

Introdução ao Scilab

Leonardo F. Guidi

DMPA – IM
UFRGS

Cálculo Numérico



Índice

- 1 **Introdução**
 - Origem
 - Execução
 - Interação com o console
- 2 **Objetos – sintaxe**
 - Tipos de objetos (variáveis)
 - Matrizes de constantes
 - Demais tipos
- 3 **Programação – sintaxe**
 - Fluxo de execução
 - Entrada e saída
 - Gráficos

Introdução

- O que é? Um pacote de softwares para computação numérica que proporciona um ambiente de desenvolvimento para aplicações científicas e de engenharia.
- Desenvolvido a partir de código livre e distribuído como tal.
Disponível em sistemas Unix, GNU/Linux e Windows.
- Criado em 1994 dentro do INRIA (Institut National de Recherche en Informatique et Automatique).
- Desde 2003, o desenvolvimento é gerenciado pelo Consórcio Scilab.

Introdução

- O que é? Um pacote de softwares para computação numérica que proporciona um ambiente de desenvolvimento para aplicações científicas e de engenharia.
- Desenvolvido a partir de código livre e distribuído como tal.
Disponível em sistemas Unix, GNU/Linux e Windows.
- Criado em 1994 dentro do INRIA (Institut National de Recherche en Informatique et Automatique).
- Desde 2003, o desenvolvimento é gerenciado pelo Consórcio Scilab.

Introdução

- O que é? Um pacote de softwares para computação numérica que proporciona um ambiente de desenvolvimento para aplicações científicas e de engenharia.
- Desenvolvido a partir de código livre e distribuído como tal.
Disponível em sistemas Unix, GNU/Linux e Windows.
- Criado em 1994 dentro do INRIA (Institut National de Recherche en Informatique et Automatique).
- Desde 2003, o desenvolvimento é gerenciado pelo Consórcio Scilab.

Introdução

- O que é? Um pacote de softwares para computação numérica que proporciona um ambiente de desenvolvimento para aplicações científicas e de engenharia.
- Desenvolvido a partir de código livre e distribuído como tal. Disponível em sistemas Unix, GNU/Linux e Windows.
- Criado em 1994 dentro do INRIA (Institut National de Recherche en Informatique et Automatique).
- Desde 2003, o desenvolvimento é gerenciado pelo Consórcio Scilab.

Introdução

- O que é? Um pacote de softwares para computação numérica que proporciona um ambiente de desenvolvimento para aplicações científicas e de engenharia.
- Desenvolvido a partir de código livre e distribuído como tal. Disponível em sistemas Unix, GNU/Linux e Windows.
- Criado em 1994 dentro do INRIA (Institut National de Recherche en Informatique et Automatique).
- Desde 2003, o desenvolvimento é gerenciado pelo Consórcio Scilab.

Introdução

- O que é? Um pacote de softwares para computação numérica que proporciona um ambiente de desenvolvimento para aplicações científicas e de engenharia.
- Desenvolvido a partir de código livre e distribuído como tal. Disponível em sistemas Unix, GNU/Linux e Windows.
- Criado em 1994 dentro do INRIA (Institut National de Recherche en Informatique et Automatique).
- Desde 2003, o desenvolvimento é gerenciado pelo Consórcio Scilab.

Consórcio Scilab

Instituições de Ensino



Organizações Públicas



INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE



Research in science and technology of information



CENTRE NATIONAL D'ÉTUDES SPATIALES

Empresas



PSA PEUGEOT CITROËN



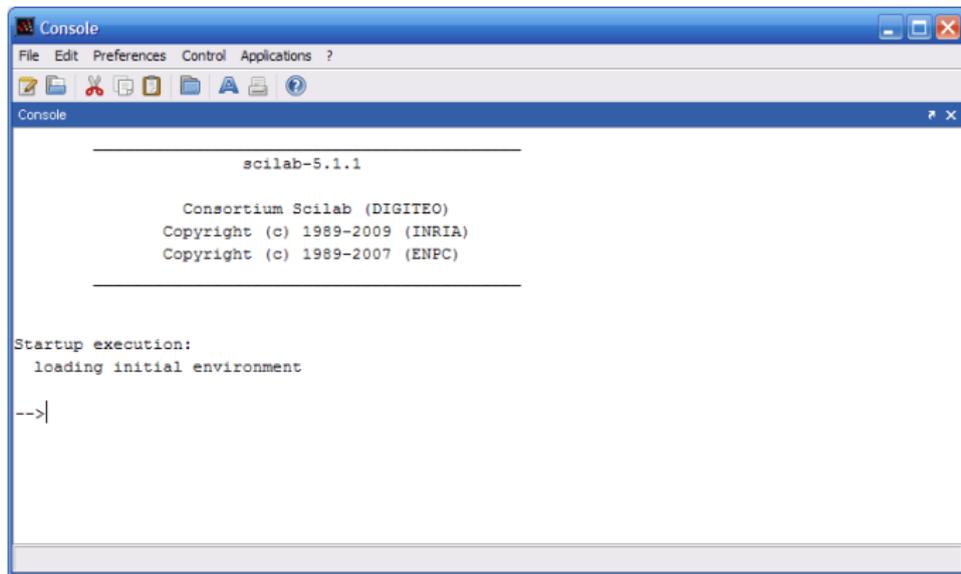
Interfaces

Durante a execução do Scilab, várias janelas podem ser utilizadas. As principais são:

Interfaces

Durante a execução do Scilab, várias janelas podem ser utilizadas. As principais são:

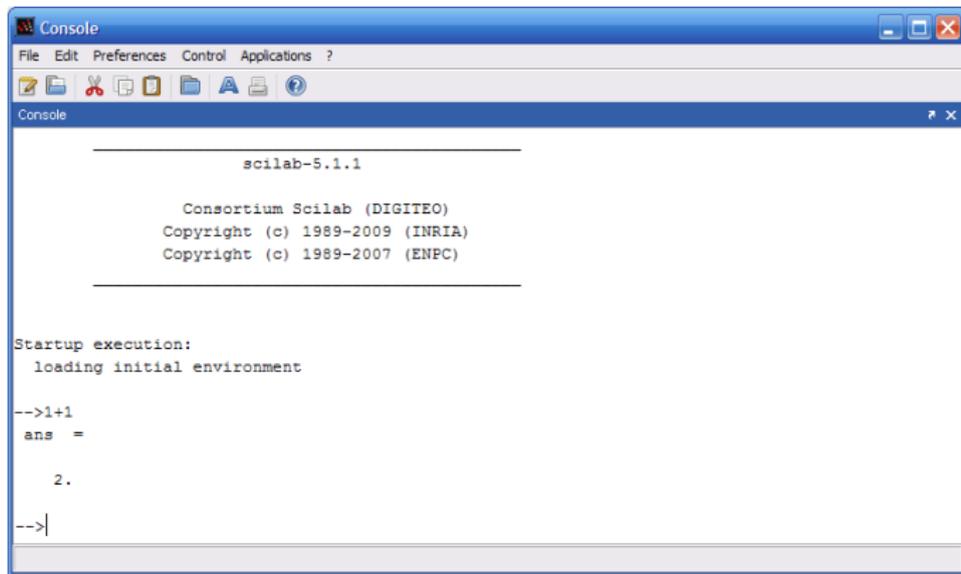
O console



Interfaces

Durante a execução do Scilab, várias janelas podem ser utilizadas. As principais são:

O console



```
scilab-5.1.1

Consortium Scilab (DIGITEO)
Copyright (c) 1989-2009 (INRIA)
Copyright (c) 1989-2007 (ENPC)

Startup execution:
loading initial environment

-->1+1
ans =

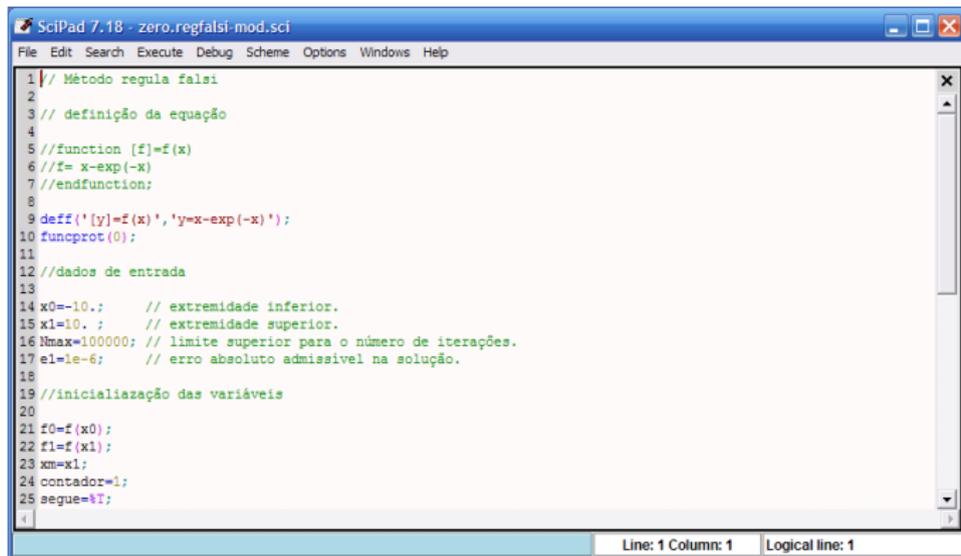
    2.

-->|
```

Interfaces

Durante a execução do Scilab, várias janelas podem ser utilizadas. As principais são:

O editor SciPad



```
SciPad 7.18 - zero.regfalsi-mod.sci
File Edit Search Execute Debug Scheme Options Windows Help

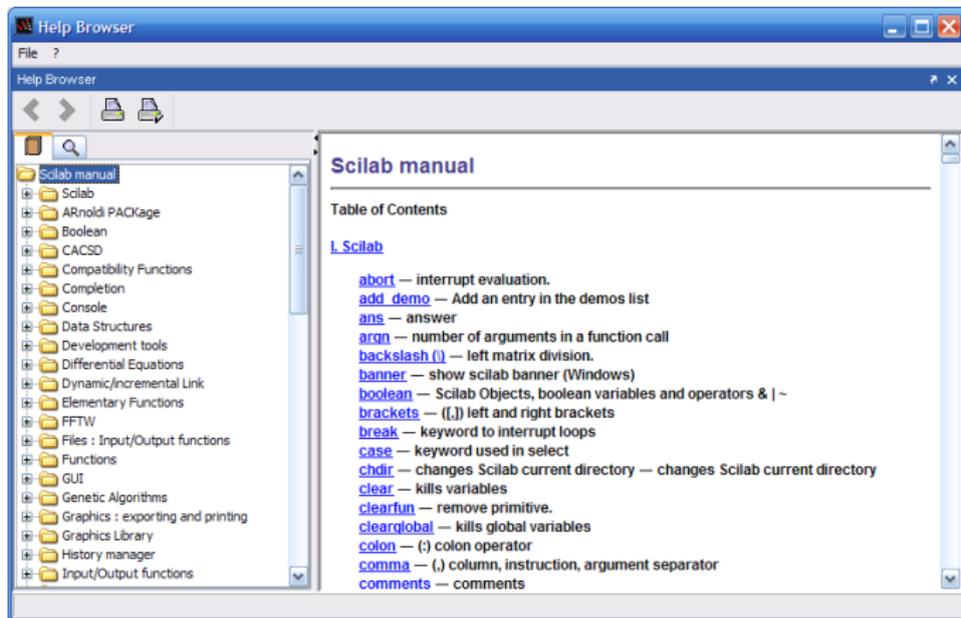
1 // Método regula falsi
2
3 // definição da equação
4
5 //function [f]=f(x)
6 //f= x-exp(-x)
7 //endfunction;
8
9 deff('[y]=f(x)', 'y=x-exp(-x)');
10 funcprot(0);
11
12 //dados de entrada
13
14 x0=-10.; // extremidade inferior.
15 x1=10. ; // extremidade superior.
16 Nmax=100000; // limite superior para o número de iterações.
17 e1=1e-6; // erro absoluto admissível na solução.
18
19 //inicialização das variáveis
20
21 f0=f(x0);
22 f1=f(x1);
23 xm=x1;
24 contador=1;
25 segue=%T;

Line: 1 Column: 1 Logical line: 1
```

Interfaces

Durante a execução do Scilab, várias janelas podem ser utilizadas. As principais são:

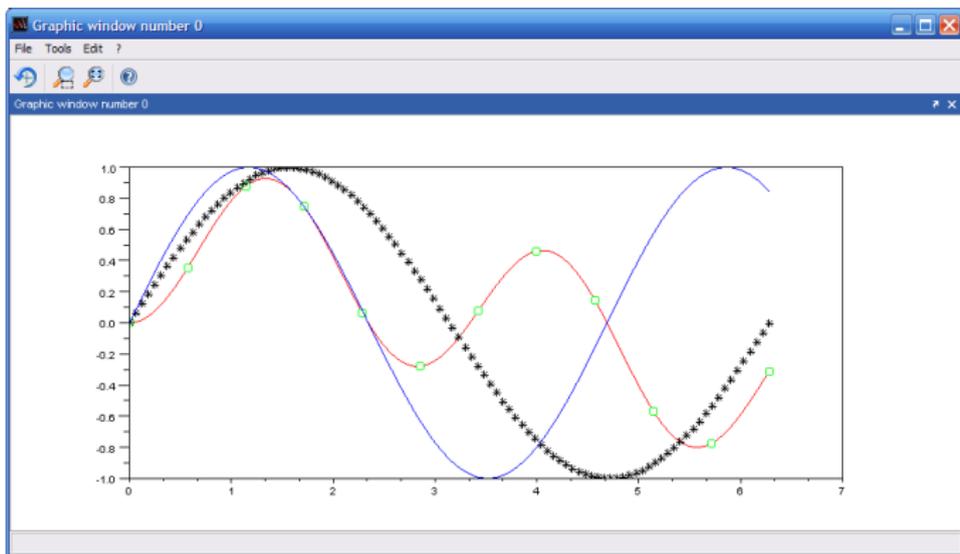
A ajuda on-line



Interfaces

Durante a execução do Scilab, várias janelas podem ser utilizadas. As principais são:

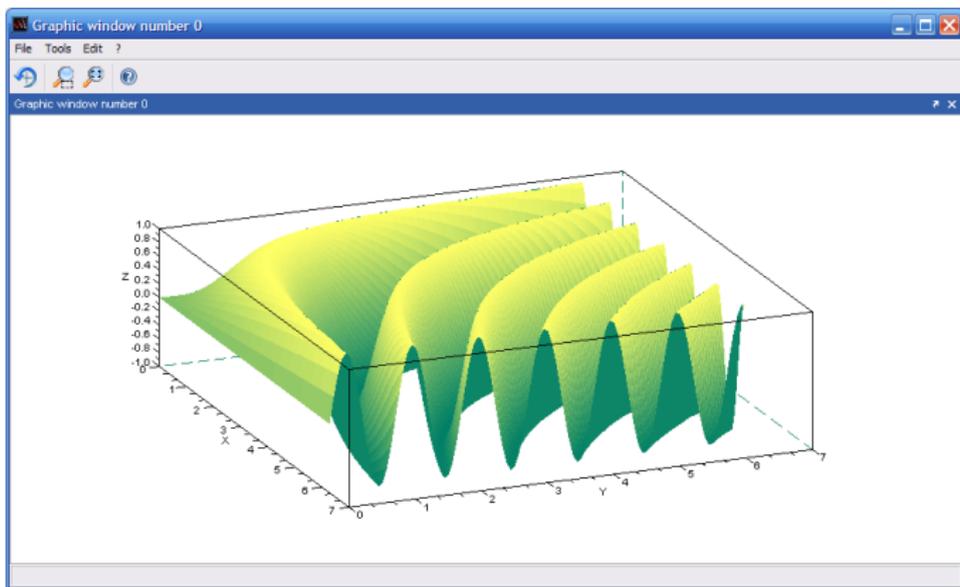
A janela gráfica



Interfaces

Durante a execução do Scilab, várias janelas podem ser utilizadas. As principais são:

A janela gráfica



Interação com o console

Em geral, a interação com o console é da forma:

Observações

- O caractere ";" impede que o resultado de uma operação seja mostrado no console.
- As reticências "." permitem a continuação do comando na linha seguinte (após o pressionamento da tecla "Enter").
- Duas barras invertidas "//" iniciam um comentário, i. e., texto que não será executado.

Interação com o console

Em geral, a interação com o console é da forma:

--> *nome*=*expressão*

nesse caso, as operações contidas no termo *expressão* são realizadas e o resultado é atribuído ao objeto *nome*.

Observações

- O caractere ";" impede que o resultado de uma operação seja mostrado no console.
- As reticências "..." permitem a continuação do comando na linha seguinte (após o pressionamento da tecla "Enter").
- Duas barras invertidas "//" iniciam um comentário, i. e., texto que não será executado.

Interação com o console

Em geral, a interação com o console é da forma:

--> *nome*=*expressão*

nesse caso, as operações contidas no termo *expressão* são realizadas e o resultado é atribuído ao objeto *nome*.

--> *expressão*

nesse caso, as operações contidas no termo *expressão* são realizadas e o resultado (se for um objeto) é atribuído ao objeto *ans*.

Observações

- O caractere ";" impede que o resultado de uma operação seja mostrado no console.
- As reticências "..." permitem a continuação do comando na linha seguinte (após o pressionamento da tecla "Enter").
- Duas barras invertidas "//" iniciam um comentário, i. e., texto que não será executado.

Interação com o console

Em geral, a interação com o console é da forma:

--> *nome*=*expressão*

nesse caso, as operações contidas no termo *expressão* são realizadas e o resultado é atribuído ao objeto *nome*.

--> *expressão*

nesse caso, as operações contidas no termo *expressão* são realizadas e o resultado (se for um objeto) é atribuído ao objeto *ans*.

Observações

- O caractere “;” impede que o resultado de uma operação seja mostrado no console.
- As reticências “...” permitem a continuação do comando na linha seguinte (após o pressionamento da tecla “Enter”).
- Duas barras invertidas “//” iniciam um comentário, i. e., texto que não será executado.

Comandos do console

Alguns comandos:

- `UP` ou `Ctrl+P` chama a entrada anterior.
- `DOWN` or `Ctrl+N` chama a entrada seguinte.
- `F1` chama a janela de ajuda.
- `F2` limpa o console.
- `Ctrl+space` ou `TAB`: o console abre uma lista com todos os nomes que iniciam-se com os caracteres presentes no momento do acionamento do comando.
- `Ctrl+A` ou `HOME` move o cursor para o início da linha.

Comandos do console

Alguns comandos:

- `Ctrl+C` se nenhum texto estiver selecionado, interrompe a execução, caso contrário, o texto selecionado é copiado para a área de transferência (“clipboard”).
- `Ctrl+D` ou `DELETE` apaga o caractere corrente.
- `Ctrl+E` ou `END` move o cursor para o final da linha.
- `Ctrl+H` ou `BACKSPACE` apaga o caractere anterior.
- `Ctrl+K` apaga a linha da posição corrente ao final.
- `Ctrl+S` seleciona tudo.
- `Ctrl+U` apaga toda a linha.

Comandos do console

Alguns comandos:

- **Ctrl+V** insere o conteúdo da área de transferência.
- **Ctrl+W** apaga a última palavra na linha.
- **Ctrl+X** Interrompe o Scilab.
- **Ctrl+LEFT** move o cursor uma palavra à esquerda.
- **Ctrl+RIGHT** move o cursor uma palavra à direita.
- **Shift+HOME** seleciona a partir da posição do cursor até o início da linha.
- **Shift+END** seleciona a partir da posição do cursor até o final da linha.
- Double-click seleciona a palavra apontada pelo mouse.

Tipos de objetos

O Scilab admite internamente uma série de tipos de objetos pré-definidos. Entre eles os que serão mais utilizados por nós são

- Constantes especiais
- Matriz de constantes.
- Matriz de *strings*.
- Matriz de inteiros.
- Matriz de booleanos.
- Matriz de polinômios.
- Funções.

Observação 1

Os objetos são associados a nomes que podem ser formados por até 24 caracteres alfanuméricos não acentuados e a sublinha “_”.

Observação 2

O tipo de um objeto pode ser obtido a partir dos comandos `type` e `typeof`.

Tipos de objetos

O Scilab admite internamente uma série de tipos de objetos pré-definidos. Entre eles os que serão mais utilizados por nós são

- Constantes especiais
- Matriz de constantes.
- Matriz de *strings*.
- Matriz de inteiros.
- Matriz de booleanos.
- Matriz de polinômios.
- Funções.

Observação 1

Os objetos são associados a nomes que podem ser formados por até 24 caracteres alfanuméricos não acentuados e a sublinha “_”.

Observação 2

O tipo de um objeto pode ser obtido a partir dos comandos `type` e `typeof`.

Tipos de objetos

O Scilab admite internamente uma série de tipos de objetos pré-definidos. Entre eles os que serão mais utilizados por nós são

- Constantes especiais
- Matriz de constantes.
- Matriz de *strings*.
- Matriz de inteiros.
- Matriz de booleanos.
- Matriz de polinômios.
- Funções.

Observação 1

Os objetos são associados a nomes que podem ser formados por até 24 caracteres alfanuméricos não acentuados e a sublinha “_”.

Observação 2

O tipo de um objeto pode ser obtido a partir dos comandos `type` e `typeof`.

Constantes especiais

São objetos pré-definidos. Não podem ser apagados ou redefinidos.

- `%i` representa $\sqrt{-1}$.
- `%e` representa a constante de Euler.
- `%pi` representa a constante π .
- `%s` representa o polinômio de 1º grau s .
- `%z` representa o polinômio de 1º grau z .
- `%eps` é uma medida de precisão da máquina.
- `%inf` representa ∞ .
- `%nan` representa o objeto “Not a Number”.
- `%t` (ou `%T`) representa a variável booleana “verdadeiro”.
- `%f` (ou `%F`) representa a variável booleana “falso”.

Matrizes de constantes

- São objetos que contém matrizes cujos coeficientes são números complexos, representados com precisão finita através de registros de ponto flutuante de 64bits, os *doubles*.
- Os escalares também fazem parte dessa classe. São considerados como matrizes 1×1 .
- Assim, $2 + i$, $(-123)_{1 \times 3}$ e $\begin{pmatrix} 1 & \pi \\ 0 & i \end{pmatrix}_{2 \times 2}$ são representados por objetos de um mesmo tipo.
- Por que? Resposta: O Scilab é otimizado para trabalhar com matrizes.

Matrizes de constantes

- São objetos que contém matrizes cujos coeficientes são números complexos, representados com precisão finita através de registros de ponto flutuante de 64bits, os *doubles*.
- Os escalares também fazem parte dessa classe. São considerados como matrizes 1×1 .
- Assim, $2 + i$, $(-123)_{1 \times 3}$ e $\begin{pmatrix} 1 & \pi \\ 0 & i \end{pmatrix}_{2 \times 2}$ são representados por objetos de um mesmo tipo.
- Por que? Resposta: O Scilab é otimizado para trabalhar com matrizes.

Matrizes de constantes

- São objetos que contém matrizes cujos coeficientes são números complexos, representados com precisão finita através de registros de ponto flutuante de 64bits, os *doubles*.
- Os escalares também fazem parte dessa classe. São considerados como matrizes 1×1 .
- Assim, $2 + i$, $(-123)_{1 \times 3}$ e $\begin{pmatrix} 1 & \pi \\ 0 & i \end{pmatrix}_{2 \times 2}$ são representados por objetos de um mesmo tipo.
- Por que? Resposta: O Scilab é otimizado para trabalhar com matrizes.

Matrizes de constantes

- São objetos que contém matrizes cujos coeficientes são números complexos, representados com precisão finita através de registros de ponto flutuante de 64bits, os *doubles*.
- Os escalares também fazem parte dessa classe. São considerados como matrizes 1×1 .
- Assim, $2 + i$, $(-123)_{1 \times 3}$ e $\begin{pmatrix} 1 & \pi \\ 0 & i \end{pmatrix}_{2 \times 2}$ são representados por objetos de um mesmo tipo.
- Por que? Resposta: O Scilab é otimizado para trabalhar com matrizes.

Matrizes de constantes

- São objetos que contém matrizes cujos coeficientes são números complexos, representados com precisão finita através de registros de ponto flutuante de 64bits, os *doubles*.
- Os escalares também fazem parte dessa classe. São considerados como matrizes 1×1 .
- Assim, $2 + i$, $(-123)_{1 \times 3}$ e $\begin{pmatrix} 1 & \pi \\ 0 & i \end{pmatrix}_{2 \times 2}$ são representados por objetos de um mesmo tipo.
- Por que? Resposta: O Scilab é otimizado para trabalhar com matrizes.

Matrizes: criação e manipulação

Criação

- Atribuição de um escalar: `var1=2+%i`.
- Atribuição de um vetor : `var1=[-1,2,3]` ou `var1=[-1 2 3]`.
- Atribuição de uma matriz: `var1=[1,%pi;0,%i]` ou `var1=[1 %pi;
0 %i]`.

Matrizes: criação e manipulação

Criação

- Atribuição de um escalar: `var1=2+%i`.
- Atribuição de um vetor : `var1=[-1,2,3]` ou `var1=[-1 2 3]`.
- Atribuição de uma matriz: `var1=[1,%pi;0,%i]` ou `var1=[1 %pi; 0 %i]`.

Observação

O comando `p:q:r` cria uma matriz $1 \times n$ com o primeiro elemento igual a p , o último igual a r e os demais com espaçamento q . O comando `p:r` retorna a matriz linha com elementos $p, p+1, \dots, r$.

Matrizes: criação e manipulação

Criação

- Atribuição de um escalar: `var1=2+%i`.
- Atribuição de um vetor : `var1=[-1,2,3]` ou `var1=[-1 2 3]`.
- Atribuição de uma matriz: `var1=[1,%pi;0,%i]` ou `var1=[1 %pi; 0 %i]`.

Observação

O comando `p:q:r` cria uma matriz $1 \times n$ com o primeiro elemento igual a p , o último igual a r e os demais com espaçamento q . O comando `p:r` retorna a matriz linha com elementos $p, p+1, \dots, r$.

- `var1=1:9` \longrightarrow `var1=[1 2 3 4 5 6 7 8 9]`
- `var1=1:2:9` \longrightarrow `var1=[1 3 5 7 9]`
- `var1=1:3:9` \longrightarrow `var1=[1 4 7]`

Matrizes: criação e manipulação

Manipulação (através de exemplos)

Vamos considerar a matriz $A=[1\ 2\ 3;4\ 5\ 6;7\ 8\ 9]$.

- O elemento (2,3) dessa matriz é referenciado como $A(2,3)$. Assim, o comando $A(2,3)=0$ retorna a matriz $[1\ 2\ 3;4\ 5\ 0;7\ 8\ 9]$.
- A segunda linha da matriz A é referenciada como $A(2,:)$. Assim, o comando $A(2,:)=4:6$ ou $A(2,:)= [4\ 5\ 6]$ retorna a matriz original.
- O último elemento da primeira coluna é referenciado como $A(\$,1)$. Assim, o comando $A(\$,1)=0$ retorna a matriz $[1\ 2\ 3;4\ 5\ 6;0\ 8\ 9]$.
- O comando $A(1:2, \$-1)$ retorna a matriz $[2;5]$. O comando $A(2:3, \$)$ retorna a matriz $[6;9]$.
- A submatriz $[A(3,1)\ A(3,2);A(2,1)\ A(2,2)]$ é obtida com o comando $A([3\ 2], [1\ 2])$. Assim, o comando $A([3\ 2], [1\ 2])$ retorna a matriz $[0\ 8;4\ 5]$.

Matrizes: criação e manipulação

Manipulação (através de exemplos)

Vamos considerar a matriz $A=[1\ 2\ 3;4\ 5\ 6;7\ 8\ 9]$.

- O elemento (2,3) dessa matriz é referenciado como $A(2,3)$. Assim, o comando $A(2,3)=0$ retorna a matriz $[1\ 2\ 3;4\ 5\ 0;7\ 8\ 9]$.
- A segunda linha da matriz A é referenciada como $A(2,:)$. Assim, o comando $A(2,:)=4:6$ ou $A(2,:)= [4\ 5\ 6]$ retorna a matriz original.
- O último elemento da primeira coluna é referenciado como $A(\$,1)$. Assim, o comando $A(\$,1)=0$ retorna a matriz $[1\ 2\ 3;4\ 5\ 6;0\ 8\ 9]$.
- O comando $A(1:2, \$-1)$ retorna a matriz $[2;5]$. O comando $A(2:3, \$)$ retorna a matriz $[6;9]$.
- A submatriz $[A(3,1)\ A(3,2);A(2,1)\ A(2,2)]$ é obtida com o comando $A([3\ 2], [1\ 2])$. Assim, o comando $A([3\ 2], [1\ 2])$ retorna a matriz $[0\ 8;4\ 5]$.

Matrizes: criação e manipulação

Manipulação (através de exemplos)

Vamos considerar a matriz $A=[1\ 2\ 3;4\ 5\ 6;7\ 8\ 9]$.

- O elemento (2,3) dessa matriz é referenciado como $A(2,3)$. Assim, o comando $A(2,3)=0$ retorna a matriz $[1\ 2\ 3;4\ 5\ 0;7\ 8\ 9]$.
- A segunda linha da matriz A é referenciada como $A(2,:)$. Assim, o comando $A(2,:)=4:6$ ou $A(2,:)= [4\ 5\ 6]$ retorna a matriz original.
- O último elemento da primeira coluna é referenciado como $A(\$,1)$. Assim, o comando $A(\$,1)=0$ retorna a matriz $[1\ 2\ 3;4\ 5\ 6;0\ 8\ 9]$.
- O comando $A(1:2, \$-1)$ retorna a matriz $[2;5]$. O comando $A(2:3, \$)$ retorna a matriz $[6;9]$.
- A submatriz $[A(3,1)\ A(3,2);A(2,1)\ A(2,2)]$ é obtida com o comando $A([3\ 2], [1\ 2])$. Assim, o comando $A([3\ 2], [1\ 2])$ retorna a matriz $[0\ 8;4\ 5]$.

Matrizes: criação e manipulação

Manipulação (através de exemplos)

Vamos considerar a matriz $A=[1\ 2\ 3;4\ 5\ 6;7\ 8\ 9]$.

- O elemento (2,3) dessa matriz é referenciado como $A(2,3)$. Assim, o comando $A(2,3)=0$ retorna a matriz $[1\ 2\ 3;4\ 5\ 0;7\ 8\ 9]$.
- A segunda linha da matriz A é referenciada como $A(2,:)$. Assim, o comando $A(2,:)=4:6$ ou $A(2,:)= [4\ 5\ 6]$ retorna a matriz original.
- O último elemento da primeira coluna é referenciado como $A(\$,1)$. Assim, o comando $A(\$,1)=0$ retorna a matriz $[1\ 2\ 3;4\ 5\ 6;0\ 8\ 9]$.
- O comando $A(1:2, \$-1)$ retorna a matriz $[2;5]$. O comando $A(2:3, \$)$ retorna a matriz $[6;9]$.
- A submatriz $[A(3,1)\ A(3,2);A(2,1)\ A(2,2)]$ é obtida com o comando $A([3\ 2], [1\ 2])$. Assim, o comando $A([3\ 2], [1\ 2])$ retorna a matriz $[0\ 8;4\ 5]$.

Matrizes: criação e manipulação

Manipulação (através de exemplos)

Vamos considerar a matriz $A=[1\ 2\ 3;4\ 5\ 6;7\ 8\ 9]$.

- O elemento (2,3) dessa matriz é referenciado como $A(2,3)$. Assim, o comando $A(2,3)=0$ retorna a matriz $[1\ 2\ 3;4\ 5\ 0;7\ 8\ 9]$.
- A segunda linha da matriz A é referenciada como $A(2,:)$. Assim, o comando $A(2,:)=4:6$ ou $A(2,:)= [4\ 5\ 6]$ retorna a matriz original.
- O último elemento da primeira coluna é referenciado como $A(\$,1)$. Assim, o comando $A(\$,1)=0$ retorna a matriz $[1\ 2\ 3;4\ 5\ 6;0\ 8\ 9]$.
- O comando $A(1:2, \$-1)$ retorna a matriz $[2;5]$. O comando $A(2:3, \$)$ retorna a matriz $[6;9]$.
- A submatriz $[A(3,1)\ A(3,2);A(2,1)\ A(2,2)]$ é obtida com o comando $A([3\ 2], [1\ 2])$. Assim, o comando $A([3\ 2], [1\ 2])$ retorna a matriz $[0\ 8;4\ 5]$.

Matrizes: criação e manipulação

Manipulação (através de exemplos)

Vamos considerar a matriz $A=[1\ 2\ 3;4\ 5\ 6;7\ 8\ 9]$.

- O elemento (2,3) dessa matriz é referenciado como $A(2,3)$. Assim, o comando $A(2,3)=0$ retorna a matriz $[1\ 2\ 3;4\ 5\ 0;7\ 8\ 9]$.
- A segunda linha da matriz A é referenciada como $A(2,:)$. Assim, o comando $A(2,:)=4:6$ ou $A(2,:)= [4\ 5\ 6]$ retorna a matriz original.
- O último elemento da primeira coluna é referenciado como $A(\$,1)$. Assim, o comando $A(\$,1)=0$ retorna a matriz $[1\ 2\ 3;4\ 5\ 6;0\ 8\ 9]$.
- O comando $A(1:2, \$-1)$ retorna a matriz $[2;5]$. O comando $A(2:3, \$)$ retorna a matriz $[6;9]$.
- A submatriz $[A(3,1)\ A(3,2);A(2,1)\ A(2,2)]$ é obtida com o comando $A([3\ 2], [1\ 2])$. Assim, o comando $A([3\ 2], [1\ 2])$ retorna a matriz $[0\ 8;4\ 5]$.

Matrizes de constantes – operações elementares

- Operações elementares ($+$ $-$ $*$ $/$) sobre escalares, operação $*$ envolvendo escalares e matrizes, operações entre matrizes ($+$ $-$ $*$). Exemplos:
 - $3 * [1 \ 1; 0 \ 2] \rightarrow [3 \ 3; 0 \ 6]$.
 - $[1 \ 1; 0 \ 2] - [1 \ 0; 0 \ 1] \rightarrow [0 \ 1; 0 \ 1]$.
 - $[1 \ 1; 0 \ 1] * [2; 3] \rightarrow [5; 3]$.
 - $[2; 3] * [1 \ 1; 0 \ 1] \rightarrow$ erro: multiplicação inconsistente! \

Matrizes de constantes – operações elementares

- Operações elementares ($\boxed{+}$ $\boxed{-}$ $\boxed{*}$ $\boxed{/}$) sobre escalares, operação $\boxed{*}$ envolvendo escalares e matrizes, operações entre matrizes ($\boxed{+}$ $\boxed{-}$ $\boxed{*}$). Exemplos:
 - $3 * [1 \ 1; 0 \ 2] \rightarrow [3 \ 3; 0 \ 6]$.
 - $[1 \ 1; 0 \ 2] - [1 \ 0; 0 \ 1] \rightarrow [0 \ 1; 0 \ 1]$.
 - $[1 \ 1; 0 \ 1] * [2; 3] \rightarrow [5; 3]$.
 - $[2; 3] * [1 \ 1; 0 \ 1] \rightarrow$ erro: multiplicação inconsistente! \

Novidades:

- operações $\boxed{+}$ e $\boxed{-}$ envolvendo escalares e matrizes:
 - $1 + [1 \ 0; 0 \ 1] \rightarrow [2 \ 1; 1 \ 2]$. O termo 1 é somado aos elementos da matriz.
 - $[1 \ 0; 0 \ 1] - 2 \rightarrow [-1 \ -2; -2 \ -1]$.

Matrizes de constantes – operações elementares

Novidades:

- operação $\boxed{/}$ envolvendo matrizes (e escalares). A operação A/B retorna a solução da equação $x*B=A$:
 - $[1\ 2\ 3]/[1\ 1\ 1;0\ 2\ 5;7\ 6\ 1] \rightarrow [0.8421053\ 0.4210526\ 0.0526316]$
 - $[1\ 2;3\ 4]/2$ equivale a $[1\ 2;3\ 4]/[2\ 0;0\ 2] \rightarrow [0.5\ 1;1.5\ 2]$.
 - $2/[1\ 2;3\ 4]$ equivale a $[2\ 0;0\ 2]/[1\ 2;3\ 4] \rightarrow [-4\ 2;3\ -1]$
- operação $\boxed{./}$ envolvendo matrizes de mesma dimensão. A operação $A./B$ retorna uma matriz com elementos da forma a_{ij}/b_{ij} :
 - $[2\ 4\ 6;3\ 6\ 9]./[1\ 2\ 3;1\ 2\ 3] \rightarrow [2\ 2\ 2;3\ 3\ 3]$
- operação $\boxed{.*}$ envolvendo matrizes de mesma dimensão. A operação $A.*B$ retorna uma matriz com elementos da forma $a_{ij}b_{ij}$:
 - $[2\ 2\ 2;3\ 3\ 3].*[1\ 2\ 3;1\ 2\ 3] \rightarrow [2\ 4\ 6;3\ 6\ 9]$

Matrizes de constantes – operações elementares

Novidades:

- operação $\boxed{/}$ envolvendo matrizes (e escalares). A operação A/B retorna a solução da equação $x*B=A$:
 - $[1\ 2\ 3]/[1\ 1\ 1;0\ 2\ 5;7\ 6\ 1] \rightarrow [0.8421053\ 0.4210526\ 0.0526316]$
 - $[1\ 2;3\ 4]/2$ equivale a $[1\ 2;3\ 4]/[2\ 0;0\ 2] \rightarrow [0.5\ 1;1.5\ 2]$.
 - $2/[1\ 2;3\ 4]$ equivale a $[2\ 0;0\ 2]/[1\ 2;3\ 4] \rightarrow [-4\ 2;3\ -1]$
- operação $\boxed{./}$ envolvendo matrizes de mesma dimensão. A operação $A./B$ retorna uma matriz com elementos da forma $a_{i,j}/b_{ij}$:
 - $[2\ 4\ 6;3\ 6\ 9]./[1\ 2\ 3;1\ 2\ 3] \rightarrow [2\ 2\ 2;3\ 3\ 3]$
- operação $\boxed{.*}$ envolvendo matrizes de mesma dimensão. A operação $A.*B$ retorna uma matriz com elementos da forma $a_{i,j}b_{ij}$:
 - $[2\ 2\ 2;3\ 3\ 3].*[1\ 2\ 3;1\ 2\ 3] \rightarrow [2\ 4\ 6;3\ 6\ 9]$

Matrizes de constantes – operações elementares

Novidades:

- operação $\boxed{/}$ envolvendo matrizes (e escalares). A operação A/B retorna a solução da equação $x*B=A$:
 - $[1\ 2\ 3]/[1\ 1\ 1;0\ 2\ 5;7\ 6\ 1] \rightarrow [0.8421053\ 0.4210526\ 0.0526316]$
 - $[1\ 2;3\ 4]/2$ equivale a $[1\ 2;3\ 4]/[2\ 0;0\ 2] \rightarrow [0.5\ 1;1.5\ 2]$.
 - $2/[1\ 2;3\ 4]$ equivale a $[2\ 0;0\ 2]/[1\ 2;3\ 4] \rightarrow [-4\ 2;3\ -1]$
- operação $\boxed{./}$ envolvendo matrizes de mesma dimensão. A operação $A./B$ retorna uma matriz com elementos da forma $a_{i,j}/b_{ij}$:
 - $[2\ 4\ 6;3\ 6\ 9]./[1\ 2\ 3;1\ 2\ 3] \rightarrow [2\ 2\ 2;3\ 3\ 3]$
- operação $\boxed{.*}$ envolvendo matrizes de mesma dimensão. A operação $A.*B$ retorna uma matriz com elementos da forma $a_{i,j}b_{ij}$:
 - $[2\ 2\ 2;3\ 3\ 3].*[1\ 2\ 3;1\ 2\ 3] \rightarrow [2\ 4\ 6;3\ 6\ 9]$

Matrizes de constantes – operações elementares

Novidades:

- operação \backslash envolvendo matrizes (e escalares). A operação $A \backslash B$ retorna a solução da equação $A * x = B$.
- operação \wedge envolvendo uma matriz A e um escalar b . A operação $A \wedge b$ retorna a potencia b da matriz A se ela for uma matriz quadrada. A operação $b \wedge A$ retorna uma matriz com as mesmas dimensões de A , cujos elementos são da forma $b^{a_{ij}}$.
- operação \wedge envolvendo uma matriz A e um escalar b . A operação $A . \wedge b$ retorna uma matriz com as mesmas dimensões de A , cujos elementos são da forma a_{ij}^b .
- operação \wedge envolvendo uma matrizes A e B de mesma dimensão. A operação $A . \wedge B$ retorna uma matriz com as mesmas dimensões de A e B , cujos elementos são da forma $a_{ij}^{b_{ij}}$.

Matrizes de constantes – operações elementares

Novidades:

- operação \backslash envolvendo matrizes (e escalares). A operação $A \backslash B$ retorna a solução da equação $A * x = B$.
- operação \wedge envolvendo uma matriz A e um escalar b . A operação $A \wedge b$ retorna a potencia b da matriz A se ela for uma matriz quadrada. A operação $b \wedge A$ retorna uma matriz com as mesmas dimensões de A , cujos elementos são da forma $b^{a_{ij}}$.
- operação \wedge envolvendo uma matriz A e um escalar b . A operação $A \wedge b$ retorna uma matriz com as mesmas dimensões de A , cujos elementos são da forma a_{ij}^b .
- operação \wedge envolvendo uma matrizes A e B de mesma dimensão. A operação $A \wedge B$ retorna uma matriz com as mesmas dimensões de A e B , cujos elementos são da forma $a_{ij}^{b_{ij}}$.

Matrizes de constantes – operações elementares

Novidades:

- operação \backslash envolvendo matrizes (e escalares). A operação $A \backslash B$ retorna a solução da equação $A * x = B$.
- operação \wedge envolvendo uma matriz A e um escalar b . A operação $A \wedge b$ retorna a potencia b da matriz A se ela for uma matriz quadrada. A operação $b \wedge A$ retorna uma matriz com as mesmas dimensões de A , cujos elementos são da forma $b^{a_{ij}}$.
- operação \wedge envolvendo uma matriz A e um escalar b . A operação $A . \wedge b$ retorna uma matriz com as mesmas dimensões de A , cujos elementos são da forma a_{ij}^b .
- operação \wedge envolvendo uma matrizes A e B de mesma dimensão. A operação $A . \wedge B$ retorna uma matriz com as mesmas dimensões de A e B , cujos elementos são da forma $a_{ij}^{b_{ij}}$.

Matrizes de constantes – operações elementares

Novidades:

- operação \backslash envolvendo matrizes (e escalares). A operação $A \backslash B$ retorna a solução da equação $A * x = B$.
- operação \wedge envolvendo uma matriz A e um escalar b . A operação $A \wedge b$ retorna a potencia b da matriz A se ela for uma matriz quadrada. A operação $b \wedge A$ retorna uma matriz com as mesmas dimensões de A , cujos elementos são da forma $b^{a_{ij}}$.
- operação \wedge envolvendo uma matriz A e um escalar b . A operação $A \wedge b$ retorna uma matriz com as mesmas dimensões de A , cujos elementos são da forma a_{ij}^b .
- operação \wedge envolvendo uma matrizes A e B de mesma dimensão. A operação $A \wedge B$ retorna uma matriz com as mesmas dimensões de A e B , cujos elementos são da forma $a_{ij}^{b_{ij}}$.

Matrizes de constantes – operações elementares

Novidades:

- operação de transposição `'` de matrizes. A operação `A'` retorna a transposta de `A`.
- a remoção de elementos de uma matriz é realizada utilizando o objeto `[]`:
 - Seja `A=[1 2 3;0 1 0]`. Então o comando `A(:,2)=[]` retorna a matriz `[1,3;0,0]`.
- a função `size` retorna as dimensões de uma matriz.
- a função `zeros` retorna matrizes com elementos nulos.
- a função `ones` retorna matrizes com elementos iguais a 1.
- a função `eye` retorna matrizes identidade.
- a função `diag` retorna matrizes com diagonal definida. Também as remove.

Matrizes de constantes – operações elementares

Novidades:

- operação de transposição `'` de matrizes. A operação `A'` retorna a transposta de `A`.
- a remoção de elementos de uma matriz é realizada utilizando o objeto `[]`:
 - Seja `A=[1 2 3;0 1 0]`. Então o comando `A(:,2)=[]` retorna a matriz `[1,3;0,0]`.
- a função `size` retorna as dimensões de uma matriz.
- a função `zeros` retorna matrizes com elementos nulos.
- a função `ones` retorna matrizes com elementos iguais a 1.
- a função `eye` retorna matrizes identidade.
- a função `diag` retorna matrizes com diagonal definida. Também as remove.

Matrizes de constantes – operações elementares

Novidades:

- operação de transposição `'` de matrizes. A operação `A'` retorna a transposta de `A`.
- a remoção de elementos de uma matriz é realizada utilizando o objeto `[]`:
 - Seja `A=[1 2 3;0 1 0]`. Então o comando `A(:,2)=[]` retorna a matriz `[1,3;0,0]`.
- a função `size` retorna as dimensões de uma matriz.
- a função `zeros` retorna matrizes com elementos nulos.
- a função `ones` retorna matrizes com elementos iguais a 1.
- a função `eye` retorna matrizes identidade.
- a função `diag` retorna matrizes com diagonal definida. Também as remove.

Matriz de constantes – funções pré-definidas

A lista das funções elementares e especiais e os respectivos manuais de uso estão disponíveis nos capítulos XXIV e XXV do manual do Scilab (versão 5.1.1 em português). Vamos apenas chamar a atenção para algumas propriedades que nos serão mais úteis.

Várias funções elementares e especiais também agem sobre matrizes de constantes. Em geral, uma função f aceita uma matriz como argumento e retorna uma matriz com mesmas dimensões:

Se $a_{i,j}$ são elementos de uma matriz A , $f(A)$ é uma matriz com elementos $f(a_{i,j})$.

Exemplo:

$A = [0 \ 0.5*\%pi \ \%pi; -\%pi \ -0.5*\%pi \ 0]$, então $\sin(A)$ retorna a matriz
 $[0 \ 1 \ 0; 0 \ -1 \ 0]$.

Matrizes de strings

O objeto strings consiste em uma sequência de caracteres delimitados por apóstrofos `'` ou aspas `"`. Exemplos:

- `var1='a'`.
- `var2="Olá mundo!"`.
- `var3="Nº 2, custo: £4s1d3"`.

Matrizes de strings

O objeto strings consiste em uma sequência de caracteres delimitados por apóstrofos `'` ou aspas `"`. Exemplos:

- `var1='a'`.
- `var2="Olá mundo!"`.
- `var3="Nº 2, custo: £4s1d3"`.

Também é possível criar e manipular matrizes de strings. Exemplos:

- `var=['A' 'raposa';'pulou' 'a cerca'];`
- `var(1,:) → A raposa`
- `var(1,)= ['A' 'ovelha'] → ['A' 'ovelha';'pulou' 'a cerca']`

Matrizes de strings

Os objetos do tipo string admitem a operação de concatenação, implementada através do símbolo `+`. Exemplos:

- `'Parte 1 '+'e parte 2'` → `'Parte 1 e parte2'`.
- `var+['' ' branca';'' ' de madeira']` → `['A' 'ovelha branca';'pulou' 'a cerca de madeira']`.

Matrizes de strings

Os objetos do tipo string admitem a operação de concatenação, implementada através do símbolo `+`. Exemplos:

- `'Parte 1 '+'e parte 2'` \longrightarrow `'Parte 1 e parte2'`.
- `var+['' ' branca';'' ' de madeira']` \longrightarrow `['A' 'ovelha branca';'pulou' 'a cerca de madeira']`.

A função `length` fornece o número de caracteres em um string. Exemplos:

- `length('')` \longrightarrow 0.
- `length('abc')` \longrightarrow 3.
- `length('a b c')` \longrightarrow 5.
- `length(var)` \longrightarrow `[1 6;5 7]`.

Matrizes de strings

Os objetos do tipo string admitem a operação de concatenação, implementada através do símbolo `+`. Exemplos:

- `'Parte 1 '+'e parte 2' → 'Parte 1 e parte2'`.
- `var+['' ' branca';'' ' de madeira'] → ['A' 'ovelha branca';'pulou' 'a cerca de madeira']`.

A função `length` fornece o número de caracteres em um string. Exemplos:

- `length('')` → 0.
- `length('abc')` → 3.
- `length('a b c')` → 5.
- `length(var)` → [1 6;5 7].

Demais funções sobre strings estão no cap. IX do manual.

Matrizes de inteiros

Os números inteiros (assim como as matrizes) dividem-se em 6 tipos:

- inteiros de 1 byte.
- inteiros de 2 bytes.
- inteiros de 4 bytes.
- inteiros de 1 byte sem sinal.
- inteiros de 2 bytes sem sinal.
- inteiros de 4 bytes sem sinal.

As funções `int8`, `int16`, `int32`, `uint8`, `uint16` e `uint32` realizam a conversão de um objeto do tipo matriz de constantes em um dos subtipos de matriz de inteiros acima. A conversão de um desses objetos em uma matriz de constantes é realizada pelo comando `double`.

Matrizes de inteiros

Os números inteiros (assim como as matrizes) dividem-se em 6 tipos:

- inteiros de 1 byte.
- inteiros de 2 bytes.
- inteiros de 4 bytes.
- inteiros de 1 byte sem sinal.
- inteiros de 2 bytes sem sinal.
- inteiros de 4 bytes sem sinal.

As funções `int8`, `int16`, `int32`, `uint8`, `uint16` e `uint32` realizam a conversão de um objeto do tipo matriz de constantes em um dos subtipos de matriz de inteiros acima. A conversão de um desses objetos em uma matriz de constantes é realizada pelo comando `double`.

Exemplos:

- `a=uint8(11);b=11;`
`b/2` → 5.5 porém `a/2` → 5.
`typeof(a/2)` → `uint8`.
- **Cuidado:** `c=uint8(255); c+1` → 0 !

Matrizes de inteiros

Os números inteiros (assim como as matrizes) dividem-se em 6 tipos:

- inteiros de 1 byte.
- inteiros de 2 bytes.
- inteiros de 4 bytes.
- inteiros de 1 byte sem sinal.
- inteiros de 2 bytes sem sinal.
- inteiros de 4 bytes sem sinal.

As funções `int8`, `int16`, `int32`, `uint8`, `uint16` e `uint32` realizam a conversão de um objeto do tipo matriz de constantes em um dos subtipos de matriz de inteiros acima. A conversão de um desses objetos em uma matriz de constantes é realizada pelo comando `double`.

Exemplos:

- `a=uint8(11);b=11;`
`b/2` → 5.5 porém `a/2` → 5.
`typeof(a/2)` → `uint8`.
- **Cuidado:** `c=uint8(255); c+1` → 0 !

Matrizes de inteiros

Os números inteiros (assim como as matrizes) dividem-se em 6 tipos:

- inteiros de 1 byte.
- inteiros de 2 bytes.
- inteiros de 4 bytes.
- inteiros de 1 byte sem sinal.
- inteiros de 2 bytes sem sinal.
- inteiros de 4 bytes sem sinal.

As funções `int8`, `int16`, `int32`, `uint8`, `uint16` e `uint32` realizam a conversão de um objeto do tipo matriz de constantes em um dos subtipos de matriz de inteiros acima. A conversão de um desses objetos em uma matriz de constantes é realizada pelo comando `double`.

Exemplos:

- `a=uint8(11);b=11;`
`b/2` → 5.5 porém `a/2` → 5.
`typeof(a/2)` → `uint8`.
- **Cuidado:** `c=uint8(255); c+1` → 0 !

Demais funções nas seções XXIV e XXXII do manual.

Matrizes de booleanos

Matrizes formadas por objetos %t e %f. São criadas e manipuladas como as demais matrizes.

Matrizes de booleanos

Matrizes formadas por objetos %t e %f. São criadas e manipuladas como as demais matrizes.

Sobre as matrizes booleanas agem os operadores lógicos $\&$ (e), \mid (ou) e \sim (não)

Matrizes de booleanos

Matrizes formadas por objetos %t e %f. São criadas e manipuladas como as demais matrizes.

Sobre as matrizes booleanas agem os operadores lógicos `&` (e), `|` (ou) e `~` (não). Exemplos:

- `~ [%t %f; %f %t] → [F T; T F]`.
- `[%t %t %f %f] & [%f %t %f %f] → [F T F F]`.
- `[%t %t %f %f] | [%f %t %f %f] → [T T F F]`.
- `~~%t → T`.

Matrizes de booleanos

Matrizes formadas por objetos %t e %f. São criadas e manipuladas como as demais matrizes.

Sobre as matrizes booleanas agem os operadores lógicos `&` (e), `|` (ou) e `~` (não). Exemplos:

- `~ [%t %f; %f %t] → [F T; T F]`.
- `[%t %t %f %f] & [%f %t %f %f] → [F T F F]`.
- `[%t %t %f %f] | [%f %t %f %f] → [T T F F]`.
- `~~%t → T`.

As operações de comparação `>`, `<`, `>=`, `<=`, `==`, e `~=` ou `<>` retornam uma matriz de booleanos quando utilizada corretamente com os demais objetos.

Matrizes de booleanos

Matrizes formadas por objetos %t e %f. São criadas e manipuladas como as demais matrizes.

Sobre as matrizes booleanas agem os operadores lógicos `&` (e), `|` (ou) e `~` (não). Exemplos:

- `~ [%t %f;%f %t] → [F T;T F]`.
- `[%t %t %f %f]&[%f %t %f %f] → [F T F F]`.
- `[%t %t %f %f]|[%f %t %f %f] → [T T F F]`.
- `~~%t → T`.

As operações de comparação `>`, `<`, `>=`, `<=`, `==`, e `~=` ou `<>` retornam uma matriz de booleanos quando utilizada corretamente com os demais objetos.

Exemplos:

- `[1 2;3 4]>=zeros(2,2) → [T T;T T]`.
- `%s^2==%s*%s → T`.
- `'casa'==' casa' → F`.

Matrizes de booleanos

Matrizes formadas por objetos %t e %f. São criadas e manipuladas como as demais matrizes.

Sobre as matrizes booleanas agem os operadores lógicos `&` (e), `|` (ou) e `~` (não). Exemplos:

- `~ [%t %f;%f %t] → [F T;T F]`.
- `[%t %t %f %f]&[%f %t %f %f] → [F T F F]`.
- `[%t %t %f %f]|[%f %t %f %f] → [T T F F]`.
- `~~%t → T`.

As operações de comparação `>`, `<`, `>=`, `<=`, `==`, e `~=` ou `<>` retornam uma matriz de booleanos quando utilizada corretamente com os demais objetos.

Exemplos:

- `[1 2;3 4]>=zeros(2,2) → [T T;T T]`.
- `%s^2==%s*%s → T`.
- `'casa'==' casa' → F`.

Demais funções nas seções VII e XXIV do manual.

Matrizes de polinômios

Os polinômios podem ser diretamente construídos através das constantes especiais %s e %z. Porém, na maioria das vezes é mais conveniente utilizar a função `poly`.

A função `poly` aceita como argumentos as raízes do polinômio ou então os coeficientes do mesmo.

Matrizes de polinômios

Os polinômios podem ser diretamente construídos através das constantes especiais %s e %z. Porém, na maioria das vezes é mais conveniente utilizar a função `poly`.

A função `poly` aceita como argumentos as raízes do polinômio ou então os coeficientes do mesmo.

Exemplos:

- Raízes como argumento. `pol1=poly([-1 -1], 'x')` →
`pol1=1+2x+x2`.
- Coeficientes como argumento. `pol2=poly([1 2 1], 'y', 'c')`
→ `pol1=1+2y+y2`.

Matrizes de polinômios

Os polinômios podem ser diretamente construídos através das constantes especiais %s e %z. Porém, na maioria das vezes é mais conveniente utilizar a função `poly`.

A função `poly` aceita como argumentos as raízes do polinômio ou então os coeficientes do mesmo.

Exemplos:

- Raízes como argumento. `pol1=poly([-1 -1], 'x')` → `pol1=1+2x+x2`.
- Coeficientes como argumento. `pol2=poly([1 2 1], 'y', 'c')` → `pol1=1+2y+y2`.

Se o primeiro argumento for uma matriz, então `poly` retorna o seu polinômio característico.

- `pol3=poly([0 -1; -1 0], 'x')` → `pol3=-1+x2`.

Matrizes de polinômios

Os polinômios podem ser diretamente construídos através das constantes especiais %s e %z. Porém, na maioria das vezes é mais conveniente utilizar a função `poly`.

A função `poly` aceita como argumentos as raízes do polinômio ou então os coeficientes do mesmo.

Exemplos:

- Raízes como argumento. `pol1=poly([-1 -1], 'x')` → `pol1=1+2x+x2`.
- Coeficientes como argumento. `pol2=poly([1 2 1], 'y', 'c')` → `pol1=1+2y+y2`.

Se o primeiro argumento for uma matriz, então `poly` retorna o seu polinômio característico.

- `pol3=poly([0 -1; -1 0], 'x')` → `pol3=-1+x2`.

Observação 1

As operações de adição, subtração, produto e divisão estão definidas, desde que os polinômios estejam em termos da mesma variável.

Matrizes de polinômios

Observação 2

Como os demais objetos, os polinômios também podem ser elementos de matrizes.

Matrizes de polinômios

Observação 2

Como os demais objetos, os polinômios também podem ser elementos de matrizes.

Outras funções úteis: `coeff` e `horner`.

Matrizes de polinômios

Observação 2

Como os demais objetos, os polinômios também podem ser elementos de matrizes.

Outras funções úteis: `coeff` e `horner`.

A função `coeff` recebe um polinômio como argumento e retorna uma matriz de constantes com os coeficientes do polinômio. Exemplo:

- `coeff(pol3) → [-1 0 1]`.

Matrizes de polinômios

Observação 2

Como os demais objetos, os polinômios também podem ser elementos de matrizes.

Outras funções úteis: `coeff` e `horner`.

A função `coeff` recebe um polinômio como argumento e retorna uma matriz de constantes com os coeficientes do polinômio. Exemplo:

- `coeff(pol3)` \longrightarrow `[-1 0 1]`.

A função `horner` retorna o valor que o polinômio assume quando a variável na qual ele está definido assume um valor ou um conjunto de valores expresso por uma matriz.

Exemplo:

- `horner(pol1,0.3)` \longrightarrow `1.69`.
- `horner(pol1,[0.3 0.4;0.5 0.6])` \longrightarrow `[1.69 1.96;2.25 2.56]`.

Matrizes de polinômios

Observação 2

Como os demais objetos, os polinômios também podem ser elementos de matrizes.

Outras funções úteis: `coeff` e `horner`.

A função `coeff` recebe um polinômio como argumento e retorna uma matriz de constantes com os coeficientes do polinômio. Exemplo:

- `coeff(pol3)` \longrightarrow `[-1 0 1]`.

A função `horner` retorna o valor que o polinômio assume quando a variável na qual ele está definido assume um valor ou um conjunto de valores expresso por uma matriz.

Exemplo:

- `horner(pol1,0.3)` \longrightarrow `1.69`.
- `horner(pol1,[0.3 0.4;0.5 0.6])` \longrightarrow `[1.69 1.96;2.25 2.56]`.

Demais funções nas seções I e XLVIII do manual.

Objeto do tipo função

- Funções são coleções de instruções executadas em um ambiente próprio que isola as variáveis nelas contidas das variáveis do ambiente (definidas no console).
- Funções podem ser criadas e executadas de várias formas. Podem depender de vários argumentos (incluindo um número variável deles).
- Possuem estruturas computacionais como estruturas de repetição e estruturas de seleção, e podem ser invocadas recursivamente.
- Funções podem ser argumentos de outras funções e mesmo elementos de uma lista.
- O modo mais prático de criar funções é através de um editor de textos (como o SciPad) mas elas também podem ser criadas diretamente a partir do console.

Objeto do tipo função

- Funções são coleções de instruções executadas em um ambiente próprio que isola as variáveis nelas contidas das variáveis do ambiente (definidas no console).
- Funções podem ser criadas e executadas de várias formas. Podem depender de vários argumentos (incluindo um número variável deles).
- Possuem estruturas computacionais como estruturas de repetição e estruturas de seleção, e podem ser invocadas recursivamente.
- Funções podem ser argumentos de outras funções e mesmo elementos de uma lista.
- O modo mais prático de criar funções é através de um editor de textos (como o SciPad) mas elas também podem ser criadas diretamente a partir do console.

Objeto do tipo função

- Funções são coleções de instruções executadas em um ambiente próprio que isola as variáveis nelas contidas das variáveis do ambiente (definidas no console).
- Funções podem ser criadas e executadas de várias formas. Podem depender de vários argumentos (incluindo um número variável deles).
- Possuem estruturas computacionais como estruturas de repetição e estruturas de seleção, e podem ser invocadas recursivamente.
- Funções podem ser argumentos de outras funções e mesmo elementos de uma lista.
- O modo mais prático de criar funções é através de um editor de textos (como o SciPad) mas elas também podem ser criadas diretamente a partir do console.

Objeto do tipo função

- Funções são coleções de instruções executadas em um ambiente próprio que isola as variáveis nelas contidas das variáveis do ambiente (definidas no console).
- Funções podem ser criadas e executadas de várias formas. Podem depender de vários argumentos (incluindo um número variável deles).
- Possuem estruturas computacionais como estruturas de repetição e estruturas de seleção, e podem ser invocadas recursivamente.
- Funções podem ser argumentos de outras funções e mesmo elementos de uma lista.
- O modo mais prático de criar funções é através de um editor de textos (como o SciPad) mas elas também podem ser criadas diretamente a partir do console.

Objeto do tipo função

- Funções são coleções de instruções executadas em um ambiente próprio que isola as variáveis nelas contidas das variáveis do ambiente (definidas no console).
- Funções podem ser criadas e executadas de várias formas. Podem depender de vários argumentos (incluindo um número variável deles).
- Possuem estruturas computacionais como estruturas de repetição e estruturas de seleção, e podem ser invocadas recursivamente.
- Funções podem ser argumentos de outras funções e mesmo elementos de uma lista.
- O modo mais prático de criar funções é através de um editor de textos (como o SciPad) mas elas também podem ser criadas diretamente a partir do console.

Objeto do tipo função

Declaração de uma função:

- `function [var_saída]=nome_da_função(var_entrada)`
 `< Instruções >`
 `endfunction`

Objeto do tipo função

Declaração de uma função:

- `function [var_saída]=nome_da_função(var_entrada)`
 `< Instruções >`
 `endfunction`

Modo alternativo:

- `deff(' [var_saída]=nome_da_função(var_entrada) ', 'Instruções')`

Objeto do tipo função

Declaração de uma função:

- `function [var_saída]=nome_da_função(var_entrada)`
 `< Instruções >`
 `endfunction`

Modo alternativo:

- `def(' [var_saída]=nome_da_função(var_entrada) ', 'Instruções')`

Exemplos:

- `-->function [y]=newf(x)`
 `--> y=x.^2-sin(%pi*x);`
 `-->endfunction`
- `-->function [y]=newf2(f,x)`
 `--> y=x.^2-f(%pi*x);`
 `-->endfunction`

Objeto do tipo função

Declaração de uma função:

- `function [var_saída]=nome_da_função(var_entrada)`
 `< Instruções >`
 `endfunction`

Modo alternativo:

- `def(' [var_saída]=nome_da_função(var_entrada)', 'Instruções')`

Exemplos:

- `-->function [y]=newf(x)`
 `--> y=x.^2-sin(%pi*x);`
 `-->endfunction`
- `-->function [y]=newf2(f,x)`
 `--> y=x.^2-f(%pi*x);`
 `-->endfunction`

Observação

Se houver mais de uma variável de saída, a função retorna ao console apenas a primeira.

Objeto do tipo função

- A não ser por funções muito simples, é mais prático editar as funções através do Scipad (que pode ser executado diretamente do console através do comando `scipad`).
- Uma função definida através dos comandos `function/endfunction` pode ser compilada através do comando `comp`.
- Uma função armazenada em formato texto em um arquivo pode ser invocada no console através do comando `exec` e `getf`.
- Demais funções nas seções I e XXIII do manual.

Objeto do tipo função

- A não ser por funções muito simples, é mais prático editar as funções através do Scipad (que pode ser executado diretamente do console através do comando `scipad`).
- Uma função definida através dos comandos `function/endfunction` pode ser compilada através do comando `comp`.
- Uma função armazenada em formato texto em um arquivo pode ser invocada no console através do comando `exec` e `getf`.
- Demais funções nas seções I e XXIII do manual.

Objeto do tipo função

- A não ser por funções muito simples, é mais prático editar as funções através do Scipad (que pode ser executado diretamente do console através do comando `scipad`).
- Uma função definida através dos comandos `function/endfunction` pode ser compilada através do comando `comp`.
- Uma função armazenada em formato texto em um arquivo pode ser invocada no console através do comando `exec` e `getf`.
- Demais funções nas seções I e XXIII do manual.

Objeto do tipo função

- A não ser por funções muito simples, é mais prático editar as funções através do Scipad (que pode ser executado diretamente do console através do comando `scipad`).
- Uma função definida através dos comandos `function/endfunction` pode ser compilada através do comando `comp`.
- Uma função armazenada em formato texto em um arquivo pode ser invocada no console através do comando `exec` e `getf`.
- Demais funções nas seções I e XXIII do manual.

Condicionais – estruturas de seleção

- Um programa no Scilab consiste em um conjunto de instruções que é executado sequencialmente.
- O fluxo de execução é controlado por estruturas de seleção e estruturas de repetição. Vamos estudá-las a seguir.

Condicionais – estruturas de seleção

As estruturas de seleção consistem em dois comandos: `if-the-else` e `case-select`.

Condicionais – estruturas de seleção

As estruturas de seleção consistem em dois comandos: `if-then-else` e `case-select`.

Forma de um comando `if-then-else` :

```
if expr1 then
instruções_1
elseif expr2 then
instruções_2
....
else
instruções_f
end
```

O comando avalia a expressão *expr1* se for um objeto booleano com valor %t, *instruções1* é executado caso contrário o comando avalia *expr2*, etc... Se nenhuma das expressões retornar %t então *instruções_f* é executado. O comando é finalizado pela expressão `end`.

Condicionais – estruturas de seleção

Observações

- As expressões else e elseif não são obrigatórias.
- O par if-then e elseif-then devem estar na mesma linha.
- O número de caracteres no corpo dessas estruturas é limitado em 16KiB.

Condicionais – estruturas de seleção

Observações

- As expressões else e elseif não são obrigatórias.
- O par if-then e elseif-then devem estar na mesma linha.
- O número de caracteres no corpo dessas estruturas é limitado em 16KiB.

Exemplo:

```
if (type(x)==1)&(abs(sign(imag(x)))+length(x)==1) then
  if x<0 then
    sinal=-1
  elseif x>0 then
    sinal=1
  else
    sinal=0
  end
end
end
```

Condicionais – estruturas de seleção

Observações

- As expressões else e elseif não são obrigatórias.
- O par if-then e elseif-then devem estar na mesma linha.
- O número de caracteres no corpo dessas estruturas é limitado em 16KiB.

Exemplo:

```
if (type(x)==1)&(abs(sign(imag(x)))+length(x)==1) then
  if x<0 then
    sinal=-1
  elseif x>0 then
    sinal=1
  else
    sinal=0
  end
end
```

Condicionais – estruturas de seleção

Forma de um comando select-case :

```
select expr,  
case expr1 then  
instruções_1,  
case expr2 then  
instruções_2 ,  
....  
else  
instruções_f ,  
end
```

O comando compara o valor *expr* e *expr1*; se forem iguais então *instruções1* é executado e o comando termina, caso contrário, compara o valor *expr* e *exp2*; se forem iguais então *instruções2* é executado, etc... Por fim, se nenhum dos casos for executado, *instruções_f* é executado. O comando é finalizado pela expressão end.

Laços – estruturas de repetição

As estruturas de repetição consistem em dois comandos: `for` e `while`.

Laços – estruturas de repetição

As estruturas de repetição consistem em dois comandos: `for` e `while`.
Forma de um comando `for` :

```
for contador=matriz,  
instruções,  
end
```

O comando repetirá o conjunto de instruções um número de vezes igual ao número de colunas do objeto *matriz*. A cada iterada, a variável contador assume o valor de uma das colunas da variável *matriz*.

Laços – estruturas de repetição

As estruturas de repetição consistem em dois comandos: `for` e `while`.
Forma de um comando `for` :

```
for contador=matriz,  
instruções,  
end
```

O comando repetirá o conjunto de instruções um número de vezes igual ao número de colunas do objeto *matriz*. A cada iterada, a variável contador assume o valor de uma das colunas da variável *matriz*.

Exemplos:

```
• x=0;  
  for i=1:5,  
    x=x+i,  
  end
```

```
• x=0;  
  for i=[-3 0 10 8],  
    x=x+i,  
  end
```

```
• x='';  
  for i=['ab' '??k' '-U$'],  
    x=x+i,  
  end
```

```
• x=[1;2;3];  
  for i=[1 2 -1;1 -1 2;-2 1 -1],  
    x=x+i,  
  end
```

Laços – estruturas de repetição

As estruturas de repetição consistem em dois comandos: `for` e `while`.
Forma de um comando `for` :

```
for contador=matriz,  
instruções,  
end
```

O comando repetirá o conjunto de instruções um número de vezes igual ao número de colunas do objeto *matriz*. A cada iterada, a variável contador assume o valor de uma das colunas da variável *matriz*.

Exemplos:

```
• x=0;  
  for i=1:5,  
    x=x+i,  
  end
```

```
• x=0;  
  for i=[-3 0 10 8],  
    x=x+i,  
  end
```

```
• x='';  
  for i=['ab' '??k' '-U$'],  
    x=x+i,  
  end
```

```
• x=[1;2;3];  
  for i=[1 2 -1;1 -1 2;-2 1 -1],  
    x=x+i,  
  end
```

Laços – estruturas de repetição

As estruturas de repetição consistem em dois comandos: `for` e `while`.
Forma de um comando `for` :

```
for contador=matriz,  
instruções,  
end
```

O comando repetirá o conjunto de instruções um número de vezes igual ao número de colunas do objeto *matriz*. A cada iterada, a variável contador assume o valor de uma das colunas da variável *matriz*.

Exemplos:

```
• x=0;  
  for i=1:5,  
    x=x+i,  
  end
```

```
• x=0;  
  for i=[-3 0 10 8],  
    x=x+i,  
  end
```

```
• x='';  
  for i=['ab' '??k' '-U$'],  
    x=x+i,  
  end
```

```
• x=[1;2;3];  
  for i=[1 2 -1;1 -1 2;-2 1 -1],  
    x=x+i,  
  end
```

Laços – estruturas de repetição

As estruturas de repetição consistem em dois comandos: `for` e `while`.
Forma de um comando `for` :

```
for contador=matriz,  
instruções,  
end
```

O comando repetirá o conjunto de instruções um número de vezes igual ao número de colunas do objeto *matriz*. A cada iterada, a variável contador assume o valor de uma das colunas da variável *matriz*.

Exemplos:

- ```
x=0;
for i=1:5,
 x=x+i,
end
```
- ```
x=0;  
for i=[-3 0 10 8],  
    x=x+i,  
end
```
- ```
x='';
for i=['ab' '??k' '-U$'],
 x=x+i,
end
```
- ```
x=[1;2;3];  
for i=[1 2 -1;1 -1 2;-2 1 -1],  
    x=x+i,  
end
```

Laços – estruturas de repetição

Forma de um comando `while` :

```
while expr ,  
instruções ,  
end
```

O comando avalia a expressão *expr*, se retornar %t as instruções são executadas, caso contrário o comando é encerrado. Quando termina a execução das instruções o comando volta a avaliar a expressão *expr*, etc...

Laços – estruturas de repetição

Forma de um comando `while` :

```
while expr ,  
  instruções ,  
end
```

O comando avalia a expressão *expr*, se retornar %t as instruções são executadas, caso contrário o comando é encerrado. Quando termina a execução das instruções o comando volta a avaliar a expressão *expr*, etc...

Exemplos:

- ```
x=1;
while x<15,
 x=x*2,
end
```

- ```
x=1;  
while 1<15,  
  x=x*2,  
  if x>30 then break,  
  end,  
end
```

Observação

O comando `break` interrompe a execução dos comandos `for` e `while`.

Laços – estruturas de repetição

Forma de um comando `while` :

```
while expr ,  
  instruções ,  
end
```

O comando avalia a expressão *expr*, se retornar %t as instruções são executadas, caso contrário o comando é encerrado. Quando termina a execução das instruções o comando volta a avaliar a expressão *expr*, etc...

Exemplos:

- ```
x=1;
while x<15,
 x=x*2,
end
```

- ```
x=1;  
while 1<15,  
  x=x*2,  
  if x>30 then break,  
  end,  
end
```

Observação

O comando `break` interrompe a execução dos comandos `for` e `while`.

Laços – estruturas de repetição

Forma de um comando `while` :

```
while expr ,  
  instruções ,  
end
```

O comando avalia a expressão *expr*, se retornar %t as instruções são executadas, caso contrário o comando é encerrado. Quando termina a execução das instruções o comando volta a avaliar a expressão *expr*, etc...

Exemplos:

- ```
x=1;
while x<15,
 x=x*2,
end
```

- ```
x=1;  
while 1<15,  
  x=x*2,  
  if x>30 then break,  
  end,  
end
```

Observação

O comando `break` interrompe a execução dos comandos `for` e `while`.

Laços – estruturas de repetição

Forma de um comando `while` :

```
while expr ,  
  instruções ,  
end
```

O comando avalia a expressão *expr*, se retornar %t as instruções são executadas, caso contrário o comando é encerrado. Quando termina a execução das instruções o comando volta a avaliar a expressão *expr*, etc...

Exemplos:

- ```
x=1;
while x<15,
 x=x*2,
end
```

- ```
x=1;  
while 1<15,  
  x=x*2,  
  if x>30 then break,  
  end,  
end
```

Observação

O comando `break` interrompe a execução dos comandos `for` e `while`.

Escopo das variáveis

Quanto ao escopo, as variáveis podem estar definidas apenas dentro de uma particular função (variáveis locais), podem estar definidas no console (variáveis de ambiente) ou podem ainda ser definidas globalmente (variáveis globais).

Escopo das variáveis

Quanto ao escopo, as variáveis podem estar definidas apenas dentro de uma particular função (variáveis locais), podem estar definidas no console (variáveis de ambiente) ou podem ainda ser definidas globalmente (variáveis globais).

- As **variáveis locais** são aquelas definidas dentro de uma função. Somente existem durante a sua execução.
- As **variáveis de ambiente** são aquelas definidas diretamente no console (além das pré-definidas). Estão disponíveis durante toda a sessão e podem ser usadas dentro das funções. Elas são eliminadas com o comando `clear`: se `var` for uma variável de ambiente, o comando `clear var` a elimina.
- As **variáveis globais** são definidas pelo comando `global`: o comando `global var` cria uma variável global `var`. Esse tipo de variável pode ser modificado dentro de uma função e ser posteriormente utilizado por outra. O seu uso dentro de uma função deve ser explicitamente indicado através do comando `global nome_da_variável`. Uma variável global é eliminada através do comando `clearglobal nome_da_variável`. O comando `clearglobal()` elimina todas as variáveis globais.

Escopo das variáveis

Observações

- Variáveis de diferente escopo podem possuir mesmo nome (e valores distintos), i. e. , pode existir uma variável local `var` definida em uma função, uma variável de ambiente `var` e uma variável global `var`.
- O conjunto de variáveis de ambiente e globais definidas em um dado instante, pode ser obtido com o comando `who`. O comando `whos` fornece informação mais detalhada sobre essas variáveis e ainda é possível utilizar o comando `whos -name var` para obter informação detalhada sobre a variável `var`.
- Se já existir uma variável de ambiente `var`, o comando `global var` cria uma variável global `var` cujo valor é o mesmo da anterior (no entanto, ele pode ser posteriormente alterado). Se não houver variável com o mesmo nome, `global var` cria uma matriz nula `[]`.

Escopo das variáveis

Exemplos:

```
--> def f(' [y]=f1(x)', ' b=0.28; y=b*x')
```

--> f1(1); disp(b) → retorna erro: nesse caso b está definida apenas durante a execução de f1.

--> b=3; f1(1) → b agora é uma variável de ambiente (mas a função ainda usa a local).

```
--> def f(' [y]=f2(x)', ' y=b*x^2')
```

--> f2(1) → como b está definida no ambiente, ela pode ser invocada dentro da função.

```
--> def f(' [y]=f3(x)', ' global b; b=sin(x); y=b')
```

```
--> def f(' [y]=f4(x)', ' global b; y=b^2')
```

--> f3(1) → 0.8414710. Nesse caso a variável global b assume o valor sin(1).

--> f4(1) → 0.7080734

--> f3(2) → 0.9092974. Agora a variável global b assume o valor sin(2).

--> f4(1) → 0.8268218. E altera o resultado de f4(1)!

Saída de dados no console

A formatação dos dados retornados pelo console é realizada pelo comando `format`. Uma das formas é `format(tipo, dígitos)` onde *tipo* é uma string ('v' ou 'e') e *dígitos* um inteiro no intervalo [2, 25].

Saída de dados no console

A formatação dos dados retornados pelo console é realizada pelo comando `format`. Uma das formas é `format(tipo, dígitos)` onde *tipo* é uma string ('v' ou 'e') e *dígitos* um inteiro no intervalo [2, 25].

- 'v' indica um número variável de dígitos.
- 'e' indica uma quantia fixa de dígitos em formato científico.
- *dígitos* indica o número máximo de caracteres utilizados na saída dos dados.

Saída de dados no console

A formatação dos dados retornados pelo console é realizada pelo comando `format`. Uma das formas é `format(tipo, dígitos)` onde *tipo* é uma string ('v' ou 'e') e *dígitos* um inteiro no intervalo [2, 25].

- 'v' indica um número variável de dígitos.
- 'e' indica uma quantia fixa de dígitos em formato científico.
- *dígitos* indica o número máximo de caracteres utilizados na saída dos dados.

Exemplos:

```
-->x=100*%pi,  
-->format('v',20)  
-->x  
-->format('e',20)  
-->x
```

Saída de dados no console

A formatação dos dados retornados pelo console é realizada pelo comando `format`. Uma das formas é `format(tipo, dígitos)` onde *tipo* é uma string ('v' ou 'e') e *dígitos* um inteiro no intervalo [2, 25].

- 'v' indica um número variável de dígitos.
- 'e' indica uma quantia fixa de dígitos em formato científico.
- *dígitos* indica o número máximo de caracteres utilizados na saída dos dados.

Exemplos:

```
-->x=100*%pi,  
-->format('v',20)  
-->x  
-->format('e',20)  
-->x
```

O comando `disp(var)` retorna o valor do objeto *var* ao console de acordo com a formatação definida pelo comando `format`.

Saída de dados no console

Uma outra forma de retornar saída formatada é através do comando `mprintf`. Esse comando possui uma sintaxe semelhante aos comandos `printf` utilizados nas linguagens C e Fortran.

Saída de dados no console

Uma outra forma de retornar saída formatada é através do comando `mprintf`. Esse comando possui uma sintaxe semelhante aos comandos `printf` utilizados nas linguagens C e Fortran.

O comando é da forma `mprintf(string)` ou `mprintf(string, var_1, var_2, ..., var_n)`. No segundo caso o texto representado pela variável `string` deve conter `n` descritores que indicam o formato no qual as variáveis `var_1` até `var_n` devem ser retornadas ao console.

Saída de dados no console

O descritor possui a seguinte forma `%[comp][.precisão]tipo`, onde os colchetes indicam que o uso é opcional.

- *comp* é um inteiro que indica a quantidade mínima de caracteres que deve ser utilizados no retorna da variável.
- *precisão* é também inteiro e indica a quantidade de dígitos utilizada para representar números com parte fracionária. A sua interpretação depende do termo *tipo*.

Saída de dados no console

O descritor possui a seguinte forma `%[comp][.precisão]tipo`, onde os colchetes indicam que o uso é opcional.

- *comp* é um inteiro que indica a quantidade mínima de caracteres que deve ser utilizados no retorna da variável.
- *precisão* é também inteiro e indica a quantidade de dígitos utilizada para representar números com parte fracionária. A sua interpretação depende do termo *tipo*.

Saída de dados no console

O descritor possui a seguinte forma `%[comp][.precisão]tipo`, onde os colchetes indicam que o uso é opcional.

- *tipo* indica a formatação a ser utilizada para retornar a variável:
 - *d* ou *i* para inteiros.
 - *s* para strings. Nesse caso, *precisão* indica o número de caracteres a ser retornado
 - *c* para caracteres. Retorna um único caracter. O termo *precisão* é ignorado.
 - *f* para doubles, notação de ponto fixo (sem expoente, na forma *parte “inteira.parte fracionária”*). Nesse caso *precisão* indica a quantidade de dígitos após o ponto.
 - *e* para doubles, notação científica. Nesse caso, *precisão* indica o número de dígitos no significando subtraído de uma unidade.
 - *g* para doubles, utiliza a notação mais apropriada de acordo com a magnitude da variável. Nesse caso, *precisão* indica a quantidade total de dígitos no significando.

Saída de dados no console

Observações

- A variável string aceita os caracteres de controle `\n` (retorno de linha) e `\t` (tabulação).
- O caracter `%` é retornado através da sequência `%%`.
- Se `var_1, \dots, var_n` forem matrizes coluna, a função `mprintf` é iterada pela número de vezes igual ao número de linhas da variável com o menor número.

Saída de dados no console

Exemplos:

```
x=100*%pi;  
texto='A terra é azul.';  
cores=['vermelho';'verde';'azul';'rosa';'preto'];  
RGB=[1 0 0;0 1 0;0 0 1;1 0.75 0.75;0 0 0];  
mprintf('100 vezes pi vale %.16e',x);  
mprintf('100 vezes pi vale %.6f',x);  
mprintf('100 vezes pi vale %.6g',x);  
mprintf('100 vezes pi vale %30.2g',x);  
mprintf('%s\n Não é verde.',texto);  
mprintf('%c\t Não é verde.',texto);  
mprintf('%d\t%s\t%f\t%f\t%f\n',(1:5)',colors,RGB);  
mprintf('%d\t%s\t%f\t%f\t%f\n',(1:2:5)',colors,RGB);
```

Entrada e saída de dados

Texto sobre os comandos `read`, `write`, `mopen`, `mclose`, `mget`, `mput`.

Gráficos

Texto sobre os recursos comando plot e plot3d.