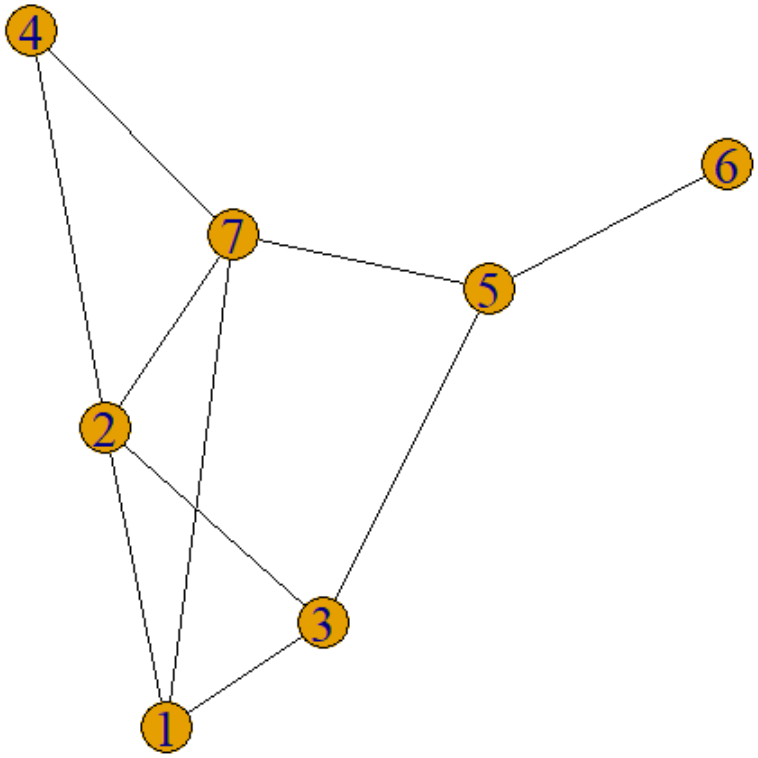


Figura 5.5:

No R a densidade pode ser calculada utilizando o comando edgedensity().

Exemplo

A rotina a seguir calcula a densidade do grafo e faz a plotagem do grafo da Figura 5.5

```
library(igraph)
g<-make_graph(edges=c(1,2,1,3,2,3,3,5,5,6,5,7,7,1,7,2,7,4,2,4),directed=FALSE)
plot(g)
edge_density(g, loops = FALSE)
```

Visualização do console:

```
[1] 0.4761905
```

Outros Exemplos

Grafo com densidade nula

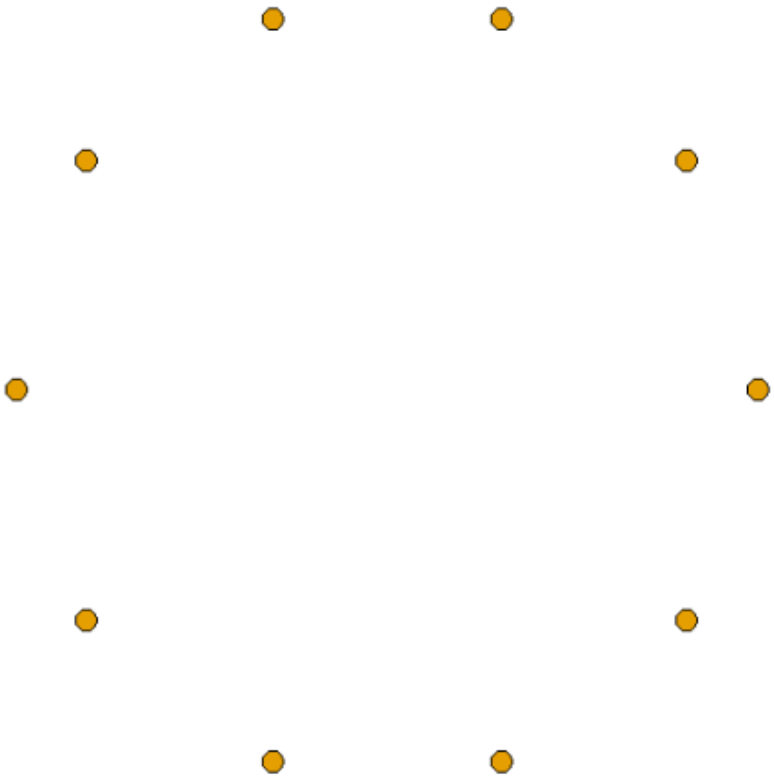


Figura 5.6:

Rotina R para grafo de densidade nula:

```
library(igraph)
library(cowplot)
g<-erdos.renyi.game(10,0, type = c("gnm"), directed = FALSE)
plot(g,vertex.size=6,vertex.label=NA,vextex.frame.color="black",edge.color="black",
     vertex.label.font=6,edge.width=1.5,layout=layout_in_circle)
edge_density(g, loops = FALSE)
```

Console:

```
[1] 0
```

Grafo com densidade 100 %

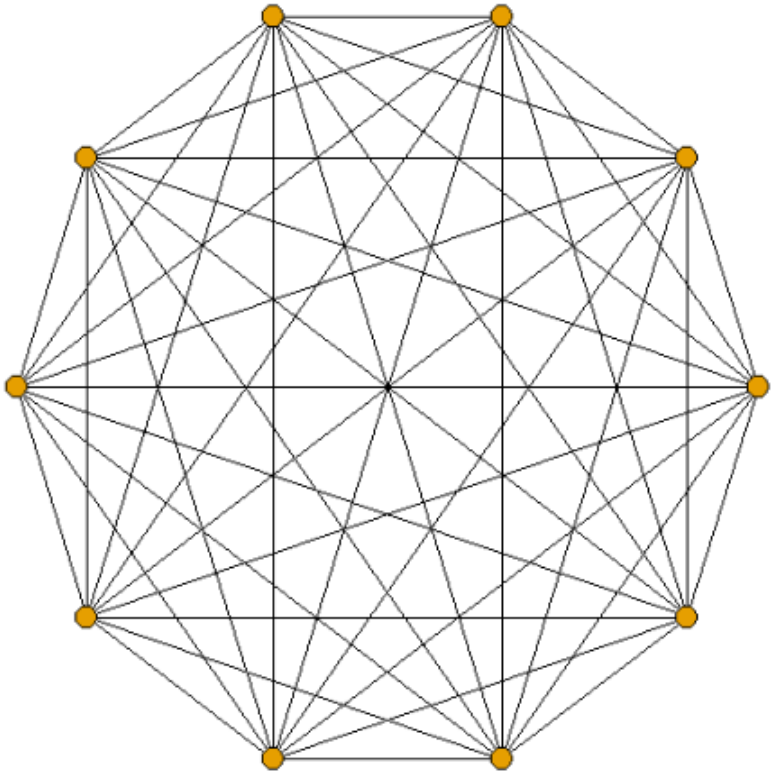


Figura 5.7:

Rotina R para grafo de densidade 100%:

```
library(igraph)
library(cowplot)
g<-erdos.renyi.game(10, 1, type = c("gnp"), directed = FALSE)
plot(g,vertex.size=6,vertex.label=NA,vextex.frame.color="black",edge.color="black",
     vertex.label.font=6,edge.width=1.5,layout=layout_in_circle)
edge_density(g, loops = FALSE)
```

Console:

```
[1] 1
```

5.4 Caminhos e distâncias

As distâncias entre vértices em uma rede podem afetar a qualidade do seu fluxo de informações. Na rede de computadores da internet, quanto maior a distância entre dois computadores, maior a quantidade de roteadores no caminho, resultando no aumento da probabilidade de falhas.

Caminho

Um caminho entre dois vértices i e j é uma lista de arestas ordenadas que traçam um percurso entre i e j. A Figura 5.8 mostra um grafo em (a) e todos os seus caminhos do vértice 1 ao vértice 5 ((b), (c) e (d)). Esses caminhos são, respectivamente:

- $1 \rightarrow 2 \rightarrow 3 \rightarrow 5$
- $1 \rightarrow 2 \rightarrow 5$
- $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$

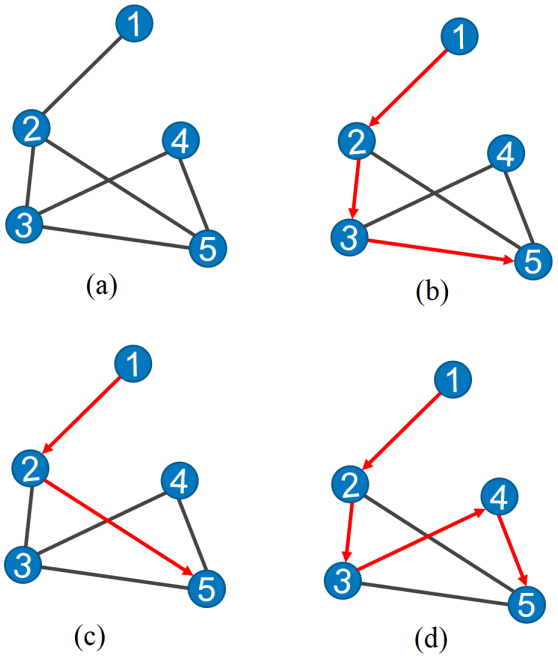


Figura 5.8: Caminhos

Para medir o comprimento do caminho entre dois vértices i e j de uma rede, devemos contar o número de arestas ao longo do caminho entre esses dois vértices i e j. Na Figura 5.8 (d) o caminho $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$ tem comprimento 4.

Na modelagem computacional com o R, usamos a função $all_simple_paths(g,i,j)$ para determinar todos os caminhos entre os vértices i e j, a função $shortest_paths(g,i,j)$ para determinar o menor caminho entre os vértices i e j. A rotina a seguir calcula essas características entre os vértices 1 e 5 do grafo da FIgura 5.8.

```
1 library(igraph)
2 g<-make_graph(edges=c(1,2,2,3,3,4,4,5,2,5,3,5),directed=FALSE)
3 all_simple_paths(g,1,5)
4 shortest_paths(g,1,5)
```

Distância (Caminho Geodésico)

O *comprimento* do menor caminho entre i e j é chamado de *distância* entre i e j, designado por d_{ij}. Na Figura 5.8, o menor caminho entre os vértices 1 e 5 é $1 \rightarrow 2 \rightarrow 5$, que possui duas arestas. Logo a distância $d_{15} = 2$.

Nesse mesmo grafo $d_{41} = 3$, $d_{23} = 1$ e $d_{33} = 0$.

O conjunto das distâncias entre todos os pares de vértices em uma rede podem ser representadas através de uma matriz de distâncias D, cujos elementos dij expressam o valor da distância entre os vértices i e j.

Para um grafo com 5 vértices, como no nosso exemplo, temos a seguinte matriz:

$$D = \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} \\ d_{21} & d_{22} & d_{23} & d_{24} & d_{25} \\ d_{31} & d_{32} & d_{33} & d_{34} & d_{35} \\ d_{41} & d_{42} & d_{43} & d_{44} & d_{45} \\ d_{51} & d_{52} & d_{53} & d_{54} & d_{55} \end{bmatrix}$$

Na matriz D a diagonal principal representa o caminho de que liga um vértice a ele próprio. Se não há loops no grafo, a diagonal principal (d_{ii}) da matriz D é igual a zero. Em grafos não-direcionados as distâncias d_{ij} e d_{ji} são iguais, e a matriz D é simétrica. Já em grafos direcionados, d_{ij} e d_{ji} não são necessariamente iguais.

Ao preencher com as distâncias d_{ij}, temos:

$$D = \begin{bmatrix} 0 & 1 & 2 & 3 & 2 \\ 1 & 0 & 1 & 2 & 1 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 2 & 1 & 0 & 1 \\ 2 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Diâmetro

O valor $d_{max} = max d_{ij}$ é chamado diâmetro da rede. No nosso exemplo $d_{max} = 3$. O diâmetro é a máxima distância entre i e j, ou seja é o maior caminho mínimo entre i e j.

Caminho Mínimo médio

Seja N o número de vértices de um grafo, O caminho mínimo médio da rede é dado por:

$$l = \frac{1}{N(N-1)} \sum_{i \neq j} d_{ij} \tag{5.9}$$

O caminho mínimo médio da Figura 5.8 (a) é $l = \frac{1}{5(5-1)} 30 = \frac{30}{20} = 1,5$

No R, usamos: $distances(g,i,j)$ para determinar a distância entre os vértices i e j: $distances(g)$ para determinar a matriz de distâncias do grafo g; $mean_distance(g)$ para calcular o caminho mínimo médio de g; $diameter(g)$ para calcular o diâmetro do grafo. A rotina a seguir calcula essas três características para a Figura 5.8 (a).

```
library(igraph)
g<-make_graph(edges=c(1,2,2,3,3,4,4,5,2,5,3,5),directed=FALSE)
distances(g,1,5)
distances(g)
mean_distance(g)
diameter(g)
```

```
> distances(g,1,5)
     [,1]
[1,]    2
> distances(g)
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    2    3    2
[2,]    1    0    1    2    1
[3,]    2    1    0    1    1
[4,]    3    2    1    0    1
[5,]    2    1    1    1    0
> mean_distance(g)
[1] 1.5
> diameter(g)
[1] 3
```

Ciclo

Caminho que começa e termina no mesmo vértice.

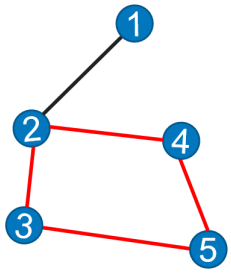


Figura 5.9: caminho $2 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 2$

Caminho Euleriano

Um caminho que atravessa cada aresta exatamente uma vez.

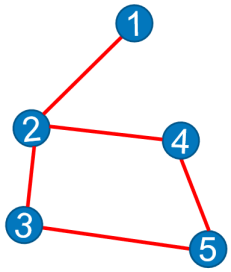


Figura 5.10: caminho $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 2$

Caminho Hamiltoniano

Um caminho que visita cada vértice exatamente uma vez.

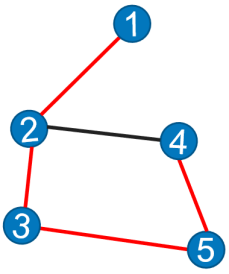


Figura 5.11: caminho $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 4$

5.5 Modularidade

A modularidade é uma medida de estrutura das redes. Esta medida foi projetada para medir a força da divisão de uma rede em módulos (ou comunidades). Redes com alta modularidade têm conexões densas entre os vértices dentro dos módulos, mas ligações esparsas entre vértices em diferentes módulos [4].

Um valor alto de modularidade indica que a densidade das arestas dentro das comunidades é maior que o esperado ao acaso, indicando uma boa partição da rede. Segundo [4] a modularidade é definida por:

$$Q = \frac{1}{2m} \sum_{(i,j)} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j) \tag{5.10}$$

onde i e j são os vertices da rede; A_{ij} representa o número de arestas entre i e j; k_i e k_j são a soma das arestas ligadas a i e j; m é a soma de todas as arestas da rede. (c_i, c_j) é a função delta de Kronecker (Equação 5.11); onde c_i e c_j são as comunidades dos vértices.

$$(c_i, c_j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases} \tag{5.11}$$

O cálculo da modularidade pode ser interpretado como a diferença entre a densidade das conexões de um dado conjunto de vértices, e a densidade das conexões sobre o mesmo conjunto tomadas aletoriamente. Os desvios sistemáticos calculados pela Equação 5.10 nos permite definir a qualidade das partições da rede [18].

A Figura 5.12 mostra várias partições de uma rede com duas comunidades óbvias. Assim fica mais simples entender a diferença entre modularidade e comunidade.

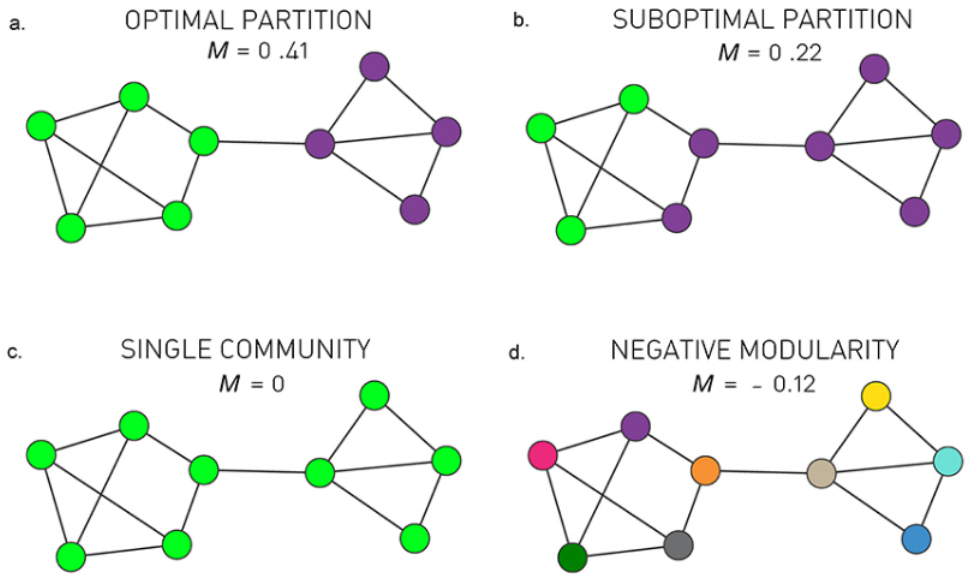


Figura 5.12: Ilustração do livro Network Science[2]: Partição ideal - A partição com modularidade máxima M = 0,41 corresponde intimamente às duas comunidades distintas. Partição subótima - Uma partição com uma modularidade abaixo do ideal, mas positiva, M = 0,22, falha ao identificar corretamente as comunidades presentes na rede. Comunidade Única - Se atribuirmos todos os vértices à mesma comunidade, obtemos M = 0, independentemente da estrutura da rede. Modularidade negativa - Se atribuirmos cada vértice a uma comunidade diferente, a modularidade é negativa, obtendo M = -0,12.

A seguir iremos mostrar a seguir o passo a passo do cálculo da modularidade para cada uma das 4 configurações de partições da Figura 5.12.

Modularidade da Configuração I

A Figura 5.13 mostra a conifguração I e a numeração atribuida às suas duas comunidades: c_1 na cor verde e c_2 na cor roxa.

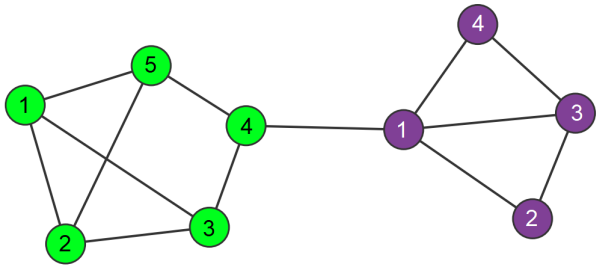


Figura 5.13: Configuração I

Matriz de adjacências de c_1:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Na matriz de adjacências A da comunidade c_1, quando $a_{ij} = 1$ significa que existe uma aresta entre os vértices i e j desse subgrafo. Se $a_{ij} = 0$ não existe ligação entre i e j (Ver seção 4.3). A matriz A representa a densidade inerente a c_1, enquanto os valores de $\frac{k_i k_j}{2m}$ representam a densidade tomada ao acaso nessa comunidade. Para fazer esse cálculo, foram considerados os graus de c_1: $k_1 = k_2 = k_3 = k_4 = k_5 = 3$, e a quantidade de arestas da comunidade original ($m = 13$). Pelo fato de termos graus com o mesmo valor, as densidades para cada par de vértices têm o mesmo valor. Como exemplo, vamos calcular a densidade ao acaso para os vértices 2 e 5 de c_1: $\frac{k_2 \cdot k_5}{2 \cdot 13} = \frac{4 \cdot 4}{26} = 0.35$

A Tabela 5.5 apresenta a diferença entre a modularidade inerente (A_{ij}) e a modularidade ao acaso ($\frac{k_i k_j}{2m}$) para todos os pares de vértices da comunidade c_1 (subgrafo verde). Como exemplo, segue o cálculo dessa diferença para os vértices 2 e 5: $A_{25} - \frac{k_2 \cdot k_5}{2 \cdot 13} = 1 - 0.35 = 0.65$.

$A_{ij} - (k_i \cdot k_j)/(2m)$					
	1	**2**	**3**	**4**	**5**
1	-0.35	0.65	0.65	-0.35	0.65
2	0.65	-0.35	0.65	-0.35	0.65
3	0.65	0.65	-0.35	0.65	-0.35
4	-0.35	-0.35	0.65	-0.35	0.65
5	0.65	0.65	-0.35	0.65	-0.35

Tabela 5.1: Diferença de densidades entre vértices do subgrafo c_1

A modularidade Q_1 é encontrada ao calculando $Q_1 = \frac{1}{2m} \sum_{(i,j)} \left(A_{ij} - \frac{k_i k_j}{2m} \right) = \frac{1}{2 \cdot 13} \cdot 5,35 = 0,20562$ da Tabela5.5.

Vamos agora calcular a modularidade da comunidade c_2 (cor roxa).

A Matriz de adjacências de c_2 é:

$$B = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

A Tabela 5.5 apresenta a diferença entre os elementos de B (B_{ij}) e a modularidade ao acaso ($\frac{k_i k_j}{2m}$) da comunidade c_2 (subgrafo roxo) da Figura 5.13. Os graus de c_2 são: $k_1 = 4, k_2 = 2, k_3 = 3, k_4 = 2$, e a quantidade de arestas do grafo completo $m = 13$.

$B_{ij} - (k_i \cdot k_j)/(2m)$				
	1	2	3	4
1	-0,62	0,69	0,54	0,69
2	0,69	-0,15	0,77	-0,15
3	0,54	0,77	-0,35	0,77
4	0,69	-0,15	0,77	-0,15

Tabela 5.2: Diferença de densidades entre vértices do subgrafo c_2

O valor da modularidade do subgrafo c_2 é $Q_2 = \frac{1}{2m} \sum_{(i,j)} \left(B_{ij} - \frac{k_i k_j}{2m} \right) = \frac{1}{2 \cdot 13} \cdot 5,35 = 0,20562$. Para encontrar a modularidade total é preciso somar as modularidades dos subgrafos.

$$Q_{total} = Q_1 + Q_2 = 0.20562 + 0.20562 = 0,41124$$

Modularidade da Configuração II

A Figura 5.14 mostra a configuração II e a numeração dos vértices para c_1 na cor verde, e c_2 na cor roxa.

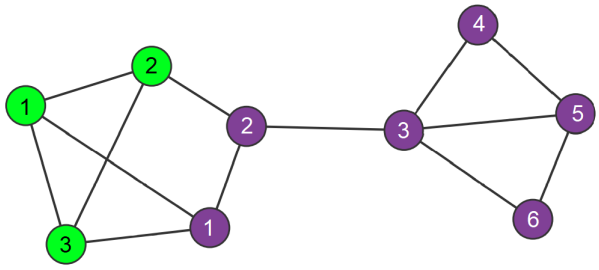


Figura 5.14: Configuração II

Matriz de adjacências de c_1:

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

A matriz A representa a densidade inerente a c_1, enquanto os valores de $\frac{k_i k_j}{2m}$ representam a densidade tomada ao acaso nessa comunidade. Para fazer esse cálculo, foram considerados os graus de c_2: $k_1 = k_2 = k_3 = 3$, e a quantidade de arestas da comunidade original ($m = 13$). O valor de $\frac{k_i k_j}{2m}$ para cada par de

vértices da comunidade c_1 é 0,35. A Tabela 5.5 mostra a diferença entre a modularidade intrínseca e a modularidade ao acaso na comunidade de cor verde.

$A_{ij} - (k_i \cdot k_j)/(2m)$			
	1	**2**	**3**
1	-0.35	0.65	0.65
2	0.65	-0.35	0.65
3	0.65	0.65	-0.35

Tabela 5.3: Diferença de densidades entre vértices do subgrafo c_1

A modularidade da comunidade c_1 é a média da diferença entre as densidades inerente e ao acaso. Aplicando a Equação 5.10, temos: Q_1= 0.1109

Matriz de adjacências de c_2:

$$B = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

A matriz B representa a densidade inerente a c_2, enquanto os valores de $\frac{k_i k_j}{2m}$ representam a densidade tomada ao acaso nessa comunidade. Para fazer esse cálculo, foram considerados os graus de c_2: $k_1 = 3, k_2 = 3, k_3 = 4, k_4 = 2, k_5 = 3, k_6 = 2$, e a quantidade de arestas da comunidade original ($m = 13$).

$A_{ij} - (k_i \cdot k_j)/(2m)$						
	1	**2**	**3**	**4**	**5**	**6**
1	-0,35	0,65	-0,46	-0,23	-0,35	-0,23
2	0,65	-0,35	0,54	-0,23	-0,35	-0,23
3	-0,46	0,54	-0,62	0,69	0,54	0,69
4	-0,23	-0,23	0,69	-0,15	0,77	-0,15
5	-0,35	-0,35	0,54	0,77	-0,35	0,77
6	-0,23	-0,23	0,69	-0,15	0,77	-0,15

Tabela 5.4: Diferença de densidades entre vértices do subgrafo c_2

O valor da modularidade do subgrafo c_2 é dado por $Q_2 = \frac{1}{2m} \sum_{(i,j)} \left(A_{ij} - \frac{k_i k_j}{2m} \right) .1 = 0,11091$. Para se encontrar a modularidade total é preciso somar as modularidades dos subgrafos.

$$Q_{total} = Q_1 + Q_2 = 0,1109 + 0,1109 = 0,2219$$

Modularidade da Configuração III

A Figura 5.15 mostra a configuração III e a numeração dos vértices para c_1 na cor verde.

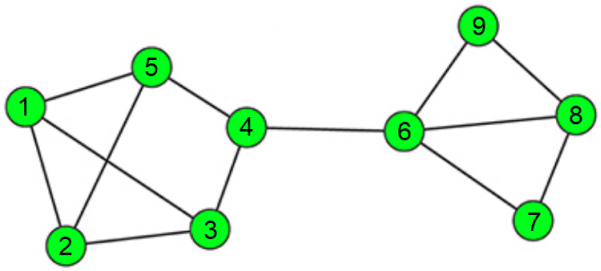


Figura 5.15: Configuração III

Matriz de adjacências de c_1:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

A matriz A representa a densidade inerente a c_1, enquanto os valores de $\frac{k_i k_j}{2m}$ representam a densidade tomada ao acaso nessa comunidade. Para fazer esse cálculo, foram considerados os graus de c_1: $k_1 = 3, k_2 = 3, k_3 = 3, k_4 = 3, k_5 = 3, k_6 = 4, k_7 = 2, k_8 = 3, k_9 = 2$, e a quantidade de arestas da comunidade original ($m = 13$). Como exemplos, vamos calcular a densidade ao acaso para os vértices 6 e 1 de c_1: $\frac{k_6 \cdot k_1}{2 \cdot 13} = \frac{4 \cdot 3}{26} = 0.46$ e os vértices 3 e 7 de c_1: $\frac{k_3 \cdot k_7}{2 \cdot 13} = \frac{3 \cdot 2}{26} = 0.23$.

A Tabela 5.5 apresenta a diferença entre a modularidade inerente (A_{ij}) e a modularidade ao acaso ($\frac{k_i k_j}{2m}$) para todos os pares de vértices da comunidade c_1 (subgrafo verde).

$A_{ij} - (k_i \cdot k_j)/(2m)$									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	-0,35	0,65	0,65	-0,35	0,65	-0,46	-0,23	-0,35	-0,23
2	0,65	-0,35	0,65	-0,35	0,65	-0,46	-0,23	-0,35	-0,23
3	0,65	0,65	-0,35	0,65	-0,35	-0,46	-0,23	-0,35	-0,23
4	-0,35	-0,35	0,65	-0,35	0,65	0,54	-0,23	-0,35	-0,23
5	0,65	0,65	-0,35	0,65	-0,35	-0,46	-0,23	-0,35	-0,23
6	-0,46	-0,46	-0,46	0,54	-0,46	-0,61	0,69	0,54	0,69
7	-0,23	-0,23	-0,23	-0,23	-0,23	0,69	-0,15	0,77	-0,15
8	-0,35	-0,35	-0,34	-0,35	-0,35	0,54	0,77	-0,35	0,77
9	-0,23	-0,23	-0,23	-0,23	-0,23	0,69	-0,15	0,77	-0,15

Tabela 5.5: Diferença de densidades entre vértices do subgrafo c_1

O valor da modularidade do subgrafo verde $Q_1 = Q_{total} = \frac{1}{2m}\sum_{(i,j)}\left(A_{ij} - \frac{k_i k_j}{2m}\right).1 = 0,00.$

Modularidade da Configuração IV

A Figura 5.16 mostra a configuração IV e a numeração de cada uma das comunidades. Nesse caso, como cada comunidade posssui apenas um único vértice, as suas matrizes de adjacência são matrizes nulas 1×1. Consequentemente, na equação $Q = \frac{1}{2m}\sum_{(i,j)}\left(A_{ij} - \frac{k_i k_j}{2m}\right)$, o termo A_{ij} será zero, e para todas as comunidades, $i = j$. Daí calcularemos a modularidade através da fórmula $Q = \frac{1}{2m}\sum_{i=1}^{9}\left(-\frac{k_i k_i}{2m}\right)$.

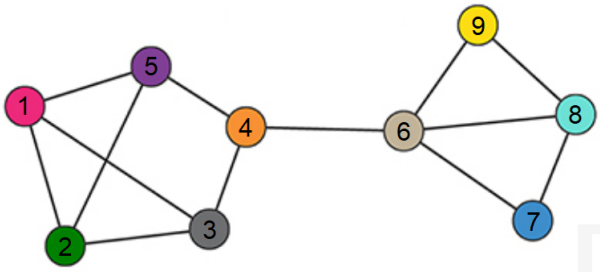


Figura 5.16: Configuração IV

Para a comunidade c_1, a modularidade é dada por $Q_1 = -\frac{k_1 k_1}{2m} = -\frac{3\times 3}{2\times 13} = -0,34615.$

Analogamente,

$Q_2 = Q_3 = Q_4 = Q_5 = -0,34615;$

$Q_6 = -0,61538;$

$Q_7 = -0,15385;$

$Q_8 = -0,34615;$

$Q_9 = -0,15385;$

Daí $Q = \frac{1}{2\times 13}\sum_{\varepsilon=1}^{9} Q_\varepsilon = -0,12.$

5.6 Distribuição de graus

A distribuição de frequências dos graus de um grafo fornece informações muito importantes para a caracterização de redes complexas. A informação mais relevante é aquela que calcula a probabilidade de que um nó selecionado aleatoriamente na rede que tenha grau k. Aqui iremos mostrar essa idéia usando a frequência relativa em uma distribuição de graus. Essa ideia pode ser depois generalizada para o cálculo da probabilidade.

Considere o grafo a seguir.

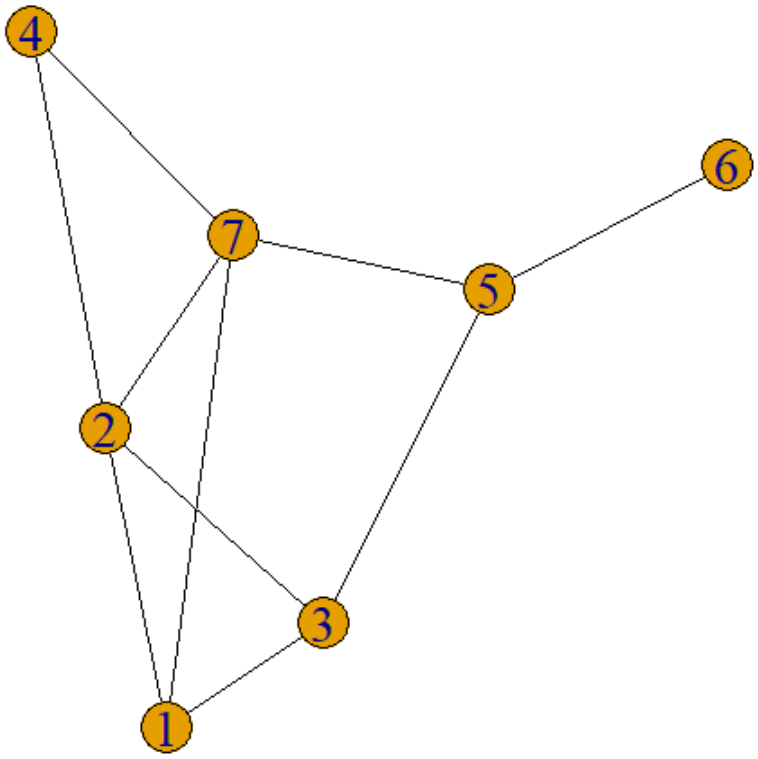


Figura 5.17:

A tabela 5.6 mostra, nas duas primeiras colunas, o grau, a frequência absoluta (f) - quantidade de vértices que possuem esse grau no grafo da Figura 5.17, e a frequência relativa (f_r) - razão entre frequência absoluta e o total de vértices.

Grau	Freq. Absoluta f	Freq. Relativa f_r
1	1	0,14
2	1	0,14
3	3	0,43
4	2	0,29

Tabela 5.6:

Note que a soma das frequências relativas é 1, ou seja $\sum_1^n f_r = 1$.

A rotina a seguir faz a plotagem da distribuição graus do grafo com sua frequ encia relativa.

```
g<-make_graph(edges=c(1,2,1,3,2,3,3,5,5,6,5,7,7,1,7,2,7,4,2,4),directed=FALSE)
grau<-degree(g)
hist(grau,prob=T,main="",xlab="Grau",ylab="fr",ylim=c(0,0.5),border=T,labels=F,
    breaks=c(1,2,3,4))
```

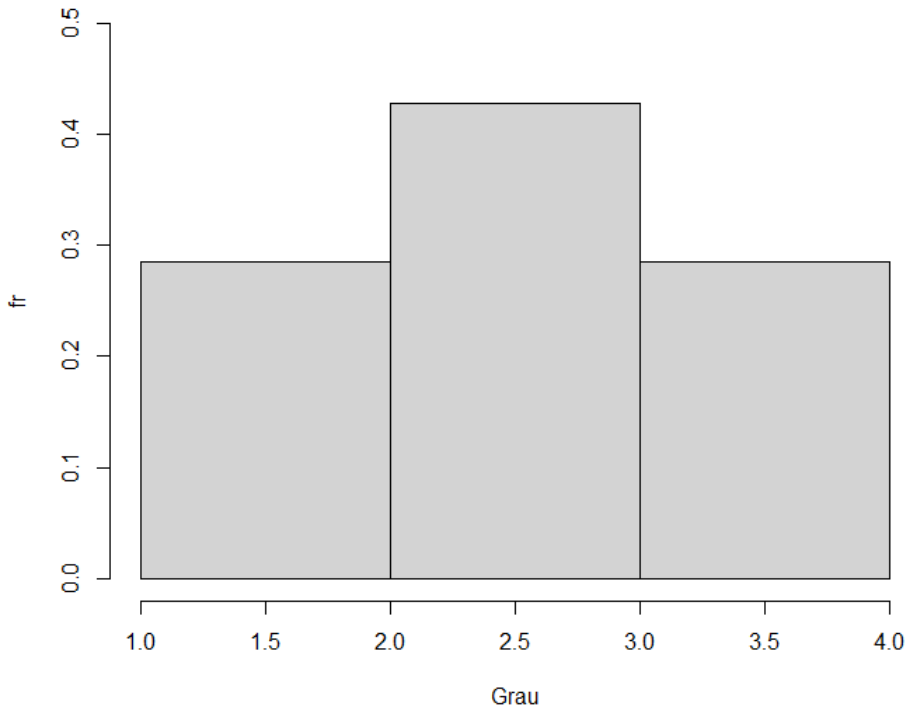


Figura 5.18: Distribuição de graus

Representação de Algumas Grafos e suas respectivas Distribuições de Graus

1. Grafo da Rede Zachary

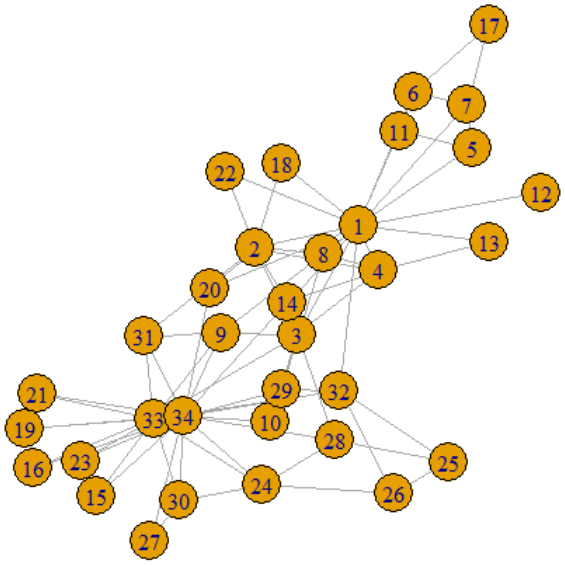


Figura 5.19:

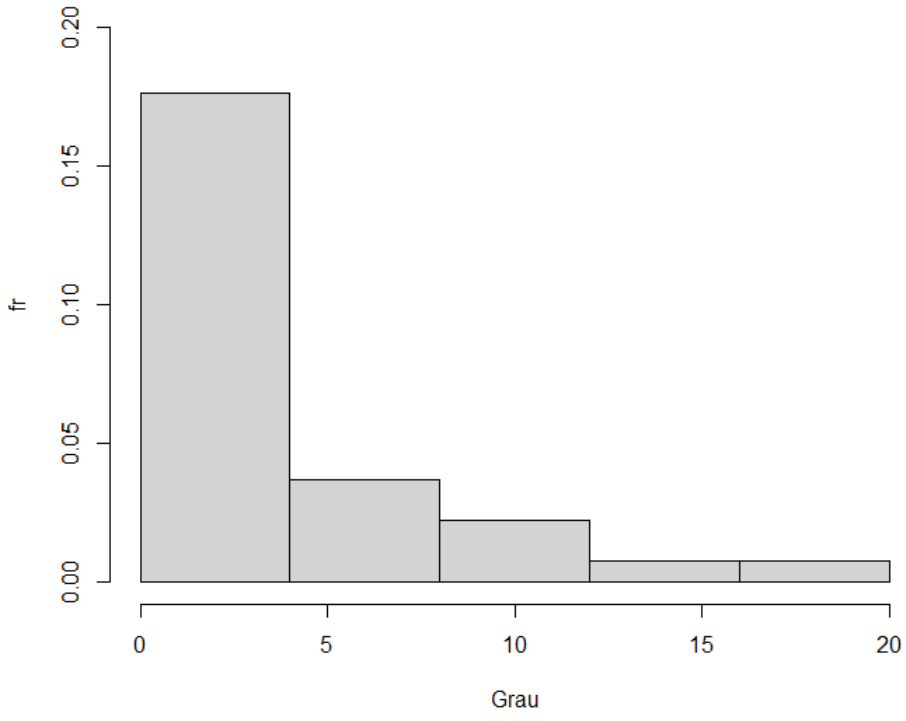


Figura 5.20:

6. Tipologia de Redes Complexas

Existem três tipos de redes que têm se tornado a base para a análise de redes complexas: Redes aleatórias, Redes Small World, e Redes Livres de Escala. Dentre as características que são usadas para distinguir essas tipos de redes, a mais importante é a distribuição de graus.

6.1 Grafos Aleatórios

Os grafos aleatórios podem ser gerados a partir dos seguintes modelos:

- Modelo G (N, m): N vértices rotulados são conectados com m arestas colocadas aleatoriamente. Esse modelo foi proposto em uma série de artigos publicados entre 1959 e 1968 escritos por Paul Erdős e Alfred Rényi. Neles foi apresentada a fusão entre probabilidade e teoria dos grafos, que foi considerado como o lançamento da teoria dos grafos aleatórios [2].
- Modelo G (N, p): Cada par de N vértices rotulados está conectado com a probabilidade p. Apesar de ter sido proposto por Edgar Nelson Gilbert esse modelo geralmente é atribuido aos autores Erdős e Rényi [2].

O comando R para gerar grafos aleatórios de acordo com o modelo Erdös-Renyi é erdos.renyi.game().

Uso	erdos.renyi.game(n, p.or.m, type = c("gnp", "gnm"), directed = FALSE, loops=FALSE, ...)
Argumentos	
N	Número de vértices do grafo
p.or.m	Admite tanto o modelo G (N, p) ou como o modelo G (N, m).
type	O tipo de grafo a ser criado: G(N, p) ou G(N, m).
directed	Parâmetro lógico; Grafo direcionado (V) ou não-direcionado (F). O default é F.
loops	Parâmetro lógico: Logical; V para admissão de loops. O default é falso.

Muitos autores preferem o modelo G(N, p) porque ele permite calcular características importantes da rede e também porque redes reais geralmente não mantém fixo o número de arestas. Dados n>0, $n \in \mathbb{N}$, e uma probabilidade p, $0 \leq p \leq 1$. Definimos o grafo G(N, p) como grafo não dirigido de n vértices e para cada par u, v $\in$ V(b) existe uma aresta (u,v) com probabilidade p.

Para formarmos um grafo aleatório com probabilidade p, tomamos inicialmente n vértices desconectados. Depois sorteamos um número $p_0 \in [0,1]$. Se $p_0 > p$ dois vértices são conectados por uma aresta, caso contrário eles não se conectam.

O comando R para gerar grafos aleatórios de acordo com o modelo Erdös-Renyi é erdos.renyi.game().

O código R a seguir gera e faz a plotagem um grafo aleatório segundo o modelo Erdös – Renyi para N=5 e p=0.5. No RStudio

```
library(igraph)
g<-erdos.renyi.game(5, 0.5, type = c("gnp", "gnm"), directed = FALSE,loops = FALSE)
plot(g,vertex.size=6,vertex.label=NA,vextex.frame.color="black",edge.color="black",
    vertex.label.font=6,edge.width=1.5,layout=layout_in_circle)
```

A Figura 6.1 ilustra a construção de grafos aleatórios a partir de vértices desconectados, chegando a um grafo conectado. Pode-se notar que, à medida que se aumenta o valor de p, o grafo vai ficando mais denso.

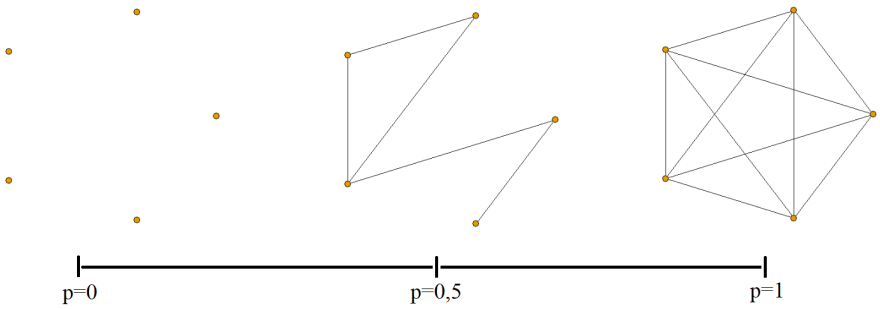


Figura 6.1:

Alternativamente grafos aleatórios podem ser modelados usando os comando sample_gnp() e sample_gnm().

```
sintaxe: sample_gnm(n,m)
```

Argumentos do comando sample_gnm():

n - é o número de vértices

m - é o número de arestas

```
sintaxe: sample_gnp(n,p)
```

Argumentos do comando sample_gnp():

n - é o número de vértices

p - é a probabilidade de conexão de dois vértices

Script para gerar um um grafo aleatório com 50 vértices e 70 arestas.

```
ga<-sample_gnm(n=50,m=70)
plot(ga, vertex.size=6, vertex.label=NA)
```

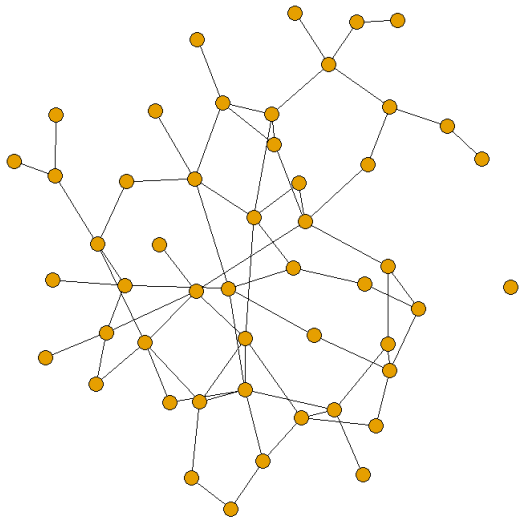


Figura 6.2: Grafo Aleatório

A distribuição de graus de um grafo aleatório, para valores de n "grandes", se aproxima de uma distribuição de Poisson [2]. A Figura mostra a distribuição de graus de uma rede aleatória Erdös – Renyi com $n = 10^4$ e grau médio $< k >= 50$.

```
g<-sample_gnm(n=10000, m=250000)
plot(g, vertex.size=6, vertex.label=NA, vextex.frame.color="black",edge.color="black",
vertex.label.font=6,edge.width=1.5)
hist(degree(g),main="Distribuicao de graus",xlab = "k",ylab="p",freq=F)
lines(density(degree(g)))
mean(degree(g))
```

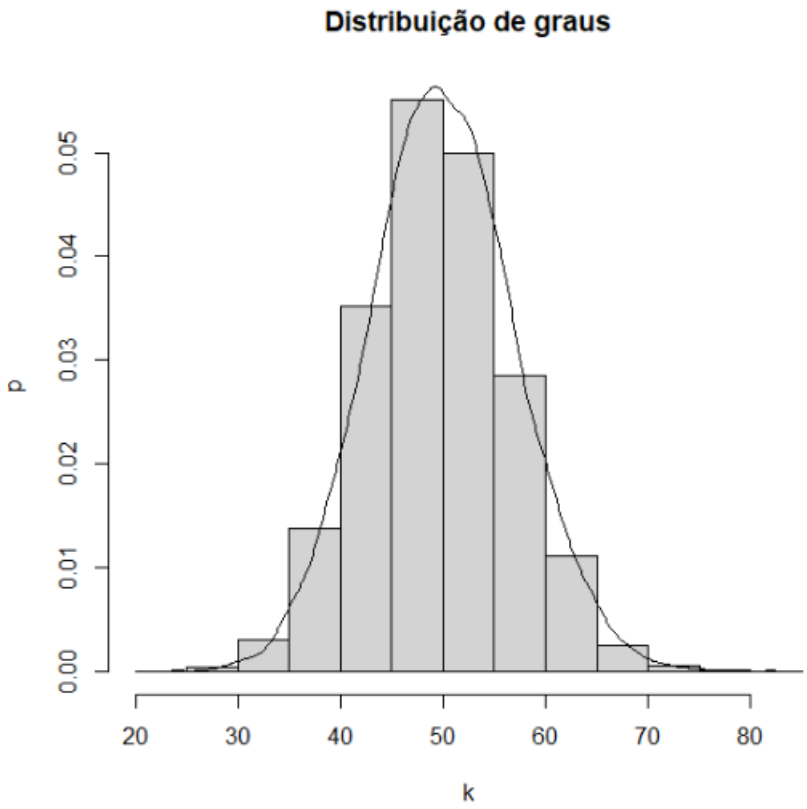


Figura 6.3: Distribuicao de Graus de um grafo aleatorio

6.2 Redes Small World

No ano de 1967, o sociólogo Stanley Milgram desenvolveu um experimento que tinha como objetivo testar a interconectividade das redes sociais. Milgram entregou 160 cartas com destinatários aos participantes da pesquisa. As cartas deveriam passar para alguém próximo, que conhecesse os destinatários, ou outra pessoa que talvez pudessem deixar mais próxima do destino, e assim continuamente, até chegar aos destinatários. Do total de cartas enviadas, 42 chegaram ao destino e, com isso, Milgram determinou o caminho médio que separava duas pessoas quaisquer nos Estados Unidos. Essa distância foi determinada como sendo 5,5 e arredondada para seis. Esse número ficou conhecido como *os seis graus de separação*, e o fenômeno a ele associado como *pequeno mundo* (small world). Segundo esse princípio, a distância média entre um monge no Nepal e um capoeirista no pelourinho, em Salvador, deve ser em torno de seis. Estudos recentes apontam um número menor para esse grau de separação rodrigues2007caracterizaccao. A descoberta de Milgram se tornou base para o estudo de redes sociais e de outras redes complexas.

Em 1988, Steven Strogatz e Duncan Watts desenvolveram um experimento semelhante ao de Milgram, e sugeriram um modelo que tem características estatísticas, mas com propriedades que se enquadram no conceito de construção de comunidades sociais. Eles desenvolveram modelos para compreender o

que Milgram também encontrou no seu experimento “ seis graus de separação". O modelo criado por Watts-Strogatz ficou conhecido como “rede de mundo pequeno” (rede small world), este modelo se encontra entre uma rede regular e uma rede aleatória 6.4. Neste tipo de rede existe um caminho “médio baixo” que possibilita a conexão de um vértice com todos, a partir de poucos passos.

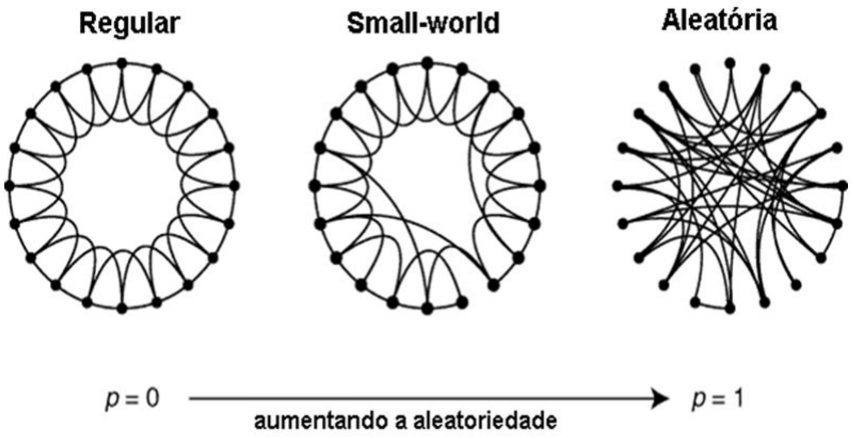


Figura 6.4: comportamento da rede em função da aleatoriedade [22]

Watts-Strogatz small-world model

No R, os grafos Small world são formados usando o comando sample_smallworld().

```
sintaxe: sample_smallworld(dim=,size=,nei=,p= )
```

Argumentos do comando sample_smallworld ():

dim - é uma constante inteira, a dimensão da rede inicial, quantidade de ligação dos vértices

size - é uma constante inteira, o tamanho da rede ao longo de cada dimensão

nei - é uma constante inteira, a vizinhança dentro da qual os vértices da rede serão conectados

p - é uma constante real entre zero e um, a probabilidade de religação

Exemplo de Grafo Mundo Pequeno usando a linguagem R

```
ws<-sample_smallworld(dim=2,size=5,nei=1,p=0.2)
plot(ws,vertex.size=6,vertex.label=NA,layout=layout_in_circle)
```

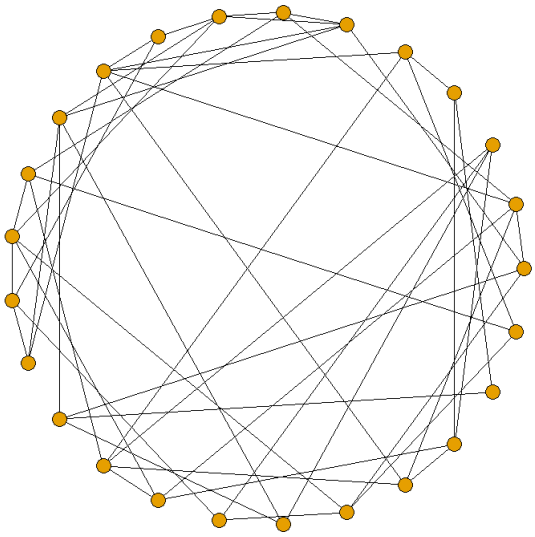


Figura 6.5: Grafo Mundo Pequeno gerado no RStudio

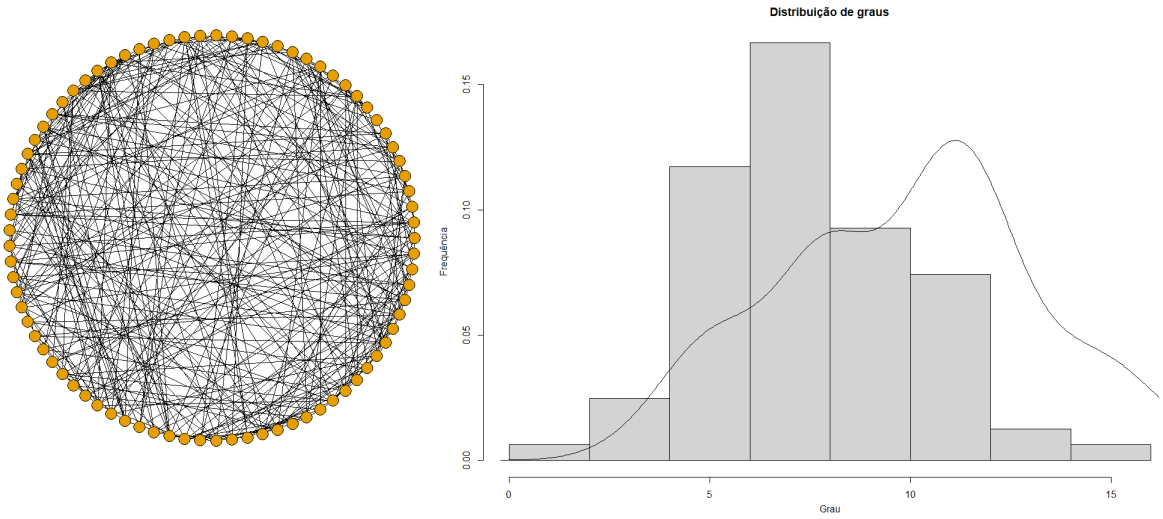


Figura 6.6: Grafo Mundo Pequeno e Distribuição de graus

No ano de 2005, o físico Albert-László Barabási queria entender como funcionavam as redes de internet, e criou um modelo chamado rede livre de escala, baseado em dois elementos: crescimento e vinculação preferencial. Barabási analisou que os vértices com mais conexões têm mais chance de receber novos vértices. Vários cientistas como sociólogos, psicólogos, biólogos entres outros, perceberam que o modelo de Barabási era uma ótima ferramenta para estudar assuntos complexos das áreas.

A partir do modelo Barabási, os estudos com redes complexas se tornaram uma ferramenta importante de análise, com objetivos de analisar estatisticamente, verificar propriedades, formular modelos em grafos e matemáticos que representam a rede estudada, e estudar comportamentos de adição ou exclusão de

vértices na rede.

No R, as Redes Livres de Escala são formados usando o comando sample_pa().

```
sintaxe: sample_pa(n=,power=,m=,directed=)
```

Argumentos do comando sample_smallworld ():

n - é o número de vértices

power - é o poder do anexo preferencial, o padrão é um

m - é a constante numérica, o número de arestas a serem adicionadas em cada intervalo de tempo

directed - é se deve criar um gráfico direcionado, o padrão é TRUE

Exemplo de Rede Livre de Escala usando a linguagem R

```
ba<-sample_pa(n=100,power=1,m=1,directed=F)
plot(ba,vertex.size=6,vertex.label=NA)
```

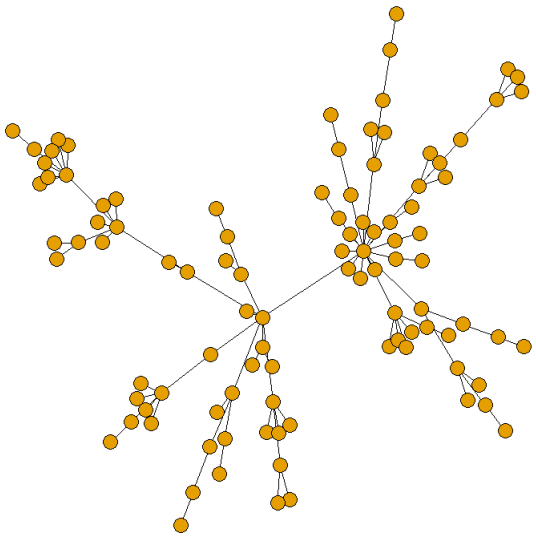


Figura 6.7: Grafo Livre de Escala gerado no RStudio

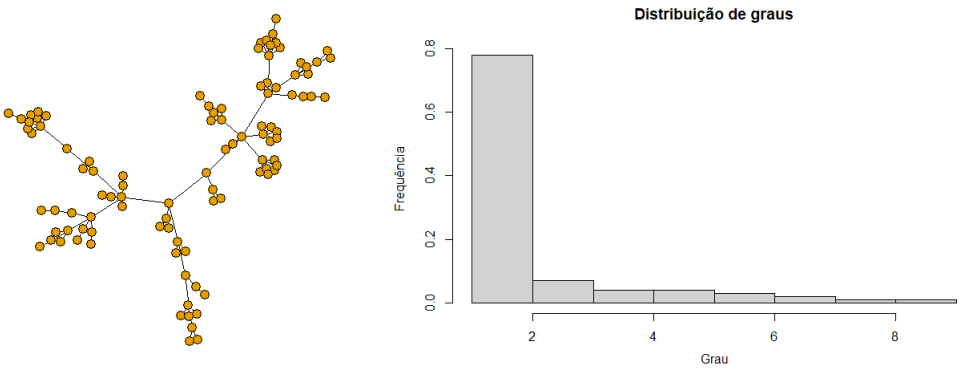


Figura 6.8: Grafo Livre de Escala e Distribuição de graus

A função plot() possui parâmetros que ajuda melhorar a visualização das redes, estes parâmetros serão apresentados a seguir:

Parâmetros	-	**Definição**
vertex.color	-	define a cor do vértice
vextex.frame.color	-	define a cor da borda do vértice
vertex.shape	-	define o formato do vértice
vertex.size	-	define o tamanho do vértice. O tamanho padrão é 15
vextex.label	-	define os nomes dos vértices
vertex.label.family	-	define a fonte dos textos dos vértices (ex:“Times”)
vertex.label.cex	-	define o tamanho da fonte
edge.color	-	define a cor das arestas
edge.width	-	define a espessura das arestas
edge.arrow.size	-	define o tamanho da seta
edge.arrow.width	-	define a espessura da seta
edge.lty	-	define o tipo da linha. Valores possíveis: 0 ou “blank”, 1 ou “solid”, 2 ou ”dotted“, 4 ou”dotdash“, 5 ou”longdash“, 6 ou”twodash“

6.3 Redes Livres de Escala

As redes livres de escala possuem como característica principal o que pode ser denominado de conexão preferencial, ou seja, cada novo vértice possui a tendencia de se concetar a um vértice da rede que tem um alto grau de conexões. As redes livres de escala são redes que possuem poucos vértices altamente conectados, e muitos vértices fracamente conectados. Os poucos vértices fortemente conectados são chamados de *Hubs*.

Barabasi-Albert preferential attachment model for scale-free graphs

```
g<-sample_fitness_pl(10000, 30000, 2.2, 2.3)
plot(degree.distribution(g),xlab="Grau")
plot(degree_distribution(g,cumulative=TRUE, mode="out"), xlab="Grau",log="xy")
```

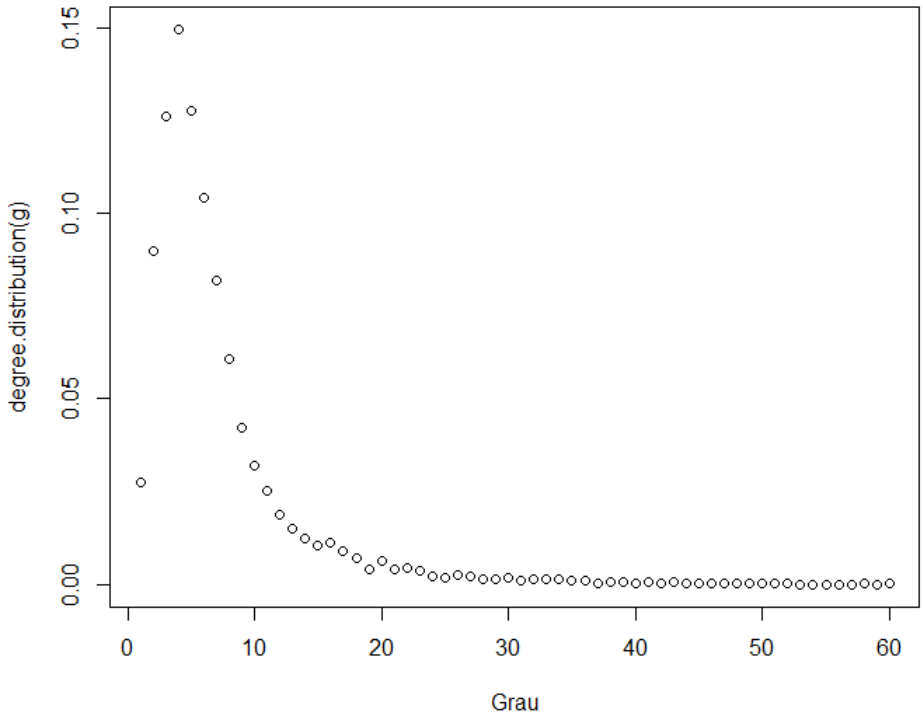


Figura 6.9: Grafo Livre de Escala gerado no RStudio

Adistribuição de graus de uma rede livre de escala obedece a uma lei de potência. Essa lei afirma que uma variação relativa em uma quantidade acaba em uma mudança relativa proporcional em outra. Uma distribuição de lei de potência tem a forma $Y = kX^{\alpha}$, onde: X e Y são variáveis de interesse, α é o expoente da lei,k é uma constante.

Se tomarmos o logaritmo da equação $Y = kX^{\alpha}$, teremos $log(Y) = log(k) + \alpha log(X)$ que é uma equação de reta de inclinação α.

O gráfico da Figura 6.10 é um gráfico log - log. A inclinação da reta é o valor de ...

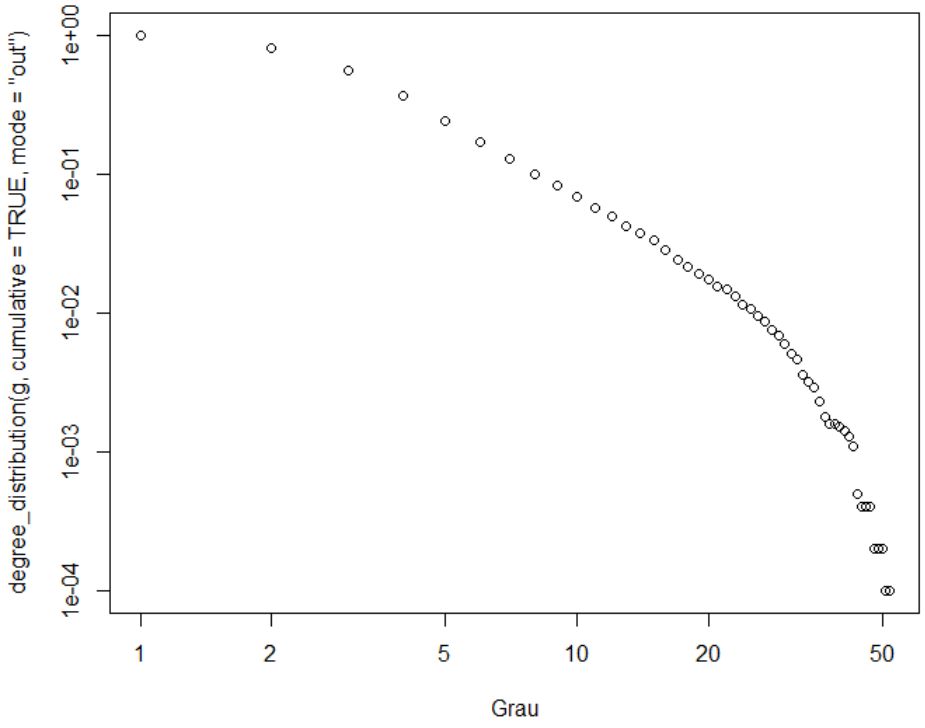


Figura 6.10: Grafo Livre de Escala gerado no RStudio

III APLICAÇÕES

7 Aplicações 105

7. Aplicações

IV

APÊNDICE

8 Apêndice 109

8.1 Distribuição de Frequências

8.2 Tipos de distribuições

8.3 Distribuição Normal

8.4 Distribuição Binomial

8.5 Distribuição de Poisson

8.6 Lei de potência

Bibliografia 125

Artigos

Livros

Manuais

Sites

Índice 129

8. Apêndice

8.1 Distribuição de Frequências

Uma pesquisa foi feita entre 40 alunos de uma escola, e uma das perguntas feitas foi: qual a sua altura? As repostas foram: 150,152,152,154,154,156,156,157,157,158,159,160,161,161,161,161,162,162,163,163, 164,164,165,166,166,166,168,168,169,171,172,172,173,174,174,174,175,175,176,180.

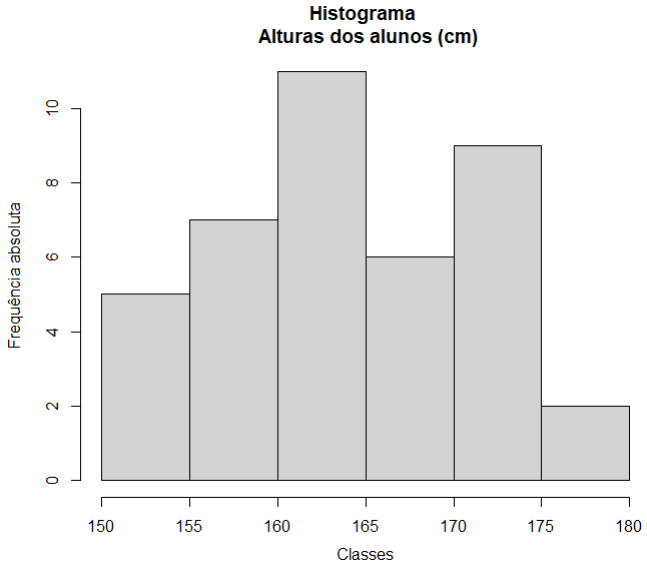
A tabela a seguir mostra uma distribuição de frequências das alturas em intervalos de classe. Nessa tabela estão registradas a frequência absoluta (f), a frequência relativa (f_r), a frequência acumulada relativa (f_r) .

Altura dos alunos cm	f	f_r	f_c
150 |-| 155	5	0,125	0,125
155 -| 160	7	0,175	0,3
160 -| 165	11	0,275	0,575
165 -| 170	6	0,15	0,725
170 -| 175	9	0,225	0,95
175 -| 180	2	0,05	1
Total	40	1	

Os dados dessa tabela podem ser representados no R através de histogramas e polígonos de frequência.

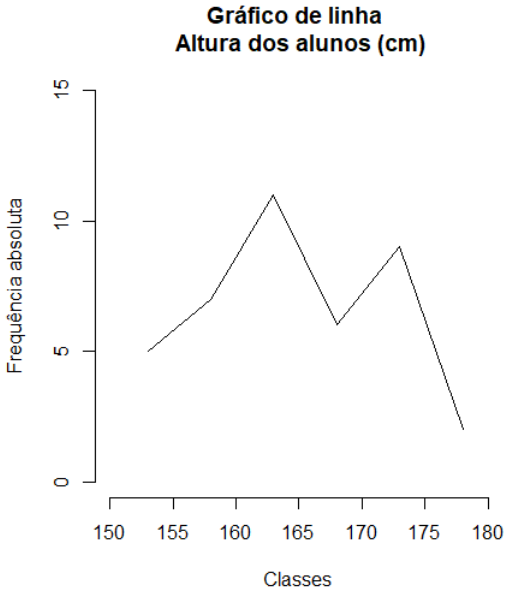
1. Histograma

```
x<-c(150,152,152,154,154,156,156,157,157,158,159,160,
     161,161,161,161,162,162,163,163,164,164,165,166,
     166,166,168,168,169,171,172,172,173,174,174,174,
     175,175,176,180)
hist(x,main="Histograma \n Alturas dos alunos (cm)",xlab="Classes",ylab="
    Frequ ncia absoluta",border="black",breaks=5)
```



2. Polígono de Frequência

```
x<-c(153,158,163,168,173,178)
y<-c(5,7,11,6,9,2)
plot(c(150,180),c(0,15),
     main="Gr fico de linha \n Alturas dos alunos (cm)",type='n',xlab="Classes
         ",ylab="Frequ ncia absoluta",)
lines(x,y)
```



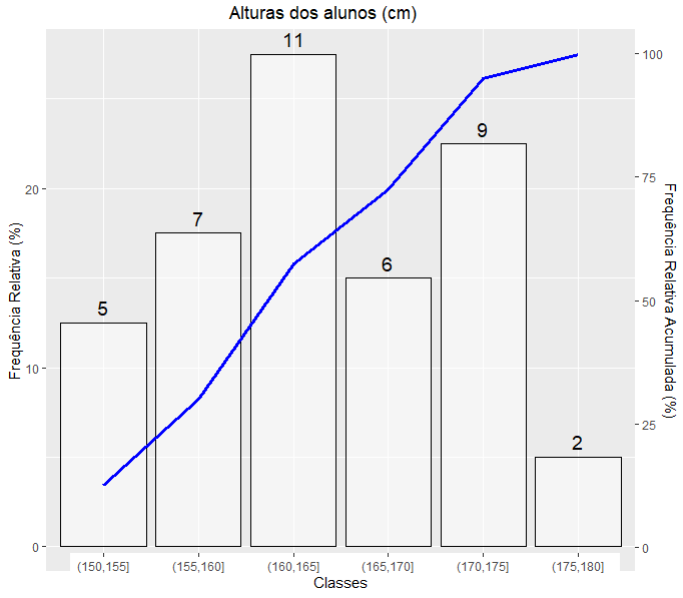
3. Histograma com Polígono de Frequência

```
x<-c(150,152,152,154,154,156,156,157,157,158,159,160,
     161,161,161,161,162,162,163,163,164,164,165,166,
     166,166,168,168,169,171,172,172,173,174,174,174,
```

```
      175,175,176,180)
dados<-cut(x,6) # Distribui a frequencia em seis classes
dados.plot<-data.frame(table(dados), table(dados)/sum(table(dados)),
        cumsum(prop.table(table(dados)))) # Calcula f, fr, fc
dados.plot<-dados.plot[, -3]
names(dados.plot)<-c("classes","fa","fr","fra")
dados.plot$fr<-dados.plot$fr*100

library(ggplot2)

ggplot(dados.plot) +
aes(x=classes,y=fr) +
geom_bar(stat="identity",fill="white",color="black",alpha=0.5) +
geom_line(aes(y=fra*max(fr), group=1),lwd=1.2,col="blue") +
labs(x="Classes", y="Frequ ncia Relativa (%)",title="Alturas dos alunos (cm)")
    +
geom_text(aes(label=fa),size=5,vjust=-0.5,color="black") +
scale_y_continuous(sec.axis=sec_axis(trans=~.*100/(max(dados.plot$fr)),
        name = "Frequ ncia Relativa Acumulada (%)"))
```



8.2 Tipos de distribuições

As distribuições podem ser simétricas, assimétricas à esquerda (assimetria negativa) ou assimétricas à direita (assimetria positiva).

Exemplos:

1. Distribuição Simétrica

x	f_i
1	2
3	4
5	7
7	4
9	2
Total	19

```
#Distribuicao simetrica
x<-c(1,1,3,3,3,3,5,5,5,5,5,5,5,5,7,7,7,7,9,9)
#media
media<-mean(x)
media
#mediana
mediana<-median(x)
mediana
#moda
mode<-function(x) {
   ux<-unique(x)
   ux[which.max(tabulate(match(x, ux)))]
}
moda<-mode(x)
moda
```

Nas distribuições simétricas Média = Mediana = Moda.

```
x<-c(1,1,3,3,3,3,5,5,5,5,5,5,5,5,7,7,7,7,9,9)
> media
[1] 5
> mediana
[1] 5
> moda
[1] 5
```

Rotina R para Distribuição simétrica:

```
x<-c(1,3,5,7,9)
y<-c(2,4,7,4,2)
```

```
barplot(y,names.arg=x,xlab="Classes",main="Grafico de Barras",space=F,ylab="
    Frequencia absoluta",col="white",border="black")
```

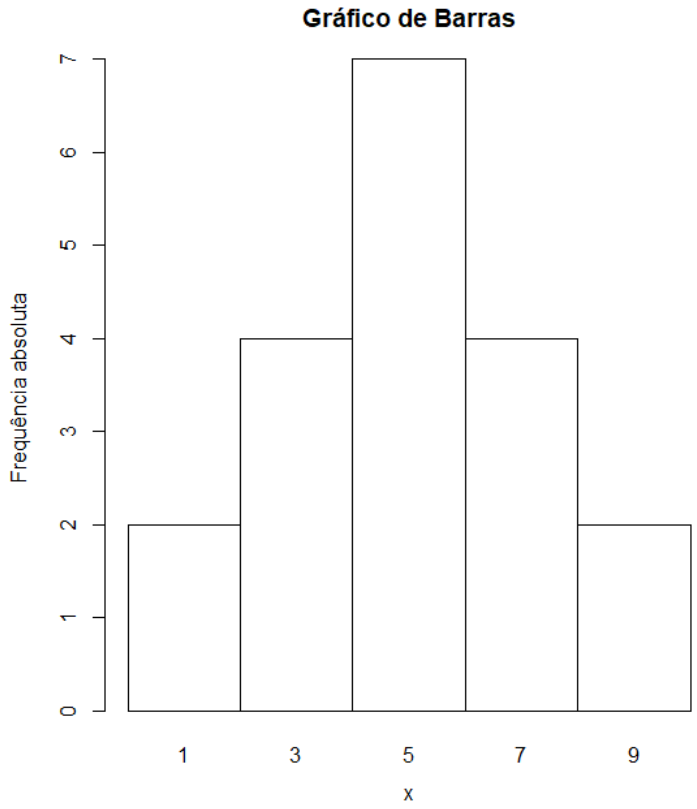


Figura 8.1: Exemplos de distribuicao simétrica no R

2.Distribuição assimétrica à direita ou positiva

x	f_i
1	2
3	7
5	4
7	3
9	2
Total	18

```
#Distribuicao assimetrica positiva
x<-c(1,1,3,3,3,3,3,3,3,3,5,5,5,5,7,7,7,9,9)
#m dia
media<-mean(x)
media
#mediana
```

```
mediana<-median(x)
mediana
#moda
mode<-function(x) {
  ux<-unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
moda<-mode(x)
moda
```

Nas distribuições assimétricas à direita: Moda < Mediana < Média

```
> x<-c(1,1,3,3,3,3,3,3,3,5,5,5,5,7,7,7,9,9)
> media
[1] 4.555556
> mediana
[1] 4
> moda
[1] 3
```

Rotina R para plotar o gráfico de barras.

```
x<-c(1,3,5,7,9)
y<-c(2,7,4,3,2)
barplot(y,names.arg=x,xlab="Classes",main="Grafico de Barras",space=F,
        ylab="Frequencia absoluta",col="white",border="black")
```

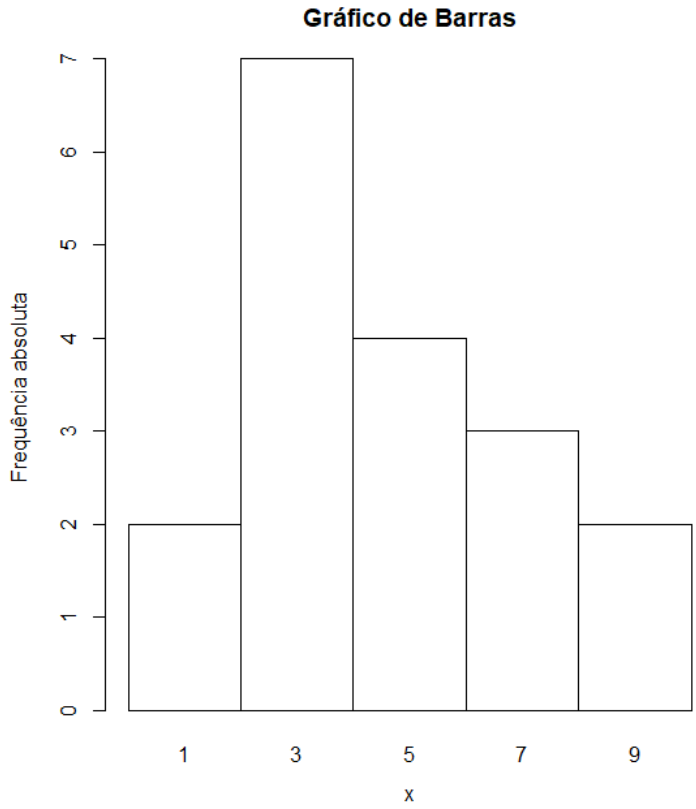


Figura 8.2: Exemplos de distribuição assimétrica positiva no R

3.Distribuição assimétrica à esquerda ou negativa

x	f_i
1	2
3	3
5	4
7	7
9	2
Total	18

```
#Distribuicao assimetrica negativa
x<-c(1,1,3,3,3,5,5,5,5,7,7,7,7,7,7,7,9,9)
#m dia
media<-mean(x)
media
#mediana
mediana<-median(x)
mediana
#moda
mode<-function(x) {
  ux<-unique(x)
  ux[which.max(tabulate(match(x, ux)))]
```

```
}
moda<-mode(x)
moda
```

Nas distribuições assimétricas à esquerda temos: Moda > Mediana > Média

```
> x<-c(1,1,3,3,3,5,5,5,5,7,7,7,7,7,7,7,9,9)
> media
[1] 5.444444
> mediana
[1] 6
> moda
[1] 7
```

Rotina para o gráfico de barras.

```
x<-c(1,3,5,7,9)
y<-c(2,3,4,7,2)
barplot(y,names.arg=x,xlab="Classes",main="Grafico de Barras",space=F,
        ylab="Frequencia absoluta",col="white",border="black")
```

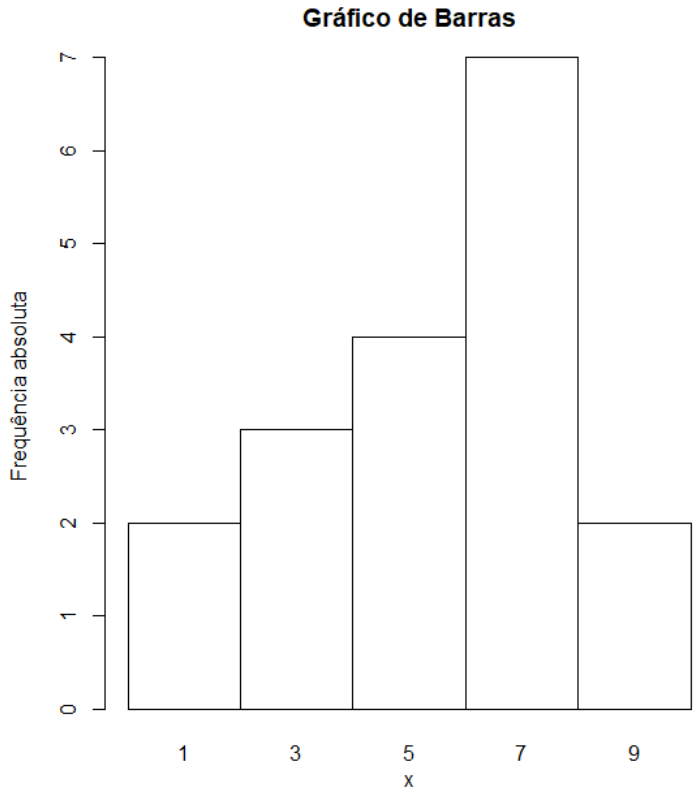


Figura 8.3: Exemplos de distribuição assimétrica negativa no R

8.3 Distribuição Normal

A distribuição normal ou Gaussiana é uma modelo estatístico contínua de probabilidade. É aplicado em inúmeros fenômenos e no desenvolvimento teórico. A densidade de probabilidade da distribuição normal é representado pela função:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\ \mathrm{e}^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Em que: $-\infty < x < +\infty$

No R podemos usar a função rnorm() para criar uma distribuição normal.

A rotina a seguir apresenta um exemplo do uso da função rnorm() mais a função curve() para representar a densidade de probabilidade.

```
1 x=rnorm(10000)
2 hist(x,prob=T,col="white",main="Distribuicao Normal",ylab="Densidade",ylim=c(0,0.5),
      xlim=c(-3,3))
3 curve(dnorm(x),col="red",add=T)
```

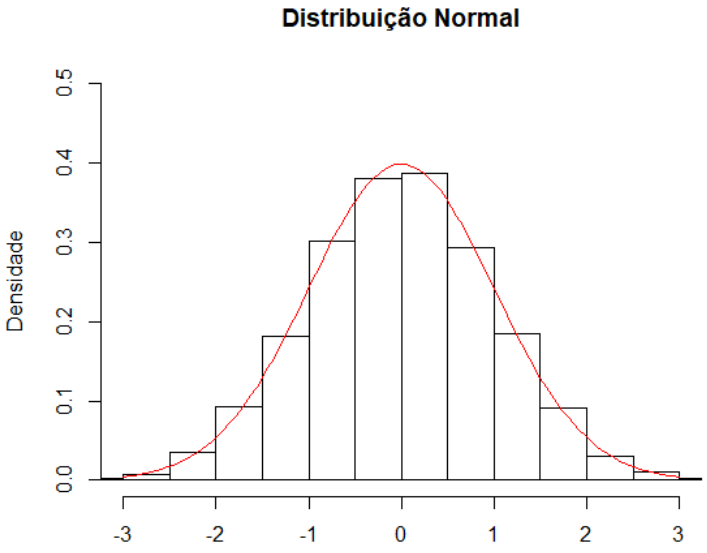


Figura 8.4: Exemplos de distribuição normal no R

8.4 Distribuição Binomial

A distribuição binomial é a distribuição de probabilidade discreta utilizado para identificar a probabilidade de ocorrência de determinado experimento aleatório a partir de sequência limitada de tentativas.

- As tentativas sao independentes;

- Cada tentativa do experimento admite apenas dois resultados;
- A probabilidade de sucesso (p) é constante.

A variável aleatória **X** indica o números de testes que resultaram em sucesso tem uma distribuição binomial com parâmetros p e n ≤ 1.

A função de probabilidade de **X** é:

$$P(X = x) = f(x;n,p) = \binom{n}{x} p^x (1-p)^{n-x}$$

para $x = 0,1,2,\ldots,n$ e onde $\binom{n}{x}$ é uma combinação.

Colocando a função completa: $f(x;n,p) = \frac{x!}{x!(n-x)!}\ p^x(1-p)^{n-x}$

Cada parte da função acima representa os seguintes dados:

- A combinação $\frac{n!}{k!(n-x)!}$ contém as ordenações possíveis;
- O número de sucesso é p^x;
- A probabilidade de fracassos é $(1-p)^{n-x}(1-p)^{n-x}$.

n - número de testes;

x - número de sucessos;

A esperança (média) e a variância são dadas por:

$$\mu = E(X) = np \text{ (esperança) e } \sigma^2 = V(x) = np(1 - p) \text{ (variância)}$$

No R podemos usar a função rbinom() para criar uma distribuição binomial. A rotina a seguir apresenta um exemplo do uso da função rbinom() mais a função points() para representar a curva da distribuicao binomial.

```
1 n=10;p=0.6
2 x=rbinom(1000,n,p)
3 hist(x,prob=T,ylab="Densidade",col="white",main="Distribuicao Binomial",ylim=c
      (0,0.25))
4 xvalores=0:n
5 points(xvalores,dbinom(xvalores,n,p),type="h",lwd=3)
6 points(xvalores,dbinom(xvalores,n,p),type="p",lwd=3)
```

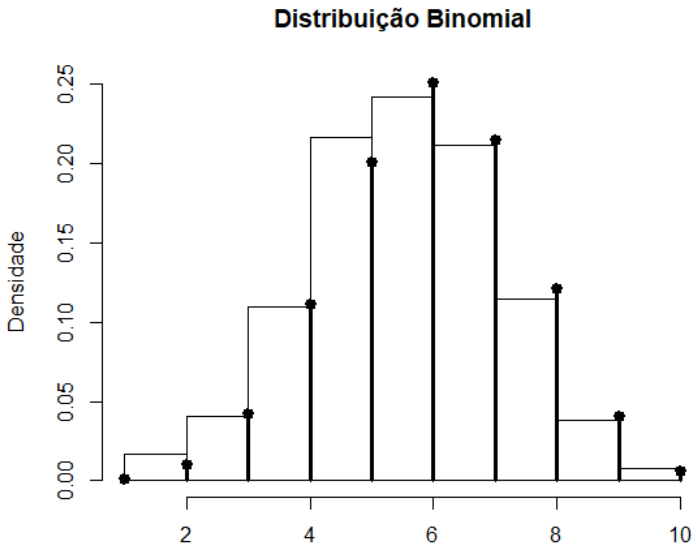


Figura 8.5: Exemplos de distribuição binomial no R

8.5 Distribuição de Poisson

A distribuição de Poisson é uma distribuição de probabilidade de variável aleatória discreta que permite indica a probabilidade de eventos em que o número de amostras pode aumentar no tempo e a probabilidade de sucesso diminuir, mantendo constante a esperança.

A distribuição Poisson está relacionada a eventos raros. Os eventos podem ser: número de acidentes automotivos, número de peças defeituosas durante a semana na fábrica A e entre outros.

Esta distribuição demostra a probabilidade de que um evento de ocorrer no intervalo de tempo, quando a taxa de ocorrência é fixa.

$$f(x;\lambda) = \frac{e^{-\lambda}\lambda^{x}}{x!},$$

x = variável aleatória de ocorrências do evento em um intervalo

λ = taxa de ocorrência do evento x (no esperado de eventos) onde: $\mu = \lambda \cdot t$

O parâmetro μ indica a taxa de ocorrência (λ) por unidade de medida (t), ou seja, λ = taxa de ocorrência e t = intervalo de tempo.

No R para criar gráficos de distribuição de Poisson usamos a função dpois() e temos que informar o parâmetro λ.

```
1 x<-2
2 lambda<-2.3
3 #Par metros x e lambda:
4 dpois(x,lambda)
```

```
x <- 0:20
p<-dpois(x, lambda)
plot(x,p,xlab="N mero de erros por mil metro",
     ylab="Probabilidade",main="Distribui  o de Poisson")
lines(x,p)
```

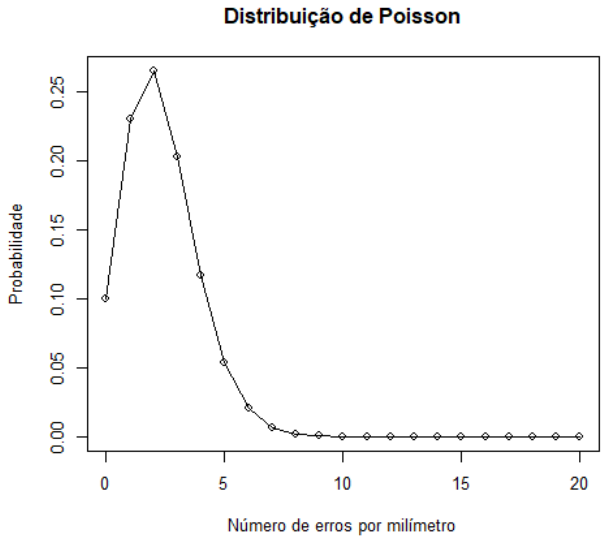


Figura 8.6: Exemplos de distribuição binomial no R

Alguns exemplos de gráficos da distribuição de Poisson com diferentes valores do parâmetro lambda.

```
par(mfrow=c(2,2))
plot(0:25,dpois(x = 0:25,lambda=1),type="h",xlab="X",ylab="P[X = x]",main=expression
    (mu==1),ylim=c(0,.4))
plot(0:25,dpois(x = 0:25,lambda=10),type="h",xlab="X",ylab="P[X = x]", main =
    expression(mu==10),ylim=c(0,.4))
plot(0:25,dpois(x = 0:25,lambda=15),type="h",xlab="X", ylab = "P[X = x]",main=
    expression(mu==15),ylim=c(0,.4))
plot(0:25,dpois(x=0:25,lambda=20),type="h",xlab="X",ylab="P[X = x]",main=expression(
    mu==20),ylim=c(0,.4))
```

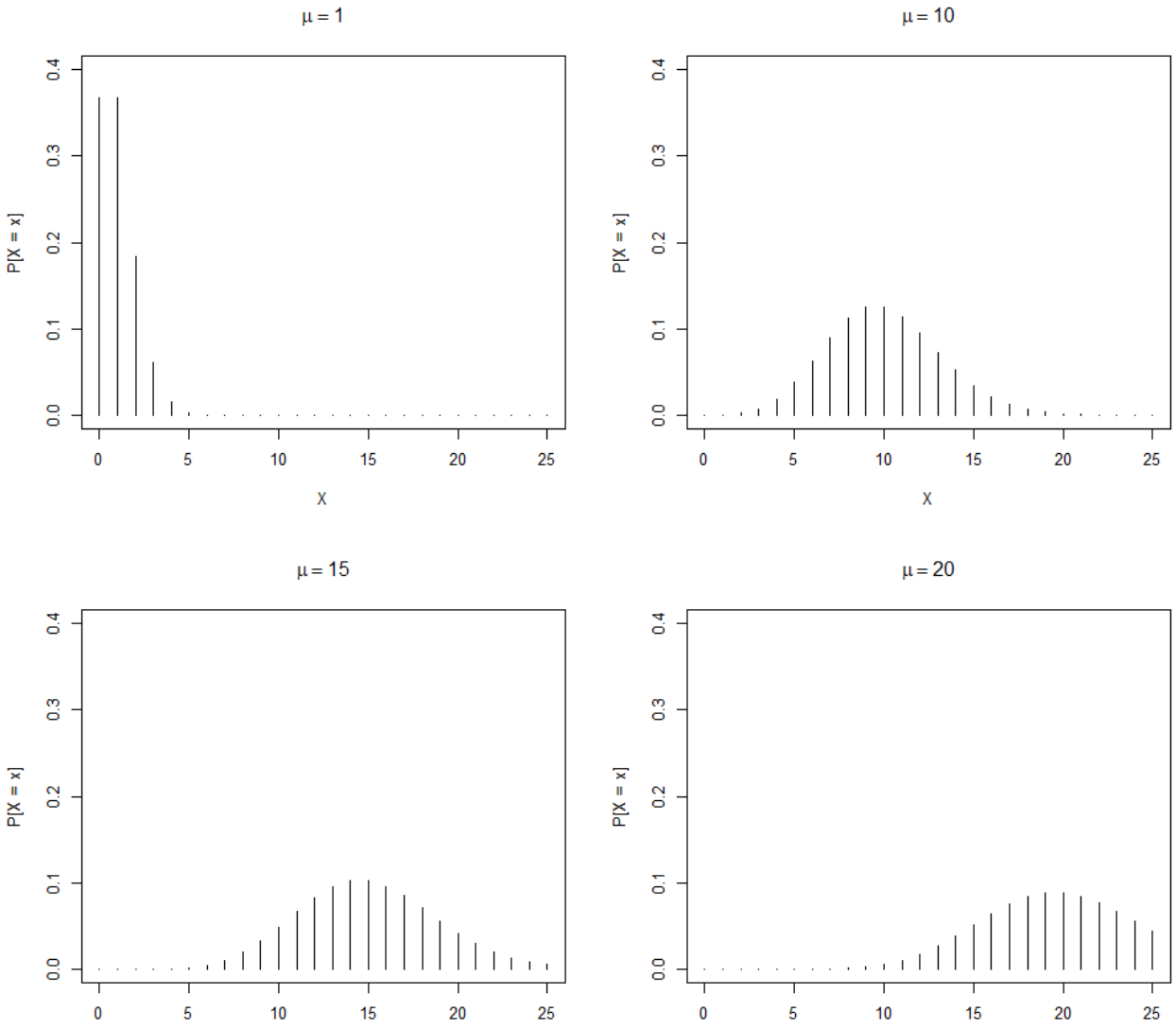


Figura 8.7: Exemplos de distribuição binomial no R

Como função de x, esta é a função de probabilidade. A distribuição de Poisson pode ser derivada como um caso limite da distribuição binomial.

8.6 Lei de potência

A lei de potência afirma que uma variação relativa em uma quantidade acaba em uma mudança relativa proporcional em outra.Uma distribuição de lei de potência tem a forma $Y = kX^{\alpha}$, onde: X e Y são variáveis de interesse, α é o expoente da lei,k é uma constante. Se tomarmos o logaritmo da equação $Y = kX^{\alpha}$, teremos $log(Y) = log(k) + \alpha log(X)$ que é uma equação de reta de inclinação α.

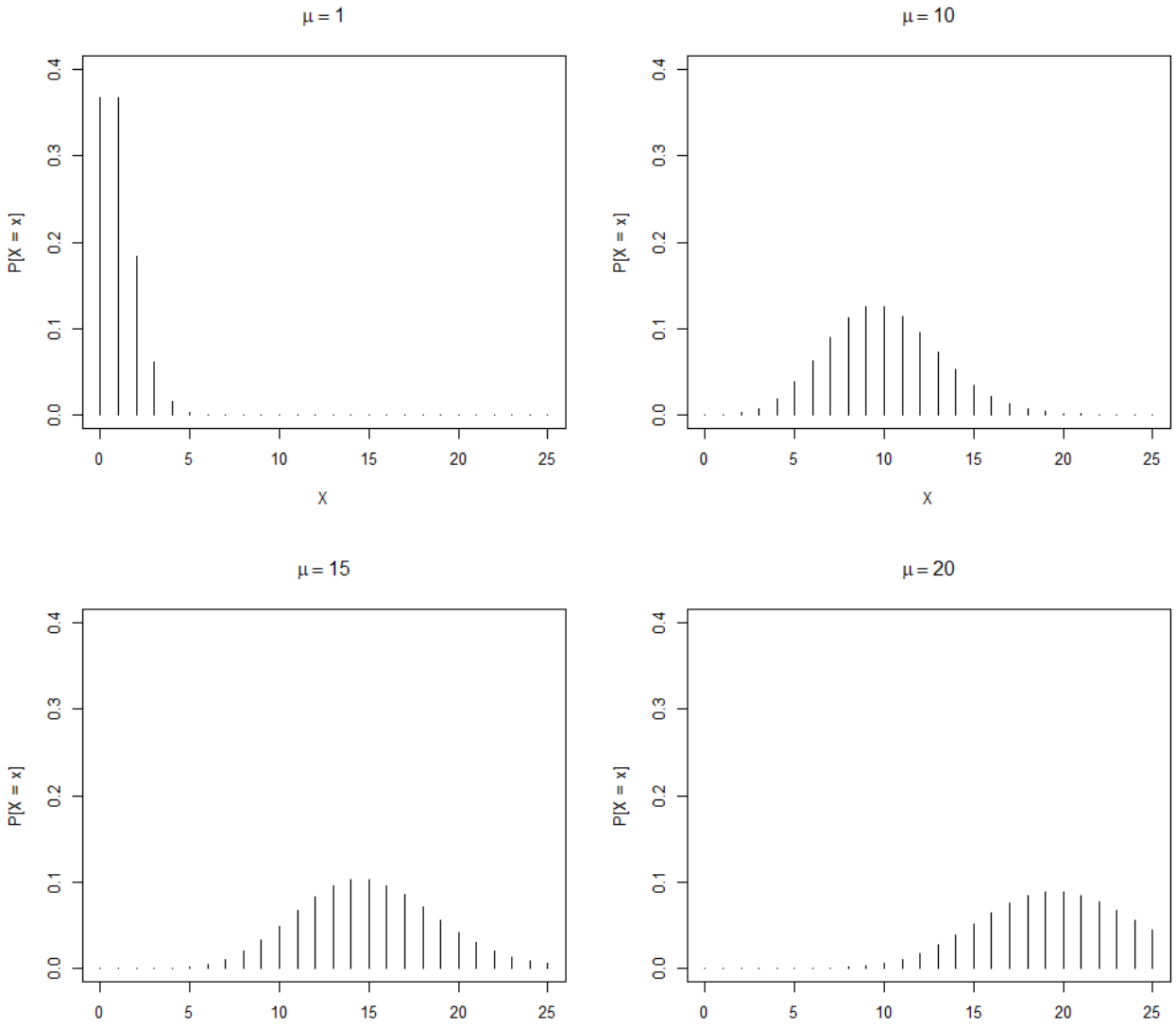


Figura 8.8: Exemplos de distribuição binomial no R

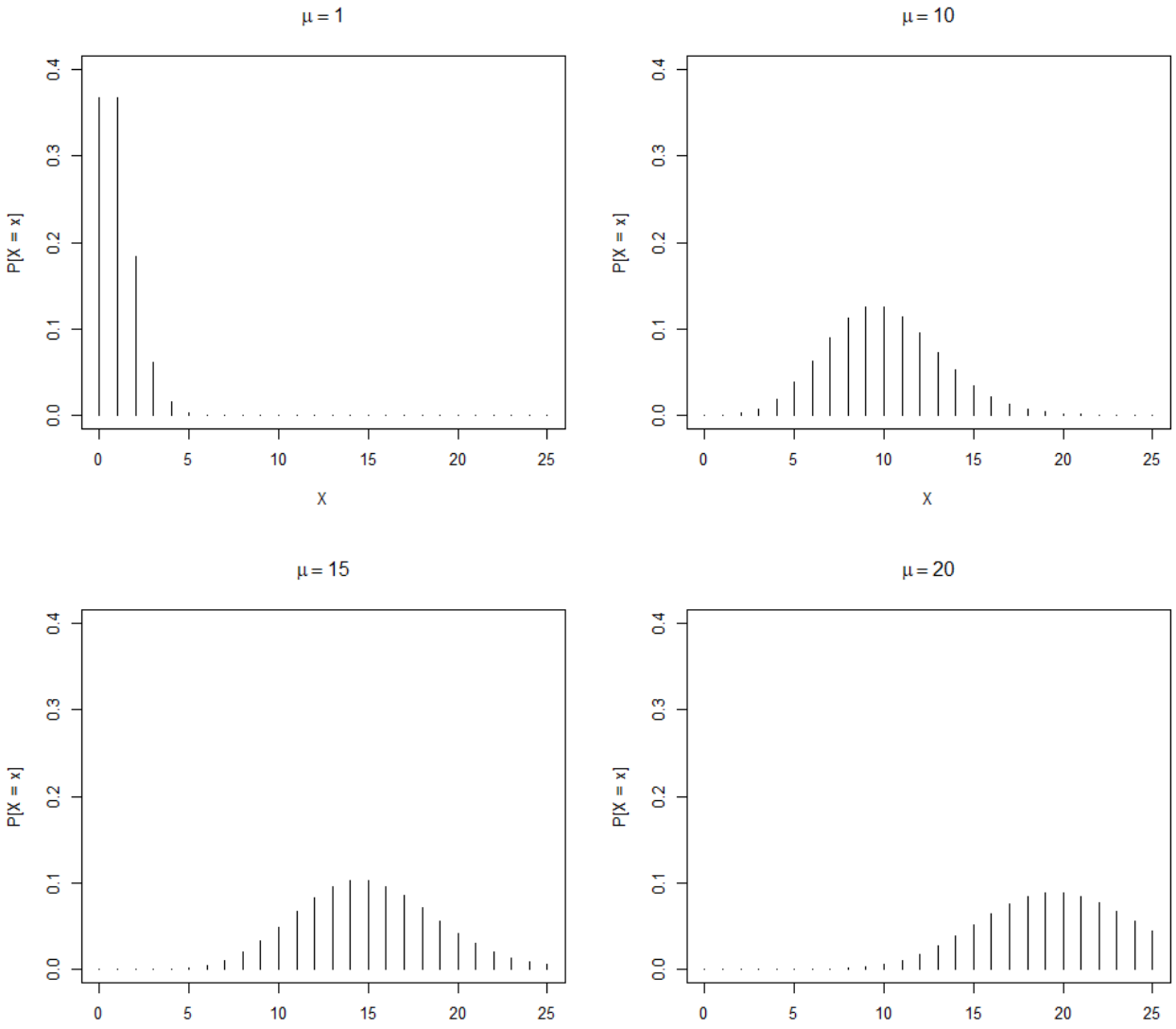


Figura 8.9: Exemplos de distribuição binomial no R

Bibliografia

Artigos

[1] Gerald Alexanderson. «About the cover: Euler and Konigsbergs Bridges: A historical view». Em: *Bulletin of the american mathematical society* 43.4 (2006), páginas 567–573 (ver página 53).

[3] Charles T Carlstrom e Timothy S Fuerst. «Agency Costs, Net Worth, and Business Fluctuations: A Computable General Equilibrium Analysis». Em: *The American Economic Review* (1997), páginas 893–910. ISSN: 0002-8282.

[4] Aaron Clauset, Mark EJ Newman e Cristopher Moore. «Finding community structure in very large networks». Em: *Physical review E* 70.6 (2004), página 066111 (ver páginas 82, 83).

[5] Gabor Csardi e Tamas Nepusz. «The igraph software package for complex network research». Em: *InterJournal* Complex Systems (2006), página 1695. URL: `https://igraph.org` (ver página 18).

[6] Ron Eglash. «Fractals, Complexity, and Connectivity in Africa». Em: (jan. de 2005).

[7] Ron Eglash e Toluwalogo B Odumosu. «Fractals, complexity, and connectivity in Africa». Em: *What mathematics from Africa* (2005), páginas 101–109.

[9] Victor Lemes Landeiro. «Introdução ao uso do programa R». Em: *Manaus: Instituto Nacional de Pesquisas da Amazônia* (2011).

[10] Qiang Li, Liwen Chen e Yong Zeng. «The Mechanism and Effectiveness of Credit Scoring of P2P Lending Platform: Evidence from Renrendai.com». Em: *China Finance Review International* 8.3 (2018), páginas 256–274.

[16] Vincenzo Quadrini. «Financial Frictions in Macroeconomic Fluctuations». Em: *FRB Richmond Economic Quarterly* 97.3 (2011), páginas 209–254.

[18] Francisco Aparecido Rodrigues e Luciano da Fontoura Costa. «Caracterização, classificação e análise de redes complexas». Em: (2007) (ver páginas 53, 61, 83).

[19] James Smith. «Título do artigo e pá». Em: 14.6 (mar. de 2013), páginas 1–8.

[21] Diego Loureiro Francisco de Souza. «Geometria: fractal e euclidiana nas construções históricas africanas». Em: *Matemática Licenciatura-Unisul Virtual* (2019).

[22] Duncan J Watts e Steven H Strogatz. «Collective dynamics of 'small-world'networks». Em: *nature* 393.6684 (1998), páginas 440–442 (ver página 97).

[24] Wayne W Zachary. «An information flow model for conflict and fission in small groups». Em: *Journal of anthropological research* 33.4 (1977), páginas 452–473 (ver página 65).

Livros

[2] Albert-László Barabási et al. *Network science*. Cambridge university press, 2016 (ver páginas 53, 84, 93, 95).

[20] Nome Sobrenome. *Título do Livro*. 1ª edição. Volume 3. 2. Cidade: Editora, jan. de 2020, páginas 123–200.

Manuais

[8] Autores ou time que escreveu. *Nome do manual*. Empresa dona do manual. Cidade, país, 2018.

[17] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2020. URL: `https://www.R-project.org/` (ver página 11).

Sites

[11] Qiang Li, Liwen Chen e Yong Zeng. *Crescimento Exponencial e Pandemias*. 3Blue1Brown, 2020. URL: `https://youtu.be/Kas0tIxDvrg`.

[12] NASA. *Pluto: 'The Other' Red Planet*. `https://www.nasa.gov/nh/pluto-the-other-red-planet`. Acessado em: 2018-12-06. 2015.

[13] OEIS. *Compilado de Símbolos Matemáticos para Latex*. `https://oeis.org/wiki/List_of_LaTeX_mathematical_symbols`. Acessado em: 18/08/2020.

[15] Site Origem da Palavra. *Origem da palavra Rede*. Origem da Palavra, 2020. URL: `https://origemdapalavra.com.br/palavras/rede/` (ver página 51).

[23] Wikipedia. *Compilado de Equações Matemáticas para Latex*. https://pt.wikipedia.org/wiki/Ajuda:Guia_de_edi%C3%A7%C3%A3o/F%C3%B3rmulas_TeX. Acessado em: 18/08/2020.

Índice

S

seção 11, 12, 21, 23–25, 27–29, 31–38, 43, 44, 46