Octave

Fernando Anselmo
http://fernandoanselmo.orgfree.com/wordpress/

Versão 1.1 em 26 de julho de 2020

Resumo

Octave [1], é um sistema interativo completo para cálculos numéricos com colaboradores em todo o mundo. É uma alternativa *Open Source* ao MATLAB. Disponível sob a **GPL** (Licença Pública Geral do GNU) possui ferramentas extensivas para a resolução de problemas lineares numéricos comuns de álgebra, cálculo de equações não-lineares, funções ordinárias, polinômios, integrais e integração numérica de equações diferenciais ordinárias e diferenciais-algébricas. Então não serve para o Cientista de Dados? Nesta desejo provar o contrário e mostrar que podemos utilizar o Octave como ferramenta de auxílio ao nosso trabalho.

1 Parte inicial

Foi escrito por **John W. Eaton** em 1988 com o objetivo de ser usado como referência para a apostila no curso "Projetos de Reatores Químicos"de *James B. Rawlings*, da Universidade *Wisconsin-Madison* e *John G. Ekerdt* da Universidade do Texas. Ou seja, originalmente idealizado como uma ferramenta especializada na solução de problemas para reatores químicos.

Figura 1: *Logo do Octave*

GNU/Octave (nesta trataremos somente por **Octave**) atualmente é utilizado de modo bem mais amplo, tanto no meio acadêmico com em aplicações comerciais. Existe um relacionamento profundo deste com o **MATLAB** (de MATrix LABoratory)que combina um ambiente desktop para análise iterativa, processos de projeto e uma linguagem de programação para expressar matrizes diretamente. Porém por ser um software proprietário cuja licença tem um valor não acessível ao usuário comum, a comunidade GNU resolveu adotar um substituto com uma grande semelhança dos comandos do Matlab e por aceitar os arquivos de mesmo formato.

Octave é uma linguagem voltada para computação numérica. Porém sua interface de comandos está bem mais parecida com o RStudio, sendo possível plotar gráficos e ler bancos de dados, imagens ou mesmo interagir com o Arduino. A grande vantagem é a disponibilização de pacotes que são criados pela comunidade e amplia os recursos do Octave para diversas experiências. Pode ser utilizado também em modo script (textos de programação).

2 No Ambiente Octave

Após realizar a instalação do Octave e executá-lo, temos a seguinte tela:

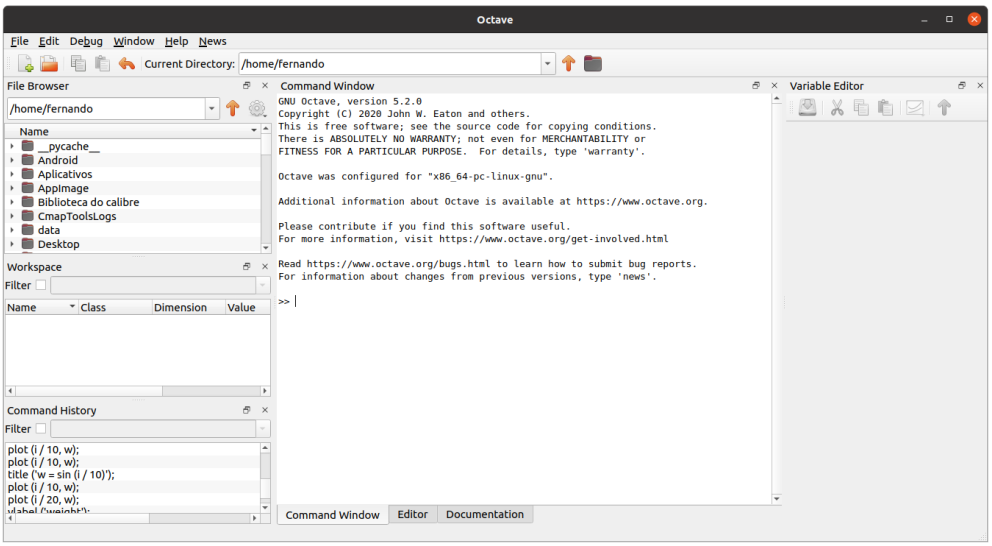

Figura 2: *Ambiente do Octave*

Bem parecido com o **RStudio**, na parte central se encontra a **Janela de Comandos** e é nesta que basicamente trabalhamos. Podemos obter um auxílio a qualquer comando:

```
>> help comando
```

Também existe a possibilidade de usar as setas para cima/baixo como forma de acessar os comandos já inseridos (é util quando desejamos repetir um comando). Começamos pelas operações matemáticas, digitar:

```
>> 6 + 5
```

E ao pressionarmos **ENTER** recebemos como resposta uma variável *ans* com o valor 11. Na janela lateral a esquerda foi criada essa variável e a qualquer momento pode ser utilizada, digitar:

```
>> ans * 2
```

E agora *ans* possui o valor 22, ou seja, cada operação matemática que realizamos alteramos o valor dessa variável que também pode ser utilizada em nossos cálculos. Além da adição (+), multiplicação (∗), divisão (/) e subtração (−) também são permitidas operações primárias como divisão indireta (\) que é realizada da direita para esquerda, ou seja, divisor é o primeiro elemento:

```
>> 10 \ 5
```

Que é equivalente a:

```
>> 5 / 10
```

Também podemos usar a potencia (ˆ):

```
>> 10 ^ 2
```

Ou seja, 10 elevado a 2. Além da variável *ans* que é criada e gerenciada automaticamente (outras são por exemplo: *pi*, *version* ou *computer*). Podemos criar nossas variáveis através do comando de atribuição (=):

```
>> a = 10 + 2 * 3
```

Existem certas regras para nomear as variáveis. Os nomes devem ser iniciados por letras, não podem conter espaços nem caracteres de pontuação, diferença entre letras maiúsculas de minúsculas e alguns nomes são reservados. Na janela a esquerda aparece todas as variáveis criadas, neste caso **a** com o valor 16. Não deveria ser 36? É respeitado a precedência matemática dos operadores assim a multiplicação é realizada antes da adição.

Com ; podemos agrupar comandos na mesma linha:
```
>> a = 2; b = 4; a * b
```

Temos a variável **a** com o valor 2, **b** com o valor 4 e **ans** com o valor 8. Listar todas as variáveis disponíveis:
```
>> who
```

whos lista além das variáveis informações adicionais sobre elas. Remover uma variável:
```
>> clear a
```

Ou todas as variáveis:
```
>> clear all
```

ATENÇÃO: Limpar a Janela de Comandos

Digitar o comando: `clc` ou simplesmente pressionar $Ctrl + L$. Outros atalhos interessantes são $Ctrl + U$ limpa a linha atual ou $Ctrl + K$ limpa da posição do cursor para frente.

2.1 Formatação e precisão numérica

Não existe a distinção entre valores inteiros ou decimais:
```
>> a = 5.0; b = 5;
```

Observamos que tanto a variável **a** quanto **b** possuem o mesmo valor **5**, e pertencem a mesma classe *double*. Normalmente os números decimais são exibidos com quatro dígitos, se esses são significativos:
```
>> a = 5.01234
```

E agora temos o valor em **a** de: 5.0123, o que aconteceu com o 4? Não perdemos nada, apesar de exibir-los dessa forma, internamente a precisão é bem maior. O comando:
```
>> format long
```

Mostra que **a** realmente contém o valor 5.012340000000000 ou seja, são 15 dígitos de precisão. Porém **b** permanece como 5 (por ter sido criada sem casas decimais significativas). O comando:
```
>> format bank
```

Trabalha em formato monetário com 2 casas decimais, isso afeta todas as variáveis (inclusive **b**). O comando:
```
>> format
```

Retorna a forma como são mostrados originalmente. Essas são as opções mais usuais, podemos ver outras com:
```
>> help format
```

Outra forma que pode assustar ao leigo é a expressão científica, por exemplo:
```
>> 0.000000000000034
```

A variável **ans** contém o valor $3.4000e-14$, que corresponde a expressão: `3.4 * 10 ^ -14`. Isso é usado como forma de simplificar a leitura.

2.2 Outros tipos

Números complexos são definidos com um parte imaginária através da variável pré-definida i:
```
>> c = 3 + 2i
```

Not a Number (NaN) é o resultado de operações que não produzem um numérico bem definido. como por exemplo:
```
>> 0 / 0
```

Já se usarmos um valor diferente de 0 no denominador, geramos Infinito (Inf):
```
>> 10 / 0
```

3 Vetores e Matrizes

Um vetor nada mais é que uma matriz com dimensão igual a 1:
```
>> vetA = [1, 2, 3, 4, 5]
```

Outra forma de gerarmos o mesmo vetor:
```
>> vetA = 1:5
```

O primeiro valor é o inicial e o segundo o final, ou ainda:
```
>> vetA = 1:1:5
```

Como por padrão o passo é 1 (elemento central), se desejar por exemplo somente os número pares entre 1 e 20:
```
>> pares = 2:2:20
```

Neste caso iniciamos no primeiro par conhecido (2) e saltamos de 2 em 2. Podemos realizar quaisquer operações matemáticas entre vetores e um valor qualquer:
```
>> vetA * 2
```

Porém entre 2 vetores, somente adição e subtração, se tiverem o mesmo número de elementos:
```
>> vetB = [2, 6, 8, 9, 10]
>> vetA + vetB
```

3.1 Matrizes

Matrizes são criadas colocando um ; para separar os vetores criando assim várias 2 dimensões:
```
>> A = [0, 1, 2; 3, 4, 5]
```

Só por uma questão de padronização matrizes são criadas com letras maiúsculas. Ao criarmos uma segunda matriz:
```
>> B = [5, 4, 3; 2, 1, 0]
```

Podemos realizar operações como adição:
```
>> A + B
```

Ou subtração:
```
>> A - B
```

No caso de multiplicação, pode ser realizada por um escalar, ou seja, multiplicar todos os elementos por um valor:
```
>> A * 3
```

Porém a multiplicação entre matrizes ambas precisam ser quadráticas (ou seja o mesmo número de linhas e colunas):
```
>> A = [5, 4; 2, 1]
>> B = [4, 3; 1, 0]
>> A * B
```

Mesma regra para divisão de matrizes:
```
>> A / B
```

O cálculo da transposta de uma matriz qualquer, isto é, transformar as linhas em colunas:
```
>> A'
```

Existem muitas outras operações que podemos realizar com matrizes, consultar a documentação oficial do Octave para vê-las.

3.2 Matrizes Especiais

Submatriz é quando obtemos uma nova matriz de uma outra:
```
>> A = [2, 4, 6; 8, 10, 12; 14, 16, 18]
>> B = A(2:3, 2:3)
```

A matriz **B** é obtida do intervalo das linhas do primeiro parâmetro e das colunas do segundo.

Matriz com todos os valores iguais a 1:
```
>> B = ones(3)
```

Gera uma matriz 3x3 com todos valores iguais a 1. Ou então:
```
>> B = zeros(3)
```

Matriz 3x3 com todos valores iguais a 0. Temos ainda algumas funções especiais:

- $eye(size(matriz))$ mostra a matriz identidade.
- $diag(matriz)$ extrai a diagonal da matriz.
- $rand(tamanho)$ matriz de números aleatórios (valores maiores que 0.0 e menores que 1).
- $inv(matriz)$ matriz inversa.

Matrizes são importantes, devemos lembrar que *DataFrames* nada mais são do que matrizes organizadas.

4 Gráficos

Agora que já sabemos tudo sobre o básico do Octave vamos nos aprofundar em seus gráficos, esses possuem diversos comandos auxiliares, tais como:

- *title('texto')*: Título do gráfico.
- *xlabel('texto')*: Legenda do eixo X.
- *ylabel('texto')*: Legenda do eixo y.
- *text(x, y, 'texto')*: Escrever um texto na posição X,y.
- *gtext('texto')*: Escrever um texto no ponto determinado pela posição do mouse no gráfico.
- *legend('textoDado1', 'textoDado2', ..., 'textoDadoN', 'location', 'posicao')*: Colocar as legendas na ordem que os dados forma plotados. Sua posição é definida pelos eixos cartesianos (em inglês): *north*, *south*, *east*, *west*, *northeast*, *northwest*, *southeast* ou *southwest*.
- *grid on/off*: Adicionar ou retirar grades no gráfico.
- *hold off/on*: Utilizado para sobrepor outro gráfico na mesma figura. Opção off mantém o gráfico antigo até que a opção on seja usado.

Vetores normalmente são mais usados para visualizar valores de forma gráfica, por exemplo:

```
>> x = linspace(0, 2 * pi)
>> y = sin(x)
>> plot(x, y)
```

A função *linspace()* é correspondente a `[0: 2 * pi / 100: 2 * pi]`, *sin()* calcula o seno (também podemos usar *cos*() para o **cosseno** ou *tan*() para **tangente**) de cada um dos elementos do vetor e *plot()* mostra de forma gráfica, assim obtemos:

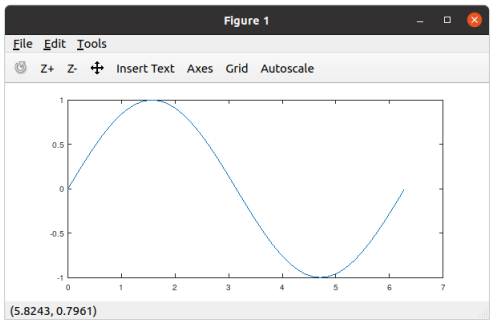

Figura 3: *Seno de X*

A função *plot(varX, varY)* gera gráficos lineares com os parâmetros **varX** variável independente e **varY** a dependente. Podemos adicionar mais séries plot(varX1, varY1, varX2, varY2) que gera dois gráficos combinados:

```
>> plot([1, 1.5, 3], [5, 3, 5], [1, 2.5, 3], [5, 7, 5])
```

Obtemos como resultado:

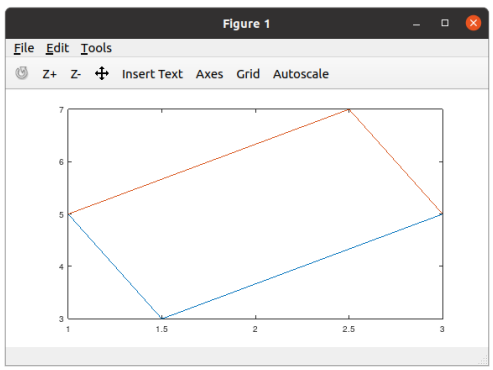

Figura 4: *Gráficos combinados*

Essas séries podem continuar indefinidamente. Vamos plotar o mesmo gráfico para senos (já visto anteriormente) com a adição do cosseno com algumas dessas opções:

```
>> x = linspace(0, 2 * pi)
>> plot(x, sin(x), 'r-', x, cos(x), 'b-')
>> legend('Seno', 'Cosseno', 'location', 'southwest')
>> title('Funções Seno e Cosseno de X')
>> xlabel('Amplitude')
>> ylabel('Frequência')
>> grid on
```

Obtemos como resultado:

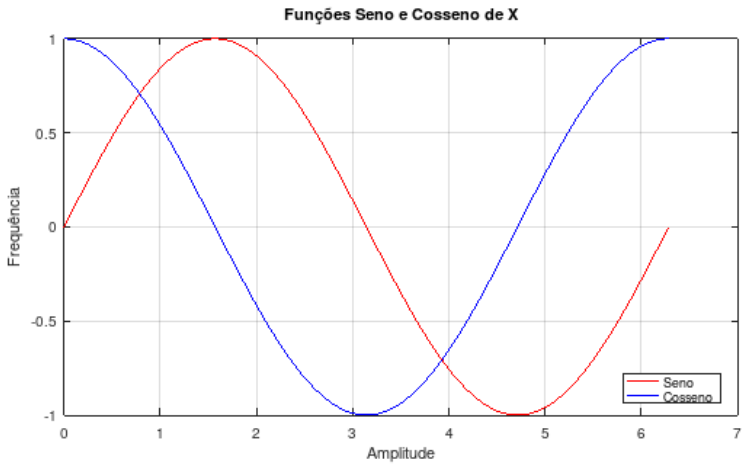

Figura 5: *Gráficos com Parâmetros*

Foram colocados um terceiro parâmetro a cada série da função *plot()* esses podem ser:

- Cor: **y** (amarelo), **b** (azul), **c** (azul claro), **w** (branco), **r** (vermelho), **k** (preto), **m** (roxo) ou **g** (verde).
- Linha: **-** (sólida por padrão), **.** (pontos), *****, **x**, **o** (diamantes) ou **v** (triângulos)

Para obtermos maiores informações sobre os parâmetros digitar `help axis`. Assim podemos dizer que esta função gráfica pode conter o seguinte: *plot(serie1, serie2, ..., serieN)*. Sendo que cada série contém os valores de X, y e parâmetros.

4.1 Barras e Histograma

Barras e Histograma são gráficos bem comuns para quem trabalha de forma ativa com dados, o Octave também permite sua geração de um modo bem simples. Com a função *bar()* temos um gráfico de barras:

```
>> x = 1:10
>> y = randn(1, 10)
>> bar(x, y)
```

O vetor **x** é criado com valores de 1 a 10, e o vetor **y** utiliza a função *randn(linha, coluna)* para gerar um vetor com 10 valores aleatórios. Obtemos como resultado:

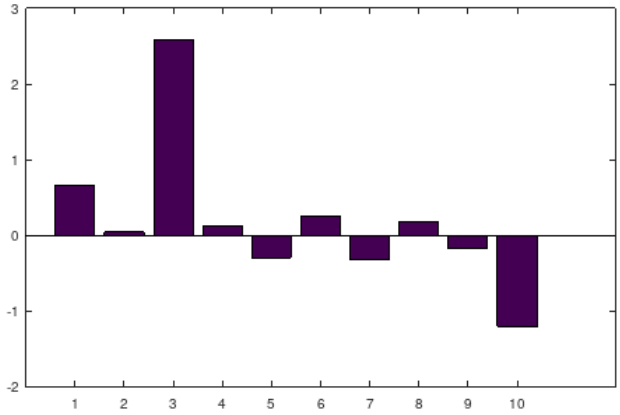

Figura 6: *Gráfico de Barras*

Como estamos com valores aleatórios o resultado pode variar. De mesmo modo podemos produzir um histograma, porém o primeiro valor passado é y (e não x) o segundo é número de bins. Podemos usar simplesmente:

```
>> hist (randn(1, 100), 25)
```

Obtemos como resultado provável:

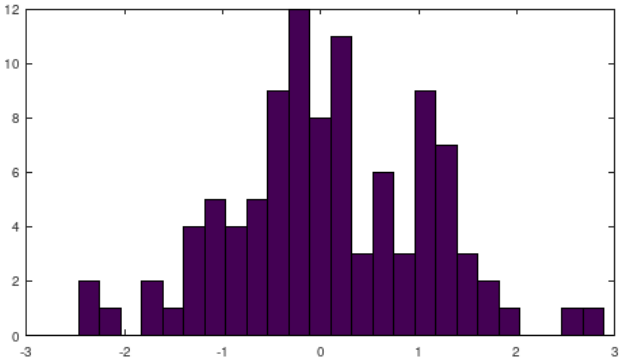

Figura 7: *Histograma*

A função *randn()* gera valores para uma distribuição normal. Podemos ainda utilizar as seguintes funções para gerarmos números aleatórios consistentes:

- *discrete_rnd(valor, probabilidade)*: distribuição discreta uni-variada.
- *empirical_rnd(dados)*: distribuição empírica.
- *rande(n)*: distruição exponencial.
- *randg(n)*: distribuição gama.

Muitas vezes desejamos valores aleatórios inteiros, eles podem ser obtidos com a função *randi()*, por exemplo:

```
>> randi (10, 150, 1)
```

Retorna 150 inteiros no range entre 1 e 10.

4.2 Gráficos Tridimensionais

A função *mesh(varX, varY, varZ)* permite a projeção de gráficos tridimensionais:

```
>> x = [0:  2 * pi / 50:  2 * pi]'
>> y = x
>> z = cos(x) * sin(y')
>> mesh(x, y, z)
```

Cuidado para não esquecer de colocar o operador de transposição ($'$). Neste exemplo temos a função $z = cos(x) \times sin(y)$ no intervalo [0, 2pi] dividido em 50 incrementos. Obtemos como resultado:

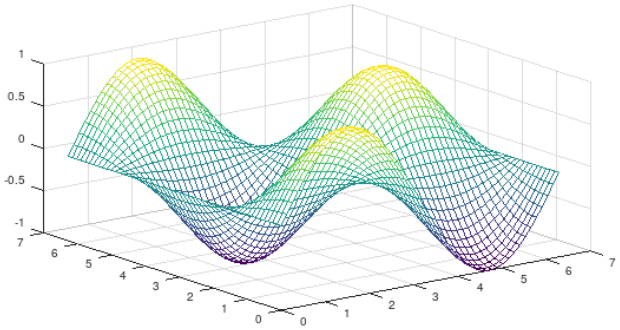

Figura 8: *Gráficos Tridimensional*

4.3 Coordenadas Polares

A função *polar(ângulo, r, 'parâmetros')* permite representar um ponto em coordenadas polares onde **r** é uma semi-reta com origem em ângulo (dados em radianos) e os parâmetros são combinações do tipo de linha e cor. Por exemplo:

```
>> t = 0 :  .01 :  2*pi
>> polar(t, sin(2*t).*cos(2*t), '-r')
```

Obtemos como resultado:

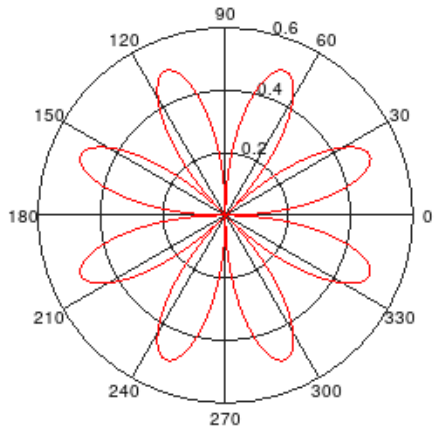

Figura 9: *Cordenadas Polares*

5 Gráficos de Sequência discreta e Escada

Esses tipos são apenas variações do gráfico de Barras, podemos obtê-los do seguinte modo:

```
>> x = 0:0.5:4*pi
>> y = sin(x)
>> subplot(2,1,1)
>> stem(x, y)
>> subplot(2,1,2)
>> stairs(x, y)
```

Criamos com a função *subplot(linha, coluna, indice)* uma área divisória para plotarmos o primeiro gráfico de sequência que é obtido com a função *stem()*. Criamos uma segunda área para o segundo gráfico de escada que é dado pela função *stairs()*. Obtemos como resultado:

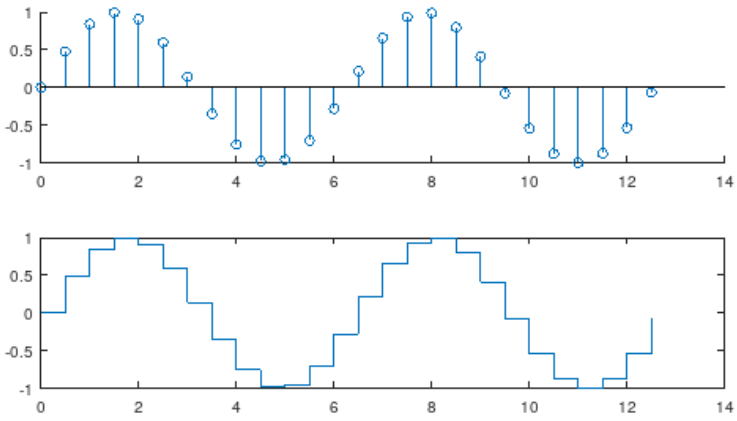

Figura 10: *Gráficos de Sequencia e Escada na mesma figura*

6 Funções

Podemos criar nossas funções personalizadas, fazemos isso para manter uma "biblioteca"de funções que mais necessitamos (como programas prontos para nosso uso). Vamos alternar da **Comand Window** para o **Editor** nas abas inferiores. Uma função tem a forma geral:

```
1 function [saida] = nomeFuncao(parametros)
2   comandos
3 endfunction
```

O arquivo deve ter o mesmo nome da função e possui a extensão ".m". Vamos criar a seguinte função:

```
1 % somaProd
2 % Recebe dois parametros e calcula
3 % a soma e o produto entre os mesmos
4 function [soma, produto] = somaProd(a, b)
5   soma = a + b;
6   produto = a * b;
7 endfunction
```

É boa prática sempre documentarmos nossas funções, tudo o que estiver na frente do carácter de percentual é desprezado. Nossa função recebe dois valores **a** e **b**, retorna duas variáveis com a soma e o produto desses dois valores. Salvamos o arquivo com o nome "somaProd.m". Podemos usá-la como se fosse uma função pré-existente no Octave. Alternar novamente para a **Comand Window**:
`>> [s, p] = somaProd(10, 3)`

A variável **s** recebeu a soma de 10 e 3 enquanto que **p** seu produto.

6.1 Fluxos de Controle

Nas funções são permitidos os seguintes fluxos de controle. Comando de decisão **if**:

```
if condicao
  comandos caso verdadeiro
else
  comandos caso falso
endif
```

O laço de repetição determinada **for** do seguinte modo:

```
for var = inicial:final
  comandos
endfor
```

Ou o laço de repetição indeterminada **while** do seguinte modo:

```
while condicao
  comandos
endwhile
```

Caso seja necessário o comando **break** é utilizado para interromper as repetições do laço. Vejamos um exemplo com esses comandos:

```
function [vetor] = ordenaVetor(vetor)
  troca = false
  while true
    for i = 1:length(vetor) - 1
      if vetor(i+1) < vetor(i)
        x = vetor(i+1)
        vetor(i+1) = vetor(i)
        vetor(i) = x
        troca = true
      endif
    endfor
    if !troca
      break
    else
      troca = false
    endif
  endwhile
endfunction
```

Como parâmetro recebemos um vetor para ordena-lo numericamente, criamos uma variável de controle para as trocas. Nosso laço indeterminado **while** (tem esse nome pois não sabemos quantas vezes seus comandos serão repetidos) será sempre repetido (em tese temos aqui um laço eterno). Entramos no laço determinado **for** (tem esse nome pois sabemos quantas vezes será executado do valor inicial da variável de entrada até o valor final) lemos da primeira posição do vetor (cuidado com outras linguagens pois aqui a primeira posição é 1) até sua última (dada pelo método *lenght()*) menos 1 (pois usaremos para comparar a posição posterior). No comando de decisão **if** verificamos se a posição posterior é menor que a anterior em caso verdadeiro realizamos uma troca de valores e indicamos para a variável *troca* que ocorreu. Ao término do laço **for** verificamos o conteúdo de **troca**, se for falso (ou seja nenhuma troca foi realizada) interrompemos o laço **while**, caso contrário voltamos o valor de **troca** para false e repetimos todo o processo.

Executamos essa função da seguinte maneira:
```
>> x = ordenaVetor([4, 2, 5, 1])
```

E como resultado teremos em **x** o valor ordenado:
```
[1, 2, 4, 5]
```

7 Pacotes adicionais

Por ser um software em constante desenvolvimento, vários outros pacotes adicionais podem ser instalados. Podemos ver uma lista completa desses em `https://octave.sourceforge.io/packages.php`. Antes de começarmos a instalar vamos verificar quais pacotes temos a disposição:
```
>> pkg list
```

Provavelmente a resposta será que não possuímos nenhum pacote instalado. Um boa biblioteca para instalarmos chama-se *image*, localize e baixe o arquivo compactado (formato **tar.gz**). O editor está apontado para um determinado diretório de trabalho, o arquivo deve estar nesse diretório.

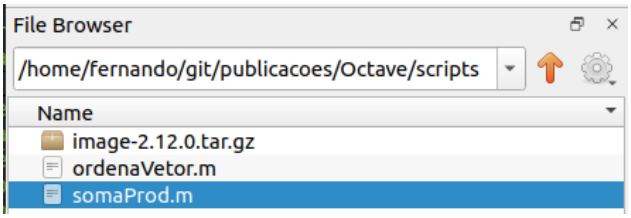

Figura 11: *Diretório de Trabalho do Octave*

Uma vez que temos isso acertado, instalamos com o comando:
```
>> pkg install image-2.12.0.tar.gz
```

ATENÇÃO: No Ubuntu

Para o Ubuntu é necessário instalarmos o aplicativo *liboctave-dev* para executarmos este comando. No terminal digite o comando: `$ sudo apt install liboctave-dev`

Com a conclusão da instalação, temos as seguintes funções a nossa disposição:

- *imread*: realizar a leitura da imagem.
- *image*: visualizar uma matriz como uma imagem.

- *imattributes*: atributos da imagem.
- *rgb2gray*: converter uma imagem colorida (RGB) em tons de cinza.
- *figure*: abrir uma nova janela gráfica.
- *colormap*: permitir a definição de um mapa de cores
- *imfinfo*: retornar uma estrutura com diversas informações sobre a imagem.

Outras funções podem ser vistas consultando a documentação do pacote. Ao realizarmos a sua instalação esse já foi carregado, porém se sairmos e ao entrarmos novamente no Octave este não será reconhecido, é necessário carregá-lo com o comando:

```
>> pkg load image
```

Colocamos uma imagem no diretório de trabalho e a verificamos detalhes sobre ela:

```
>> imfinfo("azaleia.jpg")
```

Obviamente "azaleia.jpg"é o nome da imagem. Para carregá-la:

```
>> azaleia = imread("azaleia.jpg");
```

Observe o ";"ao término do comando isso fara suprimir a saída desse ou do contrário levará um bom tempo para ler. Os próximos comandos permitem visualizarmos a imagem:

```
>> figure(1)
>> image(azaleia)
```

O primeiro abre a janela de visualização e o segundo mostra a imagem:

Figura 12: *Imagem visualizada pelo Octave*

Um erro muito comum de quem é iniciante ocorre quando convertemos a imagem para tons de cinza:

```
>> azaleiag = rgb2gray(azaleia);
>> figure(2)
>> image(azaleiag)
```

Obtemos como resultado:

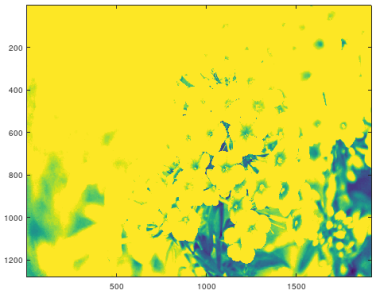

Figura 13: *Imagem em tons de cinza*

Precisamos "avisar"para o visualizador que agora deve trabalhar em tons de cinza não mais de modo colorido:

```
>> colormap(gray(256))
```

Obtemos como resultado:

Figura 14: *Imagem em tons de cinza correto*

Podemos inclusive plotar a frequência e o tom de suas cores em um gráfico:

```
>> figure(3)
>> plot(hist(azaleiag(:), 0:255))
```

E obtemos como resultado:

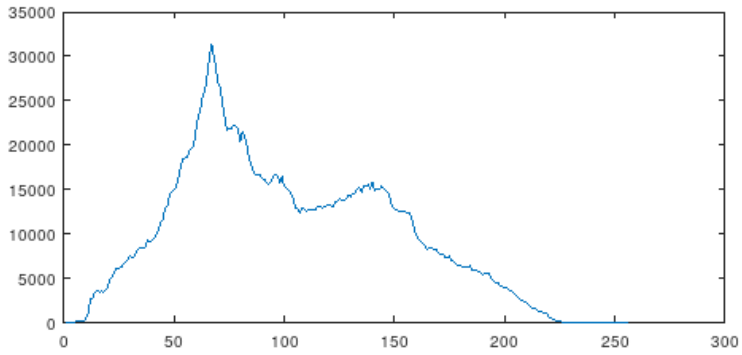

Figura 15: *Histograma da Imagem*

7.1 Dataframe

Dataframes provêm acesso a manipulações para arquivos de dados (normalmente em formato CSV - colunas separadas por vírgulas), são suportados por linguagens como Python (Pandas) e R. Veremos que o tratamento desse via Octave não é tão complicado assim e obtemos os mesmos recursos.

Agora que já aprendemos o processo completo para a instalação de um pacote, vamos instalar este de uma forma bem mais simples e usual:

```
>> pkg install -forge dataframe
```

Este modo busca o pacote diretamente do SourceForge. Vamos tratar de uma base bem conhecida chamada *mtcars.csv* (está disponível no GitHub [4] na pasta **Octave**). Iniciamos com a leitura do pacote:

```
>> pkg load dataframe
```

E ler a base: `>> df = dataframe('mtcars.csv')`

Essa base contém 32 modelos de carros com os seguintes atributos: Nome, quilometragem, número de cilindros, deslocamento (medida de poder do carro em polegada cúbica), cavalos de força, relação do eixo traseiro, peso (em libras), eficiência do gasto de combustível (por 1/4 milha), motor (0 = V-shaped, 1 = straight), câmbio (0 = automática, 1 = manual), total de marchas e carburadores.

O Octave conseguiu reconhecer corretamente todos os atributos pois estão separados no padrão por vírgulas, mas as vezes obtemos arquivos com outros separadores como ponto e vírgula por exemplo, neste caso devemos avisar qual o separador:
```
>> objeto = dataframe('arquivo.csv', 'sep', ';')
```

Outro detalhe é relativo ao tipo da variável criada, esta corresponde a *dataframe*:
```
>> class(df)
```

Este tipo só é possível pois temos o pacote que indica sua funcionalidade. Porém observe que existe um tipo de cada coluna dos dados mostrados: *char* e *double*. De mesmo modo podemos analisá-las, por exemplo verificar os tipos das colunas *name* e *mpg*: `>> class(df.name)`
```
>> class(df.mpg)
```

Porém quando utilizamos o nome da coluna vemos que na verdade o tipo da variável é um vetor de double (e não somente uma simples variável do tipo double):
```
>> df.mpg
```

Também podemos obter a coluna do seguinte modo:
```
>> df.array(:,2)
```

Obter um vetor com o nome de todas as colunas:
```
>> df.colnames
```

Localizar o índice de uma coluna pelo seu nome:
```
>> find(strcmp(cellstr(df.colnames),'hp'))
```

Obter um vetor com a quantidade de observações e colunas:
```
>> size(df)
```

Obter a quantidade de observações:
```
>> rows(df)
```

Obter a quantidade de colunas:
```
>> columns(df)
```

Descrever as colunas:
```
>> summary(df)
```

E das colunas numéricas temos o valor mínimo, 1º Quartil, 2º Quartil ou mediana, média, 3º Quartil e o valor máximo. Fica melhor visualizado se selecionarmos apenas algumas colunas numéricas:
```
>> summary(df(:,2:4))
```

Obter as dez primeiras observações:
```
>> df(1:10,:)
```

O primeiro parâmetro é o range de registros e o segundo colunas. Obter as dez últimas observações:
```
>> df(end-10:end,:)
```

A variável *end* contém a quantidade de observações. Obter as 3 primeiras colunas: nome (*name*),

autonomia em milhas por galão (*mpg*) e o número de cilindros (*cyl*):
```
>> df(:,1:3)
```

Outra forma de obter o mesmo resultado é usar pelo nome das colunas:
```
>> df(:,['name'; 'mpg'; 'cyl'])
```

Ou ainda pelos seus índices:
```
>> df(:,[1, 2, 3])
```

Obter as observações onde autonomia é maior que 20:
```
>> df(df.mpg > 20,:)
```

Obter as observações onde autonomia é maior que 20 **E** o número de cilindros é igual a 6:
```
>> df(df.mpg > 20 & df.cyl == 6, :)
```

Obter as observações onde autonomia é maior que 20 **OU** o número de cilindros é igual a 6:
```
>> df(df.mpg > 20 | df.cyl == 6, :)
```

Obter a quantidade de registros retornados:
```
>> length(df.name(df.mpg > 20 | df.cyl == 6))
```

Podemos usar o nome de qualquer coluna mas deve-se tirar os dois-pontos. Obter a média da autonomia:
```
>> mean(df.mpg)
```

Obter a média da autonomia que seja tenha um valor maior que 20:
```
>> mean(df.mpg(df.mpg > 20))
```

Adicionar uma nova observação:
```
>> novo = {'Hundai Sereno', 18.5, 4, 160, 185, 2.8, 1.6, 19}
>> df(end+1, 1:8) = x
```

Ganhamos algumas colunas com informação faltante (NA), localizar as observações que tenham coluna com informação faltante:
```
>> df(isnan(df.vs), :)
```

Devemos analisar cada uma das colunas separadamente, criar uma coluna (por exemplo com um valor aleatório):
```
>> df(:,13) = rand(size(df,1),1)
```

Renomear uma coluna:
```
>> df.colnames(13) = 'nova'
```

Remover uma coluna:
```
>> df(:,13) = []
```

Remover uma linha:
```
>> df(33,:)  = []
```

E por fim, a partir de uma matriz é possível criar um *DataFrame*:
```
>> pop = dataframe({"pais", "capital"; "Belgica", "Bruxelas"; "India", "Nova Delhi";
"Brasil", "Brasilia"})
```

Lembre-se que podemos realizar tudo isso e ainda se aproveitar dos gráficos que estão a nossa

disposição. Então será que o Octave serve ou não para Análise de Dados? Outros pacotes bem interessantes para quem trabalha com tratamento de dados são:

- **Arduino**: permite a comunicação com esta placa.
- **Database**: provê acesso ao SGBD Postgres.
- **Statistics**: inclui estatísticas descritivas, distribuições de probabilidade, testes estatísticos, geração de números aleatórios entre muitas outras funcionalidades.

Sendo assim, não percamos a chance de explorá-los.

8 Conclusão

O GNU Octave é um aplicativo que foi originalmente desenvolvido com o propósito didático, mais especificamente para o projeto de reatores químicos e surgiu a partir da intenção de criar um aplicativo no qual a programação fosse mais rápida do que nas demais linguagens. Tornou-se é uma linguagem de alto nível, destinada principalmente a cálculos numéricos e estatísticos. Fornece uma completa interface utilizada para resolver problemas lineares e não-lineares ou executar outras experiências. Pode ser usado como uma linguagem base para executar roteiros complexos.

Sou um entusiasta do mundo **Open Source** e novas tecnologias. Qual a diferença entre Livre e Open Source? Livre significa que esta apostila é gratuita e pode ser compartilhada a vontade. Open Source além de livre todos os arquivos que permitem a geração desta (chamados de arquivos fontes) devem ser disponibilizados para que qualquer pessoa possa modificar ao seu prazer, gerar novas, complementar ou fazer o que quiser. Os fontes da apostila (que foi produzida com o LaTex) está disponibilizado no GitHub [4]. Veja ainda outros artigos que publico sobre tecnologia através do meu Blog Oficial [2].

Referências

[1] Página do Octave
`https://www.gnu.org/software/octave/`

[2] Fernando Anselmo - Blog Oficial de Tecnologia
`http://www.fernandoanselmo.blogspot.com.br/`

[3] Encontre essa e outras publicações em
`https://cetrex.academia.edu/FernandoAnselmo`

[4] Repositório para os fontes da apostila
`https://github.com/fernandoans/publicacoes`