Minicurso de Matlab (guia de aula)

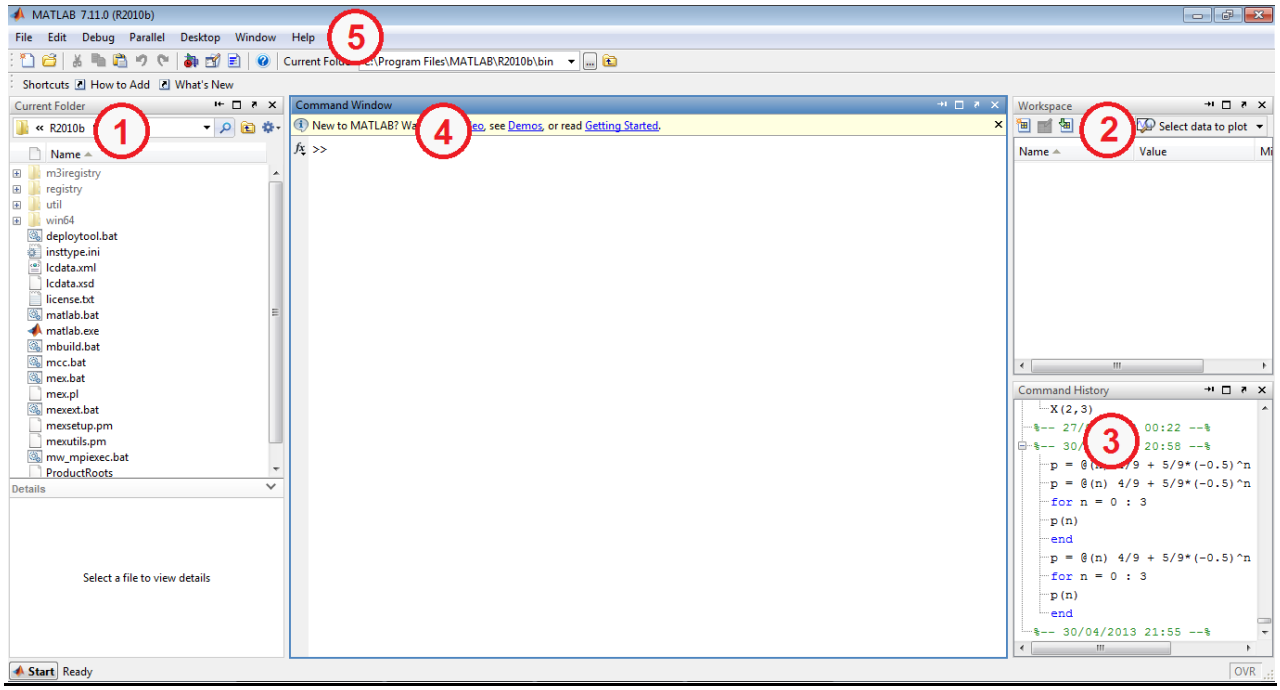
Por: Fabrício Caluza Machado

Aula 1 – Introdução

Para que serve o Matlab? Introdução ao ambiente, às janelas, à inserção e à manipulação de variáveis.

O Matlab é uma poderosa ferramenta computacional, que permite a visualização de gráficos de funções e curvas/superfícies parametrizadas, cálculo com matrizes, implementação de algoritmos e inúmeras outras aplicações através de toolboxes especializadas. Pretendo, nesse minicurso, mostrar os elementos básicos de sua linguagem e ambiente, de modo que o aluno seja capaz de pesquisar por conta própria as funções necessárias para alguma aplicação específica.

Área de Trabalho



Contém as janelas *Current Folder (1)*, com o diretório de arquivos atual (será explorada na próxima aula); *Workspace (2)*, onde são exibidas as variáveis usadas durante a sessão (veremos a seguir); *Command History (3)*, com o histórico de comandos usados nas sessões anteriores; *Command Window (4)*, onde executamos os comandos e vemos seus resultados. Essas janelas podem ser ajustadas com o mouse ou através do menu superior (5) *Desktop*.

Janela de Comando

O comando mais simples que podemos escrever é declarar uma variável, por exemplo:

```
>> a = 5
```

Vemos que o programa armazena a variável "a" com o valor 5 (veja *Workspace*). Podemos executar operações simples, como:

```
>> 2*3

>> a-1

>> b = 2*a
```

Observe que após a primeira operação, o sistema exibe o resultado em uma variável temporária `ans`, que é sobrescrita na operação seguinte (semelhante ao que ocorre em calculadoras científicas).

Podemos usar funções do Matlab (existem muitas!), por exemplo:

```
>> cos(30)

>> cos(pi)
```

Observe que as funções recebem entradas em radianos e que `pi` já é uma constante predefinida.

Ajuda

Esse é um bom momento para se mostrar como pesquisar funções ou aprender a usar alguma função específica.

Existe o guia "MatlabNumInstante" disponível em: http://paginas.fe.up.pt/~hmiranda/tele2/MatlabNumInstante1-3.pdf com algumas descrições úteis de funções, por exemplo a tabela da página 8 com funções matemáticas.

Dentro do Matlab, vá ao menu superior `Help > Function Browser`. Todas as funções do Matlab organizadas por assunto. Olhe a categoria `MATLAB > Mathematics > Elementary Math > Trigonometric` e descubra que a função `cosd` calcula o cosseno de um ângulo em graus.

Caso queira uma descrição do uso de uma função com nome conhecido, use os comandos `help` (para uma descrição na própria janela) ou `doc` (para acessar o manual, mais completo), por exemplo:

```
>> help mod

>> doc mod
```

Existe ainda o comando `lookfor`, que pesquisa os textos da ajuda de todas as funções.

```
>> lookfor clear

>> lookfor 'clear command window'
```

Observe que usamos aspas simples para pesquisar um conjunto de palavras, para mostrar que toda a expressão é um único argumento de entrada da função `lookfor`.

Por fim, vale a pena destacar que o Product Help do Matlab (F1) possui uma série de manuais com as funções padrão do MATLAB e exemplos (veja `MATLAB>Demos>3-D Visualization>Klein`

`Bottle`). Existem ainda diversas Toolboxes com conjuntos de funções para aplicações específicas do Matlab.

Manipulação de matrizes I

O Matlab enxerga todas as variáveis como matrizes, escalares são matrizes 1x1. Ao contrário do que ocorre em outras linguagens como C, no Matlab não é necessário alocar espaço ou declarar previamente as dimensões das matrizes, o ambiente trata essas coisas internamente.

Para criar uma matriz com 2 linhas e 3 colunas, por exemplo:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Usamos o comando:

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

O colchete denota a declaração de uma matriz, as vírgulas são dispensáveis e podem ser substituídas por apenas espaços, mas o ponto e vírgula não, ele denota uma nova linha.

Podemos acessar (e alterar) uma entrada específica de uma matriz do seguinte modo:

```
>> A(2,2)
ans =
   5

>> A(2,2)= 0
A =
   1   2   3
   4   0   6
```

No primeiro caso, é exibido o valor 5 e no segundo caso, alteramos este valor para 0.

Podemos acessar conjuntos de entradas na matriz (submatrizes) de forma semelhante:

```
>> A([1,2],[1,2])
ans =
   1   2
   4   0
```

E veremos o bloco com as duas primeiras linhas e colunas de `A`.

Caso queiramos acessar uma linha inteira da matriz, podemos usar o operador ":":

```
>> v = A(1,:)
v =
   1   2   3
```

E a primeira linha da matriz `A` será atribuída ao vetor `v`.

O operador ":"

O operador ":" também é usado para a criação de vetores de sequências aritméticas, a notação é a seguinte:

```
Número_inicial : incremento : Número_final
```

Por exemplo, o vetor v = (1 3 5 7 9) pode ser criado com o comando:

```
>> v=1:2:9
v =
   1    3    5    7    9
```

O incremento pode ser negativo, nesse caso o `Número_final` deve ser necessariamente menor que o `Número_inicial` e caso o incremento seja 1, ele pode ser omitido:

```
>> v2 = 1:15
v2 =
 Columns 1 through 10
   1    2    3    4    5    6    7    8    9   10
 Columns 11 through 15
  11   12   13   14   15
```

Manipulação de matrizes II

Podemos usar algumas funções específicas para a criação de matrizes. No menu de funções do Matlab (Function Browser), vá em `MATLAB > Mathematics > Elementary Matrices and Arrays` e veja algumas dessas funções, por exemplo:

```
>> B = ones(5)
B =
   1    1    1    1    1
   1    1    1    1    1
   1    1    1    1    1
   1    1    1    1    1
   1    1    1    1    1
```

Criará uma matriz 5x5 com o número 1 em todas as entradas, e:

```
>> C = zeros(5)
C =
   0    0    0    0    0
   0    0    0    0    0
   0    0    0    0    0
   0    0    0    0    0
   0    0    0    0    0
```

Criará uma matriz 5x5 com zeros.

Podemos ainda, concatenar matrizes. Fazendo:

```
>> D = [B, C ; C, B]
```

```
D =
   1   1   1   1   1   0   0   0   0   0
   1   1   1   1   1   0   0   0   0   0
   1   1   1   1   1   0   0   0   0   0
   1   1   1   1   1   0   0   0   0   0
   1   1   1   1   1   0   0   0   0   0
   0   0   0   0   0   1   1   1   1   1
   0   0   0   0   0   1   1   1   1   1
   0   0   0   0   0   1   1   1   1   1
   0   0   0   0   0   1   1   1   1   1
   0   0   0   0   0   1   1   1   1   1
```

Obtemos uma matriz 10x10 com as matrizes B e C constituindo seus blocos. Note que são passados apenas os valores numéricos de B e C para D, sem criar uma dependência, ou seja, caso B seja alterada, D não será.

Podemos combinar essas operações com o operador ":". O comando:

```
>> E = D( : , 1:2:10)
E =
   1   1   1   0   0
   1   1   1   0   0
   1   1   1   0   0
   1   1   1   0   0
   1   1   1   0   0
   0   0   0   1   1
   0   0   0   1   1
   0   0   0   1   1
   0   0   0   1   1
   0   0   0   1   1
```

Irá criar uma matriz 10x5 com as colunas ímpares de D.

Podemos remover uma linha ou coluna de uma matriz atribuindo a esta a "matriz vazia" []:

```
>> E(:,1)=[]
E =
   1   1   0   0
   1   1   0   0
   1   1   0   0
   1   1   0   0
   1   1   0   0
   0   0   1   1
   0   0   1   1
   0   0   1   1
   0   0   1   1
   0   0   1   1
```

Irá remover a primeira coluna de E.

Aula 2 – Arquivos .m

Uso de scripts e funções. Function Handle.

Editor: criação de scripts

Antes de começarmos a manipular arquivos, precisamos ajustar a pasta de trabalho do Matlab. Isso pode ser feito no menu superior, Current Folder (escolha Área de Trabalho, por exemplo). Observe também a janela Current Folder, que se encontra normalmente à esquerda da janela de comando (ou então acessível através de `Window > Current Folder`).

Abra agora o editor do Matlab (ícone no canto superior esquerdo do programa, ou `File > New > Script`). Temos um editor de texto que nos permite criar sequências de comandos para serem executadas em conjunto na janela de comando (scripts).

Como primeiro exemplo, podemos criar a sequência de comandos:

```
A = ones(6)
B = zeros(6)
B(1:2:6,:) = [1:6;7:12;13:18]
```

Antes de executá-los, salvamos o script no arquivo teste.m (`File > Save`) .

O script pode ser executado pela janela de comando, basta digitar seu nome, como se fosse um comando:

```
>> teste
```

Observe que os comandos foram executados e as variáveis criadas, como se os comandos tivessem sido executados no terminal. Outra forma de se executar o script é clicando na seta verde do menu superior (Run, F5) na própria janela do editor.

Podemos encerrar os comandos com ; para omitir a exibição dos resultados na janela de comando, observe que os comandos continuam sendo executados da mesma forma.

Comandos if /for

Esses comandos podem ser usados tal como programas em C ou outras linguagens. Apenas a sintaxe é um pouco diferente.

O funcionamento do comando `for` pode ser esquematizado como:

```
for índice = início : incremento : fim
    comandos
end
```

Sendo que o bloco `comandos` é executado uma vez para cada valor de `índice`.

Vejamos um script como exemplo:

```
v = zeros(1,10);

for i = 1 : 10
    v(i) = i*i;
end
```

Aqui, o vetor v terá o valor de i*i na entrada i. Observe que precisamos encerrar o comando for com um end.

Os comandos if else permitem a execução condicional de comandos:

```
if condição
    comandos
else
    outros comandos
end

Exemplo:

n = 5; % Dimensão da matriz criada.
A = zeros(n);

for i = 1 : n
    for j = 1 : n
        if mod(i + j, 2 ) == 0
            A(i,j) = 1;
        else
            A(i,j) = 2;
        end
    end
end
```

Esse exemplo cria uma matriz quadriculada com entradas 1 e 2.

Outros comandos úteis em scripts: elseif , while, continue, break, return. Leia mais sobre eles na ajuda do Matlab, em: `MATLAB > User's Guide > Programming Fundamentals > Basic Program Components > Program Control Statements`

Funções

As funções são muito semelhantes a scripts, com a diferença que podem receber argumentos de entrada e retornar valores. O seu escopo de variáveis é separado da janela de comandos.

Um arquivo que contenha uma função deve ser declarado como tal no início (veja exemplo) e deve ter o mesmo nome que sua função.

Exemplo:

```
function A = exemplo(n, k)

A = zeros(n);

for j = 1 : n
    for i = 1 : n
        if mod(i+j,k) == 0
            A(i,j) = 1;
        end
```

```
    end
end
```

Salvando a função no arquivo exemplo.m, podemos executá-la na linha de comando. Observe que, ao contrário do script, os nomes das variáveis na função não se relacionam com as do ambiente de trabalho e que a função retorna uma matriz de tamanho nxn. Digite na janela de comando:

>> B = exemplo(5,3)

Outro exemplo:

>> for k = 1 : 10

imagesc(exemplo(30, k))

pause(1)

end

Primeiro, observe que também podemos usar o comando for na linha de comando (tudo se passa igual ao script, apenas seu uso pode se tornar mais confuso). O comando imagesc desenha a matriz gerada pela função exemplo, atribuindo cores às suas entradas.

Vejamos outro exemplo de função:

```
function res = integra(a,b)
%INTEGRA integra a função sin
%   INTEGRA(A,B) calcula uma aproximação para a integral definida de  %
sin(x) de a até b diretamente através da sua soma de Riemann.
%
%   Exemplo: integra(0,pi)

h = (b - a) / 500;
res = 0;

for x = a : h : b
    res = res + sin(x) * h;
end
```

Function Handle

"Function Handle" é um apontador para função. Ele permite que funções sejam passadas como argumentos para outras funções. Para obter o apontador para uma função usamos o @ antes de seu nome. É mais fácil entender com um exemplo:

```
function res = integra(f,a,b)
%INTEGRA integra a função f
%   INTEGRA(A,B) calcula uma aproximação para a integral definida de  %   f(x)
de a até b diretamente através da soma de Riemann.
%
%   Exemplo: integra(@sin,0,pi)

h = (b - a) / 500;
```

```
res = 0;

for x = a : h : b
    res = res + f(x) * h;
end
```

Essa função pode ser chamada através da linha de comando da seguinte forma:

```
>> integra(@sin, 0, pi)
```

Também pode ser usada com funções personalizadas, desde que seus arquivos estejam na pasta atual.

Função Anônima

Após o exemplo anterior, você provavelmente achará úteis as funções anônimas. Elas são usadas quando queremos um apontador para uma função simples, sem ter que criar um arquivo para ela.

Defina uma função na própria linha de comando da seguinte forma:

```
>> f = @(x) 2*x;
```

Após o @, definimos os argumentos da função (pode ser mais de um), em seguida definimos a função simbolicamente. f se torna um apontador para essa função, que pode ser usada na linha de comando ou passada como parâmetro para outra função.

```
>> integra( @(x) x.^2-x, 1, 4)
```

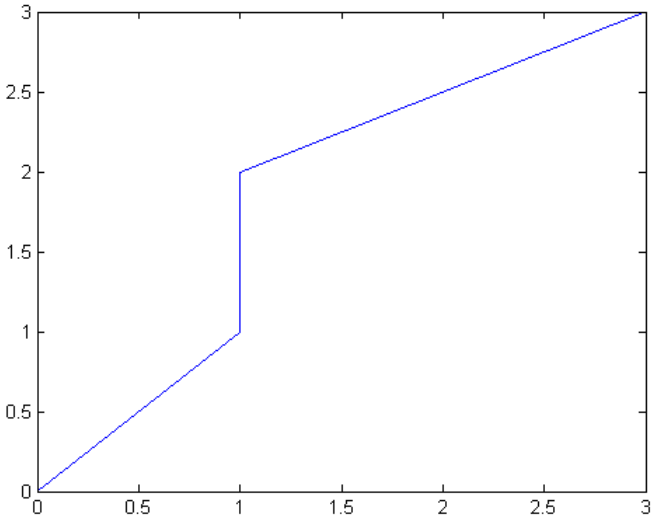
Aula 3 – Gráficos

Plot, parametrizações, janela de gráficos. Gráficos em 3D.

O comando plot

O principal comando para o desenho de gráficos e curvas é o comando `plot`. Começaremos com alguns exemplos simples para entendermos como funciona.

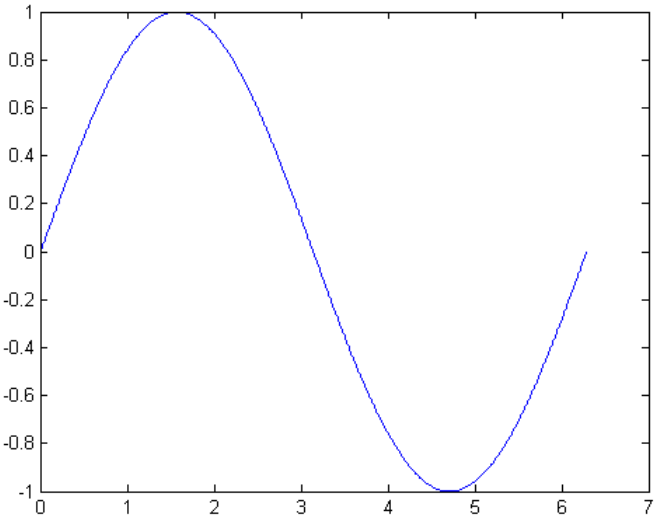
```
>> plot(  [0,1,1,3],  [0,1,2,3] )
```



Vemos que os pontos (0,0), (1,1), (1,2) e (3,3) estão ligados por segmentos de reta. É justamente isso o que o comando plot faz: dados vetores X e Y de dimensões iguais, o comando irá desenhar um gráfico ligando os pontos com coordenadas descritas por esses vetores, respectivamente.

Para visualizarmos o gráfico de uma função, devemos fazer a mesma coisa. Não é possível para o Matlab calcular a imagem de uma função em *todos* os pontos de um intervalo escolhido como domínio. O truque é fazer isso em muitos pontos, de forma que os segmentos não sejam percebidos!

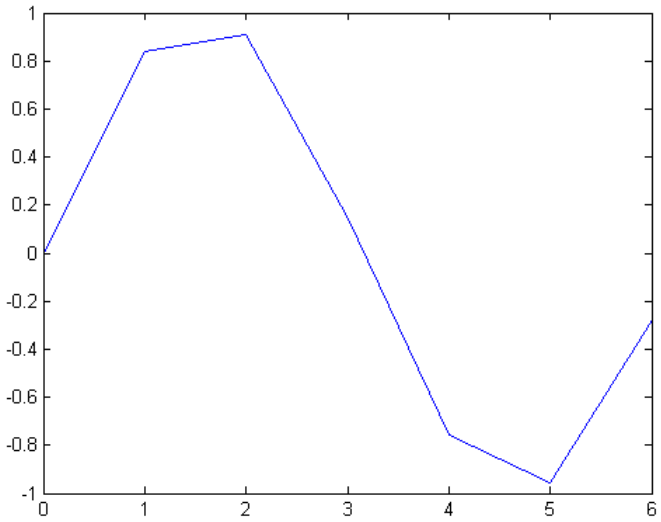
```
>> x = 0 :  0.01 : 2*pi;
>> y = sin(x);
>> plot(x,y)
```



Nesse caso, 629 pontos foram calculados e a imagem realmente parece uma curva!

Podemos nos convencer melhor sobre o que está sendo feito repetindo o processo, dessa vez com menos pontos:

```
>> x = 0 : 1 : 2*pi;
>> y = sin(x);
>> plot(x,y)
```



Um comando útil para definir o domínio de funções é o `linspace`:

```
>> x = linspace(0,10,5)

x =

    0    2.5000    5.0000    7.5000   10.0000
```

Observe que o mesmo resultado pode ser obtido com:

```
>> x = 0 : 2.5 : 10

x =

    0    2.5000    5.0000    7.5000   10.0000
```

O que o comando `linspace` faz é dividir um intervalo em um certo número de partes. A diferença para a notação : é que aqui estamos interessados na quantidade de subdivisões, não no intervalo entre elas.

Digitando no terminal:

```
>> help plot
```

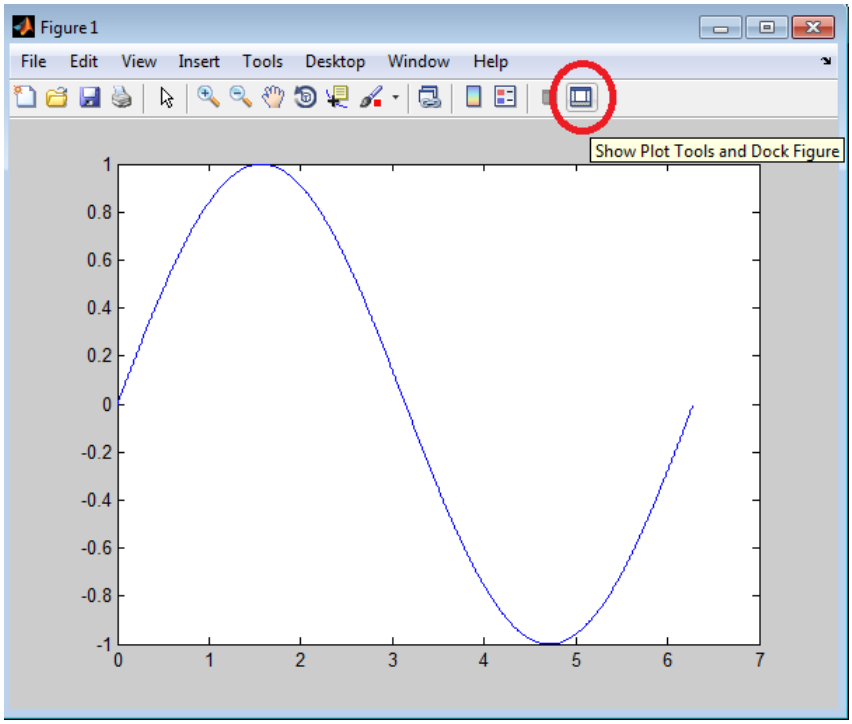
Vemos que existe um terceiro parâmetro (opcional) no comando. Este parâmetro permite alterar características do desenho, como cor da linha, estilo do tracejado e dos pontos usados.

Exemplo:

```
>> plot([0,1,1,3], [0,1,2,3], 'ro ')
```

Janela de figuras

Na janela de figuras, existe um botão que merece atenção, o botão mais a direita do menu superior, que abre a janela de ferramentas de edição de figuras.



Existem muitas coisas que podem ser alteradas no gráfico, entre elas:

1. Título
2. Título dos eixos
3. Escala dos eixos
4. Marcações nos eixos (é possível também trocar o valor por algum texto)
5. Exibição de grade
6. Inclusão de textos, setas, legendas, etc...

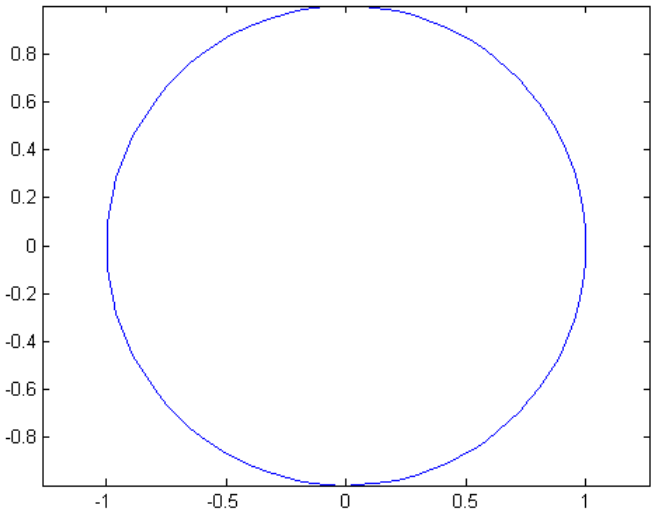
Parametrizações

Voltando ao uso do comando plot, existe um detalhe que merece ser destacado:

O uso mais comum deste comando é o desenho do gráfico de uma função, algo como o exemplo com a função `sin(x)` já apresentado. Mas o mesmo comando pode ser usado para a visualização de curvas parametrizadas. Nada impede que tanto a coordenada x, quanto a y usadas em `plot(x,y)` sejam calculadas como funções de um parâmetro t!

Exemplo:

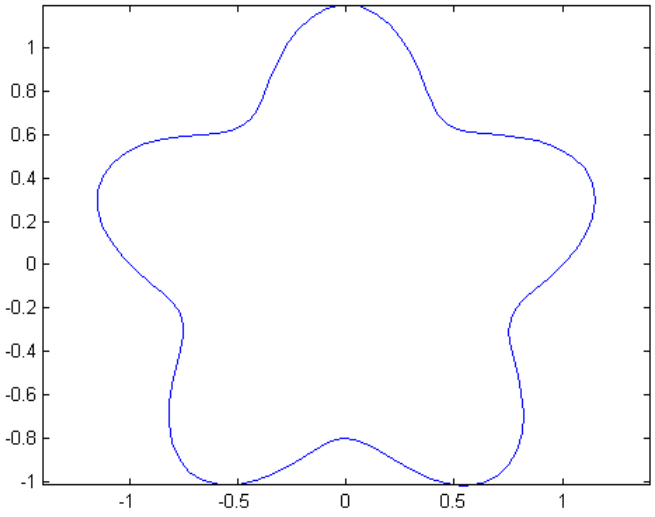
```
>> t = linspace(0,2*pi);
>> x = cos(t);
>> y = sin(t);
>> plot(x,y)
>> axis equal
```



Sendo que o último comando é usado para manter a escala entre os eixos.

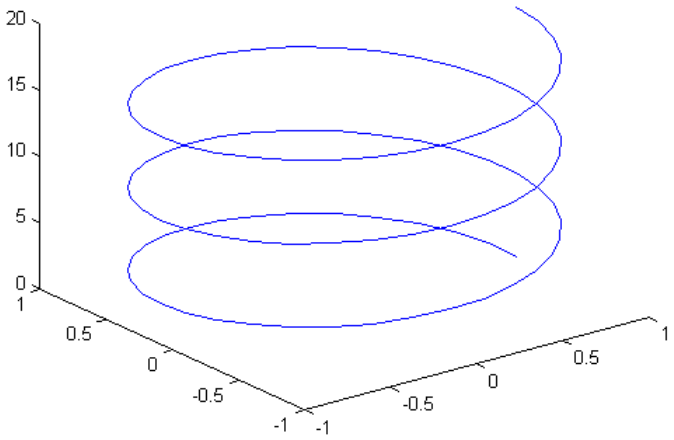
Mais um exemplo (apenas porque parametrizações são legais!)

```
>> t = linspace(0,2*pi);
>> x = cos(t).*(sin(5*t)/5 + 1);
y = sin(t).*(sin(5*t)/5 + 1);
plot(x,y)
axis equal
```



Existe também o comando `plot3`, que faz a mesma coisa que o `plot`, só que em 3 dimensões!

```
>> t = linspace(0,6*pi);
>> x = cos(t);
>> y = sin(t);
>> z = t;
>> plot3(x,y,z)
```



Usando scripts

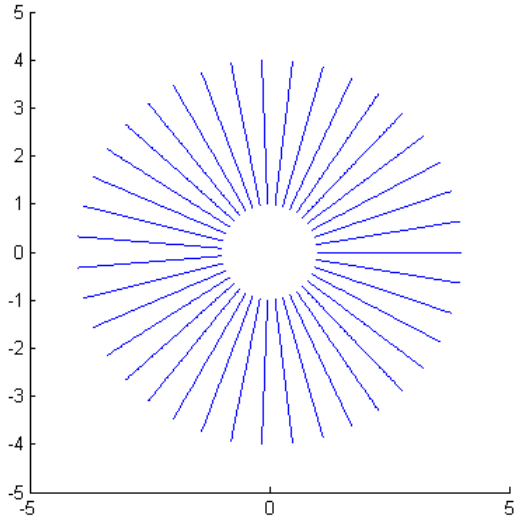
Os scripts vistos na aula anterior são muito úteis quando queremos fazer algum desenho mais complexo. Podemos desenhar usando parâmetros e alterar o desenho facilmente, depois.

Alguns comandos que podem ser úteis durante um script são:

- hold on : permite a sobreposição de desenhos.
- axis equal : mantêm a escala entre os eixos.
- xlim : permite o ajuste da escala do eixo x (ylim e zlim análogos)
- pause: pausa a execução do script por alguns segundos (dentro de um comando for gera efeitos muito legais!)
- `print`: salvar a imagem em um arquivo.

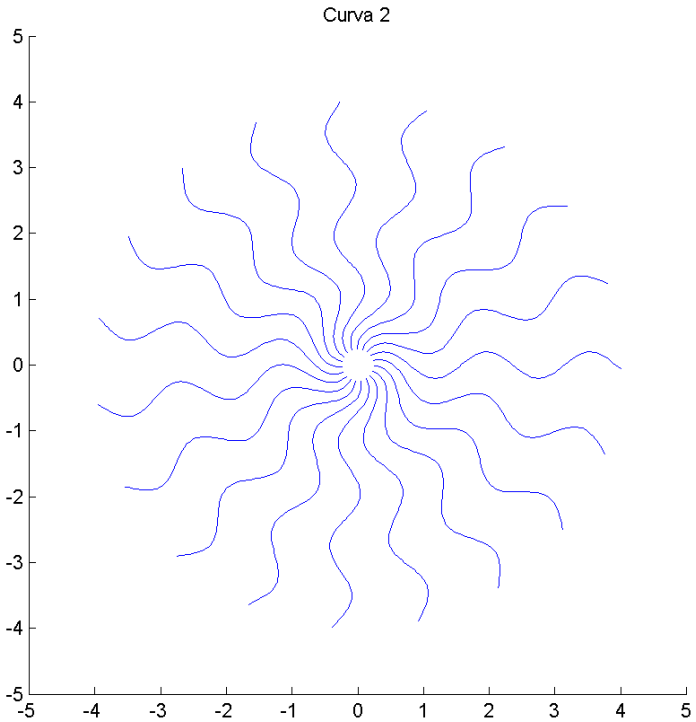
Exemplo:

```
t = linspace(1,4);
axis equal;
hold on;
xlim([-5,5]);
ylim([-5,5]);

for k = linspace(0,2*pi,40)
    x = t*cos(k);
    y = t*sin(k);

    plot(x,y);
    pause(0.1);
end
```



Um exemplo mais legal:

```
t = linspace(0.2, 4); % intervalo de domínio da curva

for s = 0 : 2 : 10 % esse parâmetro controla a "ondulação" da curva
    close; % fecha a janela (útil a partir da segunda iteração)
    axis equal; % ajusta a escala dos eixos
    hold on; % permite a sobreposição das curvas
    xlim([-5,5]); % ajusta as dimensão do eixo x
    ylim([-5,5]); % ajusta as dimensão do eixo y

    for theta = linspace(0,2*pi,20) % coordenada polar
        x = t*cos(theta) - 0.2*sin(s*t)*sin(theta);
        y = t*sin(theta) + 0.2*sin(s*t)*cos(theta);

        plot(x,y);
        pause(0.05); % pausa rapidamente para dar um efeito de animação
    end
    nome=sprintf('Curva %d', s/2); % cria um nome para a imagem
    title(nome); % cria um título
    pause(1); % pause por um período maior
    print( gcf, '-dpng', nome); % exporta a imagem no formato .png
end
```



Aula 4 – Superfícies

Gráficos em 3D: campos vetoriais e superfícies parametrizadas

Campos Vetoriais

Complementando as aplicações das funções vistas na aula anterior, existe mais uma aplicação que, apesar de usar apenas as funções já vistas, é digna de nota: a visualização de campos vetoriais.

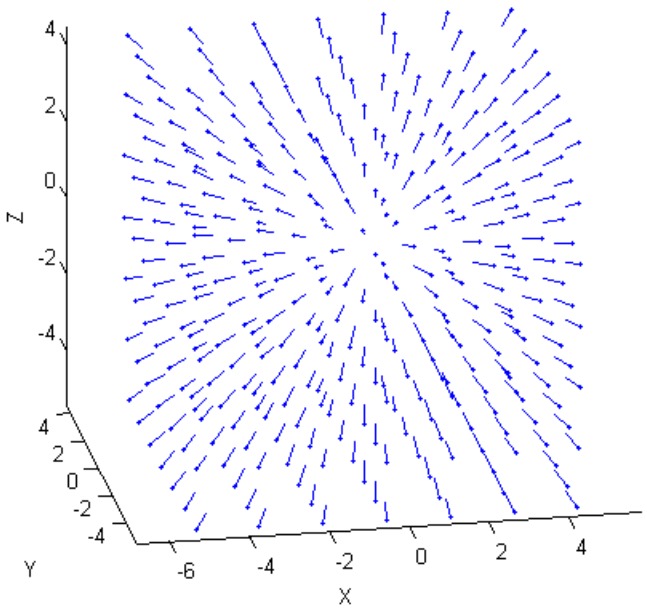
```
function desenha_campo
    %DESENHA_CAMPO Desenha um campo vetorial tridimensional.

    % Feito por: Fabrício Caluza Machado
    % Data: 03/05/2013

    k = 5;
    passo = 1.5;
    hold on;
    axis equal;
    xlabel('X');
    ylabel('Y');
    zlabel('Z');

    for x = -k : passo : k
        for y = -k : passo : k
            for z = -k : passo : k
                seta(x,y,z,@campo);
            end
        end
    end
end

function seta(x,y,z,F)
    %Desenha uma seta apontando para a direção do campo vetorial naquela
    %posição.

    v = F(x,y,z);
    escala = 2*sqrt( v(1)^2 + v(2)^2 + v(3)^2 ); % = 1; para manter dimensão
original
    v = v / escala;
    plot3( [x, x + v(1)], [y, y + v(2)], [z, z + v(3)] );
    plot3(x + v(1), y + v(2), z + v(3), 'bd ' , 'MarkerSize' , 1.7);
end

function v = campo(x,y,z)
    %Especifica o campo a ser desenhado.

    v(1) = x;
    v(2) = y;
    v(3) = z;
end
```

A primeira função pode também ser declarada como um script, com as outras duas funções sendo declaradas em arquivos separados (conforme explicado na aula 2). A vantagem de declará-la como função é que assim as três funções podem ser salvas em um mesmo arquivo: a segunda e a terceira função se tornam funções auxiliares, que só podem ser usadas internamente. (Ou seja, se salvarmos tudo num

arquivo `desenha_campo.m`, as funções `seta` e `campo` não podem ser usadas diretamente na janela de comando nem em outros arquivos.)

Analisando o conteúdo das funções, vemos que `desenha_campo` possui um "for triplo" que itera sobre vários pontos no espaço. Em cada ponto, a função `seta` desenha uma seta, usando o comando `plot3` aplicado em dois pontos: o ponto base, usado na chamada da função, e o ponto deslocado pelo valor do campo vetorial, descrito na função `campo`. Os parâmetros `k` e `passo` controlam a proximidade das setas, assim como o parâmetro `escala` controla o tamanho das setas (as setas desenhadas mostram apenas a direção do campo e não os módulos dos vetores, mas isso pode ser ajustado).

O resultado:



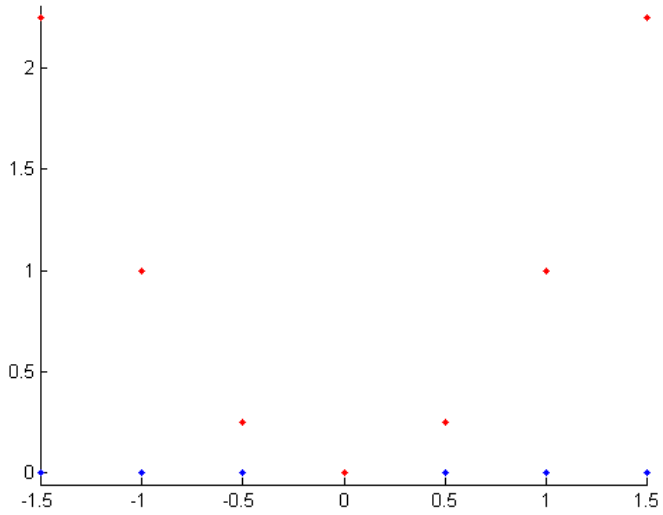
A função meshgrid

A função `meshgrid` é uma função auxiliar, usada em conjunto com as funções `mesh` e `surf` (explicadas adiante) para o desenho de superfícies. O motivo de seu uso é meio complicado, mas não se preocupe, se essa seção lhe parecer confusa, pode pular para a próxima e apenas acreditar que as coisas vão dar certo na hora de usar os comandos `mesh` e `surf`! A função `meshgrid` existe justamente para que o uso das funções `mesh` e `surf` seja mais simples!

Tal como a função linspace existe para criar um domínio para as funções de uma variável serem plotadas com o comando plot, a função meshgrid irá criar o "domínio" das funções de duas variáveis, que será algo parecido com uma "grade".

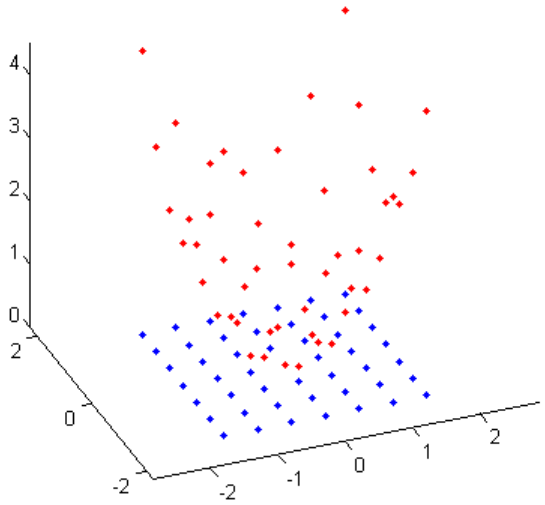
Veja os seguintes exemplos comparativos:

```
x = linspace(-1.5, 1.5, 7);

hold on;
axis equal;

z = 0*x;
plot(x, z, 'b. ');

z = x.^2;
plot(x, z, 'r. ');
```



Aqui plotamos a função f(x) = x² apenas em um conjunto discreto de pontos. Já sabemos que se usarmos uma grade mais "fina", podemos usar o comando plot para traçar segmentos de reta entre esses pontos e assim aproximar o gráfico da função. A linha `z = 0*x;` é apenas um truque: o que queremos é um vetor de mesma dimensão que o vetor x, só que com zeros, para ser usado no comando plot (os vetores usados no comando plot devem ter a mesma dimensão). Os pontos azuis estão destacando o domínio gerado pelo comando linspace.

```
x = linspace(-1.5, 1.5, 7);
y = x;
[x, y] = meshgrid(x, y);

hold on;
axis equal;

z = 0*x;
plot3(x, y, z, 'b. ');

z = x.^2 + y.^2;
plot3(x, y, z, 'r. ');
```



Aqui fizemos o análogo para funções com duas variáveis. O efeito do comando meshgrid ainda não deve estar claro, mas o seu objetivo (obter uma "grade") já deve estar ficando.

Até a segunda linha, x e y são vetores, com as coordenadas que serão consideradas em cada dimensão. O comando meshgrid transforma esses vetores em matrizes, que representarão as coordenadas dos pontos azuis na figura.

Um exemplo pequeno:

```
>> [X,Y] = meshgrid(1:2, 1:3)
X =
   1   2
   1   2
   1   2

Y =
   1   1
   2   2
   3   3
```

A saída foram duas matrizes 3x2, que representam o domínio considerado (6 pontos). A matriz X possui colunas constantes, pois representa a segunda coordenada de cada ponto, analogamente para a matriz Y. Observe como agora fica fácil calcular uma função com x e y, essa representação permite que a conta seja feita elemento a elemento:

```
>> Z = X + Y
Z =
   2   3
   3   4
   4   5
```

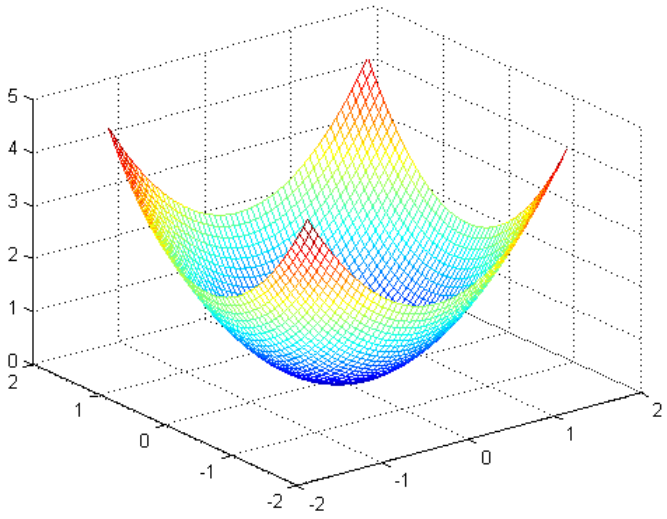
Observe que Z(3,1) = 4, pois é o resultado de X(3,1) + Y(3,1) = 1 + 3.

Por fim, note que não é necessário passar dois argumentos para a função quando estes forem iguais, isto é, `meshgrid(x)` é equivalente a `meshgrid(x,x)`.

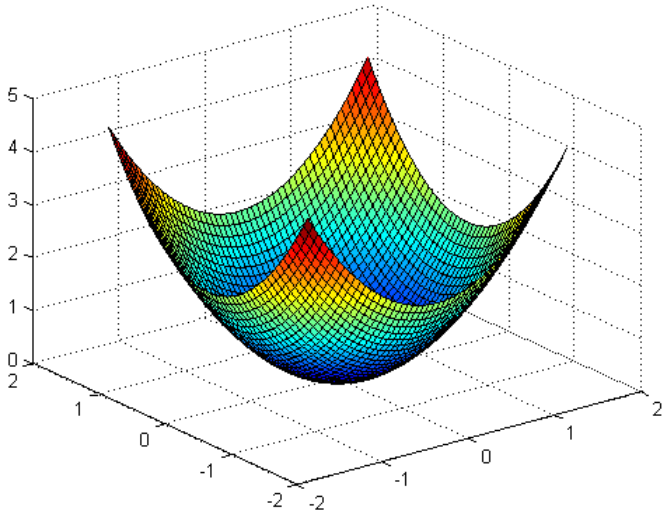
As funções mesh e surf

O uso dessas funções é bem mais simples. Veja um exemplo:

```
x = linspace(-1.5, 1.5, 50);
[x, y] = meshgrid(x);

axis equal;

z = x.^2 + y.^2;
mesh(x, y, z);
```



```
x = linspace(-1.5, 1.5, 50);
[x, y] = meshgrid(x);

axis equal;

z = x.^2 + y.^2;
surf(x, y, z);
```



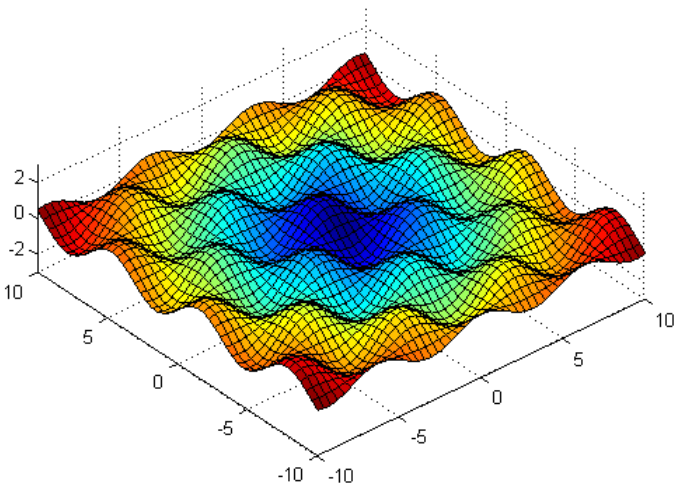
Uma vez definidas as variáveis `x` e `y` através da função `meshgrid` (basta passar como argumento um vetor representando o domínio que elas deverão ter), basta calcular o vetor `z` como função de x e y e usar o comando `mesh`/`surf` para a plotagem da superfície.

Outras funções úteis relacionadas que merecem ser pesquisadas: `meshc`, `contour`, `shading`.

Cores

Os comandos de desenho de superfícies possuem um quarto argumento, opcional, usado para a descrição das cores com que a superfície deve ser desenhado. Deve ser uma matriz com as mesmas dimensões dos outros três argumentos, que determinará a cor no respectivo ponto da superfície através de uma escala de cores. O padrão é a própria matriz Z ser usada para isso, por isso que os desenhos anteriores estão coloridos de acordo com a altura de cada ponto no eixo z.

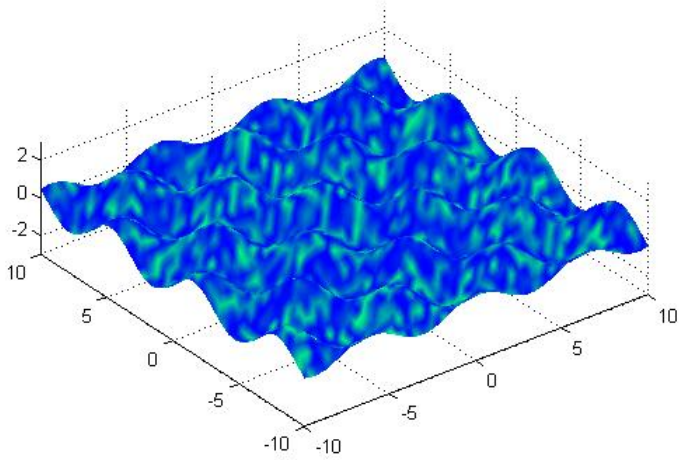
```
x = linspace(-10, 10, 50);
[X, Y] = meshgrid(x);

Z = cos(X).*sin(Y);
C = sqrt(X.^2 + Y.^2);
surf(X, Y, Z, C);

axis equal;
zlim([-3 3]);
```



As escalas de cores podem ser alteradas com o comando `colormap`.

Exemplo:

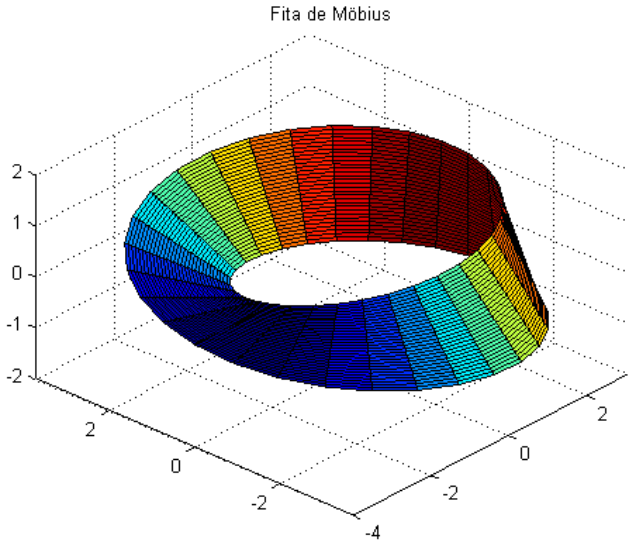
```
x = linspace(-10, 10, 50);
[X, Y] = meshgrid(x);
Z = cos(X).*sin(Y);
C = rand(50).^2;

surf(X, Y, Z, C);

axis equal;
zlim([-3 3]);
colormap winter;
shading interp;
```



Superfícies Parametrizadas

Por fim, vale mencionar que podemos desenhar superfícies parametrizadas da mesma forma que podemos desenhar curvas parametrizadas com o comando `plot`. Basta definir parâmetros u e v com o comando `meshgrid` e então calcular `x`, `y` e `z` em função de `u` e `v`!

```
u = 0 : 2*pi/30 : 2*pi;
v = -1 : 2/30 : 1;

[U,V] = meshgrid(u,v);

k = 1;

X = 3*cos(U) + V.*cos(U).*sin(U/2*k);
Y = 3*sin(U) + V.*sin(U).*sin(U/2*k);
Z = V.*cos(U/2*k);
C = cos(U);

surf(X,Y,Z,C)
title('Fita de Möbius');
axis equal;
zlim([-2,2]);
```



Disponibilizarei na página do Camecc ou na página do evento do curso criada no Facebook arquivos com exemplos de programas escritos por mim. É claro que este curso não cobriu todas os comandos ou possibilidades de aplicação do Matlab, mas espero que usos mais específicos possam ser aprendidos através de exemplos e dos comandos `help` e `doc` do programa.

Obrigado pelo interesse e atenção!