

# **Moléculas voadoras, animações, gráficos e mapas interativos...ou... Vivificando conteúdos para o Ensino Médio**

José Maurício Schneedorf Ferreira da Silva

# **Índice**

# **Como diriam no telemarketing ... !**

1. Você está cansado de buscar fontes diferente de livros e internete pra manter o interesse da Turma ?
2. Você acha que as imagens dos livros que ensina poderiam ser “*vivificadas*” pra mostrar moléculas, gráficos e mapas de modo mais interessante ?
3. Você acha que moléculas observadas em 3D poderiam tornar as aulas mais “*amigáveis*” ?
4. Você acredita que suas aulas teriam mais combustível se você pudesse também interagir com as moléculas, possibilitando alterações visuais e animações ?
5. Você gostaria de aprender como fazer isso usando somente um programa *online* ?
6. Você não se sente “*desprotegido*” quando tenta ensinar uma relação gráfica meio abstrata de uma figura de livro pra Turma ?
7. Seus alunos não acham meio chato quando precisam estudar alguma informação de cada região de um mapa num livro ?
8. Você acharia interessante apresentar aos alunos(as) gráficos dos livros, embora animados, com interatividade e mesmo simulações ?
9. Você acredita que um mapa informativo, mas interativo, do Brasil, do mundo ou qualquer região do planeta, poderia contribuir pra atenção de seus “*pimpolhos(as)*” ?
10. Você acha interessante se puder mostrar essas coisas numa simples página de *browser* (navegador), que pode ser lida num PC ou dispositivo móvel, mas sem precisar de internete ?
11. Você gostaria de aprender um única ferramenta que possibilitasse tudo isso e mais, como analisar dados dos mais divesos campos de estudo, construir belíssimos gráficos, simular situações de sua área, elaborar textos com qualidade de publicação técnico-científica, construir páginas de internete com imagens, vídeos, hiperlinks, criar livros inteiros, blogs ou mesmo um *website* completo ?
12. E isso tudo por um único programa instalado em seu computador ou melhor, pela internete, e sem pagar nada ??!

...Então...seus problemas acabaram !!!



# 1 Algumas questões prévias

## 1.1 De que se trata esse material ?

Bom, como diz o título, de *vivificar* imagens e relações numéricas encontrados em livros-texto. Em palavras mais abstratas...da tentativa de se oferecer algumas ferramentas para compor objetos didáticos para o *Ensino Médio*, principalmente para temas de *Física, Química e Biologia* contidos em *Ciências da Natureza e Suas Tecnologias*, também com *aplicações da Matemática* e baseadas em premissas de *Ensino Reprodutível*.

Agora, de modo mais claro, de instrumentalizar professores ao uso de dois programas básicos de computador, um para *visualização de moléculas em 3D*, e outro para *vivificar* algumas relações estáticas encontradas em livros-texto por *animações, interatividade, e simulações*.

## 1.2 Qual a vantagem em se familiarizar com esses programas ?

Também em módicos termos, para oferecer aos alunos uma visão mais *paupável e amigável* de conteúdos trazidos de livros-texto do nível educacional da proposta.

## 1.3 Como alunos do ensino médio podem beneficiar-se da oferta das ferramentas que serão tratadas aqui ?

Isto tem muito a ver com o conceito de *letramento visual (visual literacy)* e também com *letramento científico*. Em síntese, os livros-texto oferecem formas distintas de apresentação de conteúdos, e que exigem do estudante diferentes **níveis de abstração**, como o *simbólico* (fórmula estrutural, equação), *esquemático* (fluxogramas, geovisualização), *gráfico* (relação entre variáveis), *realístico* (resultado experimental), e de *cartoon* (estrutura molecular).

Assim, pretende-se oferecer uma ferramenta para auxiliar na abstração de *cartoon*, e outra para o *simbólico, esquemático, gráfico*, e de análise de dados presente na abstração *realística*. Essa segunda ferramenta permitirá acessar o conteúdo de livros-texto por animações e simulações.

*Na prática imediata e de curto prazo, essas ferramentas poderão possibilitar um menor nível de abstração para alguns conteúdos encontrados em livros-texto do Ensino Médio, por exemplo:*

- permitindo a visualização de uma molécula em 3D, sua rotação, tamanho, estudo estrutural e químico, funções orgânicas, relações de estrutura e função, e animações.
- permitindo a elaboração de gráficos animados a partir de equações, visualização interativa de gráficos, e simulações.

## **1.4 E quais são essas ferramentas ?**

Para a *visualização de moléculas em 3D* será empregado o **Jmol**, e para os demais mencionados o programa **R** com sua interface gráfica **RStudio**. São programas gratuitos para instalação em computador de mesa ou notebook, embora também acessáveis por navegador de internete, e nesse caso, sem necessidade de instalação.

## **1.5 Se este material envolve o uso de livros-texto, qual bibliografia será utilizada ?**

Nenhuma. Na verdade será utilizado algum conteúdo geral presente no [Material de Apoio Pedagógico para Aprendizagens - MAPA](#) para habilidades previstas no *Currículo Referência de Minas Gerais*, e pertinente principalmente à *Ciências da Natureza e Suas Tecnologias*.

## **1.6 Além dos conteúdos específicos com algum potencial para melhorar o aprendizado do estudante, há outra vantagem no aprendizado dos programas ?**

Diversas. De início, a familiarização com conceitos de programação computacional. O *Jmol* é mais fechado, já que trabalha somente com a renderização de modelos moleculares. O que já é muito bom para se visualizar a imensidate de moléculas presentes em livros-texto. Já o *R* é outra história.

O *R* permite uma infinidade de ações, quer voltadas à pesquisa ou ao ensino. É utilizado por diversas Universidades ao redor do mundo, bem por diversas Universidades e empresas (ex: Google, Facebook, LinkedIn, Twitter, Bank of America, Lenovo, Bing). Embora tenha sido originalmente desenvolvido para análise de dados, possui hoje mais de 20 mil pacotes adicionais.

Essa expansibilidade permite, por exemplo, recursos de matemática, estatística, gráficos e tabelas avançados, animações e simulações, análise de imagem e de texto, criação musical e artística, ciência de dados, inteligência artificial, comunicação com placas microcontroladores (ex: *Arduino*), internete das coisas (*IoT*), criação de *blogs*, de *websites*, e de livros (tal como usado para produzir este material).

## **1.7 E o tal *Ensino Reprodutível*, de que se trata ?**

*Ensino Reprodutível (ER)* uma metodologia ativa em ensino-aprendizagem ainda em formação, e que mistura trechos de algoritmos com conteúdos temáticos, ambos editáveis num simples bloco de notas. Na prática, o *ER* envolve aplicar códigos escritos em um programa específico para a produção de um objeto didático, incluindo aí um modelo molecular, um gráfico, uma tabela, uma mídia, um texto formatado com saída em DOCX, PDF ou EPUB, uma animação, e/ou uma simulação, dentre vários.

Dessa forma é possível imaginar que um aprendiz agregue valor a seu aprendizado ao 1) repetir a execução do código num programa específico, 2) alterar algum trecho do código para observar um resultado diferente, ou mesmo 3) criar um novo código buscando outro produto final. Nesse caso, a *reprodutibilidade* está centrada na criação/modificação do objeto didático, e que pode ser *realizada por qualquer pessoa que tenha o programa e o código*.

## 2 Um pouco sobre as ferramentas

### 2.1 Visualização de moléculas em 3D e animações

São vários os programas disponíveis para observação e estudo tridimensional de modelos atômicos, e que cujo acesso se realiza por computador, dispositivos móveis, e mesmo pela internete. Uns são gratuitos, outros de demonstração gratuita, outros são pagos. Numa lista pequena, pode-se mencionar:

- [Pymol](#)
- [Maestro](#)
- [iMolView](#)
- [VMD](#)
- [UCSF Chimera](#)
- [Vesta](#)
- [QuteMol](#)

Alguns programas também permitem a construção de moléculas e sua representação em 3D, como por exemplo:

- [MolView](#)
- [ChemSketch](#)
- [DIY-molecules](#)

Por sua vez, o [Jmol](#) é um programa multiplataforma (Windows, Mac OS X, Linux, Unix) elaborado em ambiente *Java*, e pode ser executado tanto em versão fechada (*standalone*), como integrado a outras aplicações em *Java*, ou ainda junto a buscadores de internet por auxílio de um *applet*. Diferente de programas comuns, ela não requer instalação, apenas execução de arquivo *Java*. Dessa forma, o *Jmol* pode rodar tanto a partir de uma pasta de diretório contendo seus arquivos, como a partir de um disco rígido ou mídia removível (*pendrive*). Além disso é um programa de código-fonte e distribuição livres para representação de modelos moleculares tridimensionais, permite uma diversidade de visualizações e cores, movimentos de translação e rotação das moléculas, ampliação visual, cálculos de distância, ângulos, estruturas e superfícies, otimizações moleculares, e animações, dentre outros. Existe um número muito grande de portais na internet que utilizam o *Jmol*, a começar pela [lista](#) disponível na [página da comunidade Wiki](#).

Complementarmente, o acesso ao *Jmol* pode ser realizado pela internete, sem a necessidade de arquivos no computador. Dentre os diversos *sites* que possuem o *applet JSmol* que permite esse acesso, sugerimos clicar na imagem abaixo, que direcionará ao *applet* do *website* da [St. Olaf College](#), ilustrando uma *molécula de água*:



## 2.2 Gráficos animados, simulações, e geovisualização

[R](#) é um software gratuito e de código aberto, originalmente desenvolvido para computação estatística e gráficos. Roda-se o *R* em diversas interfaces de usuário (*GUI, Graphical User Interface*), dentre as quais destaca-se o [RStudio](#), um ambiente de desenvolvimento integrado (IDE) também gratuito. O *Rstudio* possui também uma *versão online* gratuita, acessível pelo site [RStudio Cloud](#).



Pode-se trabalhar com o *Rstudio* para coisas simples, como texto, gráficos e tabelas, mas também para um largo espectro de atividades, em função de sua expansibilidade por *pacotes instaláveis*. Esses pacotes são obtidos no [site oficial do projeto - R CRAN](#), bem como desenvolvidos e disponibilizados por iniciativas individuais em sites autorais.

Para se ter uma ideia desse espectro, alguns pacotes (*library*) permitem análise e criação de textos, interface com portadores de deficiência visual, animações, gráficos e tabelas de qualidade de publicação técnico-científica, manipulação e análise de dados e de imagens, música, arte, Ciência de Dados, internet das coisas (*IoT*), linguagem profunda de máquina, e mesmo inteligência artificial.

Para a pesquisa científica o *R* possui diversos *pacotes* direcionados a temas específicos e bem diversos, como Ecologia, Estatística, Matemática, Física, Biologia, Química, Artes, Geografia, Geologia, e História, dentre muitos.

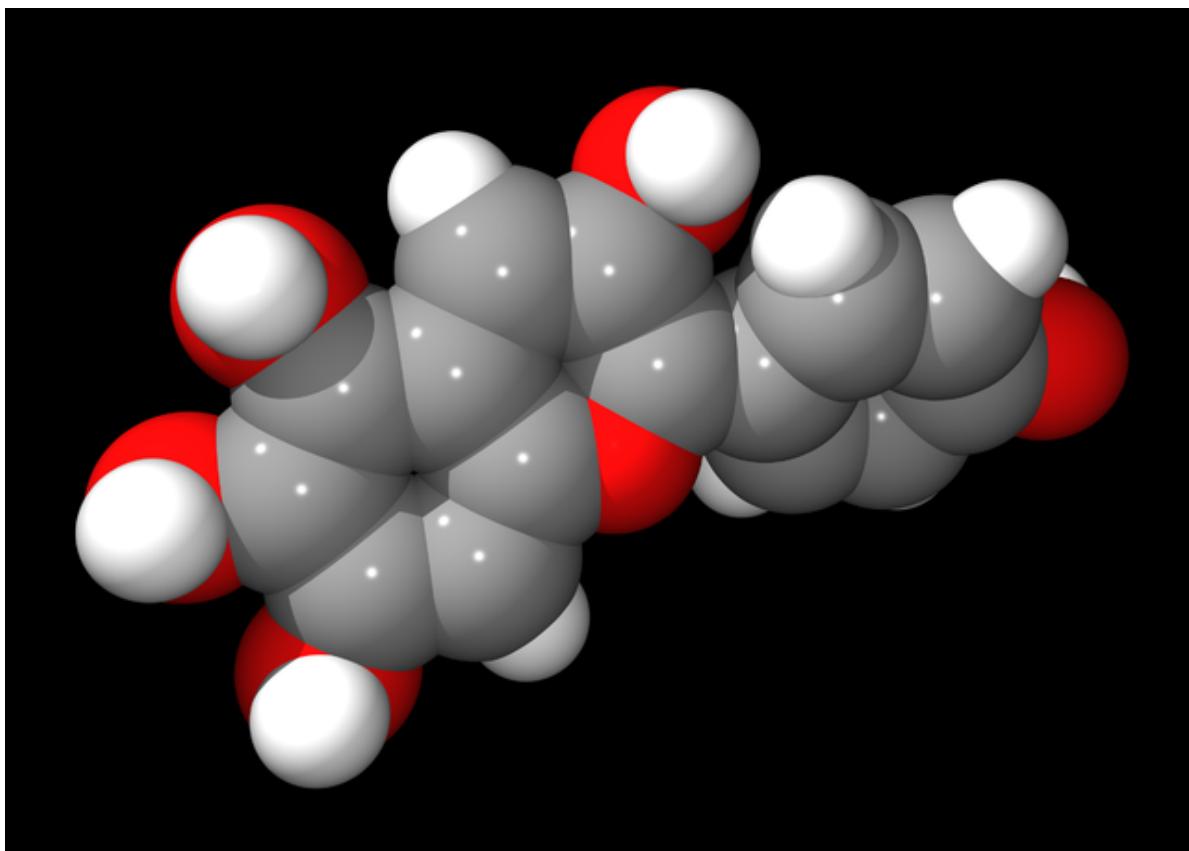


Entre os [diversos pacotes oficiais](#) (na verdade, mais de 20 mil!), existem alguns que permitem a elaboração de objetos didáticos nos moldes do subtítulo retro. Esses pacotes são os listados abaixo, e serão ilustrados com conteúdos do ensino médio constantes do [Material de Apoio Pedagógico para Aprendizagens de Minas Gerais - Mapa](#), bem como livros didáticos para Matemática, Física, Química e Biologia do nível educacional.

- `basic` - criação de gráficos básicos
- `ggplot2` - gráficos mais elaborados e muito bonitos
- `animation.plots` - animação de gráficos
- `manipulate` - simulação gráfica de equações
- `ganimate` - animação de gráficos do pacote `ggplot2`
- `plotly`- gráficos interativos
- `leaflet` - geovisualização em mapas

# **Parte I**

## **PARTE 1 - Jmol**



# 3 Usando o Jmol para observar moléculas em 3D

## 3.1 Onde começar ?

Pode-se começar a usar o *Jmol* de vários modos. Se for usar em seu computador ou notebook, ou mesmo a partir de uma mídia removível (*pendrive*), pode acessá-lo baixando, descomprimindo e executando o arquivo *Jmol.jar* presente na pasta principal no [site do Jmol](#).

Agora, se não quiser instalar nada, pode também acessá-lo *online* a partir de diversos sítios. Nesse curso vamos utilizar um bem famoso, adaptado de um dos próprios desenvolvedores do programa. Basta clicar nesse [link](#), numa nova aba, por exemplo:

```
https://chemapps.stolaf.edu/jmol/jmol.php?model=water
```

Agora, clique na molécula com o botão esquerdo do *mouse* ou com o *touchpad*, e faça movimentos. Ou então gire o botão do meio do mouse ou realize gestos de afastamento e proximidade com dois dedos no *touchpad*. A Figura ?? que segue ilustra o resultado.

Essa é a essência principal ao referenciarmos no título deste curso a ideia de **moléculas voadoras**.

## 3.2 Como carregar uma molécula *online* ?

Pra *brincar* um pouco com outra molécula, experimente mudar o modelo na própria página de internete, ao final da linha. Por exemplo, de *water* para *tylenol*:

```
https://chemapps.stolaf.edu/jmol/jmol.php?model=tylenol
```

Você pode tentar fazer isso com outras moléculas, digitando seu nome *em inglês*, por tratar-se de um *site* estrangeiro. Mas é claro que o banco de dados dessa busca não é ilimitado, e por vezes o sistema não encontrará a molécula desejada.

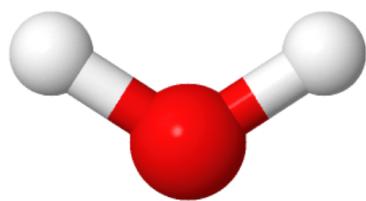


Figura 3.1: Tela (clicável) referente ao modelo da molécula de água renderizado em navegador pelo *site* do St. Olaff College.

Mas há alternativas. Uma delas é buscar o nome da molécula em um *site* utilizado como banco de dados, o [PubChem](#). Exemplificando para a *vitamina C (ácido ascórbico)*:

1. Entra no site do [PubChem] (<https://pubchem.ncbi.nlm.nih.gov/>) ;
2. Procura por "ascorbic acid" ;
3. Se existir, digite esse mesmo termo ao final da linha do \*JSmol online\*, ou seja:  
<https://chemapps.stolaf.edu/jmol/jmol.php?model=ascorbic acid>

### 3.3 Como baixar uma molécula em seu computador

Caso você digite o nome da molécula no [PubChem](#) mas não a encontre após digitar seu nome no *link* do *JSmol* acima, é possível baixá-la no computador/notebook/tablet/smartphone a partir do *site* em formato lido pelo *Jmol*. Pra isso, siga os passos seguintes, exemplificados para o *tetracloreto de carbono*:

1. Entre no site do Pubchem (<https://pubchem.ncbi.nlm.nih.gov>);
2. No campo de busca digite "Carbon Tetrachloride";
3. Clique no 1º. composto;
4. Clique em "3D";
5. Clique em "Download coordinates";
6. Salve o arquivo clicando na opção "Save" para SDF;
7. Forneça um nome mais "amigável" pro arquivo

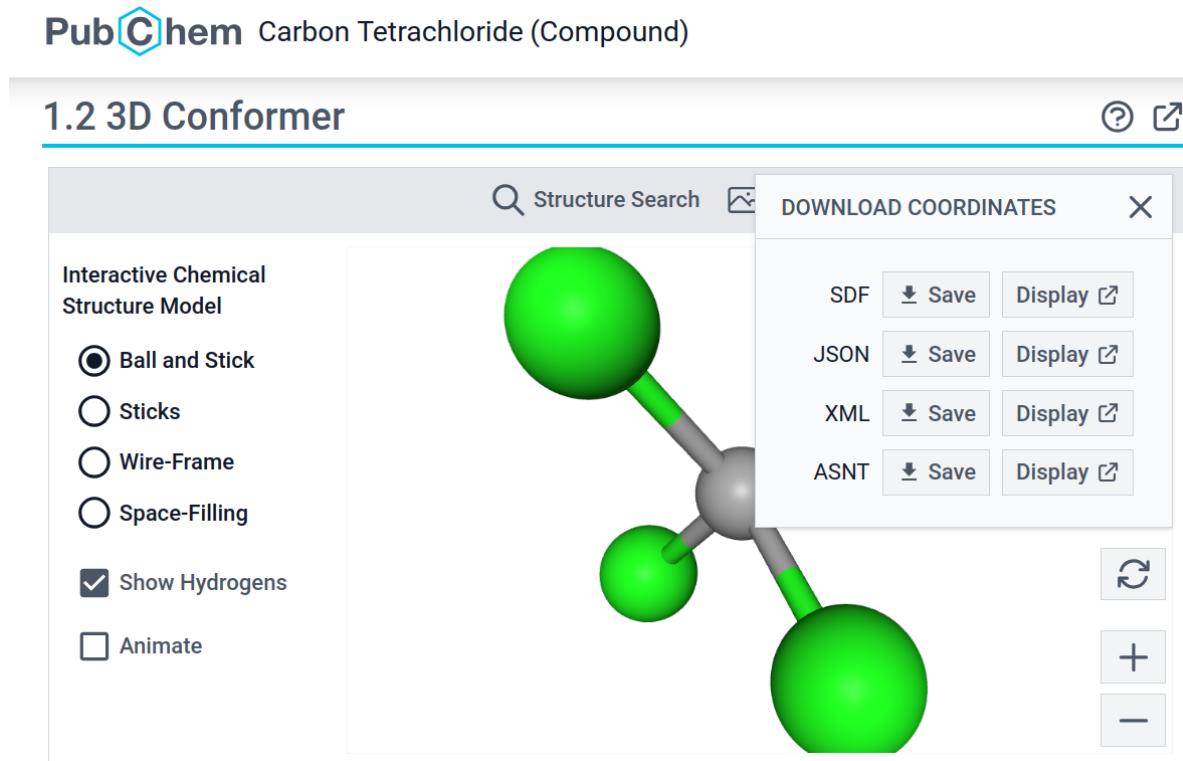


Figura 3.2: Exemplo para *download* de estrutura molecular no computador a partir de sua seleção no site *PubChem*.

### 3.4 Como construir a molécula e visualizá-la no Jmol

Existem diversos programas de desenho molecular, *online* ou por instalação, tais como os listados abaixo:

- [ChemSketch](#)
- [Chemical Sketch Tool](#)

- MolView
- PubChem Sketcher
- ChemDoodle

Também é possível *unir o útil ao (quase)agradável*, carregando uma molécula, adaptando-a ou construindo-a do zero, e visualizá-la diretamente no *JSmol*! Para isso recomendamos os sites abaixo (imagem clicável):

- [Edumol](#).

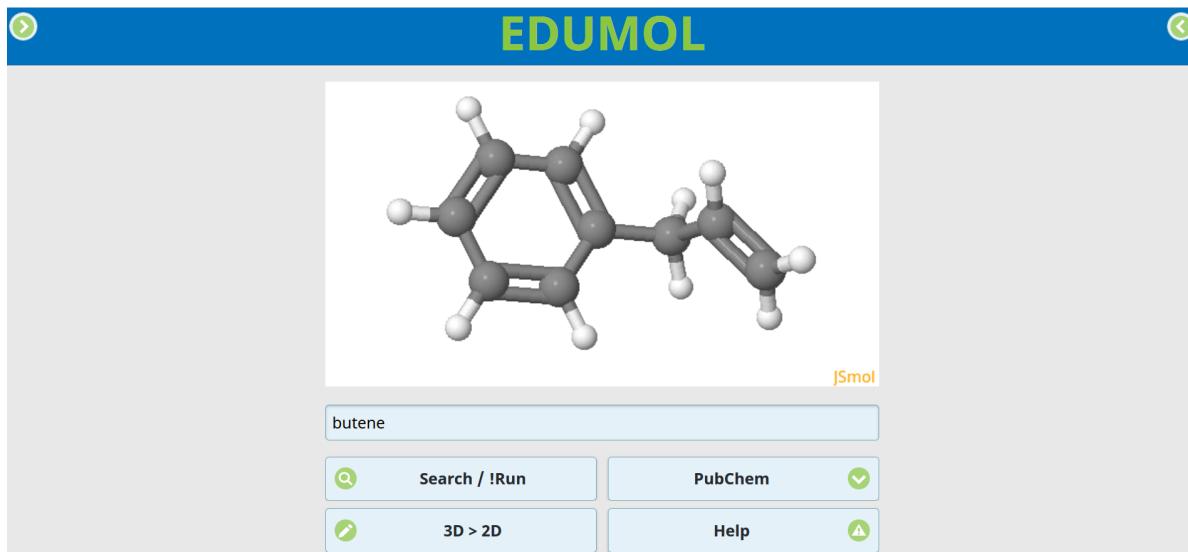


Figura 3.3: Site do *Edumol* para carregamento, modificação e criação de moléculas renderizáveis em applet *JSmol* do próprio website.

- [CheMagic Virtual Molecular Model Kit](#)

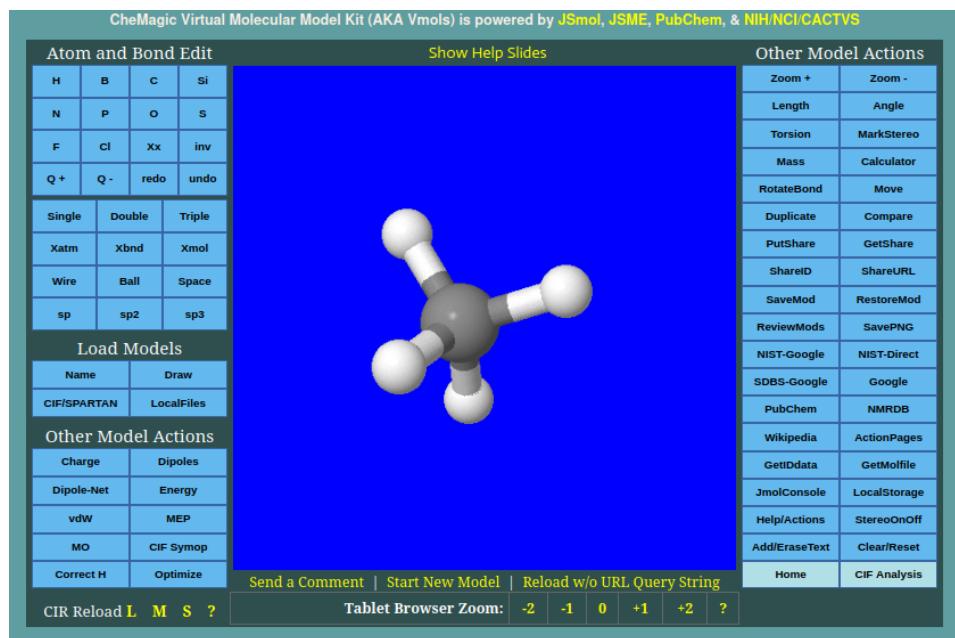


Figura 3.4: Site do ChemMagic Virtual Molecular Model Kit, que acessa diversas funções do *Jmol* por clique de mouse.

## 4 Cliques de mouse *versus* linhas de comando

### 4.1 Cliques de mouse

Qualquer programa de computador que você já tenha usado, ou mesmo de dispositivos móveis, tem sua *usabilidade* centrada na facilidade de uso por cliques e arrastes com *mouse*, *touchpad*, e mesmo os dedos (telas capacitivas). Isso facilita muito as ações rápidas pretendidas. Exemplificando para editores de texto, é comum se clicar num ícone de formatação (itálico, negrito, por ex) ou mesmo digitar seu atalho, para concluir o que se deseja no texto.

Simples, prático, e rápido. Dessa mesma forma, pode-se utilizar o *Jmol*, tanto na versão baixada no computador, como na versão *online*. Para a versão baixada basta observar a gama de ítems de menus e submenus. Já para versão de navegador, veja que não há menu !

Não obstante, a versão *online* permite visualizar a mesma informação, embora com outra formatação, bastando-se clicar *com o botão direito do mouse* em qualquer área do ecrã (nome chique pra tela contendo alguma informação, a molécula, no caso).

### 4.2 Linhas de comando

Assim como para cliques de mouse, também é possível acessar um campo de texto para digitar comandos do *Jmol*, tanto na versão baixada (*standalone*), como na versão de navegador (*applet JSmol*). Para a primeira, clique em *File->Console*, e surgirá uma janela para inserção de texto. Na versão *online*, clique com o botão direito do *mouse* em qualquer ponto do ecrã e escolha *Console*.

### 4.3 Cliques de mouse *versus* linhas de comando

Ainda que seja possível utilizar o *Jmol* tanto cliques de mouse como por comandos de texto, qual é o melhor ?

Para auxiliar na resposta, exemplifiquemos com o uso de uma planilha eletrônica, como o *Excel* do pacote MS-Office, ou o *Calc* do pacote *Libreoffice*, ou o *Planilhas* da suite Google. Suponha que você deseje fazer um gráfico simples, pegando duas colunas, cada qual para uma variável (independente ou *x*, e dependente, ou *y*). O usual seria clicar em um ítem de

menu para gráficos, selecionar as colunas desejadas em campos específicos da janela que se abre, selecionar o tipo de gráfico, clicar em *avançar* ou algum termo similar, selecionar outras características (etiquetas em *x* e *y*, por ex), e finalmente clicar em *concluir* (ou *OK*, ou termo de significado similar). Simples, rápido, e prático.

Mas (sempre tem um “mas”)...e se você precisasse, além de construir o gráfico, realizar ações adicionais, como obter o ajuste linear dos dados, apresentar a reta resultante com determinada cor e estilo, inserir a equação de reta em um ponto específico do gráfico, colocar um título, e alterar o símbolo dos pontos, tanto o tipo, quanto o tamanho e a cor. Ufa !!!

Sem problema, também...desde que você tenha um bom tutorial ao lado, claro ! Ou que já esteja familiarizado com o programa da planilha, menus e ações pertinentes aos vários cliques de mouse que serão necessário para se obter um belo gráfico de regressão linear ao final.

Agora...mais uma pequena variável a inserir ao exemplo levantado: suponha que não seja você a construir o gráfico, mas um aluno(a)(a) de sua disciplina, e que não foi treinado nem no uso da planilha, e nem nos cálculos pretendidos !

Perceba que agora haverá um certo desconforto, posto que:

1. O aluno(a) não possui conhecimento prévio no uso da planilha;
2. O aluno(a) não possui conhecimento prévio nos cálculos pretendidos;
3. Você terá de treinar o aluno(a), ou oferecer-lhe um *guia* de treinamento correlato;
4. Caso já tenha ocorrido o treinamento, mas não se esteja com o *guia* em mãos, tanto você como aluno(a) dependerão da *capacidade de retenção de memória* para efetivar com sucesso a empreita.

Agora, e se as orientações para a execução do produto final estivessem, não num *guia* para a repetição de clique de *mouse*, mas sim num pequeno texto contendo tanto os comandos em sequência como os comentários explicativos de cada ação individual, e que quando inserido no programa gerasse o gráfico todo formatado ?

#### **4.4 Vantagens do uso de linhas de comando sobre o uso de cliques**

Pelo exemplo hipotético acima, perceba que um pequeno texto contendo as linhas de comando em sequência e os comentários referentes a esses permitem:

- que o produto final seja elaborado sem prévio conhecimento do aluno(a); basta executar o código no programa;
- que o produto final seja elaborado independentemente da memória dos envolvidos (sequência de cliques, por ex);
- uma quantidade virtualmente infinita de ações sequenciais, sem necessidade de se decorar a ordem dos cliques de *mouse*;

- o aprendizado de cada comando utilizado em linguagem humana, posto que existem comentários do autor para cada linha;
- que o produto possa ser modificado para gerar um objeto diferente (alteração de cor, etiquetas de eixos, outro título, por ex)
- que se reproduza o mesmo gráfico, só que com outros valores para as variáveis ( $x$  e  $y$ );
- que o aprendiz experimente outros comandos para agregar formatações e/ou cálculos distintos ao produto;
- que você ou o aluno(a) consigam reproduzir o produto sem recorrer à memória e até por séculos depois, se as previsões de extinção em massa não vingarem;
- que qualquer pessoa consiga reproduzir o objeto, independentemente de seu grau de instrução técnica ou de operabilidade do programa;
- enfim, que se consiga ensinar determinado conteúdo de modo reproduutível...ou...**Ensino Reprodutível**.

Dessa forma, pretende-se nesse curso utilizar somente *linhas de comando*, para que se permita materializar-se as vantagens descritas acima, tangentes a uma metodologia voltada, ainda que incipiente, ao *Ensino Reprodutível*, e tanto para a ferramenta *Jmol*, como para a ferramenta *R* & *RStudio*.

Em relação ao *Jmol*, portanto, as ações sequenciais para visualização tridimensional de modelos moleculares será realizada pelo *Console* acessável conforme ítem Seção ?? acima.

## 4.5 Scripts

Colocado da forma acima, quando se tem um conjunto qualquer de linhas de comando sequenciais, permitindo atuar sobre o modelo molecular para uma infinidade de coisas, tem-se então um *script*. Tecnicamente falando, um *script* constitui um bloco de instruções sequenciais em texto para compilação em um programa.

*Scripts* podem ser elaborados no *Jmol* em *browser* por duas maneiras:

1. Separando os comandos por ";" - ex: "cpk only; color blue"
2. Separndo os comandos por linhas - ex:

```
"cpk only
color blue"
```

Se você deseja que o modelo realize uns poucos comandos, a melhor opção é separá-los por ponto e vírgula (";"). Mas se desejar algo mais “sofisticado”, sugere-se separá-los por linhas. E mais...linhas comentadas e escritas em um bloco de notas ou em qualquer editor de texto !

#### 4.5.1 Vantagens do uso de bloco de notas ou editor de texto para comandos em série

Imaginando-se uma transformação mais significativa à molécula original carregada, como efeitos de ampliação, coloração, representação e movimento, é fácil perceber que um conjunto de linhas comentadas dispostas em sequência facilita tanto a observação do que se pretende com o modelo, como a identificação de erros e ajustes.

Isto também é herdado aos conceitos de *Ensino Reprodutível*, uma vez que facilita a visualização do código (*human readable format*) e sua depuração (*code debug*). Veja o exemplo que segue, reflita sobre sua interpretação, copie e teste-o no *Console* do *Jmol*.

```
load $butene
background black # cor preta do plano de fundo
load $colesterol # carrega a molécula de colesterol
delay 1 # aguarda 1 segundo
background white # altera a cor do plano de fundo
spin 80 # gira a molécula
delay 3 # ...por 3 segundos
spin off # interrompe a rotação
cpk # renderiza como modelo de preenchimento
color cpk # coloriza no padrão de modelos moleculares
isosurface molecular # renderiza a superfície
spin 30 # gira mais um poquinho
delay 6 # ...também por 3 segundos
spin off # interrompe novamente a rotação
zoomTo 0.5 *3 # amplia 300% na tela durante 1 segundo
isosurface off # retira a superfície molecular
wireframe only # retorna à representação de varetas
wireframe 50 # espessura das varetas
zoomTo 5 * 0.01 # ... e vai sumindo aos poucos
```

Uma outra opção para adicionar *scripts* diretamente no *Jmol*, verificar a existência de erros previamente à execução dos comandos, e testar as linhas passo a passo para a depuração, pode realizar-se na opção do programa instalado (somente). Para tanto, siga os passos que seguem.

rode o programa *Jmol* que está no computador ou numa mídia removível a partir de seu arquivo *Jmol.jar* após descompactação do arquivo baixado do próprio [site do Jmol](#), e escolha do *Script Editor*

1. Entre no site do [Jmol](https://jmol.sourceforge.net/) (<https://jmol.sourceforge.net/>) ;
2. Faça o Download do programa compactado no computador ou mídia removível ;
3. Localize o arquivo compactado que foi baixado ;
4. Descompacte o arquivo ;
5. Selecione o arquivo "Jmol.jar", de execução em JAVA (é necessário que o computador tenha Java instalado);
6. Selecione "File -> Script Editor"

Para testar o *Editor de Scripts*, copie e cole o *script* abaixo para a papaína, uma proteína do mamão:

```
load=9pap # carregando a estrutura
delay 2 # aguarda 2 segundos
cartoon only # representando em desenho
color structure # coloração por estrutura 2a.
delay 1 # pausa de 1 segundo
zoom 200 # ampliação de 2 vezes
delay 1
reset # restabelece as coordenadas
spin 300 # giro rápido da molécula !
```

Veja que é possível testar se as linhas de comando estão com a sintaxe correta (*check*), testar os comandos linha a linha (*step*), ou rodar o *script* como um todo (*run*).

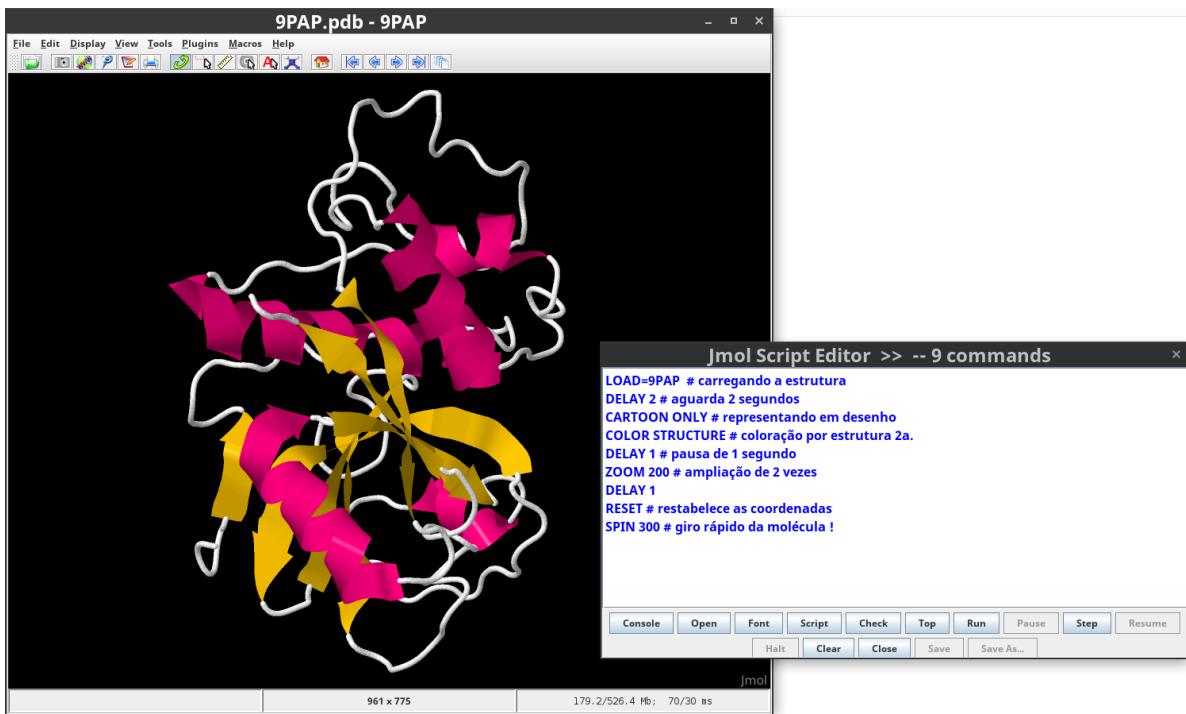


Figura 4.1: Exemplo do uso do *Script Editor* da versão instalada do *Jmol* em computador ou mídia removível.

Alguns exemplos de *scripts* são ilustrados na Seção ??.

## 5 Alguns comandos pra se aventurar nas moléculas voadoras

### 5.1 Como carregar uma molécula no JSmol

Supondo que você já tenha aberto em seu navegador a janela para o *applet* do *JSmol* mas que, contrariamente ao que foi feito antes (nome da molécula ao final do site [PubChem](#), você queira:

- carregar uma molécula a partir de outro banco de dados;
- carregar uma molécula cujo arquivo já esteja em seu computador

Bom, nesse caso você pode usar o *mouse* ou uma *linha de comando*, como preferir.

#### 5.1.1 Carregando a molécula com o *mouse*

Para isto basta clicar com o botão direito do *mouse* no ecrã, como anteriormente, e selecionar *File->Load*. As opções que se apresentam são:

```
* Open local file # abre janela para buscar o arquivo do modelo no computador;
* Open URL # abre janela para buscar o endereço de internete que possui o arquivo
* Get PDB file # abre janela para inserir um código de macromolécula do site homônimo (protein)
* Get MOL file # abre janela para buscar um arquivo *.mol
* Open script # abre janela para buscar um trecho de código no computador
```

A primeira opção é autoexplicativa (*Open local file*), a segunda opção (*Open URL*) depende do endereço correto para um determinado modelo molecular, a terceira (*Get PDB file*) refere-se ao banco de dados [Protein Data Brookhaven](#) para biopolímeros, a quarta (*Get MOL file*) envolve a busca *online* em banco de dados específico para pequenas moléculas, e a última (*Open script*), a busca de um arquivo que contenha linhas de código do *Jmol* para um conjunto de ações.

Como os livros didáticos permeiam estruturas moleculares pequenas, normalmente associadas aos grupos funcionais da *Química Inorgânica e Orgânica*, bem como exemplos específicos em áreas como *Saúde, Biotecnologia e Indústria*, incluindo também *alguns modelos*

*de macromoléculas*, pode-se concluir que é mais provável que você utilize a busca remota de pequenas moléculas (*Get MOL file*), moléculas contidas em seu computador (*Open local file*), e/ou biomacromoléculas (*Get PDB file*).

O carregamento de pequenas moléculas é *idêntico* ao que foi experimentado adicionando-se o nome do modelo ao final do endereço do [JSmol](#). O carregamento remoto para modelos de proteínas, enzimas e ácidos nucleicos envolve o conhecimento do *código PDB* desses, ou busca de palavras-chave no sítio [Protein Data Brookhaven](#).

Já o carregamento de moléculas guardadas no PC envolve algumas poucas etapas, a saber:

1. Obtém-se o modelo da molécula pela internete, ou o constroi;
2. Baixa-se o arquivo correspondente ao modelo (geralmente com um atributo \*.mol, \*.cif, \*.cml, \*.sdf, entre mais de 60 formatos);
3. Carrega-se na página do [JSmol](#) por dois meios alternativos:

- 1. Por clique de mouse: File --> Load --> Open local file ;**
- 2. Por arraste do arquivo da pasta onde se encontre para a aba do JSmol no navegador.**

Exemplificando, digamos que você queira visualizar a estrutura da *aspirina* baixada em seu computador.

- 1. Baixe o modelo estrutural da aspirina em algum site, como o PubChem ;**
- 2. Abra o Console do JSmol no navegador (clique no ecrã com o botão direito do mouse e selec;**
- 3. Alternativamente:**
  - a. Localize o arquivo no PC por "File-->Load-->Open local file", clicando depois em "Load"**
  - b. Clique no arquivo baixado ("aspirin.sdf", por ex) e arraste-o diretamente para a janela**

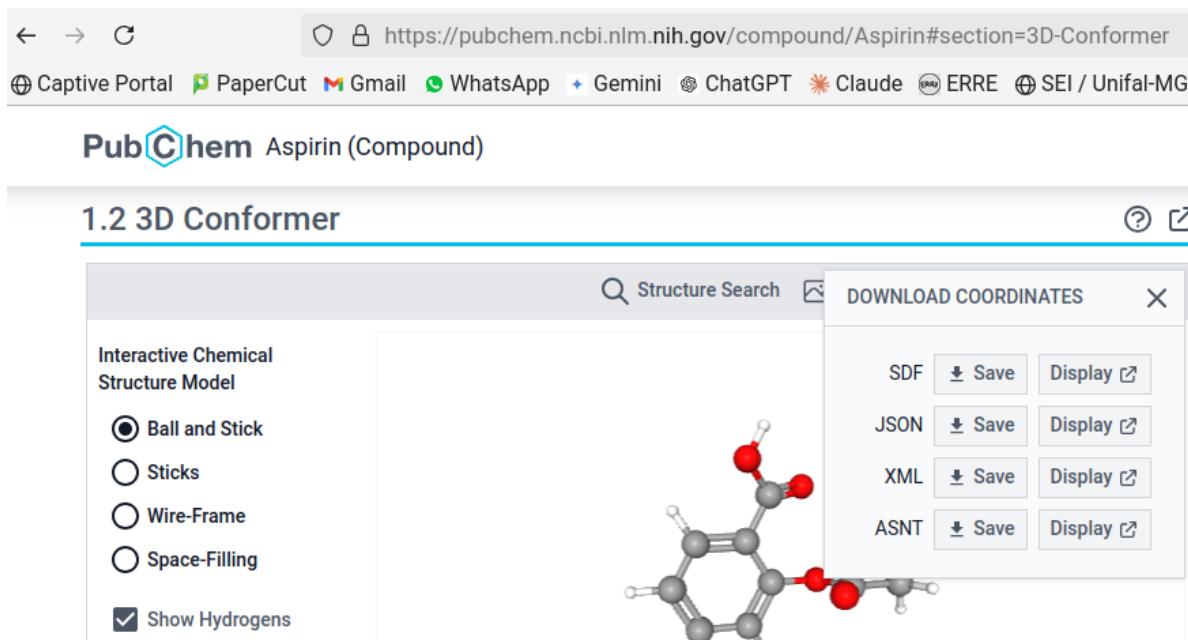


Figura 5.1: Exemplo do modelo da aspirina para *download* no PubChem.

### 5.1.2 Carregando a molécula por linha de comando

O carregamento de um modelo em particular por linha de comando restringe-se à sua busca pela internet, em banco de dados ou páginas da *web*. Para isso, abre-se o *Console* como já explicado. A parte de cima serve para apresentação dos resultados dos comandos, e a parte de baixo, para sua digitação. Nesse caso, clique no quadro inferior do *Console* e digite o comando de carregamento, aqui exemplificado para um *alcano*:

```
load $alkane
```

O *Console* do *Jmol*, ainda que constitua uma linguagem própria de programação de comandos, possui uma vantagem interessante sobre demais linguagens de programação: é possível efetuar o comando pelo *Console* tanto com letras maiúsculas como minúsculas, e tanto no singular como no plural.

Você pode tentar com outras moléculas, como *aspirin*, *cholesterol*, *phenol* etc (nomes em inglês, por conta do banco de dados). Para recuperar uma linha de comando que foi escrita antes, basta navegar entre os comandos que foram utilizados com as setas para cima e para baixo do teclado.

Os modelos moleculares são carregados a partir do banco de dados [Cactus - CADD Group Chemoinformatics Tools and User Services](#).

### 5.1.2.1 Carregando moléculas por notação SMILES

[SMILES-Simplified Molecular Input Line Entry System](#) trata de uma notação química que permite representar as moléculas inserindo-se o nome de cada átomo em sequência em ligações simples, bem como o local de ligações insaturadas, ou ligações aromáticas.

O [PubChem](#) possui também a notação *SMILES* como alternativa para carregamento de modelos. Seguem alguns exemplos da notação para carregamento da molécula no *Console* do *Jmol*.

Para hidrocarbonetos sem ligações insaturadas (*alcanos*) é possível carregar a molécula digitando somente a sequência de carbonos, como segue:

```
# Para carregar um metano  
load $C # carrega o modelo
```

```
# Para carregar um hexano  
load $CCCCCC # carrega o modelo
```

Agora, para carregar hidrocarbonetos com uma ligação insaturada (*alcenos*), basta inserir o sinal de igualdade (=) onde aparece a insaturação:

```
# Para carregar um propeno (propileno)  
load $CC=C # carrega o modelo
```

Para carregar um modelo com ligação tripla, segue-se a notação #:

```
# Para carregar um metilacetileno (ou ....propina !)  
load $CC#C # carrega o modelo
```

Para carregar uma estrutura aromática, já um pouco mais chatinho:

```
# Para carregar um benzeno  
load $C1=CC=CC=C1
```

### 5.1.3 Carregando biopolímeros (proteínas, enzimas, ácidos nucleicos) por linha de comando

Como mencionado acima, o carregamento de macromoléculas biológicas dá-se por identificação de um código alfanumérico da mesma frente ao banco de dados [PDB-Protein Data Bank](#). Após obter esse código, o comando de carregamento é:

```
load=XXXX # onde XXXX é o código da macromolécula110
# Obs: Perceba que o sinal de "$" é trocado por "=" para o PDB
```

Isso pode ser ilustrado por carregamento remoto da proteína espícula (*spike*) do vírus Sars-Cov-2, tal como segue:

1. Entre no site do PDB-Protein Data Bank - <https://www.rcsb.org/> ;
2. No campo de busca, digite "spike sars-cov-2" ;
3. Selecione a 1a opção (o site vai direcionar para várias estruturas da proteína espícula)
4. Memorize o código da 1a. opção (embora qualquer uma também sirva), ou seja, "7FCD" ;
5. Digite a linha para carregar a proteína: "load=7FCD" (tanto faz se maiúsculas ou minúsculas).

A representação padrão para proteínas no *Jmol* é a de arame (*wireframe*). Para visualizar a proteína do vírus de modo mais “amigável” e semelhante ao que aparece em textos ou na internete, digite os comandos abaixo.

```
cartoon only # representação exclusiva de desenho da estrutura de biopolímeros
color chain # coloração por "cadeias" da proteína
```

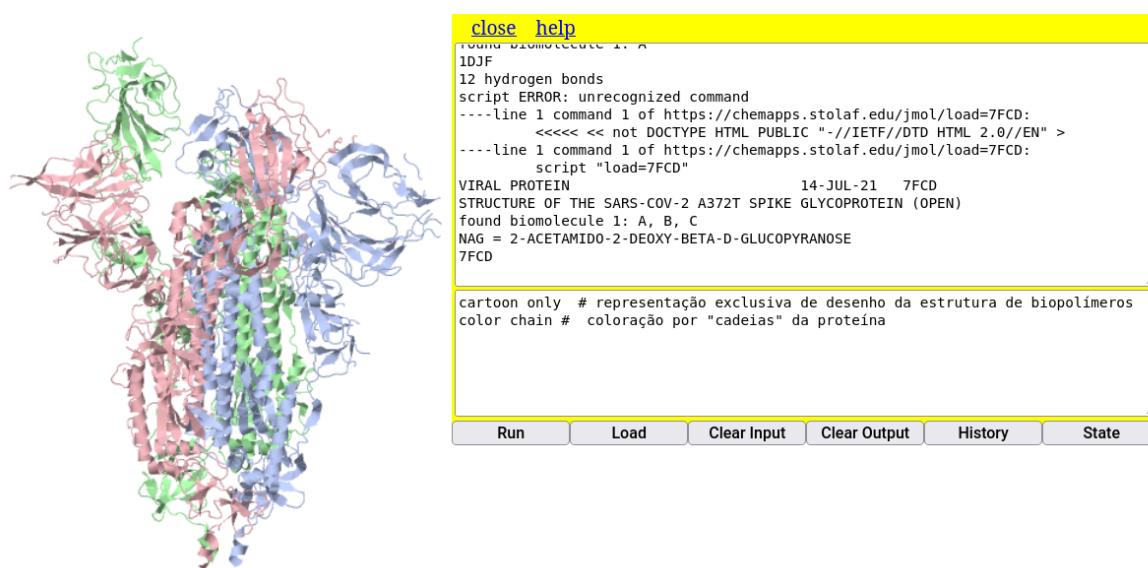


Figura 5.2: Representação da proteína *spike* do vírus de Sars-Cov-2 (coloração por número de cadeias).

Proteínas, enzimas, ácidos nucleicos, e associações macromoleculares são mais pertinentes ao estudo da *Bioquímica estrutural*. Nesse sentido lhe convido a visitar uma parte do *website*

autoral que possui descrições e representações detalhadas de estruturas bioquímicas com auxílio do *Jmol*, o site [Bioquanti](#)

Bioquímica Quantitativa

Objetivo

Página elaborada no intuito de contribuir para o ensino-aprendizagem de conteúdos de Bioquímica e áreas correlatas, e esboçada para tangenciar um **Ensino Reprodutível (ER)** com *Recursos Educacionais Abertos (REA)*. O material *autoinstrucional* permite o estudo dos temas juntamente à execução de códigos de programas de livre distribuição para

Figura 5.3: Bioquanti, um website autoral para estudos quantitativos em Bioquímica, e que inclui diversos modelos moleculares para o *Jmol*.

## 5.2 Agora que a molécula está na página do navegador, o que posso fazer com ela ?

Muuuuuiita coisa !!!

O *Jmol* possui um *menu* com diversas operações, e centenas de comandos, e talvez outra centena de tutoriais pela internete. Para observações estruturais mais diretas e imediatas, contudo, pode-se resumir as operações em:

- Movimentos com mouse (rotação, translação, *zoom*)
- Representações do modelo (bola e varetas, espaço preenchido, arame)
- Cores (modelo e plano de fundo)
- Medidas (distâncias e ângulos)
- Características moleculares (ligações de H, nuvem de van der Waals, carga parcial e efetiva)
- Superfícies (molecular, eletrostática)
- Seleção de átomos e visualização (água, hidrogênio)
- Animações (zoom, rotação automática), cortes

Uma observação importante é que *todos os comandos do Jmol possuem ajuda explicativa*. Para tanto, basta digitar:

```
help nome.do.comando
```

### 5.3 Salvamento do modelo no computador ou dispositivo móvel

Todas as ações realizadas com a molécula produzem um novo modelo que pode ser baixado para o computador. E isso é bem legal porque a molécula modificada (com alteração de cores, representações, animações, por ex) pode ser carregada no *Jmol* ou na *JSmol* (internet) como já mencionado. Para tanto, pode-se usar cliques de mouse ou linhas de comando, como segue:

1. Botão direito no ecrã -> File -> Save -> Save as PNG/JMOL # por mouse
2. write nome\_da\_molecula # por linha de comando no Console

Obs: a opção PNG/JMOL é muito interessante, já que permite visualizar o modelo trabalhado em qualquer editor de imagem, bem como no *Jmol*.

Para exemplificar essas ações, usaremos inicialmente o modelo da *vitamina C*, carregando-o com o comando `load $ascorbate`. Mas se você quiser saber todos os comandos possíveis, faça uma visita ao [site de referência do Jmol](#).

### 5.4 Movimentos com mouse

Para rotação e translação do modelo, bem como ampliação:

```
zoom - botão do meio do mouse; se não houver o botão, Shift+botão esquerdo  
rotação - botão esquerdo do mouse  
translação - Ctrl+botão direito  
rotação no eixo - Shift+botão direito
```

### 5.5 Representações do modelo

As representações referem-se ao aspecto visual do modelo (renderização). Assim, o *Jmol* pode renderizar o modelo como *vareta*, *arame*, *espaço preenchido*, *bola e vareta*, *traço*, e *desenho*. Experimente renderizar o nesses tipos, incluindo a opção `only`. Essa opção permite que a ação não seja sobreposta às anteriores (no caso, a sobreposição das representações).

```
wireframe only # arame  
cpk only # espaço preenchido  
trace only # traço  
cartoon only # desenho
```

Observe também que representação em `cartoon` não resulta numa renderização para o modelo da vitamina C. Isso decorre porque a representação em `cartoon` é restrita para biopolímeros, ou seja, proteínas e ácidos nucleicos.

Contudo, se quiser experimentar o `cartoon`, será necessário conhecer o código alfanumérico de uma proteína ou ácido nucleico. Exemplificando para a mioglobina, proteína transportadora de oxigênio em mamíferos (código: `1mcy`)

```
load=1mcy # carregando a mioglobina
```

Veja que a renderização padrão para grandes moléculas é a de bolas e varetas, pouco didática para o aprendiz. Nesse caso, pode-se representá-la como desenho exclusivo, digitando-se:

```
cartoon only # renderizando em desenho
```

Para se conseguir esse e outros códigos de proteínas e ácidos nucleicos, deve-se entrar no banco de dados do [PDB - Protein Data Bank, RCSB](#), e digitar o nome no campo de busca (no caso, `myoglobin`). O sistema retorna diversos modelos estruturais e seus códigos, bastando transcrever um desses códigos ao *Console* do *JSmol*.

## 5.6 Cores

Existe grande flexibilidade de `cores` para o *Jmol* (e, por consequência, para o *JSmol*), tanto para os modelos inteiros, partes do modelo (átomos específicos ou um conjunto), e plano de fundo. A visualização padrão de cores segue a convenção [CPK \(Corey–Pauling–Koltun\)](#). Exemplificando para o modelo anterior de vitamina C (`load $ascorbate`), experimente a variação que segue:

```
color pink  
color blue  
color lightgreen  
background yellow # plano de fundo
```

O último comando da lista acima permite variar a coloração do plano de fundo.

Adicionalmente, também é possível a coloração de ligações entre os átomos, como segue:

```
color bonds LightSeaGreen
```

Para um grande espectro de cores, você pode consultar a referência do [Jmol Colors](#), ou um *link* mais “mastigado”, de nossa autoria junto ao aprendizado do programa no ensino superior, o portal [Bioquanti](#) e, mais especificamente, o [tópico de cores para o Jmol](#).

## 5.7 Medidas

O *Jmol* permite calcular distâncias e ângulos em um modelo molecular. Para exemplificar isso, talvez seja interessante o carregamento de um *modelo de água* (`load $water`), e cujas distâncias e ângulos estão presentes em alguns livros de Química.

### 5.7.1 Para distâncias

No exemplo da molécula de água, para se determinar a distância de uma ligação O-H, por exemplo:

1. Duplo-clique do mouse no **1o.** átomo;
2. Arraste do mouse ao **2o.** átomo;
3. Clique do mouse no **2o.** átomo

Experimentando para a distância da ligação O-H, o programa retorna o valor de 0,097 nm, ou 0,97 Angstroms, o valor convencional para esse tipo de ligação covalente.

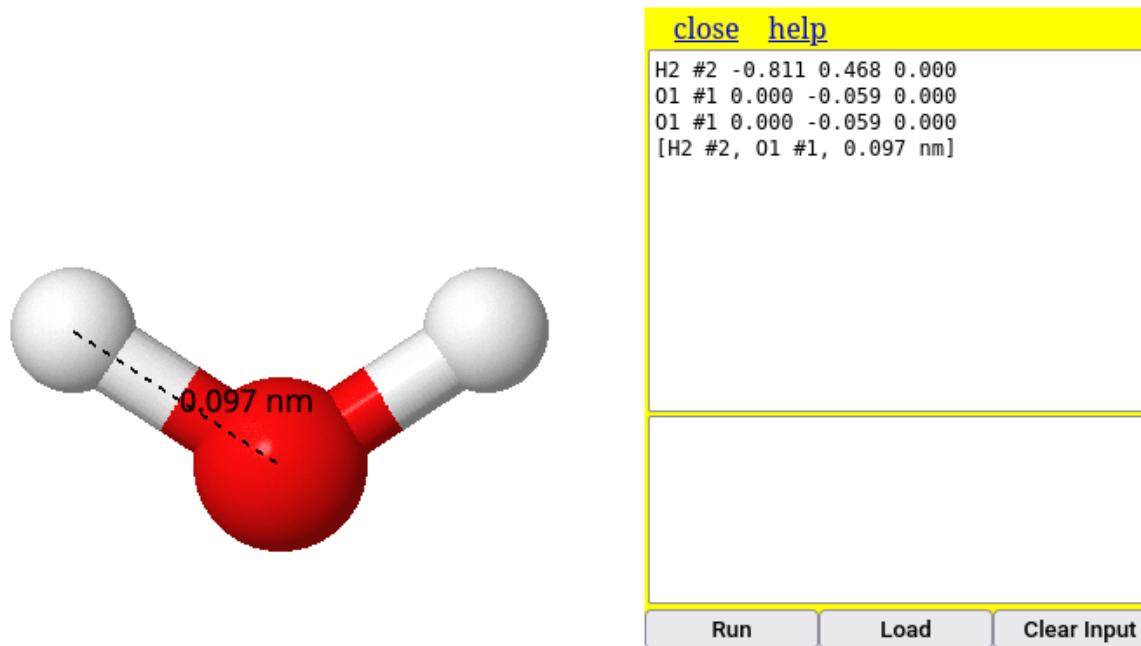


Figura 5.4: Medindo distância dentro da molécula.

### 5.7.2 Para ângulos

Para a mesma molécula de água, experimente determinar o ângulo de ligação:

1. Duplo clique no **1o.** átomo (ex: H);
2. Arrasta ao **2o.** átomo (ex: O);
3. Clique no **2o.** átomo;
4. Arraste ao **3o.** átomo (ex: o outro H);
5. Clique no **3o.** átomo

Perceba que o sistema retorna o valor de  $114^\circ$ , um valor próximo do previsto para a molécula ( $109.5^\circ$ ), ou medido ( $104.5^\circ$ ). Essa aproximação é decorrente da construção do modelo de água.

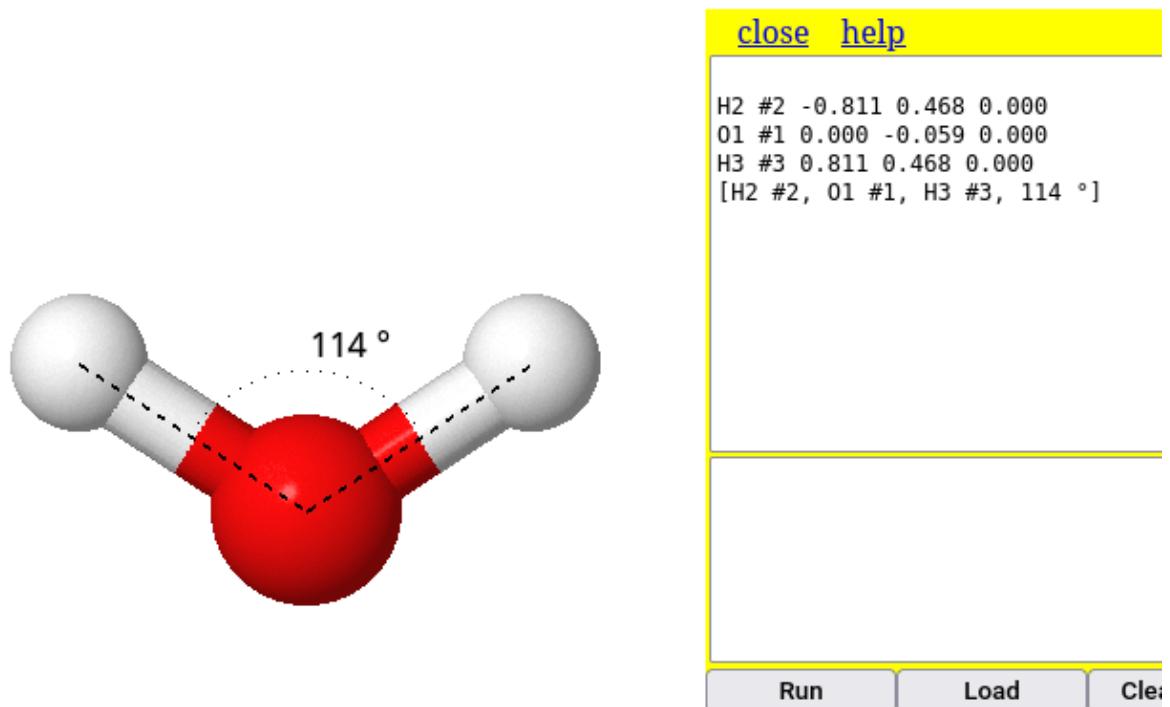


Figura 5.5: Medindo ângulo dentro da molécula.

Para limpar as medidas, use o comando:

```
measure off
```

## 5.8 Características moleculares

São diversas as informações tangíveis a um modelo molecular no *Jmol*. Exemplificando as mais básicas para a molécula de um componente do molho *shoyo*, o glutamato:

### 5.8.1 Cargas

Há dois tipos de cargas oferecidas pelo *Jmol*, *carga efetiva* (*formalCharge*) e *carga parcial* (*partialcharge*). Exemplificando, digite no *Console* os comandos abaixo:

```
calculate partialCharges # cálculo de cargas parciais do modelo  
label %P # apresentação das cargas (etiquetagem)
```

Uma característica do *Jmol* que o torna mais eficiente a execução de suas ações é a disposição sequencial de comandos. Dessa forma, não é necessário clicar em *Enter* para cada comando, bastando separar os comando por ponto e vírgula (;) como ilustrado abaixo:

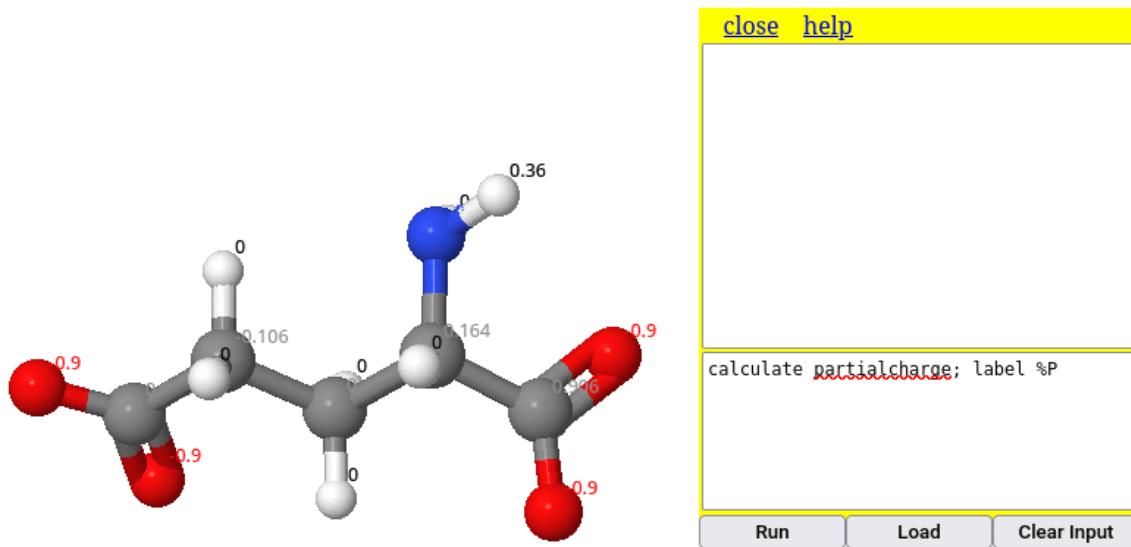


Figura 5.6: Apresentação de cargas parciais no modelo de glutamato, um componente do molho Shoyo, também ilustrando uma sequência de ações no Jmol.

Da mesma forma pode-se ilustrar a obtenção de *cargas formais* no modelo. Nessa, adicionou-se a coloração transparente, para melhor visualização da carga unitária negativa do ácido carboxílico:

```
calculate formalcharges # cálculo de cargas parciais do modelo  
label %C # apresentação das cargas (etiquetagem)
```

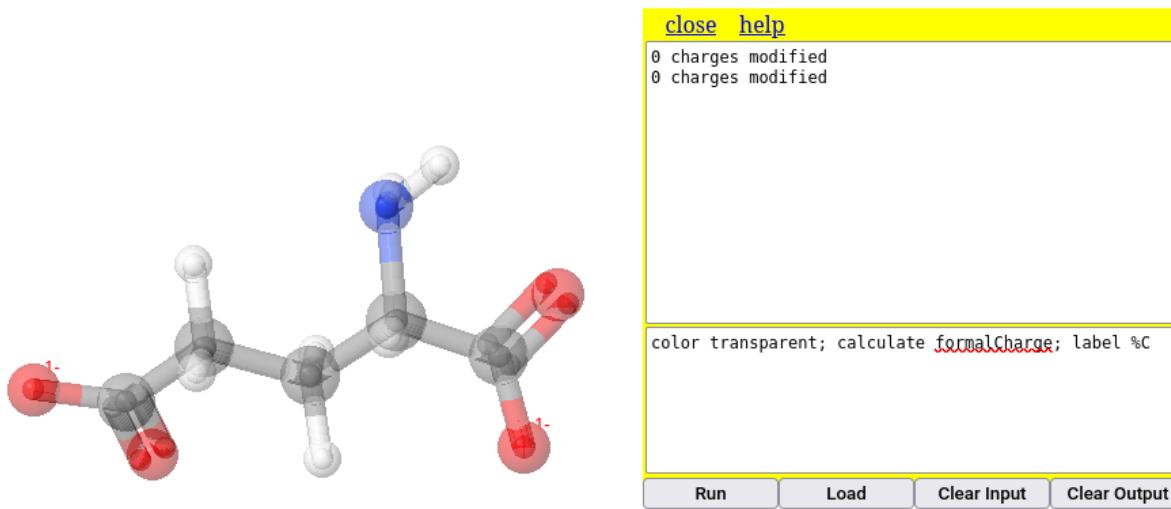


Figura 5.7: Ilustração dos comandos em sequência para visualização de cargas formais na molécula de glutamato.

Perceba que os comando da figura mistura maiúscula e minúsculas, de modo diferente da linha de comando que a antecede. Essa é uma **característica bem legal do *Jmol*, que não se importa com a capitalização ou não da fonte**. Ou seja, tanto faz se minúsculo, maiúsculo ou uma combinação de ambos; o *Jmol* executa a ação do mesmo modo.

#### 5.8.1.1 Scripts & Ensino Reprodutível

O exemplo acima apresenta uma maneira simples de concatenar comandos, facilitando a execução automático e sequencial de um conjunto desses. No entanto, a visualização da linha de comando fica um pouco prejudicada com a separação por “;”, o que pode acarretar uma poluição visual quando houver vários comandos.

A situação de contorno envolve a disposição dos comandos no formato de um *script*. Esse nada mais é do que um trecho de código contendo um comando por linha, o que melhora a visualização do código como um todo. Além disso, o *script* possui a vantagem adicional de se inserir comentários entre as linhas de comando, permitindo também uma melhor apropriação do código e de seu aprendizado.

Essas características de um *comando por linha com comentários explicativos* conferem ao *Jmol* seu aspecto para *programação* de ações sequenciais, e enraiza por consequência uma das premissas básicas para um *Ensino Reprodutível*: a redação de trechos de códigos em comandos unitários por linha, escritos como num bloco de notas, e com comentários sobre as ações do

programa em cada linha. Exemplificando para um *script* envolvendo as ações para o glutamato acima, apenas copie o trecho abaixo e cole-o no *Console* do *JSmol*, executando-o.

```
load $glu # carregamento de micromolécula  
wireframe only # renderização exclusiva de varetas  
calculate partialCharge # carga parcial  
label %P
```

Outro aspecto inerente à iniciativa de *Ensino Reprodutível* reside na *possibilidade de se avaliar o código com alguma alteração*, objetivando um produto final ligeiramente modificado. Tente repetir o trecho acima, mas para cargas efetivas, ou seja:

```
load $glu # carregamento de micromolécula  
cpk only # renderização exclusiva por espaço preenchido  
calculate formalCharge # carga efetiva  
label %C
```

Complementarmente, pode-se atuar alterando mais comandos do código, de modo a criar um resultado completamente diferente do original. Isso define outra característica do *Ensino Reprodutível*, qual seja, a de *criação de trecho de código*. Ilustrando, segue um trecho baseado no anterior, mas para minimização de energia e reestruturação dos orbitais da molécula.

```
load $glu # carregamento de micromolécula  
cpk only # renderização exclusiva por espaço preenchido  
minimize # comando para minimização de energia da estrutura
```

## 5.9 Características moleculares

Além de previsão estrutural para *carga parcial* e *carga formal* (embora essa última dependa na prática de informação adicional, o valor de pH do meio), o *Jmol* também permite evidenciar *forças fracas* no modelo, tais como *nuvem de van der Waals* e *ligações de hidrogênio*, como segue.

### 5.9.1 Nuvem de van der Waals

```
dots on # nuvem de van der Waals nos átomos do modelo (retira-se com "dots off")  
calculate hbonds # identifica ligações de hidrogênio no modelo
```

Ilustrando, copie e cole o trecho que segue no *Console*:

```
load $water
dots on # nuvem de van der Waals na estrutura da água
dots ionic # nuvem iônica sobre o modelo
```

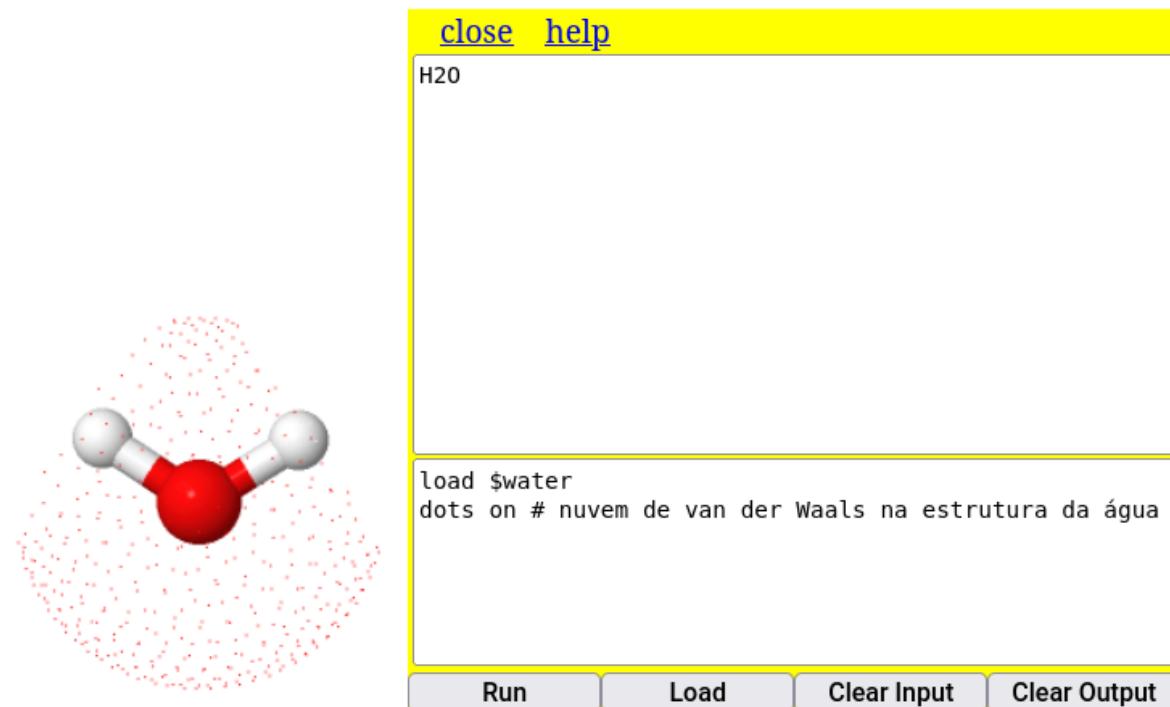
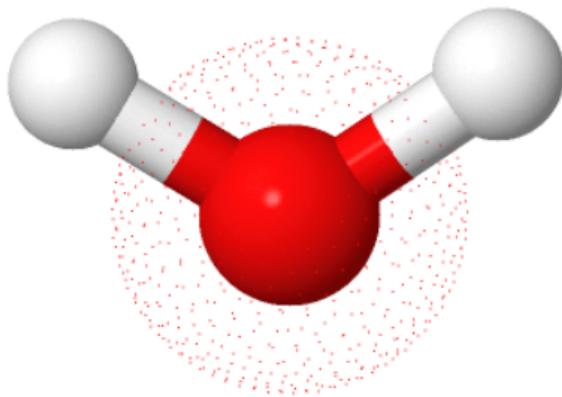


Figura 5.8: Exemplificando a sobreposição de nuvens de van der Waals nos átomos da molécula de água.



```
close help
Created by Jmol version 10.2.15 2024-01-01
GMT-0300 (Brasilia Standard Time)
solvent/molecular surface
solvent-accessible surface with radius 1.

isosurface resolution for axis 1 set to 2
isosurface resolution for axis 2 set to 2
isosurface resolution for axis 3 set to 2
216000 voxels total

script compiler ERROR: command expected
-----
>>> dot ionic <<<

dots ionic
```

**Run**    **Load**    **Clear Input**

### Ligações de hidrogênio

```
load=1djf # carrega um modelo de peptídio
calculate hbonds # apresenta as ligações de H presentes na estrutura
```

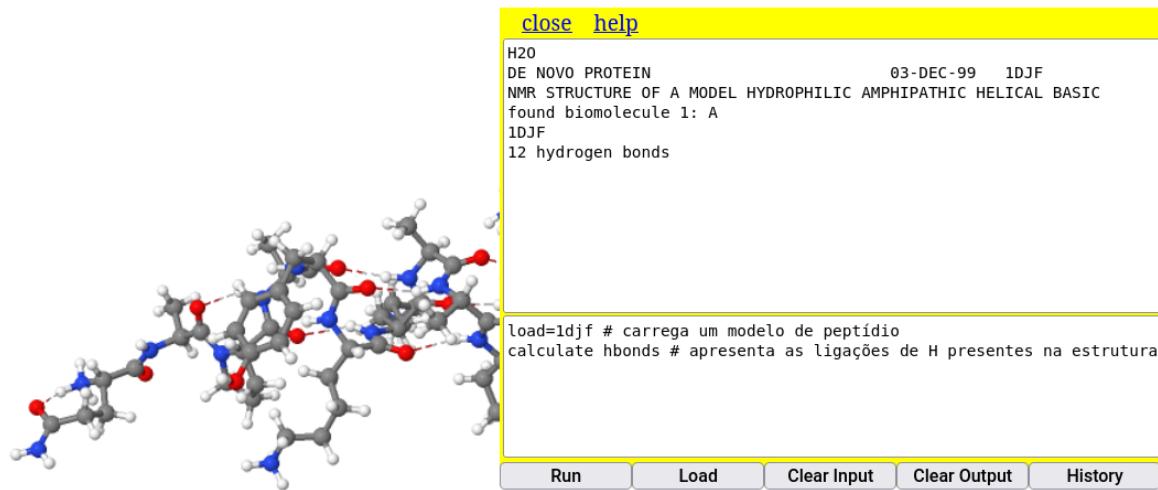


Figura 5.9: Exemplificando ligações de hidrogênio num modelo de peptídio.

## 5.10 Superfícies

Além da superfície de van der Walls (*dots on*) vista acima, o *Jmol* é capaz de representar algumas superfícies para modelos moleculares. Quanto maior a molécula, maior o cálculo interno para gerar a superfície, o que pode dificultar sua visualização. Assim, ilustrando algumas superfícies para a molécula de água:

```
isosurface molecular # superfície molecular que inclui o solvente
```

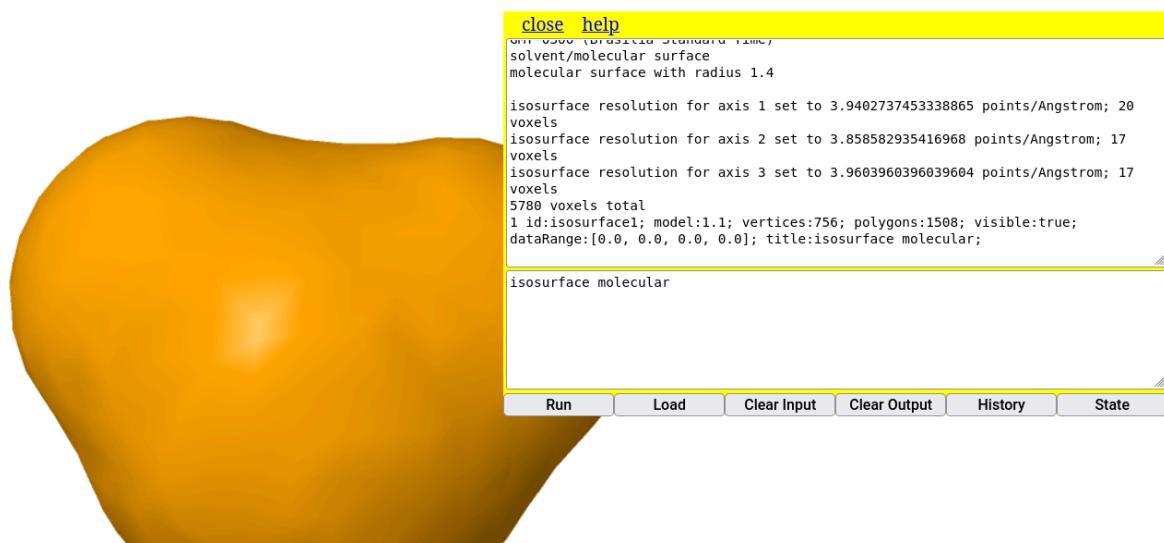


Figura 5.10: Superfície molecular para o modelo da água.

```
isosurface mep # superfície de potencial eletrostático molecular
```

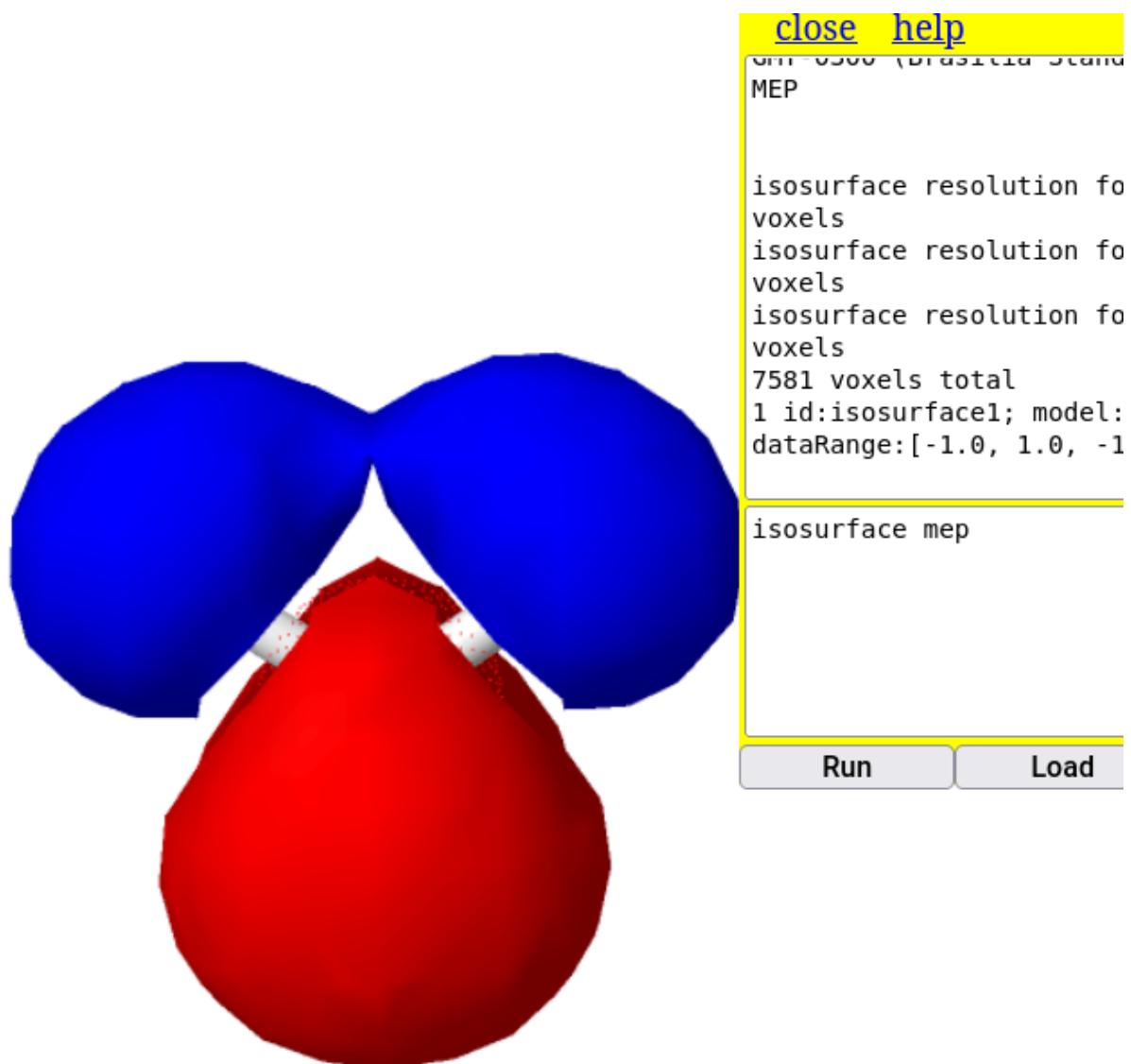


Figura 5.11: Exemplo de superfície de potencial eletrostático para a água.

```
isosurface resolution 6 molecular map mep # superfície de potencial eletrostático molecular
```

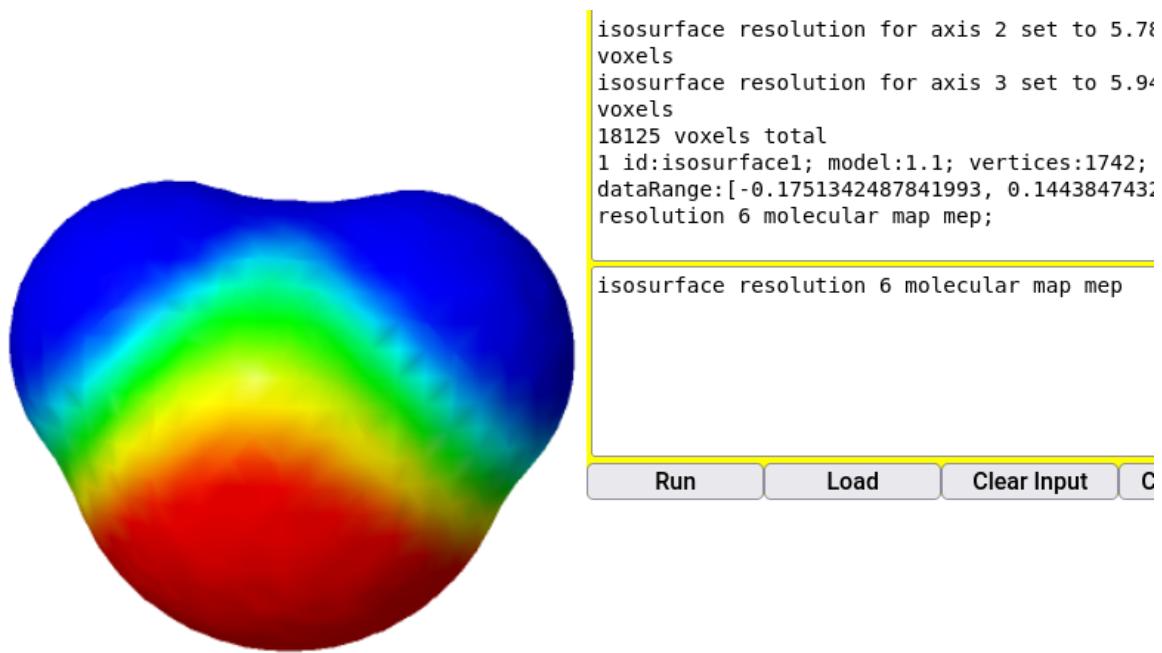


Figura 5.12: Exemplo de superfície molecular da água que inclui seu potencial eletrostático.

Várias outras combinações de superfície podem ser obtidas no [site de referência do *Jmol*.(<https://chemapps.stolaf.edu/jmol/docs/>)

# 6 Seleção de partes da molécula

## 6.1 Seleção de átomos e visualização

Por vezes faz-se interessante ou necessário chamar a atenção para partes específicas de um modelo, como um átomo, grupo de átomos, uma molécula pequena no interior de uma maior, entre outras situações. Nessa seção serão vistos alguns comandos para isso.

O *Jmol* foi concebido para estruturas mais complexas, como proteínas e polímeros, razão pela qual a maior parte dos comandos de seleção refiram-se a partes de proteínas e ácidos nucleicos. Contudo, pode-se selecionar átomos ou grupo desses em moléculas pequenas, também.

### 6.1.1 Por mouse

Usando o *mouse* pode-se selecionar um átomo por vez ou grupo de átomos (esse no *Jmol* do computador), como segue:

1. Átomo – basta clicar em algum no ecrã (modelo);
2. Grupo de átomos – Shift + clique botão esquerdo + arraste pra agrupar os átomos # funciona

### 6.1.2 Por linha de comando

Por digitação do código consegue-se uma operação bem mais abrangente que por clique de *mouse*, incluindo o agrupamento de átomos ou seleção daqueles com características comuns:

```
select "propriedade do átomo"
```

Essas características são listadas abaixo:

```

todos os átomos: all
nenhum átomo: none
solvente: solvent
água: water ou hoh
íons: ions
átomo por símbolo atômico: _N, _C, _Fe
átomo por número atômico: elemNo=7
átomo por identificação na sequência: atomNo<50

# Para biomoléculas:
proteína: protein
ácido nucleico: nucleic
DNA: dna
purina: purine
pirimidina: pyrimidine
carboidrato: carbohydrate
aminoácido: amino

# Para aminoácidos numa proteína:
abreviação de 3 letras: his, tyr, leu, etc
grandes: large
pequenos: small
ácidos: acidic
básicos: basic
polares: polar
apolares: hydrophobic
alifáticos: aliphatic
hidrofílicos: hydrophilic
aromáticos: aromatic
neutros: neutral
carregados: charged
na superfície: surface
no interior: buried
cistina: cystine
hem - grupo heme de ferroproteínas

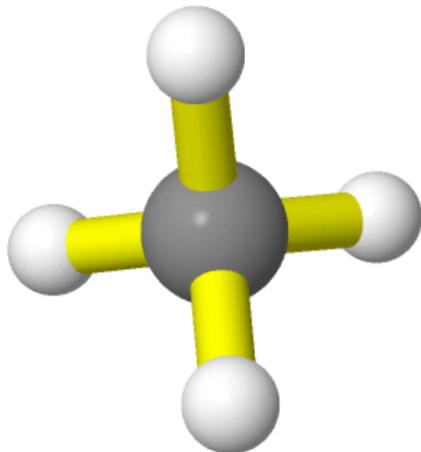
# Para proteínas
esqueleto carbônico de resíduos de aminoácidos: backbone ou spine
cadeia lateral de resíduos de aminoácidos: sidechain
átomos distintos dos presentes nos resíduos de aminoácidos: hetero
estruturas secundárias: helix, sheet, turn

```

## 6.2 E depois de selecionar átomos, o que é que eu faço ?!

Muuuiitaaa coisa, também ! As instruções sequenciais (*scripts*) ilustram algumas possibilidades de se observar detalhes de um modelo molecular com o *Jmol*. Você pode copiar e colar no *Console* para observar os resultados.

```
# Para colorir ligações as do metano  
  
load $methane # carrega o modelo ...ou load $C  
color bonds yellow # coloração para as ligações
```



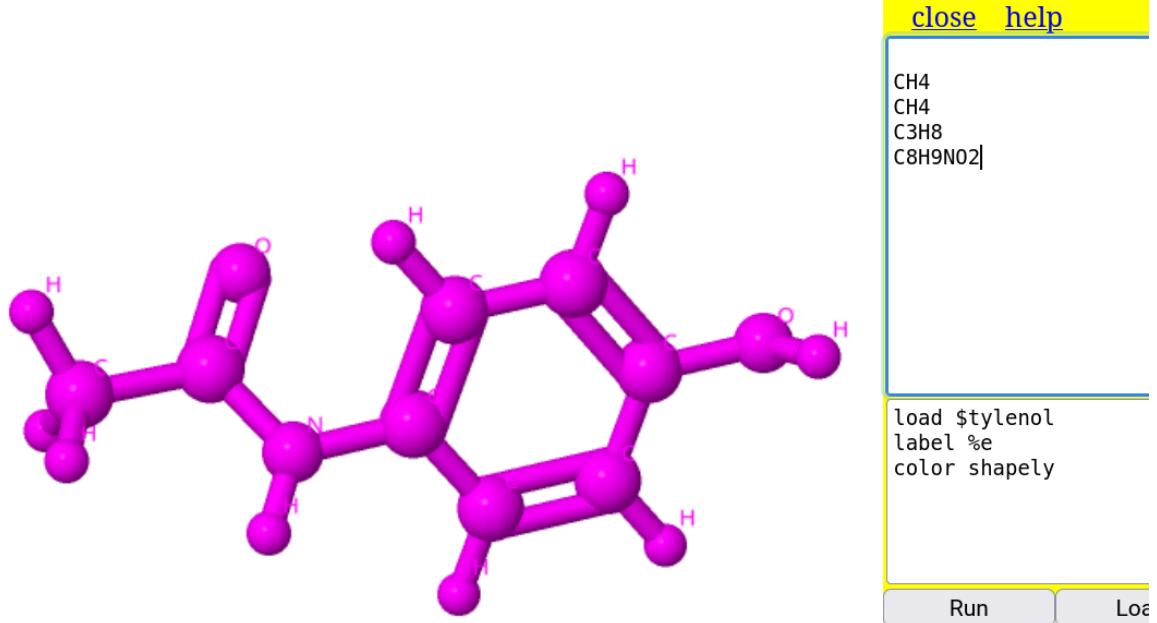
close help

| CH4

# Para colorir ligações as do metano  
load \$methane # carrega o modelo ...ou load \$C  
color bonds yellow # coloração para as ligações

Run Load Clear Input Clear Output

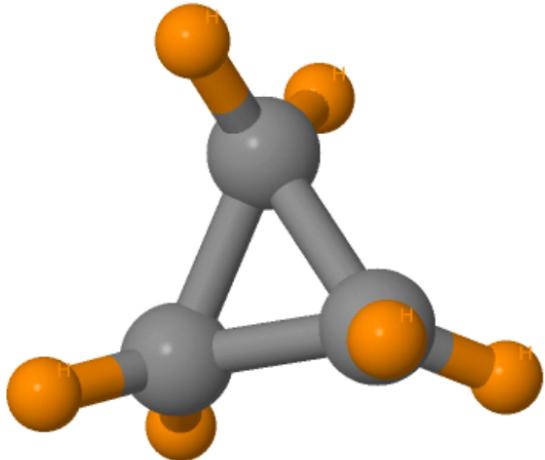
```
# Para nomear os átomos do tylenol e padronizar uma cor  
  
load $tylenol # carrega o fármaco  
label %e  
color shapely # coloração única
```



| Aqui vale uma observação sobre a etiquetagem dos átomos (nomes de cada no ecrã). Veja que não há comentário seguindo a instrução. Isso é necessário para que o comentário não seja a própria etiquetagem, e sim o símbolo do elemento.

```
# Para selecionar e colorir os átomos de H

load $cyclopropane # carrega o modelo
select _H # seleciona os átomos de hidrogênio
label %e
color AlcianBlue
```



[close](#) [help](#)

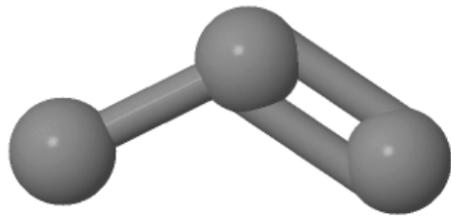
```
superfície de interação com água;
C3H6
6 atoms selected
6 atoms hidden
C3H6
6 atoms selected
script ERROR: a color or palette name (Jmol, Rasmc
-----
color AlcianBlue
C3H6
6 atoms selected
C3H6
6 atoms selected
```

```
# Para selecionar e colorir os átomos de H
load $cyclopropane # carrega o modelo
select _H # seleciona os átomos de hidrogénio
label %
color darkorange
```

[Run](#) [Load](#) [Clear Input](#) [Clear Output](#)

```
# Para selecionar e esconder os átomos de H de um propeno
```

```
load $propene # carrega o modelo
select _H # seleciona os átomos de H
hide selected # esconde os átomos de H selecionados
```



```

close help
isosurface resolution for axis 1 set to 3.8819875776397517
voxels
isosurface resolution for axis 2 set to 3.8819875776397517
voxels
isosurface resolution for axis 3 set to 3.8819875776397517
voxels
17576 voxels total
1 id:isosurface1; model:1.1; vertices:980; polygons:1854;
dataRange:[0.0, 0.0, 0.0, 0.0]; title:isosurface sasurface
superfície de interação com água;
C3H6
6 atoms selected
6 atoms hidden

```

```

# Para selecionar e esconder os átomos de H de um propeno
load $propene # carrega o modelo
select H # seleciona os átomos de H
hide selected # esconde os átomos de H selecionados

```

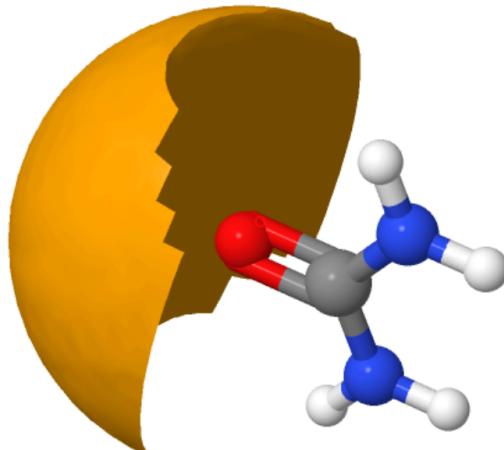
Run Load Clear Input Clear Output

```

# Para selecionar um átomo específico no modelo e apresentar sua área acessível ao solvente

load $urea
select atomno=2 # seleciona o carbono
label %
isosurface sasurface # apresenta a superfície de interação com água

```



```

close help
solvent/molecular surface
solvent-accessible surface with radius 1.2

isosurface resolution for axis 1 set to 3.8819875776397517 points/Ang
voxels
isosurface resolution for axis 2 set to 3.8819875776397517 points/Ang
voxels
isosurface resolution for axis 3 set to 3.8819875776397517 points/Ang
voxels
17576 voxels total
1 id:isosurface1; model:1.1; vertices:980; polygons:1854; visible:true;
dataRange:[0.0, 0.0, 0.0, 0.0]; title:isosurface sasurface # apresenta
superfície de interação com água;

```

```

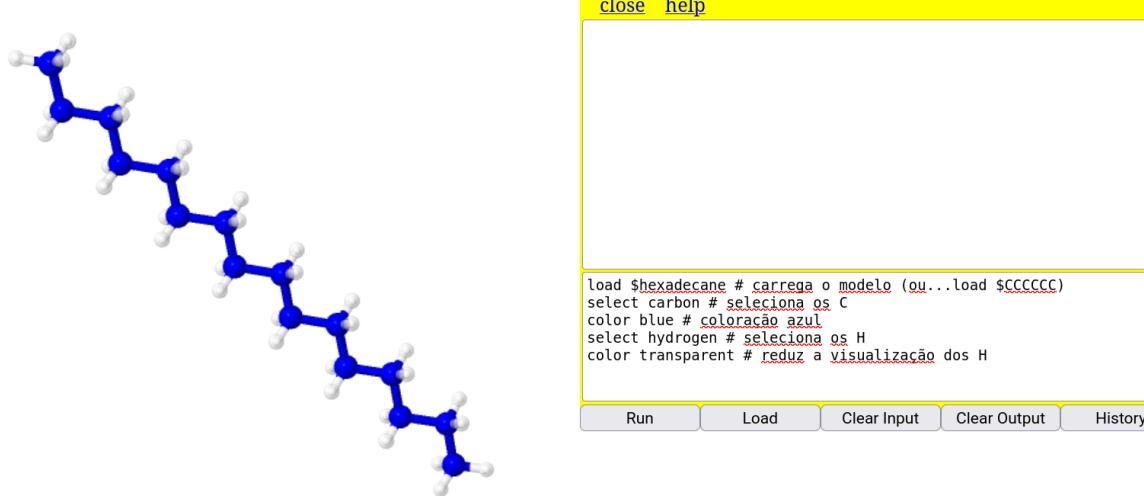
load $urea
select atomno=3 # seleciona o carbono
label %
isosurface sasurface # apresenta a superfície de interação com água

```

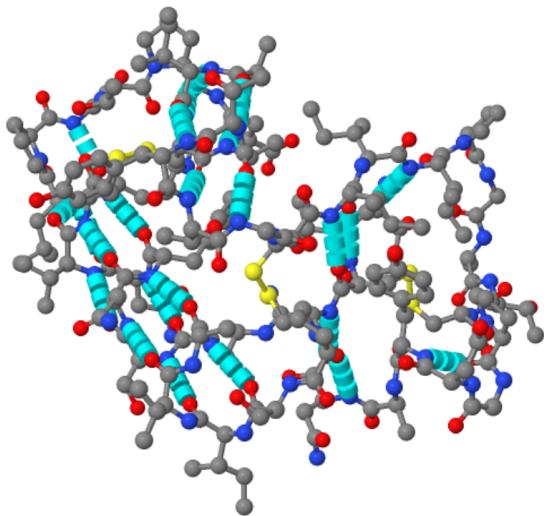
Run Load Clear Input Clear Output History

```
# Para ressaltar os carbonos de um hexadecano

load $hexadecane # carrega o modelo (ou...load $CCCCCC)
select carbon # seleciona os C
color blue # coloração azul
select hydrogen # seleciona os H
color transparent # reduz a visualização dos H
```



```
load=1crn # carrega um modelo da proteína crambina
calculate hbonds # apresenta as ligações de H presentes na estrutura
color hbonds Cyan # coloração
hbonds 0.5 # espessura da ligação de H
```



close help  
 C3H6  
 PLANT PROTEIN 30-APR-81 ICRN  
 WATER STRUCTURE OF A HYDROPHOBIC PROTEIN AT ATOMIC RESOLUTION.  
 PENTAGON RINGS OF WATER MOLECULES IN CRYSTALS OF CRAMBIN  
 found biomolecule 1: A  
 ICRN  
 24 hydrogen bonds

```

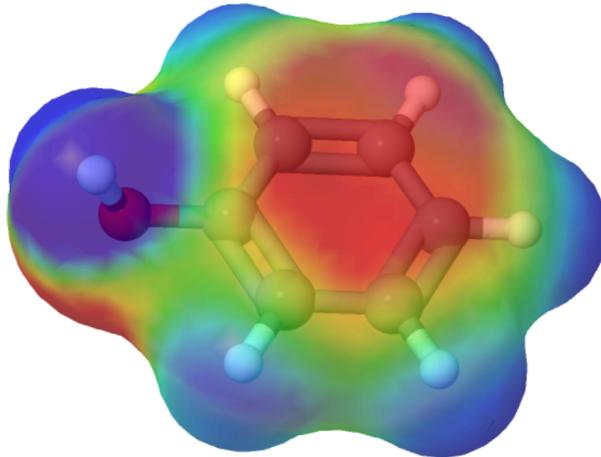
load=lcrn # carrega um modelo de proteína
calculate hbonds # apresenta as ligações de H presentes na estrutura
color hbonds Cyan # coloração
hbonds 0.5 # espessura da ligação de H
  
```

Run Load Clear Input Clear Output History

# Superfície de potencial eletrostático para o fenol

```

load $phenol # carrega o modelo
isosurface solvent color range -0.05 0.05 map mep # gera a superfície
color isosurface translucent 0.5 # mantém a estrutura no interior
  
```



close help  
 solvent/molecular surface  
 solvent-excluded surface with radius 1.2  
 isosurface resolution for axis 1 set to 3.96600566572238 points/Angstr voxels  
 isosurface resolution for axis 2 set to 3.8802734923799456 points/Angst voxels  
 isosurface resolution for axis 3 set to 3.865813324169924 points/Angstr voxels  
 20520 voxels total  
 1 id:isosurface1; model:1.1; vertices:2474; polygons:4944; visible:true  
 dataRange:[-0.15458149293193676, 0.16560078353565594, -0.05, 0.05];  
 title:isosurface solvent color range -0.05 0.05 map mep;

```

# Superfície de potencial eletrostático para o fenol
load $phenol # carrega o modelo
isosurface solvent color range -0.05 0.05 map mep # gera a superfície
color isosurface translucent 0.5 # mantém a estrutura no interior
  
```

Run Load Clear Input Clear Output History

# 7 Animando a molécula

## 7.1 Alguns comandos de animação para os modelos

Animações permitem a representação de moléculas de modo mais lúdico e representativo. Para agregar valor ainda maior à visualização tridimensional de estruturas moleculares, o *Jmol* conta com alguns comandos para o posicionamento espacial do modelo, bem como para a criação de animações, como:

## 7.2 Rotacionar e transladar

Essas ações constituem uma forma simples de reposicionar o modelo no espaço. Basta inserir o eixo e o ângulo em que se deseja o movimento ( $x,y,z$ ). Ou apenas o ângulo, com o movimento padrão no eixo das abscissas ( $X$ ). Seguem exemplos:

```
rotate 20 # 20 graus
rotate x 90 # eixo x
translate y 50 # valor representa o percentual da janela (100 - fora; 0 - centro)

Obs: para retornar à posição original: `reset`
```

## 7.3 Girar

Essa ação, por sua vez, é mais “impactante”, já que permite a rotação do modelo a uma certa velocidade indefinidamente, até um comando de parada. Como na anterior, pode-se digitar somente o comando para a rotação padrão no eixo das abcissas, ou fornecer o eixo ( $x,y,z$ ):

```
spin 10 # rotacional, com velocidade de 10 graus por quadro
spin z -15 # (eixo z)
spin off # interrompe a rotação
```

## 7.4 Ampliação dos modelos (*zoom*)

É possível também ampliar ou reduzir o tamanho de uma molécula no ecrã combinando-se o comando `zoom` à quantidades, tal como segue:

```
Ampliação 2x: zoom in
Ampliação em 3x: zoom *3
Redução em 2x: zoom out
Redução em 3x: zoom /3
Eliminar ampliação: zoom off
Restrição a um ligante e ampliação: restrict ligand; zoom 0

Obs: para retornar ao tamanho original: `zoom 0`
```

Com ampliações, reduções, rotações, translações, é possível que você queira retornar ao modelo na sua configuração de tamanho e coordenadas originais. Para isto:

```
center; zoom 0 # centraliza e redimensiona o modelo a seu tamanho original
```

### 7.4.1 Ampliação animada (*zoomTo*)

Esse recurso é bem mais “*chique*”, já que permite visualizar de forma ampliada temporalmente algumas partes de interesse do modelo. Isso é particularmente bacana com biomacromoléculas, como para sítios de interação de ligantes ou grupos prostéticos, ou sítios de reação em catálise enzimática. Mas também impacta a visualização de pequenas moléculas, posto que os modelos crescem ou reduzem de forma animada no ecrã; pode-se também focar num átomo específico para chamar a atenção na molécula. A sintaxe da expressão é:

```
zoomto tempo tamanho #...ou, se quiser um grupo de átomos em especial (ligante, por ex)...
zoomto tempo (expressão do átomo/grupo) tamanho
```

Experimente carregar moléculas pequenas e proteínas, ilustrando o comando `zoomTo` como segue:

```
Aumentar em 3x, meio segundo por vez: zoomto 0.5 *3
Aumentar em 4x, meio segundo por vez: zoomto 0.5 400
Focar num ligante com ampliação de 2x: zoomto 2(ligand) 0
Focar num ligante com ampliação de 4x, a meio segundo por vez: zoomto 0.5(ligand)* 4
```

## 7.5 Pausa

Um comando muito interessante para o *Jmol*, bem como para outras linguagens de programação, é `delay`. Essa ação permite que o *script* interrompa uma ação por algum tempo, medido em segundos, e limitado ao mínimo de um segundo. Experimente:

```
load $ribose # carrega uma ribose
spin 50 # rotaciona no eixo X
delay 5 # aguarda 5s
spin off # interrompe a rotação
wireframe only # represeta em arame, somente
wireframe 130 # espessura do arame
color translucent # cor translúcida
```

## 7.6 Laço

O *Jmol* possui alguns comandos normalmente utilizados em linguagens de programação, embora não seja esse o escopo do presente trabalho. Se quiser “*ir fundo*”, consulte o [manual de referência de comandos do Jmol](#) para os comandos de programação: *if*, *ifelse*, *for*, *while*, *step*, *break*, *wait*, *pause*, *case*, *continue*, *quit*, *loop*.

Mas entre esses comandos variados, o de *laço iterativo* (*loop*) pode ser de valia ao ensino, por exemplo, para quando se desejar chamar à atenção para uma ligação, um átomo ou grupo de átomos. Exemplificando:

```
load $acetate # carrega um modelo de acetato
color bonds red # coloração vermelha às ligações
delay 1 # aguarda 1 segundo
color bonds green # troca a coloração para verde
loop 1 # realiza o laço (efeito "piscante")
```

Para deixar o laço, ou seja, interromper a sequência em *loop*, digita-se *quit*.

## 7.7 Combinando rotações, translações, e ampliações no tempo

O *Jmol* oferece dois comandos básicos para isso: `move` e `moveTo`. A diferença é que `move` reorienta o modelo em relação à sua posição atual (*movimento relativo*), e `moveTo` o faz em relação à posição original do modelo (*movimento absoluto*).

Na prática, contudo, o comando `move` é mais simples de executar, dependendo somente de parâmetros de rotação e translação, ao passo que `moveTo` é bem mais complexo. Esse depende de parâmetros de orientação manualmente obtidos pelo *Jmol*.

### 7.7.1 Move

Para o comando `move` bastam os parâmetros que se deseja modificar (*nove* no total). Dependendo do que se deseja não há necessidade de usar todos os parâmetros, e que são fornecidos abaixo:

```
move rotX rotY rotZ zoom dx dy dz slab time
```

Onde:

`rotX, Y ou Z` = rotação no eixo X, Y, ou Z  
`zoom` = ampliação ou redução  
`dx, Y ou Z` = translação no eixo X, Y ou Z  
`slab` = plano de fatiamento da molécula  
`time` = tempo total envolvido no movimento

Seguem alguns exemplos:

```
move 90 180 0 0 0 0 0 21 # rotaciona o modelo por 45 graus em torno do eixo X e por 180 graus em torno do eixo Y
move 0 0 0 0 35 0 0 0.5 # reduz o modelo em 35% de seu tamanho, e com movimento gradual de 0.5 segundos
move 45 0 90 150 0 30 0 0 5 # rotaciona 45 graus no eixo X e 90 graus no eixo Z, ampliando o zoom em 150%
```

### 7.7.2 MoveTo

Resulta em orientação *absoluta* do modelo, não dependendo de suas coordenadas anteriores. Sua inserção não é simples, pois depende dos dados da orientação do modelo quando carregado pela 1a. vez, e que pode ser obtida por:

```
show orientation
```

Um exemplo de resultado pelo comando é:

```
moveto /* time, axisAngle */ 1.0 { 616 -708 -346 47.68} /* zoom, translation */ 400.0 0.0 0.0
#OR
#Follows Z-Y-Z convention for Euler angles
reset;center {15.174467 28.719118 4.726837}; rotate z 130.27; rotate y 44.57; rotate z -147.0
```

Perceba que há dois conjuntos de comandos um pelo *moveTo* e outro por *reset*, *center*, *rotate* e *translate*. Para obter o modelo nas coordenadas originais, basta copiar um ou outro conjunto de dados.

Usando-se o 1o. conjunto (*moveTo*), copia-se a linha e altera-se o tempo de animação, no caso o valor 1.0 em “axisAngle \*/ 1.0”.

## 7.8 Comparando dois modelos no ecrã

Compara 2 modelos e reorienta as coordenadas do segundo para justapor-se ao primeiro, por um algoritmo de correlação.

```
load files "$tyrosine" "$epinephrine";
frame *;
compare {2.1} {1.1} rotate translate 5.0
```

## 7.9 Navigate

O comando permite explorar o modelo simulando um passeio panorâmico ao interior da estrutura. Os parâmetros envolvem o tempo de percurso (ou 2s quando omitido). Exemplificando:

```
navigate depth 50 # imersão no modelo em 2s
navigate 3 rotate y 20 # rotaciona 200 no eixo y
navigate 4 trace # passeia pelo modelo em 4s
navigate 3 translate {30 50 70} modelo translada levemente por 3s
navigate 5 center {10 20 30} # sonda ao lado do modelo, e nas coordenadas x, y, z
navigate 2 depth 30 / 5 rotate 180 / depth 20 / translate X 10
```

## 7.10 Exemplos de animações para moléculas no *Jmol*

## **Parte II**

# **PARTE 2 - R & RStudio**



# 8 Como usar o R & Rstudio - instalação e nuvem

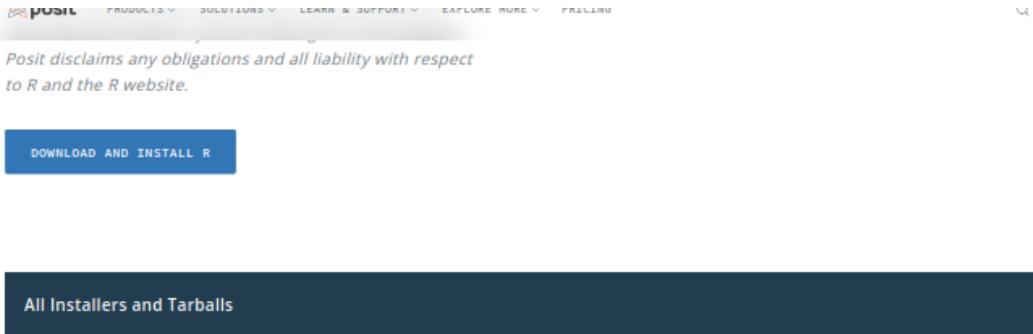
Existem dois ambientes alternativos para se utilizar o *R* (programa) e o *Rstudio* (interface do usuário): instalando no computador, ou pelas nuvens. Há poucas diferenças entre ambas as maneiras de se trabalhar com os programas, mas a fundamental é que a *instalação* pode ser utilizada *offline*, sem necessidade de internet, enquanto que pelas nuvens, bom, já sabe.

## 8.1 Instalando o R e RStudio no computador

Você precisa seguir uns poucos passos para instalar o *R* & *RStudio* no computador. Na prática, baixa-se ambos os programas e os instala como se faria com qualquer outro programa, tanto faz se para *Windows*, *Linux*, ou *Mac*. Seguem os passos:

The screenshot shows the Posit website with the URL <https://www.posit.com/r/download/>. The page has a header with the Posit logo and navigation links for PRODUCTS, SOLUTIONS, LEARN & SUPPORT, and EXPLORE MORE. The main content area is titled "1: Install R" and contains the following text:  
RStudio requires R 3.6.0+. Choose a version of R that matches your computer's operating system.  
R is not a Posit product. By clicking on the link below to download and install R, you are leaving the Posit website. Posit disclaims any obligations and all liability with respect to R and the R website.  
A blue button at the bottom right of the text block says "DOWNLOAD AND INSTALL R".

1. Acesse o site do [Rstudio](#) e faça o *download* do programa *R*.
2. Instale o programa com as opções padrão.
3. No mesmo site do [RStudio](#) procure um pouco mais abaixo pelo instalador mais apropriado a seu sistema operacional.



RStudio requires a 64-bit operating system.

Linux users may need to import [Posit's public code-signing key](#) prior to installation, depending on the operating system's security policy.

OS	Download	Size	SHA-256
Windows 10/11	<a href="#">RSTUDIO-2024.04.2-764.EXE</a>	262.79 MB	<a href="#">09E1E38A</a>
macOS 12+	<a href="#">RSTUDIO-2024.04.2-764.DMG</a>	664.40 MB	<a href="#">D8DD0395</a>
Ubuntu 20/Debian 11	<a href="#">RSTUDIO-2024.04.2-764-AMD64.DEB</a>	194.73 MB	<a href="#">87B20155</a>

3. Baixe o arquivo e instale-o como qualquer outro programa.
4. Abra o programa *RStudio*, e cuja interface será parecida com a que segue.

## 8.2 Acessando o R & Rstudio pelas nuvens

Essa é uma opção simples e que não requer qualquer instalação. A interface acessada é praticamente igual à da instalação em computador. Entre algumas vantagens destaca-se a velocidade normalmente superior pra rodar e instalar pacotes, posto que o servidor para esses já está no provedor em nuvem. Mas por ser acesso *online*, requer uma inscrição inicial, com *login* e *senha*. Seguem os passos:

1. Acesse o site do [RStudio Cloud](#).

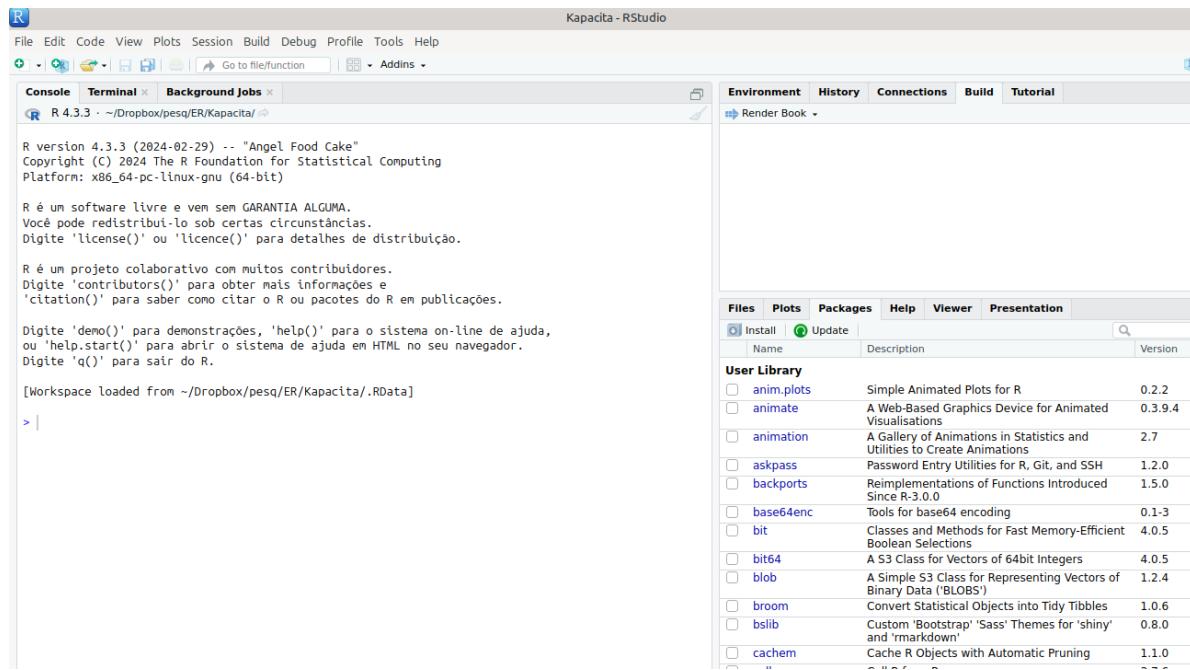
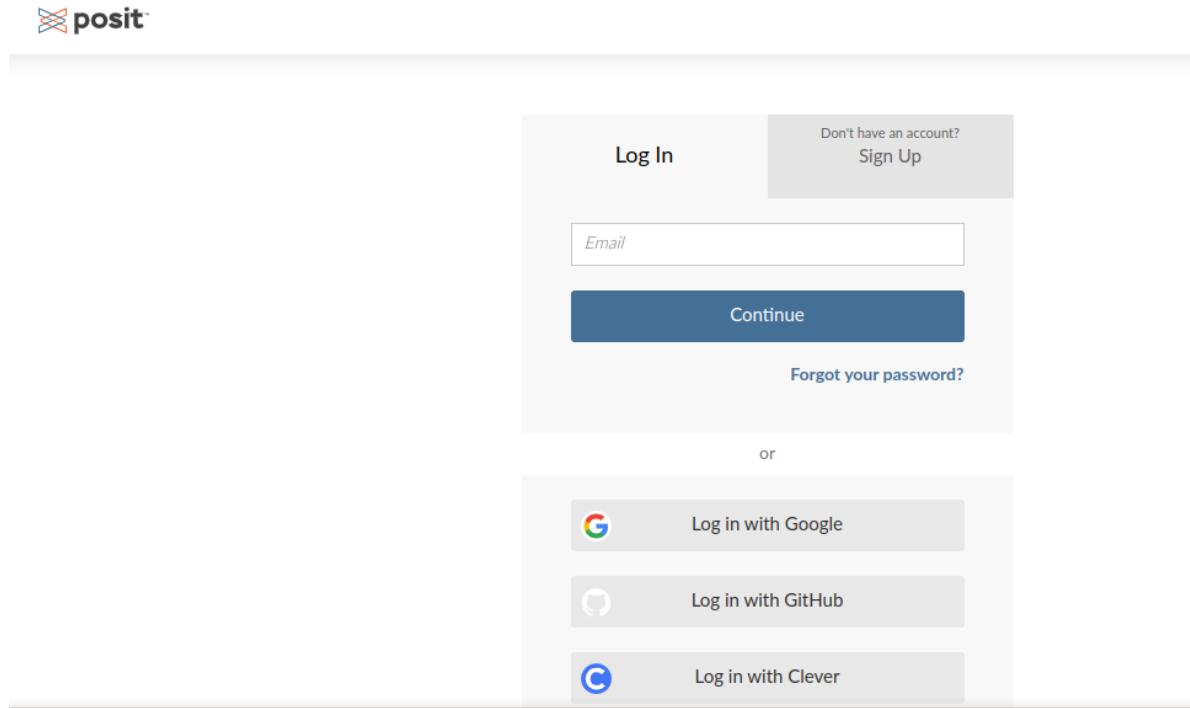


Figura 8.1



2. Realize a inscrição (*sign up*) ou acesse pelo *Google* (mais simples).
3. A janela deverá parecer-se com a que segue, embora sem os projetos listados.

The screenshot shows the 'Your Content' section of the Posit Cloud interface. At the top, there are tabs for 'Content', 'Usage', and 'About'. On the right, a user profile for 'José Maurício Schneedorf Ferreira da Silva' is displayed. Below the tabs, there are filters for 'TYPE', 'ACCESS', and 'SORT', along with a search bar. The main area lists five projects:

- erre**: RStudio Project, Private, Created Jul 9, 2024 4:01 PM
- inova-jmol**: RStudio Project, Private, Created Jun 18, 2024 8:35 AM
- inova-r**: RStudio Project, Private, Created Jun 20, 2024 3:39 PM
- new.erre**: RStudio Project, Private, Created Jul 9, 2024 4:58 PM
- webquarto**

Each project entry includes standard file operations icons: copy, delete, move, download, and more.

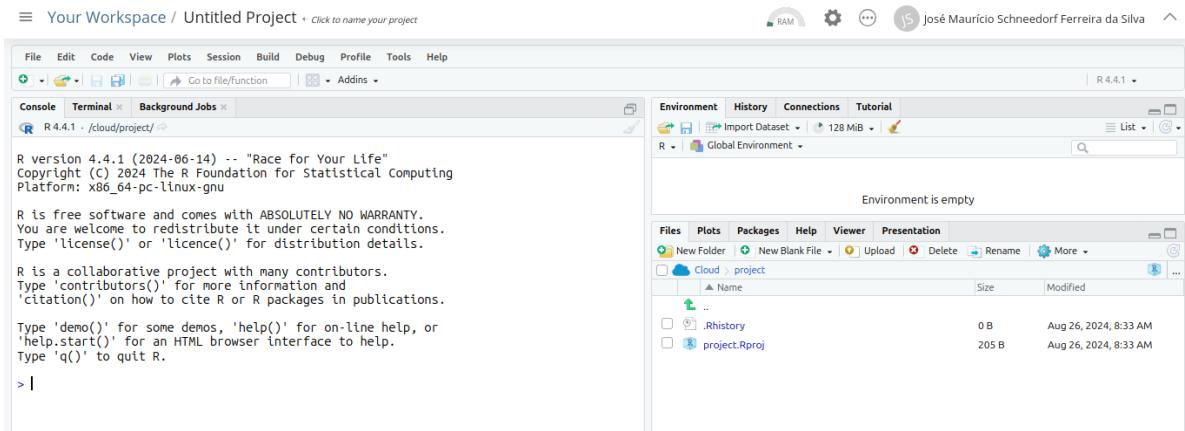
4. Agora a parte interessante. Clique em *New Project* e selecione *New Rstudio Project*.

The screenshot shows a Mozilla Firefox browser window with multiple tabs open. The active tab is 'Your Content - Your Workspace - Posit Cloud — Mozilla Firefox'. In the top right corner of the browser, a 'New Project' button is visible. A dropdown menu is open, listing three options:

- New Project from Template**
- New RStudio Project** (selected)
- New Jupyter Project**
- New Project from Git Repository**

The browser's address bar shows the URL [https://posit.cloud/content/yourself?sort=name\\_asc](https://posit.cloud/content/yourself?sort=name_asc). The bottom of the screen shows the Mac OS X dock with various application icons.

5. A imagem final será bem parecida com a apresentada pela versão instalada, como visto na Figura ??, veja:



| Pronto ! Você pode utilizar uma ou outra forma para acessar as atividades e objetos didáticos propostos neste material. A seguir serão fornecidas algumas ações e comandos rápidos para se trabalhar com a interface do Rstudio e com o R. Mas apenas para que se consiga criar, executar, e modificar alguns *scripts* criados para animações, simulações, interatividade, e geovisualização. Mão na massa, agora !

Por uma questão de simplicidade para o uso da versão em servidor, sem necessidade de instalação dos programas, bem como por maior velocidade de execução de *scripts* e instalação de pacotes, optamos neste trabalho por exemplificar os *objetos didáticos* em nuvem, ou seja, por uso do [RStudio Cloud](#).

# 9 Comandos básicos & Scripts no R

O R é um programa que opera por linha de comando. Isso é um pouco chato, como já visto, porque qualquer erro na digitação de um comando resulta na interrupção do código. Mas, por outro lado, e também como já visto, *linhas de comando encadeadas e comentadas permitem a reprodução e modificação de trechos de códigos convergentes a um produto* qualquer, no caso, objetos didáticos ao ensino médio.

Diferente do *Jmol*, contudo, não é possível mesclar fonte maiúsculas ou minúsculas, bem como singular ou plural. Para que o código funcione, é necessário sua correta digitação. Mas *pode-se tranquilamente aumentar ou reduzir o espaço entre comandos*, o que não faz diferença pro compilador do R.

Algumas operações são realizadas alternativamente por *mouse*, *linha de comando*, ou ambos, dependendo da ação. A seguir serão apresentadas algumas funcionalidades básicas para a reprodução de códigos para objetos didáticos, sem descrições detalhadas da operação própria do R & RStudio, para simplificar e tornar mais objetivo este trabalho. Se você desejar saber mais a respeito de ambos os programas, versão instalada ou em nuvem, sugerimos os inúmeros sites e tutoriais disponíveis na internet, bem como centenas de livros já escritos no assunto, e cursos *on-line* em várias plataformas de ensino.

## 9.1 Uma visão da interface RStudio

| O Rstudio nada mais faz do que permitir uma *interface gráfica para o usuário* do R (ou *GUI*, do inglês) , esse um programa estritamente rodado por uso de códigos. Diversas operações podem então realizar-se sem comandos ou códigos, como abrir e salvar um arquivo, ou visualizar um gráfico, por exemplo. Vejamos a divisão da janela principal do Rstudio.

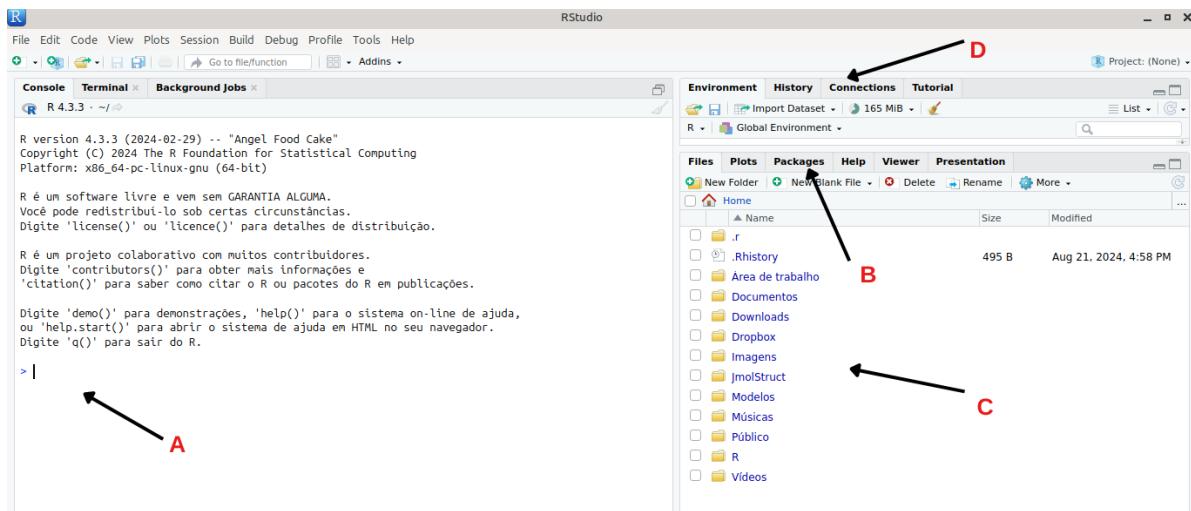
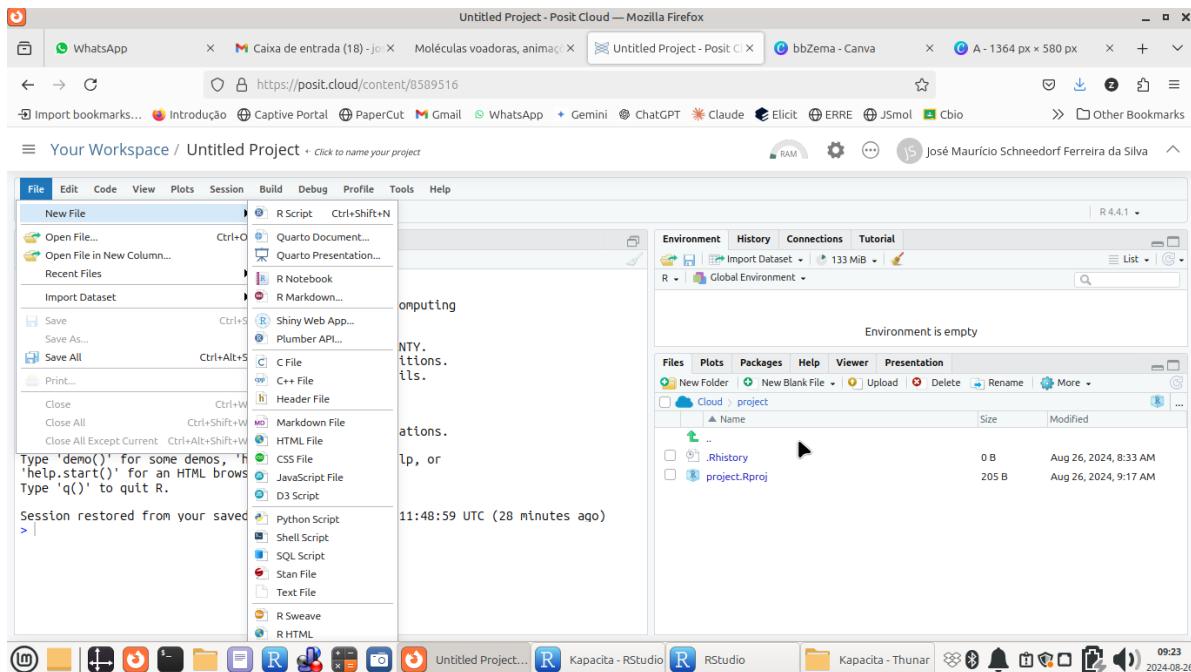


Figura 9.1: Janela básica do RStudio. A - área de digitação de comandos (*prompt*); B - área de abas de trabalho (diretório, gráficos, pacotes, etc); C - arquivos que aparecem na aba homônima; D - área de abas de administração (ambiente, história de comandos, etc).

Para nosso trabalho, contudo, será interessante uma área adicional, a *área de scripts*, a qual se acessa como segue:

```
File --> New File --> RScript
... ou por atalho:Ctrl + Shift + N
```



Veja que agora a janela principal se divide em quatro partes, incluindo a aba nova para *scripts*.

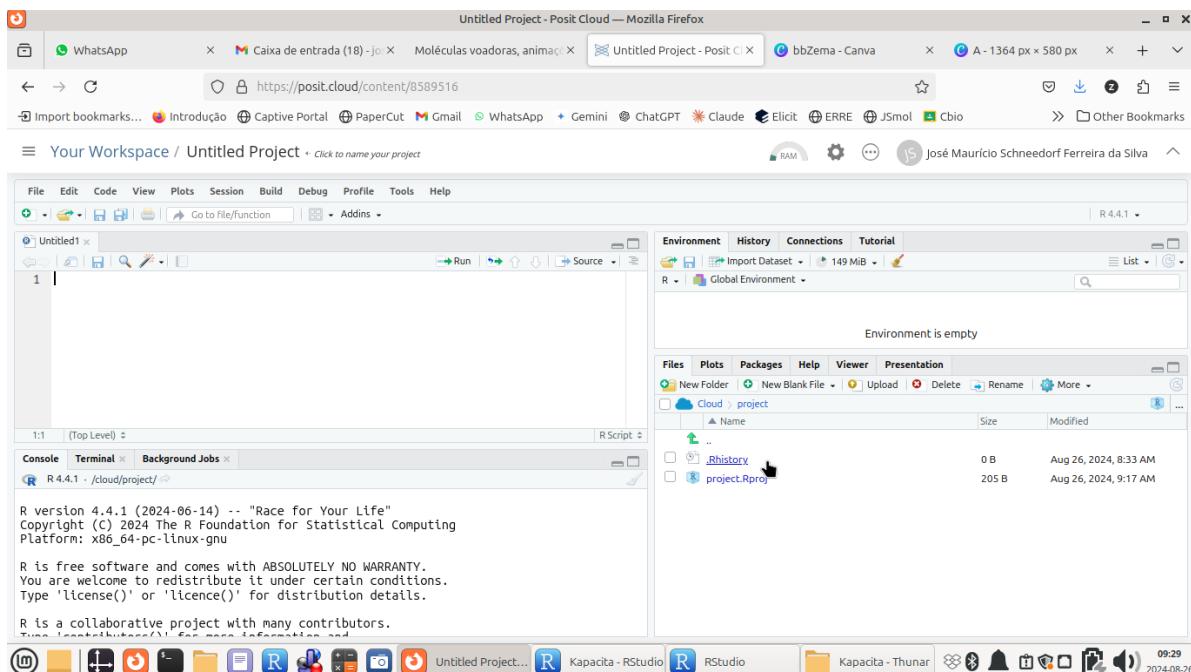


Figura 9.2: Janela principal do *RStudio* contendo a aba para elaboração e execução de *scripts*.

## 9.2 Como funcionam os comando no R

Todos os comandos do R são compostos por um *nome* seguido de *argumentos* entre parênteses. **Não é necessário a especificação de todos os argumentos;** normalmente bastam de 1 a 3. Seguem exemplos

```
comando(argumento 1, argumento 2, argumento 3, ...)
```

Exemplos:

```
plot(x,y)
mean(z)
read.csv(file = "meus.dados.csv")
```

Para se conhecer os argumentos de um comando, basta você começar a digitá-lo, tanto na *área de prompt* (canto inferior esquerdo) como na *área de script*, que o sistema apresenta temporariamente as opções. Se desejar visualizar essas opções por linha de comando, contudo, digite função `args` seguido do comando desejado, como segue:

```
args(plot)
```

## 9.3 Elaborando e executando um *script* no R

Para se produzir um *script* no R, basta redigir as linhas de comando de modo similar ao que foi realizado com o visualizador molecular 3D *Jmol*, ou seja, *separando os comandos por ponto e vírgula*, ou por linhas individuais, usado a tecla *Enter*:

```
x = 5
x^2 +7
```

E para executar o *script* acima, basta copiá-lo e colá-lo na área de *script* aberta. E aí vai uma **dica de ouro**. Veja que no canto superior direito do *script* existe um ícone de *colagem* do texto do *script*. Basta clicar nesse ícone que o texto estará copiado.

Agora é só colar na aba do *script* aberto (em nuvem, por exemplo) e executá-lo seguindo-se *alternativamente* as seguintes ações:

1. Se deseja executar algumas linhas de um \*script\*, pode-se selecionar as linhas e clicar **C** (Copy);
2. Se desejar executar todo o \*script\*, seleciona-se todo o texto (**Ctrl + A**) seguido da ação **Ctrl + C** (Copy);
3. Abre-se o *script* que se deseja executar e coloca-se o cursor no local desejado (por exemplo, na nuvem);
4. Clica-se no ícone de *colagem* (Paste);
5. Clica-se em **Enter**.

- 3.** Se desejar executar apenas uma linha, basta clicar na linha seguido de Ctrl + Enter ;  
Opcionalmente, pode-se clicar no ícone "-->Run" ;

## 9.4 Algumas recomendações sobre a digitação num *script* do R:

Existem algumas condições básicas pra que um *script* do R seja lido de forma clara por seu elaborador, bem como compilado corretamente pelo programa:

1. *Digitação*: sempre que tem um erro no *script* no *Rstudio*, surge um sinal em vermelho ao lado esquerdo da linha de comando; contornado o erro, o sinal desaparece;
2. *Comentários*: para que o *script* seja lido também por “*um ser humano*”, é aconselhável tecer comentários nas linhas de comando (iniciados por #);
3. *Indentação*: permita “*indentação*” quando a linha estiver um pouco longa, clicando na tecla *Enter* após uma separação de argumentos por “vírgula”. Dessa forma, a linha continua logo abaixo, mas com um pequeno deslocamento à direita. Isso facilita a legibilidade do código.
4. *Nomes*: os comandos do R são em língua inglesa. Dessa forma, deve-se evitar o uso de variáveis e nomes de arquivos com acentuação ou sinais gráficos do Português (ex: ç). Além disso, o R é um compilador de códigos. Se você definir um nome composto para um arquivo ou variável, ou seja, com espaço entre os termos (como é normal no cotidiano), o R tentará executar os termos separadamente, o que levará em erro. Assim, para nomes de variáveis e arquivos, dê preferência a um dos 3 tipos de *convenções comuns usadas em programação*, a saber:
  - separação por *underline*, ”\_” ou hífen; ex: minha\_variável, minha-variável
  - separação por maiúscula; ex: minhaVariável
  - separação por pontos; ex: minha.variável

# 10 Gráficos básicos e simulações

## 10.1 “Plotando” com o comando `plot` (meio óbvio!)

Você não precisa instalar mais nada pra trabalhar com gráficos no R. Isso porque o sistema já possui um conjunto de pacotes na sua instalação, incluindo um pacote para gráficos (`graphics`).

Para fazer seu primeiro gráfico, basta **copiar e colar** o trecho abaixo num *script* aberto no *Rstudio*, tanto faz se instalado ou nas nuvens (*RStudio Cloud*), executando-o em seguida.

```
y = c(1,2,4,8,16,32) # vetor de números  
plot(y) # comando pra "plotar" o gráfico
```

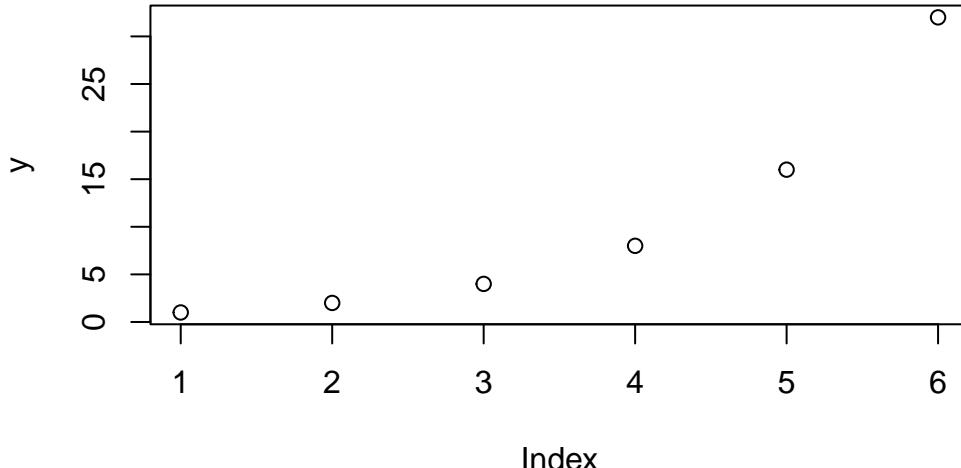


Figura 10.1: Um gráfico simples no R.

Se você obteve como resultado a imagem acima, **parabéns !!** Você fez o seu primeiro gráfico no R !!! Tá achando pouco ?! Você fez um gráfico numa **linguagem orientada a objeto em programa de computação estatística !!!**

**Uma dica importantíssima:** como se pode observar do código acima, “para se colocar valores em sequência, como numa coluna de planilha eletrônica, o R precisa “ler” um vetor iniciado por “c” (de “concatenated”), seguido dos valores entre parênteses () e separados por vírgula !

Agora, se quiser *caprichar um pouco mais*, pode colocar uma expressão sobre uma sequência de números igualmente espaçados. Exemplificando:

```
x = 1:10 # vetor de valores de 0 a 10, com intervalo unitário  
y = x^2 # expressão aplicada ao vetor "x"  
plot(x,y) # gráfico produzido
```

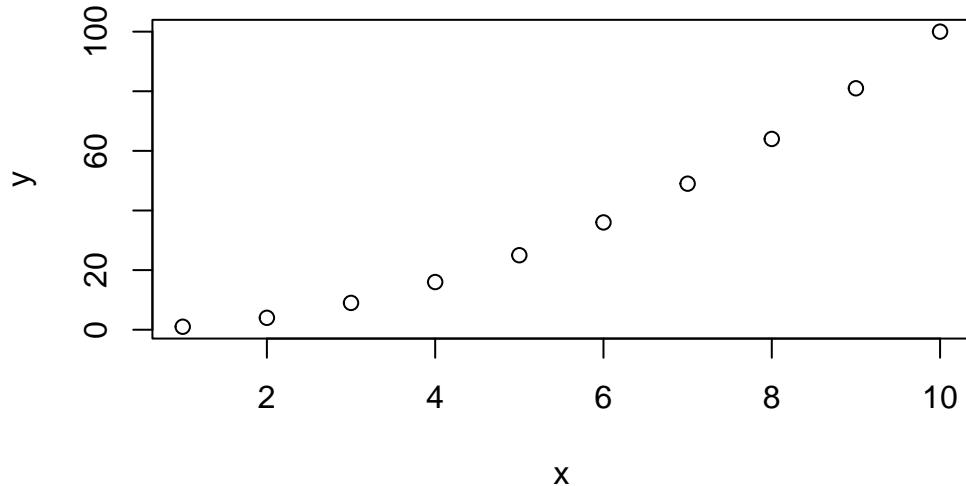


Figura 10.2: Um gráfico de relação quadrática ( $x^2$ ) entre x e y.

Também dá pra elaborar o gráfico sem precisar criar a variável independente y, apenas inserindo sua função em x no próprio comando. Experimente:

```
x= 1:10  
plot(x,x^2)
```

Ou....pode também *escolher os números* que deseja *plotar*:

```
x = c(1,2,5,12,31)
y= c(4,-5,12,47,-2)
plot(x,y)
```

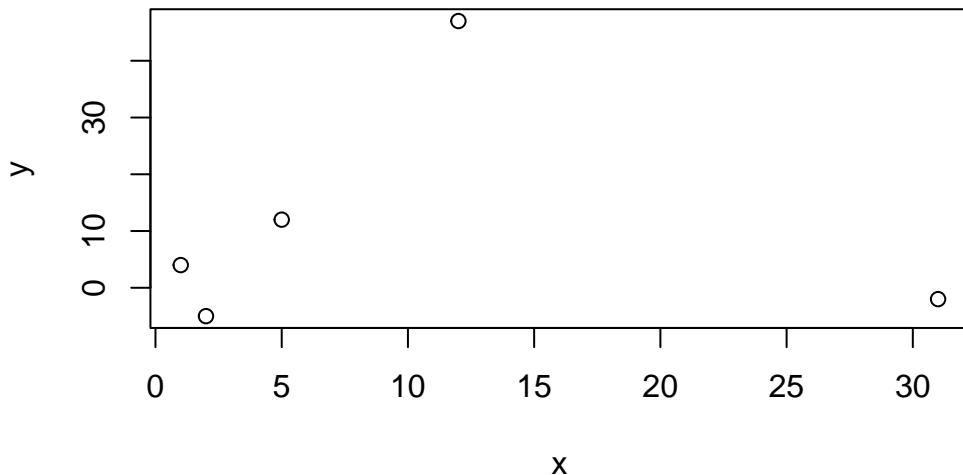


Figura 10.3: Gráfico de valores aleatórios de x e y.

## 10.2 Incrementando um pouquinho os gráficos

Existe uma quantidade imensa de tutoriais na *web*, além de uma vasta literatura sobre o uso do pacote `basic` para se construir gráficos. Mas a proposta aqui é que você aprenda apenas o fundamental pra elaborar um gráfico, pois o *objetivo é animá-lo, e não complicá-lo !!*

Mesmo assim, alguns *argumentos* da função `plot` podem ilustrar o potencial de seu uso. Como explicado anteriormente, ainda que as funções do R possuam diversos argumentos, você pode executá-la com poucos ou apenas um, somente.

Os argumentos da função `plot` são os descritos abaixo:

```
plot(x, y = NULL, type = "p", xlim = NULL, ylim = NULL,  
log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,  
ann = par("ann"), axes = TRUE, frame.plot = axes,  
panel.first = NULL, panel.last = NULL, asp = NA,  
xgap.axis = NA, ygap.axis = NA,  
...)
```

Um pouco confuso, é?! Explicando alguns dos argumentos acima, e outros que funcionam com a função `plot`:

```
type - tipo de plot: pontos "p", linhas "l", pontos+linhas com cruzamento "o", pontos+linhas  
cex - tamanho do ponto (ex: 0.5, 20);  
  
lty - "line type", tipo da linha; pode se representado por um valor (1,2,...) ou por "solid"  
  
lwd - "line width", largura da linha (6 níveis)  
  
pch - tipo de ponto (1-25)  
  
xlim, ylim - limite dos eixos; ex: ylim=c(-2,10)  
  
xlab, ylab - etiquetas ("label") dos gráficos  
  
col - cor (números ou nomes); ex: "red", "orange"  
  
main - título do gráfico  
  
sub - subtítulo do gráfico  
  
log - eixo logaritmo
```

O gráfico de pontos não é o único que se pode fazer com o pacote `graphics`. Também dá pra fazer tudo isso abaixo, com cada função apresentando os seus próprios argumentos:

```
plot() - pontos, linhas, pontos e linhas  
barplot() - plot de barras  
hist() - histograma  
boxplot() - gráfico Box-Whiskers  
persp() - gráfico 3D  
pie() - gráfico de torta
```

```
dotplot() - sequência de valores (com *jitter*, espalhamento de pontos)
pairs() - painel múltiplo com todas as variáveis plotadas
matplot() - plota vetores de matrizes, como *dataframes*
```

Mas chega de enrolação !! Para sentir de perto do “*poder*” desse pacote básico para gráficos, veja a imagem que segue retirada do [MAPA, 1o. Bimestre de Ciências da Natureza e Suas Tecnologias](#):

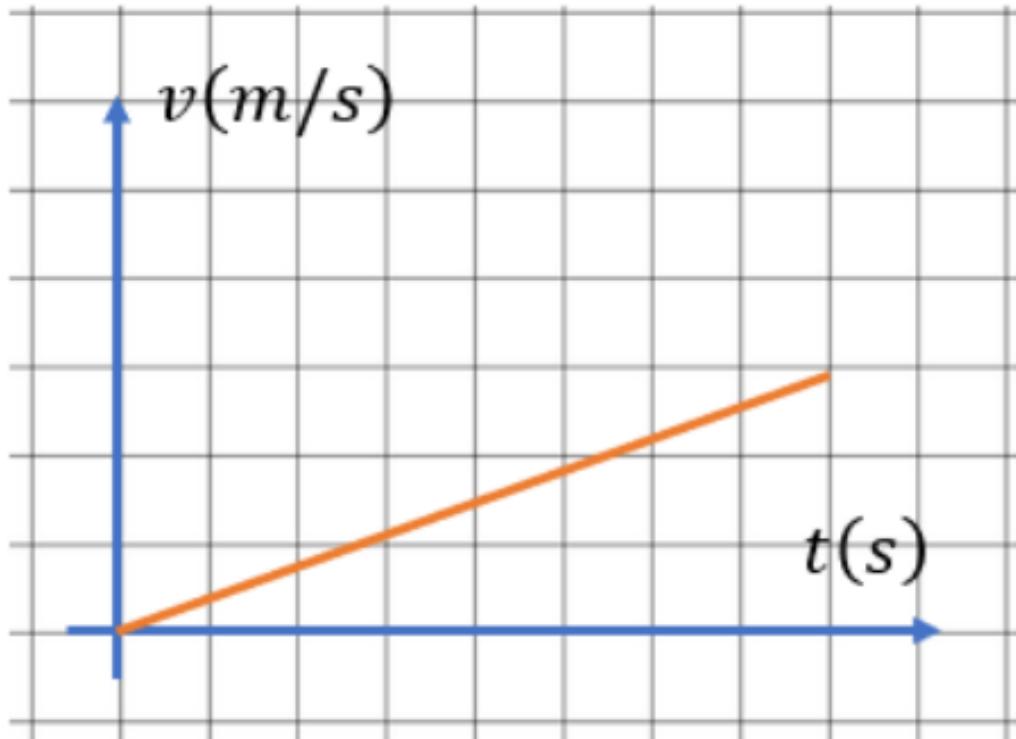


Figura 10.4

Agora experimente reproduzir a imagem do *MAPA* copiando, colando e executando o trecho abaixo numa janela de *script* do RStudio:

```
t <- c(0,1,2,3,4) # introduz os dados
v <- c(0,5,10,15,20)
plot(t,v) # faz o gráfico
```

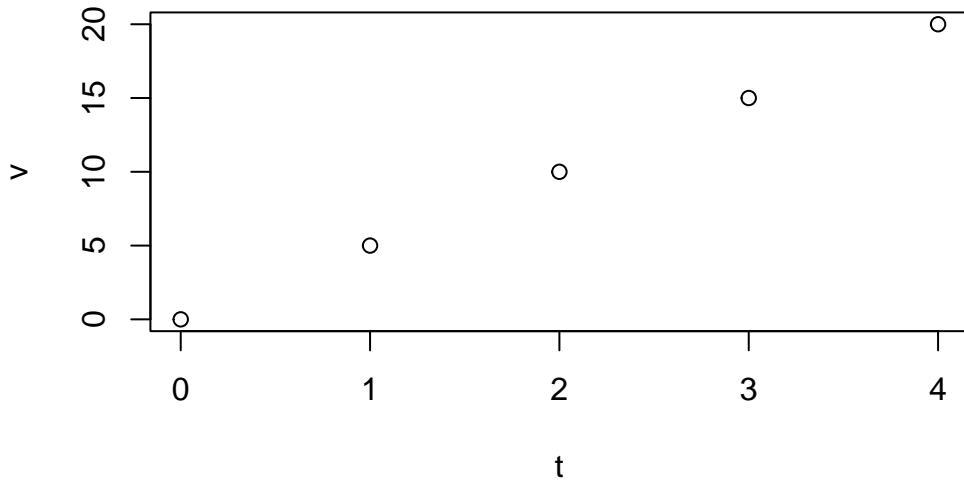


Figura 10.5: Um gráfico simples pra reprodução de figura do MAPA.

Agora, dá pra incrementar um pouco mais, para aproximar-se da imagem da figura de referência do *MAPA*. Segue o trecho pra cópia e execução:

```
t <- c(0,1,2,3,4) # introduz os dados
v <- c(0,5,10,15,20)
plot(t,v,main="Movimento Uniforme: Aceleração", # realiza o gráfico com título...
      xlab="t(s)", ylab="v(/s)", # ... etiquetas nos eixos....
      type="l", col="red") # ... tipo e cor da linha, e ....
grid() # divisão para facilitar a visualização
```

Dando tudo certo, segue o que se produz com o código:

## Movimento Acelerado

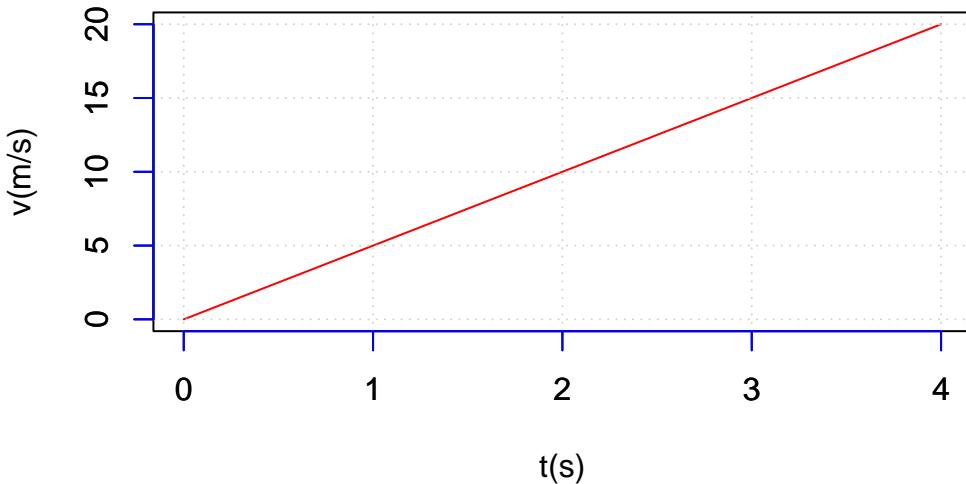


Figura 10.6: Reprodução de gráfico com pacote `basic` do R, tal como apresentado no MAPA - Ciências da Natureza, 1o. Bimestre, pg. 134 (2024).

### 10.3 Simulando curvas com a função `curve`

Uma situação bem comum no aprendizado em *Matemática* e de alguns temas em *Ciências da Natureza* se dá quando desejamos observar como uma variável se comporta em relação a outra, fornecida uma equação para tal. Nesse caso, pode-se utilizar a função `curve` do R, mais simples até em seus argumentos que a função `plot`. Para ilustrar isso, execute separadamente os exemplos abaixo num *script*:

```
curve(x^2) # função quadrática
curve(log10(x), xlim=c(1,1e3), col="blue") # função logarítmica e coloração
curve(x^2-2*x+7, xlim=c(-10,10)) # função quadrática com limites no eixo X
curve(sin(x), from=0, to=90,
      xlab="ângulo(graus)", ylab="seno(ângulo)", col="red", n=1000) # com etiquetas nos eixos
```

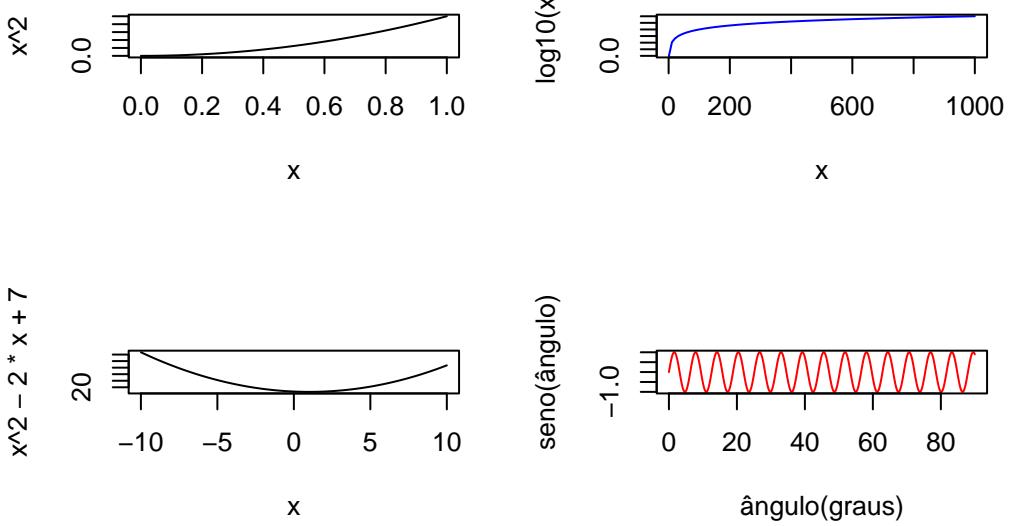


Figura 10.7: Alguns exemplos para a função `curve`.

Ilustrando-se mais objetivamente, pode-se reproduzir (**como também modificar**) o trecho de código que segue logo após a figura abaixo extraída do *MAPA*:

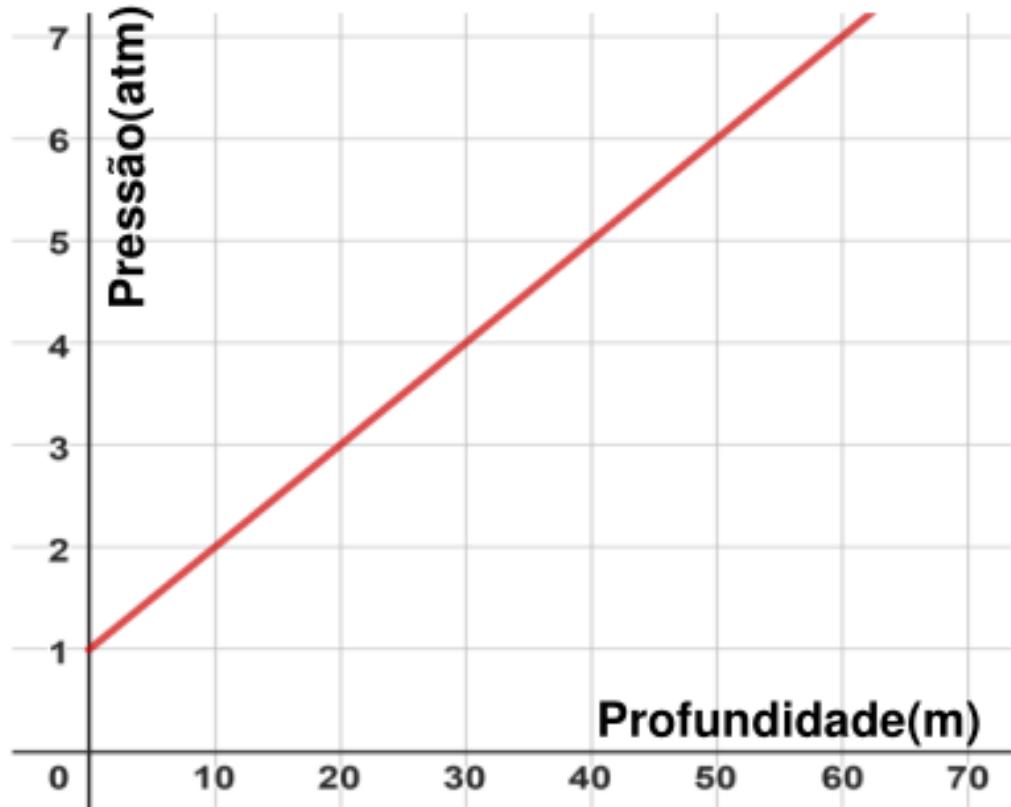


Figura 10.8: Figura extraída de exercício do MAPA, 1o. Bimestre de Matemática e Suas Tecnologias, p.32.

```
x = 0:70  
curve(1/10*x + 1,  
      xlab="profundidade", ylab="pressão")
```

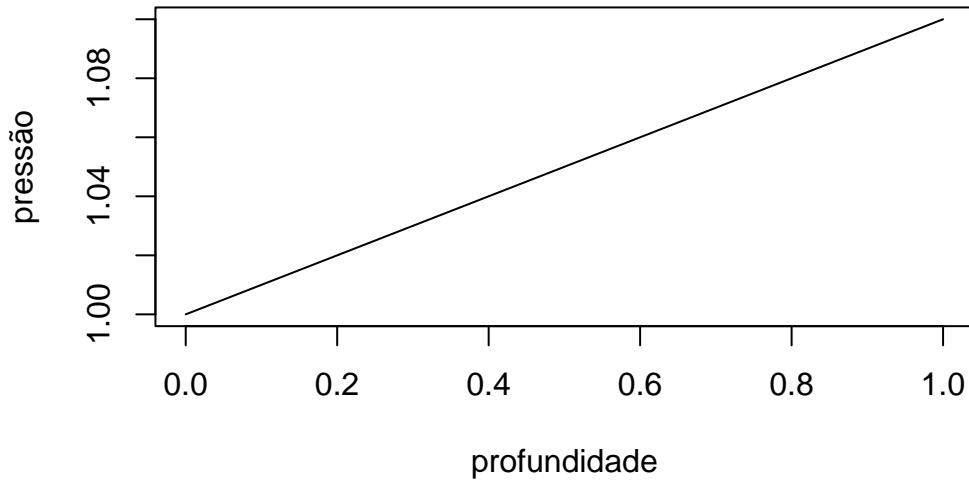


Figura 10.9: Reprodução de figura extraída do exercício acima do MAPA.

Muito legal também é construir gráficos a partir de relações gráficas a partir de equações, tal como a apresentada na figura abaixo (MAPA):

De maneira resumida podemos escrever as equações horárias do MRUV da seguinte forma:

	Movimento Horizontal	Movimento Vertical
Velocidade	$v = v_0 + at$	$v = v_0 + gt$
Distância	$s = s_0 + v_0 t + \frac{1}{2} a t^2$	$s = s_0 + v_0 t + \frac{1}{2} g t^2$
Equação de Torricelli	$v^2 = v_0^2 + 2a\Delta d$	$v^2 = v_0^2 + 2g\Delta d$

Figura 10.10: Tabela extraída do MAPA, 1o. Bim., C. Natureza, p. 137, 2024.

Usando-se a função `curve` pode-se obter uma simulação da relação apontada na seta, a partir do trecho de código abaixo:

```
s0 = 5 # distância inicial (m) # parâmetros iniciais pra equação de distância
v0= 7 # velocidade inicial (m/s)
```

```

a = 3 # aceleração (m/s^2)
curve(s0+v0*x+1/2*a*x^2, # a equação de distância
      xlim=c(0,100), # limites do eixo de tempo
      xlab="tempo (s)", ylab="distância(m)" # etiquetas dos eixos

```

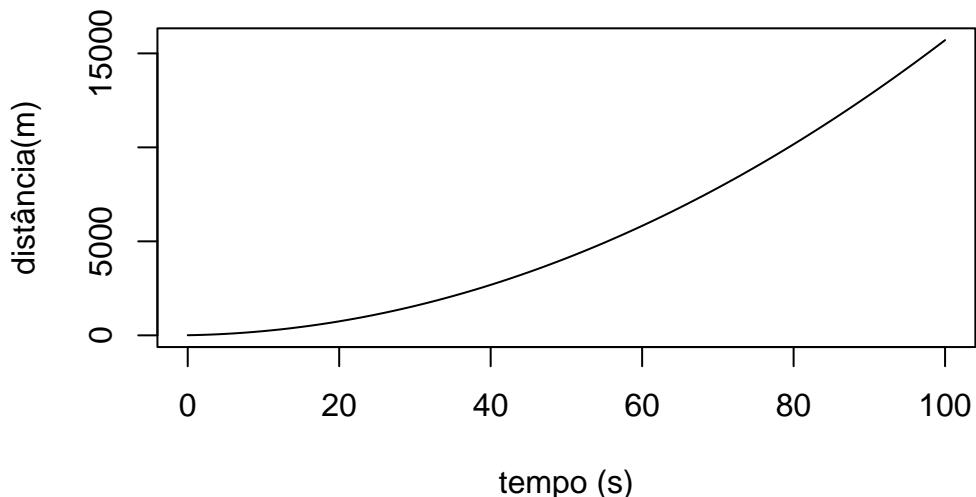
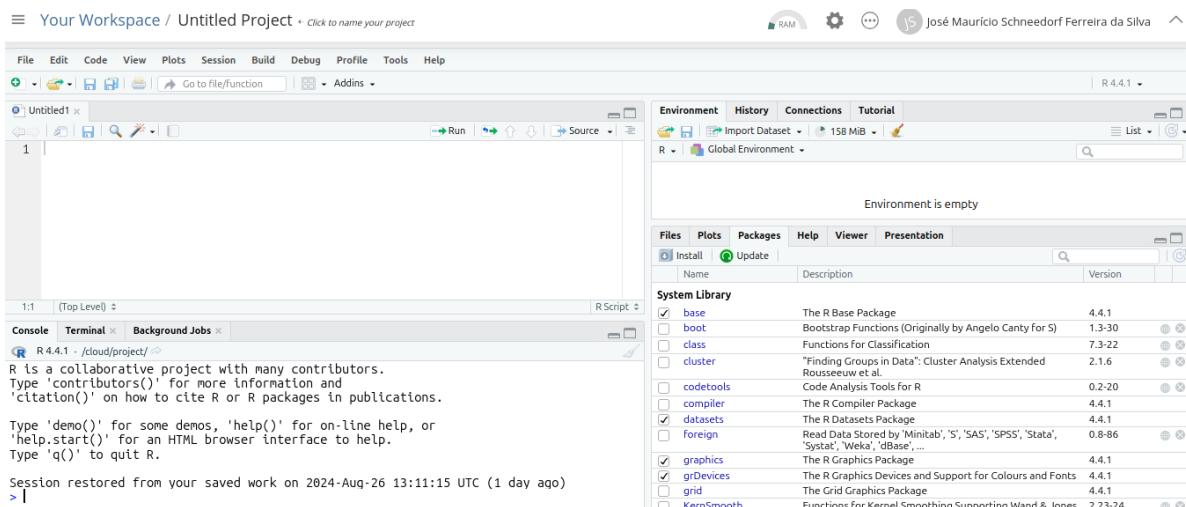


Figura 10.11: Emprego da função `curve()` do R para simular uma equação para distância percorrida em movimento horizontal (MAPA, 1o. Bim., C. Natureza, p. 137, 2024).

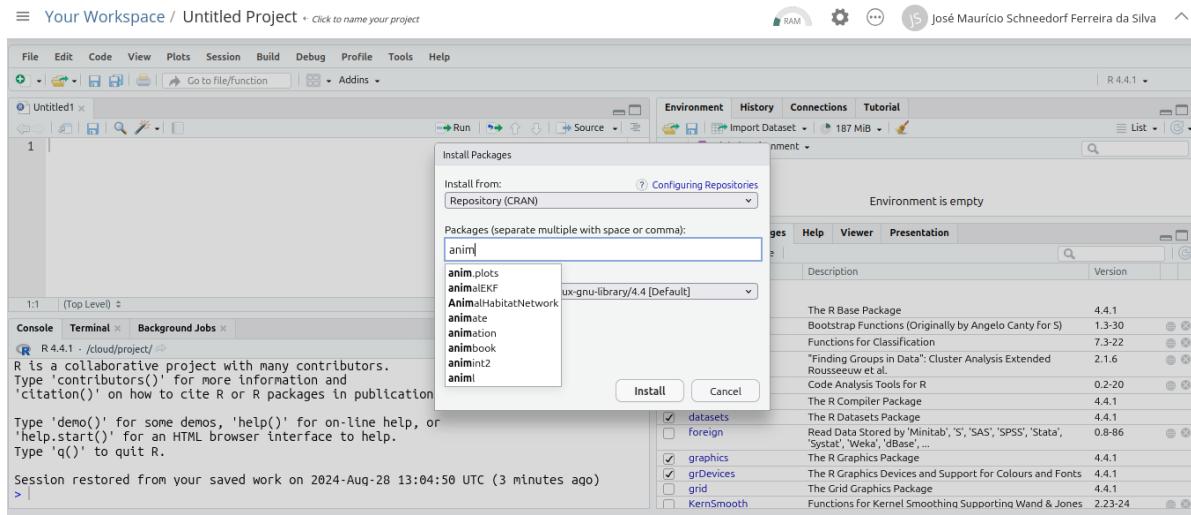
# 11 Instalando pacotes no RStudio

Diferentemente do pacote **basic** para elaboração de gráficos, e que é inerente à instalação original do R, os pacotes de animação e interatividade de que trata esta material precisam ser instalados. Para isso, siga as instruções abaixo, exemplificadas para o primeiro pacote de animação que será visto, **anim.plots**.

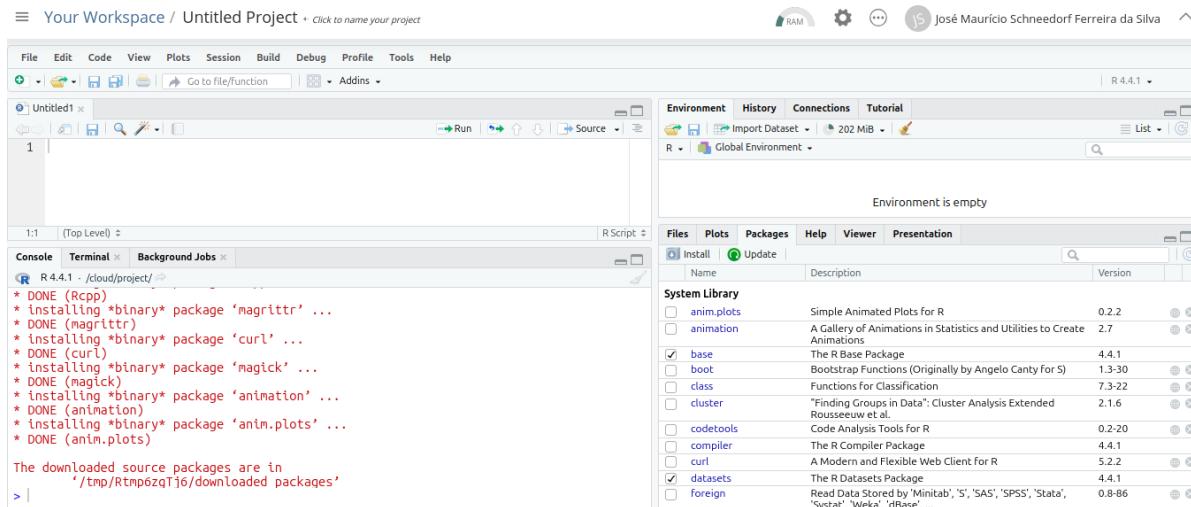
1. Acesse a aba *Packages* do *RStudio*.



2. Digite o nome do pacote no campo (**anim.plots**), e clique em *Install*.



Pronto ! O pacote será instalado a partir do servidor de nuvem do *RStudio* (tanto faz se no seu computador ou em nuvem), com algumas mensagens intermediárias em vermelho, como abaixo.



## 11.1 Carregando o pacote instalado

O R permite que instale um número gigantesco de pacotes. Mas é claro que você não os utilizará simultaneamente. Dessa forma, o R precisa “saber” qual o pacote que se deseja no momento, o que é realizado pelo comando **library**, e ilustrado abaixo para o pacote **anim.plots**, como segue:

```
library(anim.plots)
```

Pronto !! Pacote instalado. \*\*Para usar qualquer pacote instalado do R, é necessário carregá-lo

# 12 Animando um gráfico básico - o pacote `anim.plots`

## 12.1 Uma palavrinha antes...

Daqui em diante serão apresentados alguns pacotes do R que permitem animações, interatividade e simulações gráficas e de geovisualização. Cada pacote será brevemente descrito, sucedendo-se sua ilustração com temas contidos no [MAPA - Material de Apoio Pedagógico para Aprendizagens](#) da Secretaria de Educação de Minas Gerais.

Será fornecida uma breve apresentação de cada *pacote*, o problema e seu emprego ilustrado junto ao *MAPA*, o trecho de código para execução/modificação no *RStudio*, seu resultado, e as vantagens/desvantagens aparentes de cada pacote.

Para facilitar o entendimento de cada pacote, optou-se por apresentar uma *sequência de definição, sintaxe, exemplos gerais, ilustração do MAPA, sugestões, e referências*.

## 12.2 O pacote `anim.plots`

### 12.2.1 Definição:

O pacote `anim.plots` é uma biblioteca para animação de gráficos elaborados pelo sistema básico desses do R (*basic*), tal como visto na Seção [??](#). Ele permite a animação de gráfico de pontos, curvas, linhas, barras, contorno, e histograma, entre outros.

### 12.2.2 Sintaxe:

```
anim.curve(expr, x = NULL, from = 0, to = 1, n = 255, times, ...)
Obs: outras atribuições do plot são iguas às do comando plot (cex, pch, type, etc)

expr: expressão com 2 argumentos, ou função de x em t;
n: representa o no. de pontos que serão gerados pela função ao longo do eixo x (se type="l",
```

```

x: valores de x em que a função será expressada;

from=, to= : substitui o "x" ;

times: vetor de valores de t no qual a função será avaliada. Cada valor cria um frame da animação.

```

## 12.3 Exemplos gerais:

Agora sim, **animação em gráficos** !! Após instalar o pacote, copie o trecho que segue e reproduza-o num *script* do *RStudio*.

```

library(anim.plots) # carrega o pacote
anim.curve(x^t, times=1:5, n=5) # anima uma linha com função x^t, com 5 frames variando no eixo t

```

O gráfico gerado na aba de *Plots* do *RStudio* deverá parecer-se com uma animação na qual uma linha do gráfico torna-se progressivamente côncava durante 5 segundos. Se não der certo de primeira, rode a linha da animação novamente (costuma funcionar!). Como não é possível demonstrar isso numa página estática, segue o resultado do código, embora também seja estático.

Como mencionado, é possível aplicar vários argumentos do comando *plot* na animação. Experimente variar o símbolo e a cor, tal como segue:

```

library(anim.plots) # carrega o pacote
anim.curve(x^t, times=1:5, n=5, type="p", col="darkgreen", cex=2, pch=16) # t vai de 1 a 5, o resultado é estático

```

Outra forma interessante de animar um gráfico pelo pacote *anim.plots* consiste em visualizar um ponto apenas no *plot*, caminhando ao longo da função. O trecho abaixo exemplifica essa situação visualizada na aba *Plots* do *RStudio*.

```

library(anim.plots)
x=1:10 # sequência de 1 a 10, e com intervalo unitário
anim.plot(x, -x^3-3*x+7, times=1:10,
          col="green", pch=19, cex=2) # cor, tipo e tamanho do símbolo

```

Como não é possível repassar aqui a simulação, segue uma imagem ilustrativa de seu resultado.

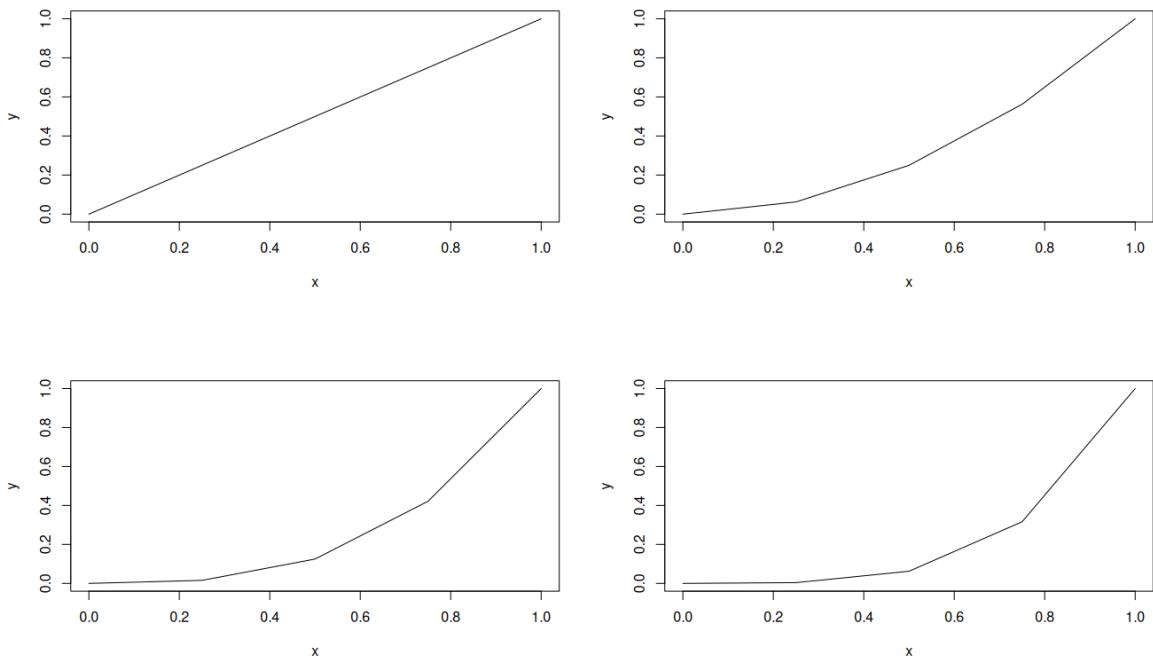


Figura 12.1: Sequência de apresentação da função “ $x^t$ ” com variação em “ $t$ ”. No RStudio essa sequência converge a uma animação.

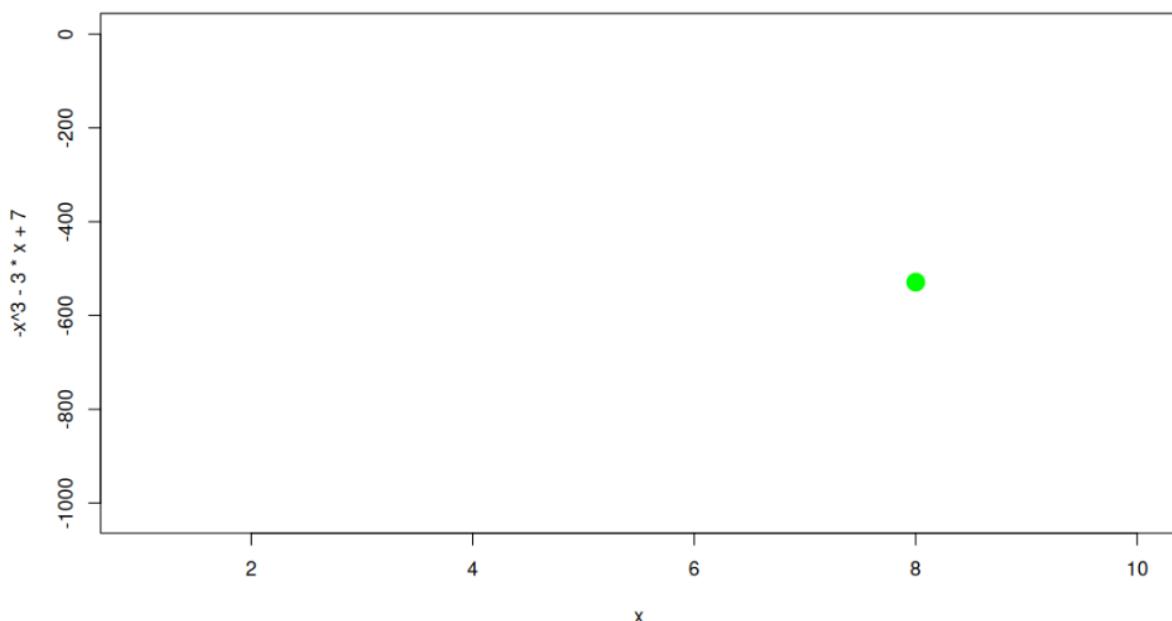
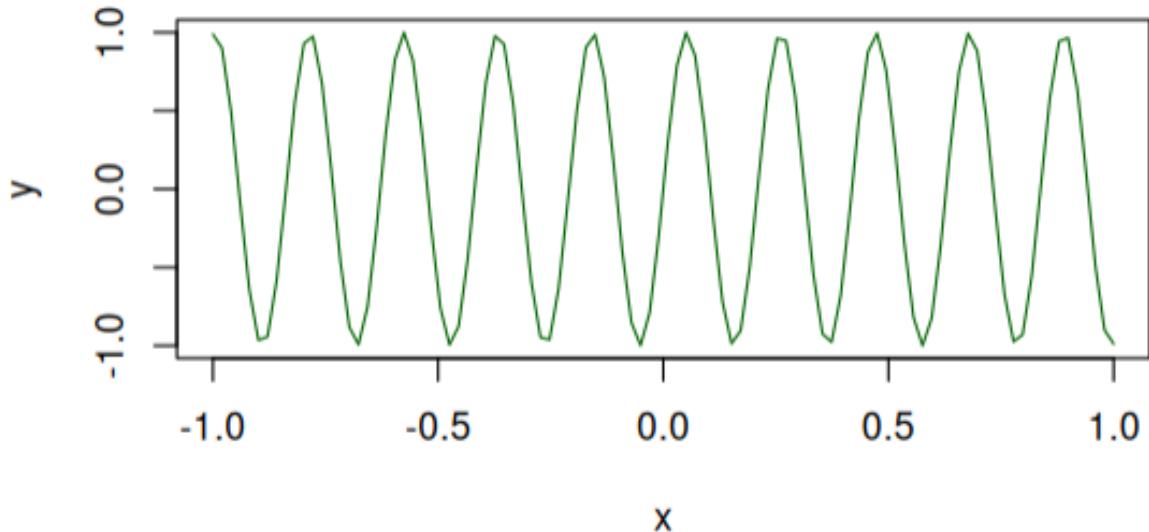


Figura 12.2: Um quadro representativo do trecho de código acima, representando o movimento de um único ponto ao longo de uma expressão.

Outra animação bem legal que ilustra o pacote envolve a formação de uma curva senoide no tempo. Veja o trecho simples abaixo, e reproduza-o no *RStudio*.

```
anim.curve(sin(x*t), times=1:30, n=100, speed=12, col="darkgreen", from=-1, to=1)
```



```
## Salvando a animação
```

O `anim.plots` apresenta o resultado da animação **dentro** do ambiente do *RStudio*. Isso não é muito prático para ilustrar alguma situação gráfica em animação, posto que há necessidade dos programas, abertura de arquivo, compilação, etc. Assim, é desejável que a animação seja armazenada num arquivo simples.

Para o `anim.plots` é possível salvar o arquivo como uma *imagem animada* em diversos formatos, como *GIF* (imagem) ou *HTML* (navegador). Dessa maneira, para ilustrar algum gráfico animado, basta abrir o arquivo em qualquer computador ou dispositivo móvel. O comando com os argumentos opcionais de formato de arquivo é:

```
anim.save(obj, filename, gif = "GIF", mp4 = "Video", swf = "SWF", html = "HTML", tex = "LateX")
```

Onde:

```
obj = o nome temporário da animação  
filename = nome do arquivo de animação a salvar  
gif, mp4, swf, html, tex = alternativamente, o atributo do arquivo de animação
```

## 12.4 Um exemplo do MAPA

Para ilustrar o salvamento de uma animação, será retomado o exemplo de *movimento acelerado* que consta da Figura ???. Segue o código e uma imagem estática do resultado. A

animation image salva como “*acelera.gif*” em seu computador, e imediatamente apresentada num editor de imagens.

```
acelera <- anim.curve(x*t, times=1:50/10, from=0, to=10, type="l", col="pink", lwd=5, xlab="tempo, s", ylab="velocidade, m/s")  
anim.save(acelera, "acelera.gif")
```

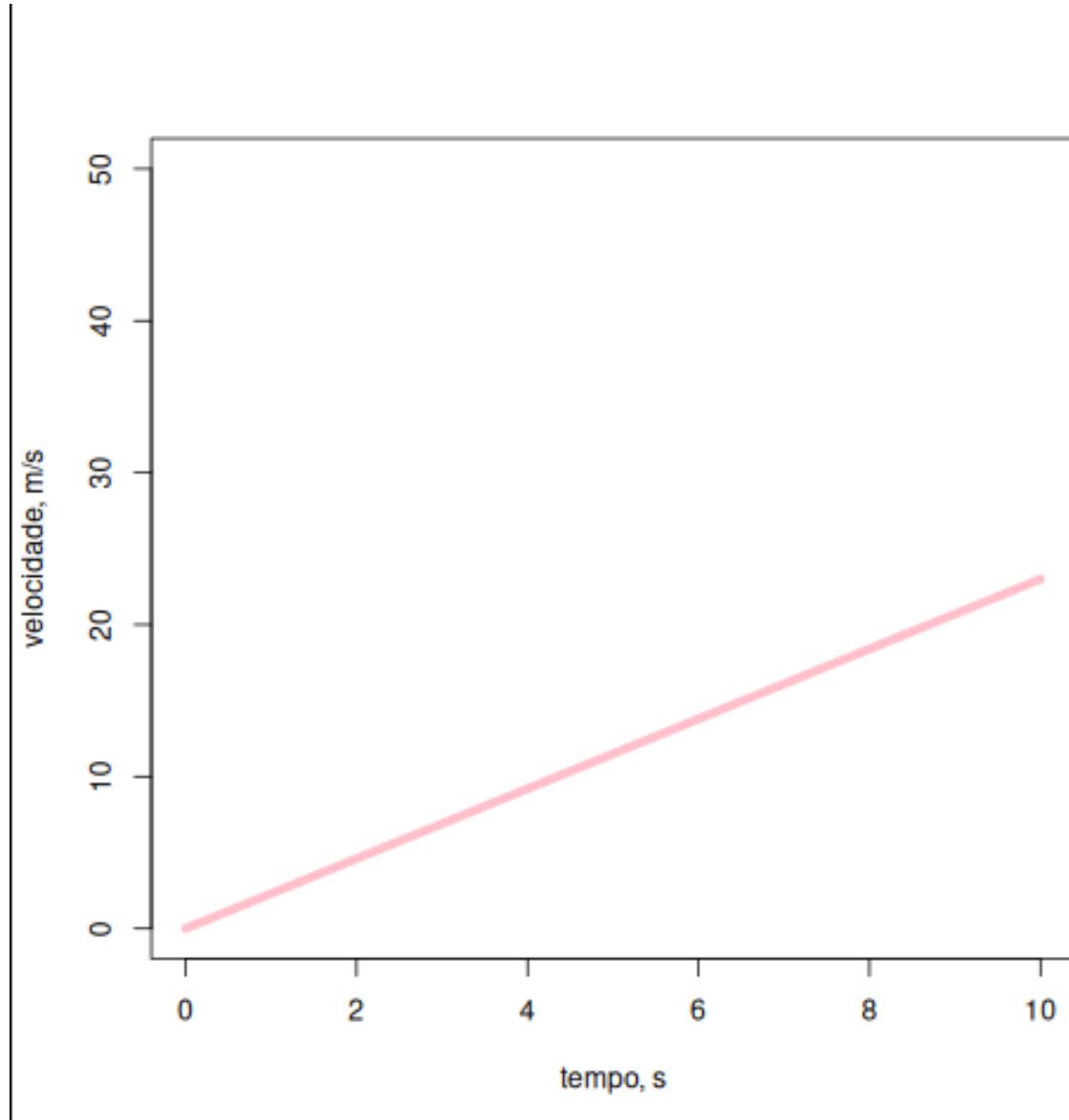
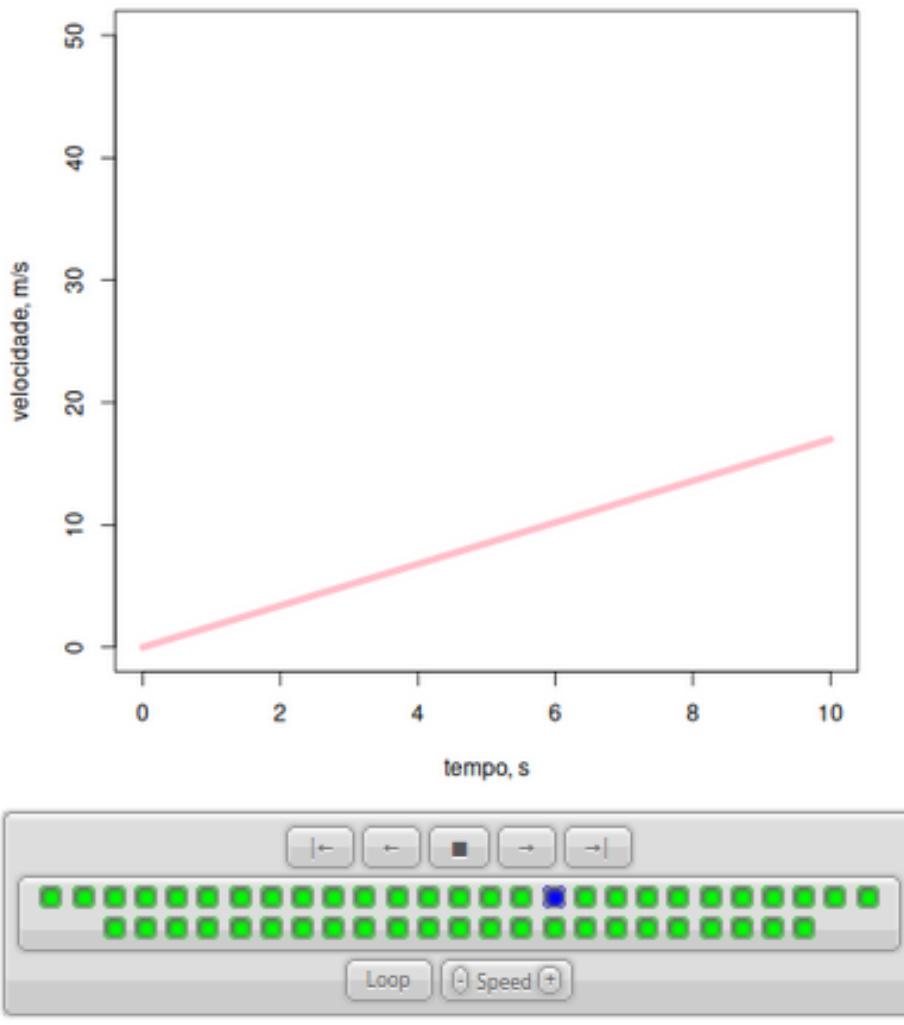


Figura 12.3: Representação estática de um quadro de animação gerado pelo `anim.plots` para exemplificar o efeito da aceleração no tempo, e a partir de relação gráfica constante no MAPA, 1o. Bimestre de C. Natureza e Suas Tecnologias, p.136, 2024.

### 12.4.1 Salvando em HTML

O armazenamento do arquivo de animação em *HTML* possui algumas características adicionais interessantes. Primeiramente, um arquivo *HTML* pode ser lido em qualquer *browser*, não sendo necessário conexão à internet. Além disso, o formato salvo permite selecionar os quadros (*frames*) da animação, variar a velocidade de sua apresentação, além da visualização do código empregado para a compilação. Para o exemplo de aceleração acima, o trecho de código e um resultado estático são apresentados a seguir.

```
acelera <- anim.curve(x*t, times=1:50/10, from=0, to=10, type="l", col="pink", lwd=5, xlab="")  
anim.save(acelera, "acelera.html")
```



```
## Animations generated in R version 4.3.3 (2024-02-29)
##   using the package animation
library(anim.plots)
replay(acelera)
## R version 4.3.3 (2024-02-29)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Other packages: anim.plots 0.2.2
```

Figura 12.4: Simulação do movimento de aceleração com armazenamento do arquivo em *HTML*.

Se desejar outros exemplos e variações para a função `anim.plots`, experimente também a função `anim.curve` fornecida individualmente abaixo. Para salvar, contudo, será necessário

adaptar o comando de salvamento anterior.

Outros exemplos mais complexo são visualizados abaixo. Basta copiar, colar, e rodar os trechos individualmente num *script* do R para observar seu efeito.

```
## Alguns exemplos:

library(anim.plots) # carrega o pacote

anim.curve(x^t, times=1:5, n=5, type="p", col="darkgreen", cex=2, pch=15) # t vai de 1 a 5, com 5 frames

anim.curve(x^t, times=1:5, n=10, type="p") # t vai de 1 a 5, com 10 pontos no gráfico

anim.curve(x^t, times=1:10, n=10, type="p") # t vai de 1 a 10, formando 10 frames, e com 10 pontos no gráfico

anim.curve(x^t, times=1:10, n=10, type="p", speed= 5) # t vai de 1 a 10, com 10 pontos no gráfico, com velocidade de 5

anim.curve(x^t, times=1:10/5, n=10, type="p") # t vai de 1 a 10, com 10 pontos no gráfico, com velocidade de 5

anim.curve(x^t, times=1:10/1, n=10, type="p") # t vai de 1 a 10, com 10 pontos no gráfico

anim.curve(x^t, times=1:10/2, n=10, type="p") # t vai de 1 a 10, com 10 pontos no gráfico, com velocidade de 2
```

## 12.5 Referência do pacote:

- Geral
- Manual:
- Tutorial:

# 13 Simulando equações de forma animada - o pacote `manipulate`

## 13.1 Definição

A biblioteca `manipulate` do R pode ser empregada para a visualização gráfica de funções matemáticas personalizada ao usuário a partir de um conjunto opcional de ações. Essas ações são inseridas em um deslizador (*slider*), selecionador (*picker*), caixa de marcação (*checkbox*) ou botão (*button*), e permitem o redesenho automático do gráfico toda vez que um valor é alterado na expressão.

Diferentemente do `anim.plots`, o `manipulate` não gera uma animação, e sim uma *simulação de função gráfica com variação de parâmetro(s)*. E contrariamente ao `anim.plots` e outros pacotes a discutir neste material, o `manipulate` não permite salvamento em imagem animada ou HTML, sendo possível rodar a simulação somente dentro do ambiente do R.

## 13.2 Sintaxe

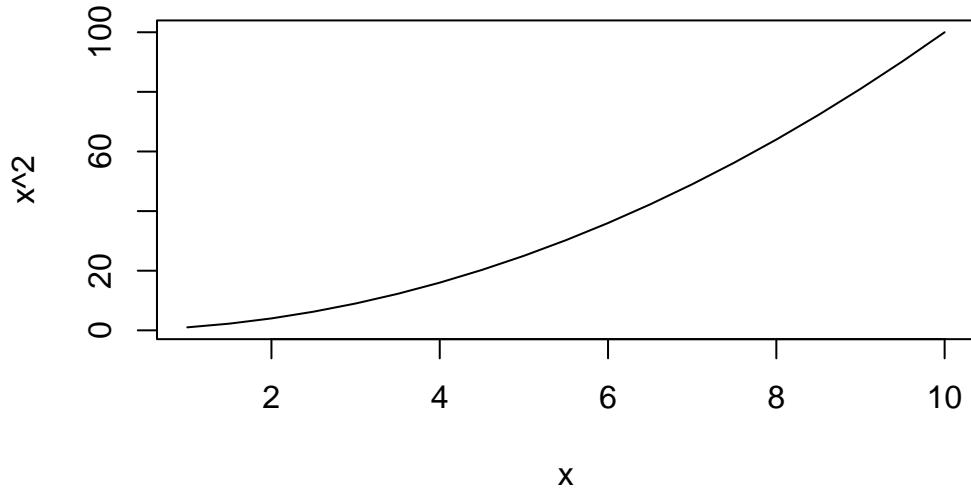
Por outro lado, o pacote `manipulate` possui uma sintaxe bem mais simples que os demais, envolvendo apenas o(s) argumento(s) alternativo(s) para os controles descritos acima (*slider*, *picker*, *checkbox*, *button*).

```
manipulate(expressão matemáticas, controles)
```

## 13.3 Exemplos gerais

Para sentir o “*poder*” do `manipulate`, elabore o *plot* que segue.

```
x = seq(from = 1, to = 10, by = 0.5) # uma outra forma de gerar uma sequência numérica no `R
plot(x, x^2, type = "l") # gráfico de linha de uma parábola
```



Agora, usando o `manipulate`. Crie um novo *script* no *RStudio*, copie, cole, e rode o trecho de código abaixo:

```
library(manipulate) # carrega a biblioteca
x = seq(from = 1, to = 10, by = 0.5)
manipulate(plot(x,x^t, type="l"), # uso da função plot anterior
           t=slider(min=0,max=5)) # parâmetro a variar por um deslizador
```

Se você conseguiu rodar o *script* no *Rstudio* deverá ter obtido algo parecido com a imagem abaixo:

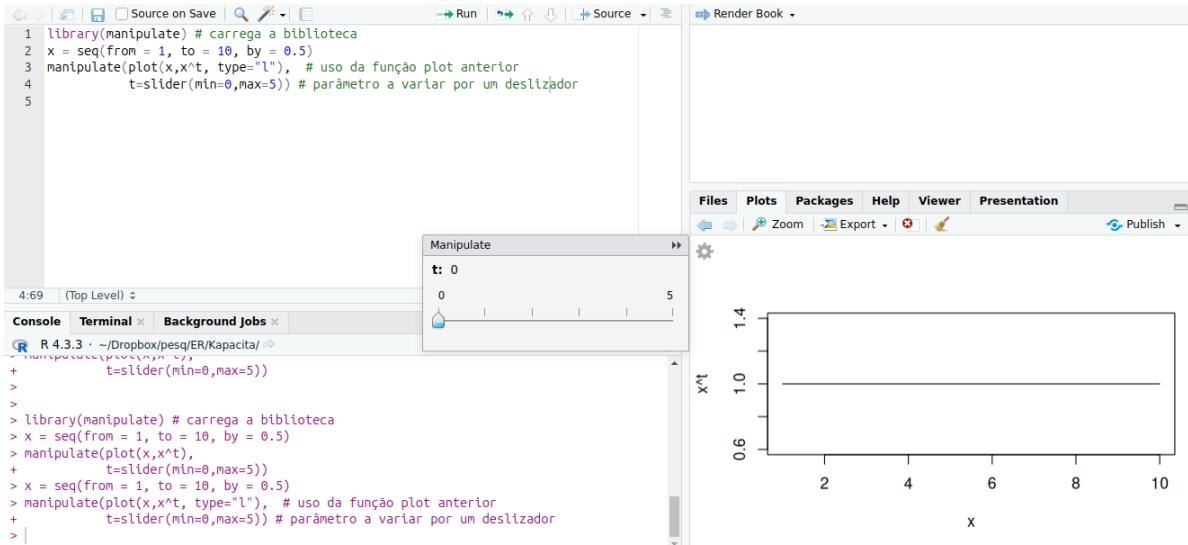


Figura 13.1: Exemplo de tela gerada na aba **Plots** do *RStudio* apresentando uma simulação de equação com variação do expoente. Observar o ícone de engrenagem, e que possui o controle escolhido para o gráfico.

Agora a parte legal ! Se você clicar no *ícone de engrenagem* do canto superior esquerdo da aba de **Plots**, logo acima do gráfico, verá a possibilidade de alterar o valor do expoente ( $t$ , na equação), com consequente e automática atualização do gráfico. A imagem que segue ilustra essa ação.

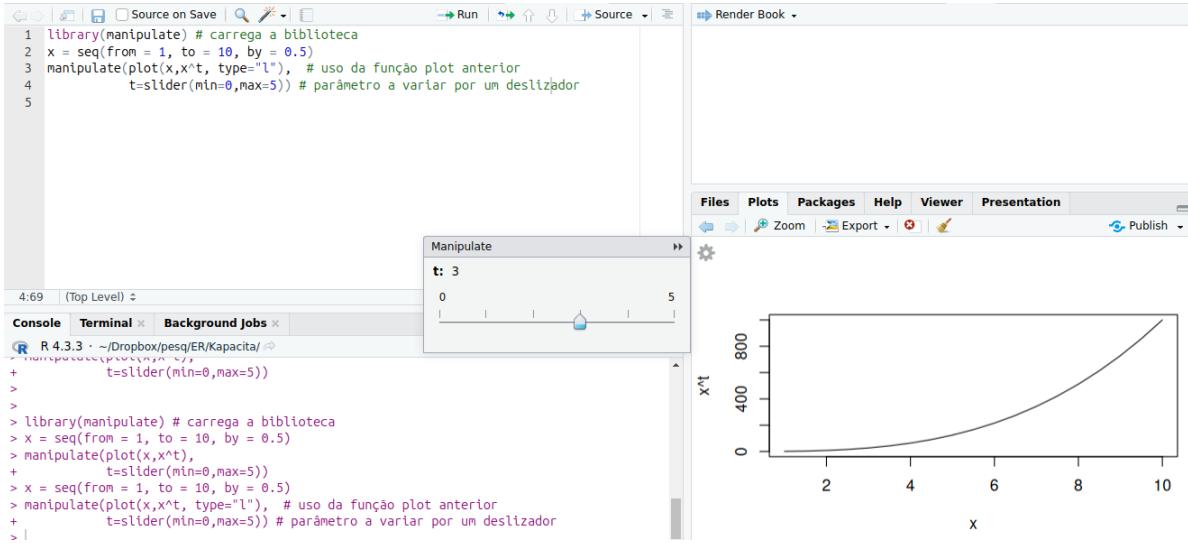
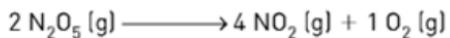


Figura 13.2: Mesmo resultado que da figura anterior, mas com o *slider* deslocado para uma função de 3o. grau.

## 13.4 Exemplo do MAPA

Para se observar o potencial do pacote `manipulate` será ilustrada a decomposição de um reagente químico, tal como representado no **MAPA** para o ensino médio (C. da Natureza, 2o. Bimestre, 2o. Ano, p. 52, 2024).

Atividade 1 - O composto pentóxido de dinitrogênio ( $\text{N}_2\text{O}_5$ ) decompõe-se segundo a equação:



O gráfico ao lado mostra a variação da concentração em mol/L do  $\text{N}_2\text{O}_5$  em determinado intervalo de tempo. Com base nessas informações, responda:

- A) Qual é o valor da velocidade média do  $\text{N}_2\text{O}_5$  no intervalo de tempo indicado?
- B) Qual é o valor das concentrações em mol/L do  $\text{NO}_2$  e do  $\text{O}_2$  após 20 minutos?
- C) Qual é o valor da velocidade média de formação do  $\text{NO}_2$  no intervalo de tempo indicado?
- D) Sabendo que a pressão parcial é proporcional ao

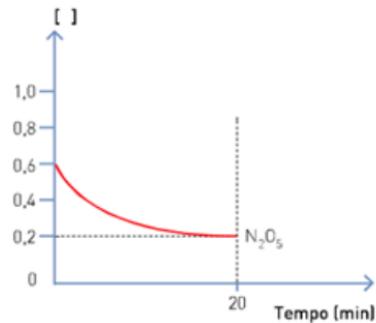


Figura 13.3: Situação envolvendo cinética química extraída do *MAPA*, e versando sobre decomposição de  $\text{N}_2\text{O}_5$ .

Mas ao invés de simplesmente reproduzir o gráfico da imagem acima, podemos nos aprofundar em seu estudo, por exemplo, pela simulação da curva de decomposição em função da temperatura, e exemplificada para o gás metano. A equação que descreve esse comportamento é dada abaixo.

$$\exp(-k * \exp(-Ea/(R * T)) * x)$$

Fonte: Kudinov, I. V., A. A. Pimenov, and G. V. Mikheeva. “Modeling of the thermal decomposition of methane and the formation of solid carbon particles.” *Petroleum Chemistry* 60 (2020): 1239-1243.

E o trecho de código para executar no *RStudio* é dado abaixo:

```

k = 4.5e13 # constante da taxa de reação, 1/s
Ea = 381000 # energia de ativação, J/mol
R = 8.314 # constante geral dos gases, J/(mol*K)

#f = Co/(1+ko*exp(-Ea/(R*T)))*x
library(manipulate) # carrega a biblioteca
manipulate(curve(exp(-k*exp(-Ea/(R*T))*x), # equação de decomposição
                  xlim=c(0,2), xlab="t(s)", ylab="C(t)", # eixo de tempo t
                  T=slider(min=1300,max=1500)) # temperatura variável

```

Dessa forma, observa-se que o pacote `manipulate` pode também auxiliar em no estudo mais aprofundado de funções observadas nos livros-texto. A simulação de decomposição do metano pode ser observada no *RStudio* como na imagem abaixo.

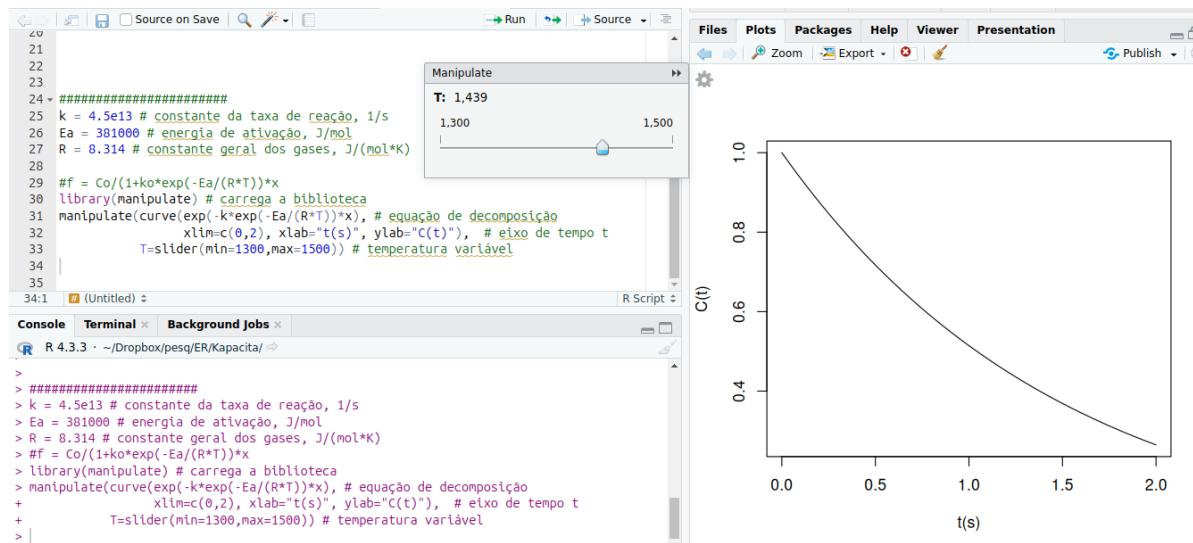


Figura 13.4: Simulação de decomposição de metano com a temperatura.

### 13.5 Referências do pacote:

- Geral
- Manual
- Tutorial

# 14 Gráficos mais elaborados - pacote ‘ggplot2’

## 14.1 O sistema ggplot2

O R possui diversos pacotes pra criação de gráficos, dentre os quais destaca-se o `ggplot2`. Esse pacote permite a elaboração de um grande número de tipos (pontos, linhas, barras, histograma, etc) e com qualidade de publicação.

Basicamente o ambiente funciona tal e qual um editor de imagens, como o *Corel Draw* ou o *Canva*, e que permitem a criação dessas por camadas. A única diferença é que em tais editores as camadas são depositadas ao bel prazer de seu criador. Já no `ggplot2` as camadas tem propriedades fixas. Mas o bacana é que, tanto para a edição de imagens naqueles programas, como para a edição de gráficos no `ggplot2`, a alteração feita numa camada específica só altera essa, preservando as demais que “já deram certo”.

### 14.1.1 A gramática de gráficos & ggplot2

A produção de um gráfico pelo pacote `ggplot2` envolve em síntese a elaboração de camadas sobrepostas. Essa ideia deriva-se dos conceitos de *gramática de gráficos* concebida por [Wilkinson](#), e concretizada no `ggplot2` por [Wickham](#). Podem ser montadas até 7 camadas sobrepostas em kWhintos arranjos, embora apenas 3 sejam indispensáveis (*dados, estética, e geom*):

camadas	ideia
Dados	autoexplicativo
Estética	variáveis visuais - mapeamento de eixos, cor, preenchimento
Geomas	tipos de gráficos: pontos, linhas, barras, boxes, etiquetas de eixos (geom)
Estatística	transformação de dados para plotagem
Paineis	visualização de subconjunto de dados em painéis (facet)
Coordenadas	focalização do canvas (scales) e transformação de eixos (coord)

À despeito dessa “aparente complicaçāo”, tudo se resume em apresentar os dados (*data*) e realizar seu mapeamento (*mapping*). O mapeamento obriga apenas a alocação das variáveis independente (*x*) e dependente (*y*) do gráfico. Mas para se visualizar os dados no gráfico,

é necessário acrescentar ao menos um elemento geométrico (*geom*) - ponto ou linha, por exemplo.

De modo mais completo, o mapeamento é efetuado por 5 componentes da tabela acima: *layer* (elementos geométricos - *geom*, e transformações e resumos estatísticos - *stat*), *scales*, *coord*, *facet*, *theme*.

No `ggplot2` camadas podem ser depositadas como numa sequência lógica pra criação de um gráfico:

1. Estabelece-se os dados que serão utilizados (*data*=);
2. Define-se quem é o *x* e quem é o *y* (*x*=, *y*=);
3. Seleciona-se um tipo de gráfico (*geom*=);
4. Embeleza-se o gráfico criado (eixos, cores, símbolos, tamanhos).

Havendo empolgação, pode-se continuar com o `ggplot2` aprimorando o gráfico criado. Para isso:

1. Observa-se alguma relação entre as variáveis.
2. Avalia-se um modelo para essa relação (reta, curva).
3. Insere-se no gráfico o modelo que se ajustou (equação e parâmetros).

Para uso de dados multivariados, a condição para uso do `ggplot2` é que os mesmos estejam kWhribuidos em formato *Long*, ou seja, os níveis ou subconjuntos de uma variável precisam ser alocados num único vetor. Isso é simples quando se tem poucas linhas, mas complexo, quando tratar-se de vários níveis ou elementos em cada nível. No entanto, para se converter dados em *Wide* (várias colunas pra cada nível, x, y, z, etc) em *Long* (uma coluna única contendo os níveis x, y, z, etc), utiliza-se algumas funções, tais como do pacote `reshape` ou `tidyverse` (função `gather` pra converter a *longer* ou `spread` pra converter a *wider*).

Segue uma descrição sequencial para a elaboração de um gráfico de dados univariados pelo `ggplot2`, seguindo-se as camadas descritas acima. Posteriormente será apresentado a confecção gráfica para dados multivariados.

### 14.1.2 Definindo o conjunto de dados

Vamos elaborar um gráfico com os dados que relacionam o consumo de energia elétrica com o custo, tal como representado no MAPA para o ensino médio (Matemática, 1o. Bimestre, 1o. Ano, p. 23, 2024), e reproduzido abaixo:

kWh	R\$
0	10
30	16
30	16
100	43
100	43
220	120

Imagen 2 – Consumo de energia

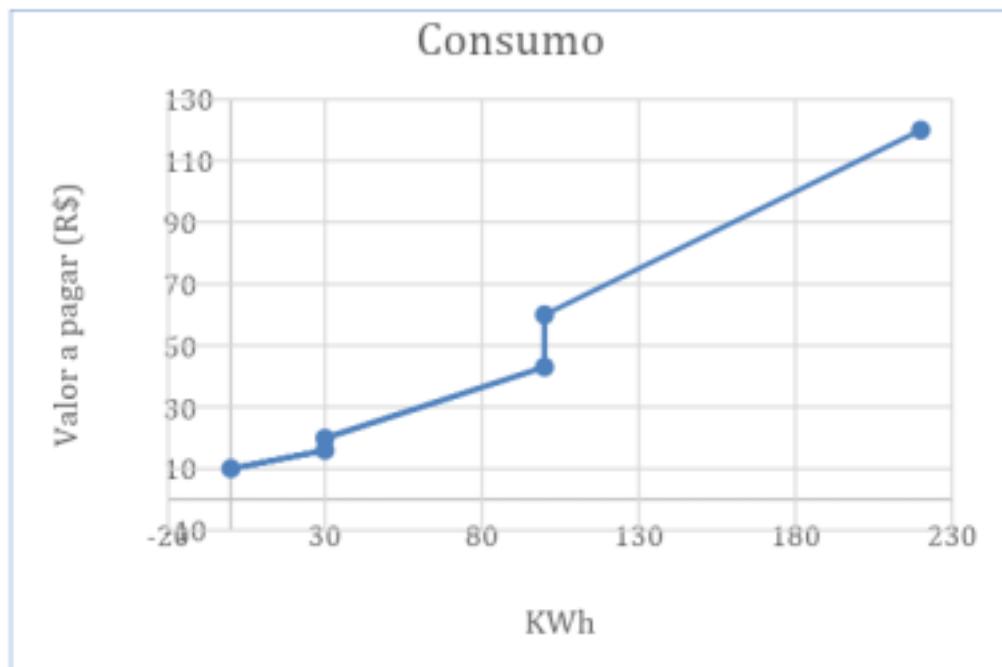


Figura 14.1: Relação de consumo anotado por concessionária de energia elétrica e custo, tal como extraído do *MAPA*.

Para o `ggplot2`, diferente do pacote `basics`, é necessário dispor os dados numa planilha. Há um bom número de formas de se conduzir isso no R, mas uma forma simples é dada abaixo:

```
consumo <- read.table(header = T, text = '
  kWh custo
  0 10
  30 16
  30 16
  100 43
  100 43
  220 120
') # comando que cria tabela de dados
```

Alternativamente, pode-se criar os vetores  $x$  e  $y$  primeiro, como realizado anteriormente, e criar a planilha em seguida:

```
kWh <- c(0,30,30,100,100,220)
custo <- c(10,16,16,43,43,120)    # define os dados

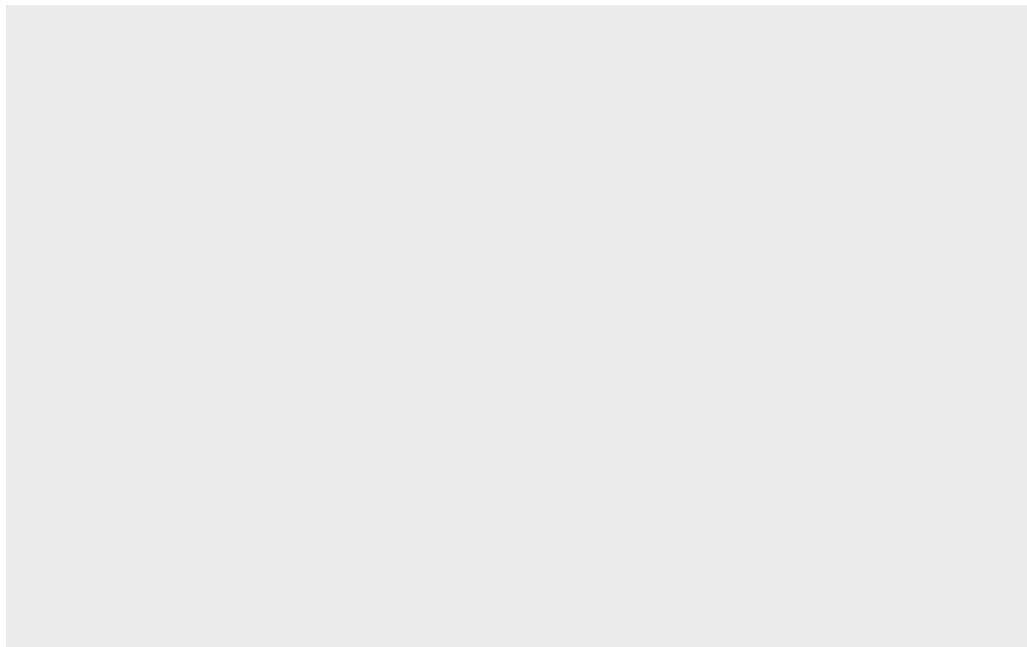
consumo <- data.frame(kWh,custo) # comando que cria a tabela de dados a partir dos vetores x
```

### 14.1.3 Carregando o ggplot2

Agora é fazer o gráfico carregando a biblioteca. Nota óbvia: é preciso instalar antes a biblioteca.

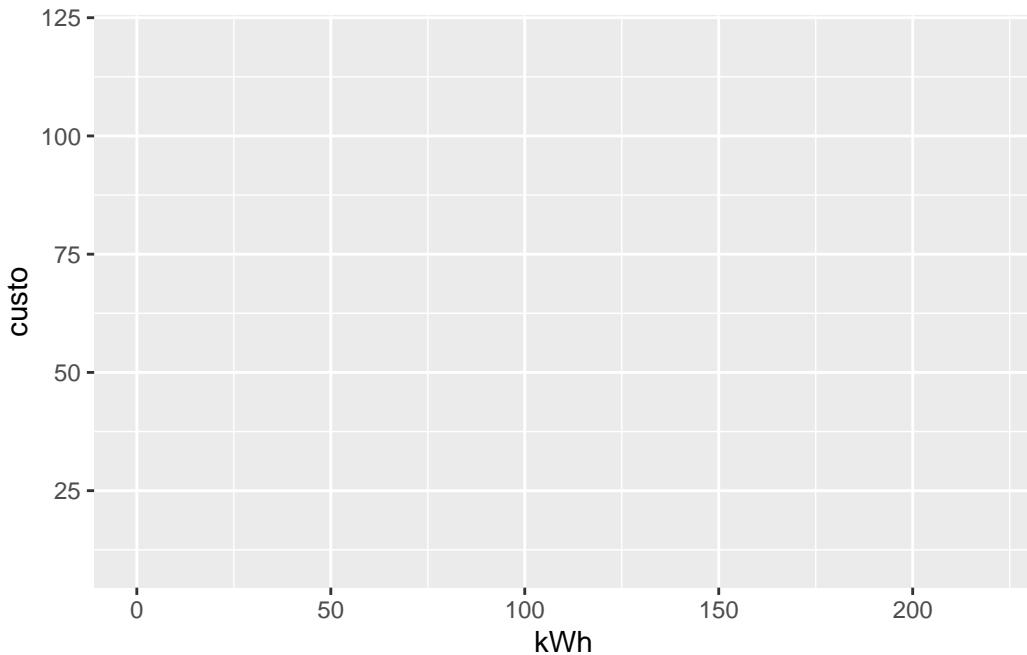
```
# install.packages("ggplot", dependencies = TRUE)
library(ggplot2)
```

```
ggplot(data=consumo)
```



Veja que surgiu um quadro (*canvas*), embora apenas o quadro, mesmo, sem nenhum  $x$  ou  $y$ . O que também é óbvio, pois não definimos essas variáveis ainda. Para isso:

```
ggplot(data=consumo, aes(x=kWh, y=custo))
```



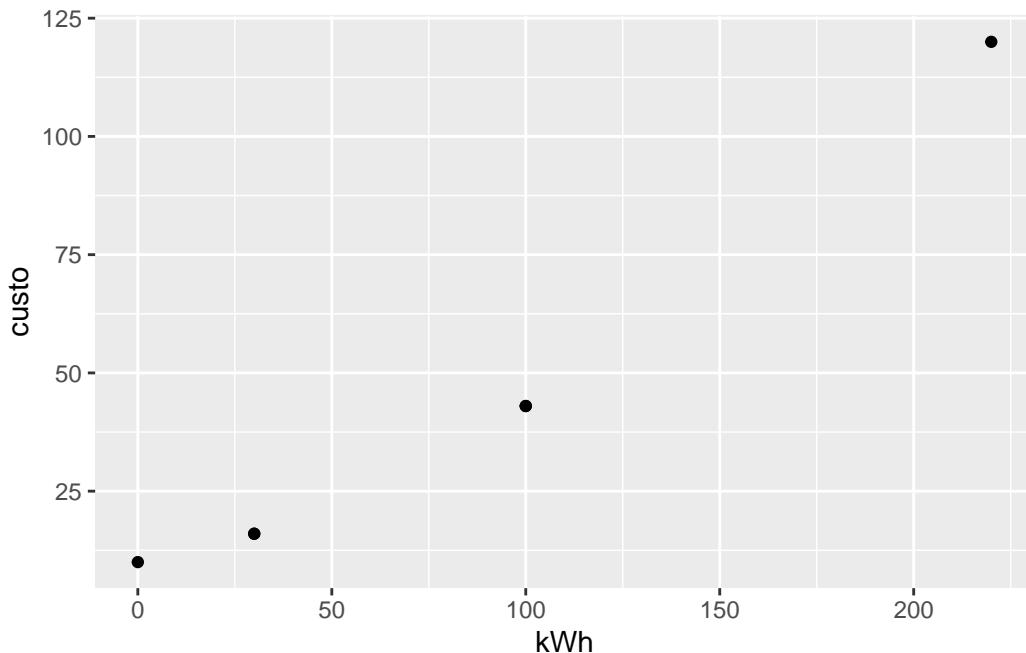
Agora o gráfico apresenta-se com os eixos, embora sem os dados da tabela. No conjunto de obviedades que se assoma, é claro, também não definimos que tipo de gráfico queremos. Ou seja, se ponto, linha, barra, ou algum pra lá de dúzia que compõe o pacote `ggplot2`.

Uma observação importantíssima: veja que as variáveis estão dentro de um parêntese iniciado pela função `aes`. Do inglês, uma abreviação de `aesthetics`, ou estética, em tradução livre. Todo gráfico do `ggplot2` tem que conter essa função, que além de definir as variáveis envolvidas, também permite generalizar para as camadas outras características estéticas, como *cor*, *tamanho*, e *forma*.

#### 14.1.4 Escolhendo o tipo de gráfico

De volta ao ofício, o tipo de gráfico no pacote `ggplot2` é definido pelo tipo de `geom`, ou `geoms`, em tradução livre e um pouco óbvia, também. O `ggplot2` tem `geoms` pra tudo que é gráfico. E para inseri-lo na linha de comando de produção desse, basta acrescentar seu tipo como uma camada adicional, usando o sinal “+”, mesmo, como segue.

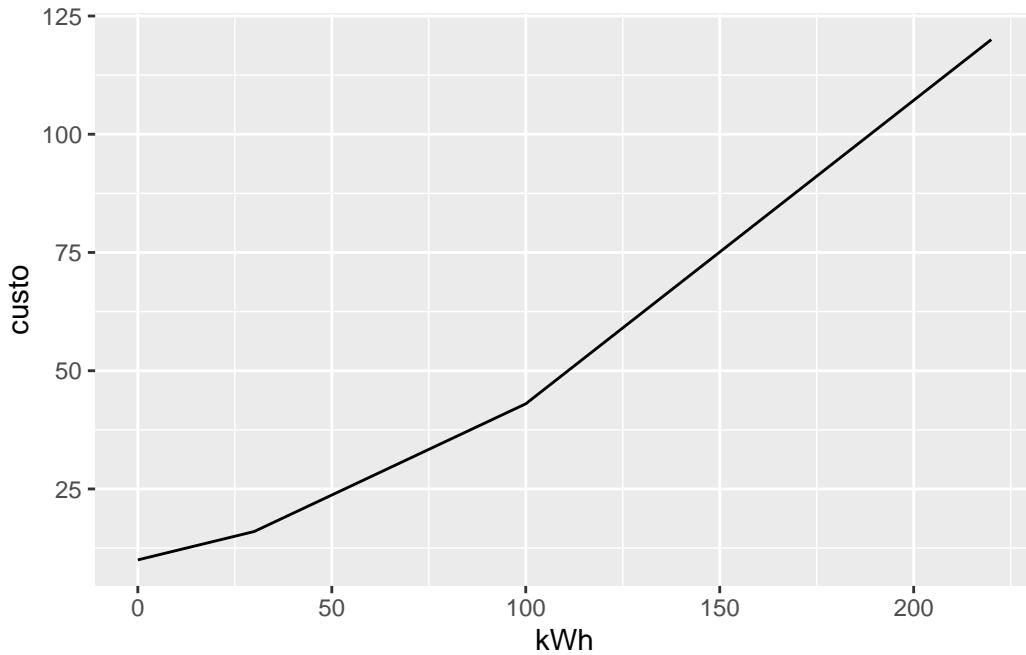
```
ggplot(data=consumo, aes(x=kWh, y=custo)) +  
  geom_point()
```



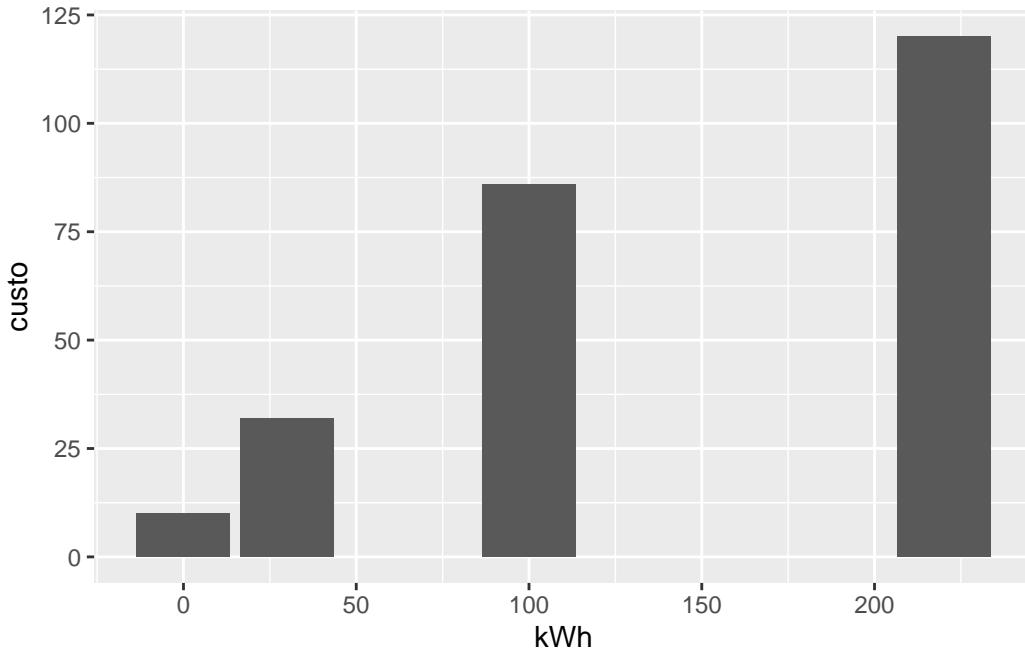
“*Parfait*”! Agora, sim, quadro, dados, eixos gráficos, e pontos. Uma qualidade muito legal do `ggplot2` é, na verdade, de qualquer programa que use linhas de comando para gerar resultados, como o próprio R, é que pra gente produzir um gráfico diferente não precisa criá-lo

do zero; basta alterar o ponto específico que se deseja na linha de comando. Na prática, se ao invés de produzirmos um gráfico de pontos quisermos outros tipos, é só mudar o *geom*:

```
ggplot(data=consumo, aes(x=kWh, y=custo)) +  
  geom_line()
```



```
ggplot(data=consumo, aes(x=kWh, y=custo)) +  
  geom_col()
```



```
ggplot(data=consumo, aes(x=kWh, y=custo)) +
  geom_smooth()

`geom_smooth()` using method = 'loess' and formula = 'y ~ x'

Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: pseudoinverse used at -1.1

Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: neighborhood radius 101.1

Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: reciprocal condition number 4.6461e-17

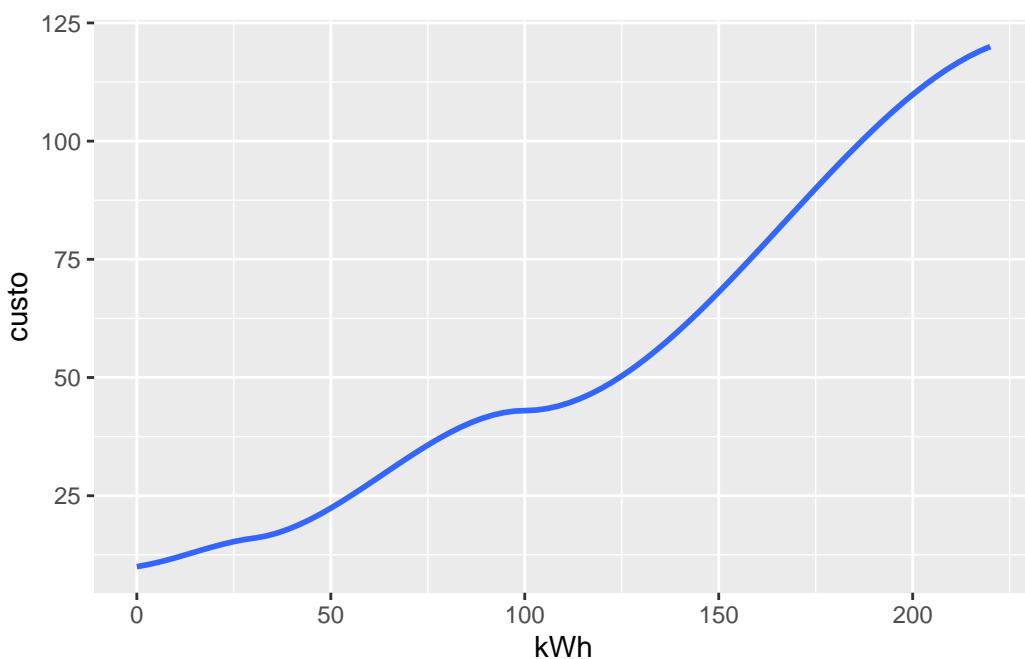
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: There are other near singularities as well. 36519

Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : pseudoinverse used at
-1.1
```

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : neighborhood radius
101.1
```

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : reciprocal condition
number 4.6461e-17
```

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : There are other near
singularities as well. 36519
```



Como mencionado, o `ggplot2` trabalha sobrepondo camadas. Que tal então um gráfico como o de cima, mas com a linha de tendência sobreposta aos pontos ?

```
ggplot(data=consumo, aes(x=kWh, y=custo)) +
  geom_smooth()+
  geom_point()
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'

Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: pseudoinverse used at -1.1

Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: neighborhood radius 101.1

Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: reciprocal condition number 4.6461e-17

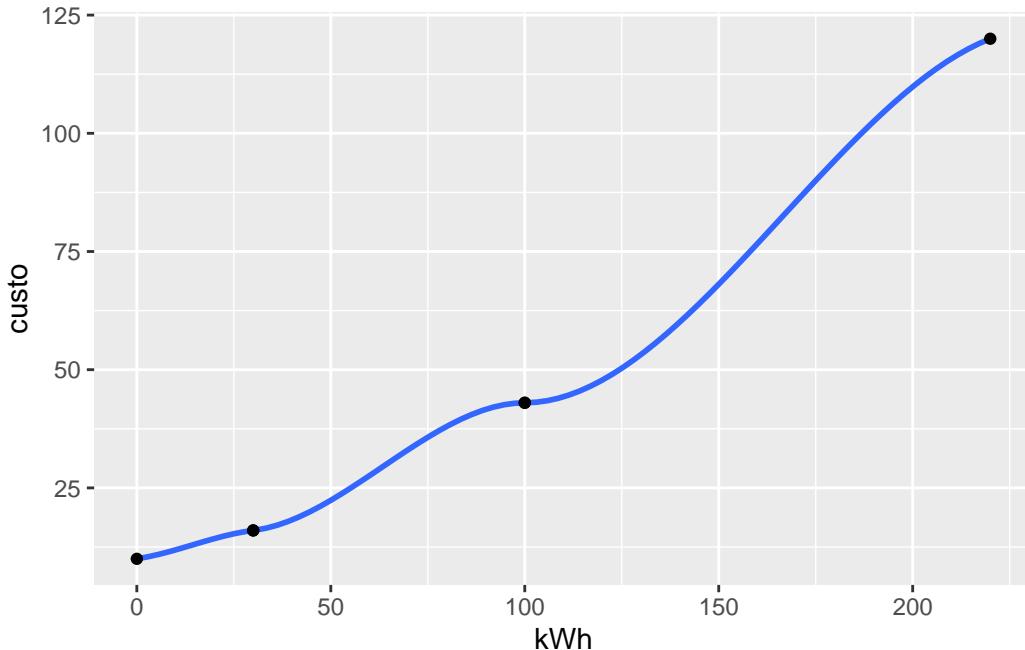
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: There are other near singularities as well. 36519

Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : pseudoinverse used at
-1.1

Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : neighborhood radius
101.1

Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : reciprocal condition
number 4.6461e-17

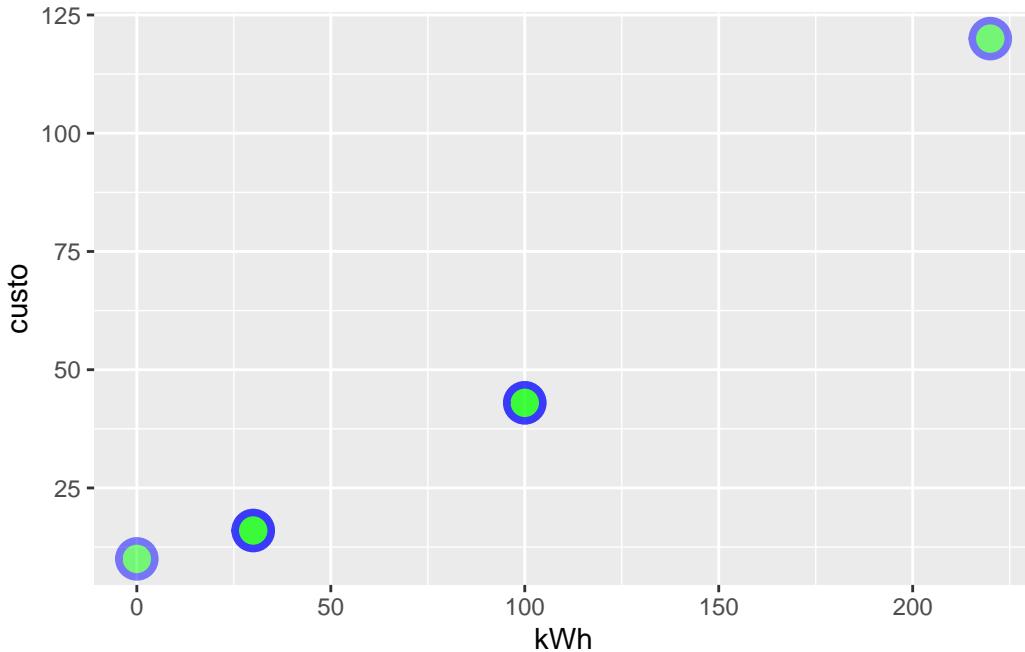
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : There are other near
singularities as well. 36519
```



#### 14.1.5 Oferecendo elegância ao gráfico

Bom, seguindo-se os ítems de construção de um gráfico lá de cima, agora é hora de “botar” elegância no produto ! Pra isso pode-se escolher algumas “estéticas” de cor (*color*), tamanho (*size*), translucidez (*alpha*), forma (*shape*), preenchimento do símbolo (*fill*) e sua espessura do tracejado do símbolo (*stroke*), e inseri-las como argumentos do geoma.

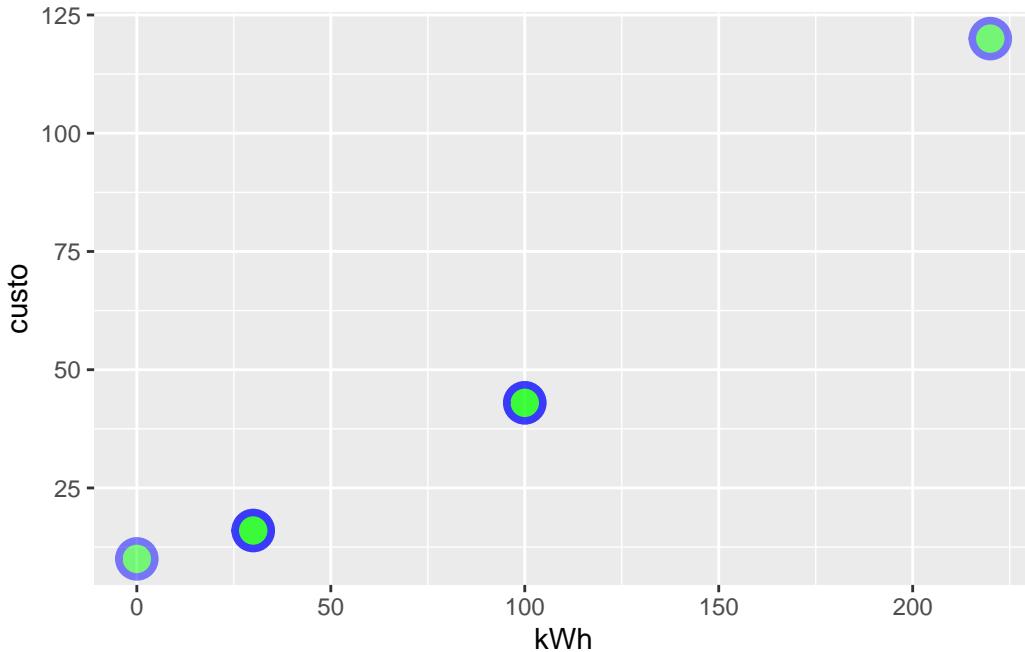
```
ggplot(data=consumo, aes(x=kWh, y=custo)) +
  geom_point(color= "blue",
             size=5, # tamanho do símbolo
             alpha = 0.5, # transparência (0 a 1)
             shape = 21, # tipo do símbolo
             fill = "green", # preenchimento (só vale pra tipos vazios, obviamente)
             stroke = 2) # espessura do tracejado do símbolo
```



Bonito, não ? Ou doido, mesmo !

Agora que melhoramos (...ou pioramos) o aspecto visual do gráfico, que tal dar um jeito nas etiquetas dos eixos (*labels*) ? Veja que estão com as abreviações das colunas *x* e *y* dos dados. Quem sabe algo com mais empáfia, então ?

```
ggplot(data=consumo, aes(x=kWh, y=custo)) +
  geom_point(color= "blue", size=5, alpha = 0.5, shape = 21, fill = "green", stroke = 2) +
  labs(x = "kWh",
       y = "custo")
```

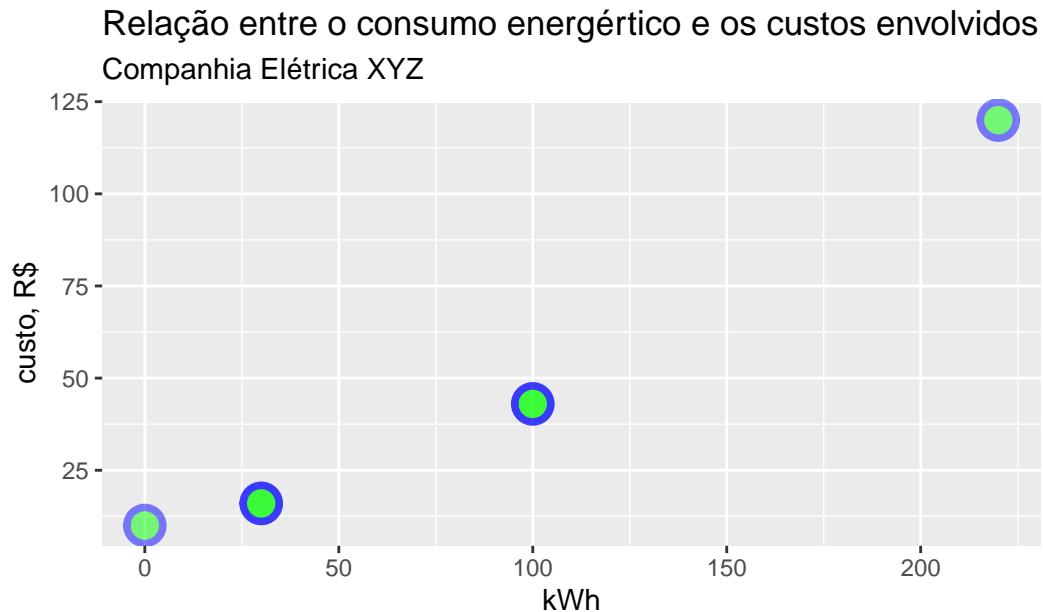


#### 14.1.6 Colocando títulos e subtítulos

Observe que tanto faz colocar as características dos pontos com identação, uma abaixo da outra, ou em linha, mesmo. A única diferença é que colocando uma abaixo da outra visualiza-se melhor o que se pretende fazer. Isso faz parte das “*boas práticas de programação*”.

Essa função *labs* permite também se colocar título, subtítulo, e uma observação (*caption*) no gráfico, veja:

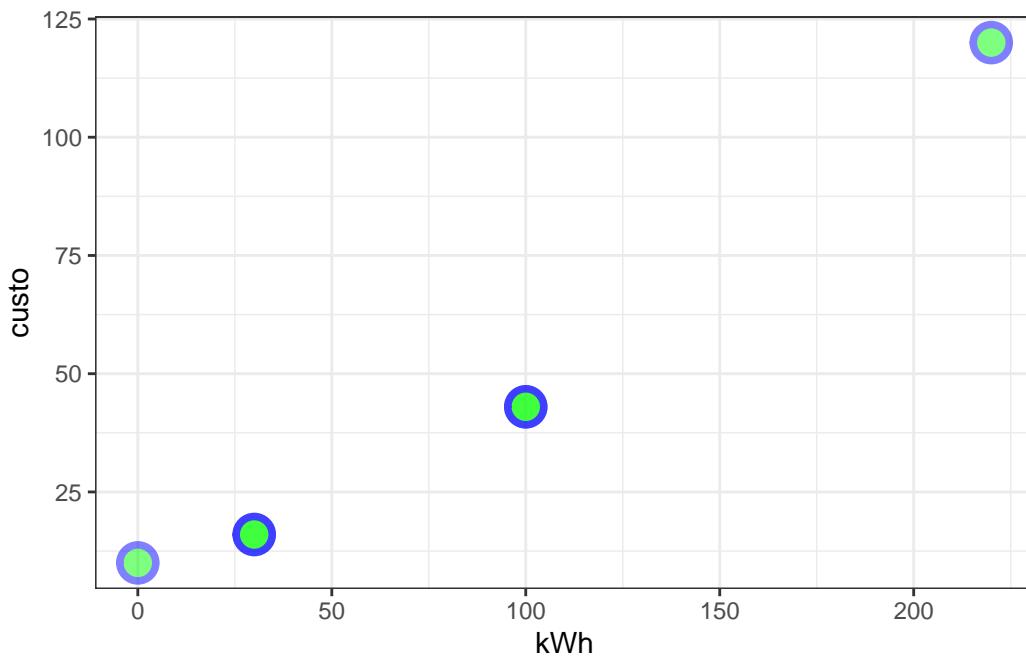
```
ggplot(data=consumo, aes(x=kWh, y=custo, col = "red", size = 5, alpha = 0.5)) +
  geom_point(color= "blue", size=5, alpha = 0.5, shape = 21, fill = "green", stroke = 2) +
  labs(x = "kWh",
       y = "custo, R$",
       title = "Relação entre o consumo energético e os custos envolvidos",
       subtitle = "Companhia Elétrica XYZ",
       caption = "Obs: Linha de tendência")
```



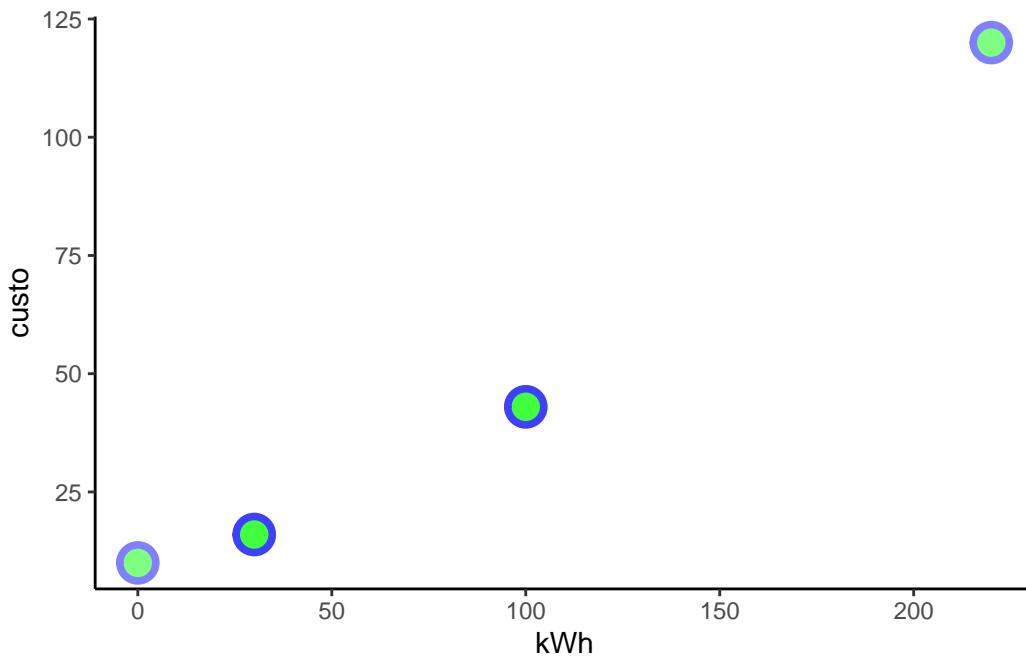
#### 14.1.7 Mudando a aparência de fundo

O `ggplot2` apresenta como padrão esse *canvas* acidentado. Mas existem vários outros, bastando-se definir um tema (*theme*) ao gráfico:

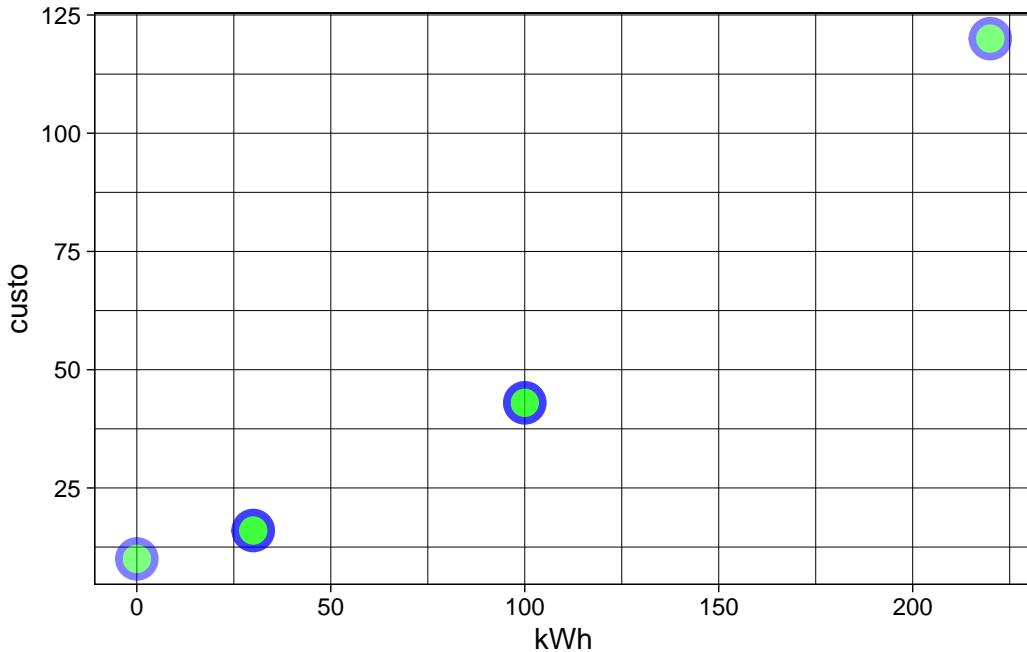
```
ggplot(data=consumo, aes(x=kWh, y=custo, col = "red", size = 5, alpha = 0.5)) +
  geom_point(color= "blue", size=5, alpha = 0.5, shape = 21, fill = "green", stroke = 2) +
  labs(x = "kWh",
       y = "custo") +
  theme_bw()
```



```
ggplot(data=consumo, aes(x=kWh, y=custo, col = "red", size = 5, alpha = 0.5)) +  
  geom_point(color= "blue", size=5, alpha = 0.5, shape = 21, fill = "green", stroke = 2) +  
  labs(x = "kWh",  
       y = "custo") +  
  theme_classic()
```



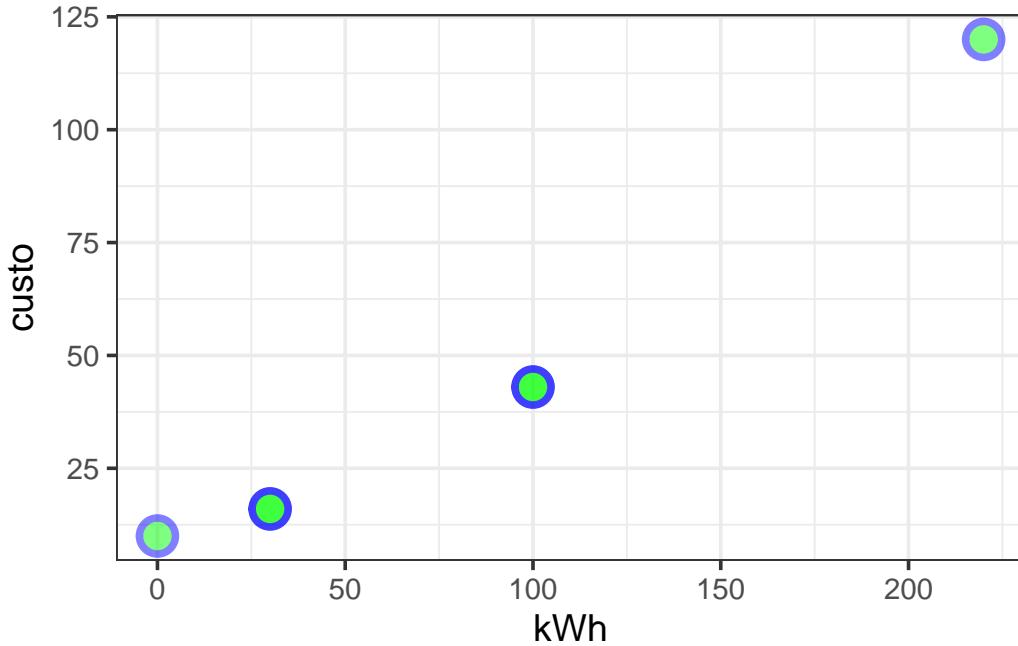
```
ggplot(data=consumo, aes(x=kWh, y=custo, col = "red", size = 5, alpha = 0.5)) +  
  geom_point(color= "blue", size=5, alpha = 0.5, shape = 21, fill = "green", stroke = 2) +  
  labs(x = "kWh",  
       y = "custo") +  
  theme_linedraw()
```



#### 14.1.8 Alterando o tamanho da fonte

Nessa versão dá pra notar que o tamanho dos valores nos eixos, bem como o tamanho de suas etiquetas, deixam um pouco a desejar à estética do gráfico. Aumentando esses valores um pouquinho dentro da função *theme* (o padrão ou *default* é fonte 11)...

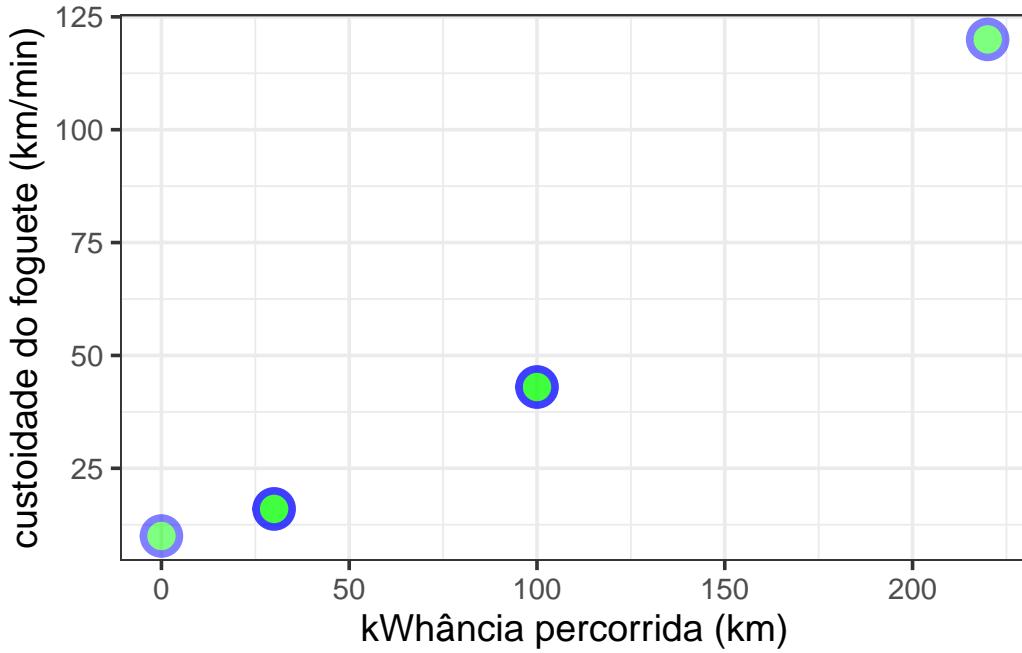
```
ggplot(data=consumo, aes(x=kWh, y=custo, col = "red", size = 5, alpha = 0.5)) +
  geom_point(color= "blue", size=5, alpha = 0.5, shape = 21, fill = "green", stroke = 2) +
  labs(x = "kWh",
       y = "custo") +
  theme_bw(base_size = 14)
```



```
# theme(axis.text = element_text(size = 14),
#       axis.text.x = element_text(size = 14))
```

Se desejar omitir alguma informação do gráfico, basta omiti-la da linha de comando. Por exemplo, se desejarmos retirar título, subtítulo e nota de observação:

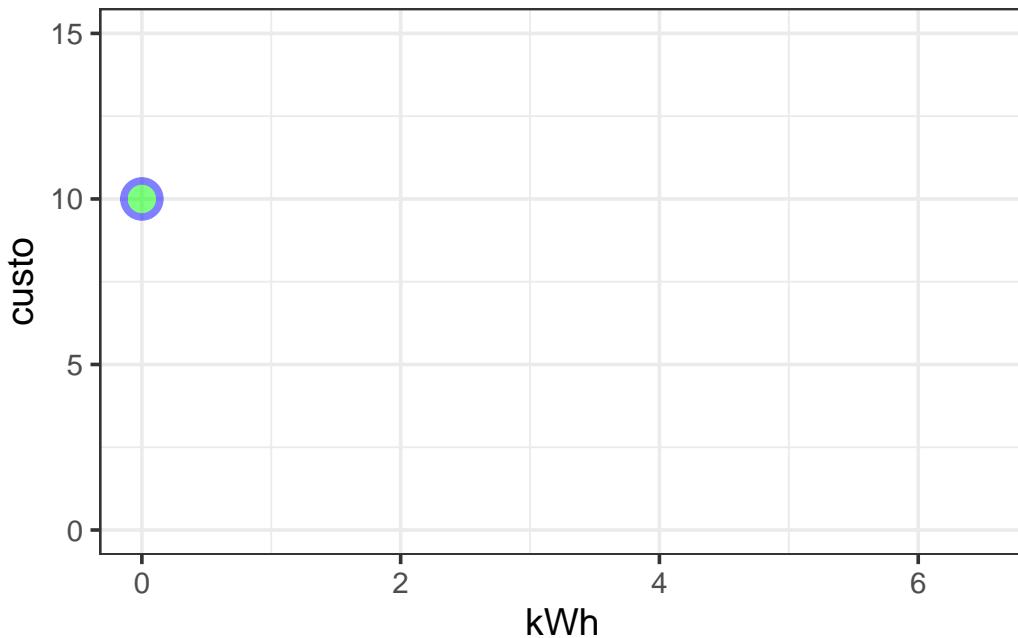
```
ggplot(data=consumo, aes(x=kWh, y=custo, col = "red", size = 5, alpha = 0.5)) +
  geom_point(color= "blue", size=5, alpha = 0.5, shape = 21, fill = "green", stroke = 2) +
  labs(x = "kWhância percorrida (km)",
       y = "custoidade do foguete (km/min)") +
  theme_bw(base_size = 14)
```



#### 14.1.9 Mudando os limites dos eixos

Dá pra melhorar mais um pouco a visualização ?! Sempre dá, claro. Por exemplo, observe que nosso gráfico não inicia na *origem*, ou seja, quando  $x$  e  $y$  tem valor *zero*. Se quisermos alterar então esses eixos pra iniciarem no zero...bom...lá vai mais uma camada !!

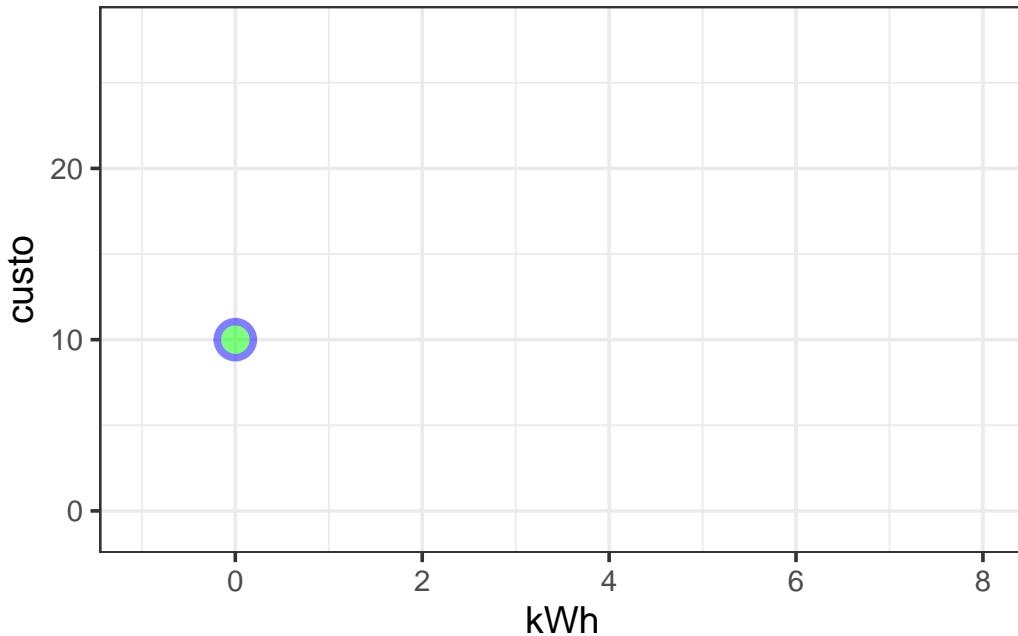
```
ggplot(data=consumo, aes(x=kWh, y=custo, col = "red", size = 5, alpha = 0.5)) +
  geom_point(color= "blue", size=5, alpha = 0.5, shape = 21, fill = "green", stroke = 2) +
  labs(x = "kWh",
       y = "custo") +
  theme_bw(base_size = 14) +
  coord_cartesian(xlim = c(0,6.5), ylim=c(0,15))
```



Apesar de mais “pomposo” tanto o nome como as possibilidades dessa camada `coord_cartesian`, também é possível obter o mesmo efeito apenas com os argumentos `xlim` e `ylim`, e de modo um pouquinho diferente. Pra não repetir o gráfico acima, segue um exemplo com limites kWh nos eixos:

```
ggplot(data=consumo, aes(x=kWh, y=custo, col = "red", size = 5, alpha = 0.5)) +
  geom_point(color= "blue", size=5, alpha = 0.5, shape = 21, fill = "green", stroke = 2) +
  labs(x = "kWh",
       y = "custo") +
  theme_bw(base_size = 14) +
  xlim(-1,8) +
  ylim(-1,28)
```

Warning: Removed 5 rows containing missing values or values outside the scale range  
(`geom\_point()`).



#### 14.1.10 Modelando os dados

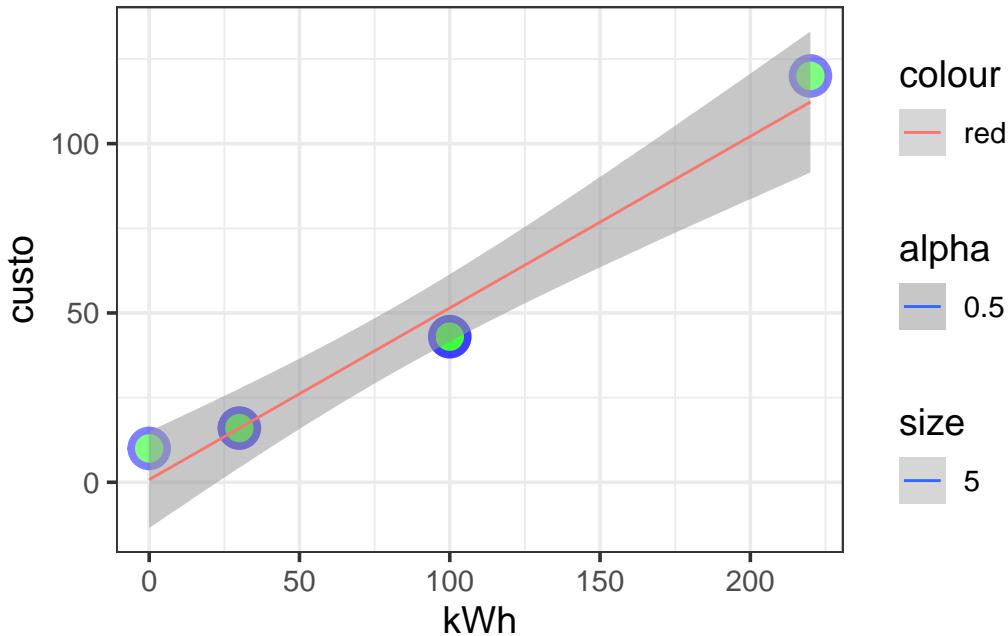
##### 14.1.10.1 Ajuste linear

Agora que nosso gráfico tá (...ou deveria estar) mais incrementado, que tal colocar uma linha de ajuste dos pontos ? Modelos pra isso não faltam, mas optemos inicialmente pelo mais simples: um *ajuste ou regressão linear*. Resumidamente, trata-se de observar se a resposta  $y$  segue uma tendência linear com os valores de  $x$ . E como o `ggplot2` trabalha com camadas, adicionemos essa camada, então:

```
ggplot(data=consumo, aes(x=kWh, y=custo, col = "red", size = 5, alpha = 0.5)) +
  geom_point(color= "blue", size=5, alpha = 0.5, shape = 21, fill = "green", stroke = 2) +
  labs(x = "kWh",
       y = "custo") +
  theme_bw(base_size = 14) +
  geom_smooth(method = 'lm', linewidth = 0.5)
```

Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.  
i Please use `linewidth` instead.

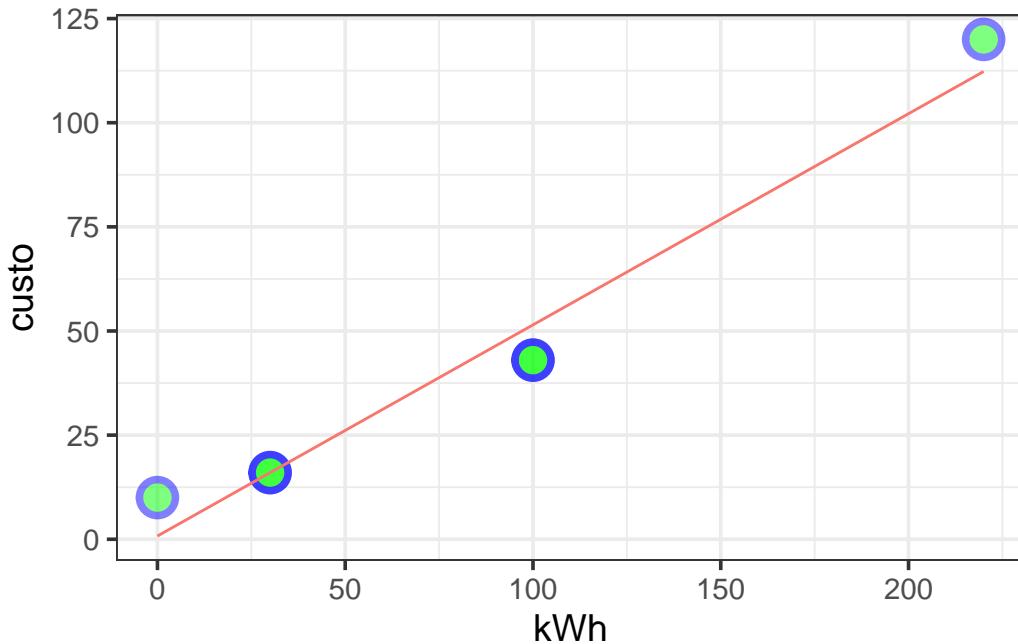
`geom\_smooth()` using formula = 'y ~ x'



Agora ficou chique !! E também indecifrável...ou quase. Indo por partes: o método escolhido foi `lm`, ou modelo linear (*linear model*). O termo poderia ser também substituído por '`y ~x`', uma notação do R que indica a dependência de  $y$  em relação a  $x$ . O argumento `linewidth` sinaliza pra espessura do tracejado. E a banda acidentada indica o *intervalo de confiança* do ajuste (*erro padrão*). Pra retirar essa última, bem como a legenda doida, basta fazer:

```
ggplot(data=consumo, aes(x=kWh, y=custo, col = "red", size = 5, alpha = 0.5)) +
  geom_point(color= "blue", size=5, alpha = 0.5, shape = 21, fill = "green", stroke = 2) +
  labs(x = "kWh",
       y = "custo") +
  theme_bw(base_size = 14) +
  geom_smooth(method = 'lm', linewidth = 0.5, se = FALSE, show.legend = FALSE)

`geom_smooth()` using formula = 'y ~ x'
```



Existem diversos parâmetros que são obtidos pela função `lm`. Dada a natureza de *programação orientada a objeto* do R, esses parâmetros podem ser acessados individualmente em seus objetos. Exemplificando:

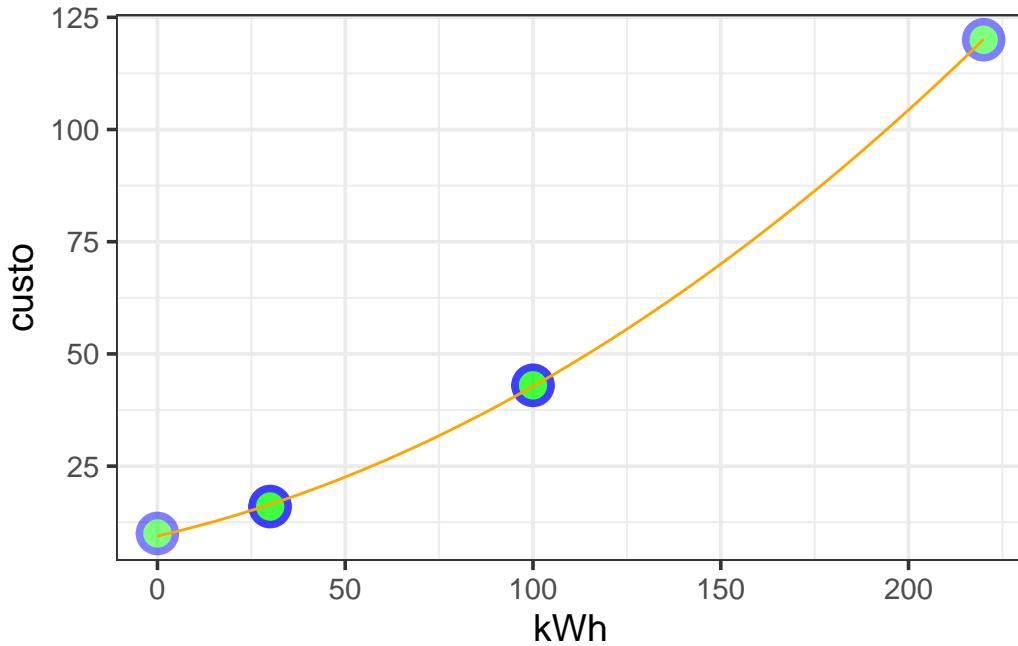
```
attributes(lm)
```

```
NULL
```

#### 14.1.10.2 Ajuste polinomial

Apesar de bem incrementado esse gráfico final, observe que a tendência dos pontos não é propriamente linear. Tá parecendo mais uma curva. E dentre as curvas possíveis, quem sabe uma parábola. Pra isso, basta que acrescentemos qual a `formula` a ser empregada:

```
ggplot(data=consumo, aes(x=kWh, y=custo, col = "red", size = 5, alpha = 0.5)) +
  geom_point(color= "blue", size=5, alpha = 0.5, shape = 21, fill = "green", stroke = 2) +
  labs(x = "kWh",
       y = "custo") +
  theme_bw(base_size = 14) +
  geom_smooth(method = 'lm', formula = y ~ poly(x, 2), linewidth = 0.5, color = "orange", se
```

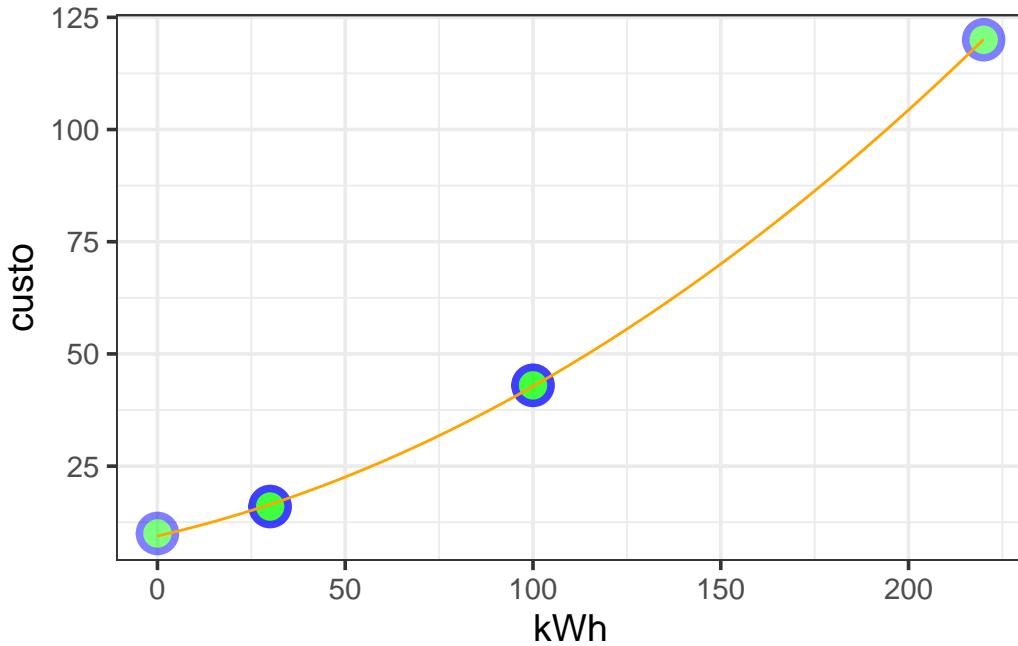


Apesar de um pouco confuso, a expressão `formula = y ~ poly(x, 2)` indica tratar-se de um ajuste para um *polinômio de 2a. ordem* em  $x$ . E, claro, uma “corzinha” adicional no tracejado não faz mal a ninguém !

#### 14.1.11 Salvando o gráfico

Agora, pra se guardar esse gráfico, um tanto trabalhoso, convenhamos, o R permite o salvamento em diversos formatos, como JPG, BMP, TIFF, PDF, PNG, EPS, por exemplo. Dois jeitos de salvar o gráfico: um se dá pelo *menu* de *Plots* do canto inferior esquerdo do *RStudio* (*Plots* → Export). Se quisermos o armazenamento por linha de comando, por outro lado, basta utilizar a função `ggsave` logo após a criação do gráfico.

```
ggplot(data=consumo, aes(x=kWh, y=custo, col = "red", size = 5, alpha = 0.5)) +
  geom_point(color= "blue", size=5, alpha = 0.5, shape = 21, fill = "green", stroke = 2) +
  labs(x = "kWh",
       y = "custo") +
  theme_bw(base_size = 14) +
  geom_smooth(method = 'lm', formula = y ~ poly(x, 2), linewidth = 0.5, color = "orange", se
```



```
# Salvando o gráfico...
```

```
ggsave(filename = "plot.consumo.png", dpi = 300, width = 4, height = 5)
```

Se você não especificou onde salvar o gráfico, veja no diretório raiz do computador. Explicando brevemente a função, `dpi` representa a resolução do gráfico, enquanto que `width` e `height` especificam o tamanho.

Mas também dá pra salvar o gráfico sem qualquer especificação. Exemplificando pra um arquivo *PDF*:

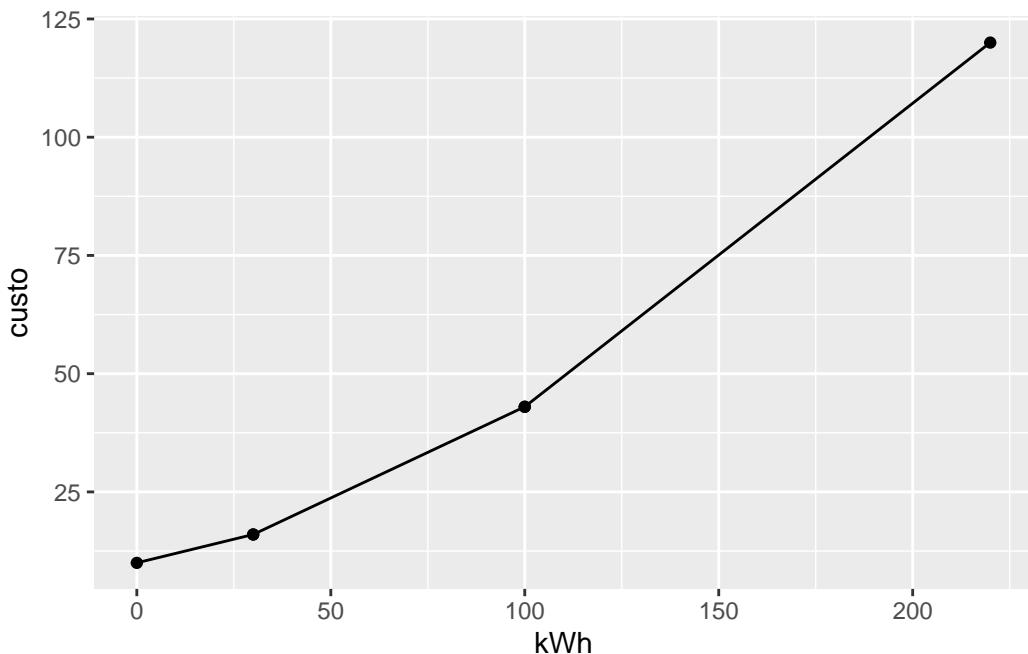
```
ggsave(filename = "plot.consumo.pdf")
```

Saving 5.5 x 3.5 in image

#### 14.1.12 Simplificando a construção de um gráfico

Bom, ainda que o gráfico acima tenha um certo ar de sofisticação, não é sempre que se deseja isso. Na maior parte das vezes, o que se quer é plotar os dados e observar alguma tendência. E, pra isso, não é necessário todas essas camadas, funções e argumentos. Para se plotar algo “parecido” com o gráfico acima, mas sem toda essa pompa, basta fazermos:

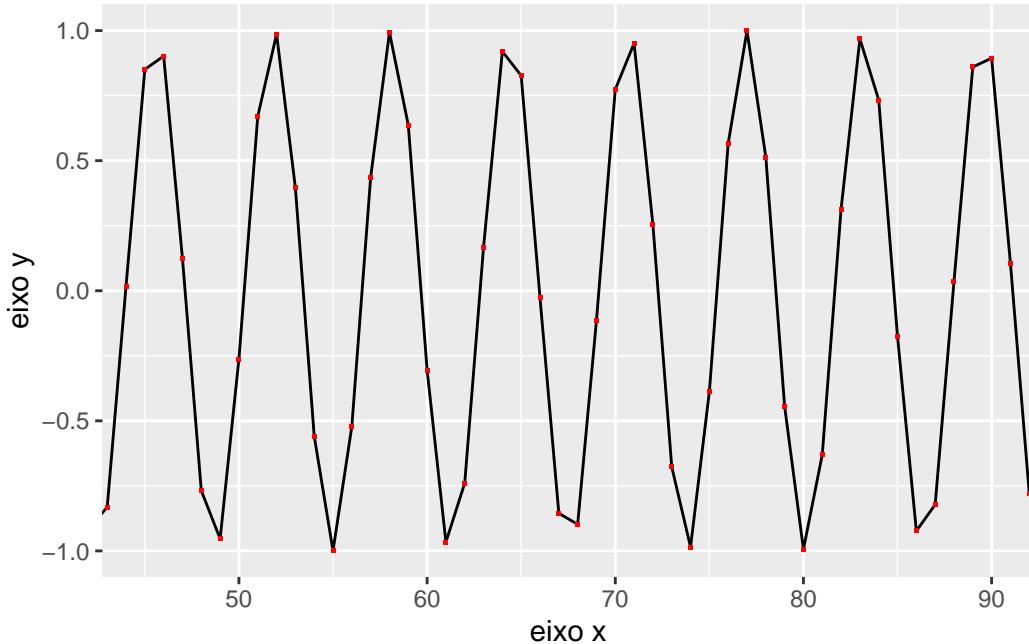
```
ggplot(data=consumo, aes(x=kWh, y=custo)) +
  geom_point() +
  geom_line()
```



#### 14.1.13 Elaborando um gráfico apenas com vetores

O pacote `ggplot2` é normalmente empregado para a criação de gráficos a partir de um *dataset*. Não obstante, é possível produzir também um gráfico apenas com vetores, tal como segue:

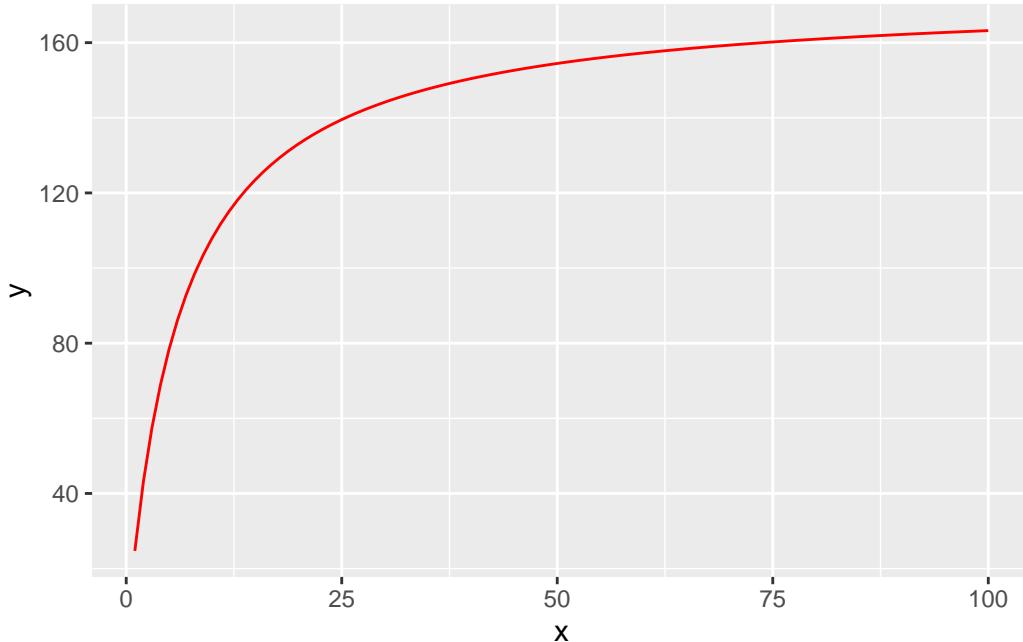
```
x <- 1:360
y = sin(x)
library(ggplot2)
ggplot( data = NULL ,
        mapping = aes( x = x , y = y ) ) +
  geom_line() +
  geom_point(shape = 15, col = "red", size = 0.5) +
  labs(x="eixo x", y="eixo y") +
  coord_cartesian(xlim=c(45,90))
```



#### 14.1.14 Simulação & função em ggplot2

Como o sistema base, o `ggplot2` possui também uma função para a produção de tendências por equações introduzidas, `geom_function`. Exemplificando:

```
require(ggplot2)
ggplot(data.frame(x = 1:100), aes(x)) +
  geom_function(fun=function(x) 173*x/(6+x), colour = "red")
```



#### 14.1.15 ggplot2 para dados multivariados

Como explicado acima, o `ggplot2` trabalha *exclusivamente* com dados em formato *Long*, ou seja, uma variável para cada quantidade. Dessa forma, os subconjuntos de uma variável (ou nível) devem situar-se nessa única variável. Ilustrando:

```
tempo <- c(rep(1:4, 4), 4)
resposta <- c(1,2,3,4,2,4,6,8,10,14,18,22,24,30,36,42)
niveis <- c(rep("A",4),rep("B", 4), rep("C", 4), rep("D", 4)) # criação de dataframe com níveis

df <- data.frame(tempo, resposta, niveis); df
```

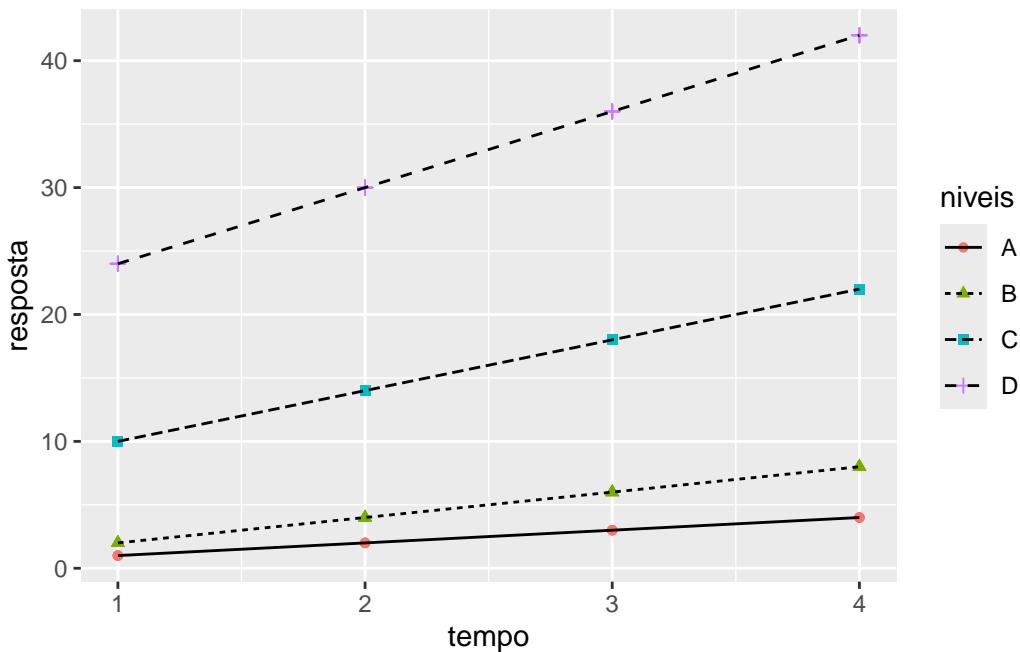
	tempo	resposta	niveis
1	1	1	A
2	2	2	A
3	3	3	A
4	4	4	A
5	1	2	B
6	2	4	B
7	3	6	B
8	4	8	B
9	1	10	C

10	2	14	C
11	3	18	C
12	4	22	C
13	1	24	D
14	2	30	D
15	3	36	D
16	4	42	D
17	1	1	A
18	2	2	A
19	3	3	A
20	4	4	A
21	1	2	B
22	2	4	B
23	3	6	B
24	4	8	B
25	1	10	C
26	2	14	C
27	3	18	C
28	4	22	C
29	1	24	D
30	2	30	D
31	3	36	D
32	4	42	D
33	1	1	A
34	2	2	A
35	3	3	A
36	4	4	A
37	1	2	B
38	2	4	B
39	3	6	B
40	4	8	B
41	1	10	C
42	2	14	C
43	3	18	C
44	4	22	C
45	1	24	D
46	2	30	D
47	3	36	D
48	4	42	D
49	1	1	A
50	2	2	A
51	3	3	A
52	4	4	A

53	1	2	B
54	2	4	B
55	3	6	B
56	4	8	B
57	1	10	C
58	2	14	C
59	3	18	C
60	4	22	C
61	1	24	D
62	2	30	D
63	3	36	D
64	4	42	D

```
library(ggplot2)

ggplot(data=df, aes(x=tempo, y=resposta)) +
  geom_point(aes(shape=niveis, color=niveis)) +
  geom_line(aes(linetype=niveis))
```



Note que cada elemento estético pode ser configurado para cada *geom*. Alternativamente, pode-se conferir os elementos estéticos combinando-os para o gráfico geral, bem como para cada camada:

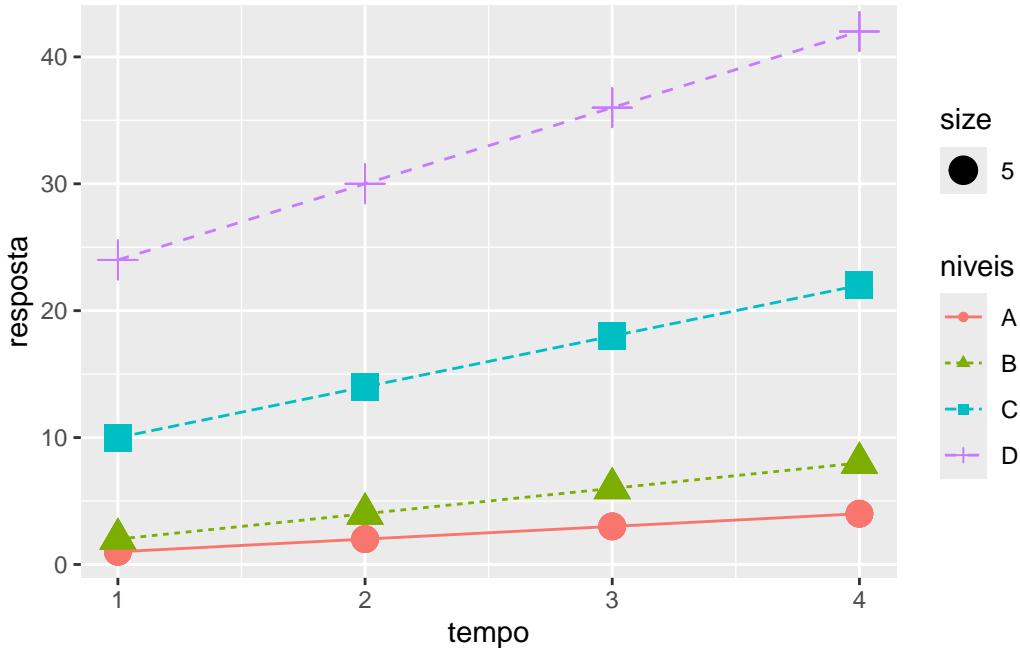
```

tempo <- c(rep(rep(1:4, 4),4))
resposta <- c(1,2,3,4,2,4,6,8,10,14,18,22,24,30,36,42)
niveis <- c(rep("A",4),rep("B", 4), rep("C", 4), rep("D", 4)) # criação de dataframe com níveis

df <- data.frame(tempo, resposta, niveis)

library(ggplot2)
ggplot(data=df, aes(x=tempo, y=resposta, shape=niveis, color=niveis)) +
  geom_point(aes(size=5)) +
  geom_line(aes(linetype=niveis))

```

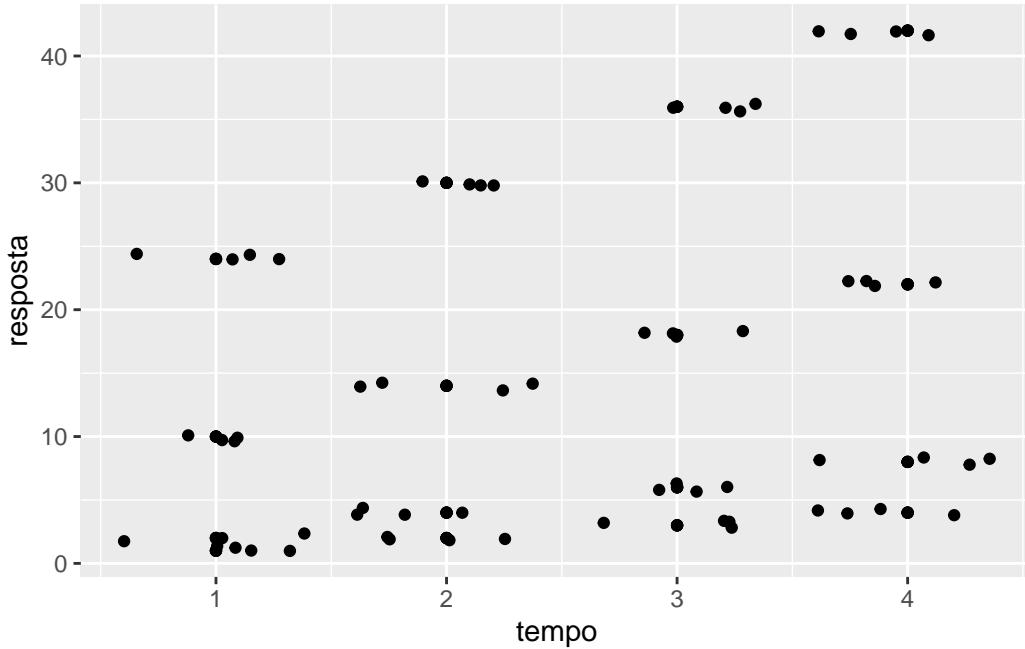


Para um conjunto com visual sobreposição de pontos, pode-se também empregar a função `jitter`, como no `graphics`:

```

ggplot(data=df, aes(x=tempo, y=resposta)) +
  geom_point() +
  geom_jitter()

```

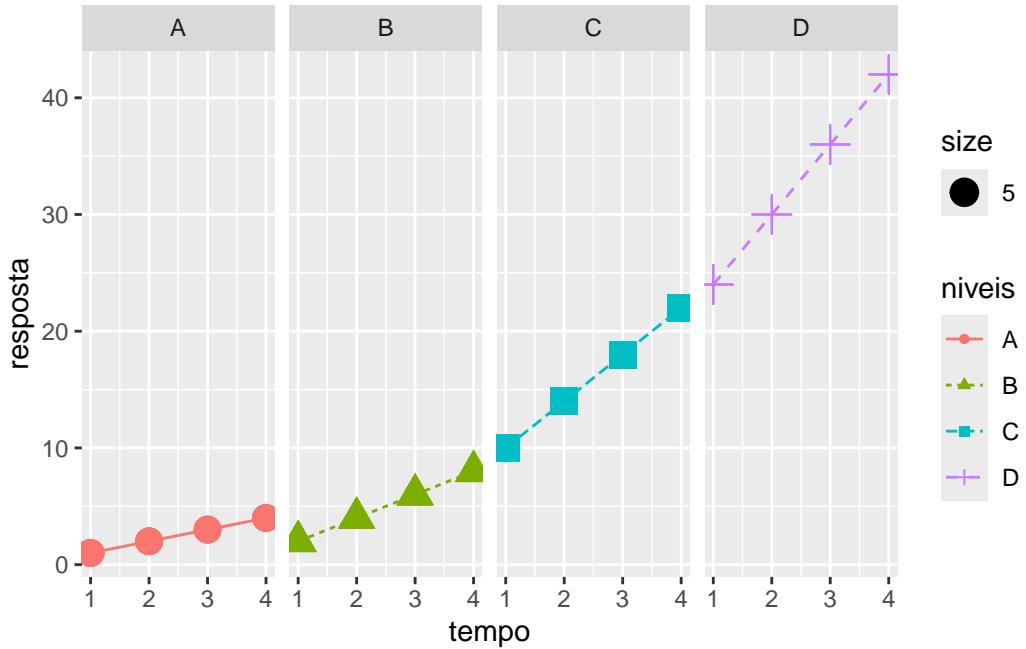


#### 14.1.16 Paineis (*faceting*)

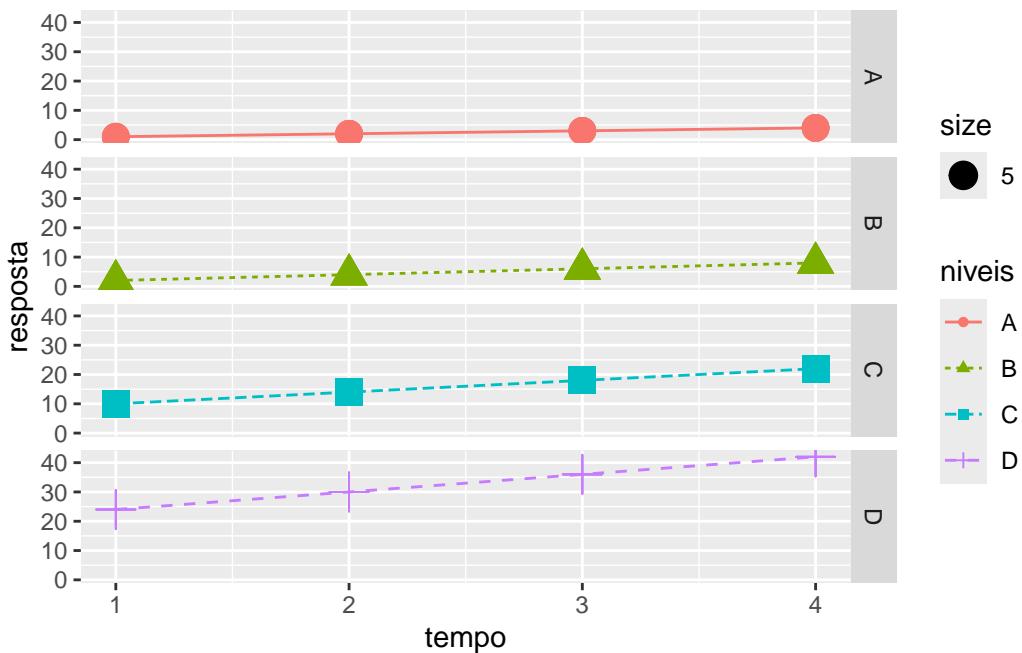
Como um recurso estético e analítico significativo, o *faceting* permite ao `ggplot2` a visualização multivariada em painéis. Seguem exemplos para o *dataset* acima.

```
tempo <- c(rep(1:4, 4),4)
resposta <- c(1,2,3,4,2,4,6,8,10,14,18,22,24,30,36,42)
niveis <- c(rep("A",4),rep("B", 4), rep("C", 4), rep("D", 4)) # criação de dataframe com níveis

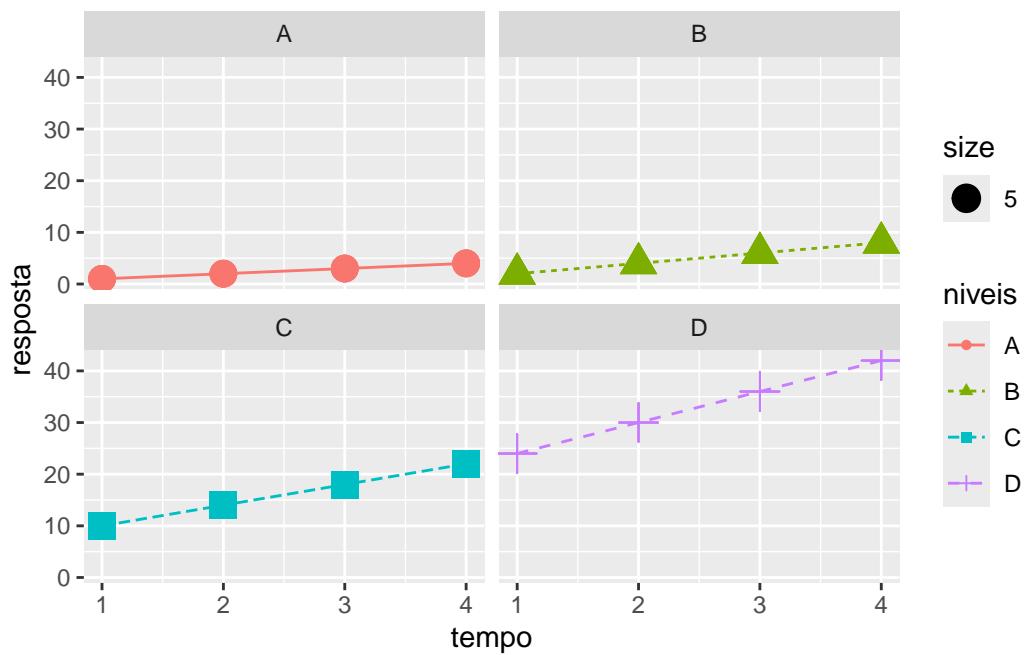
df <-data.frame(tempo, resposta, niveis)
ggplot(data=df, aes(x=tempo, y=resposta, shape=niveis, color=niveis)) +
  geom_point(aes(size=5)) +
  geom_line(aes(linetype=niveis))+ 
  facet_grid(cols=vars(niveis)) # painéis verticais (útil para visualização da ordenada)
```



```
ggplot(data=df, aes(x=tempo, y=resposta, shape=niveis, color=niveis)) +
  geom_point(aes(size=5)) +
  geom_line(aes(linetype=niveis))+
  facet_grid(rows=vars(niveis)) # painéis horizontais (útil para visualização na abscissa)
```



```
ggplot(data=df, aes(x=tempo, y=resposta, shape=niveis, color=niveis)) +
  geom_point(aes(size=5)) +
  geom_line(aes(linetype=niveis))+
  facet_wrap(vars(niveis)) # painéis divididos (útil na visualização equilibrada de abscissa)
```



## **15 Animando gráficos do ggplot2 - o pacote gganimate**

## **16 Gráficos interativos - o pacote plotly**

## **17 Mapas interativos - o pacote leaflet**