

INSTITUTO FEDERAL
Espírito Santo
Campus Serra

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO SUPERIOR
**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO
ESPÍRITO SANTO**
COORDENADORIA DE AUTOMAÇÃO INDUSTRIAL

Prof. Me. Alan Afif Helal

Introdução ao Matlab para Cálculo Numérico

Serra - ES

2019

Sumário

1	CONSIDERAÇÕES INICIAIS	3
1.1	Sobre o Autor	3
1.2	Sobre esta Apostila	3
1.3	O Matlab	3
2	CONHECENDO O MATLAB	4
2.1	Tela Principal	4
2.2	Atribuições	6
2.2.1	Escalares	6
2.2.2	Vetores e Matrizes	9
2.2.3	<i>Strings</i>	12
2.3	Operações Matemáticas	12
2.4	Comandos e Funções nativas do Matlab	15
2.4.1	clear	15
2.4.2	clc	16
2.4.3	help	16
2.4.4	sqrt, exp, log	16
2.4.5	Outras funções matemáticas	16
2.4.6	round, ceil, floor	17
2.4.7	sum, min, max, mean, prod, sort	17
2.4.8	length	18
2.4.9	linspace e logspace	19
2.4.10	plot, plot3 e subplot	19
3	APROFUNDANDO NO MATLAB	24
3.1	Scripts	24
3.2	Funções	25
3.2.1	Sub-função	27
3.3	Entrada e saída de dados para o usuário	27
3.4	Programação Estruturada	29
3.4.1	Estruturas de Tomada de Decisões	30
3.4.1.1	if, if...else, if...elseif	30
3.4.1.2	switch	33
3.4.2	Estruturas de Repetição	34
3.4.2.1	for	34
3.4.2.2	while	36

3.4.3	Função Anônima	37
4	CONSIDERAÇÕES FINAIS	39
4.1	Octave	39

1 Considerações Iniciais

1.1 Sobre o Autor

Alan Afif Helal é Engenheiro de Computação formado pela Universidade Federal do Espírito Santo (UFES) e Mestre em Ciência da Computação Redes de Computadores e Sistemas Distribuídos, pela mesma Universidade. Associado à Sociedade Brasileira de Computação (SBC) e *Association for Computing Machinery* (ACM), com atuação nas áreas de Segurança da Informação e Redes de Computadores. É especialista em Engenharia de Produção e atualmente está cursando Engenharia de Segurança do Trabalho e Segurança da Informação. Maiores detalhes sobre projetos em andamento estão disponíveis no GitHub¹ do autor.

1.2 Sobre esta Apostila

Esta apostila foi desenvolvida para facilitar o uso do Matlab para os alunos que estão cursando a disciplina de Cálculo Numérico. O aluno, após utilizar esta apostila, não terá adquirido conhecimento para se tornar um especialista em Matlab. O propósito desta apostila é fornecer os conhecimentos necessários para que o aluno possa desenvolver as atividades propostas no decorrer do curso de Cálculo Numérico. Caso seja do seu interesse se aprofundar no Matlab, deixo como sugestão cursos da Udemy² ou Coursera³, ou o livro Matlab para Leigos⁴. **Esta obra está licenciada com uma Licença Creative Commons Atribuição-NãoComercial-SemDerivações 4.0 Internacional⁵.**

1.3 O Matlab

O Matlab pode ser adquirido diretamente do site da MathWorks⁶. A versão de estudante custa U\$ 29,00 e o pacote Matlab + Simulink custa U\$ 55,00. É possível testar⁷ o software (Matlab + Simulink) gratuitamente por 30 dias antes de efetuar a compra.

¹ <https://www.github.com/helalbr>

² <https://www.udemy.com>

³ <https://www.coursera.org>

⁴ <http://www.altabooks.com.br/matlab-para-leigos.html>

⁵ <http://creativecommons.org/licenses/by-nc-nd/4.0/>

⁶ <https://www.mathworks.com>

⁷ <https://www.mathworks.com/campaigns/products/trials.html>

2 Conhecendo o Matlab

2.1 Tela Principal

Ao iniciar o Matlab será carregado o seu ambiente principal. Conforme pode ser observado na Figura 1, existem quatro pontos que devemos conhecer para conseguir interagir com o Matlab:

1. **Mostra a pasta atual.** Você pode navegar para outras pastas e, caso haja algum arquivo que o Matlab reconheça, pode ser executado dando dois cliques do mouse sobre ele.
2. **Janela de Comandos.** Local para inserção de comandos e dados. Você pode efetuar desde contas simples até resolução de problemas sofisticados, executando linha por linha.
3. **Área de Trabalho.** Mostra as variáveis, e seus respectivos valores, que foram criadas/definidas até o momento. Você pode consultar o valor de uma variável caso não se lembre de seu valor.
4. **Barra de Estado.** Mostra se o Matlab está ocupado processando alguma informação ou se está pronto para o próximo processamento.

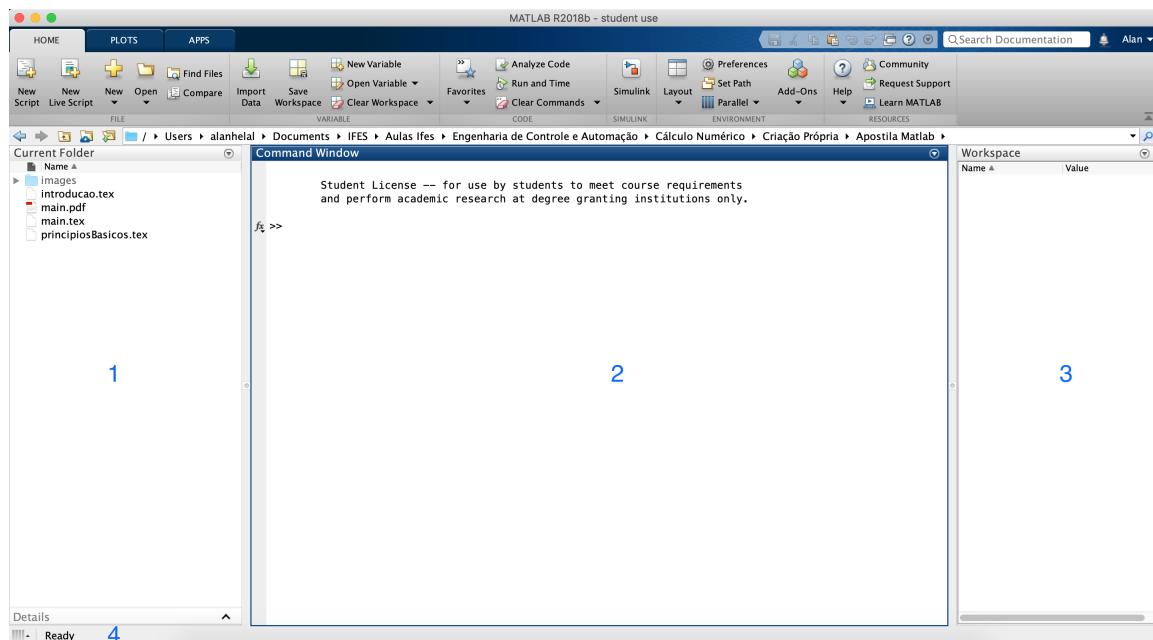


Figura 1 – Ambiente principal do Matlab

A Janela de Comandos pode ser utilizada para realização de contas como se fosse uma calculadora. A Figura 2 mostra algumas operações matemáticas realizadas na Janela de Comandos e como ela retorna o resultado para o usuário. É importante ressaltar que, de forma análoga a uma calculadora científica, se o resultado de uma operação não for implicitamente direcionado para o armazenamento em uma variável¹, automaticamente o Matlab o armazena em uma variável chamada *ans*. Para saber o último valor armazenado em *ans*, podemos olhar a Área de Trabalho ou digitar *ans* na Janela de Comandos.

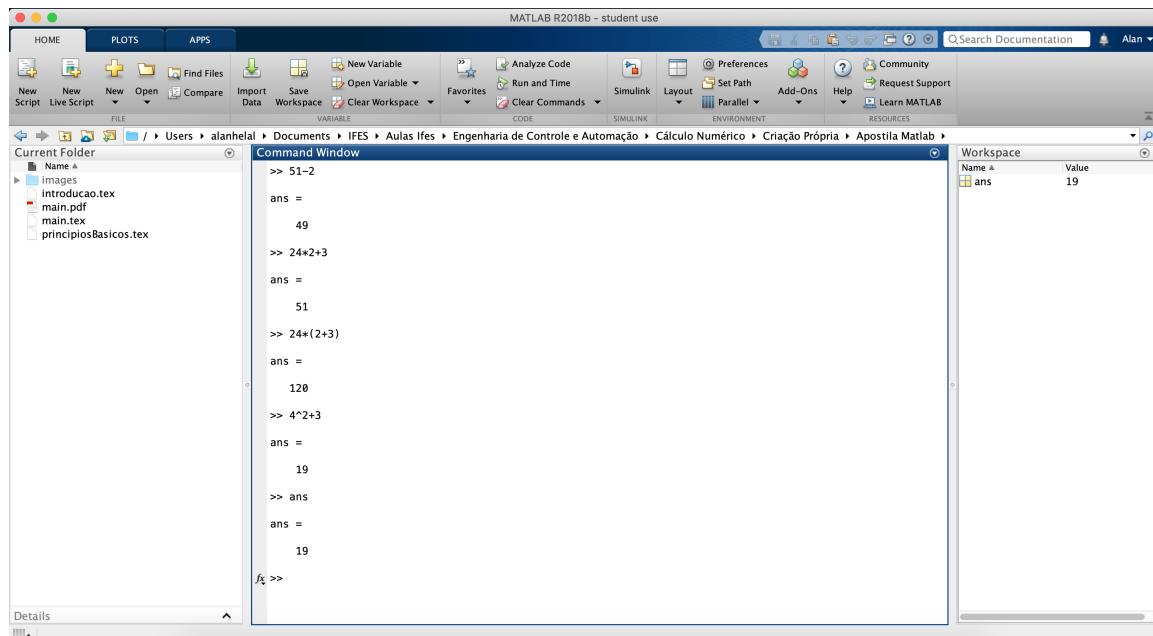


Figura 2 – Contas matemáticas básicas sendo realizadas na Janela de Comando

A Tabela 1 mostra os principais operadores matemáticos para utilização em operações matemáticas básicas².

Operador	Operação
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
[^]	Exponenciação

Tabela 1 – Operações matemáticas básicas

¹ Aprenderemos na próxima seção sobre variáveis

² Na seções seguintes serão introduzidos novos operadores

2.2 Atribuições

2.2.1 Escalares

É possível atribuir valores a variáveis escalares através da Janela de Comandos. Para definir uma variável basta digitar o nome que deseja para a variável seguido do operador "`=`" e então o valor que se deseja armazenar. Para dar nome a uma variável deve-se atentar as seguintes condições:

- O nome da variável não pode começar com um número. Exemplo: `4lados`.
- O nome da variável não pode ser uma palavra reservada do Matlab³
- O nome da variável não pode conter espaço e nem caracteres especiais. Exemplo: `quatro lados, lado*2, lado&diagonal.`

A Figura 3 mostra a atribuição de valores para duas variáveis (`idadePessoa1` e `idadePessoa2`) e depois a criação de outra variável (`somaIdade`) que recebe a soma dessas duas idades. Repare que ao tentar atualizar o valor da variável `idadePessoa1` de 27 para 30, houve um erro de digitação (`idadePeso1`). Como o Matlab é fracamente tipado, esse erro de digitação irá criar uma outra variável, chamada `idadePeso1`, e lhe atribuirá o valor 30. Observe que na Área de Trabalho agora existem quatro variáveis. Outro ponto importante é que o Matlab é *case sensitive*, ou seja, ele diferencia letras maiúsculas de minúsculas. Repare que apesar da variável `idadePessoal` existir, e ter o valor de 27, ao ser solicitado que o Matlab mostrasse na Janela de Comandos o valor dessa variável, o mesmo retornou um erro (pois ao invés de digitar `idadePessoal`, foi digitado `idadepessoal`). Note que o Matlab sugere uma correção. Ele identificou que não existe `idadepessoal` mas que existe `idadePessoal` e pergunta ao usuário se é esse o comando que ele deseja digitar.

Também é possível realizar operações com números imaginários. Para isso basta utilizar a letra *i*. Conforme mostrado na Figura 4, foram definidos dois números imaginários e atribuídos para as variáveis `img1` e `img2`. Conforme pode ser observado, anteriormente estávamos armazenando valores inteiros e agora valores imaginários. Entretanto, não é necessário modificar o modo de atribuição da variável. Ele continua o mesmo - independentemente do tipo da variável. Por exemplo, a variável `img1` está armazenando o número imaginário `-0.2 -i`. Se quisermos trocar o valor dessa variável para o inteiro 10, basta digitar `img1 = 10`. Se quisermos trocar agora para um valor *float* (número fracionário), basta digitar `img1 = 2.7`, por exemplo. Vale ressaltar que, caso se deseja atribuir a essa variável uma *string* (conjunto de caracteres - uma frase), devemos colocar o valor entre aspas. Por exemplo: `img1 = "Isto é uma string"`.

O Matlab possui implementado nativamente algumas funções e valores de constantes. Para isso algumas palavras são alocadas para receberem esses valores. Algumas delas

³ Veremos adiante o que é uma palavra reservada

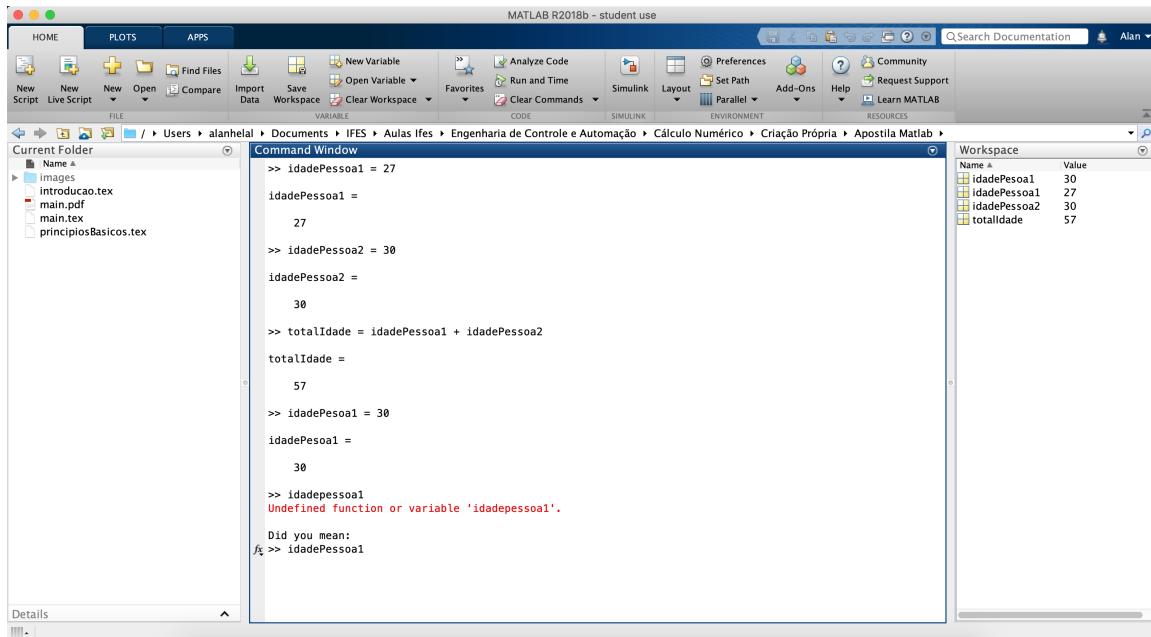


Figura 3 – Atribuição de valores de variáveis escalares

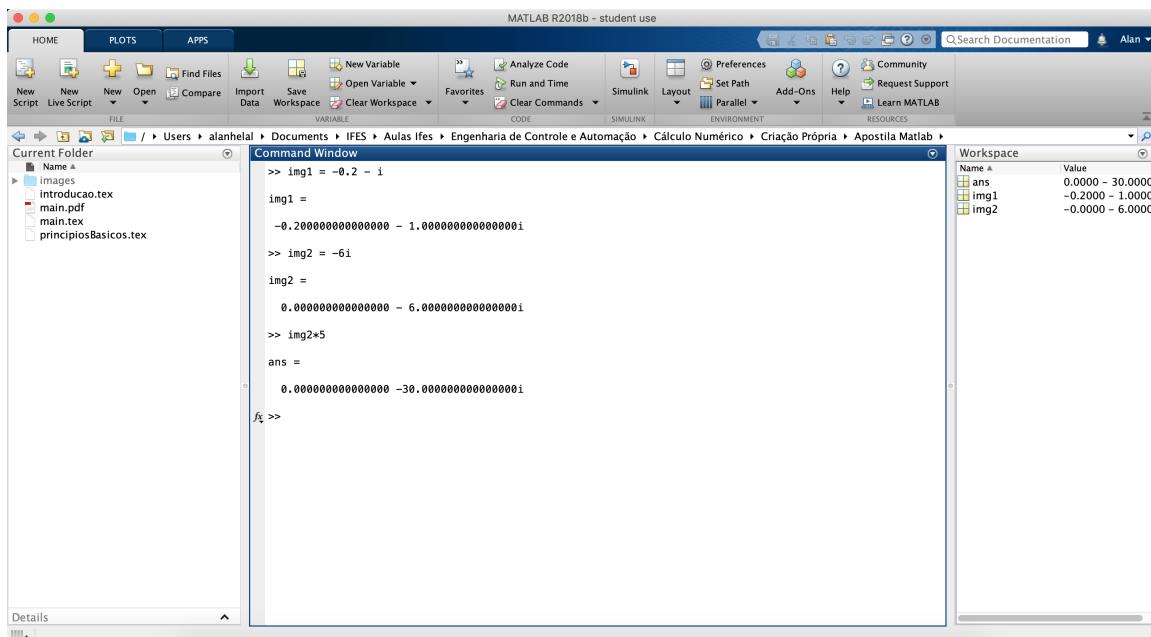


Figura 4 – Atribuição e operação matemática de números imaginários

você pode sobrescrever (apesar de não ser recomendado). Outras, você não pode pois são palavras reservadas do Matlab. Para verificar se uma palavra é reservada, pode-se utilizar a função `iskeyword()`. Ela retorna o valor lógico 1 (verdadeiro) se a palavra for reservada ou o valor lógico 0 (falso) se a palavra não for reservada. A Figura 5 mostra um exemplo dessa situação. O valor do número pi já vem implementado no Matlab. Basta digitar `pi` na Janela de Comandos para ver seu valor. Depois foi verificado se a palavra `pi` é uma palavra reservada do Matlab. Como o resultado foi falso, isso significa que podemos utilizar essa

palavra para outros fins. Neste caso, o valor de pi foi alterado para 4. Enquanto a variável pi estiver definida na sua Área de Trabalho, ela irá valer 4 (ou o valor definido para ela). Caso queira voltar ao valor inicial desta variável, deve-se apaga-la da Área de Trabalho (clicando com o botão direito do mouse em cima dela e selecionando Apagar) ou digitando `clear pi` na Janela de Comandos. Uma lista completa das palavras reservadas do Matlab pode ser obtida digitando `iskeyword()` na Janela de Comandos. Vale ressaltar que, palavras alocadas para funções, desde que não sejam reservadas, também podem sofrer esse tipo de alteração. Por exemplo, como veremos mais adiante, a função `plot` é utilizada para plotar um gráfico. Entretanto, você pode atribuir esse nome a uma variável (por exemplo, `plot = 2`). Ao tentar usar a função `plot`, o Matlab irá mostrar um erro pois existe uma função com o nome de `plot` e uma variável com o mesmo nome.

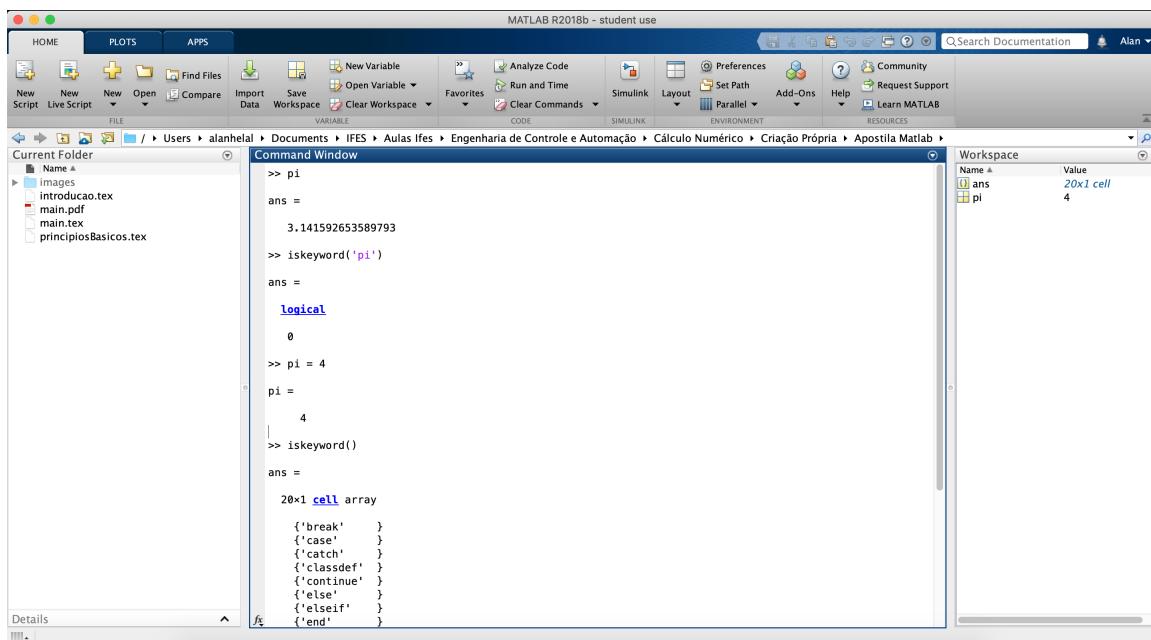


Figura 5 – Uso da função `iskeyword()`

É possível alterar a forma com que o Matlab apresenta os valores da variáveis na Janela de Comandos. A Figura 6 mostra o valor da variável pi (nativa do Matblab). Ela está sendo apresentada com 15 dígitos significativos. É possível alterar o formato de saída para que seja exibido com 5 dígitos significativos no formato de ponto fixo. Para isso utiliza-se o comando `format short`. Caso queira a saída no formato com 5 dígitos significativos no formato ponto flutuante (`float`), pode-se utilizar o comando `format short e`. Caso esteja manipulando valores de moedas, pode-se utilizar o comando `format bank` para que a saída seja com duas casas decimais. É importante ressaltar dois pontos apresentados na Figura 6. Se, ao atribuir um valor a uma variável ou executar um comando, for colocado o caractere `;` no final do comando, o Matlab omite a saída padrão. Ou seja, ao digitar `a = 4;`, por exemplo, o Matlab responde com `a = 4` na Janela de Comandos. Essa é a resposta padrão do Matlab ao ter um valor atribuído a uma variável. Ao realizar `a = 4;`, estamos solicitando

ao Matlab que omita essa resposta. Esse caractere também pode ser utilizado para fazer uma série de atribuições ou execução de comandos em uma linha. Essa situação ocorreu, por exemplo, em *format shor; pi*. Outro ponto importante, é que esses formatos apenas modificam o formato de saída da variável e não o seu valor. Na Figura 6 podemos ver que foi definida uma variável chamada *a* com o valor 1.005. Como o último formato selecionado foi o *format bank*, o Matlab imprimiu na Janela de Comandos o valor *a = 1.00*. Mas repare que ao multiplicar *a* por dois, o resultado foi 2.01 e não 2.00, provando que o valor da variável não foi modificado devido ao formato de saída selecionado.

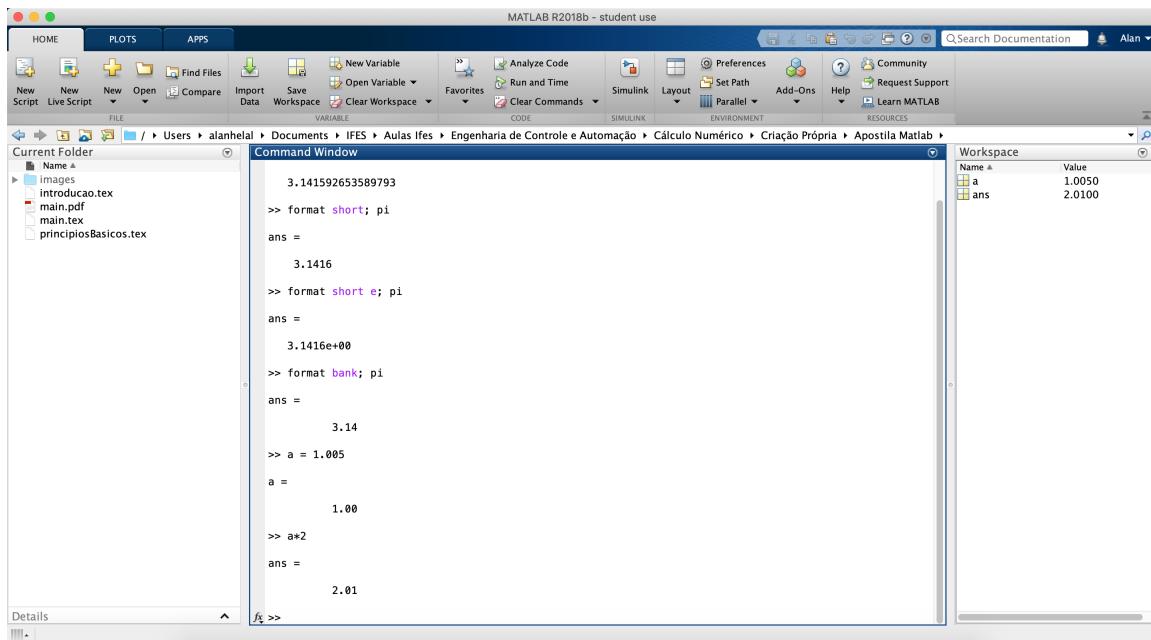


Figura 6 – Alterando o formato de saída das variáveis

2.2.2 Vetores e Matrizes

Para definir um vetor, basta digitar seus valores, separados por vírgula, dentro de colchetes. A Figura 7 mostra esse procedimento, onde foi criado um vetor com 6 valores ([0 2 4 6 8 10]). Note que para transpor um vetor, basta colocar uma aspa simples após o nome da variável que contém os valores desse vetor. também é possível acessar valores dentro do vetor. Para isso utiliza-se o nome da variável que contém o vetor, seguido da posição em que se encontra o elemento que se deseja acessar entre parênteses. É importante ressaltar que, diferente de diversas outras linguagens de programação, o Matlab começa o seu *index* no número 1 e não no 0. Ou seja, para acessar a posição *n* do vetor, se utiliza o *index n* e não *n - 1*.

Para definir uma matriz, deve-se colocar o valor de suas linhas entre colchetes e separando uma linha da outra com ;. Também é possível transpor uma matriz da mesma forma que foi realizado com o vetor. Para acessar um elemento da matriz, deve-se passar

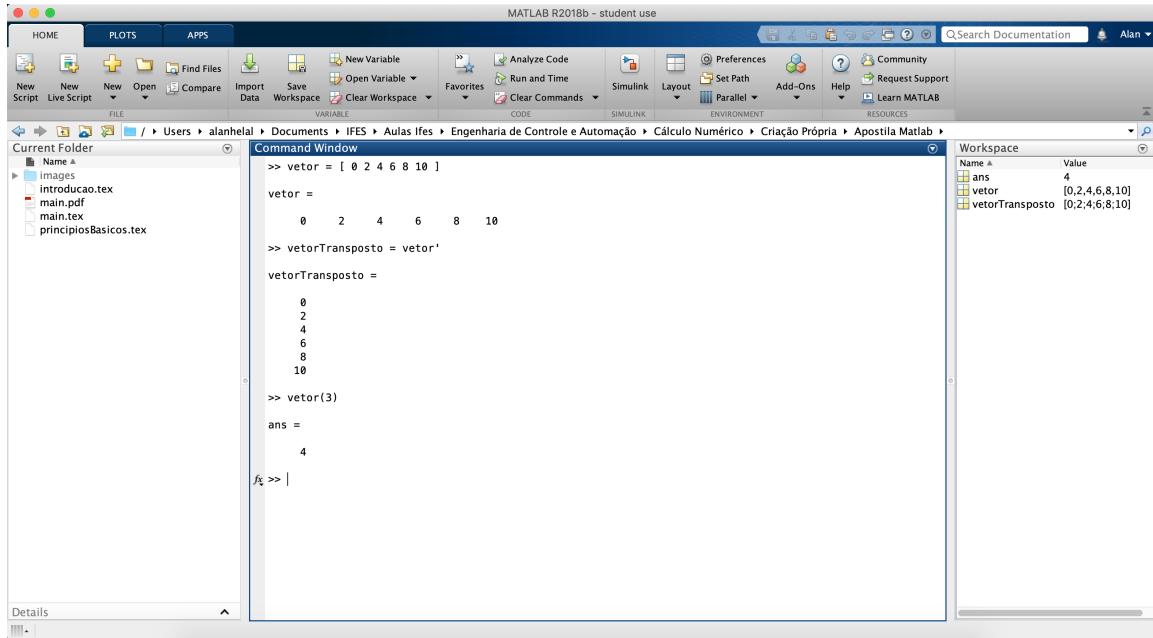


Figura 7 – Criação e manipulação de vetores

o número da linha e da coluna em que se encontra esse elemento. Essas situações são demonstradas na Figura 8.

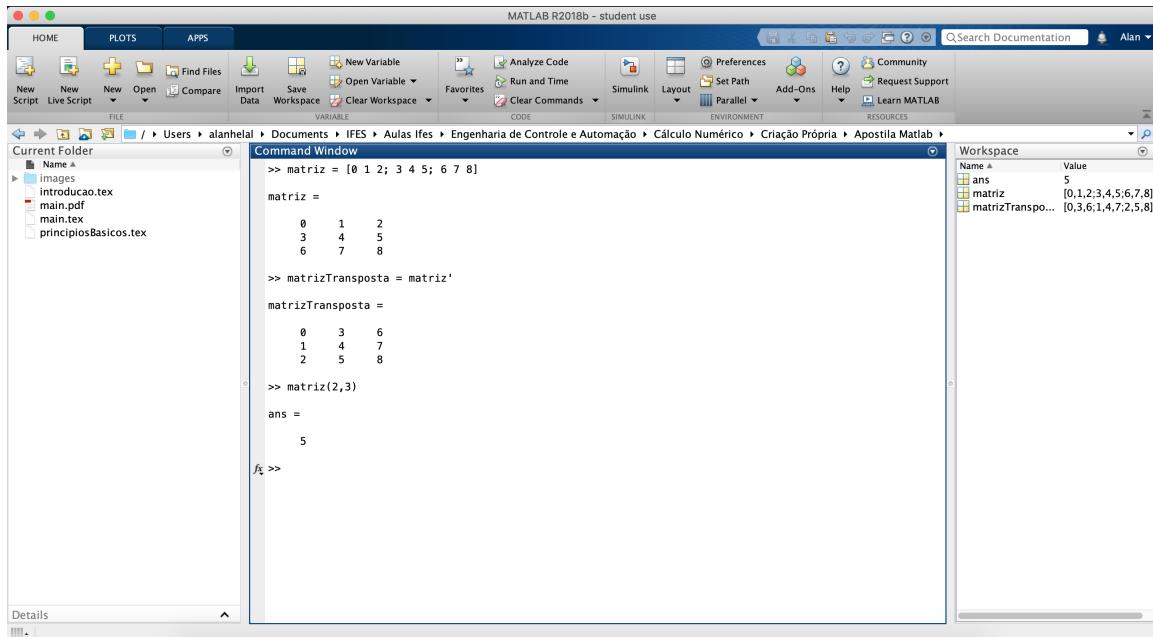


Figura 8 – Criação e manipulação de matrizes

Como pode-se perceber, a criação de um vetor nada mais é que uma matriz de apenas uma linha. Existem duas funções nativas do Matlab que auxiliam na criação de vetores e matrizes. As funções `zeros()` e `ones()` ajudam na criação de matrizes ou vetores de zeros ou uns, respectivamente. A Figura 9 mostra a criação de um vetor de 5 elementos, sendo eles todos zeros. Como um vetor é uma matriz de uma linha, foi passado com

argumento os números 1 e 5 (uma linha e cinco colunas). Também é possível a criação de matrizes de zeros e uns conforme demonstrado na figura. No caso da criação de uma matriz quadrada, não é necessário passar o número de colunas. Ao digitar apenas um número como parâmetro, o Matlab entende que se trata de uma matriz quadrada cria uma matriz com n linhas e n colunas (sendo n o valor passado como parâmetro).

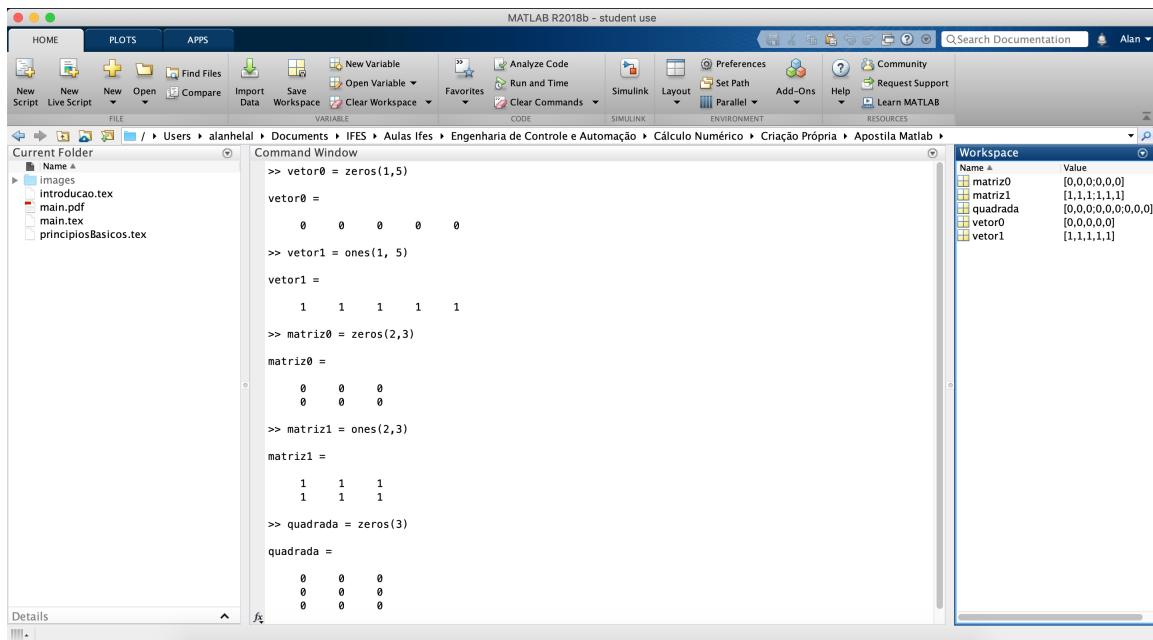


Figura 9 – Criação de vetores e matrizes utilizando `zeros()` e `ones()`

Para a criação de vetores ou intervalos, podemos utilizar o caractere `:`. Ele pode ser utilizada de diversas formas, conforme mostrado na Figura 10. Se utilizado entre dois números, será criado uma sequência que vai do número a esquerda do caractere até o número a direita do caractere, sendo incrementado de forma unitária. Foi utilizado esse método na criação do vetor *tempo*. É possível também definir número inicial da sequencia, o número final e a forma de incremento. Para isso utiliza-se o caractere `:` da seguinte maneira: *valorInicial:incremento:valorFinal*. Esse processo foi utilizado para a criação dos vetores *tempo2* e *tempo3* (repare que para realizar decrementos basta utilizar número negativo). A terceira forma de utilização do caractere `:` é para selecionar intervalos dentro de matrizes e vetores. Lembre-se que para acessar UM elemento da matriz deve-se utilizar a notação (*linha,coluna*). Para acessar a primeira linha inteira da matriz definida no exemplo, foi utilizado `(1,:)` (1 para especificar que se trata da primeira linha, e o caractere `:` para pegar todas as colunas (como não foi definido um valor inicial e um valor final, todos os elementos são selecionados. É possível definir um intervalo da seguinte maneira:

- **valorInicial:valorFinal** - dessa forma estamos especificando um intervalo definido que queremos pegar. Por exemplo selecionar do terceiro ao sexto elemento de um vetor: `vetor(3:6)`

- **valorInicial:** - dessa forma estamos especificando um intervalo que começa em um valor definido e segue até o fim da sequência. Por exemplo selecionar todos os elementos de um vetor a partir da décima posição: vetor(10):
- **:valorFinal** - dessa forma estamos especificando um intervalo que começa no primeiro elemento de uma sequência e vai até um valor final definido. Por exemplo, selecionar os dez primeiros elementos de um vetor: vetor(:10)

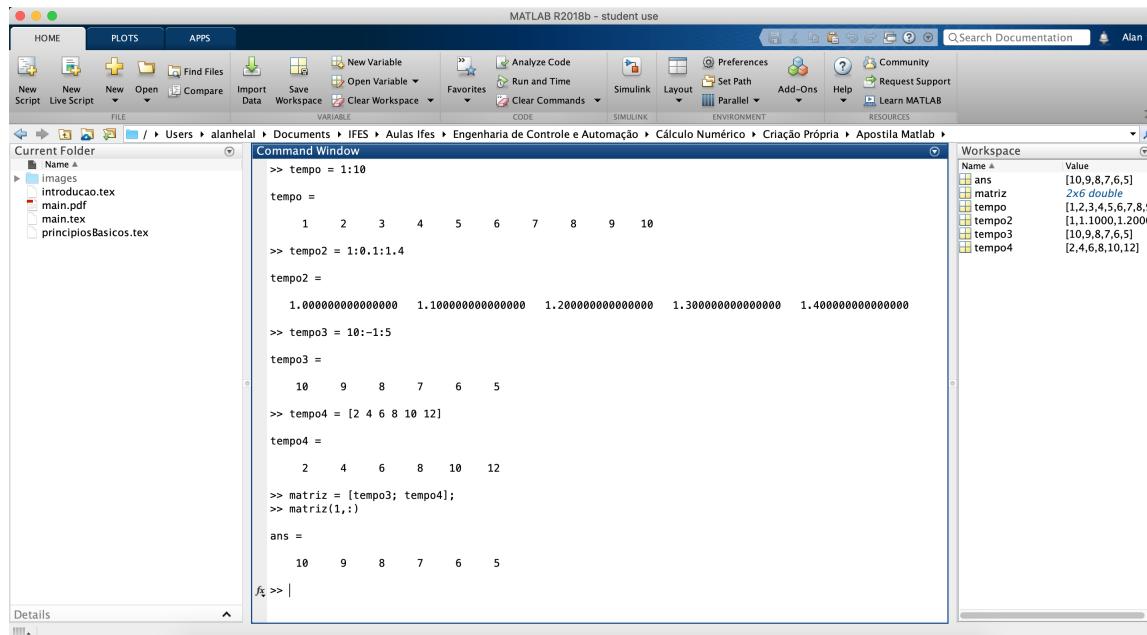


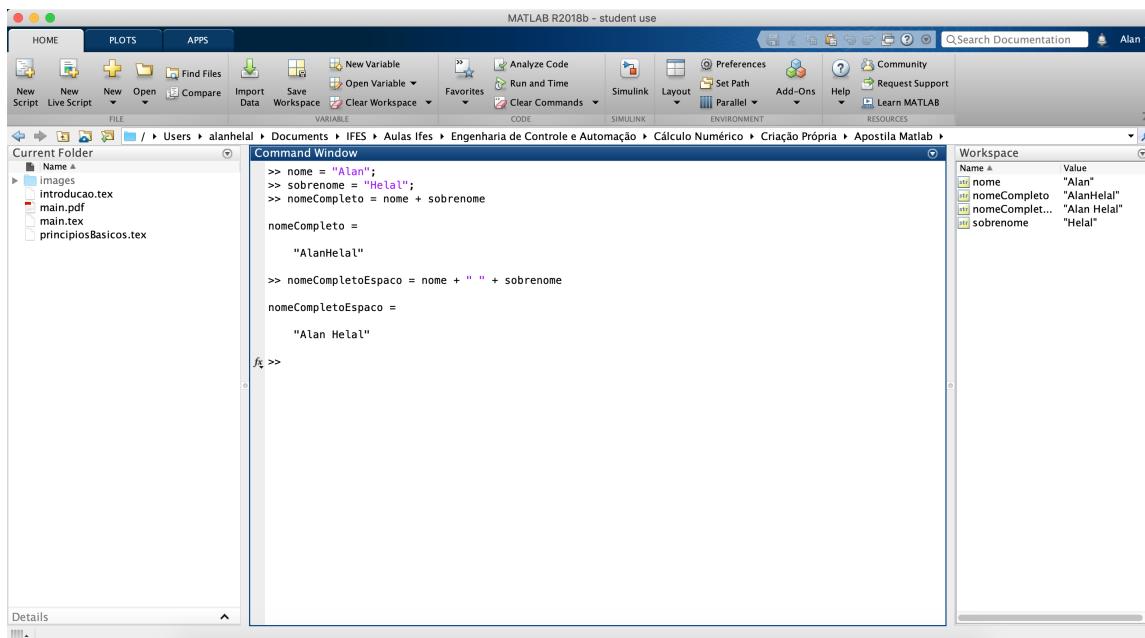
Figura 10 – Formas de uso do caractere ":"

2.2.3 Strings

Uma sequência de caracteres recebe o nome de *string*. Conforme visto anteriormente, para alocar uma *string* em uma variável basta colocar o seu valor dentro de aspas. A Figura 11 mostra a alocação de duas variáveis, *nome* e *sobrenome*, que receberam as *strings* Alan e Helal, respectivamente. Repare que é possível "somar" esses dois valores. Essa "soma" é chamada de concatenação e consiste em unir duas ou mais *strings*. Repare que ao concatenar o *nome* e *sobrenome*, o resultado mostrado em *nomeCompleto* consiste da união dos dois. É possível adicionar espaço, ou até mesmo outra *string*, concatenando a variável *nome* com uma *string* que contém um espaço e concatenando novamente com a variável *sobrenome*, conforme mostrado na variável *nomeCompletoEspaco*.

2.3 Operações Matemáticas

As operações matemáticas seguem uma ordem de prioridade. Segue abaixo a lista com os operadores mais utilizados (começando do de maior prioridade para o de menor

Figura 11 – Operações com *string*

prioridade):

- Parênteses
- Exponenciação
- Multiplicação e divisão
- Soma e subtração

Uma lista completa de precedência dos operadores pode ser consultada no site oficial da MathWorks⁴.

A Figura 12 mostra alguns exemplos de operações matemáticas utilizando a precedência de operadores. Primeiro foi atribuída a variável *aceleracaoGravidade* o valor 9.81. Depois foi calculada a sua raiz quadrada utilizando a exponenciação a 0.5. Um exemplo de como o parênteses altera o resultado da operação pode ser observado nas variáveis *quadrado* e *quadradop*. Por fim, ao atribuir a variável *peso* o valor de *aceleracaoGravidade* dividido pela valor da variável *quadrado*. Entretanto, houve um erro de digitação. Observe que como estamos atribuindo um valor a uma variável, nesse caso o Matlab apresenta uma mensagem de erro dizendo que não encontrou tal variável na Área de Trabalho.

Um dos grandes diferenciais do Matlab é sua habilidade de realizar operações matemáticas com vetores e matrizes. Sempre se lembre que um vetor nada mais é que uma matriz de uma linha e n colunas. Na Figura 13 foi definido uma matriz com 1 linha e 5 colunas (variável *vettor1* - observe que foi utilizado o caractere : para definir os valores

⁴ https://www.mathworks.com/help/matlab/matlab_prog/operator-precedence.html

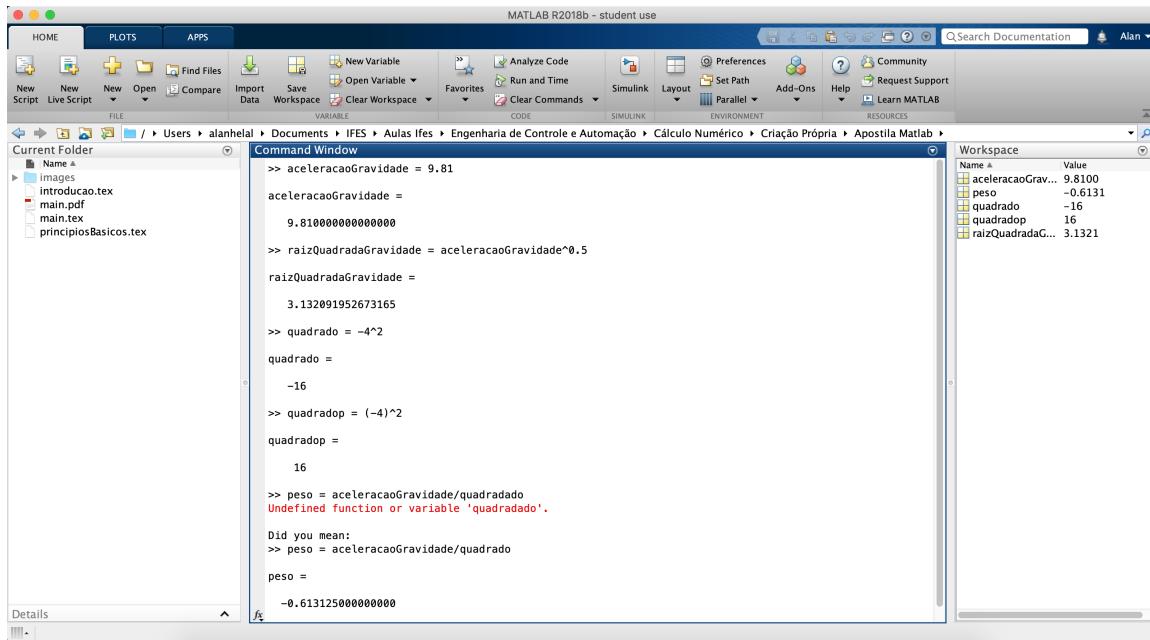


Figura 12 – Operações matemáticas

dessa matriz). Depois foi criado uma matriz de 5 linhas e 1 coluna (variável *vetor2* - observe que foi definido como um vetor transposto). Multiplicando *vetor1* por *vetor2* temos como resultado uma matriz de dimensões 1x1 (uma linha e uma coluna). Mas se efetuarmos a multiplicação *vetor2* por *vetor1* obtemos uma matriz de dimensões 5x5.

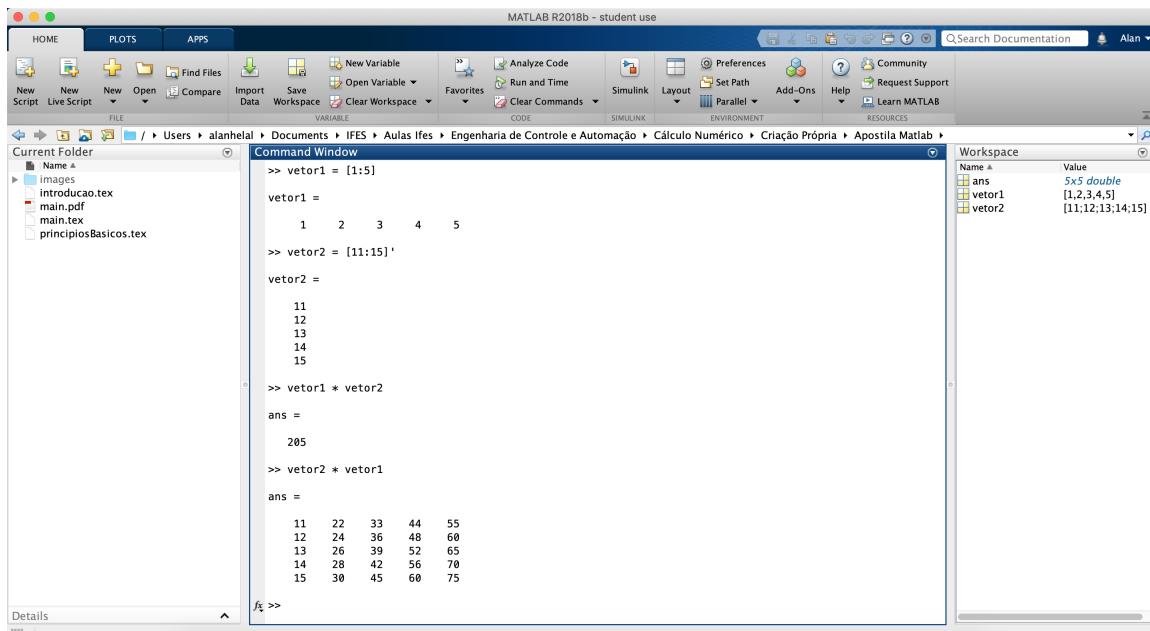


Figura 13 – Operações matemáticas com vetores e matrizes

Considere agora uma matriz 6x6, conforme demonstrado na Figura 14. Observe que se efetuarmos o comando *matriz*² estamos dizendo ao Matlab para multiplicar a matriz por ela mesma. Pode ser que estivéssemos interessado em realizar operações em cada elemento

interno da matriz, como por exemplo elevar cada elemento da matriz ao quadrado. Para que isso ocorra, deve-se utilizar o operador `.^` precedido da operação que se deseja realizar. Repare na Figura 14 ao realizar $\text{matriz}.\hat{2}$ não estamos mais multiplicando a matriz por ela mesma! Dessa forma estamos elevando cada elemento da matriz ao quadrado (repare que o resultado de $\text{matriz}.\hat{2}$ é diferente de $\text{matriz}.\hat{2}$).

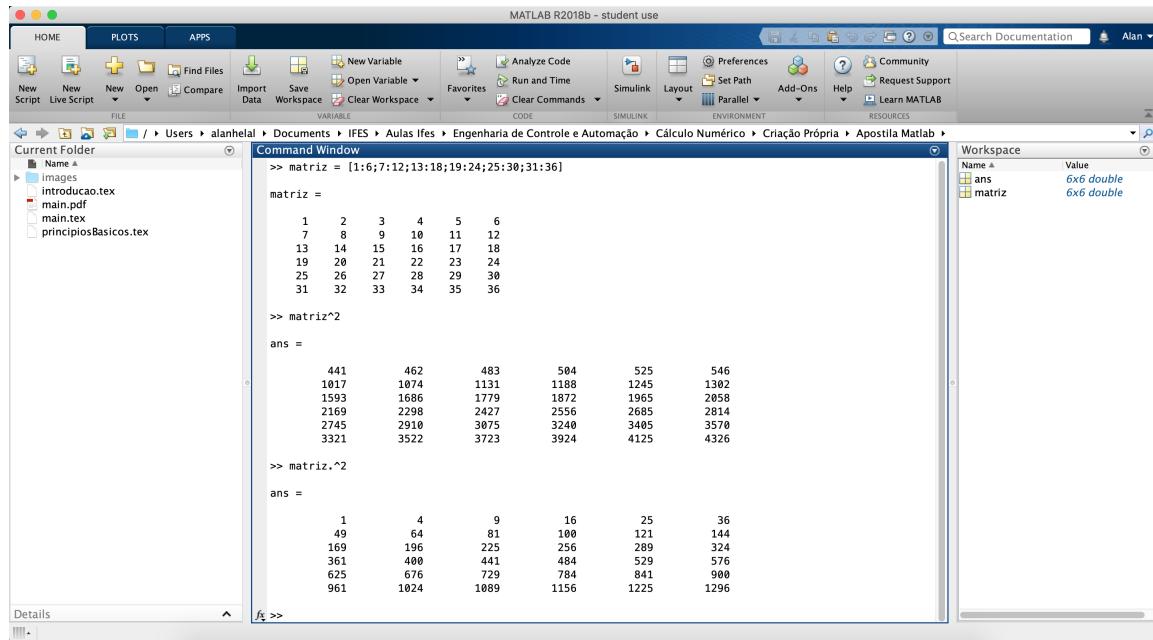


Figura 14 – Operação elemento por elemento de uma matriz

2.4 Comandos e Funções nativas do Matlab

O Matlab possui nativamente implementado funções para facilitar a manipulação dos dados para o usuário. Nesta seção serão apresentados as principais funções e comandos para utilização no curso de Cálculo Númerico⁵

2.4.1 clear

O comando `clear` pode ser utilizado de duas formas:

- `clear` - se for digitado somente `clear` será apagado todas as variáveis da Área de Trabalho
- `clear nomeVariavel` - se for passado um nome de variável existente na Área de Trabalho, então será apagado somente essa variável da Área de Trabalho

⁵ Existem centenas de outras funções e comandos nativos do Matlab. Aqui foram apresentados somente os principais para a utilização em Cálculo Numérico

2.4.2 clc

Esse comando serve para apagar o que está impresso na Janela de Comandos. É bastante utilizado em conjunto com o comando *clear* antes de executar algum *script*⁶ para limpar a Janela de Comandos e as variáveis já definidas na Área de Trabalho.

2.4.3 help

Serve para conseguir ajuda sobre algum comando ou função (mesmo que a função não seja definida nativamente pelo Matlab). Basta digitar *help nomefunção* e o Matlab mostrará como utilizar tal função (ou comando), bem como o que ele faz.

2.4.4 sqrt, exp, log

Para o cálculo de raiz quadrada pode-se utilizar a função *sqrt* (contração do inglês *square root*). Para calcular exponenciação do número de Euler pode-se utilizar a função *exp*. Para o cálculo de logaritmos o Matlab possui três funções implementadas:

- **log** - calcula o log natural (neperiano⁷) de um número
- **log2** - calcula o logaritmo na base 2 de um número
- **log10** - cacalcula o logaritmo na base 10 de um número

A Figura 15 mostra exemplo de utilização dessas funções.

Para a grande maioria das funções que tratam de operações matemáticas em números, é possível adaptá-las para operação em matrizes. Para isso deve-se acrescentar a letra "m" no final do nome da função. Exemplo: *expm* e *sqrtm*.

2.4.5 Outras funções matemáticas

A grande maioria das funções de operações matemáticas possuem o nome igual ao que é usado no dia-a-dia. Seguem alguns exemplos:

- **abs** - calcula o valor absoluto de um número (real ou imaginário)
- **sin, cos, tan** - calcula o seno, cosseno e tangente de um número
- **asin, acos, atan** - calcula o arco seno, arco cosseno e arco tangente de um número
- **sinh, cosh, tanh** - calcula o seno hiperbólico, cosseno hiperbólico e tangente hiperbólica de um número

⁶ Veremos posteriormente o que é e como criar um *script*

⁷ Apesar do criador do logaritmo neperiano ter utilizado como base 1/e, ele é comumente utilizado para designar o logaritmo natural

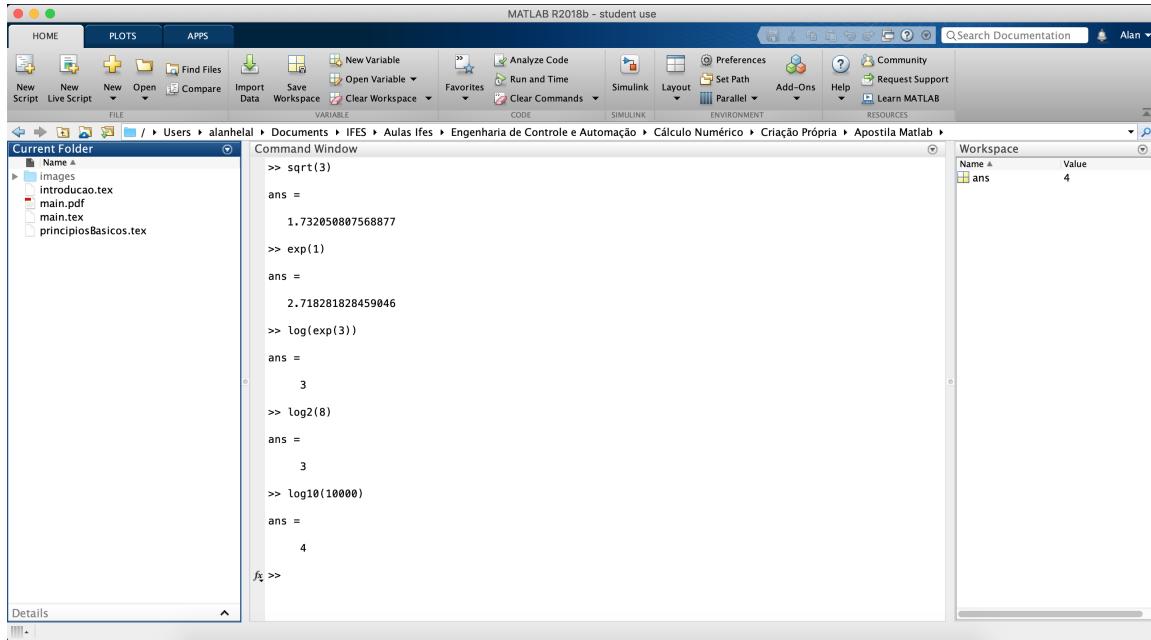


Figura 15 – Exemplos de utilização das funções *sqrt*, *exp* e *log*

2.4.6 round, ceil, floor

São funções para arredondamento de números. Funcionam da seguinte forma:

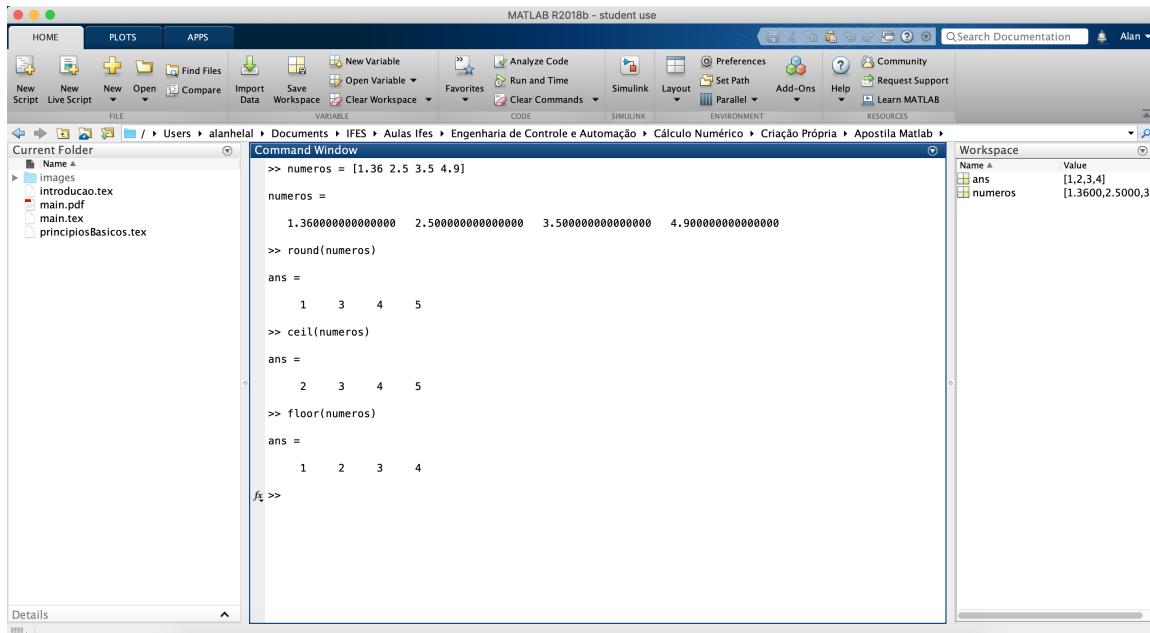
- *round* - arredonda os números para baixo se a parte fracionária for menor 5, e arredonda para cima se a parte fracionária for maior ou igual a 5
- *ceil* - arredonda todos os números para cima, independentemente da sua parte fracionária
- *floor* - arredonda todos os números para baixo, independentemente da sua parte fracionária

A Figura 16 mostra o funcionamento dessas três funções.

2.4.7 sum, min, max, mean, prod, sort

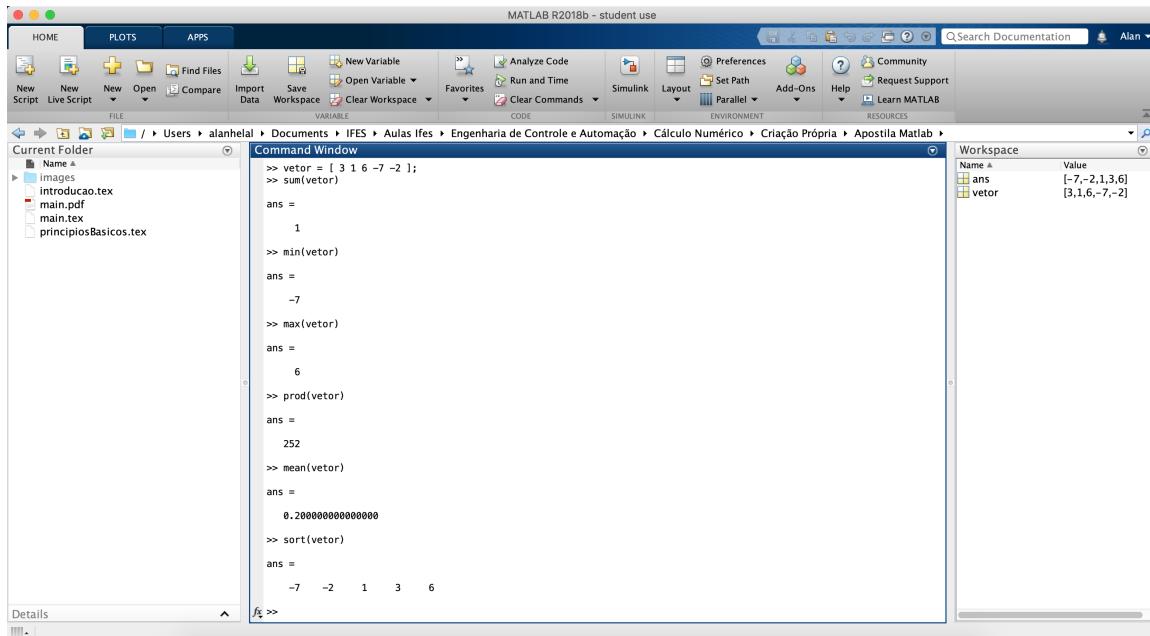
São funções para operação em vetores de números:

- *sum* - efetua a soma de todos os elementos do vetor
- *min* - retorna o menor número do vetor
- *max* - retorna o maior número do vetor
- *mean* - retorna a média aritmética dos elementos do vetor
- *prod* - retorna o produto dos elementos do vetor

Figura 16 – Exemplos de utilização das funções *round*, *ceil* e *floor*

- *sort* - retorna o vetor ordenado do menor elemento para o maior

Na Figura 17 foi criado um vetor e aplicada essas funções nele.

Figura 17 – Exemplos de utilização das funções *sum*, *min*, *max*, *mean*, *prod* e *sort*

2.4.8 length

Retorna o comprimento de um vetor, ou seja, a quantidade de elementos que ele possui. Por exemplo: Considere o seguinte vetor= [0:2:100] (ele possui todos os números pares de

0 até 100). A função `length(vetor)` retorna o valor 51 (existem 51 elementos nesse vetor).

2.4.9 `linspace` e `logspace`

Conforme vimos anteriormente, podemos criar intervalos de números com o operador `:`. Entretanto, as vezes não conseguimos achar um espaçamento que conteplete o primeiro e o último elemento da sequência que desejamos criar. Observe na Figura 18 que ao tentar criar um vetor que começa no número 1 e termina no número 3 com espaçamentos de 0.3, não foi possível chegar até o número 3. As funções `linspace` e `logspace`, criam um espaçamento linear e logarítmico, respectivamente. A utilização dessas funções seguem explicadas abaixo:

- `linspace(inicio,fim,numeroElementos)` - escolhe-se o valor inicial da sequência, o valor final da sequência e a quantidade de elementos que se deseja na sequência. O Matlab automaticamente calcula qual deve ser o incremento de forma que seja linear
- `logspace(inicio,fim,numeroElementos)` - escolhe-se o valor inicial da sequência, o valor final da sequência e a quantidade de elementos que se deseja na sequência. O Matlab automaticamente calcula qual deve ser o incremento de forma que seja logarítmico

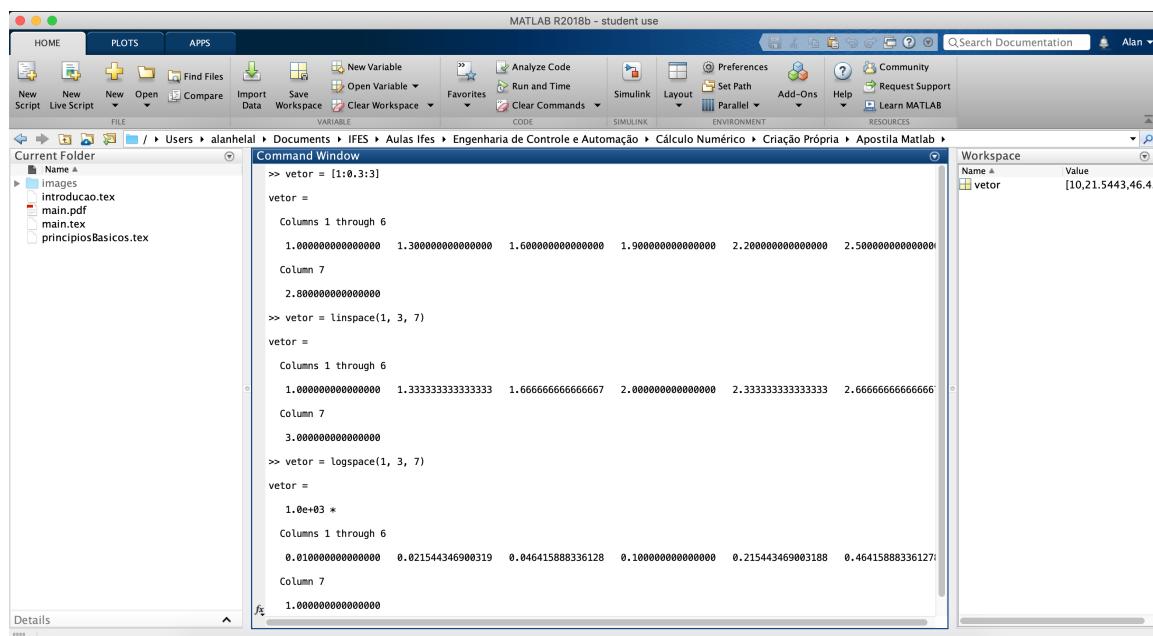


Figura 18 – Exemplos de utilização das funções `linspace` e `logspace`

2.4.10 `plot`, `plot3` e `subplot`

A função `plot` é uma das mais utilizadas no Matlab. Com ela é possível plotar gráficos (além de poder estilizar o gráfico de diversas maneiras). Sua sintaxe é bem simples: `plot(t,`

v) - será plotado um gráfico de *t* em função de *v*. Para facilitar o entendimento do gráfico, é possível colocar textos e grid:

- *title* - Adiciona um título ao gráfico
- *xlabel* - Adiciona um texto explicando sobre o que o eixo x está representado
- *ylabel* - Adiciona um texto explicando sobre o que o eixo y está representando
- *grid* - Adiciona *grid* (linhas horizontais e verticais) ao gráfico para facilitar a visualização

A Figura 19 mostra a criação de um gráfico da função seno com seu intervalo variando de 0 até 2π com espaços de $\pi/100$. Após ter plotado o gráfico foi definido um nome para ele, colocado texto para explicar os eixos X e Y e habilitado o *grid* para facilitar a visualização.

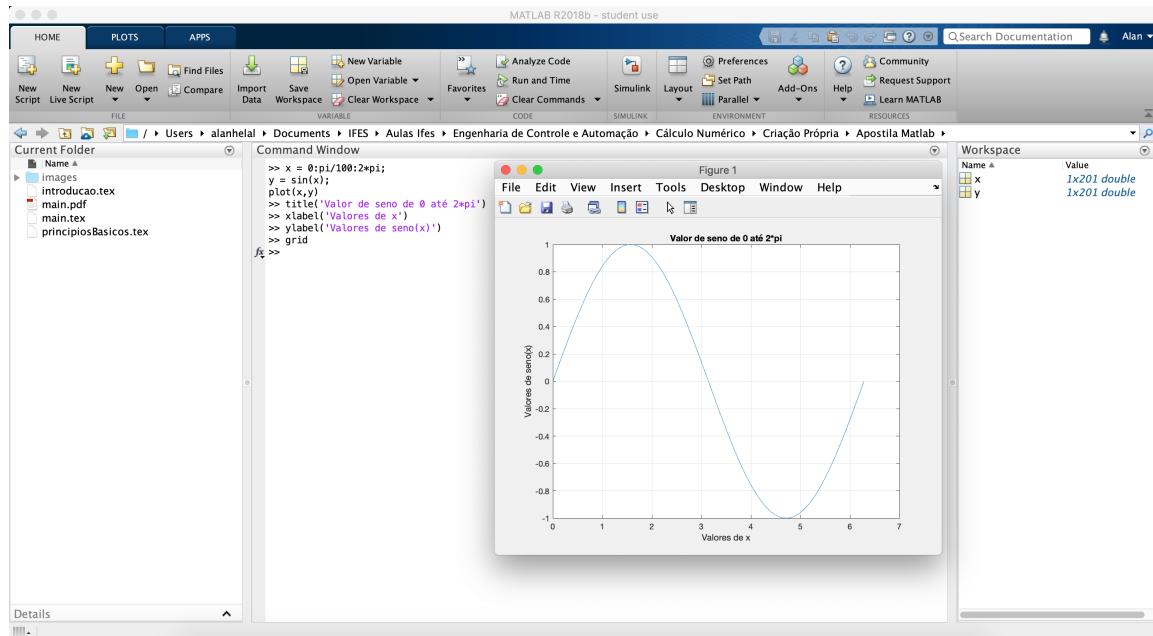


Figura 19 – Exemplo de criação de um gráfico com a função *plot*

Toda vez que a função *plot* for invocada, ela apagará o gráfico anterior. Caso queira manter o gráfico anterior ao plotar um novo gráfico pode user o comando *hold*:

- *hold on* - mantém o gráfico anterior e plota o novo gráfico na mesma janela
- *hold off* - se estiver ativado o *hold on* e deseja voltar ao estado padrão, onde o comando *plot* apaga o gráfico anterior antes de plotar o novo gráfico, deve-se utilizar *hold off*

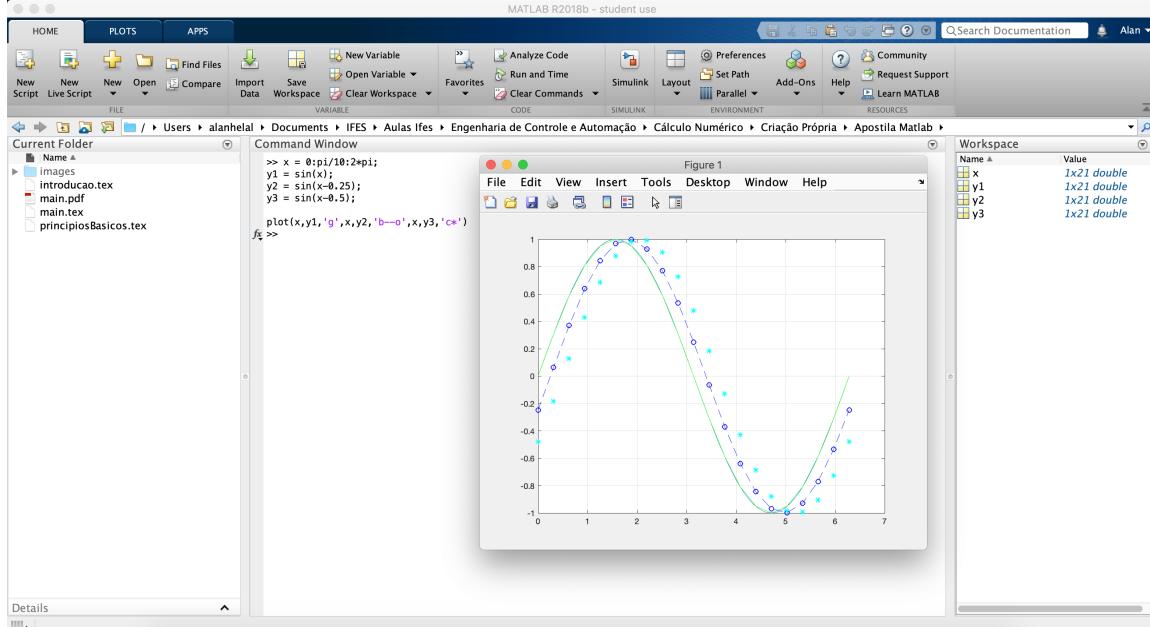


Figura 20 – Exemplo de criação de diversos gráficos com a função *plot*

Também é possível plotar mais de um gráfico ao mesmo tempo. Para isso basta colocar como parâmetros (vetor1, vetor2, propriedadesLinhas, vetor3, vetor4, propriedadesLinhas,...) conforme demonstrado na Figura 20.

As principais propriedades que podem ser utilizadas na linha do gráfico são:

- **Cor:**

- **b** - azul
- **g** - verde
- **r** - vermelho
- **c** - ciano
- **m** - magenta
- **y** - amarelo
- **k** - preto
- **w** - branco

- **Símbolos:**

- **.** - ponto
- **o** - círculo
- **x** - símbolo X
- **+** - mais
- ***** - estrela

– s - quadrado

- **Tipos de linha:**

- - - sólida
- : - pontilhada
- - . - traço-ponto
- — - tracejada

- **LineWidth** - espessura da linha. Deve ser utilizada da seguinte maneira: 'LineWidth', valor. Por exemplo: ('LineWidth', 5), especifica a espessura da linha para 5.

Na Figura 20 foi definido como verde a cor do primeiro gráfico. Para o segundo gráfico foi definido como azul, utilizando linha tracejada e símbolos quadrados. Já para o terceiro foi definido a linha com a cor ciano e utilizando símbolos estrela.

Outra função importante para a plotagem de gráficos é a função *subplot*. Com ela é possível dividir a janela onde os gráficos serão plotados em m linhas e n colunas e então selecionar em qual posição deseja-se plotar o gráfico (a contagem começa em 1 e segue da esquerda para a direita da primeira linha até a última linha).

A Figura 21 mostra essa função sendo utilizada para plotar o gráfico de quatro diferentes funções seno. Repare que para cada gráfico devemos antes especificar qual o *subplot* será utilizado. Na primeira linha foi definido que a área de postagem de gráficos será dividida em 2 linhas e 2 colunas e que esse primeiro gráfico ocupará a primeira posição, por isso o comando: *subplot(2,2,1)*. Os demais gráficos seguem a mesma analogia.

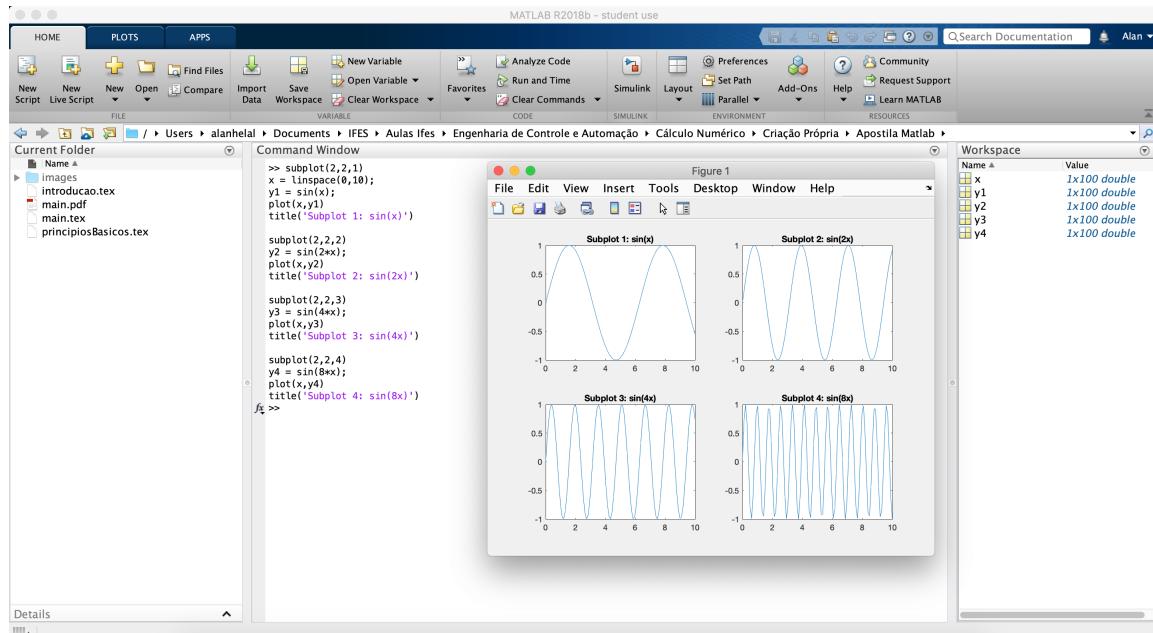


Figura 21 – Exemplo de criação de diversos gráficos com a função *subplot*

Já o `plot3` (X_1, Y_1, Z_1, \dots), onde X_1, Y_1, Z_1 são vetores ou matrizes, plota uma ou mais linhas no espaço tridimensional através dos pontos cujas coordenadas são os elementos de X_1, Y_1 e Z_1 . Assim como no `plot` ou `subplot`, os valores em X_1, Y_1 e Z_1 podem ser numéricos, datetime, duração ou valores categóricos.

Um exemplo da utilização do `plot3` é demonstrado na Figura 22 onde foi definido t com os valores entre 0 e π , e definidos st e sc como os valores de seno e cosseno, respectivamente, dos valores de t . Como pode ser observado, o resultado é uma hélice em 3D.

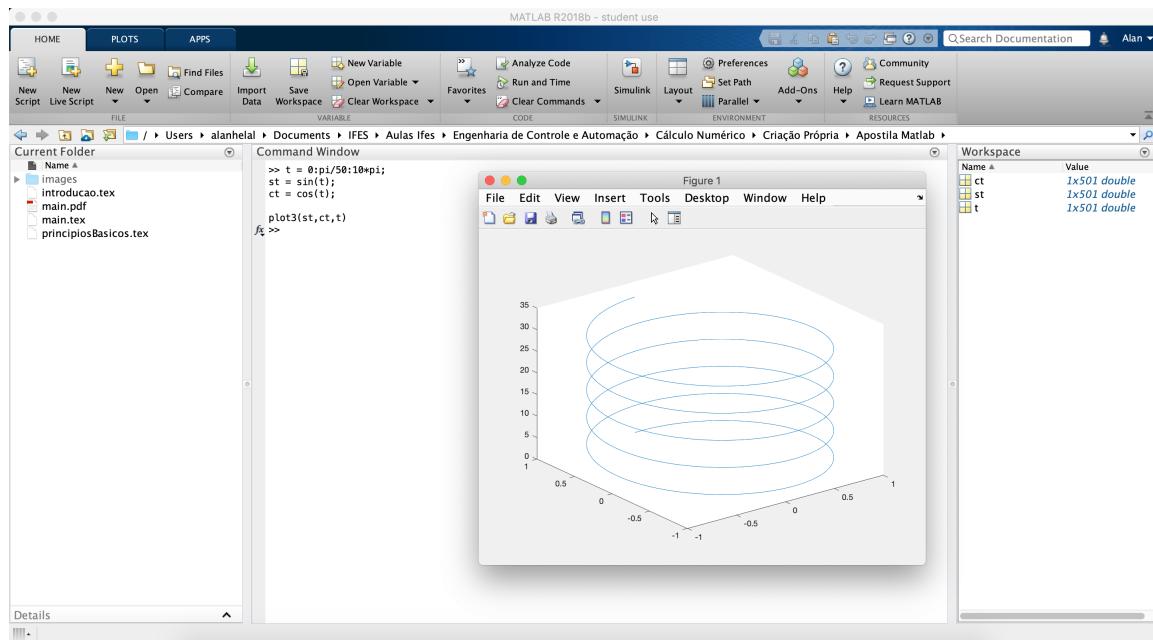


Figura 22 – Exemplo de criação de gráficos com a função `plot3d`

3 Aprofundando no Matlab

3.1 Scripts

Um *script* do Matlab trata-se de uma série de comandos do Matlab que são salvos em um arquivo (com extensão *.m* e podem ser executados posteriormente. Ao executar um *script*, o Matlab reproduz linha a linha do que está programado no *script*. Para executar um script podemos usar a janela que mostra a Pasta Atual e selecionar o *script* dando dois cliques com o botão esquerdo do mouse, ou clicar no botão Abrir e então selecionar o *script*. Para criar um *script* basta clicar no botão Novo e então selecionar *script*.

A Figura 23 mostra a criação de um *script*. Perceba que ao clicar em novo -> *script*, o Matlab nos apresenta uma nova janela. Esta é a Janela de Edição e nela que definiremos nosso *script*. Repare que foi definida uma variável investimento contendo um valor de 15000, uma variável juros contendo o valor de 0.01 e uma variável meses contendo o valor 10. No final é realizada uma conta que nos retorna o total. Não é difícil perceber que se trata de uma aplicação com juros de 1% ao mês durante 10 meses.

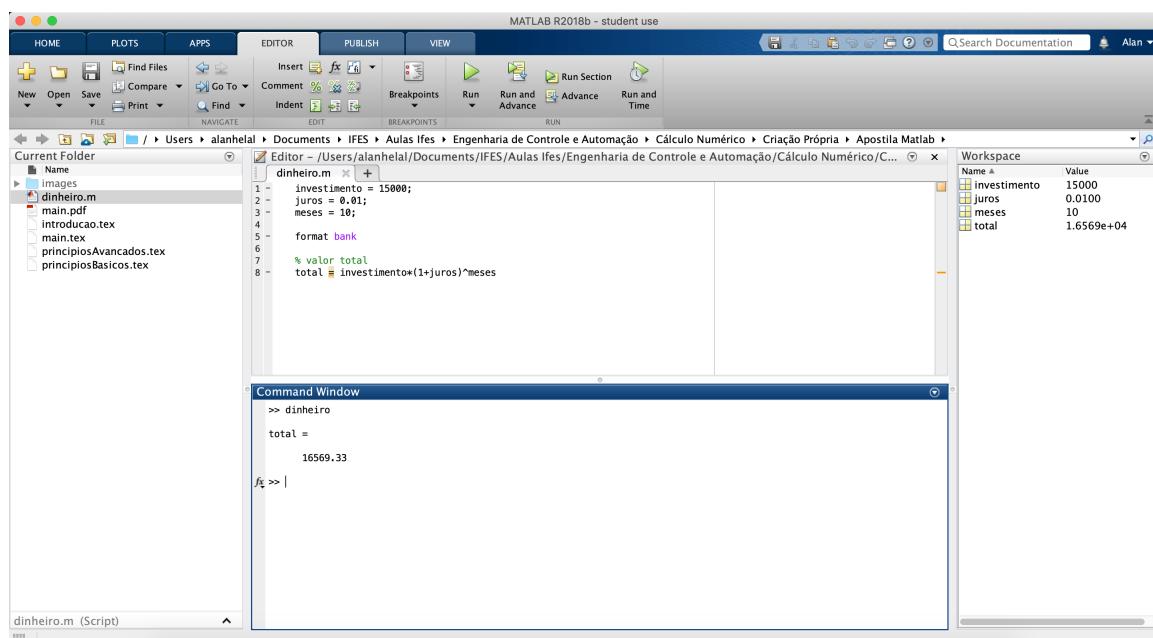


Figura 23 – Criando e executando um script

É importante frisar os seguintes pontos:

- Uma vez salvo o *script* para o executar basta apertar o botão *Run* ou digitar o nome do script na Janela de Comandos. Nesse exemplo o script foi salvo com o nome "dinheiro". Se a Pasta Atual for a pasta que contém esse *script*, basta digitar dinheiro

na Janela de Comandos e o Matlab irá executar o *script*, conforme mostrado na Figura 23

- O nome do *script* **não pode** ser igual ao nome de qualquer variável contida no *script*
- Repare que após executar o *script*, a Área de Trabalho manteve as variáveis do *script* com seus respectivos valores
- Imagine que o nome do *script* estivesse salvo como juros (repare que juros é o nome de uma variável dentro do *script*). Como o Matlab salva na Área de Trabalho as variáveis do *script*, ao digitar juros (com a intenção de executar o script) o Matlab ficaria confuso se você deseja visualizar o valor da variável juros ou executar o *script*. Por isso que o nome do *script* não pode ser igual ao nome de qualquer variável dentro do *script*
- Note que é possível deixar comentários (trechos que não serão interpretados pelo Matlab) no *script*. Para colocar um comentário basta colocar o símbolo % na frente do que se deseja comentar. O símbolo % diz ao Matlab para ignorar todo o restante daquela linha

3.2 Funções

Diferente dos *scripts*, as funções devem ser declaradas de uma maneira especial:

```
function variavel_saida = nome_funcao(lista_argumentos)
% comentarios de ajuda da funcao
declarações
variavel_saida = valor
```

Para criar uma função, clique no botão Novo e depois em Função. Será aberta uma janela de Edição para que possa ser definida a função. Toda função deve **obrigatoriamente** começar com a palavra *function*. Depois vem uma variável responsável por entregar o resultado de saída. O nome dado a função deve ser o mesmo nome que será salvo o arquivo da função. Por exemplo, se criar uma função chamada "media" e salvar com o nome "contas", quando for chamar a função na Janela de Comandos deverá digitar "contas" ao invés de "media". Isso pode gerar dúvidas, por isso é recomendado que o nome da função seja o mesmo do nome do arquivo salvo. Por fim, deve-se colocar a lista de argumentos que serão passados para a função. Tomando como exemplo o *script* mostrado anteriormente, vamos transformá-lo em uma função. Sabemos que a variável de saída será total e que devemos passar como parâmetros os valores de investimento, juros e meses.

A Figura 24 mostra como ficou definida a função. Para chamar a função criada basta digitar o nome dela na Janela de Comandos, passando entre parênteses os valores que

serão utilizados por ela. Note que, diferentemente o *script*, os valores utilizados na função não são salvos na Área de Trabalho (eles existem somente dentro da função). Se quisermos salvar o resultado dessa função, devemos atribuir a saída da função a uma variável. Por exemplo: $valorFinal = rendimento(15000, 0.01, 10)$.

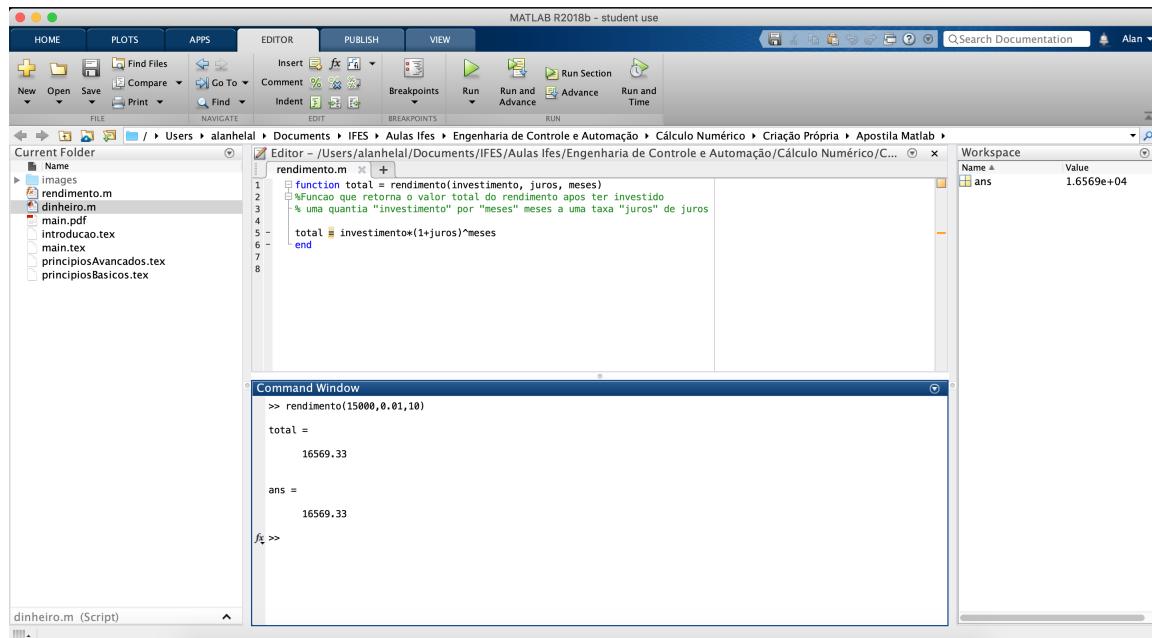


Figura 24 – Criando e executando uma função

É importante ressaltar que a variável de saída **obrigatoriamente** deve conter algum valor. É possível criar funções que contenham mais de uma variável de saída, conforme demonstrado na Figura 25.

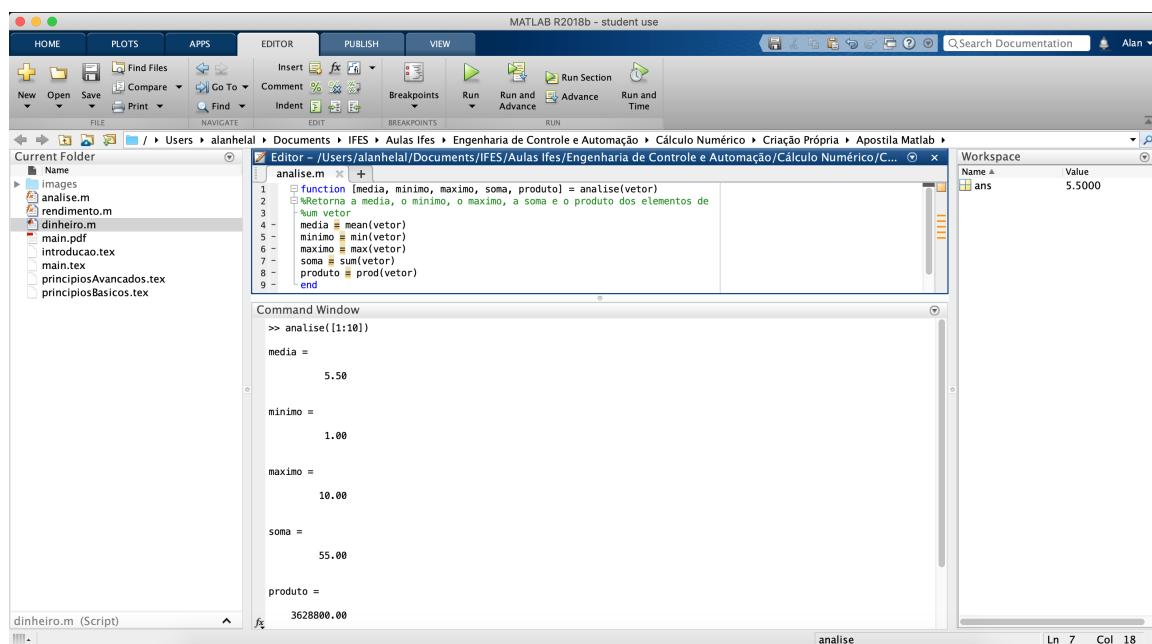


Figura 25 – Criando e executando uma função com múltiplas saídas

As seguintes combinações de funções podem ser criadas:

- Uma saída e uma entrada
- Uma saída e zero entradas
- Uma saída e múltiplas entradas
- Zero saídas e uma entrada
- Zero saídas e zero entradas
- Zero saídas e múltiplas entradas
- Múltiplas saídas e uma entrada
- Múltiplas saídas e zero entradas
- Múltiplas saídas e múltiplas entradas

3.2.1 Sub-função

Ao criar um novo arquivo de função, é possível definir mais de uma função nesse mesmo arquivo. Ao definir mais de uma função em um mesmo arquivo, a primeira função definida é a função principal (função que poderá ser invocada pela Janela de Comandos) e as demais funções são sub-funções. As sub-funções somente podem ser acessadas pela função principal e só existem dentro do escopo da função principal.

Para exemplificar, vamos criar um novo arquivo de função e definir uma função principal que recebe um vetor de números e retorna a média dos elementos desse vetor. Apesar de já existir uma função nativa do Matlab que nos diz a média dos elementos de um vetor, iremos implementar essa função.

A Figura 26 mostra essa função implementada (chamada calcula media). Repare que dentro da função calculamedia é chamada uma outra função (mediavetor). Essa função mediavetor é uma sub-função. Ao executar a função calculamedia na Janela de Comandos temos como resposta o valor da média dos elementos do vetor que foi passado como parâmetro. Note que ao tentar invocar a sub-função mediavetor na Janela de Comandos, o Matlab retorna um erro informando que tal função não está definida. Ou seja, a função principal (calculamedia) consegue invocar a sub-função (mediavetor) mas não é possível invocar diretamente a sub-função.

3.3 Entrada e saída de dados para o usuário

Conforme visto no *script*, os dados a serem calculados foram previamente digitados no *script*. Mas é possível obter esses dados do usuário no momento da execução do *script*.

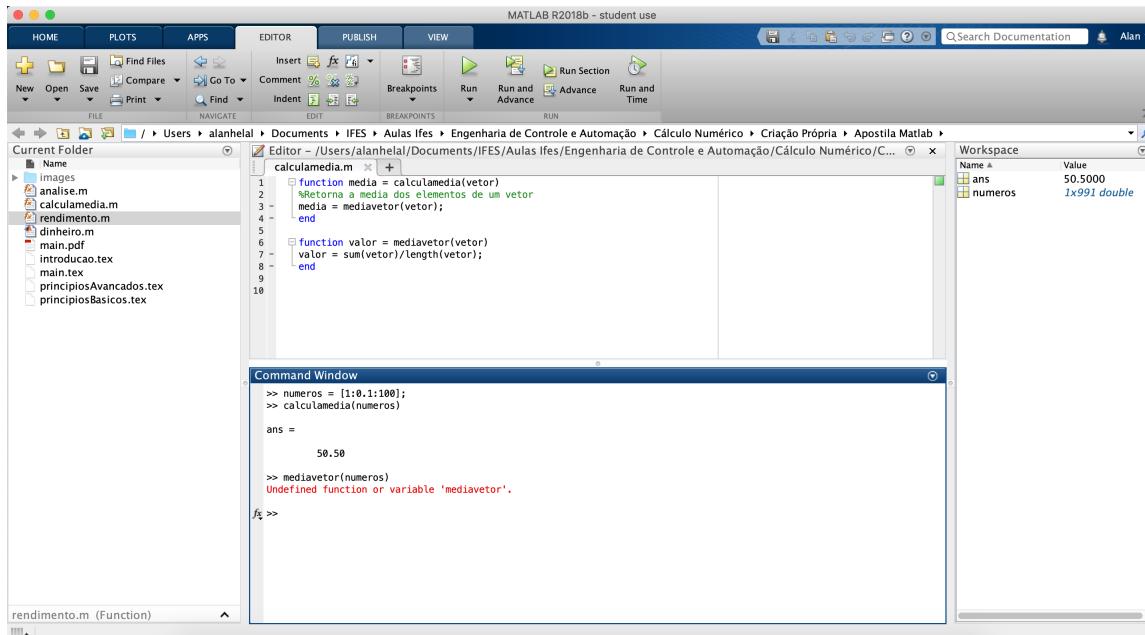
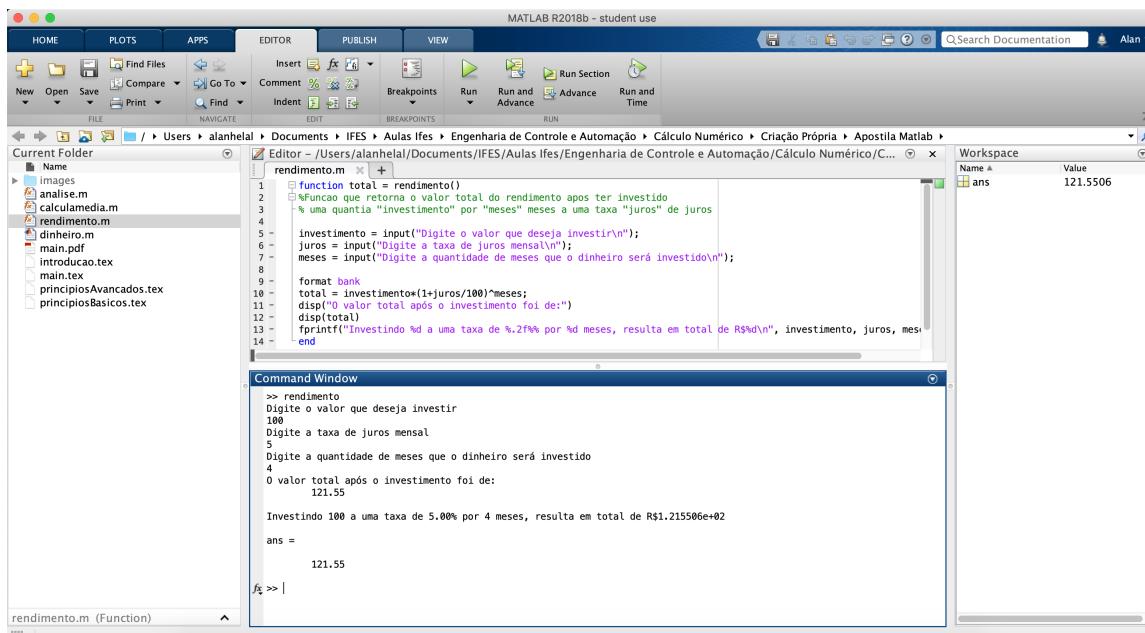


Figura 26 – Criando e executando uma sub-função

Para isso será utilizado a função *input*. Essa função, nativa do Matlab, imprime um texto na Janela de Comandos e espera o usuário digitar o valor desejado e apertar enter. A Figura 27 mostra *script* utilizado para calcular o rendimento, modificado para receber os parâmetros de entrada pelo usuário.

Figura 27 – Execução de *script* utilizando entrada de dados pelo usuário através da função *input*

Ainda na Figura 27 podemos observar que existem dois métodos de mostrar uma mensagem para o usuário. Primeiro utilizando a função *disp*. Com ela é possível imprimir

uma mensagem na Janela de Comandos **ou** imprimir o valor de uma variável¹. Repare na Figura 27 que foi utilizado o comando *disp* para imprimir na Janela de Comandos a mensagem: "O valor total após o investimento foi de:". Um ponto importante do *disp* é que ele automaticamente coloca uma quebra de linha (o equivalente a apertar um enter para ir para a próxima linha). Para imprimir o valor total do investimento, foi utilizado o comando *disp* para imprimir o valor da variável total. O comando *disp* é útil quando se deseja imprimir na Janela de Comandos mensagens informativas.

O outro método é utilizando a função *fprintf*. Essa função permite imprimir informações na Janela de Comandos de uma maneira mais completa e informativa. Repare na Figura 27 que foi utilizada a função *fprintf* para imprimir uma mensagem em que no meio da mensagem foi impresso valores de variáveis. Dessa maneira, a mensagem impressa no final é bem mais informativa ("Investindo 100 a uma taxa de 5.00% por 4 meses, resulta em um total de R\$121.55").

Para dizer ao Matlab que no meio da *string*, que será impressa pela função *fprintf* na Janela Comandos, existe uma variável são utilizados códigos de formato. Diferentemente do *disp*, o *fprintf* não insere uma quebra de linha, devendo ser explicitamente informado através de códigos de controle. A Tabela 2 mostra os principais códigos de formato e controle.

Código de Formato	Descrição
%d	Será impresso o valor de uma variável inteira
%f	Será impresso o valor de uma variável de ponto flutante
Código de Controle	Descrição
\n	Indica que deve iniciar uma nova linha (enter)
\t	Indica que deve realizar uma tabulação horizontal

Tabela 2 – Códigos de formato e controle

É importante ressaltar que o modo que será impresso o número também pode ser alterado utilizando *fprintf*. Por exemplo, considere o número $\pi = 3.14159265359$. Suponha que queiramos imprimir seu valor utilizando a função *fprintf* porém mostrando somente até a sua quinta casa decimal. Conforme vimos na Tabela 2, para imprimir um número de ponto flutuante utilizamos o código de formato %f. Para indicar que queremos apenas até a quinta casa decimal, colocamos ".5" entre o % e o f". Dessa forma, o Matlab ao identificar %0.5f irá imprimir 3.14159 (com apenas 5 casas decimais).

3.4 Programação Estruturada

Na criação de *scripts* e *funções* muitas vezes é preciso tomar decisões durante a execução ou repetir diversos trechos de códigos. Nessa seção iremos aprender sobre duas estruturas

¹ É possível concatenar texto e utilizar funções de type casting de números para string e imprimir tudo de uma vez utilizando o disp. Mas esse não é o foco dessa apostila.

de tomada de decisões (*if* e *switch*) e duas estruturas de repetição (*for* e *while*).

3.4.1 Estruturas de Tomada de Decisões

3.4.1.1 if, if...else, if...elseif

O *if* é o condicional "se". O Matlab irá realizar alguma operação se a condição for verdadeira (ou falsa, depende de como foi programado). Pode ser utilizado de três maneiras distintas: *if* (se), *if...else* (se...senão) ou *if...elseif* (se...senão...).

Para todos os casos devemos utilizar uma condição que ou é verdadeira ou é falsa. Começando pela estrutura *if*, sua sintaxe genérica é a seguinte:

```
if condição
    declarações
end
```

A condição (tanto para o *if* quanto para as demais estruturas de tomada de decisões e estruturas de repetição) devem conter operadores relacionais. Operadores relacionais comparam a relação entre dois valores e retorna se é verdadeiro ou falso. Vale ressaltar que valores booleanos² também podem ser utilizados, sendo o valor booleano 1 para verdadeiro e o valor booleano 0 para falso. A Tabela 3 mostra os operadores relacionais mais utilizados³.

Exemplo	Operador Relacional	Relação
$x == 0$	$==$	Igual
$\text{divisor } \sim= 0$	$\sim=$	Diferente
$\text{nota} < 60$	$<$	Menor que
$\text{nota} > 60$	$>$	Maior que
$(\text{media} + \text{final})/2 \geq 60$	\geq	Maior ou igual a
$\text{faltas} \leq 25$	\leq	Menor ou igual a

Tabela 3 – Operadores relacionais mais utilizados

É possível verificar mais de uma condição ao mesmo tempo. Para isso utiliza-se as condições lógicas **Negação**, **E** e **Ou**, conforme apresentadas na Tabela 4. Assim como nos operadores aritméticos, existe uma ordem de prioridade para a avaliação desses operadores lógicos. O operador de mais alta prioridade é a **Negação**, seguido do operador lógico **E** e o operador lógico **Ou** sendo o de menor prioridade. O Matlab ao se deparar em uma situação em que deve avaliar operadores lógicos de mesma prioridade, será avaliado da esquerda para a direita. Assim como nos operadores aritméticos, pode-se utilizar parênteses para alterar essa ordem de prioridade.

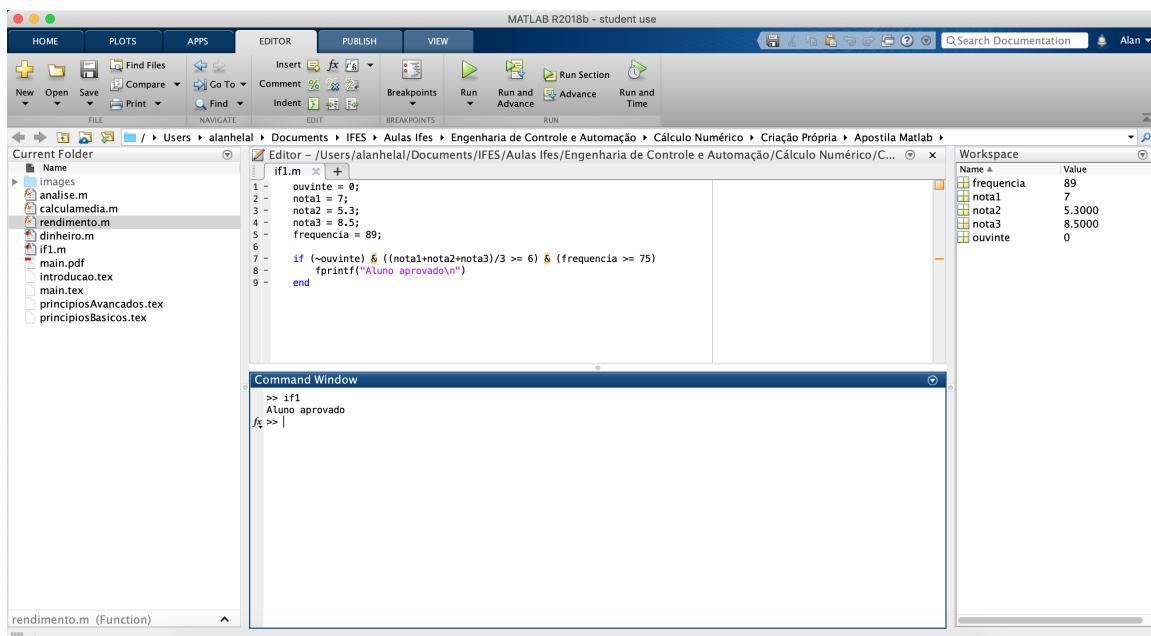
² Para maiores informações sobre bool: <https://pt.wikipedia.org/wiki/Booleano>

³ Esses operadores serão utilizados nas estruturas de tomada de decisão e de repetição que serão apresentadas nessa apostila

Exemplo	Condição Lógica	Relação
$\sim \text{expressao}$	\sim	Nega uma condição lógica
nota & frequencia	&	Ambas as condições verdadeiras
expressao1 expressao2		Pelo menos uma condição verdadeira

Tabela 4 – Condições Lógicas

A Figura 28 mostra um exemplo de utilização do *if* com três expressões lógicas a serem avaliadas. Nesse exemplo é verificado se o aluno não é ouvinte, se a média aritmética das notas obtidas pelo aluno é maior ou igual que 60 e se a frequência do aluno é maior ou igual a 75. Se essas três condições forem verdadeiras, é apresentado a mensagem "Aluno aprovado". Repare que na variável ouvinte, foi utilizado o valor booleano 0 para dizer que é falso. Então o operador lógico **Negação** nega esse valor (a negação de falso é verdadeiro, e vice versa).

Figura 28 – Exemplo de aplicação do *if*

A Tabela 5 mostra duas variáveis x e y e o resultado da avaliação das três condições lógicas (Negação, E e Ou) dependendo dos valores de x e y .

x	y	$\sim x$	$x \& y$	$x y$
Verdade	Verdade	Falso	Verdade	Verdade
Verdade	Falso	Falso	Falso	Verdade
Falso	Verdade	Verdade	Falso	Verdade
Falso	Falso	Verdade	Falso	Falso

Tabela 5 – Avaliação das três condições lógicas entre duas variáveis

O outro método é a estrutura de condição *if...else*. Sua sintaxe genérica é a seguinte:

```
if condição
    declarações
else
    outras_declaracões
end
```

Nela, é verificado uma condição e, se for verdadeira, são realizadas as declarações que se encontra entre o *if* e o *else*. Caso essa condição não seja verdadeira, são realizadas as declarações que se encontram entre o *else* e o *end*. A Figura 29 mostra um exemplo dessa situação. Nela é verificado se o divisor é igual a zero. Se for igual a zero é apresentada uma mensagem de erro na Janela de Comandos. Para **qualquer outro valor** que não seja zero, é realizada a divisão de 4 pelo divisor escolhido.

Repare que a função *error* apresenta uma mensagem de erro na Janela de Comandos. Na mensagem é mostrada a linha em que houve o erro e a mensagem escolhida pelo programador para ser mostrada. Um ponto importante de utilizar a função *error* é que ela **termina a execução do programa imediatamente**. Ou seja, o Matlab ao encontrar esse comando, para de executar o *script* ou *função*.

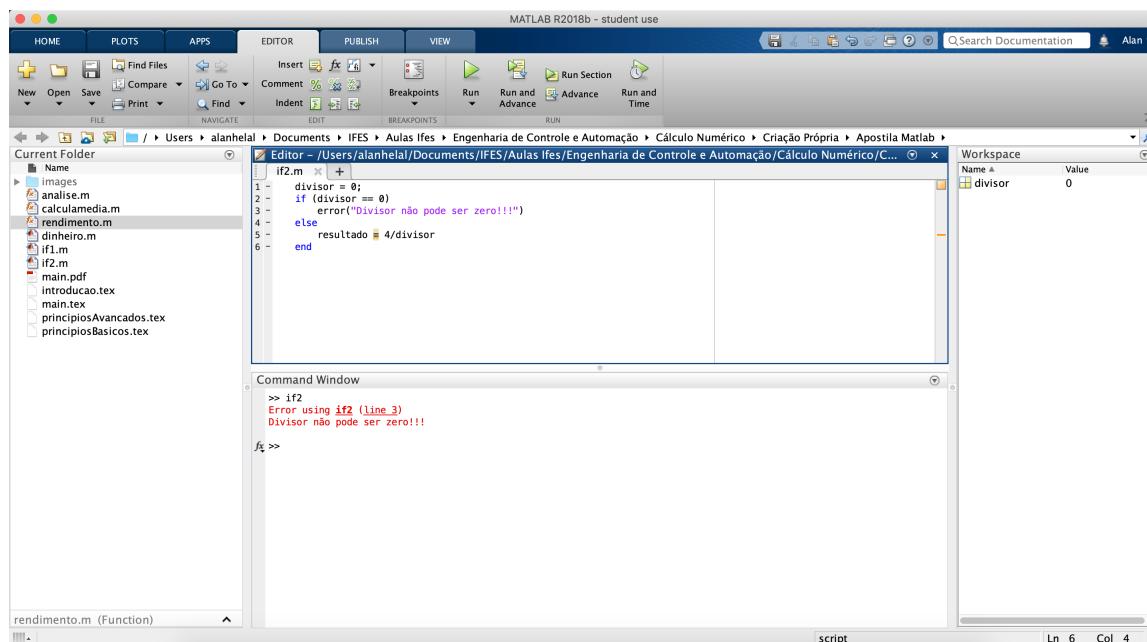


Figura 29 – Exemplo de aplicação do *if...else*

A situação acima é utilizada quando precisamos analisar se uma condição é verdadeira e, se não for, qualquer outra situação não precisa ser analisada. Por exemplo, não se pode dividir um número por zero, então só precisamos testar se o divisor é zero, qualquer outro valor atende a divisão (nesse exemplo em que é dividido 4 pelo divisor). Entretanto, existem situações em que diversas condições precisão ser analisadas. Para isso utilizamos o *if...elseif* (se...senão...). Abaixo segue a sintaxe genérica dessa estrutura:

```

if condição1
    declarações1
elseif condição2
    declarações2
.
.
.
else
    declarações
end

```

Nela é testada a primeira condição e, se for verdade, são executadas as declarações1. Se não for verdade, é testada a segunda condição e, se for verdade, são executadas as declarações2. E assim suscetivelmente até chegar no *else* (caso exista algo a se fazer caso nenhuma das condições anteriores seja verdadeira) ou no *end* (finalizando o *if*).

A Figura 30 mostra uma situação em que diversas análises devem ser levadas em conta. Imagine que você está decidindo se vai sair ou não. Se seu amigo for, você sai não importando o lugar que estão indo. Se seu amigo não for, você só sai se for para um lugar legal. Se seu amigo não for e o local não for legal, você só sai se estiver sol e você tiver dinheiro. Se as situações acima não forem verdade, você não sai de casa. Repare que no exemplo, para dizer se as variáveis analisadas são verdadeiras ou falsas, foram utilizados números booleanos (1 ou 0). Como temos quatro variáveis e cada uma pode ter dois valores distintos, temos um total de 16 combinações possíveis. Mas repare que dessas 16 combinações, apenas 3 situações são importantes para a sua escolha. Por isso temos 3 verificações sendo realizadas (uma no *if* e duas nos *elseif*). Já para todas as outras combinações não importa pois você ficará em casa. Por isso foi colocado o *else* para aglomerar todas essas combinações que resultam na mesma situação: você ficar em casa.

3.4.1.2 switch

Diferente da estrutura *if*, a estrutura *switch* testa diferentes valores de uma mesma variável. Dependendo do valor dessa variável testada, diferentes ações podem ser tomadas. A estrutura *switch* tem a seguinte sintaxe genérica:

```

switch variavel
case valor1
    declarações1
case valor2
    declarações2
case valor3
    declarações3

```

```

otherwise
declarações
end

```

Imagine uma situação em que o usuário digite dois números. Depois ele deve escolher qual operação matemática quer realizar entre esses dois números. Suponha que 1 seja para somar, 2 para subtrair, 3 para multiplicar e 4 para dividir. Qualquer outro valor que for digitado deve ser mostrado uma mensagem dizendo que esse valor é inválido. Para isso podemos utilizar a estrutura *switch*. Um exemplo é demonstrado na Figura 31 em que o usuário digita o número 10, depois o número 2 e escolhe a opção 2(subtração). O Matlab então retorna o resultado de $10 - 2 = 8$.

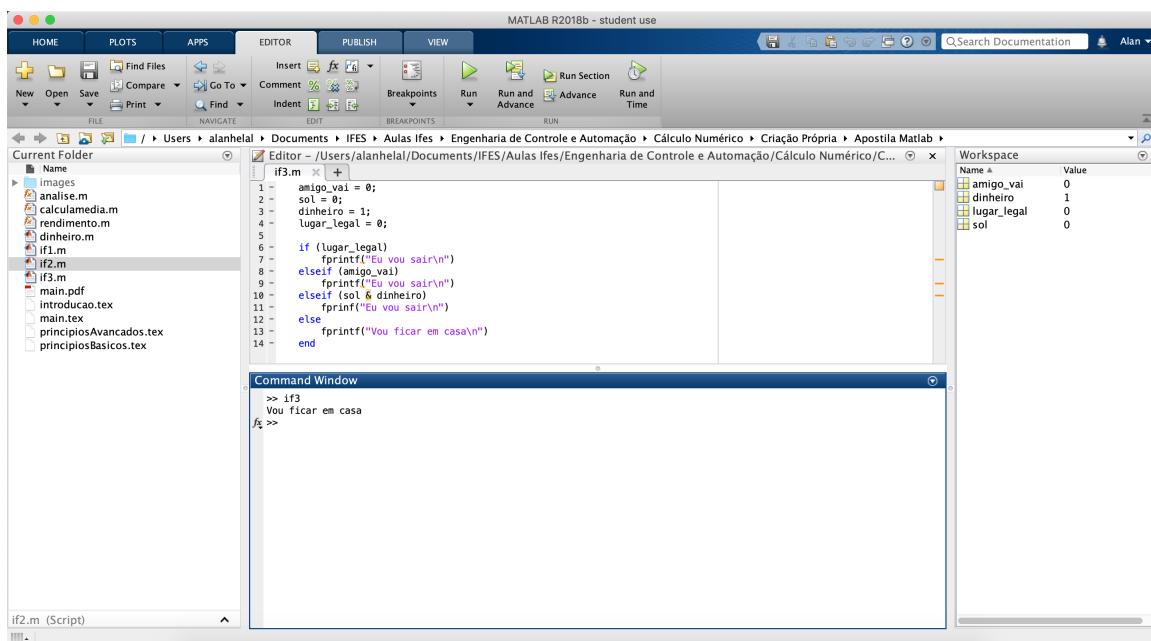


Figura 30 – Exemplo de aplicação do *if...elseif*

3.4.2 Estruturas de Repetição

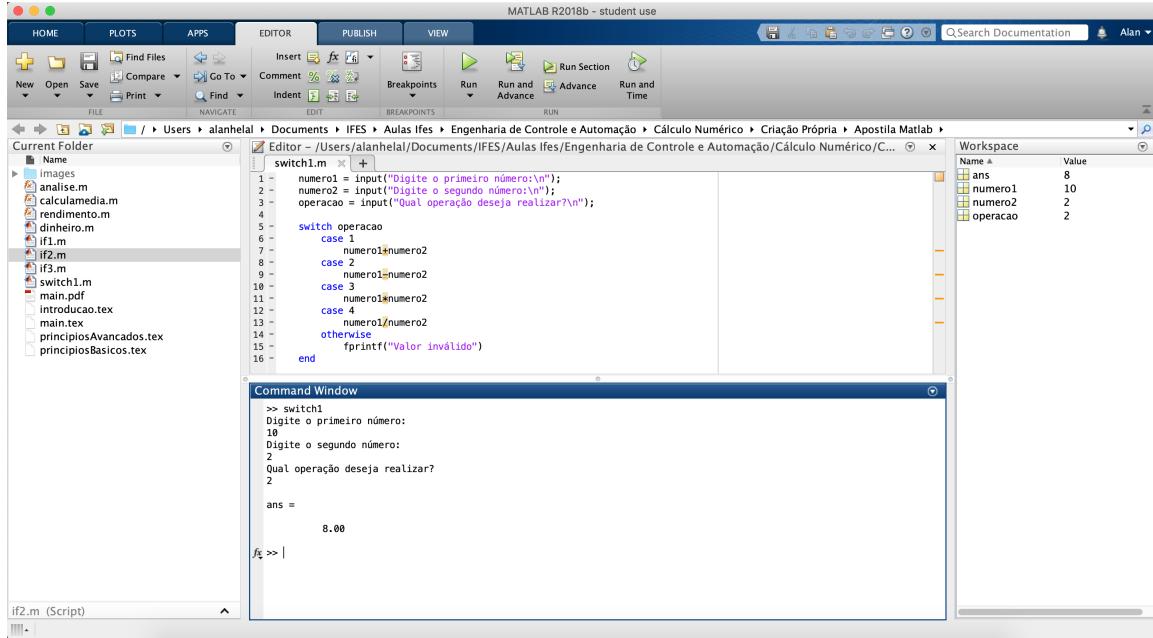
3.4.2.1 for

O *for* é uma estrutura de repetição em que se repete os comandos inseridos nele por um número específico. Sua sintaxe genérica é:

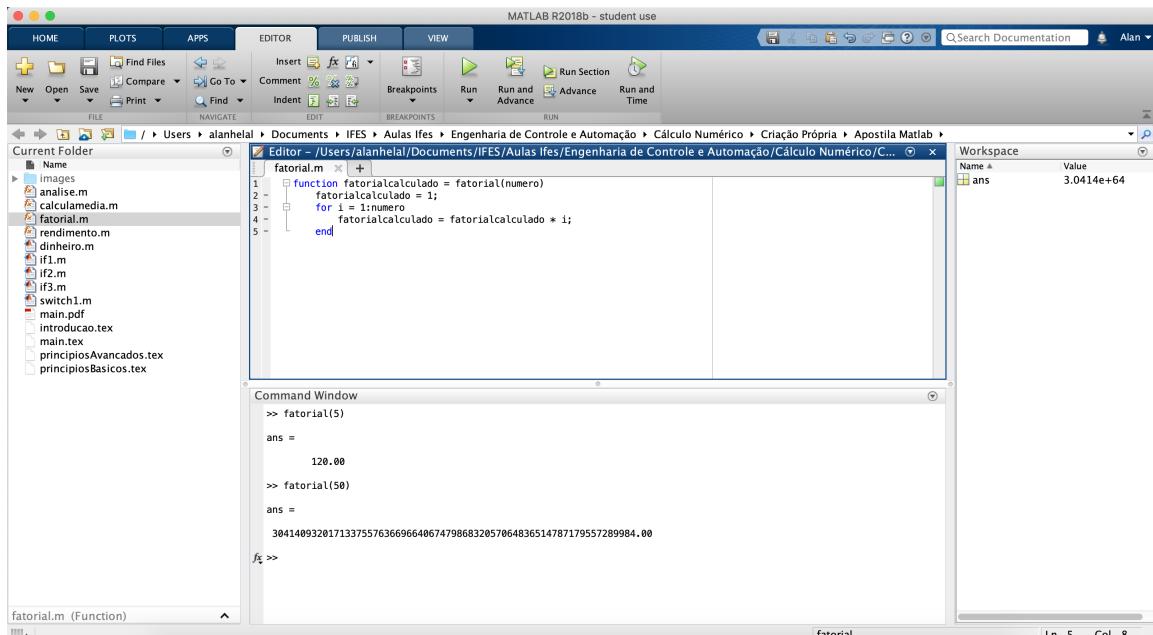
```

for indice = inicio:passo:fim
declarações
end

```

Figura 31 – Exemplo de aplicação do *switch*

Lembre-se de quando aprendemos sobre o caractere ":". Se ao digitar um intervalo e não colocar o número de passos, será utilizado incremento de 1. $1:5 = 1:1:5 = 1\ 2\ 3\ 4\ 5$. Um exemplo bastante didático dessa estrutura de repetição é a função para calcular o factorial de um determinado número, conforme mostrado na Figura 32. Observe que após definir a função factorial, podemos invoca-la para calcular diferentes fatoriais. Nesse exemplo foi calculado o factorial de 5 e 50.

Figura 32 – Exemplo de aplicação do *for*

3.4.2.2 while

A estrutura *while* se repete enquanto uma condição lógica for verdadeira. Sua sintaxe geral é a seguinte:

```
while condição
    declaração
end
```

Para que a execução saia dessa estrutura de repetição é necessário que a condição não seja mais verdadeira ou que seja explicitamente utilizado um comando para parar a execução dessa estrutura (estamos falando do comando *break* e será visto logo adiante).

Vamos modificar a função que calcula o fatorial de um número para utilização da estrutura *while*. Primeiro vamos usar um método que fará com que a condição deixe de ser verdadeira para acabar a execução do *while* e, depois, vamos utilizar o comando *break*.

A Figura 33 mostra a função fatorial modificada para utilizar o *while*. O número passado por parâmetro (número que se deseja calcular o fatorial) é atribuído a variável valor. Depois inicia o *while* baseado nessa variável (a cada iteração é verificada se a condição continua verdadeira). Nesse caso a condição verdadeira é o valor ser maior que zero). Dentro do *while* é realizada a multiplicação do valor atual com o valor anterior da variável factorialcalculado. Note na linha 6 que é subtraído 1 da variável valor. Essa é a nossa condição de parada. É devido a essa linha que uma hora a condição ($\text{valor} > 0$) deixará de ser verdadeira e, consequentemente, acabará a estrutura de repetição *while*.

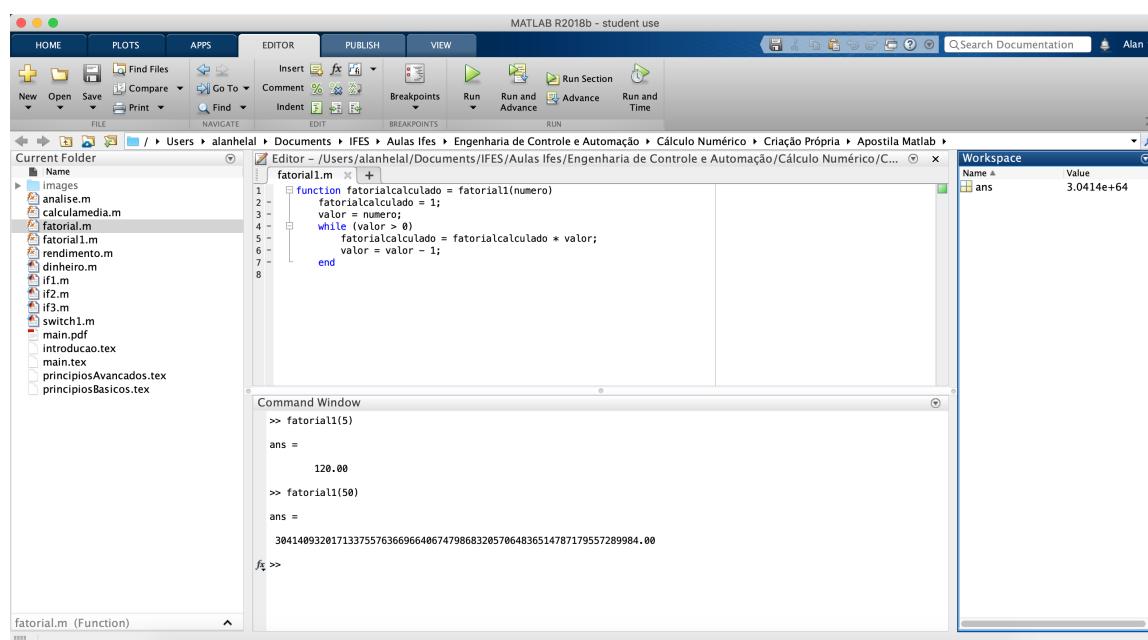


Figura 33 – Exemplo de aplicação do *while*

Vamos agora modificar essa função para utilizar como método de parada o comando *break*. Repare na Figura 34 que o *while* foi iniciado baseado na condição 1. Ou seja, ele

sempre será verdade! Não tem como você modificar essa condição para falso. O programa ficará preso dentro dessa estrutura eternamente. Para que isso não ocorra, é realizada uma verificação da variável valor e, se ela for igual a zero, é executado o comando *break*. Repare que nos três casos (utilizando *for*, *while* ou *while...break*) foi possível obter o mesmo resultado. A escolha de qual estrutura de repetição utilizar fica a cargo do programador sendo que não existe uma estrutura absolutamente melhor do que a outra. Cada uma tem suas vantagens dependendo da solução que está sendo utilizada.

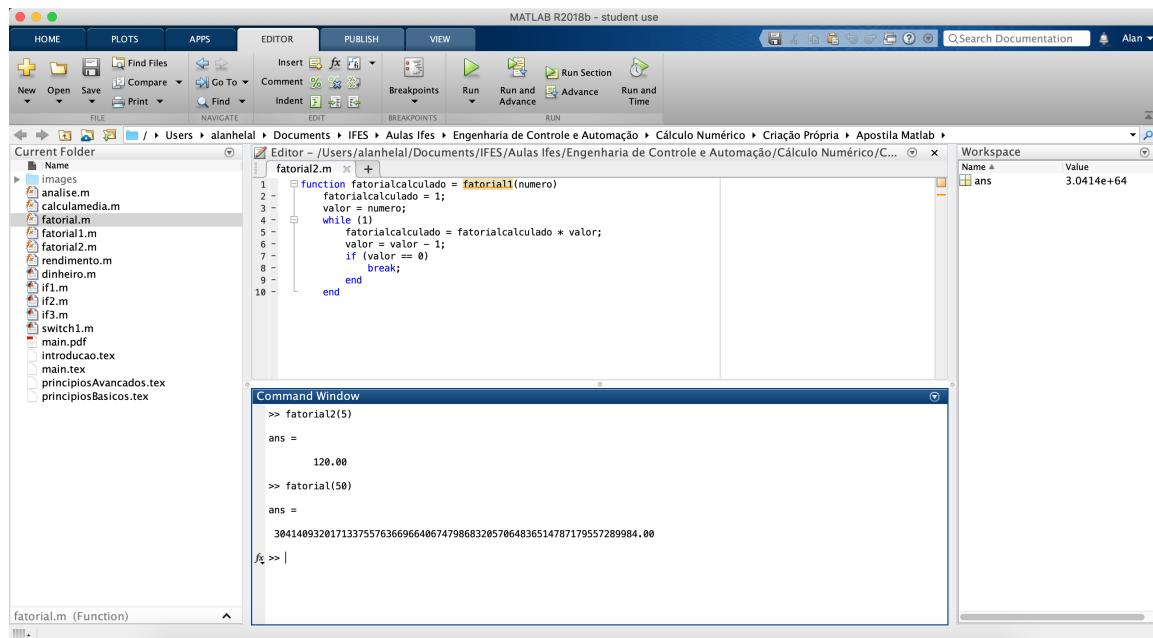


Figura 34 – Exemplo de aplicação do *while...break*

3.4.3 Função Anônima

Funções anônimas permitem criar funções matemáticas simples sem precisar salvar em um arquivo (seja *script* ou função). Elas podem ser definidas e acessadas pela Janela de Comandos. Sua sintaxe geral é:

```
nome_funcao = @(lista_variaveis) expressao
```

Repare que na Figura 35 foram definidas duas funções anônimas (*funcao1* e *funcao2*). A primeira possuindo apenas como variável *x*, e a segunda tendo *x*, *y* e *z* como variáveis. Uma vez definido é possível invocar essas funções e passar como parâmetro o valor que se deseja atribuir a variável da função.

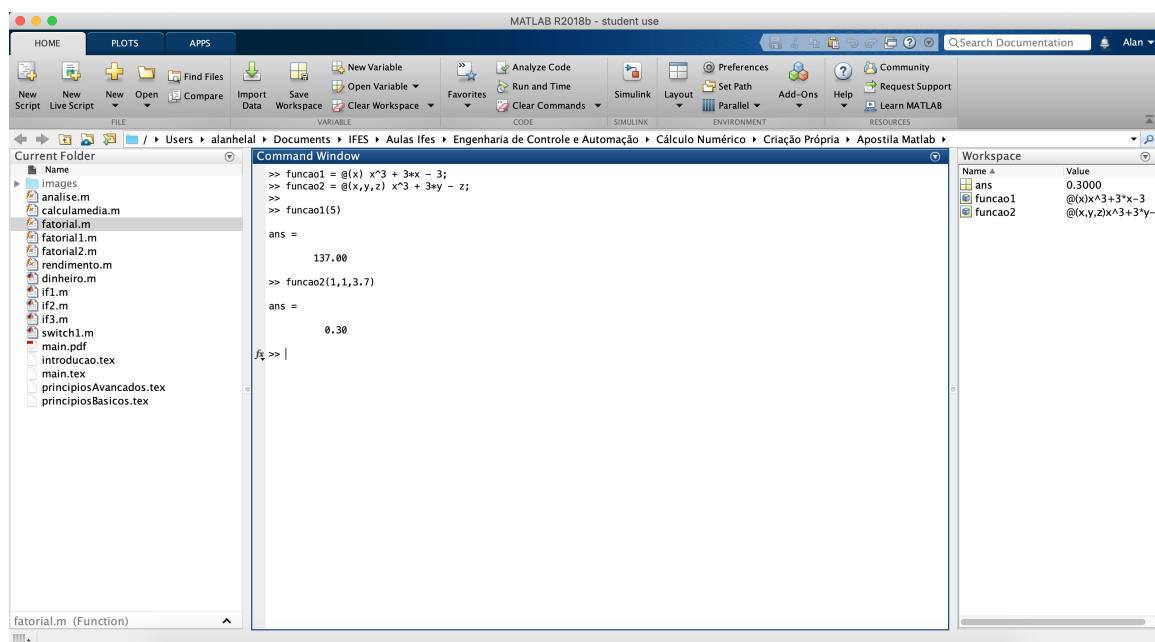


Figura 35 – Exemplo de definição de função anônima

4 Considerações Finais

4.1 Octave

O GNU Octave é uma linguagem de alto nível, destinada principalmente a cálculos numéricos. Ele fornece uma interface de linha de comando conveniente para resolver problemas lineares e não-lineares numericamente e para realizar outros experimentos numéricos usando uma linguagem que é mais compatível com o Matlab. Também pode ser usado como uma linguagem orientada por lote.

O Octave possui extensas ferramentas para resolver problemas comuns de álgebra linear numérica, encontrando as raízes de equações não-lineares, integrando funções comuns, manipulando polinômios e integrando equações diferenciais e algébricas-diferencial comuns. É facilmente extensível e personalizável através de funções definidas pelo usuário escritas no próprio idioma do Octave, ou usando módulos carregados dinamicamente escritos em C++, C, Fortran ou outros idiomas.

O GNU Octave também é um software livre para redistribuição. Você pode redistribuí-lo e / ou modificá-lo sob os termos da Licença Pública Geral GNU (GPL), conforme publicada pela Free Software Foundation.

Octave foi escrito por John W. Eaton e muitos outros. Como o Octave é um software livre, você é encorajado a ajudar a tornar o Octave mais útil escrevendo e contribuindo com funções adicionais para ele, e relatando qualquer problema que possa ter¹.

O Octave pode ser instalado no Windows, Linux ou Mac e pode ser obtido gratuitamente em sua página oficial da internet². Caso não queira instalar o Octave, é possível utilizá-lo em uma versão WEB³. Todo o conhecimento expresso nessa apostila funciona perfeitamente no Octave.

¹ <https://www.gnu.org/software/octave/about.html>

² <https://www.gnu.org/software/octave/download.html>

³ <https://octave-online.net>