Apostila básica do Octave

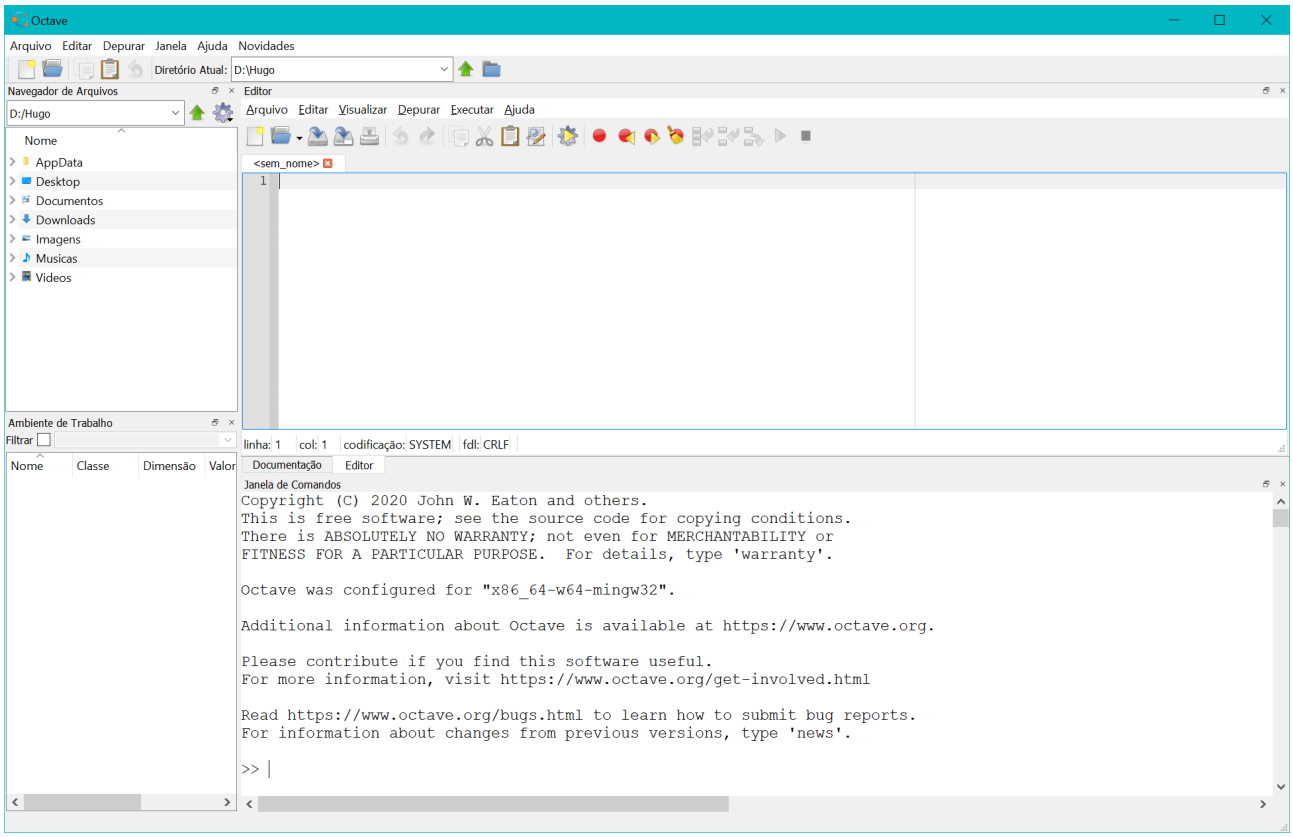
1 Link para fazer o download

https://www.gnu.org/software/octave/index

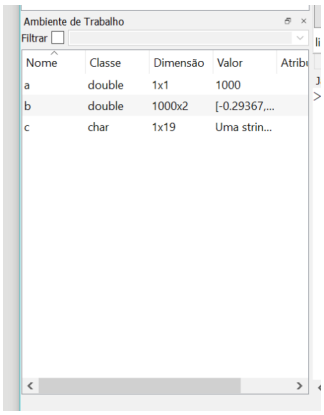
Atenção: recomenda-se fortemente baixar a versão de 64 bits.

2 Interface básica

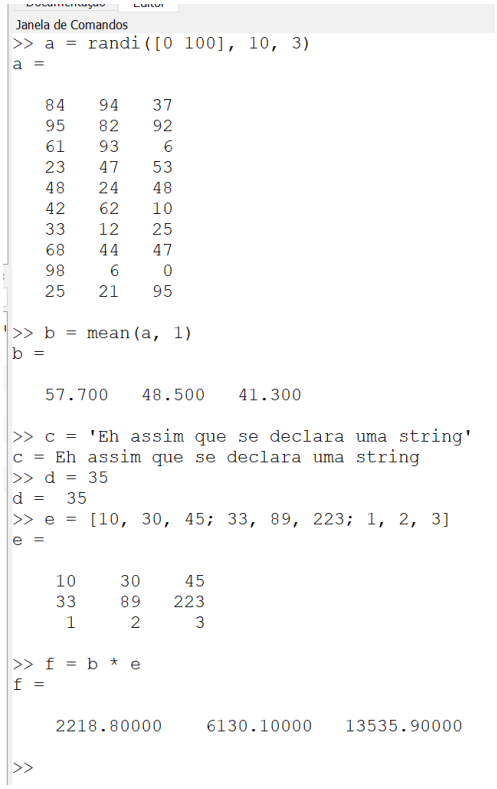
A interface básica do Octave é composta, começando pelo canto superior esquerdo e em sentido horário, pelo navegador de arquivos, o editor, o console/janela de comandos e o workspace/ambiente de trabalho:



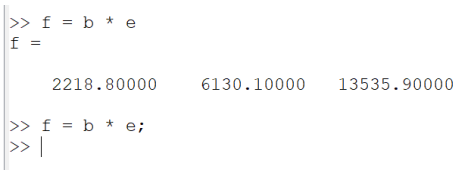
O workspace apresenta todas as variáveis, arrays e estruturas:



No console é possível escrever comandos, expressões, chamar funções, declarar variáveis etc:



Observe na figura acima que as respostas aos comandos aparecem logo abaixo. Porém, se adicionar ponto-e-vírgula ao fim do comando, a resposta é omitida, embora seja calculada normalmente:



O editor de texto permite você escrever scripts – programas sem a declaração de nenhuma função – ou funções. Esse é o componente de interface que você mais utilizará no curso de Matemática Computacional e será abordado mais na próxima seção.

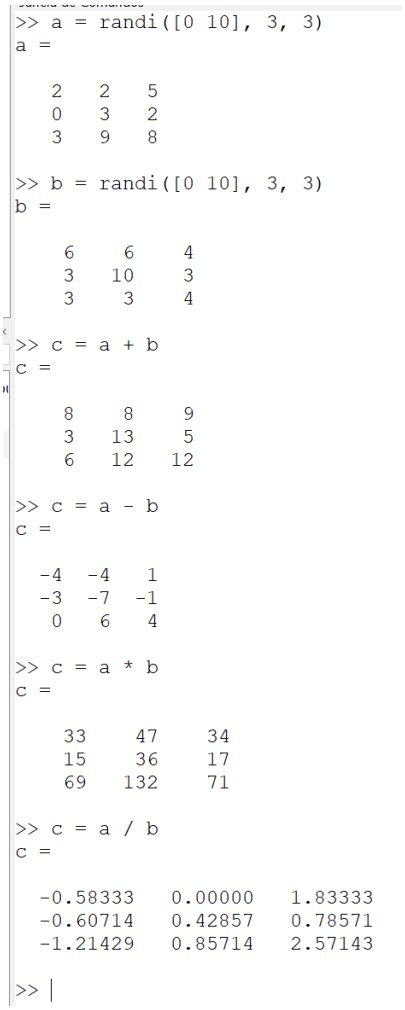
3 Programando no Octave

3.1 Declaração de variáveis, vetores e matrizes

O Octave é uma implementação *open-source* do MATLAB, embora existam diferenças pontuais na sintaxe de alguns comandos entre eles. O MATLAB é uma linguagem muito utilizada por cientistas e engenheiros, porém, ultimamente o Python vem aparecendo como um concorrente à altura.

Sendo muito compatível com o MATLAB, o Octave herda uma de suas características mais notáveis: sempre que possível, deve-se utilizar a programação vetorial ou matricial em vez da programação escalar. Isso quase não será abordado na disciplina, porém, operações básicas com vetores,

como somas, subtrações, multiplicações, divisões, média aritmética e diversos outros operadores e funções estão disponíveis no Octave:



No exemplo acima, a soma e a subtração de matrizes foram realizadas de maneira ponto-a-ponto. Já a multiplicação e a divisão foram feitas seguindo as regras da Álgebra Linear. Caso se deseje fazer multiplicação e divisão ponto a ponto, deve-se utilizar:

```
c = a .* b;
c = a ./ b;
```

Ainda, há a exponenciação:

```
c = a .^ 2;          % eleva cada elemento de A ao quadrado
```

A declaração de variáveis, vetores e matrizes é simples. O Octave é uma linguagem tipada e o tipo padrão de dados é o double (ponto-flutuante de 64 bits). Para strings, o tipo de dados é char .

```
a = 42; (escalar)
b = [10, 20, 30; 40, 50, 60; 70, 80, 90];   % matriz 3x3
c = 'Uma string qualquer';                  % string
d = [10, 20, 30, 40, 50, 60, 70, 80, 90];   % vetor 1x9
e = [10; 20; 30; 40; 50; 60; 70; 80; 90];   % vetor 9x1
```

```
f = 10:10:90;                                   % igual a ‘d’
```

Vetores são indexados a partir da posição 1 e qualquer variável pode virar um vetor, mas é altamente **não-recomendado** fazer como está abaixo e **não é aceito nas atividades na disciplina**:

```
a = 42;             % escalar
a(2) = 52;          % vetor
```

Se for preencher iterativamente um vetor, é necessário que faça a sua pré-alocação com zeros:

```
a = zeros(10, 1);   % vetor 10x1 preenchido com zeros
a(1) = 30; a(2) = 98; c(3) = 87;
```

Inclusive, os índices de vetores também podem ser vetores, caso se queira atribuir um vetor em vez de um escalar:

```
a(4:6) = a(1:3);
a(7:10) = randi([100 1000], 1, 3);
```

Matrizes também são indexadas começando em 1. A posição (1, 1) refere-se ao canto superior esquerdo da matriz. Se elas já podem ser declaradas na totalidade, há, basicamente três formas de declaradas das seguintes formas. Os comandos abaixo produzem exatamente o mesmo resultado:

```
a = [10, 20, 30; 40, 50, 60; 70, 80, 90];   % matriz 3x3
b = [10:10:30; 40:10:60; 70:10:90];
c = reshape(10:10:90, 3, 3)'
```

Se as matrizes forem preenchidas iterativamente, deve-se realizar a pré-alocação:

```
a = zeros(3, 3);
a(1, 1) = 3; a(1, 2) = 8; a(1, 3) = 9;
a(2, :) = randi([85, 95], 1, 3);
a(3, :) = [1, 2, 3];
```

3.2 Declarando funções

A declaração é simples e uma função assume a forma:

```
function saida = nomeFuncao(entrada1, entrada2)
    ...
    saida = umValorQualquer;
end
```

Se a função retornar mais de um valor, a declaração deve ser:

```
function [saida1, saida2] = nomeFuncao(entrada1, entrada2)
    ...
    saida1 = valor1;
```

```
        saida2 = valor2;
end
```

Em todo caso, as variáveis `saida1` e `saida2` podem ser escalares, vetores e matrizes.

A chamada da função é:

```
x = nomeFuncao(a, b, c);
[x, y] = nomeFuncao(a, b, c);
```

Caso se precise chamar uma função que obrigatoriamente retorna dois valores, mas quer descartar um deles, deve-se utilizar o caractere til no lugar do parâmetro de saída a ser descartado:

```
[~, y] = nomeFuncao(a, b, c);
```

Além disso, o Octave permite a declaração de funções anônimas:

```
f = @(x) nomeFuncao(x, b, c);
```

Nesse caso, a expressão `g = f(y)` será transformada em `g = nomeFuncao(y, b, c)`. Observa-se que a declaração de funções anônimas pode ter mais de um parâmetro de entrada e ela assume a forma:

```
f = @(x, y) nomeFuncao(x, b, y);
```

Ainda, podem ser colocadas expressões como em:

```
f = @(x, y, z) x + y / z;
```

3.3 Scripts e funções

Scripts são arquivos-texto em que existe uma série de comandos, como:

```
clear
clc
a = 35.78;
b = 87.98;
c = a + b;
fprintf('O resultado de %5.2f + %5.2f é %5.2f!\n', a, b, c);
```

Por outro lado, programas comumente possui diversas funções. Contudo, ***scripts* não aceitam nenhuma declaração de função**, somente chamadas a funções já existentes ou no Octave ou na pasta onde se encontra o script em questão. Para se declarar funções, há duas alternativas: 1) em arquivos separados, que é o ideal para projetos grandes, e 2) **em um único arquivo, que é o caso adotado na disciplina**.

Vamos ao primeiro caso. Deve-se criar um arquivo **com o mesmo nome da função**, que deve estar na mesma pasta dos outros arquivos do programa para que o Octave consiga localizar as funções.

Arquivo com o nome `nomeFuncao.m`:

```
– BOF –
```

```
function saida = nomeFuncao(entrada1, entrada2)
     ...
     saida = umValorQualquer;
end
– EOF –
```

O arquivo `nomeFuncao.m` só pode conter o que está entre `BOF` (*begin-of-file*) e `EOF` (*end-of-file*).

No caso das atividades da disciplina, o arquivo deverá ser chamado, por exemplo: atividade1.m (1, 2, 3 etc…) e o seu conteúdo será:

```
– BOF –
function atividade1()
     % considere-o como equivalente ao main() da linguagem C
     ...
     ...
     x = funcao1(a, b);
     y = funcao2(c) + funcao3(x);
end
function saida = funcao1(entrada1, entrada2)
     ...
     saida = umValorQualquer;
end
function saida = funcao2(entrada1)
     ...
     saida = umValorQualquer;
end
function saida = funcao3(entrada1)
     ...
     saida = umValorQualquer;
end
– EOF –
```

Observe que a função principal é atividade1() e **ela deve ser a primeira função a ser declarada no arquivo atividade1.m**.

Tanto para *scripts* quanto para funções, **o recomendado é sempre colocar ponto-e-vírgula ao fim de cada comando** para evitar a impressão de resultados no console.

3.4 Blocos de controle de fluxo

3.4.1 Operadores lógico-relacionais

Os operadores são semelhantes aos da linguagem C, com diferença no operador não:

Operador	Significado
`&&`	E
`||`	Ou
`~`	Não
`>`	Maior
`>=`	Maior-ou-igual
`<`	Menor
`<=`	Menor-ou-igual
`==`	Igual
`~=`	Diferente

3.4.2 If

A forma geral do `if` semelhante ao da linguagem C, com a diferença que não se precisa colocar a expressão a ser avaliada entre parênteses:

```
if expressão1
     ...
elseif expressão2
     ...
else
     ...
end
```

Exemplo:

```
if a > b
     ...
elseif a == b
     ...
else
     ...
end
```

3.4.3 For

A forma geral do `for` é:

```
for variavel = faixa de valores
```

```
        ...
    end
```

Exemplos:

```
    for cont = 1:10
        ...
    end
    for cont = 1:2:10
        ...
    end
    for cont = [1, 2, 4, 6, 10, 11, 12]
        ...
    end
```

3.4.4 While

A forma geral do `while` é:

```
    while expressão
        ...
    end
```

Exemplos:

```
    a = 0;
    while a < 20;
        ...
        a = a + 1;
    end
    while true
        ...
        if c < 20 && c >= 15
            break;
        end
    end
```

3.5 Funções comuns do Octave

```
    a = sin(t);         % seno de t (matriz/vetor)
    b = cos(t);         % cosseno de t (matriz/vetor)
```

```
c = tan(t);         % tangente de t (matriz/vetor)

d = mean(x);        % média aritmética de x (matriz/vetor). No caso de x ser uma matriz, essa chamada equivalerá a d = mean(x, 1)

e = mean(x, 1);     % média aritmética percorrendo as linhas da matriz x (dimensão 1).

f = mean(x, 2);     % média aritmética percorrendo as colunas da matriz x (dimensão 2).
```

3.6 Figuras

Vocês serão frequentemente exigidos a fazer gráficos dos problemas de Matemática Computacional.

Basicamente, vocês poderão utilizar as funções `figure()`, `subplot()`, `plot()`, `contour()`, `plot3()`, `surfc()`, `mesh()`, `title()`, `xlabel()`, `ylabel()`, `view()`, `hold()`, dentre outras.

O arquivo `plot.m` que está no Moodle mostra quatro exemplos de geração de figuras e que podem te auxiliar.

3.7 Estrutura básica do programa

Um exemplo de estrutura básica está no arquivo `estrutura.m`, também no Moodle.