MATLAB para leigos e desinteressados: uma introdução simples com exemplos banais

Rodrigo S. Portugal
portugal@ige.unicamp.br
Departmento de Geologia e Recursos Naturais – Instituto de Geociências
UNICAMP

Você pode estar estranhando o título, mas a verdade é que este texto se destina às pessoas que são obrigadas a aprender MATLAB ou pela imposição de um orientador ou para alguma disciplina, ou por qualquer outro motivo. Eu acho que num primeiro momento estas pessoas não estão tão interessadas assim em aprender, mas, sim, apenas tentando cumprir uma tarefa. Outro público são aqueles que se interessam timidamente, mas sempre têm receio em se envolver com máquinas, algoritmos e programas.

Percebi que para todo este público faltava algo ainda mais introdutório que os tutoriais que circulam pela internet. Você ainda pode pensar que este texto é uma solução muito paternalista para estes desamparados. Se este for o seu caso, então não leia, você não está entre este público. Por outro lado, se você sente que há um pingo de verdade no que disse, está desesperado ou não sabe por onde começar, vá em frente, leia o texto. Seja bem-vindo ao mundo do MATLAB!

1 Introdução

O MATLAB (do inglês, "Matrix Laboratory") é um programa interativo que se destina a cálculos numéricos e gráficos científicos. Seu ponto forte está na manipulação e cálculos matriciais, como por exemplo, resolução de sistema lineares, cálculo de autovalores e autovetores, fatoração de matrizes, entre outros. Além disso, muitas funções especializadas já estão internamente implementadas, de modo que em muitos casos não há a necessidade de se construir muitas coisas.

Outros dois pontos fortes do MATLAB são a criação e manipulação de gráficos científicos e a possibilidade de extensão por meios de pacotes comerciais ou escritos pelo próprio usuário. Em geral o usuário e a funcionalidade do MATLAB crescem juntos, pois com o tempo, ele começa a escrever suas próprias funções, agregando capacidades específicas. Isto faz com que, aos poucos, o MATLAB comece a mimetizar, ou até mesmo suplantar, softwares científicos específicos de diversas áreas.

Uma característica interessante é que o MATLAB é muito mais fácil de aprender do que as linguagens científicas convencionais, tais como C e Fortran. Entretanto, quanto maiores e mais complexas as rotinas e funções da biblioteca, pior fica a sua performance quando comparada à sua equivalente em C e/ou Fortran.

Este documento se destina ao usuário iniciante, para "quebrar o gelo" entre ele e o MATLAB, na promessa de uma parceria duradoura. Em nenhum momento este texto servirá de manual, uma vez que outros bem mais completos estão disponíveis nas bibliotecas, livrarias e na internet também. Finalmente cabe dizer que os exemplos aqui apresentados foram testados na versão 6.5 do programa, sendo assim, apesar de uma quase certeza, não há garantias de que irão funcionar em versões diferentes.

2 Comandos básicos e gráficos simples

Nesta seção são apresentados os comandos mais básicos para inserção e manipulação de dados, bem como comandos para se criar os mais simples gráficos. Além disso são abordados as representações de números, vetores e matrizes à luz do paradigma do MATLAB. De fato, esta seção é crítica, pois é nela que vamos realmente quebrar o gelo.

2.1 Modo interativo

Logo que o MATLAB inicia, há uma tela de para entrada de comandos. Esta tela está no modo interativo (parecido com MS-DOS, lembra?), onde o computador fica

esperando um instrução no lugar marcado pelos símbolos >> (em inglês chamado de prompt). Assim que o usuário digita o seu comando, ele deve finalizá-lo apertando a tecla <enter>, como se tivesse mandando o MATLAB efetuar a instrução da linha.

Eis alguns exemplos para o leitor praticar, não se esquecendo de digitar a tecla <enter> ao final da linha:

```
>> x = 1

x =

     1

>> y = 2

y =

     2

>> z = x + y

z =

     3

>> w = x * y

w =

     2
```

A mensagem de saída logo após a entrada da instrução é às vezes útil, mas pode ficar aborrecendo em muitas outras ocasiões. Para suprimí-la, basta inserir um ponto-e-vírgula (;) ao final da instrução, do mesmo modo que nos exemplos abaixo

```
>> x = 1;
>> y = 2;
>> z = x + y;
>> w = x * y;
```

No modo interativo, é extremamente recomendável que o usuário iniciante guarde os comandos (e os resultados) que foram inseridos em uma sessão. Aos invés de escrever tudo em uma folha de papel (o que pode ser muito útil também), o usuário pode requisitar ao MATLAB que guarde tudo em um arquivo. Basta entrar com o comando

```
diary <nome_do_arquivo.txt>
```

que irá guardar todos os comandos subseqüentes até o final da sessão.

Finalmente, para sair do MATLAB usando o modo interativo, basta digitar `exit`.

2.2 Pedindo ajuda

Certamente o comando mais básico que todo usuário deve conhecer é o `help`. Este comando é o companheiro nas horas mais ingratas, quando tudo dá errado. Muitas vezes achamos que sabemos usar um comando (ou programa) até o momento em que as coisas dão errado *mesmo*. Muito depois de quebrar a cabeça, usando o famoso método da tentativa e erro, lembramos de usar o `help` e aprendemos a maneira realmente correta de usar o tal comando problemático. Sendo assim para aprender como se usa um comando em MATLAB, basta digitar na linha de comando

```
>> help <nome_do_comando>
```

ou ainda consultar o manual. Por exemplo, como já vimos alguns comandos, você pode tentar fazer

```
>> help help
>> help diary
>> help exit
```

e ver o que acontece.

Outra maneira de pedir ajuda, que, por incrível que pareça, funciona melhor, é pedir dicas ao colega sentado ao lado. Depois de um tempo, prepare-se pois vai acabar tirando a dúvida de alguém `:)` .

2.3 Matrizes, matrizes, matrizes

O paradigma do MATLAB é que **tudo** são matrizes. Um número é uma matriz de um por um, um vetor de n números é uma matriz de n por um e strings (cadeias de caracteres) são matrizes caracteres. Existem outros exemplos mais complexos que não valem a pena serem citados neste momento, porém, para quem ainda não fixou a idéia, exite um velho ditado que diz

"Tudo no MATLAB *são matrizes."*

Para verificar este fato, observe o que o comando `whos` dispõe na tela, logo após o exemplo da Seção 2.1:

```
>> x = 1;
>> y = 2;
>> z = x + y;
>> w = x * y;
>> whos
  Name      Size                    Bytes  Class

  w         1x1                         8  double array
  x         1x1                         8  double array
  y         1x1                         8  double array
  z         1x1                         8  double array

Grand total is 4 elements using 32 bytes
```

Sem entrar em detalhes, observe que a saída do comando `whos` é exibir um sumário das váriaves que estão guardadas na memória, descrevendo quais são as suas dimensões, quantos bytes gastam e de qual tipo pertencem. Do exemplo acima podemos concluir que todas as variáveis são escalares (ou matrizes 1x1, no entedimento do MATLAB), gastam 8 bytes cada uma e são reais (double). Para aprender mais sobre o comando `whos` digite

```
>> help whos
```

2.4 Exemplos

Para fixar os conceitos apresentados até aqui, acho que vale a pena o leitor testar os seguintes exemplos

1. Criando uma matriz 4×3 (4 linhas e 3 colunas). Cada linha é separada por um ponto-e-vírgula e cada entrada é separada por uma vírgula

   ```
   >> A = [1, 2, 3; 1, 4, 7; 2, 6, 1; 4, 5, 0]
   ```

2. Criando uma matriz 2×4 com todas as entradas iguais a um

   ```
   >> O = ones(2,4)
   ```

3. Criando uma matriz 4×4 nula

   ```
   >> Z = zeros(3,4)
   ```

4. Criando uma matriz identidade 3×3

```
>> E = eye(3)
```

Uma vez que os exemplos acima foram testados, os comandos abaixos podem ser inseridos (observe que o texto após o caracter % não precisa ser digitado)

```
>> C = 2*A                                          % multiplicacao por escalar
>> D = [3, 4, 5; 1, 6, 3; 2, 2, 9; -1, 0, 17]      % inserindo uma nova matriz
>> F = A + D                                        % adição de matrizes
>> G = A - D                                        % subtração de matrizes
>> T = G'                                           % transposta de matriz
>> M = F * T                                        % multiplicação de matrizes
>> P = inv(M)                                       % inversa de matriz
>> a = det(P)                                       % determinante de matriz
```

2.5 Quero plotar uma função, o que o faço?

Existem duas maneiras de graficar uma função. A mais simples é usar o comando `fplot`. Basicamente, você deve fornecer como primeiro argumento a função que pretende usar entre apóstrofes e como segundo o intervalo sobre o qual a função será graficada. Para fixar a idéia, seguem alguns exemplos:

```
>> fplot('sin(x)',[ -pi, pi])                       %
>> fplot('x^2+3', [ -1, 2])                         %
>> fplot('sin(x)',[ -0, pi])                        %
```

Existem outras opções que podem ser verificadas com `help fplot`.

A outra maneira, mais difícil, porém mais flexível, usa o paradigma de que tudo em MATLAB são matrizes. Neste caso, considera-se somente uma versão discretizada da função. Mas afinal, o que é uma discretização de uma função? Para responder a essa pergunta temos que dar uma paradinha e explicar esse conceito.

2.5.1 Discretização de funções

Basicamente uma discretização de uma função qualquer $f(x)$ é quando uma seqüência de números é gerados a partir de outra seqüência de números usando a própria função como a regra para a geração. A seqüência à qual será aplicada a função é chamada de domínio discretizado e a outra seqüência de imagem discretizada. Para que tudo corra bem, no entanto, há a condição de que todos os números do domínio discretizado sejam distintos. Se existir repetição, ainda vai funcionar para todos os efeitos, mas não é mais uma função, é outra coisa que está fora do escopo desta introdução.

Por exemplo se a função é $f(x) = x^2$, podemos considerar a seqüência de números

$$x_n = [-2, -1, 0, 1, 2, 3, 4]$$

e quando aplicamos a função f a cada um dos números de x_n geramos outra seqüência de números a saber:

$$y_n = [4, 1, 0, 1, 4, 9, 16].$$

Se nós juntarmos as duas seqüências em pares, então obtemos uma discretização da função

$$[(-2, 4), (-1, 1), (0, 0), (1, 1), (2, 4), (3, 9), (4, 16)]$$

Resumindo, discretização é quando uma função é aproximadamente descrita por vários pares ordenados (x_n, y_n), onde os x_n são todos distintos e, de preferência, mas não necessariamente, ordenados de forma crescente.

Pode-se notar, portanto, que um vetor (geralmente muito grande) pode representar uma discretização de uma função, quando o domínio, isto é, a seqüência de números à qual será aplicada a função, é o conjunto de números naturais.

De outra forma, uma função discretizada deve ser representada por dois vetores com a mesma dimensão (geralmente grande), onde um representa o domínio discretizado e o outro a imagem discretizada.

Uma vez que o leitor já entendeu o que é uma discretização, falta ainda a explicação de uma capacidade específica que o MATLAB possui em realizar operações com vetores, as chamadas operações elemento-a-elemento ("elementwise"), vistas a seguir.

2.5.2 Operações elemento-a-elemento (eae)

Para multiplicar duas matrizes, é necessário que as as dimensões sejam compatíveis, como no exemplo

$$A_{(3\times5)}B_{(5\times1)} = C_{(3\times1)}.$$

O leitor pode conferir que o número de colunas da matriz à esquerda (A) é igual ao número de linhas da matriz à direita (B), satisfazendo a condição para a multiplicação entre matrizes.

Por outro lado, não é possivel realizar a multiplicação de duas matrizes com mesma dimensão $m \times n$, com $m \neq n$, como por exemplo ($m = 5$ e $n = 3$)

$$A_{(5\times3)}B_{(5\times3)} = ?? \quad \longleftarrow \quad \text{Impossível !}$$

Outra coisa impossível de se fazer é a divisão de uma matriz por outra, não importando quais sejam as dimensões

$$\frac{A_{(m\times n)}}{B_{(p\times q)}} = ?? \quad \longleftarrow \quad \text{Impossível !}$$

Esta operação causa graves problemas de saúde em professores quando alunos insistem em realizá-la.

Contudo, para matrizes que possuam as mesmas dimensões, digamos $m \times n$, é possível definir o que se chama de operação *elemento-a-elemento* (eae), incluindo a *multiplicação-eae* e *divisão-eae*. Para que isso funcione, no MATLAB, basta você fazer o seguinte: use o sinal de ponto do lado esquerdo da operação desejada (sem espaço entre os símbolos).

Multiplicação-eae

```
>> A = B .* C
```

Divisão-eae

```
>> A = B ./ C
```

Potenciação-eae

```
>> A = B .^ C
```

Para fixar a idéia das operações eae, vale a pena testar os comandos abaixos

```
>> A = [3, 4, 5; 1, 6, 3; 2, 2, 2; -1, 1, 2]    % inserindo uma nova matriz
>> D = [2, 0, 4; 0, 1, 6; 4, 1, 0; -1, 0, 7]    % inserindo uma nova matriz
>> F = A .* D                                    % multiplicação eae
>> G = D ./ A                                    % divisão eae
>> T = G .^ A                                    % potenciação eae
```

2.5.3 Criação de funções mais complexas

Dado essa grande quantidade de ferramentas eae e os conceitos de discretização, é possível unir esses dois mundos para criar funções (discretizadas) bem complexas no MATLAB.

Por exemplo, suponha que desejamos criar uma função polinomial do quinto grau

$$f(x) = -0.1x^5 + 40x^3 - 2000x - 5$$

para examinarmos o seu gráfico. Como devemos proceder?

Em primeiro lugar devemos definir o domínio discretizado como o comando `linspace`

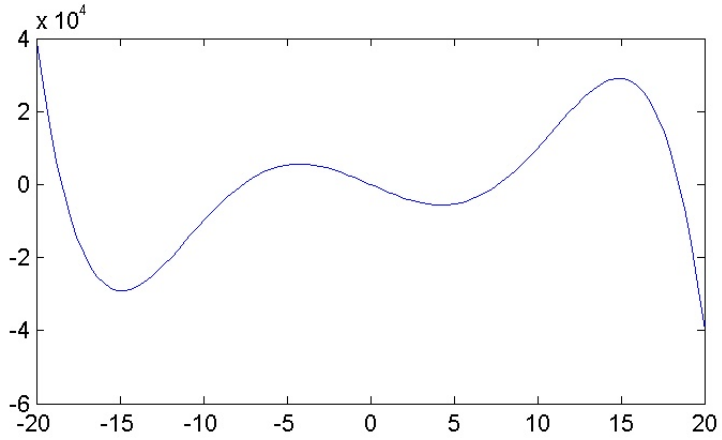

Figura 1: Gráfico da função polinomial

```
>> x = linspace(-20, 20, 100);
```

que quer dizer o seguinte: o vetor `x` terá 100 elementos, onde o primeiro é -20 e o último é 20, e os intermediários estão igualmente espaçados.

Depois deve-se criar o vetor que vai representar a imagem da função discretizada

```
>> y = -0.1*(x.^5) +  40*(x.^3) - 2000*(x) - 5;
```

Para, finalmente, plotar o gráfico da função discretizada, utilizamos

```
>> plot(x,y)
```

cujo resultado pode ser conferido na Figura 1.

2.6 Operações com escalares, uma pequena exceção

Rigorosamente falando, é impossível somar uma constante (que é uma matriz um por um, lembra?) a uma matriz qualquer de dimensões maiores do que um, afinal de contas as dimensões "não batem". No entanto o MATLAB permite este "abuso" entendendo que se deseja somar a constante a todas as entradas da matriz. Isto funciona para todas as outras operações, tais como subtração, multiplicação e divisão, entre outras. Entretanto, o leitor deve tomar cuidado, pois isso não funciona com a potenciação.

Para verificar esta propriedade, teste os seguintes exemplos

```
>> A = [1, 2, 1; 0, 1, 2; 1; 3; 0]
>> B = 5 + A
>> C = 2 * A
>> D = 0.5 - A
```

3 Exemplos avançados

3.1 Curvas planas

Comandos abordados: `help(), linspace(), plot(), sin(), cos()`

A forma paramétrica de uma curva plana Γ pode ser descrita através do seguinte par de funções

$$\Gamma : \begin{cases} x = f(t) \\ y = g(t) \end{cases} \tag{1}$$

onde t é um parâmetro real que assume valores em um intervalo $[a, b]$ e $f(t)$ e $g(t)$ são funções quaisquer. Quando $t = a$, o ponto $(x_a, y_a) = (f(a), g(a))$ é o início da curva e quando $t = b$, o ponto $(x_b, y_b) = (f(b), g(b))$ é o final da curva.

Para fixar a idéia, examine os seguintes exemplos:

$$1) \quad \Gamma_1 : \begin{cases} x = t + \sin(2t) \\ y = t + 2\cos(5t) \end{cases}$$

$$2) \quad \Gamma_2 : \begin{cases} x = \cos(t) - \cos(80t)\sin(t) \\ y = 2\sin(t) - \sin(80t); \end{cases}$$

Finalmente, para exibir as curvas acima, são necessários os seguintes comandos abaixo (observe que o comentário após o `%` não necessita ser digitado)

Curva Γ_1

```
>> t = linspace(0,10*pi,1001);   % gera o domínio discretizado
>> x = t + 3*sin(2*t);           % coordenadas x da curva discretizada
>> y = t + 5*cos(5*t);           % coordenadas y da curva discretizada
>> figure(1)                     % aciona uma janela gráfica
>> plot(x,y)                     % desenha a curva
```

Curva Γ_2

```
>> t = linspace(0,5*pi,5001);
>> x = cos(t) - cos(80*t).*sin(t);
>> y = 2*sin(t) - sin(80*t);
>> figure(2)
>> plot(x,y)
```

O leitor mais corajoso certamente vai mexer nos parâmetros das curvas acima para ver o que acontece. Os gráficos resultantes destes comandos podem ser conferidos na Figura 3.1.

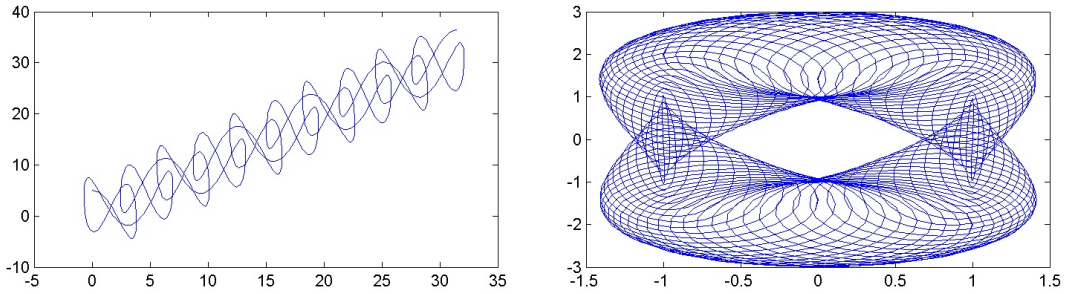

Figura 2: Curvas planas Γ_1 (equerda) e Γ_2 (direita).

3.2 Geração de números aleatórios

O comando básico para geração de números aleatórios é o `rand`. Basicamente ele gera uma matriz com números com distribuição uniforme $[0, 1]$ (faça `>> help rand` para saber mais). Seguem abaixo alguns exemplos de sua utilização

```
>> x = rand(100,1);           % vetor coluna com 100 numeros aleatorios
>> y = rand(1,100);           % vetor linha com 100 numeros aleatorios
>> A = rand(100);             % matriz de 100x100 numeros aleatorios
>> B = rand(50,30);           % matriz de 50x30 numeros aleatorios
>> z = 2*rand(100,1) - 1;     % vetor 100x1 com distr. unif. [-1,1]
```

Às vezes pode ser útil que o vetor aleatório gerado seja sempre o mesmo. Para conseguir isso você deve fixar a semente do gerador de números aleatórios. Primeiramente guarde a semente atual com o comando

```
>> seed =  rand('state');     % guarda a semente do gerador
```

onde `seed` é uma variável qualquer, que pode ter outro nome da sua escolha. Em seguida, fixe a semente no gerador com o comando

```
>> rand('state', seed);       % fixa a semente no gerador
```

para se assegurar que ele vai repetir a mesma seqüência de números aleatórios.

Observe o exemplo abaixo, em que a semente não é fixada

```
>> x = rand(100,1);           % gera um vetor aleatório
>> y = rand(100,1);           % gera outro vetor aleatório
>> isequal(x,y)               % verifica se x e y são iguais

ans =

     0
```

Aqui a função `isequal` retorna `0` indicando que os vetores `x` e `y` são diferentes. Por outro, lado verifique, através do exemplo abaixo, quando a semente é fixada

```
>> seed =  rand('state');      % guarda a semente do gerador
>> x = rand(100,1);            % gera um vetor aleatório
>> rand('state',seed);         % fixa a semente previamente guardada
>> y = rand(100,1);            % gera outro vetor aleatório
>> isequal(x,y)

ans =

     1
```

Aqui a função `isequal` retorna `1` indicando que os vetores `x` e `y` são iguais. Como já foi dito, isso que significa que sempre que o comando `rand('state',seed);` for usado, a geração dos números aletórios vai se repetir.

3.3 Vetorização

3.4 Estatísticas

A Tabela de funções

AS funções em MATLAB podem ser divididas em dois tipos:

1. Funções que preservam a dimensão do argumento;
2. Funções que trazem uma saída com dimesões diferentes (em geral menores) em relaão ao seu argumento;

Função	Argumento	Resultado
	escalar	escalar
`fun1`	vetor	vetor
	matriz $(m \times n)$	matriz $(m \times n)$
	escalar	escalar
`fun2`	vetor	escalar
	matriz $(m \times n)$	vetor linha $(1 \times n)$

As funções do tipo **fun1** podem ser

Função	Descrição
`cos`	Cosseno de um ângulo (em radianos)
`sin`	Seno de um ângulo (em radianos)
`tan`	Tangente de um ângulo (em radianos)
`exp`	Exponecial na base e
`log`	Logaritmo natural
`sqrt`	Raiz quadrada

e as funções do tipo **fun2** podem ser