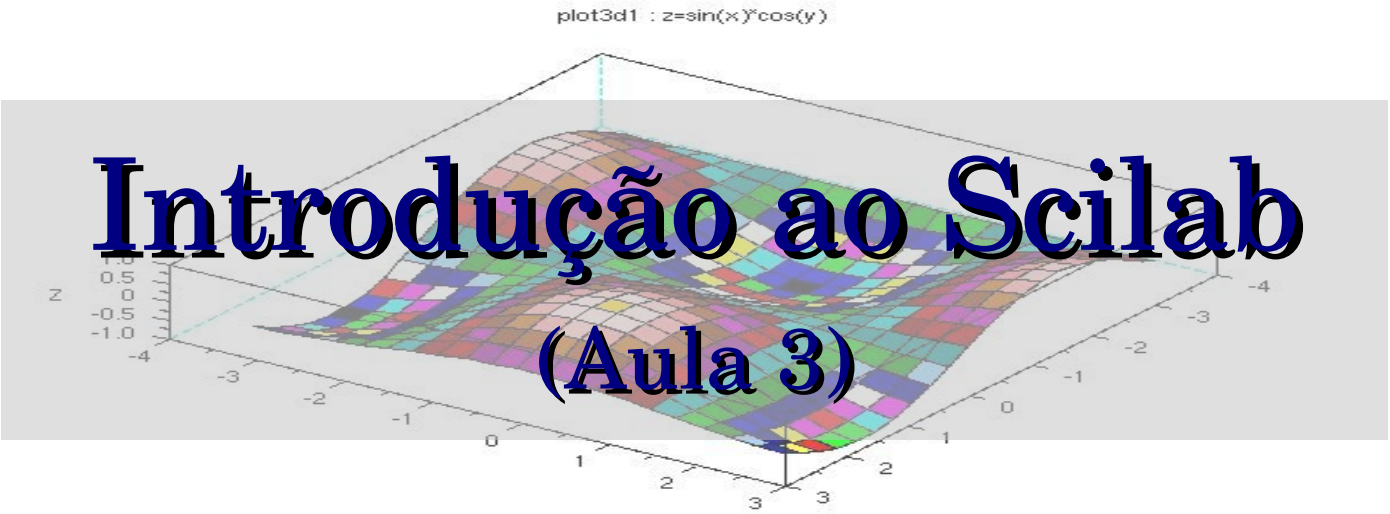
Universidade do Estado do Rio de Janeiro
Faculdade de Engenharia
Laboratório de Engenharia Elétrica



Introdução ao Scilab

(Aula 3)

Elaine de Mattos Silva
Orientador: Prof. José Paulo Vilela Soares da Cunha

Abril de 2007

Apoio: Programa de Estágio Interno Complementar do CETREINA/SR-1/UERJ

Contatos

- E-mail:
  - elaine@lee.eng.uerj.br

- Página do curso:
  - http://www.lee.eng.uerj.br/~elaine/scilab.html

- Apostila *Introdução ao Scilab versão 3.0*:

  Prof. Paulo Sérgio da Motta Pires (UFRN)

  http://www.dca.ufrn.br/~pmotta

Conteúdo Geral

- Aula 1
  - O que é o Scilab
  - Principais Características do ambiente Scilab
  - Operações Básicas
- Aula 2
  - Polinômios, Vetores e Matrizes
- Aula 3
  - Listas
  - Programação com Scilab
- Aula 4
  - Gráficos em Scilab
  - Introdução ao Scicos

Aula 3

- Aula 3

1 – Listas

Programação com Scilab

2 - Características da linguagem Scilab

3 - Comandos para Iterações

3.1 - O laço *for*

3.2 - O laço *while*

4 - Comandos Condicionais

4.1 - *If-then-else*

4.2 - *Select-case*

Aula 3 (cont.)

- Aula 3 (cont.)

5 - *Scripts*

6 - Funções

6.1 - Variáveis globais x locais

7- Exercícios

1 – Listas

- Uma lista é um agrupamento de objetos não necessariamente do mesmo tipo.

- Uma lista simples é definida no Scilab pelo comando *list*, que possui esta forma geral:

$$\text{list}(a_1,a_2,\ldots,a_n)$$

onde a_1, a_2,..., a_n são os elementos da lista

1 – Listas (cont.)

- Para exemplificar criou-se uma lista composta do número “23” , o caracter “q” e uma matriz identidade 2x2.

$$L=(23,q,\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix})$$

```
-->L=list(23, 'q',eye(2,2))
 L  =
        L(1)        //elemento 1= 23
     23.
        L(2)        //elemento 2= q
  q
        L(3)        //elemento 3=matriz identidade 2x2
    1.    0.
    0.    1.
```

1 – Listas (cont.)

- Criando sublistas:

Podemos criar sublistas, ou seja, listas dentro de listas. Para exemplificar transformamos o segundo elemento da lista L em uma lista de dois elementos. Repare que o primeiro elemento continua sendo o caracter "q" mas o segundo elemento passa a ser uma *string* "abc".

$$L=(23,q,\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}) \qquad L=(23,(q,abc),\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix})$$

1 – Listas (cont.)

```
-->L(2)=list('q','abc')  //faz L(2) = (q , abc)
 L  =
        L(1)
     23.
        L(2)

        L(2)(1)
     q
        L(2)(2)
     abc
        L(3)
     1.    0.
     0.    1.
```

1 – Listas (cont.)

- Acesso a elementos de uma lista:

Ex.1: acessando o primeiro elemento:

```
-->L(1)
 ans  =

    23.
```

Ex.2: acessando o segundo elemento da sub lista dentro de L(2):

```
-->L(2)(2)
 ans  =

    abc
```

Programação com Scilab

2 - Características da Linguagem Scilab

- O Scilab é um interpretador de comandos e por isso o código gerado não precisa ser compilado.

- Facilidade e simplicidade da linguagem estruturada.

- Não há necessidade de declaração prévia das variáveis.

3 – Comandos para Iteração

- Existem duas estruturas de controle iterativo no Scilab: o laço *for* e o comando *while*.

3 – Comandos para Iteração

3.1 - O laço *for*

- Forma geral:

```
for variavel = vetor_linha ou lista
    instrucao_1
    instrucao_2
    instrucao_3
end
```

  - O comportamento das iterações é baseado no vetor linha ou na lista. Se o vetor tem 3 elementos, existirão 3 iterações.

3 – Comandos para Iteração (cont.)

3.1 - O laço *for*

Ex.1(usando variável tipo vetor):

```
m=1:3;
for k=m
    a=k+1
end
```

```
-->m=1:3;
-->for k=m
-->a=k+1
-->end
 a   =
    2.
 a   =
    3.
 a   =
    4.
```

3 – Comandos para Iteração (cont.)

3.1 - O laço *for*

Ex.2 (usando variável tipo vetor):

```
 y=0;
 m=3:5;
for k=m,  y=y+k,end
```

```
-->y=0;
-->for k=3:5,y=y+k,end
 y  =
    3.
 y  =
    7.
 y  =
    12.
```

3 – Comandos para Iteração (cont.)

3.1 - O laço *for*

Ex.3 (usando variável tipo lista):

```
L=list(2, [2 4;6 8], 'exemplo');
for k=L,disp(k), end
```

```
->L=list(2,[2 4;6 8], 'exemplo');
-->for k=L,disp(k),end  //
    2.

    2.    4.
    6.    8.

 exemplo
```

3 – Comandos para Iteração (cont.)

3.2 - O laço *while*

- Forma geral:

```
while condicao
        instrucao_1
        instrucao_2
          ...     ...
        instrucao_n
 end
```

- O laço *while* repete uma seqüência de instruções enquanto uma condição for satisfeita.
- Útil quando não se sabe o número de iterações.

3 – Comandos para Iteração (cont.)

3.2 - O laço *while*

- Operadores permitidos:

== ou =	(igual a)
<	(menor que)
>	(maior que)
<=	(menor ou igual)
>=	(maior ou igual)
<> ou ~=	(diferente)

3 – Comandos para Iteração (cont.)

3.2 - O laço *while*

Ex.1: x=1;

```
while x<14
    x = 2*x
end
```

```
-->x=1;
-->while x<14     //enquanto x for menor que 14
-->x=2*x          //instrucao : faca x = 2x
-->end
 x  =
    2.            //x=1*2=2
 x  =
    4.            //x=2*2=4
 x  =
    8.            //x=4*2=8
 x  =
    16.           //x=8*2=16. Como x>14 o loop termina.
```

4 – Comandos Condicionais

- Comandos condicionais são usados para executar seqüências de instruções a partir da avaliação de condições booleanas.

4 – Comandos Condicionais

4.1 - *if – then – else*

- Forma simples:

```
if condicao_1 then
        sequencia_1
else
        sequencia_2
end
```

- Avalia a *condicao_1* se ela for verdadeira (T, *true*) executa a *sequencia_1*, caso contrário executa a *sequencia_2*.

4 – Comandos Condicionais (cont.)

4.1 - *if – then – else*

- Forma geral:

```
if condicao_1 then
        sequencia_1
elseif condicao_2
        sequencia_2
else
        sequencia_3
end
```

- Se a *condicao_1* for verdadeira executa a *sequencia_1*.
- Se a *condicao_1* for falsa avalia a *condicao_2* e assim por diante.
- Se todas as condições são falsas executa a *sequencia_3*.

4 – Comandos Condicionais (cont.)

4.1 - *if – then – else*

- Ex.1(forma simples):

```
-->x=-1;

-->if x<0 then          //se x for menor que 0
-->y=-x;                //faca y=-x
-->else                 //caso contrario
-->y=x;                 //faca x=x
-->end
-->disp(y)              //mostra valor de y

    1.
```

4 – Comandos Condicionais (cont.)

4.1 - *if – then – else*

- Ex.2(forma geral):

```
-->x=10;
-->if x<0 then
-->y=x;
-->elseif x==1
-->y=2*x;
-->elseif x==2
-->y=3*x;
-->elseif x==3
-->y=4*x;
-->else
-->y=5*x;
-->end
-->disp(y)
50.
```

4 – Comandos Condicionais (cont.)

4.2 - *select - case*

- Forma geral:

```
select variavel_de_teste
    case expressao_1
        sequencia_1
    case expressao_n
        sequencia_n
    else
        sequencia_n+1
    end
```

- O valor da *variavel_de_teste* é comparado às expressões.
- Se os valores são iguais, a seqüência correspondente é executada.

4 – Comandos Condicionais (cont.)

4.2 - *select - case*

- Ex.:

```
-->M=['a' 'b'];   //define matriz simbolica
-->select M(1,2)  //seleciona elemento (1,2) de M

-->case 'a'                          //se o elemento (1,2)=a
-->disp('letra a encontrada')  //escreve 'letra a ...'

-->case 'b'                          //se o elemento (1,2)=b
-->disp ('letra b encontrada') //escreve 'letra b...'


-->end
```

5 – *Scripts*

- Os *scripts* são arquivos de texto puro que contém comandos que seriam usados em um *prompt* do Scilab.

- Por convenção estes arquivos possuem extensão *.sce*

- Os arquivos são criados no editor de texto do Scilab, o Scipad(ou em qualquer outro editor de texto).
- Os arquivos são executados no Scilab:
    - com o comando *exec*, ou
    - com o menu *File> File Operations* selecionando o arquivo e clicando no botão *exec*

5 – *Scripts* (cont.)

- Ex.1 – *script* que calcula as raízes quadradas dos números inteiros de -10 a 10.

Obs.: Este *script* deve ser digitado em um editor de textos e salvo com a extensão .sce

```
//script que calcula raizes dos numeros inteiros
//de -10 a 10
n=0;
for x=-10:10
   n=n+1;
   y(n)=sqrt(x);
end
y
```

5 – *Scripts* (cont.)

- Para executar este script use o comando *exec nome_do_script.sce* no *prompt* do Scilab:

```
-->exec raizes.sce        //chamando o script raizes.sce
-->//script que calcula as raízes quadradas dos números
inteiros de -10 a 10.
-->n=0;
-->for x=-10:10
-->n=n+1;
-->y(n)=sqrt(x);
-->end
-->y
 y  =
    3.1622777i
    3.i
    2.8284271i
    2.6457513i
    ...
```

5 – *Scripts* (cont.)

- Caso o comando *exec* seja executado com ' ; ' (ponto e vírgula) no fim apenas os resultados são apresentados:

```
-->exec raizes.sce;
     y  =
        3.1622777i
        3.i
        2.8284271i
        2.6457513i
        2.4494897i
        2.236068i
        2.i
        1.7320508i
        1.4142136i
        i
        0
        1.
        ...
```

6 – Funções

- É possível definir novas funções no Scilab;

- O que distingue uma função de um *script* é que a função possui um ambiente local, separado do global, mas que se comunica através de argumentos de entrada e saída;

- Variáveis definidas no escopo da função (variáveis locais) não permanecem no ambiente após a execução da função;

6 – Funções

- Definição :

Uma função pode ser definida de três formas:

- no ambiente Scilab;
- usando o comando *deff* ou
- digitando o texto no Scipad e clicando no menu *Execute*, opção *load into Scilab*

6 – Funções

- Definição no ambiente:

function [y1,...,yn]= nome_da_funcao(x1,...,xm)

instrucao_1

instrucao_2

...

instrucao_p

endfunction

onde:

- x1,...,xm são os argumentos de entrada;
- y1,...,yn são argumentos de saída e
- instrucão_1,...,instrucao_p são as instruções executadas pela função.

6 – Funções (cont.)

- Ex.(definição no ambiente):

```
-->function[y1]=funcao1(x1,x2)
-->y1=x1+x2
-->endfunction


-->[a]=funcao1(1,2)  //chamando a funcao

 a  =

    3.

-->//o valor da função retorna na variavel 'a'
```

6 – Funções (cont.)

- Definição usando o comando *deff*:

deff('[y1,...,yn]=nome_da_funcao(x1,...,xm)','instrucao_1,...,instrucao_p')

onde:

- x1,...,xm são os argumentos de entrada;
- y1,...,yn são argumentos de saída e
- instrucão1,...,instrucao_p são as instruções executadas pela função.

6 – Funções (cont.)

- Ex.(definição usando o comando *deff)*:

```
-->deff('[y1]=funcao2(x1,x2)', 'y1=x1+x2')
-->[a]=funcao2(1,2)   //chamando a funcao


 a  =
    3.
```

6 – Funções (cont.)

6.1 - *variáveis globais* x *variáveis locais*

- Variáveis globais são válidas no ambiente Scilab;
- Variáveis locais são válidas apenas no escopo de uma função.

Ex. variáveis globais:

```
-->x3=5;
-->function[y1]=f(x1,x2)
-->y1=x1+x2+x3
-->endfunction
-->[r1]=f(1,3)      //entrada=(1,3), saída=[r1]
 r1  =
    9.
```

6 – Funções (cont.)

6.1 - *variáveis globais* x *variáveis locais*

Ex. variáveis locais:

```
-->function[y1]=f(x1,x2)
-->y1=x1+x2
-->endfunction
-->[r1]=f(1,3)      //entrada=(1,3), saída=[r1]
 r1  =
    4.
-->y1                //y1 nao existe fora da funcao

  !--error 4
undefined variable : y1
```

Exercícios

7 – Exercícios

7.1 - *Exercícios com scripts*

- Calcular $\sqrt{2}$ usando o método de Newton-Raphson.

A raiz de uma função, f(x) pode ser obtida através da expressão:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

7 – Exercícios

7.1 - *Scripts*

A função que permite calcular $\sqrt{2}$ é:

$$f(x) = x^2 - 2$$

usando a fórmula de Newton-Raphson:

$$x_{i+1} = x_i - \frac{x^2 - 2}{2x}$$

7 – Exercícios

7.1 - *Scripts*

- É necessária uma aproximação inicial. Podemos dizer o resultado será próximo de 1, então fazemos x_0=1.

- Façamos i variar de 1 a 10, completando 10 iterações.

- Consideremos um erro de 10^{-5}.

7 – Exercícios

7.1 - *Scripts*

Script para calcular a raiz de 2 através do método de Newton-Raphson*

```
N = 10;                              //número máximo de iterações
x0 = 1.0;                            //aproximação inicial
erro = 10^(-5);

xn = x0;                             //valor inicial da raiz
for n = 1:N
    xn1 = xn – (xn * xn - 2)/(2 * xn);
    if abs((xn1-xn) / xn1) < erro then
       printf( ' Valor da raiz = %10.7f ' , xn1 )
       return
    end

       xn = xn1;
end
```

* este script foi retirado da apostila Introdução ao Scilab – ver referências no fim destes slides

7 – Exercícios

7.1 - *Exercícios com scripts (cont).*

- Digite este *script* no editor do Scilab e execute-o no ambiente Scilab clicando em Execute -> Load into Scilab' ;

- O resultado esperado é:

```
--> Valor da raiz =  1.4142136
```

7 – Exercícios

7.2 - *Exercícios com funções*

Calcular o fatorial de um número usando uma função recursiva:

- *Para calcular o fatorial definiremos a função fat(x). Dizemos que ela é recursiva porque chama a si mesma.*
- *Para definir a função usamos o editor do Scilab:*

```
function y = fat(x)
if x <= 1 then
   y = 1;
   else
      y = x*fat(x-1);
end
endfunction
```

Referências

- Pires, P.S.M. (2004). *Introdução ao Scilab*, Rio Grande do Norte.
- Noble, B. e Daniel, J.W. (1986). *Álgebra Linear Aplicada*, Prentice Hall do Brasil, Rio de Janeiro.