4Bytes
Treinamentos

MINICURSO MATLAB

IGOR AMORIM SILVA

CUIDADO!!!

O conteúdo a seguir é muito viciante! :)

SUMÁRIO

1. INTRODUÇÃO

1.1. O que é o Matlab?

Matlab, de Matrix Laboratory, é um ambiente interativo para computação envolvendo matrizes. Matlab foi desenvolvido no início da década de 80 por Cleve Moler, no Departamento de Ciência da Computação da Universidade do Novo México, EUA. Seu principal objetivo era resolver problemas que envolvessem matrizes, vetores, Álgebra Linear, etc.

1.2. O que ele faz?

Atualmente, o Matlab chega a abranger a maioria das ciências (Engenharias, Biologia, Química, Matemática, Física etc.), estando presente em tópicos já fundamentados nas ciências e até em alguns tópicos emergentes como Big Data e Visão Computacional...

1.3. O que ele não consegue fazer?

Apesar de apresentar inúmeras possibilidades, o Matlab não faz mágicas nem milagres, devemos sempre levar em conta: os requisitos da máquina utilizada, os requisitos do que se pretende desenvolver e o custo final do projeto. É importante lembrar que o Matlab em alguns casos pode não ser a melhor escolha de ferramenta, levando em conta os fatores citados.

1.4. Por que usar Matlab?

Aberto a interpretações pessoais.

2. INTERFACE DO MATLAB

2.1 HOME

Nesta aba encontram-se as principais opções referentes, a criação de novos arquivos, controle de variáveis de ambiente, layout do Matlab, preferências, Add-Ons e Parallel Pool.

2.2 PLOTS

Nesta aba são apresentados os diferentes tipo de plots de gráficos possíveis no Matlab, alguns necessitando ser chamado de forma diferenciada do habitual "plot(x,y)".

2.3 APPS

Apresenta os aplicativos criados pela MathWorks, sua comunidade ou pelo usuário (você). Os aplicativos em sua definição mais simplificada, representam um conjunto de funções apresentados em um interface.

2.4 Editor

No Editor, podemos encontrar grande parte dos controles necessários a edição de um script (*.m): indentação, comentários, inserções de funções, seções, etc.

2.5 View

Opções de visualização e definições e layout do Matlab.

2.6 Diretórios

Mostra o diretório atual ou caso seja a primeira utilização, o diretório padrão do Matlab. Para que seus scripts possam ser "encontrados" pelo Matlab eles devem estar em uma pasta no diretório atual ou estar na pasta PATH do Matlab (a segunda opção pode ser perigosa caso não se tenha um bom conhecimento da ferramenta).

2.6 Workspace

De forma análoga a um sistema operacional, no Workspace encontram-se as variáveis declaradas (ou variáveis de ambiente em um S.O.), é importante notar que o Matlab mostra o nome, valor (que foi atribuido à variável) e valores mínimos e máximos. No caso de variáveis simples, apenas o valor que foi atribuido a variável.

2.7 Command Window

Local para execução dos comandos de forma interativa.

3.1 Variáveis

Baseado no que já foi visto até agora, você já deve ter um noção de como é a interface do Matlab. Agora, avançaremos para um dos alicerces do aprendizado de Matlab, as variáveis.

Assim como Python e JavaScript, o Matlab facilita e muito a definição de variáveis por apresentar um definição de tipo dinâmica, ou seja, a mesma variável que foi declarada como um Integer, pode em outro momento estar recebendo uma String.

Vamos ver isso na prática, abra seu Matlab e na Command Window digite:

```
>> a=10
```

O resultado esperado deve ser:

```
a =

    10
>>
```

Note que no Workspace a variável declara apareceu. Agora, faremos uso da mesma variável recebendo uma String:

```
>> a='4Bytes'
```

Resultando em:

```
a =
4Bytes
>>>
```

Se você já possui alguma experiência com linguagens de programação, deve ter percebido que não foi usado ";". O motivo disto é porque para o Matlab o ";" serve para suprimir o feedback da saída, ou seja, ele executa um comando e ao invés de mostrar o que resultado ele apenas espera pelo próximo comando a ser digitado. Para ver isso na prática, digite os mesmos comandos (em sequência) com o diferencial do ";" ao final dos comandos.

3.2 Operadores

Igual a muitas linguagens, os operadores básicos no Matlab não mudam sua declaração e uso, por exemplo:

```
>> 1233 + 3434
ans =
      4667
>> 1233 - 3434
ans =
     -2201
>> 1233 / 3434
ans =
   0.3591
>> 1233 * 3434
ans =
    4234122
>>
```

3.3 Palavras Reservadas

As palavras reservadas no Matlab seguem um padrão parecido a diversas linguagens como C, Python, JAVA, JavaScript e outras, no entanto é preciso atentar que não podemos criar Scripts ou funções (a serem vistos posteriormente) sem gerar um conflito com funções já padrão do Matlab (não tente fazer isso, só gera problemas).

Exemplos de palavras reservadas:
for
if
else
while
end
... (sim isto é uma palavra reservada)
switch

[NOTA] Não serão mostrados exemplos de cada uma das palavras reservadas porque elas serão postas em prática nos exercícios posteriores.

4. VETORES

Chegamos em um dos alicerces na história do Matlab. Ainda hoje eles fazem um papel muito importante na em qualquer aplicação feita em Matlab, mas como são declarados ?

Existem n formas de se declarar um vetor no Matlab, a melhor maneira dependerá do que você pretende fazer com ele. Seguem-se alguns exemplos de declaração de vetores de vários tipos:

```
>> vetor=['alguma coisa']
vetor =

alguma coisa

>> vetor=[1 2 3 4 5 6 7 8 9]

vetor =
    1    2    3    4    5    6    7    8    9

>> vetor=[1 2 3 4 5 6 7 8 9]

vetor =

    1    2    3    4    5    6    7    8    9

>> vetor=1:10

vetor =

    1    2    3    4    5    6    7    8    9   10
```

4. VETORES

As mesmas operações que fizemos anteriormente com variáveis simples podem ser realizadas com vetores, só deve ser levado em conta que a multiplicação e divisão são um pouco diferentes, como será visto no próximo tópico.

4.1 Operações com Vetores

Você talvez tenha reparado que ao realizarmos alguma operação matemática sem guardar o resultado em uma variável o Matlab retorna uma variável "ans" guardando o que foi calculado. Essa variável é definida automaticamente pelo Matlab quando uma variável de resultado não foi especificada, você pode fazer as mesmas ações que uma variável normal, entretanto tome cuidado pois ela será atualizada automaticamente sempre que você executar algum comando sem guardar seu resultado (se possuir) em alguma variável definida por você.

O Matlab também possui alguns operadores especiais utilizados para multiplicar e dividir matrizes e vetores. Um deles é o ".*" que multiplica um vetor ou matriz por outro elemento por elemento.

Outro operador especial é o "./" ou ".\", que apesar de serem parecidos possuem usos e resultados diferentes. Ambos são muito importantes quando trabalhamos com matrizes e vetores.

Agora, veremos ambos os exemplos na prática:

```
>>> a=[1 2 3]; b=[4 5 6];

>>> a.*b
 ans =

    4    10    18
```

Mesmas variáveis para divisão:

```
>>> a=[1 2 3]; b=[4 5 6];
>>> a./b
 ans =

    0.2500    0.4000    0.5000
```

Utilizando ".\":

```
>>> a=[1 2 3]; b=[4 5 6];
>>> a.\b
 ans =

    4.0000    2.5000    2.0000
```

O que deu errado ?

Primeiramente, se acalme, tome um café e respire fundo. Seu Matlab não está "bugado", isso aconteceu porque apesar dos comandos "./" e ".\" serem parecidos, eles interpretam os vetores em ordens diferentes.

No caso do "./" ele pega o valor na primeira posição do vetor a e divide pelo primeiro valor do vetor b enquanto o comando ".\" faz o contrário, pegando o valor na primeira posição do vetor b e divide por a.

5. SCRIPTS

Ufa! Você deve estar cansado por ter chegado até esta parte, felizmente, começamos a parte mais divertida no aprendizado de Matlab. Por isso tente ao máximo fixar os conteúdos abordados nas próximas seções pois serão de grande valia em seu desenvolvimento com a ferramenta.

Os Scripts ou (.m files) são o que podemos chamar de roteiros a serem seguidos pelo Matlab. Um Script pode conter diversas coisas ou fazer de uso de diversas ações para atingir um objetivo.

Bom, chega de enrolação, vamos para a prática, abra o arquivo entitulado "exemplo_script01.m" na pasta do seu material. Em sua command Window digite o nome no arquivo citado sem o ".m". E então, gostou do resultado ?

Scripts também podem servir para automatizar o uso de algumas funções do Matlab, abra o arquivo "exemplo_script02", analize o código (caso hajam funções que você não conhece, não tenha vergonha use "help [nome da função]"). Execute o script "exemplo_script02" na Command Window.

Modelo de entrada:

```
>>
Digite seu nome: '4';
Digite seu sobrenome:'Bytes';
```

6. FUNÇÕES

Após uma rápida passagem pelos Scripts, veremos as funções. Uma forma de inserir dados com maior velocidade e praticidade.

Para entendermos melhor o conceito de função no Matlab usaremos um conceito amplamente utilizado de caixa preta que temos uma entrada, um processo ocorre dentro dessa caixa e gera a saída desejada (ou não). Como você deve ter percebido anteriormente, para que um Script possa ser efetivo é necessário ter as variáveis declaradas ou recebê-las utilizando "input", contrariamente a isso, as funções podem receber diretamente os valores que você deseja colocar.

No arquivo "exemplo_funcao01" temos um exemplo bem simples de uma função que calcula os autovalores pelo método de Jacobi, não atente muito ao que o código faz, mas sim na forma em que foi organizado.

Perceba que há uma entrada:

```
function [Autovalores,Autovetores] = exemplo_funcao01(M,erro)
```

Uma definição que isso é uma função e seu nome:

```
function [Autovalores,Autovetores] = exemplo_funcao01(M,erro)
```

E finalmente, mas não menos importante, uma saída:

```
function [Autovalores,Autovetores] = exemplo_funcao01(M,erro)
```

Agora que finalizamos as explicações básicas sobre funções daremos início a função que será utilizada na criação de nossa interface nos tópicos seguinte, nosso objetivo será criar um solucionador de equações diferenciais ordinárias pelo método de Runge Kutta (por incrível que pareça são dois autores). Primeiramente veremos uma breve explicação de como esse método funciona.

Método de Runge-Kutta

Antes de partirmos para a parte matemática, faremos algumas perguntas iniciais. Por que usar um método? Quando é mais vantajoso uma solução aproximada? Quão certa é minha "solução"? Qual o melhor método, se existem tantos?

Ufa, muitas perguntas (hora de tomar um café). Respondendo as perguntas em ordem, o principal objetivo de se usar um método numérico é porque o custo de se achar uma solução exata iria requerer um tempo de processamento muito grande indo de encontro ao pouco tempo em que necessita desta resposta.

Baseado na resposta anterior, é mais vantajoso se ter um solução aproximada, quando o custo computacional para se calcular sua resposta exata ou você deseja analisar apenas uma parte da solução que atende ao seu problema.

Uma pergunta um tanto capciosa, o melhor método é aquele que resolve eficientemente seu problema, mas nem sempre o método que lhe apresenta a solução mais rápida será o melhor, por isso, quando se fala em métodos computacionais, deve-se levar em conta o erro.

O método de Runge-Kutta consiste em comparar um polinômio de Taylor para eliminar o cálculo das derivas, ou seja, a cada passo (ou iteração no caso do algoritmo) será gerada uma nova função assim por diante, podendo ser construidos para qualquer ordem alpha. Felizmente o Matlab facilita bastante nosso trabalho já possuindo implementações nativas do método. A função "ode23" e "ode45" utiliza o método de Runge-Kutta para solucionar EDO's não rígidas, para iniciantes em Matlab, utilizar essas duas funções pode ser um pouco trabalhoso, por isso normalmente utiliza-se a função "dsolve" que automatiza essas funções tornando mais fácil seu uso.

Modelo de declaração do dsolve:

```
>>> dsolve('D2y = x*y', 'x');
```

Umas das formas de se declarar o dsolve na Command Window é colocando uma string com a ED e uma outra string indicando a variável que será derivada.

7. GRÁFICOS

Já aprendemos a fazer scripts e funções bem legais no Matlab até agora, mas falta algo, ver o resultado em um gráfico. Neste tópico veremos como criar, salvar e configurar gráficos.

7.1 Tipos de Gráficos

Como citado no começo do material, na aba PLOTS do Matlab, encontram-se os principais as principais funções (não confundir com funcionalidade) de plot existentes no Matlab. Tendo como base o exemplo do tópico anterior, onde foi usado a função "dsolve" se você analisar o resultado pode perceber que requer um certo trabalho gerar o gráfico utilizando o comando "plot" (comando mais utilizado para criação de gráficos), para sanar esse problema foi desenvolvida uma função específica para gerar gráficos de funções, o "ezplot" é uma função de fácil uso que requer apenas a função de entrada e o intervalo onde será plotado o gráfico.

```
>> s=dsolve('Dy=2*y+t','y(0)=1','t')
s =

(5*exp(2*t))/4 - t/2 - ¼

>> ezplot(s,[-2,0])
```

7.2 PLOT

O comando "plot", é um dos mais utilizados para criação de gráficos a partir de dois valores (x,y), ou em alguns casos apenas o (x).

Uso do comando plot:

```
>>> a=0:0.1:50;

>>> plot(cos(a));
```

Se seu Matlab não tiver nenhum erro de instalação após executar estes comando será gerado o gráfico dessa função em uma nova janela ou figure. Caso esteja se perguntando o que o operador ":" faz, ele apenas cria um vetor de acordo com o intervalo entre seus extremos, no caso desse exemplo, ele cria um vetor de 0 a 50 com uma iter-

7.3 SUBPLOT

O comando subplot funciona plotando dois ou mais gráficos na mesma figure, o que facilita quando se necessita analisar vários gráficos de significados distintos sem misturá-los. Abra o arquivo e execute o script "exemplo_subplot" para ver o funcionamento do comando.

Significado das entradas:

```
subplot(2,1,1);
```

O número com a cor azul, representa o número de linhas (gráficos na horizontal). No número representado com a cor vermelha, representam-se o número de colunas (gráficos na vertical), e por fim o número em verde que significa o sequencial, por exemplo, esse foi o primeiro gráfico o próximo será o (2,1,2) e assim por diante.

7.4 Configurações de Gráficos

O assunto está muito legal mas os gráficos estão muitos estáticos vamos personalizá-los um pouco. O Matlab oferece todo um conjunto de features para configurações de gráficos que vão desde colocar título à alterar que partes do gráfico serão mostradas ou até mesmo alterar a forma de representar a função.

Veremos uma pequena lista das configurações e suas características:

hold - permite plotar mais de um gráfico na mesma figura

axis - define o tamanho do intervalo (min e max) em que será exibido o gráfico

color - cor do traçado do gráfico

title - título exibido acima do gráfico

legend - legenda da figura gerada

xlabel - descrição do eixo x

ylabel- descrição do eixo y

8. ARQUIVOS

Imagine a seguinte situação, você acaba de rodar uma função ou script que você fez (no meu caso um método computacional) e precisa levar os resultados para outro lugar que infelizmente não possui Matlab, e agora ? Você precisará copiar todos os resultados para um arquivo tendo o cuidado de não perder a formatação? Felizmente, isso não é necessário, pois podemos usar os arquivos no Matlab.

Existem dois principais tipos de arquivos a serem salvos no Matlab, os arquivos em texto e em imagem (nossos gráficos). Neste treinamento serão mostrados apenas esses dois, não serão vistas as técnicas de arquivo de video, audio e animação de gráficos (que pena).

8.1 Criação de Arquivos

O comando principal para impressão de arquivos em texto (.txt) é o "fprinf", entretanto é necessário seguir alguns passos antes de poder imprimir algum texto no arquivo.

Vamos a eles, primeiramente precisamos criar o arquivo em nosso diretório, para isso usaremos o comando "fopen" como segue abaixo:

```
fid=fopen('[nome do arquivo].txt', 'w');
```

Criamos uma variável para receber este arquivo, pois precisaremos dela depois na hora de usarmos o fprintf. O 'w' dá ao Matlab permissão para escrever (write), em alguns casos você pode precisar usar outra permissão como 'a' (append) ou 'r' (read).

Agora que nosso arquivo foi criado ele já está pronto para uso, só precisamos de algo para imprimir, mas antes disso vamos a uma pequena explicação do fprintf. Esse comando, imprime na Command Window se for colocado na forma:

```
>>> fprintf('teste');
teste
```

Se for deixado desta forma, ele não imprimirá no arquivo, para fazer isso, precisamos de uma FileID ou seja, uma variável que possua o endereço de um arquivo com permissões de escrita, e adivinha, nós temos.
Utilizaremos a variável fid criada anteriormente para imprimir a mensagem abaixo no arquivo criado.

Modelo:

```
fprintf(fid, '4 Bytes Engenharia Júnior');
```

Se você seguiu os passos corretamente, teremos o resultado esperado no arquivo salvo no diretório atual de trabalho. Mas acalme-se aidna não acabou, como você pode ver criamos e imprimimos em um arquivo, o que falta agora? Simples falta fechá-lo (desalocar o uso dele pelo MAtlab). Para fechar um arquivo, usamos o comando fclose na forma abaixo:

```
fclose('fid') % no caso foi usado fid, pois é a variável que detem os direitos sobre o nosso arquivo.
```

Aviso: Caso você se esqueça de usar fclose no arquivo, e tente apagá-lo receberá uma mensagem de erro dizendo que o arquivo ainda está em uso pelo Matlab. Para corrigir isso é necessário fechar o Matlab, deletar o arquivo e só então reabri-lo.

9. CRIAÇÃO DE INTERFACES (GUIDE)

Chegamos em uma das partes mais problemáticas e divertidas deste treinamento a criação de interfaces gráficas no Matlab (sim, ele também faz isso).

Apesar de ser algo que grande parte dos usuários de Matlab adorem utilizar, poucos se interessam por fazê-las.

Existem duas formas de se desenvolver uma interface no Matlab, uma delas trabalhosa mas com um código mais maleável e a outra é mais simples e rápida (de se construir) com a customização depen-dendo de seus conhecimentos em Matlab.

Para este treinamento useramos a segunda opção, o GUIDE que facilita bastante o processo de criação de uma interface no Matlab.

Chegou a hora de desenvolvermos uma interface para o método que foi citado no tópico sobre funções.

Para inicializar o GUIDE, digite o comando abaixo:

```
>>> guide
```

9. CRIAÇÃO DE INTERFACES (GUIDE)

9.1 Callbacks

As Callbacks representam o cerne de um GUI no Matlab, elas determinam como um botão ou menu deve agir a determinada ação, como ele responde a isso, etc. Analisando o código gerado pelo GUIDE, é possível notar que para cada novo elemento um nova call-back é gerada, isso é porque o Matlab trata esses objetos de forma separada, ou seja, cada função de callback é indepentende de outra.

9.2 Interface Simples

Criação de interface básica para o método mencionado no tópico 8.