Universidade Tecnológica Federal do Paraná - UTFPR
Campus Curitiba
Departamento Acadêmico de Eletrônica - DAELN
Curso de Engenharia Eletrônica
Prof. Gustavo B. Borba

Atividade #01
Vale nota, individual, observar prazo e instruções de entrega no moodle

Arquivos necessários

1. componente.mat
2. gamble_mega.m
3. getMega.m
4. megasena.csv
5. queryMega.m

1.1) Gráficos 2D

Exemplo

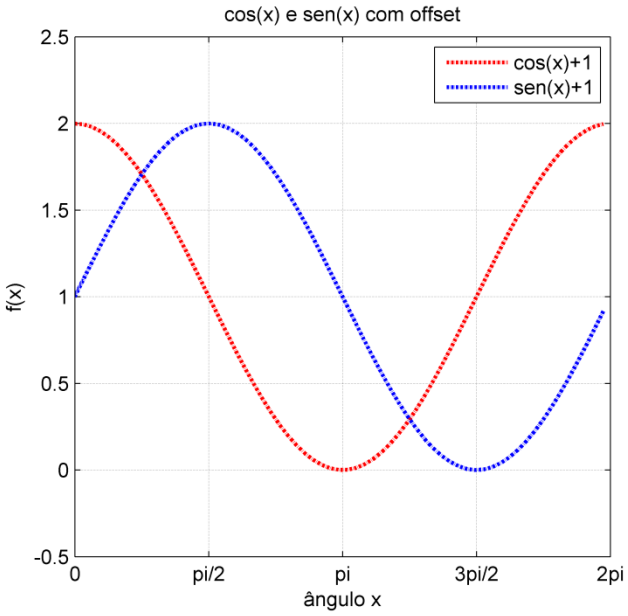
```
%y = f(x)
x = 0:0.1:2*pi; %eixo x do gráfico
y = cos(x)+1;  %função 1
y2 = sin(x)+1; %função 2

plot(x, y, 'r-.', 'LineWidth', 2);
hold on %mantém a útima curva plotada
plot(x, y2, 'b-.', 'LineWidth', 2);
hold off %desabilita o hold

%Legenda
legend('cos(x)+1', 'sen(x)+1', 1)
%1:canto sup. dir. 2:canto sup. esq.
%3:canto inf. esq. 4:canto inf. dir.

%limites dos eixos: axis([xmin xmax ymin ymax])
axis([0 2*pi -0.5 2.5]);
%Espaçamento entre os "ticks"
set(gca,'XTick',0:pi/2:2*pi);
%Label dos "ticks"
set(gca,'XTickLabel', {'0', 'pi/2', 'pi',...
    '3pi/2', '2pi'});
grid on;

xlabel('ângulo x'); %label do eixo X
ylabel('f(x)'); %label do eixo y
title('cos(x) e sen(x) com offset'); %título
```



Elabore um script que plota as duas funções especificadas ao lado no mesmo gráfico. O grid deve ser mostrado.

$$P(t) = \frac{C}{1 + ae^{-bt}}$$

a) $P(t)$ **para:**
C = 250
a = 60
b = 0.6
Intervalo: de 0 até 20 com passo igual a 1.
Cor: vermelho
Legenda: '0.6'

b) $P(t)$ **para:**
C = 250
a = 60
b = 0.4
Intervalo: de 0 até 20 com passo igual a 1.
Cor: verde
Legenda: '0.4'

Nome do .m: atv01_01.m

1.2) Gráficos 3D

Exemplo

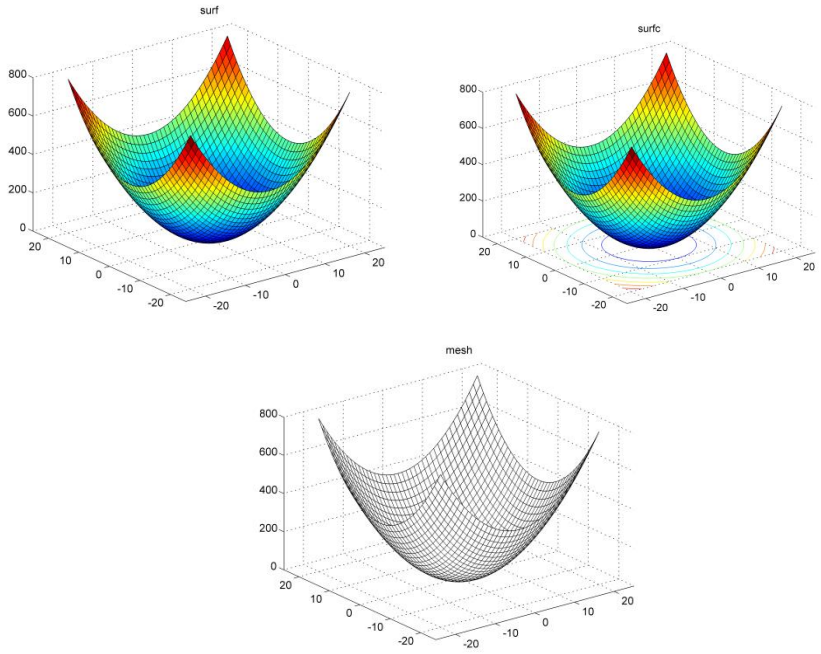
```
clear all
close all
clc

%Define o plano xy para o qual
%a função Z será avaliada
x = -20:1:20;
y = -20:1:20;
%X e Y contém todos os pares de
%coordenadas do plano xy
[X Y] = meshgrid(x,y);
%Calcula Z para todos os pares
%de coordenada do plano xy
Z = X.^2 + Y.^2;

%Função surf
figure
surf(X, Y, Z)
xlim([-25 25]), ylim([-25, 25])
title('surf')

%Função surfc
figure
surfc(X, Y, Z)
xlim([-25 25]), ylim([-25, 25])
title('surfc')

%Função mesh com linhas pretas
figure
mesh(X, Y, Z, 'EdgeColor',…
'black')
xlim([-25 25]), ylim([-25, 25])
title('mesh')
```



Elabore um script que plota o gráfico 3D da função ao lado. Use a função que preferir: *surf*, *surfc* ou *mesh*.

$$z(x,y) = x^4 - 4x^2y^2 + y^4$$

Intervalo: de -10 até +10 com passo igual a 1 para ambos os eixos.

Nome do .m: atv01_02.m

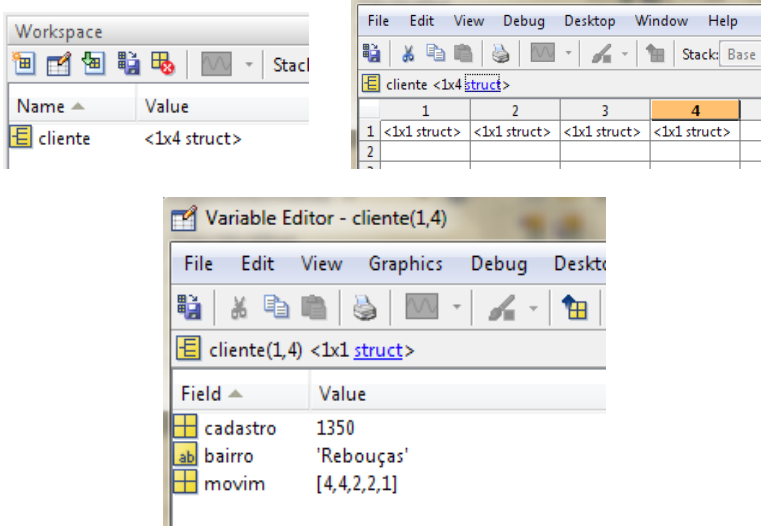
1.3) Função do usuário, estrutura, vetor de estruturas, vetorização

Exemplo

```
%Script que cria vetor com 4 estruturas
%Cada estrutura contém os dados de um
%cliente de uma locadora
clear all;

%Campos da estrutura "cliente":
%cadastro: número da carteirinha
%bairro: residência do titular (string)
%movim: vetor com número de locações em
%cada mes [mes1 mes2 mes3 mes4 mes5]

cliente(1) = struct('cadastro', 1347,...
    'bairro', 'Rebouças',...
    'movim', [0 1 3 2 3]);
cliente(2) = struct('cadastro', 1348,...
    'bairro', 'Água Verde',...
    'movim', [2 3 4 4 5]);
cliente(3) = struct('cadastro', 1349,...
    'bairro', 'Centro',...
    'movim', [1 0 5 6 2]);
cliente(4) = struct('cadastro', 1350,...
    'bairro', 'Rebouças',...
    'movim', [4 4 2 2 1]);
```



Função que varre as estruturas usando laço for

```
function cad = clienteMaiorMov(clienteLoc, mes)
%cad = clienteMaiorMov(clientLoc, mes)
%Função que varre as estruturas em clienteLoc e
%devolve em cad o cadastro do cliente com maior
%movimentação no mês especificado pelo
%parâmetro de entrada mes.
%Exemplo: c = clienteMaiorMov(cliente, 3);
%Veja também: script que cria o vetor de
%estruturas 'cliente'.

%O número de clientes é o número de estruturas
%do vetor de estruturas
nClient = length(clienteLoc);

%Aloca
movimMes = zeros(1,length(clienteLoc));

%Varre as estruturas
for k=1:nClient
    %armazena em movimMes todas as movimentações
    %do mês especificado por mes
    movimMes(k) = clienteLoc(k).movim(mes);
end

%O índice do máximo valor do vetor é o
%índice da struct correspondente ao
%cliente com maior movimentação
[valor, indice] = max(movimMes);

%Retorna o cadastro do cliente
%com máxima movimentação
cad = clienteLoc(indice).cadastro;
```

Mesma função vetorizada (sem o laço for)

```
function cad = clienteMaiorMovVec(clienteLoc, mes)
%cad = clienteMaiorMov(clientLoc, mes)
%Função que varre as estruturas em clienteLoc e
%devolve em cad o cadastro do cliente com maior
%movimentação no mês especificado pelo
%parâmetro de entrada mes.
%Exemplo: c = clienteMaiorMov(cliente, 3);
%Veja também: script que cria o vetor de %estruturas
'cliente'.

%O número de clientes é o número de estruturas
%do vetor de estruturas
nClient = length(clienteLoc);
%O número de meses é o comprimento do vetor movim
nMes = length(clienteLoc(1).movim);

%Concatena os vetores movim de todas as estruturas
temp = [clienteLoc.movim];

%Cada coluna contém a movim de um cliente
allMovim = reshape(temp, nMes, nClient);
%Movimentação de cada cliente no mes desejado
movimMes = allMovim(mes,:);

%O índice do máximo valor do vetor é o
%índice da struct correspondente ao
%cliente com maior movimentação
[valor, indice] = max(movimMes);

%Retorna o cadastro do cliente com máxima
movimentação
cad = clienteLoc(indice).cadastro;
```

Ao abrir o arquivo *componente.mat* no Octave, um vetor de estruturas chamado 'componente' é carregado no workspace. Cada estrutura possui os dados dos testes de um determinado componente mecânico. Os campos *(fields)* são os seguintes:

id: identificador (escalar)
tipo: 'A' ou 'B' (caractere)
horas: número de horas de operação até uma falha (escalar)

Elabore uma função denominada *atv01_03_estatComponente* para levantar alguns dados estatísticos dos testes. As especificações da função são:

[med dp] = atv01_03_estatComponente(comp, tipoc)
Entrada:
comp: vetor de estruturas 'componente'
tipoc: 'A' ou 'B'
Saída:
med: média das horas de todos os componentes do tipo especificado pelo parâmetro de entrada tipoc.
dp: desvio padrão das horas de todos os componentes do tipo especificado pelo parâmetro de entrada tipoc.

Nome do .m: atv01_03_estatComponente.m

1.4) Cell array

Exemplo

```
clear all, close all
clc

%Aloca uma cell array
myCell = cell(1,4);

%Cell do lado esquerdo do '='
myCell{1} = [0 2 4 8];
myCell{2} = [1 3 5 7]';
myCell{3} = [0 1 2; 3 4 5];
myCell{4} = 'charaquiteres';
%Display
whos myCell
disp('myCell{1} = '), disp(myCell{1})
disp('myCell{2} = '), disp(myCell{2})
disp('myCell{3} = '), disp(myCell{3})
disp('myCell{4} = '), disp(myCell{4})
disp(' ')
disp('Cell array é versátil!')
disp('Armazena dados de qualquer')
disp('classe e tamanho.')
disp('Aperta uma tecla aí pra continuar.')
disp('================================')
pause

%Cell do lado esquerdo do '='
myCell{1} = 1000;
myCell{2}(5) = 9;
myCell{3}(2,3) = 50;
myCell{4}(end-1:end) = [];
```

Continua na coluna ao lado

```
%Display
disp('myCell{1} = '), disp(myCell{1})
disp('myCell{2} = '), disp(myCell{2})
disp('myCell{3} = '), disp(myCell{3})
disp('myCell{4} = '), disp(myCell{4})
disp(' ')
disp('Vc alterou o conteúdo das cells do cell array.')
disp('Aperta uma tecla aí pra continuar.')
disp('================================')
pause

%Cell do lado direito do '='
%'()' retorna uma cell do cell array
d = myCell(3);
%Display
whos d
disp('Fazendo d = myCell(3), ''d'' é uma cell')
disp('Aperta uma tecla aí pra continuar.')
disp('================================')
pause

%Cell do lado direito do '='
%'{}' retorna o conteúdo de uma cell do cell array
e = myCell{3};
%Display
whos e
disp('Fazendo e = myCell{3}, ''e'' é o conteúdo da
cell')
disp('Aperta uma tecla aí pra continuar.')
disp('================================')
pause

disp('Acabou')
```

Elabore um script que cria uma cell array de três elementos. Armazene em cada um uma matriz magic de dimensões 5-por-5, 4-por-4 e 3-por-3. Mostre o valor da soma das colunas de cada matriz magic.

Nome do .m: atv01_04

1.5) Função do usuário, importação de dados, vetorização

Você vai usar o script *gamble_mega*, as funções *getMega* e *queryMega* e o arquivo *megasena.csv* (os sorteios da megasena de verdade!). Abaixo são mostrados apenas os pedaços destes códigos referentes à descrição dos seus conteúdos.

```
%gamble_mega [script]

%============================== USER
% Sua aposta de 6 números
% em ordem crescente
aposta = [7 13 36 42 53 57];
%========================== END USER
```

Este script chama as funções ao lado

```
function [ms] = getMega(csvFile)
% [ms] = getMega(csvFile) lê o arquivo .CSV 'csvFile'
% com todos os sorteios da megasena e retorna os valores
% em uma matriz m-por-n, onde n é o número de
% sorteios e n=6. 'csvFile' default = 'megasena.csv'.
% Veja também: gamble_mega, queryMega.

function [smp dcb] = queryMega(aps, allMs)
% [smp dcb] = queryMega(aps, allMs) calcula a distância
% city-block (diferença absoluta) entre a aposta 'aps' e todos
% os sorteios em 'allMs'. Retorna o sorteio que apresentou a
% menor ditância em 'smp' e a distância em 'dcb'.
% Veja também:  gamble_mega, getMega.
```

Elabore uma função chamada *atv01_05_queryMegaVec* que substitui o laço *for* de *queryMega* por operações com matrizes (vetorização). Teste a sua função *queryMegaVec* chamando-a a partir do script *gamble_mega*. O resultado deve ser idêntico ao da função *queryMega*.

Nome do arquivo .m: atv01_05_queryMegaVec

1.6) Função do usuário, importação de dados, vetorização

O framework da atividade 1.5 é legal pra aprender Octave, mas não pra saber se você realmente se daria bem na megasena.

Crie e teste uma função chamada *atv01_06_queryMegaHits*, para ser chamada a partir do script *gamble_mega*, que ordena os sorteios de acordo com o número de hits (acertos). Observe que a *queryMega* ordena e compara apenas as dezenas da mesma posição. Este não é o procedimento correto para saber se você acertou. Na situação abaixo, por exemplo, a função *queryMega* diria que o número de acertos é 1, mas o correto é 3.

Aposta: 4, 5, 6, 27, 34, 40
Sorteio: 1, 4, 5, 30, 33, 40

Nome do arquivo .m: atv01_06_queryMegaHits