

# Introdução ao Scilab

## Versão 3.0

Prof. Paulo Sérgio da Motta Pires

Departamento de Engenharia de Computação e Automação  
Universidade Federal do Rio Grande do Norte  
Natal-RN, Julho de 2004

## Resumo

Scilab é um ambiente utilizado no desenvolvimento de programas para a resolução de problemas numéricos. Criado e mantido por pesquisadores pertencentes ao *Institut de Recherche en Informatique et en Automatique*, INRIA, através do Projeto MÉTALAU (*Méthods, algorithmes et logiciels pour l'automatique*) e à *École Nationale des Ponts et Chaussées*, ENPC, Scilab é gratuito (*free software*) e é distribuído com o código fonte (*open source software*). A partir de maio de 2003, Scilab passou a ser mantido por um consórcio de empresas e instituições francesas denominado de **Consórcio Scilab**.

Embora seja apresentado como um software **CASCD**, *Computer Aided Control System Design* - Projeto de Sistemas de Controle Auxiliado por Computador, Scilab pode ser usado para desenvolvimento ou prototipação de software numérico de propósito geral.

Este é um documento sobre a utilização e as principais características deste ambiente de programação numérica. É importante ressaltar que as referências definitivas sobre Scilab são os manuais que acompanham o software. Por exemplo, podemos citar *Introduction to Scilab - User's Guide* [1], documento no qual este texto se baseia.

O objetivo principal é apresentar um texto introdutório, em português, sobre Scilab. Nosso interesse é fazer deste documento um complemento aos textos utilizados em disciplinas como Métodos Computacionais, Cálculo Numérico, Computação Numérica, Álgebra Linear Computacional e correlatas. Nos interessa, também, mostrar que o Scilab é uma excelente ferramenta de suporte para linhas de pesquisa onde o uso de computadores na resolução numérica de problemas é intensivo.

A versão mais recente deste trabalho está disponível, no formato pdf, em <http://www.dca.ufrn.br/~pmotta>. Comentários ou sugestões podem ser enviados para [pmotta@dca.ufrn.br](mailto:pmotta@dca.ufrn.br).

## Agradecimentos

Ao Klaus Steding-Jessen, pelo L<sup>A</sup>T<sub>E</sub>X-demo. A maioria das informações sobre “como se faz isso em L<sup>A</sup>T<sub>E</sub>X ?” podem ser encontradas no documento escrito pelo Klaus<sup>1</sup>;

Ao Dr. Jesus Olivan Palacios, pelas “conversas” sobre o Scilab. O Dr. Palacios sintetiza com brilhantismo as vantagens de se utilizar software livre e de código aberto;

Aos colegas, pelas contribuições e sugestões que melhoraram a apresentação deste trabalho.

## Distribuição

**Este trabalho pode ser copiado e distribuído livremente, mantidos os créditos ao seu autor.**

---

<sup>1</sup>Informações sobre o L<sup>A</sup>T<sub>E</sub>X-demo em particular ou sobre o L<sup>A</sup>T<sub>E</sub>X em geral podem ser obtidas em <http://biquinho.furg.br/tex-br/>

## Histórico deste Documento

- Fevereiro de 1999 - Versão 0.1 - Início.
- Julho de 2001 - Versão 0.2 - Correções e atualizações.
- Julho de 2001 - Versão 0.3 - Correções e atualizações.
- Julho/Novembro de 2001 - Versão 1.0 - Reorganização do trabalho e correções. Disponibilização deste documento no *site* do Scilab - INRIA (<http://scilabsoft.inria.fr/books.html>)
- Maio/Julho de 2004 - Versão 3.0 - Reorganização do trabalho, correções e atualizações. O Scilab passa a ser mantido pelo **Consórcio Scilab** a partir de maio de 2003. O número da versão deste documento passa a ser igual ao número da versão do Scilab que ele descreve.

Este trabalho foi totalmente desenvolvido utilizando *free ou open source* software. O Scilab versão 3.0<sup>2</sup> foi instalado, a partir do código fonte, no Linux distribuição Slackware 9.1<sup>3</sup>, kernel versão 2.4.24<sup>4</sup>. A digitação L<sup>A</sup>T<sub>E</sub>X foi feita usando o Xemacs<sup>5</sup>. As figuras, em **jpg** foram capturadas usando o **GIMP**<sup>6</sup> versão 1.2.5. O texto completo foi transformado em **pdf** através do **pdflatex**.

A máquina utilizada é um Pentium MMX 200, 64 MB de RAM com um disco rígido de 15 GB.

---

<sup>2</sup>Página do Scilab : <http://scilabsoft.inria.fr>

<sup>3</sup>Página do Linux distribuição Slackware : <http://www.slackware.com>

<sup>4</sup>Página do kernel Linux <http://www.kernel.org>

<sup>5</sup>Página do Xemacs <http://www.xemacs.org>

<sup>6</sup>Página do GIMP <http://www.gimp.org>

# Sumário

Scilab - Versão 3.0 . . . . .	1
Resumo . . . . .	i
Agradecimentos . . . . .	ii
Distribuição . . . . .	ii
Histórico deste Documento . . . . .	iii
Sumário . . . . .	iv
Lista de Figuras . . . . .	vi
Lista de Tabelas . . . . .	vii
Lista de Códigos . . . . .	viii
<b>1 Introdução</b>	<b>1</b>
<b>2 O Ambiente Scilab</b>	<b>4</b>
2.1 Introdução . . . . .	4
2.2 O Ambiente Gráfico do Scilab . . . . .	5
2.3 Variáveis Especiais . . . . .	10
2.4 Manipulação de Arquivos e Diretórios . . . . .	11
2.5 O <i>help</i> do Scilab . . . . .	14
2.6 Arquivos com Comandos Scilab . . . . .	17
<b>3 Operações Básicas com Scilab</b>	<b>19</b>
3.1 Introdução . . . . .	19
3.2 Utilizando as Funções Internas do Scilab . . . . .	23
<b>4 Polinômios, Vetores, Matrizes e Listas</b>	<b>25</b>
4.1 Polinômios . . . . .	25
4.2 Vetores . . . . .	27
4.3 Matrizes . . . . .	31
4.4 Acesso a Elementos de Vetores e de Matrizes . . . . .	35
4.5 Matrizes com Polinômios . . . . .	41
4.6 Matrizes Simbólicas . . . . .	43
4.7 Matrizes Booleanas . . . . .	45
4.8 Operações com Vetores e Matrizes . . . . .	46
4.9 Listas . . . . .	53
<b>5 Programação</b>	<b>56</b>
5.1 Comandos para Iterações . . . . .	56
5.1.1 O <i>Loop for</i> . . . . .	56
5.1.2 O <i>Loop while</i> . . . . .	58
5.2 Comandos Condicionais . . . . .	59
5.2.1 Comando <i>if-then-else</i> . . . . .	60
5.2.2 Comando <i>select-case</i> . . . . .	61

5.3	Definindo <i>Scripts</i> . . . . .	62
5.4	Definindo Funções . . . . .	65
5.4.1	Variáveis Globais e Variáveis Locais . . . . .	66
5.4.2	Arquivos com Funções . . . . .	68
5.4.3	Comandos Especiais . . . . .	74
<b>6</b>	<b>Gráficos no Scilab</b> . . . . .	<b>76</b>
6.1	A Janela de Gráficos do Scilab . . . . .	76
6.2	Gráficos Bi-dimensionais . . . . .	77
6.2.1	Outros Comandos . . . . .	83
6.2.2	Gráficos 2D Especiais . . . . .	85
6.3	Gráficos Tri-dimensionais . . . . .	86
6.3.1	Gráficos 3-D Especiais . . . . .	87
<b>A</b>	<b>Instalação do Scilab</b> . . . . .	<b>89</b>
A.1	Instalação no Linux Slackware - Código Fonte . . . . .	89
<b>B</b>	<b>Ligação do Scilab com Programas em C</b> . . . . .	<b>92</b>
B.1	A Ligação Dinâmica . . . . .	92
<b>C</b>	<b>Instalação de <i>Toolboxes</i></b> . . . . .	<b>96</b>
C.1	Instalação . . . . .	96
<b>D</b>	<b>Funções Pré-definidas - Scilab 3.0</b> . . . . .	<b>99</b>
D.1	Programming . . . . .	99
D.2	Graphics Library . . . . .	101
D.3	Elementary Functions . . . . .	104
D.4	Input/Output Functions . . . . .	106
D.5	Handling of functions and libraries . . . . .	107
D.6	Character string manipulations . . . . .	108
D.7	GUI and Dialogs . . . . .	108
D.8	Utilities . . . . .	108
D.9	Linear Algebra . . . . .	109
D.10	Polynomial calculations . . . . .	110
D.11	General System and Control . . . . .	111
D.12	Robust control toolbox . . . . .	112
D.13	Optimization and simulation . . . . .	112
D.14	Signal Processing toolbox . . . . .	113
D.15	Arma modelisation and simulation toolbox . . . . .	114
D.16	Metanet: graph and network toolbox . . . . .	114
D.17	Sound file handling . . . . .	115
D.18	Language or data translations . . . . .	115
D.19	PVM parallel toolbox . . . . .	115
D.20	TdCs . . . . .	116
D.21	TCL/Tk interface . . . . .	116
D.22	Statistic basics . . . . .	116
D.23	Cumulative Distribution Functions; Inverses, grand . . . . .	117
D.24	Identification . . . . .	117
D.25	Matlab to Scilab conversion tips . . . . .	118
	<b>Referências Bibliográficas</b> . . . . .	<b>120</b>

# Lista de Figuras

2.1	Tela inicial do Scilab no ambiente gráfico do Linux. . . . .	5
2.2	Tela com as opções de operações sobre arquivos, <u>File Operations</u> . . . . .	6
2.3	Programas de demonstração, opção <u>Demos</u> , do Scilab 3.0. . . . .	7
2.4	Tela da sub-opção <u>Help browser</u> com navegador padrão do Scilab. . . . .	8
2.5	Tela de configuração para a escolha do navegador do <u>Help</u> do Scilab. . . . .	8
2.6	Tela inicial do <u>Scipad</u> , editor incorporado ao Scilab. . . . .	10
2.7	Tela de <u>help</u> para a função <u>det</u> . . . . .	15
2.8	Comando <u>help</u> para a função <u>det</u> . . . . .	15
2.9	Texto do <u>help</u> para a função <u>besselk</u> . . . . .	16
2.10	Comando <u>diary</u> para gravação de sessões desenvolvidas no ambiente Scilab. . . . .	17
3.1	Rodando o exemplo de utilização da função <u>fft</u> apresentado no <u>help</u> do Scilab. . . . .	23
3.2	A função K de Bessel, <u>besselk</u> . O exemplo apresentado no <u>help</u> da função é copiado para o editor SciPad, selecionado e executado através da sub-opção <u>Evaluate Selection Ctrl+y</u> da opção <u>Execute</u> . . . . .	24
5.1	Escrevendo uma função usando o editor do Scilab. . . . .	69
6.1	Janela gráfica do Scilab . . . . .	76
6.2	Saídas para a função <u>plot2d([x], y)</u> . Cada sub-gráfico refere-se a um dos itens da sessão do Scilab mostrada anteriormente. Observar que os gráficos dos Itens 4 e 5 possuem valores de abcissas diferentes dos demais. . . . .	80
6.3	Saídas para a função <u>plot2d([x], y, &lt;opt_args&gt;)</u> . . . . .	83
6.4	Saídas para a função <u>subplot()</u> . . . . .	85
6.5	Exportando gráficos para o <u>L<sup>A</sup>T<sub>E</sub>X</u> . . . . .	85
6.6	Exemplo de saída gráfica 3-D. . . . .	87
6.7	Exemplos de gráficos 3-D especiais. . . . .	88
C.1	Procedimentos para a utilização do <u>toolbox ANN</u> e <u>help</u> com as funções disponíveis no <u>toolbox</u> . . . . .	98

# Lista de Tabelas

2.1	Teclas de edição linhas de comando no <i>prompt</i> do Scilab. . . . .	14
4.1	Sintaxe de alguns operadores usados em operações vetoriais ou matriciais. . . . .	46
5.1	Operadores condicionais . . . . .	59
6.1	Variações do comando <code>plot2d()</code> . . . . .	83



# Lista de Códigos

1	O <i>script</i> que implementa o método de Newton-Raphson para obter $\sqrt{2}$ . . . . .	63
2	Programa principal, implementação do método de Runge-Kutta. . . . .	70
3	A função $f(x, y)$ . . . . .	70
4	A solução exata da equação diferencial. . . . .	70
5	Programa para resolver um sistema triangular. . . . .	74
6	O <i>script</i> utilizado para gerar o gráfico da Figura 6.2. . . . .	81
7	Função Runge-Kutta escrita em C. . . . .	93

# Capítulo 1

## Introdução

Scilab<sup>1</sup> é um ambiente voltado para o desenvolvimento de software para resolução de problemas numéricos. Scilab foi criado em 1990 por um grupo de pesquisadores do INRIA<sup>2</sup>-*Institut de Recherche en Informatique et en Automatique* e do ENPC<sup>3</sup>-*École Nationale des Ponts et Chaussées*.

Desde 1994, quando passou a ser disponível na Internet, Scilab é gratuito, *free software*, e distribuído com o código fonte, *open source software*. Além da distribuição com o código fonte, existem, também, distribuições pré-compiladas do Scilab para vários sistemas operacionais. Na versão 3.0, na data em que este documento foi escrito, Scilab está disponível para as seguintes plataformas:

- Plataformas UNIX/Linux:
  - Scilab 3.0 - arquivo binário para Linux (scilab-3.0.bin.linux-i686.tar.gz);
  - Scilab 3.0 - arquivo com o código fonte do Scilab (scilab-3.0.src.tar.gz).
- Plataformas Windows 9X/NT/2000/XP:
  - Scilab 3.0 - instalador da versão binária do Scilab (scilab3.0.exe);
  - Scilab 3.0 - código fonte do Scilab (scilab-3.0.src.zip)

A partir de maio de 2003, Scilab passou a ser mantido por um consórcio de empresas e instituições francesas denominado de **Consórcio Scilab**. Os principais objetivos deste consórcio são:

- Organizar a cooperação e o intercâmbio entre os desenvolvedores do Scilab objetivando incorporar ao software os últimos avanços científicos na área da computação numérica;
- Organizar a cooperação e o intercâmbio entre os usuários do Scilab objetivando fazer com que o software possa ser utilizado de maneira mais efetiva na indústria, na educação e na pesquisa, e
- Angariar recursos para a manutenção da equipe de desenvolvedores e para garantir um suporte mais adequado às necessidades dos usuários.

Embora seja apresentado pelos seus mantenedores como um software **CASCD** - *Computer Aided Control System Design* - Projeto de Sistemas de Controle Auxiliado por Computador,

---

<sup>1</sup>Página do Scilab: <http://scilabsoft.inria.fr>

<sup>2</sup>Página do INRIA : <http://www.inria.fr>

<sup>3</sup>Página do ENPC : <http://www.enpc.fr>

Scilab é um ambiente para desenvolvimento ou prototipação de software numérico de propósito geral.

O objetivo principal deste trabalho é divulgar o ambiente Scilab através de um texto escrito em português. Com este objetivo em mente, a ênfase maior é dada na apresentação das características do próprio ambiente. Assim, apesar do rigorismo, não há preocupações excessivas em relação aos tipos de problemas tratados ou em relação aos exemplos apresentados. Partimos do princípio que o leitor deste trabalho já possua conhecimentos práticos, mesmo rudimentares, sobre programação.

O objetivo secundário, também relevante, é mostrar que a utilização de software livre e de código aberto, *free/open source software*, do ponto de vista do usuário, traz grandes vantagens. Algumas delas, apresentadas em [2], são:

- A última versão do software está sempre disponível, geralmente através da Internet;
- O software pode ser **legalmente** utilizado, copiado, distribuído, modificado;
- Os resultados obtidos podem ser divulgados sem nenhuma restrição;
- Os programas desenvolvidos podem ser transferidos para outras pessoas sem imposições ou constrangimentos de quaisquer natureza;
- O acesso ao código fonte, evitando surpresas desagradáveis;
- O acesso a informação de alta qualidade, e
- A certeza de estar participando de uma comunidade cujo principal valor é a irrestrita difusão do conhecimento.

Existem, ainda, algumas pretensões com a divulgação deste trabalho. Uma delas é fazer deste documento um complemento para os textos utilizados em disciplinas como Métodos Computacionais, Cálculo Numérico, Computação Numérica, Álgebra Linear Computacional e correlatas. Uma outra, é mostrar que Scilab é uma excelente ferramenta de suporte para linhas de pesquisa onde o uso de computadores na resolução numérica de problemas é intensivo. A última versão deste trabalho encontra-se disponível em <http://www.dca.ufrn.br/~pmotta>. Comentários ou sugestões sobre esse documento são sempre bem-vindas e podem ser enviados para [pmotta@dca.ufrn.br](mailto:pmotta@dca.ufrn.br).

É importante ressaltar que as referências definitivas sobre Scilab permanecem sendo os manuais que acompanham o software. Na data em que este trabalho foi escrito, estavam disponíveis na *homepage* do Scilab os seguintes documentos, [1]:

- **Introduction to Scilab** - manual de introdução ao Scilab, documento no qual este texto se baseia, nos formatos HTML, PDF, Postscript, com os arquivos fontes em  $\LaTeX$ ;
- **Communication Toolbox Documentation** - documentação sobre o *toolbox* de comunicação, nos formatos HTML, PDF, Postscript, com os arquivos fontes em  $\LaTeX$ ;
- **Signal Processing** - documentação sobre o *toolbox* de processamento de sinais, nos formatos PDF, Postscript, com os arquivos fontes em  $\LaTeX$ ;
- **Lmitool: Linear Matrix Inequalities Optimization Toolbox** - documentação sobre o *toolbox* de otimização, nos formatos HTML, PDF, Postscript, com os arquivos fontes em  $\LaTeX$ ;
- **Metanet User's Guide and Tutorial** - tutorial sobre a utilização do *toolbox* de grafos Metanet, nos formatos HTML, PDF, Postscript, com os arquivos fontes em  $\LaTeX$ ;

- **Scicos**, documentação sobre o ambiente de simulação do Scilab nos formatos HTML, PDF, Postscript, com os arquivos fontes em  $\text{\LaTeX}$ ;
- **Scilab's Internals Documentation** - documentação sobre as características internas do Scilab, nos formatos HTML, PDF, Postscript, com os arquivos fontes em  $\text{\LaTeX}$ ;
- **HOWTO's Scilab** - várias dicas sobre a utilização do Scilab, no formato HTML;
- **Scilab's demonstrations** - programas de demonstração de funcionalidades do Scilab, no formato HTML;
- **Intersci** - documentação sobre a interconexão do Scilab com programas escritos nas linguagens C ou FORTRAN, nos formatos PDF, Postscript, com os arquivos fontes em  $\text{\LaTeX}$ , e
- **Inline help pages** - documentação contendo o *help* de funções do Scilab nos formatos HTML, PDF, Postscript, com os arquivos fonte em  $\text{\LaTeX}$ .

Este documento, desenvolvido para satisfazer os objetivos estabelecidos em parágrafos precedentes, está dividido em seis Capítulos e quatro Apêndices. Neste Capítulo, mostramos o contexto no qual o ambiente Scilab e este trabalho estão inseridos.

No Capítulo 2, apresentamos uma visão geral das principais características do ambiente Scilab. Descrevemos as suas diversas opções e apresentamos os comandos básicos utilizados na edição de comandos no ambiente Scilab.

No Capítulo 3, apresentamos diversos exemplos de manipulações numéricas básicas que podem ser realizadas com o software. São enfatizadas operações com números (reais, complexos) e dados alguns exemplos de utilização de funções internas do Scilab.

O Capítulo 4 é dedicado aos vários tipos de dados que podem ser manipulados pelo Scilab. Apresentamos polinômios, vetores, matrizes e listas.

No Capítulo 5, são dados exemplos de desenvolvimento de programas no Scilab e, finalizando, no Capítulo 6, utilizamos comandos do Scilab voltados para a geração de gráficos bi-dimensionais e tri-dimensionais.

No Apêndice A, mostramos os procedimentos para a instalação do software, a partir do código fonte, em máquinas com o sistema operacional Linux (a instalação foi realizada em uma máquina com distribuição Slackware 9.1, kernel versão 2.4.24). Os procedimentos para a instalação das distribuições binárias do Scilab, por serem específicos de cada plataforma, não são apresentados. O usuário é aconselhado a buscar estas informações na página do Scilab. Descrevemos, ainda, os principais arquivos e diretórios que compõem o ambiente Scilab.

No Apêndice B, apresentamos um procedimento que permite executar códigos escritos em linguagem C dentro do ambiente Scilab.

No Apêndice C, apresentamos os procedimentos padrões para a instalação de *toolboxes* no Scilab.

No Apêndice D, apresentamos uma listagem de todas as funções pré-definidas disponíveis no ambiente Scilab-3.0.

Por tratar-se de um texto introdutório, deixamos de apresentar diversas características do ambiente Scilab que, entretanto, podem ser consultadas nos documentos citados anteriormente.

Acreditamos que a maneira mais adequada de ler este documento é em frente a um computador com Scilab instalado e funcionando. Os exemplos apresentados e a própria funcionalidade do software poderão, desta forma, ser explorados com maior eficiência.

Este trabalho pode ser copiado e distribuído livremente, dados os devidos créditos ao seu autor.

## Capítulo 2

# O Ambiente Scilab

Neste Capítulo, apresentamos algumas características do ambiente Scilab em plataforma gráfica Linux. Em seguida, mostramos exemplos de manipulação de arquivos e de diretórios a partir desse ambiente. O objetivo é a familiarização com o software.

### 2.1 Introdução

Scilab é um ambiente de programação numérica bastante flexível. Suas principais características são:

1. É um software de distribuição gratuita, com código fonte disponível. Sua linguagem é simples e de fácil aprendizado;
2. Possui um sistema de auxílio ao usuário, *help*;
3. É um ambiente poderoso para geração de gráficos bi-dimensionais e tri-dimensionais, inclusive com animação;
4. Implementa diversas funções para manipulação de matrizes. As operações de concatenação, acesso e extração de elementos, transposição, adição e multiplicação de matrizes são facilmente realizadas;
5. Permite trabalhar com polinômios, funções de transferência, sistemas lineares e grafos;
6. Apresenta facilidades para a definição de funções;
7. Permite o acesso a rotinas escritas nas linguagens FORTRAN ou C;
8. Pode ser acessado por programas de computação simbólica como o Maple<sup>1</sup>, que é um software comercial, ou o MuPAD<sup>2</sup>, que é livre para uso em instituições de ensino/pesquisa;
9. Suporta o desenvolvimento de conjuntos de funções voltadas para aplicações específicas, os chamados *toolboxes*.

Além dos *toolboxes* desenvolvidos pelo Grupo Scilab, outros estão disponíveis também gratuitamente. Para exemplificar, destacamos o ANN (*Artificial Neural Network Toolbox*), para redes neurais, o FISLAB (*Fuzzy Logic Inference Toolbox*), para lógica difusa, e o FRACLAB (*Fractal, Multifractal and Wavelet Analysis Toolbox*), para análise de sinais utilizando fractais e wavelets. No Apêndice C, apresentamos os procedimentos necessários para a instalação do *toolbox* de redes neurais, ANN, no ambiente Scilab.

---

<sup>1</sup>Página do Maple: <http://www.maplesoft.com>

<sup>2</sup>Página do MuPAD: <http://www.mupad.de>

Existem trabalhos desenvolvidos tendo Scilab como ferramenta principal como, por exemplo, o apresentado em [2] e em alguns documentos introdutórios, [3, 4, 5, 6, 7]. Também, Scilab, através de uma extensão chamada de **Scilab Paralelo** [8], Scilab//, pode ser executado em máquinas paralelas ou em redes de estações de trabalho, as *NOWs - Network of Workstations*, usando as funções do *toolbox* PVM (Parallel Virtual Machine). Com o Scilab//, processos podem ser ativados, programas podem ser executados em estações remotas, com comunicação entre eles, e os resultados agregados.

Algumas das funções implementadas no Scilab baseiam-se em bibliotecas bem estabelecidas. Por exemplo<sup>3</sup>,

- Funções de Álgebra Linear - baseadas nas bibliotecas LINPACK, EISPACK, LAPACK e BLAS
- Funções para Resolução de Equações Diferenciais - baseadas nas bibliotecas ODEPACK, SLATEC;
- Funções de Otimização - baseadas na biblioteca MINPACK;

entre outras. A adoção de bibliotecas bem estabelecidas contribui para a estabilidade e a qualidade dos resultados apresentados pelo Scilab.

## 2.2 O Ambiente Gráfico do Scilab

Após a realização dos procedimentos de instalação descritos no Apêndice A, podemos começar a trabalhar com Scilab. Assumiremos que o software esteja instalado no sistema operacional Linux. Em uma *shell* no ambiente gráfico do Linux<sup>4</sup>, basta digitar `scilab` para começar a utilizar o programa. A tela inicial do Scilab é apresentada na Figura 2.1.

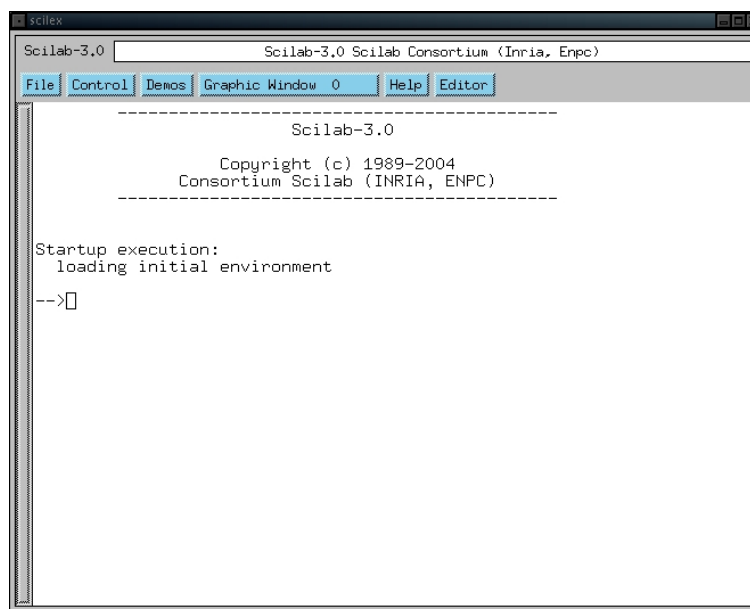


Figura 2.1: Tela inicial do Scilab no ambiente gráfico do Linux.

Na Figura 2.1, observamos que o *prompt* do Scilab é representado por uma seta, `-->` e que o cursor do Scilab é representado pelo símbolo `[]`. Este *prompt* é chamado de *prompt* inicial

<sup>3</sup>O código fonte dessas bibliotecas está disponível em <http://www.netlib.org>

<sup>4</sup>Scilab pode ser executado, também, no ambiente texto do Linux. Basta digitar `scilab -nw`. No ambiente texto, os gráficos que porventura forem gerados, serão apresentados no terminal gráfico, acessível via `Ctrl-Alt-F7`, caso este esteja disponível.

ou *prompt* de nível zero. Ainda na Figura 2.1, podemos observar a existência de um menu horizontal com seis opções: **File**, **Control**, **Demos**, **Graphic Window 0**, **Help** e **Editor**. Utilizando o *mouse* para escolher cada uma das opções, verificamos que:

- A opção **File** possui três sub-opções:
  - **File Operations**, que permite carregar arquivos, funções e executar o conteúdo de arquivos com comandos Scilab, entre outras ações. Na Figura 2.2, apresentamos o Scilab e a tela correspondente à essa opção.
  - **Kill**, que permite interromper de maneira abrupta o processamento, saindo do ambiente Scilab.
  - **Quit**, que permite sair do ambiente Scilab de forma natural.

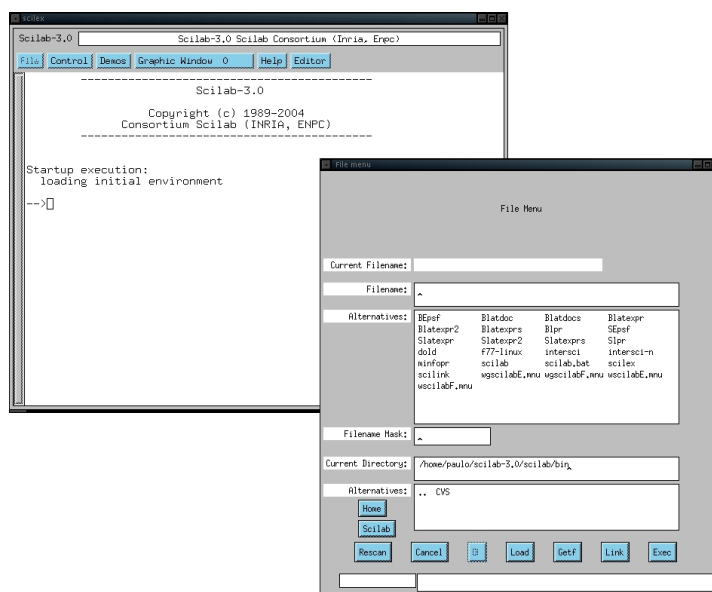


Figura 2.2: Tela com as opções de operações sobre arquivos, **File Operations**.

- A opção **Control**, que também possui três sub-opções:
  - **Resume** - continua a execução após uma **pause** ter sido dada através de um comando em uma função ou através de **Stop** ou **Ctrl-c**.
  - **Abort** - aborta a execução após uma ou várias **pause**, retornando ao *prompt* inicial.
  - **Stop** - interrompe a execução do Scilab e entra em modo **pause**. No *prompt*, equivale a um **Ctrl-c**.

Essas operações são mostradas na sessão Scilab:

```
-->          // Ctrl-c no prompt inicial

-1->         // leva ao prompt de primeiro nivel

-1->         // Ctrl-c no prompt de primeiro nivel

-2->         // leva ao prompt de segundo nivel

-2->resume   // retorna ao prompt de primeiro nivel
```

```

-1->resume      // retorna ao prompt inicial

-->             // Ctrl-c

-1->           // Ctrl-c

-2->           // Ctrl-c

-3->abort      // retorna ao prompt inicial

-->

```

No Scilab, os comentários sempre começam com os caracteres `//`, como foi mostrado no exemplo anterior.

- A opção `Demos` - permite executar os vários programas de demonstração que acompanham a distribuição Scilab. Na Figura 2.3, são apresentados os programas de demonstração disponíveis no Scilab versão 3.0. É interessante, e muito importante, em um primeiro contato com o programa, executar algumas dessas rotinas de demonstração.

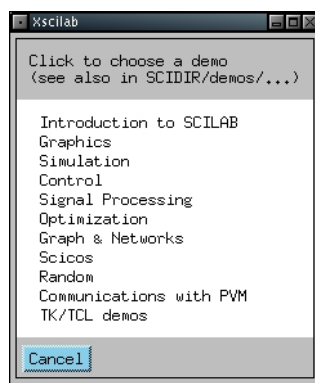


Figura 2.3: Programas de demonstração, opção `Demos`, do Scilab 3.0.

- A opção `Graphics Window N` permite manipular janelas gráficas. Aqui, N representa a janela gráfica que está sendo utilizada. Ao inicializar, Scilab utiliza  $N = 0$ , significando que `Graphics Window 0` é a primeira janela gráfica ou a janela gráfica *default*. Esta opção possui cinco sub-opções:
  - `Set (Create) Window`
  - `Raise (Create) Window`
  - `Delete Graphics Window` - permite apagar uma janela gráfica,
  - `+` - passa para a próxima janela gráfica  $\{N+1\}$ .
  - `-` - retorna para a janela gráfica anterior  $\{N-1\}$ .
- A opção `Help` permite obter informações sobre as diversas funções e comandos do Scilab. Essa opção possui três sub-opções:
  - `Help browser` - ativa o navegador *default* do Scilab. Esse navegador carrega os textos com o *help* das funções implementadas no Scilab, seus *toolboxes* e eventuais *toolboxes* instalados pelo usuário (ver Apêndice C). O navegador de *help* também



pode ser ativada diretamente no *prompt* do Scilab. Usando o *mouse* na sub-opção Help browser temos a tela de `help` mostrada na Figura 2.4 com o navegador padrão do Scilab.

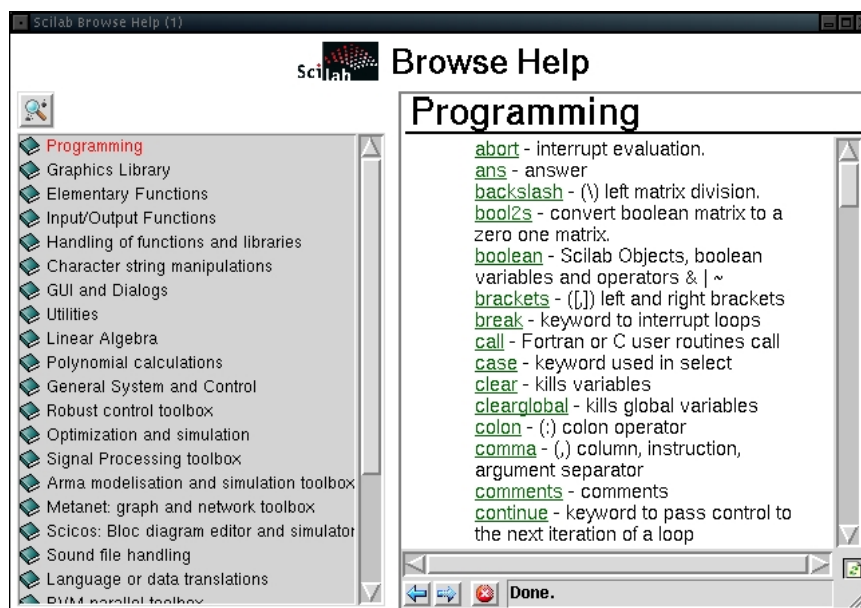


Figura 2.4: Tela da sub-opção Help browser com navegador padrão do Scilab.

- Apropos - ativa uma janela onde pode ser digitada uma palavra chave do assunto sobre o qual se deseja algum tipo de auxílio. Essa opção também pode ser ativada diretamente no *prompt* do Scilab.
- Configure - permite que seja escolhido um outro navegador em substituição ao navegador *default* do *help* do Scilab. A Figura 2.5 mostra as opções de navegadores para a versão 3.0 do Scilab.

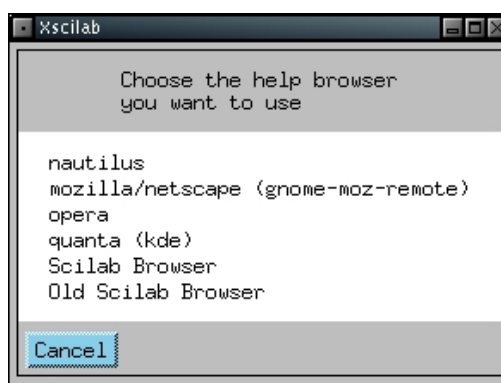


Figura 2.5: Tela de configuração para a escolha do navegador do `Help` do Scilab.

Para Scilab versão 3.0, o *help* está disponível para os seguintes conjuntos de funções:

- Programming - conjunto de comandos que podem ser utilizados na programação com o Scilab;
- Graphic Library - conjunto de comandos gráficos;
- Elementary Functions - conjunto de funções elementares;
- Input/Output Functions - conjunto de funções para entrada e saída de dados;

- Handling of functions and libraries - conjunto de funções para manipulação de funções e bibliotecas;
  - Character string manipulations - conjunto de funções para manipulação de *strings*;
  - GUI and Dialogs - conjunto de funções que permitem a criação de diálogos (menus, por exemplo);
  - Utilities - conjunto de funções com utilidades diversas;
  - Linear Algebra - conjunto de funções usadas em álgebra linear;
  - Polynomial calculations - conjunto de funções usadas em cálculos com polinômios;
  - General System and Control - conjunto de funções na área de controle;
  - Robust control toolbox - conjunto de funções do *toolbox* de controle robusto;
  - Optimization and simulation - biblioteca de funções não-lineares para utilização em otimização e simulação;
  - Signal Processing toolbox - conjunto de funções do *toolbox* de processamento de sinais;
  - Arma modelization and simulation toolbox - conjunto de funções do *toolbox* para modelamento e simulação ARMA-*Autoregressive Moving Average*;
  - Metanet: graph and network toolbox - conjunto de funções do *toolbox* Metanet para análise de grafos;
  - Scicos: Bloc diagram editor and simulator - conjunto de funções para modelagem e simulação de sistemas dinâmicos;
  - Sound file handling - conjunto de funções para manipulação de arquivos de som;
  - Language or data translations - conjunto de funções para conversão de dados entre o Scilab e alguns aplicativos;
  - PVM parallel toolbox - conjunto de funções que permitem o gerenciamento da comunicação com outras aplicações usando máquinas paralelas virtuais;
  - TdCs - conjunto de funções com utilidades diversas;
  - TCL/Tk interface - conjunto de funções que permitem a interface com as linguagens TCL/Tk;
  - Statistic basics - conjunto de funções para cálculos estatísticos;
  - Cumulative Distribution Functions; Inverse, grand - conjunto de funções de distribuição cumulativa, inversa e geradora de números randômicos;
  - Identification - conjunto de funções para tratamento de sistemas discretos;
  - Matlab to Scilab conversion tips - conjunto de funções para a conversão de arquivos de programas Matlab em Scilab.
- A opção **Editor** permite utilizar o editor incorporado ao Scilab, chamado SciPad, para escrever comandos e funções. Na Figura 2.6, apresentamos a tela inicial do editor SciPad.

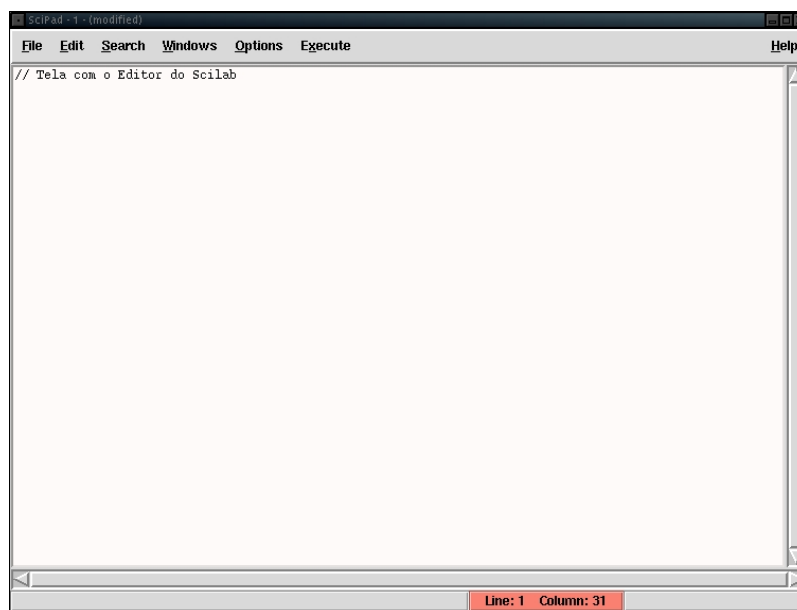


Figura 2.6: Tela inicial do Scipad, editor incorporado ao Scilab.

Algumas funções do editor do Scilab serão exploradas em Capítulos subsequentes.

## 2.3 Variáveis Especiais

Existem variáveis que assumem valores pré-definidos no Scilab. Elas podem ser vistas através do comando `who`. Essas variáveis são protegidas e não podem ser apagadas. Algumas destas variáveis são pré-fixadas com o caracter `%`. A saída do comando `who`, logo após o Scilab 3.0 ter sido inicializado, é mostrada em seguida. É conveniente lembrar que, no *prompt* do Scilab, os comandos são interpretados e executados após o usuário pressionar a tecla **Enter**.

```
-->who
your variables are...

%scipad_fontsize      show_startupinfo    LCC      %toolboxes_dir      %toolboxes
scicos_pal            %scicos_menu        %scicos_short      %scicos_help
%scicos_display_mode      modelica_libs      scicos_pal_libs    with_gtk  with_tk  demolist
%helps  LANGUAGE  SCI      MSDOS  home  PWD      TMPDIR  xdesslib  with_texmacs
percentlib      polylib  intlbr  elembr  utillbr  statslib  alglbr  siglbr  optlbr
autolbr  roblbr  soundlbr  metalbr  armalbr  tkscilbr  tdcslib  s2flbr  mtlbr  %F
%T      %z      %s      %nan    %inf    COMPILER  %gtk    %pvm    %tk    $
%t      %f      %eps    %io     %i      %e
using    15025 elements out of 100000.
and      60 variables out of 9231

your global variables are...

LANGUAGE %helps  demolist %browsehelp      %toolboxes      %toolboxes_dir  LCC
%scipad_fontsize
using    1097 elements out of 11000.
and      8 variables out of 767

-->
```

A variável `%i` representa o resultado de  $\sqrt{-1}$ , `%pi` é a variável que representa  $\pi =$

3,1415926..., e %e é a variável que representa a constante de Euler  $e = 2.7182818\dots$ . Uma outra variável pré-definida é %eps que representa a precisão da máquina na qual Scilab está instalado (%eps é o maior número para o qual  $1+\%eps = 1$ ). São pré-definidas, ainda, as variáveis %inf que significa “Infinito” e %nan que significa “Não é um Número”, *NotANumber*. A variável %s é definida pelo comando `s = poly(0, 's')`. No Scilab são definidas, também, variáveis com valores booleanos: %T significando “verdadeiro” (*true*) e %F significando “falso” (*false*).

Scilab também é carregado com algumas funções pré-definidas, chamadas de funções primitivas ou funções intrínsecas<sup>5</sup>. No Capítulo 3, apresentamos alguns exemplos de utilização dessas funções.

Atenção especial deve ser dada às variáveis SCI e PWD. Elas representam, respectivamente, o diretório no qual o Scilab foi instalado<sup>6</sup> e o diretório no qual o Scilab foi lançado e está rodando. A variável home possui valor idêntico ao da variável PWD.

```
-->SCI          // Diretorio onde Scilab foi instalado
SCI =

/usr/local/scilab-3.0

-->PWD          // Diretorio onde Scilab foi lançado
PWD =

/home/paulo

-->home        // Mesmo valor da variavel PWD
home =

/home/paulo

-->
```

As variáveis pré-definidas e protegidas estão no arquivo de inicialização `SCI/scilab.star`. Se desejar, o usuário pode pré-definir as suas próprias variáveis e, depois, colocá-las no arquivo `.scilab` localizado na sua área de trabalho.

Como mostrado nos exemplos anteriores, os comentários sempre começam com os caracteres `//`. Também, é importante salientar que os comentários (e os nomes das variáveis e funções utilizadas no Scilab) **NÃO** devem ter qualquer tipo de acentuação.

## 2.4 Manipulação de Arquivos e Diretórios

Scilab possui funções que podem ser utilizadas para manipular arquivos e diretórios. A função `pwd`, não confundir com a variável PWD da seção anterior, mostra o diretório no qual estamos trabalhando. Assim,

```
-->pwd          // Mostra o diretorio de trabalho

ans =

/home/paulo
```

<sup>5</sup>Ver Apêndice D para a listagem dessas funções.

<sup>6</sup>Ver Apêndice A.

-->

Usando a função `chdir`, mudamos para o diretório de trabalho `teste`,

```
-->chdir('teste')    // Mudando o diretorio de trabalho
ans =

    0.
```

-->

Uma observação importante: para Scilab, uma resposta igual a 0 (zero) para determinados tipos de comandos indica que a ação foi realizada com sucesso. É o caso da resposta 0 obtida quando do comando `chdir('teste')`.

Por termos mudado de diretório de trabalho, o valor de retorno da função `pwd` foi alterado mas o valor da variável `PWD` permanece inalterada, como podemos verificar pela seqüência de comandos,

```
-->pwd                // Mostrando o novo diretorio de trabalho
ans =

/home/paulo/teste

-->PWD                // PWD permanece inalterado.
PWD =

/home/paulo
```

-->

As variáveis criadas durante os trabalhos realizados no ambiente Scilab podem ser armazenadas em um arquivo. Vamos considerar as variáveis,

```
-->a = 1
a =

    1.
```

```
-->b = 2
b =

    2.
```

-->

Para salvar as variáveis `a` e `b` em um arquivo, que chamamos de `dados.dat`, usamos o comando `save` com a sintaxe

```
-->save('dados.dat',a,b)
```

-->

O comando `save` cria o arquivo `dados.dat` no diretório de trabalho. O arquivo `dados.dat` é um arquivo binário. Para recuperar os valores de `a` e `b`, usamos o comando `load`, conforme mostrado no exemplo,

```
-->clear                // Eliminando as variaveis nao protegidas

-->a
  !--error      4
undefined variable : a

-->b
  !--error      4
undefined variable : b

-->load('dados.dat','a','b')

-->a, b
a =

  1.
b =

  2.

-->
```

Neste exemplo, o comando `clear` elimina todas as variáveis não protegidas do ambiente Scilab. Por esse motivo, as variáveis `a` e `b`, definidas anteriormente, quando chamadas após `clear`, fazem com que Scilab apresente a mensagem de erro `undefined variable`, variável indefinida. Em seguida, através do comando `load`, as variáveis são lidas do arquivo `dados.dat` e retomam seus valores originais, passando novamente a existirem no ambiente Scilab.

A função `unix_w` permite a comunicação do Scilab com a *shell* Linux (Unix). Usando esta função, as respostas são apresentadas na própria janela do Scilab.

```
-->pwd
ans =

/home/paulo/teste

-->unix_w('ls')        // Mostrando o conteudo de /home/paulo/teste
Makefile
Relatorio.pdf
app
app.c
app.o
chromosome.c
chromosome.h
chromosome.o

-->unix_w('mkdir outro_dir') // Criando o diretorio outro_dir
```

```

-->unix_w('ls')
Makefile
Relatorio.pdf
app
app.c
app.o
chromosome.c
chromosome.h
chromosome.o
outro_dir          // outro_dir aparece na listagem

-->chdir('outro_dir')          // Mudando de diretorio
ans =

    0.

-->pwd
ans =

/home/paulo/teste/outro_dir

-->

```

Os comandos digitados a partir do *prompt* do Scilab podem ser editados. Na Tabela 2.1, mostramos algumas combinações de teclas que permitem esta edição.

Ctrl-p ou ↑	recupera o comando digitado anteriormente
Ctrl-n ou ↓	recupera o comando seguinte (se houver)
Ctrl-b ou ←	move o cursor um caracter para trás
Ctrl-f ou →	move o cursor um caracter para a frente
Delete ou ←	apaga o caracter anterior (tecla <i>backspace</i> )
Ctrl-h	mesmo efeito da linha anterior
Ctrl-d	apaga o caracter sob o cursor
Ctrl-a	move o cursor para o início da linha
Ctrl-e	move o cursor para o final da linha
Ctrl-k	apaga da posição do cursor até o final da linha
Ctrl-u	cancela a linha
!prev	recupera a linha de comando que começa com prev

Tabela 2.1: Teclas de edição linhas de comando no *prompt* do Scilab.

## 2.5 O *help* do Scilab

A qualquer momento, o usuário pode obter informações sobre as funções internas do Scilab digitando o comando `help` diretamente no *prompt* ou acessando a sub-opção `Help browser` do menu `Help`, como descrito anteriormente. Por exemplo, vamos usar o comando `help` na linha de comando do Scilab para obter informações sobre a função `det`, que calcula o determinante de uma matriz. Temos, então,

```

-->help

```

O comando `help` ativará o navegador de `help` com um menu contendo as famílias das funções disponíveis, como foi visto na Figura 2.4 anterior. A função que calcula o determinante de uma matriz pertence à família das funções de Álgebra Linear, indicada por `Linear Algebra` na tela da Figura 2.4. Escolhendo-se a função `det` tem-se as informações desejadas. Todo esse processo está resumido na tela apresentada na Figura 2.7.

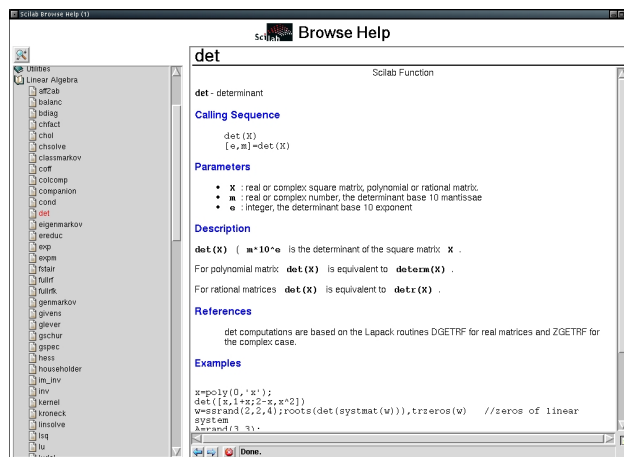


Figura 2.7: Tela de help para a função `det`.

O mesmo efeito é conseguido digitando-se o comando

```
-->help det
```

diretamente no *prompt* do Scilab, como podemos verificar na Figura 2.8

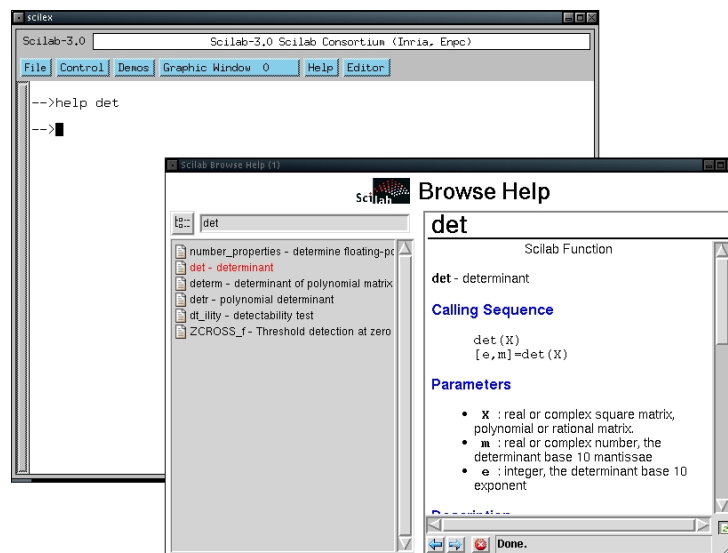


Figura 2.8: Comando `help` para a função `det`.

Outro exemplo, agora com uma função pertencente à família de Funções Elementares, *Elementary Functions*, serve para ilustrar a quantidade (e a qualidade) das informações presentes no `help` do Scilab. Escolhemos a função modificada de Bessel de segunda ordem,  $K_\alpha(x)$ , implementada no Scilab através da função `besselk`, cujo texto de `help` reproduzimos na Figura 2.9.



## Scilab Function

besselk - Modified Bessel functions of the second kind (K sub alpha).

## Calling Sequence

```
y = besselk(alpha,x)
y = besselk(alpha,x,ice)
```

## Parameters

x : real vector with non negative entries  
alpha : real vector with non negative entries regularly spaced with increment equal to one  
alpha=alpha0+(n1:n2)  
ice : integer flag, with default value 1

## Description

besselk(alpha,x) computes modified Bessel functions of the second kind (K sub alpha), for real, non-negative order alpha and argument x . alpha and x may be vectors. The output is m -by- n with m = size(x,'\*') , n = size(alpha,'\*') whose (i,j) entry is besselk(alpha(j),x(i)) .

K\_alpha and I\_alpha (see besseli ) modified Bessel functions are 2 independant solutions of the modified Bessel 's differential equation :

$$x^2 y'' + x y' - (x^2 + \alpha^2) y = 0, \quad \alpha \geq 0$$

If ice is equal to 2 exponentialy scaled Bessel functions is computed (K\_alpha\_scaled(x) = exp(x) K\_alpha(x)).

## Examples

```
// example : display some K bessel functions
x = linspace(0.01,10,5000)';
y = besselk(0:4,x);
ys = besselk(0:4,x,2);
xbasc()
subplot(2,1,1)
plot2d(x,y, style=2:6, leg="K0@K1@K2@K3@K4", rect=[0,0,6,10])
xtitle("Some modified bessel functions of the second kind")
subplot(2,1,2)
plot2d(x,ys, style=2:6, leg="K0s@K1s@K2s@K3s@K4s", rect=[0,0,6,10])
xtitle("Some modified scaled bessel functions of the second kind")
```

## See Also

besselj , besseli , bessely ,

## Author

W. J. Cody, L. Stoltz (code from Netlib (specfun))

Figura 2.9: Texto do help para a função besselk.

Como podemos observar, no texto do *help* estão especificados:

- O nome da função, como implementado pelo Scilab;
- O(s) comando(s) de chamada da função, *Calling Sequence*;
- Os parâmetros da função, *Parameters*;
- Uma descrição da função implementada, *Description*;
- Exemplos de utilização da função, *Examples*;
- Funções relacionadas, *See Also*, e neste caso,
- Autor da função, *Author*.

## 2.6 Arquivos com Comandos Scilab

Como vimos, o comando `save` pode ser utilizado para armazenar variáveis em um arquivo binário. Essas variáveis podem ser recuperadas através da utilização do comando `load`.

Além do armazenamento de variáveis, Scilab permite que os comandos digitados em seu ambiente durante uma sessão sejam armazenados em um arquivo, construindo uma espécie de “memória de cálculos”. O armazenamento dos comandos é feito através da utilização do comando `diary('nome_do_arquivo')`.

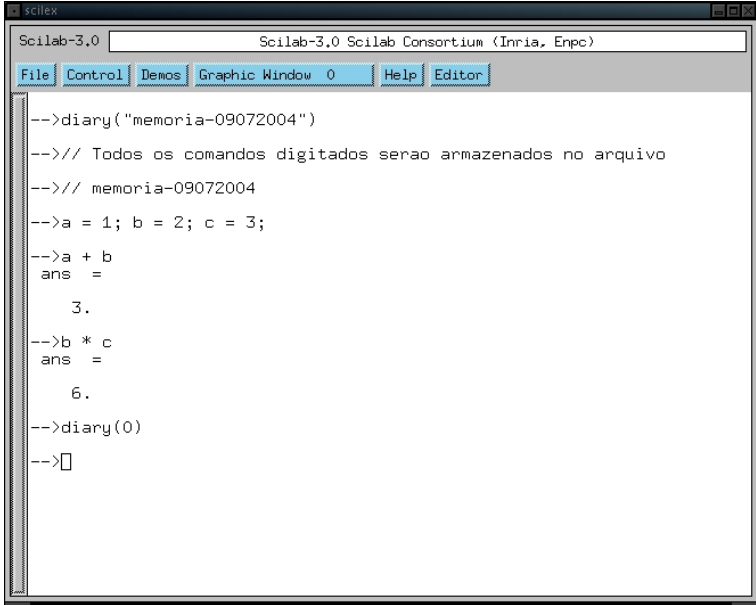
Na Figura 2.10 mostramos um exemplo da utilização do comando `diary` para armazenar uma sessão de utilização do ambiente Scilab. Neste exemplo, através do comando

```
-->diary('memoria-09072004')
```

instruímos o Scilab para armazenar todos os comandos subsequentes em um arquivo chamado `memoria-09072004`. O armazenamento dos comandos será realizado até que seja executado o comando

```
--diary(0)
```

O comando `diary(0)` fecha o arquivo `memoria-09072004`.



```
Scilab-3.0 Scilab-3.0 Scilab Consortium (Inria, Erpc)
File Control Demos Graphic Window 0 Help Editor
-->diary("memoria-09072004")
-->// Todos os comandos digitados serao armazenados no arquivo
-->// memoria-09072004
-->a = 1; b = 2; c = 3;
-->a + b
ans =
    3.
-->b * c
ans =
    6.
-->diary(0)
-->
```

Figura 2.10: Comando `diary` para gravação de sessões desenvolvidas no ambiente Scilab.

O arquivo `memoria-09072004` é um arquivo texto puro,

```
paulo@none:~$ cat memoria-09072004
```

```
-->// Todos os comandos digitados serao armazenados no arquivo
```

```
-->// memoria-09072004
```

```
-->a = 1; b = 2; c = 3;
```

```
-->a + b
```

```
ans =
```

3.

```
-->b * c  
ans =
```

6.

```
-->diary(0)  
paulo@none:~$
```

podendo, portanto, ser editado.

No Capítulo sobre programação, veremos novos tipos de arquivos de comandos do Scilab.

Neste Capítulo, apresentamos as principais características do ambiente Scilab. No próximo Capítulo, exploramos um pouco mais esse ambiente através da realização de algumas operações básicas envolvendo grandezas reais e complexas e da utilização de funções pré-definidas do Scilab.

## Capítulo 3

# Operações Básicas com Scilab

Scilab é um ambiente para resolução de problemas numéricos.

A interação do usuário com o Scilab pode ocorrer de duas formas distintas. Na primeira, os comandos são digitados diretamente no *prompt* do Scilab. Ao ser pressionada a tecla **enter**, os comandos digitados são interpretados e imediatamente executados. Neste modo de utilização, Scilab funciona como uma sofisticada e poderosa calculadora. Na segunda forma, um conjunto de comandos é digitado em um arquivo texto. Este arquivo, em seguida, é levado para o ambiente Scilab e executado. Neste modo, o Scilab funciona como um ambiente de programação.

Neste Capítulo, apresentamos algumas características do ambiente gráfico do Scilab. Através de alguns exemplos de operações que podem ser realizadas em linha de comando, mostramos o Scilab funcionando como uma sofisticada calculadora.

Scilab como ambiente de programação é apresentado no Capítulo 5.

### 3.1 Introdução

No Scilab, o ponto-e-vírgula no final de um comando inibe a apresentação de seu resultado. Alguns exemplos,

```
-->// 0 ponto-e-virgula suprime a apresentacao do resultado

-->A = 1;      // a variavel A assume o valor 1

-->b = 2;      // atribuindo a variavel b o valor 2

-->A + b      // Adicao de A e b
ans =

    3.

-->
```

Uma observação importante: Scilab é *case sensitive*. Assim, por exemplo, a variável `incr` é diferente das variáveis `INCR`, `Incr` ou `INcr`.

As grandezas no Scilab também podem ser complexas. Para atribuir à variável `A` o valor complexo  $5 + 2i$  e à variável `B` o valor complexo  $-2 + i$ , fazemos

```
-->A = 5 + 2 * %i // Atribuindo a A o valor 5 + 2i
A =
```

```

5. + 2.i

-->B = -2 + %i      // Atribuindo a B o valor -2 + i
B =

- 2. + i

-->

```

Observar que a não utilização do ponto-e-vírgula no final dos comandos de atribuição permitiu a apresentação do resultado de cada comando.

As variáveis complexas A e B podem ser multiplicadas, divididas, somadas ou subtraídas, como mostramos a seguir.

```

--> // Operacoes com variaveis complexas

-->A * B          // Multiplicacao
ans =

- 12. + i

-->A / B          // Divisao
ans =

- 1.6 - 1.8i

-->A + B          // Adicao
ans =

3. + 3.i

-->A - B          // Subtracao
ans =

7. + i

-->

```

É importante observar que a resposta ao uso da função interna `sqrt()` com argumento negativo inclui o número complexo  $i = \sqrt{-1}$ . Por exemplo,

```

-->sqrt(-2)      // Funcao raiz quadrada com argumento negativo
ans =

1.4142136i

-->

```

É possível digitar vários comandos em uma mesma linha,

```

-->m = 1.5; b = 35; c = 24;      // Varios comandos em uma unica linha

-->

```

Também é possível desdobrar um único comando em várias linhas utilizando ... ao final do comando. Por exemplo,

```
-->A = 3 * m ^ 2 + ...           // Um comando em varias linhas
-->   4 * 5 + ...
-->   5 * 3
A =

41.75

-->
```

Um vetor de índices possui a forma geral

```
Variavel = valor_inicial:incremento:valor_final
```

Por exemplo, através do comando `I=1:3` atribuímos os valores 1, 2, e 3 à variável I. Quando não especificado, `incremento` é igual a 1. Assim,

```
-->I = 1:3                       // Definindo I como um vetor com 3 posicoes
I =

!  1.   2.   3. !

-->j = 1:2:5                       // Indice j com incremento igual a 2
j =

!  1.   3.   5. !

-->
```

O valor do `incremento` pode ser negativo,

```
-->k = 5:-1:1                     // Definindo k como um vetor com 5 posicoes
k =

!  5.   4.   3.   2.   1. !

-->
```

No Scilab existe o conceito de ambientes definidos via uma hierarquia de *prompts*. Muda-se de ambiente através do comando `pause` ou através de `Ctrl-c`. Todas as variáveis definidas no primeiro ambiente são válidas no novo ambiente. Observar que a mudança de ambiente modifica a forma de apresentação do *prompt*. Este passa a indicar o ambiente no qual estão sendo efetuados os comandos. O retorno ao ambiente anterior dá-se através da utilização dos comandos `resume` ou `return`. Com este tipo de retorno, perde-se as variáveis definidas no ambiente anterior. A utilização de ambientes é importante para a realização de testes.

No exemplo a seguir, atribuímos a `a` o valor 1.5 e, através do comando `pause`, mudamos de ambiente.

```
-->// Definindo a e mudando de ambiente

-->a = 1.5; pause

-1-> // Mudanca no prompt
```

Observar que houve uma mudança no formato do *prompt*. A variável *a*, definida no ambiente anterior, ainda é válida no novo ambiente, como podemos verificar através da seqüência de comandos,

```
-1->a
a =

    1.5

-1->
```

Vamos definir, no novo ambiente, a variável *b* igual a 2.5,

```
-1->// Definindo b no novo ambiente

-1->b = 2.5;

-1->// Mostrando a e b no novo ambiente

-1->a, b
a =

    1.5
b =

    2.5

-1->
```

O retorno ao ambiente anterior usando o comando `resume` faz com que a variável *b* fique indefinida,

```
// Retornando ao ambiente anterior

-1->resume          // Pode ser usado o comando return

--> // Mostrando a e b. Observar que a variavel b foi perdida

-->a, b
a =

    1.5
!--error      4
undefined variable : b
```

O valor da variável *b* pode ser preservado no ambiente original através da seqüência de comandos,

```
-->a = 1.5          // Definindo a variavel a no ambiente original
a =

    1.5
```

```

-->pause           // Mudando de ambiente

-1->b = 1.5        // Definindo a variavel b no novo ambiente
b =

1.5

-1->b = resume(b) // Enviando b para o ambiente original

-->a, b
a =

1.5
b =

1.5

-->

```

### 3.2 Utilizando as Funções Internas do Scilab

O Scilab é carregado com algumas funções pré-definidas<sup>1</sup>. Como vimos no Capítulo anterior, na Figura 2.9, o *help* do Scilab explica cada uma delas e, também, apresenta exemplos de sua utilização.

Uma maneira de verificar a forma de utilização e o comportamento de uma determinada função interna do Scilab é usando o exemplo que o próprio *help* do Scilab apresenta. Neste caso, basta copiar o exemplo de uso da função apresentado no *help* para o ambiente do Scilab<sup>2</sup>. Na Figure 3.1, mostramos como esse procedimento funciona usando a função *fft* do *toolbox* de Processamento de Sinais, *Signal Processing toolbox*, que acompanha o Scilab.

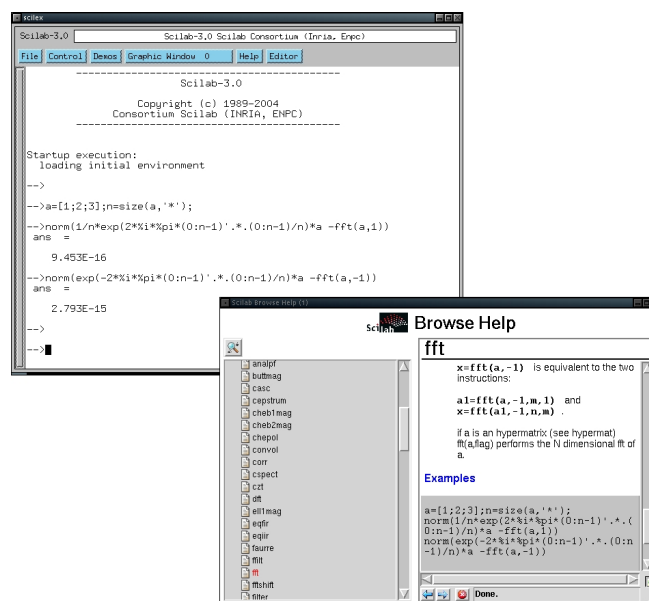


Figura 3.1: Rodando o exemplo de utilização da função *fft* apresentado no *help* do Scilab.

<sup>1</sup>Ver Apêndice D para a listagem dessas funções.

<sup>2</sup>Especificamente, o processo consiste em copiar do ambiente *help* e colar no ambiente Scilab.



Os exemplos apresentados no *help* também podem ser executados através de uma facilidade implementada no editor SciPad, incorporado ao Scilab. Neste caso, o exemplo deve ser copiado do ambiente *help* e colado no ambiente do Editor. Depois, no Editor, o exemplo deve ser selecionado, através do *mouse*, para ser executado usando a sub-opção Evaluate Selection Ctrl+y da opção **Execute** apresentada no menu do Editor. Esses procedimentos, e seus resultados, com os exemplos fornecidos pelo *help* para a função `besselk`, apresentados na Figura 2.9, são mostrados na Figura 3.2.

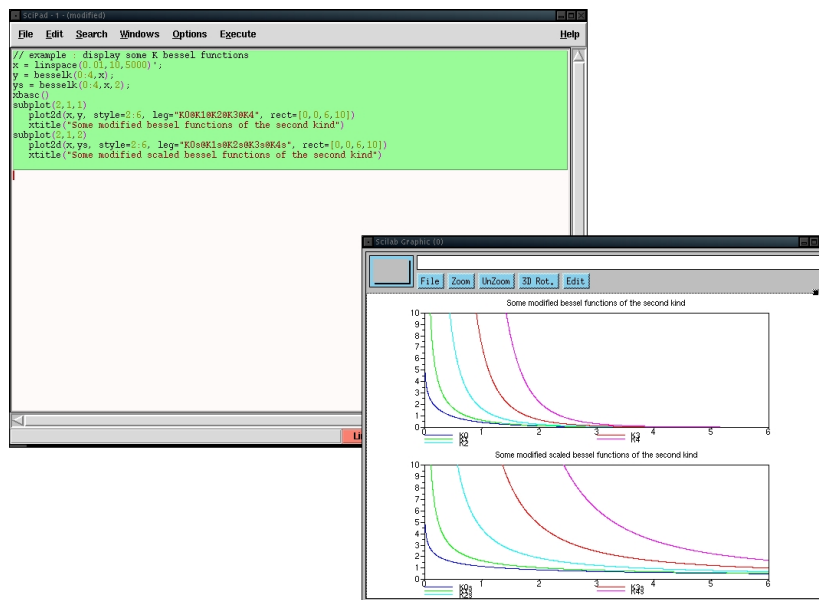


Figura 3.2: A função K de Bessel, `besselk`. O exemplo apresentado no *help* da função é copiado para o editor SciPad, selecionado e executado através da sub-opção Evaluate Selection Ctrl+y da opção **Execute**.

O usuário interessado é convidado a repetir os procedimentos apresentados nessa sessão utilizando outras funções do Scilab.

Neste Capítulo, apresentamos algumas operações básicas envolvendo grandezas reais e complexas e exemplos de utilização de funções pré-definidas no Scilab. No Capítulo 4, mostramos os outros tipos de dados que podem ser manipulados pelo Scilab.

## Capítulo 4

# Polinômios, Vetores, Matrizes e Listas

No Scilab, podemos trabalhar com vários tipos de dados. As constantes, reais ou complexas, as variáveis booleanas, os polinômios, as *strings* e as frações envolvendo polinômios são considerados dados escalares. Com estes objetos podemos definir vetores e matrizes. Os outros tipos de dados reconhecidos pelo Scilab são as listas e as listas com definição de tipo. O objetivo deste Capítulo é apresentar alguns exemplos de utilização de cada um desses tipos de dados.

### 4.1 Polinômios

Os polinômios são criados no Scilab através da utilização da função `poly`. Salientamos que polinômios de mesma variável podem ser somados, subtraídos, multiplicados e divididos entre si. Por exemplo, o polinômio  $p = s^2 - 3s + 2$ , que possui raízes 1 e 2, pode ser criado através do comando,

```
-->// Polinomio definido pelas suas raizes
```

```
-->p = poly([1 2], 's')
```

```
p =
```

$$s^2 - 3s + 2$$

```
-->
```

Com a função `roots`, comprovamos que as raízes de `p` são, realmente, 1 e 2,

```
-->roots(p)
```

```
ans =
```

```
!  1.  !
```

```
!  2.  !
```

```
-->
```

Um polinômio também pode ser criado a partir da especificação de seus coeficientes. Por exemplo, o polinômio  $q = 2s + 1$  é criado através do comando,

```
-->// Polinomio definido pelos seus coeficientes
```

```

-->q = poly([1 2], 's', 'coeff')
q =

    1 + 2s

-->roots(q)      // Obtendo as raizes do polinomio q
ans =

    - 0.5

-->

```

Para complementar o exemplo, os dois polinômios podem ser multiplicados, divididos, somados ou subtraídos como mostra a seqüência de comandos,

```

-->p * q          // Multiplicacao
ans =

    2 + s2 - 5s + 2s3

-->p / q          // Divisao
ans =

    2 - 3s + s2
-----
    1 + 2s

-->[r, q] = pdiv(p,q) // Efetuando a divisao: q=quociente, r=resto
q =

    - 1.75 + 0.5s
r =

    3.75

-->p + q          // Adicao
ans =

    3 - s + s2

-->p - q          // Subtracao
ans =

    1 - 5s + s2

-->

```

Para obter valores de polinômios, usamos a função **horner**,

```

-->x = poly(0, 'x')
x =

      x

-->p = x^2 - 3*x + 5    // definindo o polinomio
p =

      2
    5 - 3x + x

-->horner(p, 2)        // avaliando o polinomio em x = 2
ans =

    3.

-->

```

## 4.2 Vetores

Vamos considerar  $\mathbb{R}$  o conjunto dos números reais<sup>1</sup>. Dizemos que  $\mathbf{x}$  é um vetor de dimensão  $n$  em  $\mathbb{R}$ , indicado por  $\mathbf{x} \in \mathbb{R}^n$ , se, e somente se,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Nessa definição, cada um dos elementos do vetor  $\mathbf{x}$ ,  $x_i$ , pertence a  $\mathbb{R}$ ,

$$x_i \in \mathbb{R}$$

O elemento  $x_i$  é o  $i$ -ésimo elemento do vetor  $\mathbf{x}$ .

O vetor  $\mathbf{x}$  definido anteriormente é um vetor coluna. Para explicitar esta condição, escrevemos

$$\mathbf{x} \in \mathbb{R}^{n \times 1}$$

Essa notação indica que o vetor  $\mathbf{x}$  possui  $n$  linhas e apenas uma coluna.

No Scilab, os vetores são criados colocando-se seus componentes entre colchetes, [ ]. Os elementos de um vetor coluna são separados por ponto-e-vírgula. Assim,

```

-->x = [ 1; 2; 3]      // vetor coluna. Elementos separados por ;
x =

!  1. !
!  2. !
!  3. !

-->

```

---

<sup>1</sup>Todas as considerações sobre vetores e matrizes podem ser estendidas para o conjunto dos números complexos,  $\mathbb{C}$ .

Um vetor linha,  $\mathbf{y}$ , de dimensão  $n$  em  $\mathbb{R}$  pode ser escrito na forma

$$\mathbf{y} = [y_1, y_2, \dots, y_n]$$

Para explicitar a condição de vetor linha, escrevemos

$$\mathbf{y} \in \mathbb{R}^{1 \times n}$$

Essa notação indica que o vetor  $\mathbf{y}$  possui apenas uma linha e  $n$  colunas.

No Scilab, os componentes de um vetor linha são separados por espaço ou por vírgula.

```
-->y = [ 1 2 3]           // vetor linha; Elementos separados por espaço
y =
```

```
!  1.   2.   3. !
```

```
-->z = [ 4, 5, 6]       // vetor linha; Elementos separados por virgula
z =
```

```
!  4.   5.   6. !
```

```
-->
```

Se  $\mathbf{x}$  é um vetor coluna,  $\mathbf{x}^T$  (lê-se “x transposto”) é um vetor linha. Essa operação é realizada no Scilab através da utilização do símbolo ‘ (apóstrofo).

```
-->x = [1; 2; 3]        // vetor coluna
x =
```

```
!  1. !
!  2. !
!  3. !
```

```
-->x'                   // x transposto = vetor linha
ans =
```

```
!  1.   2.   3. !
```

```
-->
```

Vetores podem ser multiplicados ou divididos por quantidades escalares. Também, vetores de mesma dimensão podem ser somados ou subtraídos. Para exemplificar algumas dessas operações, vamos considerar os vetores:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad \text{e} \quad \mathbf{y} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

Observar que os dois vetores possuem a mesma dimensão, isto é,  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{3 \times 1}$ . Temos,

```
-->x = [ 1; 2; 3]       // Definindo o vetor x
x =
```

```
!  1. !
```

```

! 2. !
! 3. !

-->y = [ 4; 5; 6]      // Definindo o vetor y
y =

! 4. !
! 5. !
! 6. !

-->size(x)           // Dimensao do vetor x
ans =

! 3. 1. !

-->size(y)           // Dimensao do vetor y
ans =

! 3. 1. !

-->

-->3 * x             // Multiplicando o vetor x por uma constante
ans =

! 3. !
! 6. !
! 9. !

-->x / 2             // Dividindo o vetor x por uma constante
ans =

! 0.5 !
! 1. !
! 1.5 !

-->x + y             // Somando os dois vetores
ans =

! 5. !
! 7. !
! 9. !

-->

```

Dados dois vetores de mesma dimensão,  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{n \times 1}$ , define-se o produto escalar ou produto interno entre  $\mathbf{x}$  e  $\mathbf{y}$  através da expressão vetorial,

$$\mathbf{z} = \mathbf{x}^T \mathbf{y}$$

Assim, considerando os vetores  $\mathbf{x}$  e  $\mathbf{y}$  definidos anteriormente, temos:

```

-->z = x' * y        // Atribuindo a z o produto escalar entre x e y
z =

```

32.

--&gt;

Observar que essa operação, em uma linguagem convencional, teria que ser realizada através de uma rotina que implementasse a operação (escalar):

$$z = \sum_{i=1}^n x_i y_i$$

Se os vetores  $\mathbf{x}$  e  $\mathbf{y}$  possuem dimensões diferentes, isto é,  $\mathbf{x} \in \mathbb{R}^{m \times 1}$  e  $\mathbf{y} \in \mathbb{R}^{n \times 1}$ , podemos definir o produto vetorial ou produto externo entre eles através da expressão,

$$\mathbf{C} = \mathbf{xy}^T$$

Vamos considerar

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad \text{e} \quad \mathbf{y} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

Observar que os dois vetores possuem dimensões diferentes, isto é,  $\mathbf{x} \in \mathbb{R}^{3 \times 1}$  e  $\mathbf{y} \in \mathbb{R}^{2 \times 1}$ . Temos,

```
-->x = [1; 2; 3]      // Definindo o vetor x
x =

!  1.  !
!  2.  !
!  3.  !

-->y = [4; 5]        // Definindo o vetor y
y =

!  4.  !
!  5.  !

-->size(x)          // Dimensao do vetor x
ans =

!  3.   1.  !

-->size(y)          // Dimensao do vetor y
ans =

!  2.   1.  !

-->size(y')         // Dimensao do vetor y transposto
ans =

!  1.   2.  !
```

```

-->C = x * y'          // Produto vetorial de x por y
C =

!  4.    5.  !
!  8.    10. !
!  12.   15. !

-->

```

Nos exemplos a seguir, mostramos outras maneiras de construir vetores, usando índices e algumas funções internas do Scilab:

```

-->v = 5: -0.5: 3      // Vetor com elementos decrementados

v =

!  5.    4.5    4.    3.5    3.  !

-->m = ones(1:4)      // Vetor constituído de elementos iguais a 1
m =

!  1.    1.    1.    1.  !

-->z = zeros(1:5)     // Vetor constituído de elementos iguais a 0
z =

!  0.    0.    0.    0.    0.  !

-->

```

### 4.3 Matrizes

Seja  $\mathbb{R}$  o conjunto dos números reais. Dizemos que  $A$  é uma matriz de dimensão  $m \times n$  em  $\mathbb{R}$ , indicado por  $A \in \mathbb{R}^{m \times n}$ , se, e somente se,

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

onde cada um dos elementos  $a_{i,j} \in \mathbb{R}$ . Nessa notação, a variável  $m$  indica o número de linhas e a variável  $n$  indica o número de colunas da matriz  $A$ . Se  $A$  for uma matriz quadrada, o número de linhas é igual ao número de colunas e, então,  $m = n$ .

Vamos considerar as matrizes  $A, B \in \mathbb{R}^{2 \times 3}$ ,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 5 & -8 & 9 \end{bmatrix} \quad \text{e} \quad B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

No Scilab, as matrizes são representadas entre colchetes, [ ]. Os elementos que constituem as linhas das matrizes são separados por espaços ou por vírgulas. A indicação de término de cada linha da matriz é feita com ponto-e-vírgula.

Nos exemplos a seguir, para fixar conceitos, a matriz  $A$  é digitada com os elementos de suas linhas separados por espaços enquanto a matriz  $B$  é digitada com os elementos de suas linhas separados por vírgula. Assim,



```

-->// Matriz A - Elementos das linhas separados por espaco
-->A = [1 2 3; 5 -8 9]
A =

!  1.   2.   3. !
!  5.  -8.   9. !

-->// Matriz B - Elementos das linhas separados por virgulas
-->B = [1, 2, 3; 4, 5, 6]
B =

!  1.   2.   3. !
!  4.   5.   6. !

-->size(A)           // Dimensao da matriz A
ans =

!  2.   3. !

-->size(B)           // Dimensao da matriz B
ans =

!  2.   3. !

-->

```

Uma outra forma de digitar matrizes no ambiente Scilab, é separando os elementos de uma linha por espaço (ou por vírgula) e as linhas separadas por `enter`,

```

-->M = [ 1 2 3 4
-->5 6 7 8
-->9 11 13 15]
M =

!  1.   2.   3.   4. !
!  5.   6.   7.   8. !
!  9.  11.  13.  15. !

-->

```

Matrizes podem ser multiplicadas ou divididas por quantidades escalares. Também, matrizes de mesma dimensão podem ser somadas ou subtraídas. Considerando as matrizes  $A$  e  $B$  do exemplo anterior, temos:

```

-->2 * A           // Multiplicacao por um escalar
ans =

!  2.   4.   6. !
! 10. -16.  18. !

```

```

-->A / 2          // Divisao da matriz A por uma constante
ans =

!  0.5   1.   1.5 !
!  2.5  -4.   4.5 !

-->A + B          // Somando as duas matrizes
ans =

!  2.   4.   6. !
!  9.  -3.  15. !

-->

```

Se  $A \in \mathbb{R}^{m \times n}$ , a transposta da matriz  $A$ , indicada por  $A^T$ , é tal que  $A^T \in \mathbb{R}^{n \times m}$ . Como no caso dos vetores, a transposição é indicada pelo símbolo ' (apóstrofo).

Considerando a matriz  $B$  do exemplo anterior, temos:

```

-->B = [1, 2, 3; 4, 5, 6]
B =

!  1.   2.   3. !
!  4.   5.   6. !

-->size(B)        // Dimensao da matriz B
ans =

!  2.   3. !

-->C = B'         // C = transposta da matriz B
C =

!  1.   4. !
!  2.   5. !
!  3.   6. !

-->size(C)        // Dimensao da matriz C
ans =

!  3.   2. !

-->

```

Se  $A \in \mathbb{R}^{m \times p}$  e  $B \in \mathbb{R}^{p \times n}$ , podemos definir o produto das matrizes  $A$  e  $B$ ,

$$C = A \times B \in \mathbb{R}^{m \times n}$$

Observar que, para que possa haver a multiplicação entre duas matrizes, é necessário que o número de colunas da primeira matriz seja igual ao número de linhas da segunda matriz.

Considerando as matrizes  $A$  e  $B$ ,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad \text{e} \quad B = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

temos:

```
-->A = [ 1 2 3; 4 5 6; 7 8 9]
```

```
A =
```

```
!  1.   2.   3.  !
!  4.   5.   6.  !
!  7.   8.   9.  !
```

```
-->B = [ 1 4; 2 5; 3 6]
```

```
B =
```

```
!  1.   4.  !
!  2.   5.  !
!  3.   6.  !
```

```
-->size(A)
```

```
ans =
```

```
!  3.   3.  !
```

```
-->size(B)
```

```
ans =
```

```
!  3.   2.  !
```

```
-->A * B
```

```
ans =
```

```
!  14.   32.  !
!  32.   77.  !
!  50.   122.  !
```

```
-->
```

Podemos usar funções internas do Scilab para gerar matrizes. Por exemplo, usamos a função `ones` para criar a matriz  $D \in \mathbb{R}^{2 \times 3}$ , com todos os elementos iguais a 1,

```
-->D = ones(2,3)
```

```
D =
```

```
!  1.   1.   1.  !
!  1.   1.   1.  !
```

```
-->
```

ou a função `zeros` para criar a matriz  $E \in \mathbb{R}^{3 \times 3}$ , com todos os elementos iguais a 0,

```
-->E = zeros(3,3)
```

```
E =
```

```
!  0.   0.   0.  !
```

```
!  0.  0.  0.  !
!  0.  0.  0.  !
```

```
-->
```

ou, ainda, a criação de uma matriz identidade, I através da função interna `eye`,

```
-->I = eye(4,4)
I =
```

```
!  1.  0.  0.  0.  !
!  0.  1.  0.  0.  !
!  0.  0.  1.  0.  !
!  0.  0.  0.  1.  !
```

```
-->
```

Podemos criar matrizes a partir de elementos de outras matrizes,

```
-->// Definido as matrizes A, B e C
```

```
-->A = [1 2; 3 4];
```

```
-->B = [5 6; 7 8];
```

```
-->C = [9 10; 11 12];
```

```
-->// Definindo a matriz D
```

```
-->D = [A B C]
D =
```

```
!  1.  2.  5.  6.  9.  10.  !
!  3.  4.  7.  8.  11. 12.  !
```

```
-->// Definindo uma matriz E a partir dos elementos de D
```

```
-->E = matrix(D,3,4)
E =
```

```
!  1.  4.  6.  11.  !
!  3.  5.  8.  10.  !
!  2.  7.  9.  12.  !
```

```
-->
```

Observar que a matriz E, com três linhas e quatro colunas, é criada usando a função `matrix`. Esta função gera a matriz E a partir da organização dos elementos da matriz D por colunas.

#### 4.4 Acesso a Elementos de Vetores e de Matrizes

O acesso a elementos de um vetor ou de uma matriz pode ser realizado de diversas maneiras. Dentre elas, podemos citar:

- a utilização explícita dos índices do elemento a ser acessado,
- a utilização do símbolo : (dois pontos)
- a utilização do símbolo \$ ou
- a utilização de operações booleanas.

Vamos considerar o vetor linha  $v = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7]$ . O acesso a um elemento deste vetor é feito de forma convencional, o índice do vetor indicando qual elemento que está sendo acessado. Assim,

```
-->v = [1 2 3 4 5 6 7]      // definicao do vetor v
v =
```

```
!  1.   2.   3.   4.   5.   6.   7. !
```

```
-->v(1)      // acesso ao primeiro elemento de v
ans =
```

```
1.
```

```
-->v(5)      // acesso ao quinto elemento de v
ans =
```

```
5.
```

```
-->
```

O símbolo : permite definir formas compactas de acesso a elementos de um vetor. Por exemplo,

```
-->v(2:4)      // acesso aos elementos 2, 3 e 4 de v
ans =
```

```
!  2.   3.   4. !
```

```
-->v(:)        // acesso a todos os elementos de v
ans =
```

```
!  1. !
```

```
!  2. !
```

```
!  3. !
```

```
!  4. !
```

```
!  5. !
```

```
!  6. !
```

```
!  7. !
```

```
-->v(1:2:7)    // acesso aos elementos impares de v
ans =
```

```
!  1.   3.   5.   7. !
```

```
-->
```

enquanto o símbolo \$ permite acessar o último elemento do vetor,

```
-->v($)          // acesso ao ultimo elemento de v
ans =
```

7.

```
-->
```

Também, podemos utilizar operações booleanas para acessar elementos de um vetor. Na sessão,

```
-->v([%f %t %f %t %t]) // acesso usando %t e %f
ans =
```

```
!  2.   4.   5. !
```

```
-->
```

acessamos o segundo, quarto e quinto elemento do vetor v. Lembrar que %t significa *true*, verdadeiro, e que %f significa *false*, falso.

Para exemplificar acessos a elementos de matrizes, vamos considerar a matriz A com duas linhas e três colunas,  $A \in \mathbb{R}^{2 \times 3}$ ,

```
-->// Definindo uma matriz A
```

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

```
!  1.   2.   3. !
!  4.   5.   6. !
```

```
-->
```

O acesso a um elemento dessa matriz é feito da maneira convencional: o elemento da linha  $i$  e coluna  $j$ ,  $a_{i,j}$ , é acessado através do comando  $A(i,j)$ , com  $i$  e  $j$  tendo seus valores numéricos explicitados. Por exemplo, para acessar o elemento  $a_{1,2}$  da matriz A, usamos o comando  $A(1,2)$ ,

```
-->// Acessando o elemento da primeira linha e segunda coluna de A
```

```
-->A(1,2)
ans =
```

2.

```
-->
```

O comando  $M = A([1 \ 2], 2)$ , permite construir uma matriz, M, composta pelo primeiro e segundo elementos, indicados pelo vetor  $[1 \ 2]$ , da segunda coluna da matriz A,

```
-->M = A([1 2], 2)
M =
```

```
! 2. !
! 5. !
```

```
-->
```

Através do operador `:` Scilab implementa formas compactas que permitem acessar elementos de uma matriz. Considerando  $A \in \mathbb{R}^{m \times n}$ , a notação  $A(k, :)$  representa a  $k$ -ésima linha da matriz  $A$ ,

$$A(k, :) = [a_{k,1}, a_{k,2}, \dots, a_{k,n}]$$

e a notação  $A(:, k)$  representa a  $k$ -ésima coluna da matriz  $A$ ,

$$A(:, k) = [a_{1,k}, a_{2,k}, \dots, a_{m,k}]$$

Nesse contexto, e para facilitar a compreensão, o símbolo `:` (dois pontos) assume o significado de “todos os elementos”. Assim,  $A(k, :)$  pode ser lido como “todos os elementos da  $k$ -ésima linha da matriz  $A$ ” e  $A(:, k)$  pode ser lido como “todos os elementos da  $k$ -ésima coluna da matriz  $A$ ”.

Considerando a matriz  $A$  do exemplo anterior, o comando `A(:,3)`, permite acessar todos os elementos da terceira coluna da matriz  $A$ ,

```
->// Todos os elementos da terceira coluna da matriz A
```

```
-->A(:, 3)
```

```
ans =
```

```
! 3. !
! 6. !
```

```
-->
```

enquanto o comando `A(2,:)` permite acessar todos os elementos da segunda linha da matriz  $A$ ,

```
->// Todos os elementos da segunda linha da matriz A
```

```
-->A(2,:)
```

```
ans =
```

```
! 4. 5. 6. !
```

```
-->
```

O comando `A(:, 3:-1:1)` permite formar uma matriz constituída por todos os elementos das colunas três, dois e um da matriz  $A$ . Lembrar que `3:-1:2` é idêntico ao vetor `[3 2 1]`.

```
-->// Todos os elementos da terceira, segunda e primeira colunas de A
```

```
-->A(:, 3:-1:1)
```

```
ans =
```

```
! 3. 2. 1. !
! 6. 5. 4. !
```

```
-->A(:, [3 2 1]) // Forma equivalente
```

```
ans =
```

```
! 3. 2. 1. !
! 6. 5. 4. !
```

```
-->
```

Vamos considerar a utilização do símbolo \$ para acessar elementos da matriz A. Neste contexto, o símbolo \$ significa “número total de”. Usando o comando `A(1:2, $-1)`, acessamos o primeiro e o segundo elementos, indicados por 1:2, da segunda coluna, indicado por \$-1, da matriz A. Lembrar que a matriz A possui duas linhas e três colunas. Com o comando, `A($:-1:1, 2)`, estamos acessando o segundo e o primeiro, nessa ordem, elementos da segunda coluna da matriz A. Escrever `$:-1:1` é equivalente, neste caso, a escrever `2:-1:1` já que a matriz A possui duas linhas.

```
-->// Primeiro e segundo elementos da segunda coluna de A
```

```
-->A(1:2, $-1)
```

```
ans =
```

```
! 2. !
! 5. !
```

```
-->// Segundo e primeiro elementos da segunda coluna de A
```

```
-->A($:-1:1, 2)
```

```
ans =
```

```
! 5. !
! 2. !
```

```
-->// Acesso ao ultimo elemento de A
```

```
-->A($)
```

```
ans =
```

```
6.
```

```
-->
```

Os elementos de uma matriz são armazenados por coluna. Daí, usando o comando `A($)` na sessão anterior, acessamos o último elemento de A. Assim, o primeiro elemento da matriz A pode ser acessado através do comando `A(1)` e o quinto elemento da matriz A pode ser acessado através do comando `A(5)`,

```
-->// Primeiro elemento de A
```

```
-->A(1)
```

```
ans =
```

```
1.
```

```
-->// Quinto elemento de A
```



```

-->A(5)
ans =

    3.

-->// Todos os elementos armazenados por coluna

-->A(:)
ans =

!   1. !
!   4. !
!   2. !
!   5. !
!   3. !
!   6. !

--> // Mesmo efeito do comando anterior

-->A([1 2 3 4 5 6])
ans =

!   1. !
!   4. !
!   2. !
!   5. !
!   3. !
!   6. !

-->

```

Podemos usar variáveis booleanas para acessar elementos de uma matriz. Com o comando `A([%t %f %f %t])`, acessamos o primeiro e o quarto elementos da matriz A, indicados por %t, não querendo o segundo e terceiro elementos, indicados por %f.

```

-->// Acesso ao primeiro e quarto elementos

```

```

-->A([%t %f %f %t])
ans =

!   1. !
!   5. !

-->

```

Com o comando `A(%t, [2 3])`, acessamos os primeiros elementos das segunda e terceira colunas.

```

-->// Acessando os primeiros elementos da colunas 2 e 3

--> A(%t, [2 3])

```

```

ans =

!  2.   3. !

-->

```

É possível, caso seja necessário, alterar os valores de elementos de uma matriz. Considerando a matriz A, podemos mudar o valor do seu elemento A(2,1) através do comando de atribuição A(1,2) = 10,

```

-->// Atribuir a A(1,2) o valor 10

-->A(1,2) = 10
A =

!  1.   10.   3. !
!  4.    5.   6. !

-->

```

Depois, atribuímos os valores [-1; -2] aos primeiro e segundo elementos da segunda coluna da matriz A,

```

-->// A(1,2) = -1 e A(2,2) = -2

-->A([1 2], 2) = [-1; -2]
A =

!  1.  - 1.   3. !
!  4.  - 2.   6. !

-->

```

Finalmente, modificamos os elementos A(1,1) e A(1,2) da matriz A.

```

-->// A(1,1) = 8 e A(1,2) = 5

-->A(:,1) = [8;5]
A =

!  8.  - 1.   3. !
!  5.  - 2.   6. !

-->

```

## 4.5 Matrizes com Polinômios

Os elementos de uma matriz podem ser polinômios,

```

-->// Definindo um polinomio

-->x = poly(0, 'x'); p = 2 + 3 * x + x ^ 2

```

p =

$$2 + 3x + x^2$$

-->// Definindo uma matriz polinomial, M

-->M = [p, p-1; p+1, 2]

M =

$$\begin{bmatrix} 2 + 3x + x^2 & 1 + 3x + x^2 \\ 3 + 3x + x^2 & 2 \end{bmatrix}$$

-->// Avaliando a matriz M em x = 2

-->horner(M, 2)

ans =

$$\begin{bmatrix} 12. & 11. \\ 13. & 2. \end{bmatrix}$$

-->// Obtendo a inversa de M

-->inv(M)

ans =

$$\begin{bmatrix} \frac{-1 - 3x - x^2}{1 - 6x^2 - 11x^3 - 6x^4} & \frac{2 + 3x + x^2}{1 - 6x^2 - 11x^3 - 6x^4} \\ \frac{-3 - 3x - x^2}{1 - 6x^2 - 11x^3 - 6x^4} & \frac{2 + 3x + x^2}{1 - 6x^2 - 11x^3 - 6x^4} \end{bmatrix}$$

-->// Obtendo o determinante de M

-->det(M)

ans =

$$1 - 6x^2 - 11x^3 - 6x^4$$

-->

A partir de uma matriz formada por elementos que são polinômios racionais,

```
-->// Definindo uma matriz F de polinomios racionais
```

```
-->s = poly(0, 's');
```

```
-->F = [ 1/s,      (s + 1)/(s + 2); ...
```

```
-->      s/(s+3),      s^2      ]
```

```
F =
```

```
!  1      1 + s  !
!  -      ----- !
!  s      2 + s  !
!                    !
!                    2  !
!      s      s      !
!  -----  -      !
!  3 + s    1      !
```

```
-->
```

podemos criar outra matriz apenas com o numerador das frações,

```
-->F('num') // Pegando os numeradores
```

```
ans =
```

```
!  1      1 + s  !
!                    !
!                    2  !
!  s      s      !
```

```
-->
```

ou com seus denominadores,

```
-->F('den') // Pegando os denominadores
```

```
ans =
```

```
!  s      2 + s  !
!                    !
!  3 + s    1      !
```

```
-->
```

## 4.6 Matrizes Simbólicas

O Scilab permite a criação e manipulação de matrizes simbólicas. Vamos considerar uma matriz  $B \in \mathbb{R}^{1 \times 2}$ , constituída por elementos simbólicos,

```
-->// Matriz simbolica
```

```
-->B = [ 1/%s, (%s + 1)/(%s - 1)]
```

```

B =

!  1      1 + s  !
!  -      ----- !
!  s      - 1 + s  !

```

```
-->
```

Os elementos de uma matriz simbólica são acessados utilizando os mesmos comandos para acessar elementos de uma matriz numérica. Nos dois comandos seguintes, apresentamos exemplos de acesso aos elementos da matriz B,

```
-->// Acessos a elementos da matriz B
```

```
-->B(1,1)
```

```
ans =
```

```

1
-
s

```

```
-->B(1, $)
```

```
ans =
```

```

1 + s
-----
- 1 + s

```

```
-->
```

Podemos, também, atribuir valores simbólicos a elementos de uma matriz, Considerando a matriz  $A = [1 \ -1 \ 3; \ 5 \ -2 \ 6]$ , temos,

```
-->A(1,1) = %s      // Atribuicao do valor simbolico s ao elemento A(1,1)
```

```
A =
```

```

!  s  - 1    3  !
!                    !
!  5  - 2    6  !

```

```
-->A($) = %s + 1    // Atribuindo s + 1 ao ultimo elemento de A
```

```
A =
```

```

!  s  - 1    3    !
!                    !
!  5  - 2    1 + s !

```

```
-->
```

As matrizes simbólicas também podem ser constituídas por elementos compostos por *strings* de caracteres. Elas são criadas da mesma maneira que as matrizes com elementos numéricos. As *strings* são escritas entre apóstrofes ou entre aspas.

```

-->// Matriz de strings

-->A = ['x' 'y'; 'z' 'w+v']
A =

!x y !
! !
!z w+v !

-->// Atribuindo valores

-->x=1;y=2;z=3;w=4;v=5;

// Obtendo o valor numerico dos elementos de A

-->evstr(A)
ans =

! 1. 2. !
! 3. 9. !

-->

```

## 4.7 Matrizes Booleanas

Matrizes booleanas são matrizes construídas com as constantes %t (t é *true*, verdadeiro) e %f (f é *false*, falso). Alguns exemplos de construção matrizes booleanas,

```

-->// Matriz booleana A

-->A = [%t, %f, %t, %f, %f, %f]
A =

! T F T F F F !

-->// Matriz booleana B

-->B = [%t, %f, %t, %f, %t, %t]
B =

! T F T F T T !

-->

```

Podemos realizar operações lógicas com as matrizes definidas anteriormente,

```

-->// A ou B

-->A|B
ans =

! T F T F T T !

```

```
-->// A e B

-->A & B
ans =

! T F T F F F !

-->
```

### 4.8 Operações com Vetores e Matrizes

A Tabela 4.1, apresenta a sintaxe de alguns dos operadores disponíveis no ambiente Scilab que podem ser utilizados em operações com vetores ou com matrizes.

SÍMBOLO	OPERAÇÃO
'	transposta
+	adição
-	subtração
*	multiplicação
/	divisão à direita
\	divisão à esquerda
^	exponenciação
.*	multiplicação elemento-a-elemento
.\	divisão, à esquerda, elemento-a-elemento
./	divisão, à direita, elemento-a-elemento
.^	exponenciação elemento-a-elemento
.*.	produto de Konecker

Tabela 4.1: Sintaxe de alguns operadores usados em operações vetoriais ou matriciais.

As operações envolvendo os operadores ', +, -, \* e / já foram apresentadas em parágrafos anteriores. Os outros operadores mostrados na Tabela 4.1 serão apresentados nessa Seção.

Vamos analisar a utilização do operador \. Para isso, definimos um sistema de equações lineares,

$$\begin{aligned}
 a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n}x_n &= b_1 \\
 a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,n}x_n &= b_2 \\
 \dots & \\
 a_{n,1}x_1 + a_{n,2}x_2 + \dots + a_{n,n}x_n &= b_n
 \end{aligned}$$

que pode ser escrito na forma matricial

$$Ax = \mathbf{b}$$

onde

$$A = \begin{bmatrix}
 a_{1,1} & a_{1,2} & \dots & a_{1,n} \\
 a_{2,1} & a_{2,2} & \dots & a_{2,n} \\
 \vdots & \vdots & \ddots & \vdots \\
 a_{n,1} & a_{n,2} & \dots & a_{n,n}
 \end{bmatrix}$$

com

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{e} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Nas expressões anteriores,  $A \in \mathbb{R}^{n \times n}$  é a matriz dos coeficientes,  $\mathbf{x} \in \mathbb{R}^{n \times 1}$  é o vetor das incógnitas e  $\mathbf{b} \in \mathbb{R}^{n \times 1}$  é o vetor de termos independentes.

Resolver um sistema linear é obter o valor do vetor  $\mathbf{x}$ . Na situação mais simples, a matriz  $A$  é não-singular (admite inversa) e a solução, única, é dada pela expressão

$$\mathbf{x} = A^{-1}\mathbf{b}$$

onde  $A^{-1}$  é a inversa da matriz  $A$ . A expressão anterior pode ser representada no Scilab como

```
--> x = inv(A) * b
```

onde `inv`, com vimos em exemplo anterior, é uma função interna do Scilab que calcula a inversa de uma matriz.

Para exemplificar, vamos considerar um sistema linear com

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix} \quad \text{e} \quad \mathbf{b} = \begin{bmatrix} 1 \\ 8 \end{bmatrix}$$

Temos,

```
--> // Solucao de Ax = b usando a funcao inv

-->A = [2 0; 0 4]      // Matriz A
A =

!  2.   0. !
!  0.   4. !

-->inv(A)              // A admite inversa
ans =

!  0.5   0.   !
!  0.    0.25 !

-->b = [1; 8]          // Vetor b
b =

!  1. !
!  8. !

-->x = inv(A) * b      // Solucao do sistema linear
x =

!  0.5 !
!  2.  !

-->
```



O mesmo resultado pode ser encontrado utilizando-se o operador  $\backslash$ . Temos, então,

```
--> Resolucao de Ax = b usando o operador \
```

```
-->x = A \ b
x =
```

```
!  0.5 !
!  2.  !
```

```
-->
```

É importante observar que o símbolo  $\backslash$  não define uma divisão matricial. Indica, apenas, uma outra forma de se obter a solução de um sistema linear.

O operador  $.$  (ponto), como pode ser visto na Tabela 4.1, é utilizado com outros operadores ( $*$ ,  $\backslash$ ,  $/$ ,  $\wedge$ ) para realizar operações elemento a elemento de vetores ou de matrizes. A sessão do Scilab a seguir mostra exemplos dessas operações utilizando vetores.

```
--> Definicao do vetor x
```

```
-->x = [1 3 4 6]
x =
```

```
!  1.   3.   4.   6. !
```

```
--> Definicao do vetor y
```

```
-->y = [2 4 6 8]
y =
```

```
!  2.   4.   6.   8. !
```

```
-->x .* y
ans =
```

```
!  2.   12.  24.  48. !
```

```
-->x * y
      !--error    10
inconsistent multiplication
```

```
-->
```

A operação  $.*$  gera um vetor formado pelo produto dos elementos dos vetores  $x$  e  $y$ . Apenas para fixar conceitos, verificamos que a operação  $x * y$  não pode ser realizada.

Continuando com os exemplos, usamos os operadores  $./$  para dividir os elementos do vetor  $x$  pelos elementos do vetor  $y$ ,

```
-->x ./ y
ans =
```

```
! 0.5 0.75 0.6666667 0.75 !
```

```
-->
```

e o operador `.\` para dividir os elementos do vetor `y` pelos elementos do vetor `x`,

```
-->x .\ y
ans =
```

```
! 2. 1.3333333 1.5 1.3333333 !
```

```
-->
```

Essa operação é equivalente à operação

```
-->y ./ x
ans =
```

```
! 2. 1.3333333 1.5 1.3333333 !
```

```
-->
```

A utilização do operador `.^` é mostrada nos exemplos apresentados em seguida,

```
-->x .^ y
ans =
```

```
! 1. 81. 4096. 1679616. !
```

```
-->y .^ x
ans =
```

```
! 2. 64. 1296. 262144. !
```

```
-->
```

Vamos verificar também a utilização do operador `.` (ponto) em conjunção com os operadores `*`, `\`, `/`, `^` quando se trata de matrizes. Para isso, vamos considerar a matriz quadrada  $A \in \mathbb{R}^{3 \times 3}$ ,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

A sessão do Scilab a seguir mostra exemplos dessas operações utilizando matrizes.

```
-->// Definindo a matriz A
```

```
-->A = [ 1 2 3; 4 5 6; 7 8 9]
```

```
A =
```

```
! 1. 2. 3. !
! 4. 5. 6. !
! 7. 8. 9. !
```

```
-->A .* A
ans =
```

```
! 1. 4. 9. !
! 16. 25. 36. !
! 49. 64. 81. !
```

```
-->A ^ 2
ans =
```

```
! 30. 36. 42. !
! 66. 81. 96. !
! 102. 126. 150. !
```

```
-->A * A
ans =
```

```
! 30. 36. 42. !
! 66. 81. 96. !
! 102. 126. 150. !
```

```
-->
```

Após definir a matriz  $A$  no ambiente Scilab, foi feito o produto  $A .* A$ . O resultado é uma matriz com elementos iguais ao produto de cada elemento da matriz  $A$  pelo seu correspondente. Observar que o resultado obtido pela operação  $.*$  é completamente diferente do resultado obtido fazendo-se  $A ^2$ . Este último resultado é idêntico ao resultado obtido fazendo-se  $A * A$ .

Continuando com os exemplos, a operação  $A ./ A$ ,

```
-->A ./ A
ans =
```

```
! 1. 1. 1. !
! 1. 1. 1. !
! 1. 1. 1. !
```

```
-->
```

apresenta uma matriz com todos os elementos iguais a 1, como era de se esperar, já que os elementos da matriz gerada são obtidos dividindo-se cada um dos elementos da matriz  $A$  pelos seu correspondente. Para fixar conceitos, vamos considerar a matriz quadrada  $B \in \mathbb{R}^{3 \times 3}$ ,

$$B = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$

Temos,

```
-->// Definicao da matriz B
```

```
-->B = [ 2 2 2; 2 2 2; 2 2 2]
```

```
B =
```

```
! 2. 2. 2. !
! 2. 2. 2. !
! 2. 2. 2. !
```

```
-->A ./ B
```

```
ans =
```

```
! 0.5 1. 1.5 !
! 2. 2.5 3. !
! 3.5 4. 4.5 !
```

```
-->
```

como era de se esperar.

Continuando com os exemplos, temos

```
-->A .^ B
```

```
ans =
```

```
! 1. 4. 9. !
! 16. 25. 36. !
! 49. 64. 81. !
```

```
-->
```

onde cada elemento da matriz A foi elevado ao elemento correspondente na matriz B, que equivale, no caso a

```
-->A .^ 2
```

```
ans =
```

```
! 1. 4. 9. !
! 16. 25. 36. !
! 49. 64. 81. !
```

```
-->
```

Temos, ainda, a operação

```
-->A .\ B
```

```
ans =
```

```
! 2. 1. 0.6666667 !
! 0.5 0.4 0.3333333 !
! 0.2857143 0.25 0.2222222 !
```

-->

que equivale à operação

```
-->B ./ A
ans =
```

```
! 2.      1.      0.6666667 !
! 0.5     0.4     0.3333333 !
! 0.2857143 0.25   0.2222222 !
```

-->

O produto de Kronecker entre duas matrizes,  $A \in \mathbb{R}^{m \times n}$  e  $B \in \mathbb{R}^{p \times q}$ ,

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \quad \text{e} \quad B = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,q} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{p,1} & b_{p,2} & \cdots & b_{p,q} \end{bmatrix}$$

é representado por  $A \otimes B \in \mathbb{R}^{(m \cdot p) \times (n \cdot q)}$  e definido por:

$$A \otimes B = \begin{bmatrix} a_{1,1}B & a_{1,2}B & \cdots & a_{1,n}B \\ a_{2,1}B & a_{2,2}B & \cdots & a_{2,n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1}B & a_{m,2}B & \cdots & a_{m,n}B \end{bmatrix}$$

Para exemplificar, vamos considerar as matrizes

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \text{e} \quad B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

No Scilab, o produto de Kronecker é implementado através do operador `.*.`, como podemos ver no exemplo,

```
--> // Definindo as matrizes A e B
```

```
-->A = [1 2; 3 4]
A =
```

```
! 1.  2.  !
! 3.  4.  !
```

```
-->B = [1 2 3; 4 5 6]
B =
```

```
! 1.  2.  3.  !
! 4.  5.  6.  !
```

```
-->A .* B      // Produto de Kronecker via operador .*
ans =
```

```
!  1.   2.   3.   2.   4.   6.  !
!  4.   5.   6.   8.  10.  12. !
!  3.   6.   9.   4.   8.   12. !
!  12.  15.  18.  16.  20.  24. !
```

```
-->
```

ou através da função interna `kron`,

```
-->kron(A, B)    // Produto de Kronecker usando funcao interna
ans =
```

```
!  1.   2.   3.   2.   4.   6.  !
!  4.   5.   6.   8.  10.  12. !
!  3.   6.   9.   4.   8.   12. !
!  12.  15.  18.  16.  20.  24. !
```

```
-->
```

## 4.9 Listas

Uma lista é uma coleção de objetos não necessariamente do mesmo tipo. Uma lista simples é definida pela função `list`. Esta função tem a forma geral

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

onde os  $a_i$  são os elementos da lista.

Vamos criar uma lista simples, que chamamos de `L`, composta por três elementos : o elemento 1, associado a `L(1)`, o elemento `w`, associado a `L(2)` e uma matriz 2x2 composta de elementos iguais a 1, associada a `L(3)`,

```
-->// Uma lista simples com 3 elementos
```

```
-->L = list(1, 'w', ones(2,2))
L =
```

```
    L(1)
```

```
    1.
```

```
    L(2)
```

```
    w
```

```
    L(3)
```

```
!  1.   1.  !
!  1.   1.  !
```

É importante observar que a indexação de elementos de uma lista, no Scilab, inicia-se por 1.

Vamos transformar o elemento L(2) da lista do exemplo anterior em uma lista cujo primeiro elemento, L(2)(1), é *w* e cujo segundo elemento, L(2)(2), é uma matriz 2x2 de números aleatórios,

```
-->// Transformando o elemento L(2) em uma lista
```

```
-->L(2) = list('w', rand(2,2))
```

```
L =
```

```
    L(1)
```

```
    1.
```

```
    L(2)
```

```
        L(2)(1)
```

```
        w
```

```
        L(2)(2)
```

```
!   0.2113249    0.0002211 !
!   0.7560439    0.3303271 !
```

```
    L(3)
```

```
!   1.    1.  !
!   1.    1.  !
```

```
-->
```

A seguir, mostramos o comando necessário para acessar o elemento (1,2) do segundo elemento de L(2),

```
-->L(2)(2)(2,1)
```

```
ans =
```

```
    0.7560439
```

```
-->
```

As lista tipadas são um outro tipo de dado aceito pelo Scilab. As listas tipadas são definidas através da função `tlist`. A função `tlist` possui, obrigatoriamente, como primeiro argumento um *string* ou um vetor de *strings* e os demais argumentos são os elementos da lista. A seguir, alguns exemplos de manipulação de listas tipadas.

```
-->// Definicao de uma lista tipada
```

```

-->L = tlist(['Carro'; 'Cidade'; 'Valores'], 'Natal', [2,3])
L =

      L(1)

!Carro    !
!         !
!Cidade   !
!         !
!Valores  !

      L(2)

Natal

      L(3)

!  2.    3. !

-->// Acessando elementos

-->L('Cidade')
ans =

Natal

-->L('Valores')
ans =

!  2.    3. !

-->L(1)(3)
ans =

Valores

-->

```

Observar que os índices de uma lista tipada podem ser *strings* definidas no primeiro argumento da função `tlist()`.

Neste Capítulo, apresentamos os tipos de dados que podem ser manipulados pelo Scilab. Diversos exemplos foram mostrados utilizando polinômios, vetores, matrizes e listas. Os exemplos foram apresentados a partir do *prompt* do Scilab.

No próximo Capítulo, vamos mostrar com podemos desenvolver programas na linguagem Scilab.



# Capítulo 5

## Programação

Uma das características mais importante do Scilab é a facilidade com que o usuário pode criar seus próprios programas.

Apesar de simples, a linguagem Scilab disponibiliza a maioria das estruturas das linguagens de programação convencionais. A diferença principal é que, na programação Scilab, não há a necessidade da declaração prévia dos tipos das variáveis que serão utilizadas ao longo do programa.

Um fator a ser levado em consideração é que Scilab é um interpretador de comandos. Os programas escritos na linguagem Scilab são, portanto, normalmente executados em um tempo maior que os programas semelhantes escritos em linguagens compiláveis. Este fato é mais relevante quando precisamos desenvolver programas para a realização de simulações ou de otimizações. Nesses casos, pode ser conveniente escrever o código responsável pela lentidão do processamento em uma linguagem convencional (no caso, C ou FORTRAN) e rodar esse código dentro do ambiente Scilab. No Apêndice B, mostramos os procedimentos necessários à ligação de códigos escritos em C com programas escritos em Scilab. Deve ser enfatizado, entretanto, que a vantagem na utilização do Scilab advém da facilidade de prototipação de programas e da disponibilidade de uma poderosa biblioteca de funções gráficas. Como sempre ocorre nessas situações, cabe ao usuário encontrar a sua solução de compromisso.

Nos Capítulos anteriores, vimos como escrever e executar comandos a partir do *prompt* do Scilab. Neste Capítulo, apresentamos as principais estruturas de controle de fluxo de programas. Essas estruturas são utilizadas, depois, para gerar programas, chamados de *scripts* ou de funções, que serão executados no ambiente Scilab.

### 5.1 Comandos para Iterações

Existem dois comandos que permitem a realização de iterações, *loops*, no Scilab: o *loop* implementado com o comando `for` e o *loop* implementado com o comando `while`.

#### 5.1.1 O *Loop* `for`

O comando `for` tem a forma geral:

```
for variavel = vetor_linha
    instrucao_1
    instrucao_2
    ... ..
    instrucao_n
end
```

No ambiente Scilab, a forma acima é equivalente a

```
-->for variavel=vetor_linha
-->  instrucao_1
-->  instrucao_2
-->  instrucao_n
-->end
```

como mostrado no exemplo,

```
--> Loop for em linha de comando
```

```
-->for k = 1:3
-->a = k + 1
-->end
a =

    2.
a =

    3.
a =

    4.
-->
```

Como vimos no exemplo, no *loop for* o comportamento das iterações é baseado no conteúdo do vetor linha.

No exemplo a seguir, vamos considerar que a variável *k* do comando **for** assuma os valores estabelecidos pelo vetor linha  $v = [2 \ 3 \ 4 \ 5 \ 6]$ . O número de iterações, portanto, será igual ao número de componentes do vetor linha *v*. Teremos, dessa forma, cinco iterações. Na primeira iteração, o valor da variável *k* será igual ao primeiro elemento do vetor *v*,  $v(1)$ , que é igual a 2, e na última iteração o valor da variável *k* será igual ao último elemento do vetor *v*,  $v(5)$ , que vale 6. No ambiente Scilab, temos:

```
-->v = [2 3 4 5 6]; // v tambem pode ser escrito como v = 2:6

-->y = 0;

-->for k = v
-->y = y + k
-->end
y =

    2.
y =

    5.
y =

    9.
y =
```

```

14.
y =

```

```

20.

```

```
-->
```

O vetor  $v$  poderia ter sido descrito na forma  $v = 2:6$ .

A variável do comando `for` também pode ser uma lista. Neste caso, a variável assume os valores dos elementos da lista, como no exemplo:

```
-->L = list(1, [1 2; 3 4], 'teste')
```

```
L =
```

```

L(1)

```

```

1.

```

```

L(2)

```

```

!  1.   2. !
!  3.   4. !

```

```

L(3)

```

```

teste

```

```
-->for k=L
```

```
-->disp(k)
```

```
-->end
```

```

1.

```

```

!  1.   2. !
!  3.   4. !

```

```

teste

```

```
-->
```

### 5.1.2 O *Loop* while

O comando `while` tem a forma geral,

```

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

```

A forma acima é equivalente à forma

```
-->while condicao
-->  instrucao_1
-->  instrucao_2
-->  instrucao_n
-->end
```

no ambiente Scilab.

O *loop* baseado no `while` realiza uma seqüência de instruções enquanto uma determinada condição estiver sendo satisfeita. A condição geralmente inclui comparações entre objetos. Na Tabela 5.1, apresentamos os operadores que permitem fazer comparações entre valores de objetos no Scilab.

Operadores	Significado
<code>==</code> ou <code>=</code>	igual a
<code>&lt;</code>	menor do que
<code>&gt;</code>	maior do que
<code>&lt;=</code>	menor ou igual a
<code>&gt;=</code>	maior ou igual a
<code>&lt;&gt;</code> ou <code>~=</code>	diferente

Tabela 5.1: Operadores condicionais

A seguir, apresentamos um exemplo da utilização do *loop* baseado no comando `while`,

```
-->x = 1;

-->while x < 14
-->x = x * 2
-->end
x =

    2.

x =

    4.

x =

    8.

x =

   16.

-->
```

## 5.2 Comandos Condicionais

O Scilab implementa dois tipos de comandos condicionais: `if-then-else` e `select-case`.

### 5.2.1 Comando if-then-else

O comando if-then-else tem duas formas. Na forma mais simples, o comando é escrito como

```
if condicao_1 then
    sequencia_de_instrucoes_1
else
    sequencia_de_instrucoes_2
end
```

enquanto na sua forma mais geral o comando é escrito como,

```
if condicao_1 then
    sequencia_de_instrucoes_1
elseif condicao_2
    sequencia_de_instrucoes_2
... ..
elseif condicao_n
    sequencia_de_instrucoes_n
else
    sequencia_de_instrucoes_n+1
end
```

A forma acima é equivalente à forma

```
--> if condicao_1 then
-->     sequencia_de_instrucoes_1
-->elseif condicao_2
-->     sequencia_de_instrucoes_2
--> elseif condicao_n
-->     sequencia_de_instrucoes_n
--> else
-->     sequencia_de_instrucoes_n+1
-->end
```

no ambiente Scilab.

A *condicao\_1* do comando if-then-else avalia uma expressão. Se esta expressão for verdadeira, *true*, será executada a instrução ou instruções subseqüentes. Se for falsa, *false*, será executada a instrução ou instruções após o *else* ou o *elseif*, conforme o caso. Alguns exemplos da utilização do condicional if-then-else,

```
-->x = -1
x =

- 1.

-->if x < 0 then
-->  y = -x          // apresenta a resposta
y =

1.
-->else
```

```

-->  y = x
-->end

-->// Outra forma

-->x = 1          // Inicializando
x =

    1.

-->if x > 0 then, y = -x, else, y=x, end
y =

    - 1.

-->x = -1
x =

    - 1.

-->if x > 0 then, y = -x, else, y=x, end
y =

    - 1.

-->

```

### 5.2.2 Comando select-case

O condicional select-case tem a forma geral,

```

select variavel_de_teste
case expressao_1
    sequencia_de_instrucoes_1
case expressao_2
    sequencia_de_instrucoes_2
    ... ..
case expressao_n
    sequencia_de_instrucoes_n
else
    sequencia_de_instrucoes_n+1
end

```

A forma acima é equivalente à forma

```

-->select variavel_de_teste
-->case expressao_1
-->  sequencia_de_instrucoes_1
-->case expressao_2
-->  sequencia_de_instrucoes_2
-->case expressao_n
-->  sequencia_de_instrucoes_n

```

```
-->else
-->  sequencia_de_instrucoes_n+1
-->end
```

O condicional `select-case` compara o valor de uma variável de teste com as várias expressões dos `case`. Serão executadas as instruções que possuírem o valor da expressão do `case` igual ao valor da variável de teste. Um exemplo de utilização do condicional `select-case`,

```
-->x = -1
x =

- 1.

-->select x
-->case 1
-->  y = x + 5
-->case -1
-->  y = sqrt(x)
y =

i
-->end

-->x = 1
x =

1.

-->select x, case 1, y = x+5, case -1, y = sqrt(x), end
y =

6

-->
```

### 5.3 Definindo *Scripts*

Vimos, no Capítulo 3, que é possível, através dos comandos `save` e `load`, armazenar e recuperar valores de variáveis utilizadas no ambiente Scilab. Vimos, também, que é possível, através do comando `diary`, criar arquivos onde armazenamos a memória das sessões que realizamos no Scilab.

Além desses dois tipos de arquivos, podemos criar arquivos contendo comandos do Scilab que serão executados posteriormente dentro do seu ambiente. Um desses arquivos, chamados de arquivos *scripts* ou simplesmente *scripts*, são formados por texto puro, sem acentuação, contendo uma seqüência de comandos que o usuário digitaria em uma sessão interativa no *prompt* do Scilab. Por convenção, os *scripts* do Scilab possuem extensão `sce` e são executados através do comando

```
-->exec('nome_do_arquivo_de_comandos.sce')
```

São características dos arquivos *scripts*:

- Todas as variáveis definidas no arquivo de comandos permanecem válidas no ambiente Scilab após a execução dos comandos do arquivo, e
- Não há uma definição clara das entradas e saídas do *script*. Esse fato pode dificultar a correção de possíveis erros.

Para exemplificar, vamos escrever um *script* na linguagem Scilab para obter a  $\sqrt{2}$  usando o método iterativo de Newton-Raphson. Segundo esse método [9], a raiz de uma função,  $f(x)$  pode ser calculada através da expressão,

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

onde  $f'(x_i)$  representa a derivada da função  $f(x)$  no ponto  $x_i$  e  $i = 0, 1, \dots, n$  representa o número de iterações.

A função  $f(x)$  que permite obter a  $\sqrt{2}$  é

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

Assim, usando a fórmula de Newton-Raphson, temos

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

Usando  $x_0 = 1.0$  como aproximação inicial, escrevemos o *script* mostrado em Código 1,

```
// script: metodo de Newton-Raphson
// f(x) = x * x - 2

N = 10;      // Numero maximo de iteracoes
x0 = 1.0;    // Aproximacao inicial
delta = 10^(-5); // Erro

xn = x0;
for n=1:N
    xn1 = xn - (xn * xn - 2)/(2 * xn);
    if abs((xn1-xn)/xn1) < delta then
        printf('Valor da raiz = %10.7f', xn1)
        return
    end
    xn = xn1;
end
printf('Nao converge em n=%f iteracoes", N)
```

Código 1: O *script* que implementa o método de Newton-Raphson para obter  $\sqrt{2}$ .

Observar o ; após alguns comandos. Isso evita que a execução do *script* seja um processo “ruidoso” no ambiente Scilab.

A execução do *script*, que foi armazenado em um arquivo chamado `newton.sce`, e o resultado é mostrado na sessão,

```
-->exec newton.sce      // executando o script

-->// script: metodo de Newton-Raphson
```



```

-->// f(x) = x * x - 2

-->N = 10;      // Numero maximo de iteracoes

-->x0 = 1.0;    // Aproximacao inicial

-->delta = 10^(-5); // Erro

-->xn = x0;

-->for n=1:N
-->    xn1 = xn - (xn * xn - 2)/(2 * xn);
-->    if abs((xn1-xn)/xn1) < delta then
-->        printf('Valor da raiz = %10.8f',xn1)
-->        return
-->    end
-->    xn = xn1;
-->end
Valor da raiz = 1.41421356      // valor de sqrt(2) !
-->

```

Como indicado anteriormente, todas as variáveis utilizadas no *script* permanecem ativas após a sua execução,

```

Valor da raiz = 1.4142136      // Final da execucao
-->// Variaveis permanecem ativas no ambiente Scilab

-->N
N =

    10.

-->x0
x0 =

    1.

-->xn1
xn1 =

    1.4142136

-->delta
delta =

    0.00001

-->

```

Um outro exemplo de *script* é mostrado em Código 6, Capítulo 6.

## 5.4 Definindo Funções

Uma função obedece a uma estrutura da forma:

```
function [y1, ..., yn] = foo(x1, ..., xm)
    instrucao_1
    instrucao_2
    ...
    instrucao_p
endfunction
```

onde `foo` é o nome da função,  $x_i$ ,  $i=1,\dots,m$ , são os seus argumentos de entrada,  $y_j$ ,  $j=1,\dots,n$ , são os seus argumentos de saída e `instrucao_i`,  $i=1,\dots,p$ , representa a seqüência de instruções que devem ser executadas pela função.

Toda função no Scilab é executada chamado seu nome seguido de seus argumentos. No exemplo, a função `foo` é executada através do comando

```
-->foo(x1, ..., xm)
```

Como pode ser observado, uma função possui uma estrutura pré-determinada. As principais características das funções são:

- As variáveis definidas na função, chamadas de variáveis locais, não permanecem no ambiente Scilab após a execução da função;
- As entradas e saídas do programa são claramente definidas, e
- Uma função, após ser definida, pode ser chamada a qualquer tempo.

Uma função pode ser criada usando um dos seguintes procedimentos:

1. Digitação no próprio ambiente,

```
-->Digitação uma funcao no ambiente Scilab
```

```
-->function [y1, y2] = exemplo(x1, x2)
-->// Entrada: x1, x2
-->// Saida: y1, y2
-->y1 = x1 + x2
-->y2 = x1 * x2
-->endfunction
```

```
-->[a,b] = exemplo(2, 3)
```

```
b =
```

```
6.
```

```
a =
```

```
5.
```

```
-->
```

Observar que a função retorna primeiro o último valor calculado.

2. Usando o comando `deff`,

```

-->Usando deff

-->deff('[y1, y2]=exemplo(x1, x2)', 'y1=x1+x2, y2=x1*x2')

-->[a, b] = exemplo(3,4)
b =

    12.
a =

    7.

-->

```

3. Digitando o texto da função em um arquivo e, em seguida, carregando esse arquivo no ambiente Scilab. Por convenção, as funções definidas pelo usuário possuem extensão `sci` e são carregadas no ambiente Scilab através do comando:

```

-->getf('nome_do_arquivo_de_comandos.sci')

```

Exemplos de funções construídas desta maneira serão apresentados na seção 5.4.2.

A escolha de um dos procedimentos anteriores depende da conveniência do usuário.

#### 5.4.1 Variáveis Globais e Variáveis Locais

As variáveis globais são válidas no ambiente do Scilab enquanto as variáveis locais são válidas apenas no escopo de uma função. Para exemplificar os conceitos, vamos considerar a função  $f(x1, x2)$ ,

```

-->function [y1, y2] = f(x1, x2)
-->y1 = x1 + x2
-->y2 = x1 - x2
-->endfunction

```

Observe que  $y1$  e  $y2$  são as variáveis de saída enquanto  $x1$  e  $x2$  são as variáveis de entrada da função. Vamos considerar alguns exemplos de chamadas desta função.

Inicialmente, a função é chamada com argumentos  $x1 = 1$  e  $x2 = 3$  tendo seus parâmetros de retorno associados às variáveis  $m1$  e  $m2$ . Observe que  $y1$  e  $y2$ , apesar de terem sido calculados dentro da função (definição local), não são definidas no ambiente do Scilab.

```

-->[m1, m2] = f(1,3) // Retorno associado as variaveis [m1, m2]
m2 =

- 2.
m1 =

    4.

```

```

--> // Provocando erro : y1 e y2 nao sao globais

```

```

-->y1
!--error      4

```

```
undefined variable : y1
```

```
-->y2
```

```
!--error      4
```

```
undefined variable : y2
```

```
-->
```

Continuando com o exemplo, a função é chamada sem a associação explícita de variáveis de retorno. Este caso é como se a função  $f(x1, x2)$  tivesse sido chamada com a associação de apenas uma variável de retorno, uma chamada do tipo  $z = f(1,3)$ .

```
-->f(1,3)      // Chamada equivalente a z = f(1,3)
```

```
ans =
```

```
4.
```

```
-->
```

O exemplo continua e um erro é provocado quando a função é chamada com apenas um argumento. Logo em seguida, o argumento é definido no ambiente (definição global) e a função é, novamente, chamada com apenas um argumento sem que haja a ocorrência de erro.

```
-->f(1)      // Erro por indefinicao de argumento
```

```
!--error      4
```

```
undefined variable : x2
```

```
at line      2 of function f
```

```
called by :
```

```
f(1)
```

```
-->x2 = 3     // Definindo x2 no ambiente (global)
```

```
x2 =
```

```
3.
```

```
-->f(1)      // Chamando a funcao com apenas um argumento
```

```
ans =
```

```
4.
```

```
-->
```

Como está claro pela sessão anterior, a chamada de uma função sem que todos os seus argumentos de entrada tenham sido previamente definidos ocasiona erro. Os argumentos de entrada devem ser definidos explicitamente na chamada ou através de definições via variáveis globais. Considerando a função anterior, teremos, como casos interessante (mas não aconselháveis!), os exemplos,

```
-->x1 = 2; x2 = 3; // Definindo x1 e x2
```

```
-->f()      // Chamando a funcao f sem nenhum argumento
```

```
ans =
```

```

5.
-->[m1, m2] = f() // Retorno associado as variaveis [m1, m2]
m2 =

- 1.
m1 =

5.

-->

```

### 5.4.2 Arquivos com Funções

Como vimos nas seções anteriores, é possível definir funções dentro do próprio ambiente Scilab. Entretanto, quando a função possui muitos comandos, é mais conveniente utilizar um editor de textos para criar, fora do ambiente Scilab, um arquivo contendo a função. É importante observar que o nome desse arquivo não é, necessariamente, o nome que deve ser dado à função.

Como primeiro exemplo [9], vamos desenvolver um programa para resolver a equação diferencial ordinária,

$$\frac{dy}{dx} = (x - y)/2$$

com condição inicial  $y(0) = 1$ , utilizando o método de Runge-Kutta de quarta ordem. Vamos considerar o intervalo de integração  $[0, 3]$  e o passo de integração igual a  $h = 1/8$ . A solução numérica obtida por Runge-Kutta será comparada com valores da solução exata que é  $y(x) = 3e^{-x/2} + x - 2$ .

O método Runge-Kutta de quarta ordem, para resolver uma equação diferencial ordinária de primeira ordem,

$$\frac{dy}{dx} = f(x, y)$$

é representado pelas equações,

$$y_{k+1} = y_k + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4)$$

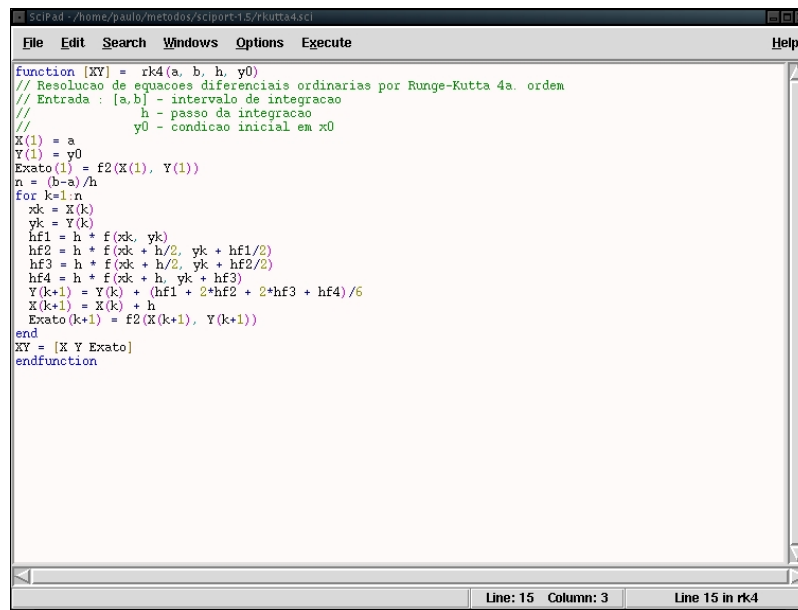
com os coeficientes  $f_i$  definidos por :

$$\begin{aligned} f_1 &= f(x_k, y_k) \\ f_2 &= f\left(x_k + \frac{h}{2}, y_k + \frac{h}{2}f_1\right) \\ f_3 &= f\left(x_k + \frac{h}{2}, y_k + \frac{h}{2}f_2\right) \\ f_4 &= f(x_k + h, y_k + hf_3) \end{aligned}$$

e pode ser implementado através do Algoritmo 1:

**Algoritmo 1:** Método de Runge-Kutta de Quarta Ordem.**Entrada:**  $[a, b]$ ,  $h$  e  $y_0$ Fazer  $x_0 = a$ Fazer  $n = (b - a)/h$ **for**  $k = 0$  *to*  $n$  **do**    Calcular  $f_1, f_2, f_3$  e  $f_4$     Calcular  $y_{k+1} = y_k + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4)$     Fazer  $x_{k+1} = x_k + h$ **end****Resultado:** Apresentar valores de  $x_k$  e  $y_k$ 

Podemos utilizar um editor de textos, por exemplo, vi, joe, para, fora do ambiente Scilab, criar o programa. No caso, usamos o SciPad, o editor do Scilab, como mostrado na Figura 5.1.



```

function [XY] = rk4(a, b, h, y0)
// Resolucao de equacoes diferenciais ordinarias por Runge-Kutta 4a. ordem
// Entrada : [a,b] - intervalo de integracao
//          h - passo da integracao
//          y0 - condicao inicial em x0
//
X(1) = a
Y(1) = y0
Exato(1) = f2(X(1), Y(1))
n = (b-a)/h
for k=1:n
    xk = X(k)
    yk = Y(k)
    hf1 = h * f(xk, yk)
    hf2 = h * f(xk + h/2, yk + hf1/2)
    hf3 = h * f(xk + h/2, yk + hf2/2)
    hf4 = h * f(xk + h, yk + hf3)
    Y(k+1) = Y(k) + (hf1 + 2*hf2 + 2*hf3 + hf4)/6
    X(k+1) = X(k) + h
    Exato(k+1) = f2(X(k+1), Y(k+1))
end
XY = [X Y Exato]
endfunction

```

Figura 5.1: Escrevendo uma função usando o editor do Scilab.

O programa criado é mostrado em Código 2.

```

1 function [XY] = rk4(a, b, h, y0)
2 // Resolucao de equacoes diferenciais ordinarias por Runge-Kutta 4a. ordem
3 // Entrada : [a,b] - intervalo de integracao
4 //           h - passo da integracao
5 //           y0 - condicao inicial em x0
6 X(1) = a
7 Y(1) = y0
8 Exato(1) = f2(X(1), Y(1))
9 n = (b-a)/h
10 for k=1:n
11     xk = X(k)
12     yk = Y(k)
13     hf1 = h * f(xk, yk)
14     hf2 = h * f(xk + h/2, yk + hf1/2)
15     hf3 = h * f(xk + h/2, yk + hf2/2)
16     hf4 = h * f(xk + h, yk + hf3)
17     Y(k+1) = Y(k) + (hf1 + 2*hf2 + 2*hf3 + hf4)/6
18     X(k+1) = X(k) + h
19     Exato(k+1) = f2(X(k+1), Y(k+1))
20 end
21 XY = [X Y Exato]
22 endfunction

```

Código 2: Programa principal, implementação do método de Runge-Kutta.

Como podemos observar, o programa chama a função  $f(x, y)$ , nas linhas 13 a 16, e a função com a solução exata da equação diferencial dada<sup>1</sup>, nas linhas 8 e 19 de Código 2. A função  $f(x, y)$  é mostrada em Código 3,

```

1 function [fxy] = f(x,y)
2 // funcao exemplo
3 fxy = (x - y)/2
4 endfunction

```

Código 3: A função  $f(x, y)$ .

e a solução exata é mostrada em Código 4,

```

1 function [fexato] = f2(x,y)
2 // funcao solucao
3 fexato = 3 * exp(-x/2) + x - 2
4 endfunction

```

Código 4: A solução exata da equação diferencial.

Os programas podem ser carregados no Scilab logo após sua digitação através da utilização da sub-opção Load into Scilab Ctrl+I da opção **Execute** do Editor. Ou podem ser carregados no ambiente Scilab através do comando `getf()`. Considerando que o programa principal e as funções estejam em arquivos localizados no diretório onde o Scilab é lançado. Assim sendo, o comando

<sup>1</sup>Devemos salientar que nem sempre a solução analítica, exata, estará disponível

```
-->getf('fdexy.sci') // arquivo com a funcao f(x,y) = (x - y)/2
```

```
-->
```

carrega a função  $f(x,y)$  no ambiente Scilab. Depois, com o comando

```
-->getf('fsolucao.sci') // arquivo com a funcao solucao = 3exp(-x/2)+x-2
```

```
-->
```

a função com a solução exata é carregada. Finalmente, o comando `getf('rkutta4.sci')` carrega o arquivo que define a função `rk4`<sup>2</sup> e o comando `rk4(0, 3, 1/8, 1)` executa o programa. Os resultados obtidos são :

```
-->getf('rkutta4.sci') // arquivo com o metodo de Runge-Kutta 4a. ordem
```

```
-->rk4(0, 3, 1/8, 1) // executando o programa
```

```
ans =
```

```
!  0.      1.      1.      !
!  0.125   0.9432392  0.9432392 !
!  0.25    0.8974908  0.8974907 !
!  0.375   0.8620874  0.8620874 !
!  0.5     0.8364024  0.8364023 !
!  0.625   0.8198470  0.8198469 !
!  0.75    0.8118679  0.8118678 !
!  0.875   0.8119457  0.8119456 !
!  1.      0.8195921  0.8195920 !
!  1.125   0.8343486  0.8343485 !
!  1.25    0.8557844  0.8557843 !
!  1.375   0.8834949  0.8834947 !
!  1.5     0.9170998  0.9170997 !
!  1.625   0.9562421  0.9562419 !
!  1.75    1.0005862  1.0005861 !
!  1.875   1.049817   1.0498169 !
!  2.      1.1036385  1.1036383 !
!  2.125   1.1617724  1.1617723 !
!  2.25    1.2239575  1.2239574 !
!  2.375   1.2899485  1.2899483 !
!  2.5     1.3595145  1.3595144 !
!  2.625   1.4324392  1.432439   !
!  2.75    1.5085189  1.5085188 !
!  2.875   1.5875626  1.5875625 !
!  3.      1.6693906  1.6693905 !
```

```
-->
```

Na primeira coluna são mostrados os valores de  $x$ , na segunda coluna são mostrados os valores da solução aproximada,  $y$ , e na terceira coluna são mostrados os valores da solução exata,  $3e^{-x/2} + x - 2$ .

---

<sup>2</sup>Esta ordem é irrelevante. Observe que os nomes das funções e os nomes dos arquivos que as contém são, intencionalmente, diferentes.



Como segundo exemplo de programação, [9], vamos resolver um sistema triangular superior de equações lineares, Este exemplo requer a leitura de um arquivo externo contendo dados.

Vamos considerar o sistema triangular superior,

$$\begin{array}{rccccrcr} 4x_1 & - & x_2 & + & 2x_3 & + & 2x_4 & - & x_5 & = & 4 \\ & & - & 2x_2 & + & 6x_3 & + & 2x_4 & + & 7x_5 & = & 0 \\ & & & & x_3 & - & x_4 & - & 2x_5 & = & 3 \\ & & & & & & - & 2x_4 & - & x_5 & = & 10 \\ & & & & & & & & & 3x_5 & = & 6 \end{array}$$

Para resolver estes tipos de sistemas, usamos a matriz dos coeficientes aumentada, definida por

$$[A \mid \mathbf{b}] = \left[ \begin{array}{ccccc|c} 4 & -1 & 2 & 2 & -1 & 4 \\ 0 & -2 & 6 & 2 & 7 & 0 \\ 0 & 0 & 1 & -1 & -2 & 3 \\ 0 & 0 & 0 & -2 & -1 & 10 \\ 0 & 0 & 0 & 0 & 6 & 6 \end{array} \right]$$

e o processo de substituição reversa, indicado pelo Algoritmo 2:

---

**Algoritmo 2:** Método da Substituição Reversa.

---

```

 $x_n = \frac{b_n}{a_{n,n}}$ 
for  $r = n - 1$  até 1 do
   $soma = 0$ 
  for  $j = r + 1$  até  $n$  do
     $soma = soma + a_{r,j} * x_j$ 
  end
   $x_r = \frac{b(r) - soma}{a_{r,r}}$ 
end

```

---

A matriz aumentada, neste caso, é armazenada em um arquivo ASCII, que chamamos de `arquivo1`. O conteúdo de `arquivo1`, digitado com um editor qualquer, pode ser visto no ambiente Linux usando o comando `cat`,

```
paulo:~/metodos/funcoes/aula3$ cat arquivo1
```

```
4 -1 2 2 -1 4
0 -2 6 2 7 0
0 0 1 -1 -2 3
0 0 0 -2 -1 10
0 0 0 0 3 6
```

Este arquivo é lido pelo Scilab através do comando:

```
-->Ab = read('arquivo1', 5, 6)
Ab =
```

```
! 4. - 1. 2. 2. - 1. 4. !
! 0. - 2. 6. 2. 7. 0. !
```

```
! 0. 0. 1. - 1. - 2. 3. !
! 0. 0. 0. - 2. - 1. 10. !
! 0. 0. 0. 0. 3. 6. !
```

```
-->
```

onde  $Ab$  é a variável que contém a matriz aumentada. O comando `read` lê as cinco linhas e as seis colunas de dados do `arquivo1` que está armazenado no diretório de trabalho.

Caso seja necessário, a matriz dos coeficientes,  $A$ , e o vetor dos termos independentes,  $\mathbf{b}$ , podem ser recuperados da matriz  $Ab$  através da seqüência de comandos:

```
-->// Numero de linhas, nl, numero de colunas, nc, de Ab
```

```
-->[nl nc] = size(Ab)
```

```
nc =
```

```
6.
```

```
nl =
```

```
5.
```

```
-->A = Ab(:,1:nc-1) // Matriz dos coeficientes
```

```
A =
```

```
! 4. - 1. 2. 2. - 1. !
```

```
! 0. - 2. 6. 2. 7. !
```

```
! 0. 0. 1. - 1. - 2. !
```

```
! 0. 0. 0. - 2. - 1. !
```

```
! 0. 0. 0. 0. 3. !
```

```
-->b = Ab(:,nc) // Vetor dos termos independentes
```

```
b =
```

```
! 4. !
```

```
! 0. !
```

```
! 3. !
```

```
! 10. !
```

```
! 6. !
```

```
-->
```

O programa para resolver o sistema linear é mostrado no Código [5](#),

```

1  function X = subst(Tsup)
2  // Entrada : matriz triangular superior, Tsup
3  // Saida : Vetor solucao X
4  [nl, nc] = size(Tsup);
5  n = nc-1;
6  A = Tsup(:,1:n); // Matriz A
7  b = Tsup(:,nc) // Vetor b
8  X = zeros(n,1);
9  X(n) = b(n)/A(n,n);
10 for r = n-1:-1:1,
11     soma = 0,
12     for j = r+1:n,
13         soma = soma + A(r,j) * X(j);
14     end,
15 X(r) = (b(r) - soma)/A(r,r);
16 end
17 endfunction

```

Código 5: Programa para resolver um sistema triangular.

Usando a matriz aumentada  $Ab$  como entrada para este programa, obtemos como vetor solução,

```

-->subst(Ab)
ans =

! 5. !
! 4. !
! 1. !
! - 6. !
! 2. !

-->

```

### 5.4.3 Comandos Especiais

Scilab possui alguns comandos especiais que são, exclusivamente, utilizados por funções :

- `argn` - retorna o número de argumentos de entrada e de saída de uma função;
- `warning` e `pause` - suspendem, temporariamente, a execução de uma função;
- `break` - força o final de um *loop*;
- `return` ou `resume` - utilizado para passar as variáveis locais do ambiente da função para o ambiente que chamou a função.

Alguns dos comandos especiais apresentados anteriormente são utilizados na função,

```

function [z] = foo(x, y)
[out, in] = argn(0);

if x == 0 then,
    error('Divisao por zero');
end,

```

```
slope = y/x;
pause,
z = sqrt(slope);
s = resume(slope);
endfunction
```

Vamos chamar esta função com argumento  $x = 0$ . O valor  $x = 0$  ocasiona um erro, apresentado ao usuário através do comando `error`, com a conseqüente interrupção das operações. Há o retorno ao *prompt* do Scilab. Estas operações são apresentadas no exemplo,

```
-->getf('f3.sci')      // Carregando a funcao

-->z = foo(0, 1)       // Provocando um erro
!--error 10000
Divisao por zero
at line      5 of function foo          called by :
z = foo(0, 1)

-->
```

Na segunda chamada, mostrada a seguir, desta vez com os argumentos corretos, a função suspende a operação após o cálculo de `slope`. Neste ponto, o usuário pode verificar valores calculados pela função até o ponto em que houve a interrupção. O *prompt* `-1->`, causado pelo `pause`, indica a mudança de ambiente. O controle pode ser retornado à função através do comando `return`. As operações da função podem ser encerradas usando os comandos `quit` ou `abort`. Após digitar `resume`, a função calcula o valor de `z` e disponibiliza a variável `s`, local à função, para o ambiente que a chamou.

```
-->// Mudando de ambiente

-->z = foo(2,1)

-1->resume
z =

    0.7071068

-->s
s =

    0.5

-->
```

Neste Capítulo, apresentamos alguns programas desenvolvidos na linguagem Scilab. No Capítulo 6, vamos mostrar comandos que podem ser utilizados para traçar gráficos no ambiente Scilab.

## Capítulo 6

# Gráficos no Scilab

Apresentamos alguns comandos que podem ser utilizados para traçar gráficos bi-dimensionais e tri-dimensionais. Informações mais detalhadas sobre todos os comandos disponíveis na biblioteca gráfica do Scilab<sup>1</sup> podem ser acessadas através do navegador de *help*.

### 6.1 A Janela de Gráficos do Scilab

Todas as saídas gráficas de comandos do Scilab são apresentadas em uma janela gráfica. Essa janela é mostrada na Figura 6.1.

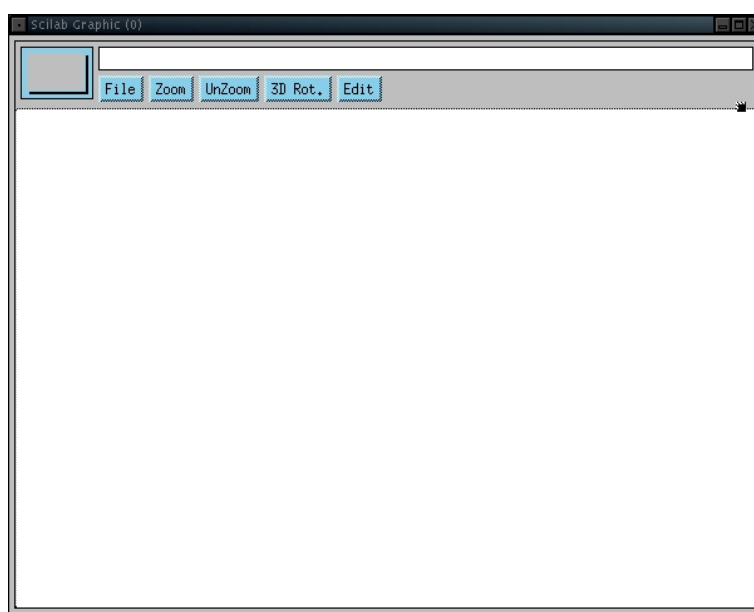


Figura 6.1: Janela gráfica do Scilab

Na Figura 6.1, observamos a existência de um menu horizontal com cinco opções: **File**, **Zoom**, **UnZoom**, **3D Rot.** e **Edit**. Utilizando o *mouse* para selecionar cada uma dessas opções, verificamos que:

- A opção **File** - possui sete sub-opções que permitem manipular arquivos relacionados com gráficos gerados: **Clear**, **Select**, **Print**, **Export**, **Save**, **Load** e **Close**;

---

<sup>1</sup>O Apêndice D contém uma listagem de todos os comandos da biblioteca gráfica do Scilab

- A opção `Zoom` - permite a ampliação de uma parte do gráfico. Escolhendo esta opção e delimitando uma área, a parte do gráfico dentro da área escolhida será expandida. Esta opção ainda não está implementada para gráficos tri-dimensionais;
- A opção `UnZoom` - desfaz as manipulações realizadas através da opção `Zoom`;
- A opção `3D Rot.` - permite efetuar rotações em gráficos bi-dimensionais e tri-dimensionais, e
- A opção `Edit` - possui sete sub-opções que permitem manipular o gráfico gerado: `Select`, `Redraw`, `Erase`, `Figure Properties`, `Current Axes Properties`, `Start Entity Picker`, `Stop Entity Picker`.

Algumas dessas opções serão utilizadas no decorrer deste Capítulo.

No Scilab, gráficos sucessivos são sobrepostos em uma mesma janela gráfica. Para evitar que isso ocorra, podemos utilizar o comando `clf()`.

As janelas gráficas podem ser manipuladas através da função `scf()`. Por exemplo,

```
-->// Manipulacao de janelas graficas
-->scf(0) // Acesso a janela grafica 0 (default)
-->scf(1) // Acesso a janela grafica 1
-->
```

## 6.2 Gráficos Bi-dimensionais

Gráficos bi-dimensionais podem ser gerados através da utilização da função `plot2d()`<sup>2</sup>.

A forma mais simples da função `plot2d()` é:

$$\text{plot2d}([x], y)$$

onde  $x$  e  $y$  podem ser matrizes ou vetores reais. Os colchetes,  $[ \text{ e } ]$ , envolvendo  $x$  indicam que este parâmetro é opcional.

Vamos fazer algumas considerações sobre os parâmetros  $x$  e  $y$ :

1. Se  $x$  e  $y$  são vetores, a função `plot2d()` permite traçar o gráfico de  $y$  em função de  $x$ . É importante observar que os dois vetores devem ter a mesma dimensão, isto é, os dois vetores devem ter o mesmo número de elementos;
2. Se  $x$  é um vetor e  $y$  é uma matriz, a função `plot2d()` permite traçar o gráfico de cada coluna da matriz  $y$  em função do vetor  $x$ . Neste caso, o número de elementos das colunas da matriz  $y$  deve ser igual ao número de elementos do vetor  $x$ ;
3. Se  $x$  e  $y$  são matrizes, a função `plot2d()` permite traçar o gráfico de cada coluna da matriz  $y$  em função de cada coluna da matriz  $x$ . Neste caso, as matrizes devem ter as mesmas dimensões;
4. Se  $y$  é um vetor, a função `plot2d()` permite traçar o gráfico do vetor  $y$  em função do vetor `[1:size(y)]`, e

---

<sup>2</sup>O comando `plot()` está obsoleto.

5. Se  $y$  é uma matriz, a função `plot2d()` permite traçar o gráfico da matriz  $y$  em função do vetor `[1:size(y)]`.

Vamos apresentar exemplos de gráficos gerados para cada uma das opções de entrada  $x$ ,  $y$  apresentadas anteriormente. Os gráficos serão gerados no intervalo  $[0, 2\pi]$ , com incremento 0.1. No ambiente Scilab, temos,

```
-->// Definindo o vetor das abcissas, x

-->x = [0:0.1:2*%pi]; // Intervalo [0, 2pi], incremento 0.1

-->// Item 1 - x vetor, y vetor

-->y = sin(x);

--> // Os dois vetores devem ter a mesma dimensao

-->size(x)
ans =

!   1.   63. !   // Observar que x possui 63 colunas

-->size(y)
ans =

!   1.   63. !   // Observar que y possui 63 colunas

-->plot2d(x,y)

-->clf()           // Limpa a tela grafica - evitar sobreposicao

--> // Item 2 - x vetor, y matriz

-->Y = [sin(x)' cos(x)']; // Definindo a matriz Y

--> // Observar que a matriz Y possui 63 elementos em cada coluna

-->size(Y)
ans =

!   63.   2. !

-->plot2d(x,Y)

-->clf()

-->// Item 3 - x e y sao matrizes

-->// Definindo uma variavel auxiliar

-->t = [0:0.1:2*%pi];
```

```
-->// Criando a matriz X

-->X = [t' t'];

--> // A matriz X possui 63 elementos em cada coluna

-->size(X)
ans =

! 63.    2. !

-->// Criando a matriz Y

-->Y = [cos(t)' sin(t)'];

-->// A matriz Y possui 63 elementos em cada coluna

-->size(Y)
ans =

! 63.    2. !

-->plot2d(X,Y)

-->clf()

-->// Item 4 - y vetor (sem x explicito)

-->plot2d(sin(x))

-->clf()

--> // Item 5 - Y matriz (sem x explicito)

-->plot2d(Y)

-->
```

Verificar que, após a apresentação de cada gráfico, limpamos a tela através do comando `clf()`, para evitar que o próximo gráfico não se sobreponha ao anterior.

Os resultados dessa sessão, para cada um de seus itens, podem ser agrupados em uma única janela gráfica, como mostrado na Figura 6.2.



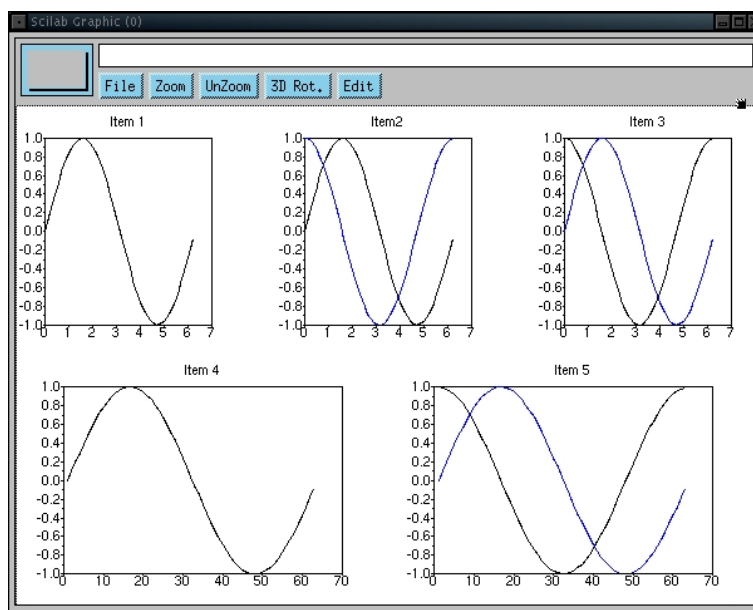


Figura 6.2: Saídas para a função `plot2d([x], y)`. Cada sub-gráfico refere-se a um dos itens da sessão do Scilab mostrada anteriormente. Observar que os gráficos dos Itens 4 e 5 possuem valores de abcissas diferentes dos demais.

Na geração da Figura 6.2, foram utilizados os comandos

```
xtitle(titulo)
```

para colocar o título em cada um dos gráficos apresentados e

```
xsetech([x, y, largura, altura])
```

para sub-dividir a janela gráfica do Scilab. O argumento do comando `xtitle()` é uma *string* especificando o título do gráfico. O argumento da função `xsetech` é explicado em seguida.

A janela gráfica do Scilab, mostrada na Figura 6.1, é definida com largura e altura iguais a 1 e com o seu sistema de referências com origem (0,0) no canto superior esquerdo da janela. O eixo x possui valores crescentes para a direita e o eixo y possui valores crescentes para baixo. Ambos os eixos possuem valores máximos iguais a 1.

O comando `xsetech()` permite criar sub-janelas gráficas dentro da janela gráfica do Scilab. Por exemplo, para o gráfico com título Item 1, foi utilizada a linha de comando

```
-->xsetech([0, 0, 0.3, 0.5]); xtitle('Item 1'); plot2d(x,y)
```

Os demais gráficos foram traçados com comandos semelhantes.

Nesse caso, o argumento utilizado na função `xsetech()` permitiu traçar um gráfico de largura 0.3 e altura 0.5 a partir da origem do sistema de referências da janela gráfica. O gráfico com título Item 2, foi traçado com o comando

```
-->xsetech([0.35, 0, 0.3, 0.5]); xtitle('Item 2'); plot2d(x,Y)
```

Todos os comandos utilizados na sessão anterior foram gravados em um arquivo através do comando `diary()`. Este arquivo, após ser editado, foi transformado no *script* apresentado em Código 6,

```

// Script para gerar a Figura 2 do capítulo 6
// Graficos - Scilab 3.0

// Definindo o vetor das abcissas, x

x = [0:0.1:2*%pi];

// Item 1 - y vetor

y = sin(x);

// xsetech[abscissa, ordenada, largura, altura] do grafico
xsetech([0, 0, 0.3, 0.5]); xtitle("Item 1"); plot2d(x,y)

// Item 2 - y matriz

Y = [sin(x)' cos(x)']; // Definindo a matriz Y

xsetech([0.35, 0, 0.3, 0.5]); xtitle("Item2"); plot2d(x,Y)

// Item 3 - x e y sao matrizes

// Definindo uma variavel auxiliar

t = [0:0.1:2*%pi];

// Criando a matriz X

X = [t' t'];

// Criando a matriz Y

Y = [cos(t)' sin(t)'];

xsetech([0.70, 0, 0.3, 0.5]); xtitle("Item 3"); plot2d(X,Y)

// Item 4 - y vetor

xsetech([0, 0.5, 0.5, 0.5]); xtitle("Item 4"); plot2d(sin(x))

// Item 5 - Y matriz

xsetech([0.5, 0.5, 0.5, 0.5]); xtitle("Item 5"); plot2d(Y)

```

Código 6: O *script* utilizado para gerar o gráfico da Figura 6.2.

Esse *script*, quando executado, gera o gráfico apresentado na Figura 6.2.

A forma geral para da função `plot2d()` inclui um terceiro argumento, `<opt_args>`,

$$\text{plot2d}([x], y, \langle \text{opt\_args} \rangle)$$

onde `<opt_args>` é uma seqüência de opções que determinam as características do gráfico bi-dimensional,

$$\langle \text{opt\_args} \rangle := \text{opcao}_1 = \text{valor}_1, \text{opcao}_2 = \text{valor}_2, \dots, \text{opcao}_n = \text{valor}_n$$

As opções podem ser:

- **style** - é utilizada para especificar o padrão para a curva (ou curvas) que estão sendo traçadas. O valor associado à essa opção deve ser um vetor com valores inteiros positivos ou negativos. Se o valor associado for positivo, a curva é contínua. Nesse caso, o valor associado à opção define, também, a cor da curva que está sendo traçada. Se o valor associado à opção for negativo ou zero, a curva será desenhada usando marcadores.
- **logflag** - define a escala, logarítmica ou linear, a ser utilizada nos eixos **x** e **y** do gráfico. Os valores associados à essa opção são *strings*, “**nn**”, “**n1**”, “**1n**” e “**11**”, onde **1** indica a escala logarítmica, **n** indica escala normal e a segunda letra indica o tipo de graduação dos eixos (normal ou logarítmica). O valor *default* desta opção é “**nn**”, isto é, escala normal com graduação normal dos eixos;
- **rect** - é utilizada para estabelecer os limites do gráfico. O valor associado à essa opção é um vetor real com quatro entradas, [**xmin**, **ymin**, **xmax**, **ymax**], onde **xmin**, **xmax** e **ymin**, **ymax** indicam os valores mínimo e máximo para os eixos **x** e **y**, respectivamente;
- **frameflag** - É utilizada para controlar a escala dos eixos coordenados. O valor associado à essa opção é um número inteiro no intervalo 0 e 8, inclusive;
- **axesflag** - especifica como os eixos serão traçados. O valor associado à essa opção é um número inteiro variando entre 0 e 5, inclusive;
- **nax** - permite definir os nomes e as marcas nos eixos **x** e **y**. O valor associado à essa opção, válido apenas quando a opção **axesflag=1**, é um vetor com quatro entradas inteiras, [**nx**, **Nx**, **ny**, **Ny**]. O parâmetro **Nx** é o número de marcações principais (*tics*) utilizadas no eixo **x**; **nx** é o número de divisões (*subtics*) entre as marcações principais do eixo **x**; **Ny** e **ny** têm significados semelhantes, tratando-se do eixo **y**, e
- **leg** - permite definir as legendas das curvas. O valor associado à esse parâmetro é uma *string* de caracteres para cada gráfico traçado.

A sessão Scilab,

```
-->x = [-%pi:0.1:%pi];
-->y = [sin(x)' cos(x)'];
-->plot2d(x,y, style=[1, -1], rect=[-%pi, -1.5, %pi, 1.5],axesflag=5, ...
-->leg = "sin(x)@cos(x)")
-->
```

exemplifica a forma de utilização de algumas opções do comando `plot2d()`. A saída é mostrada na Figura 6.3.

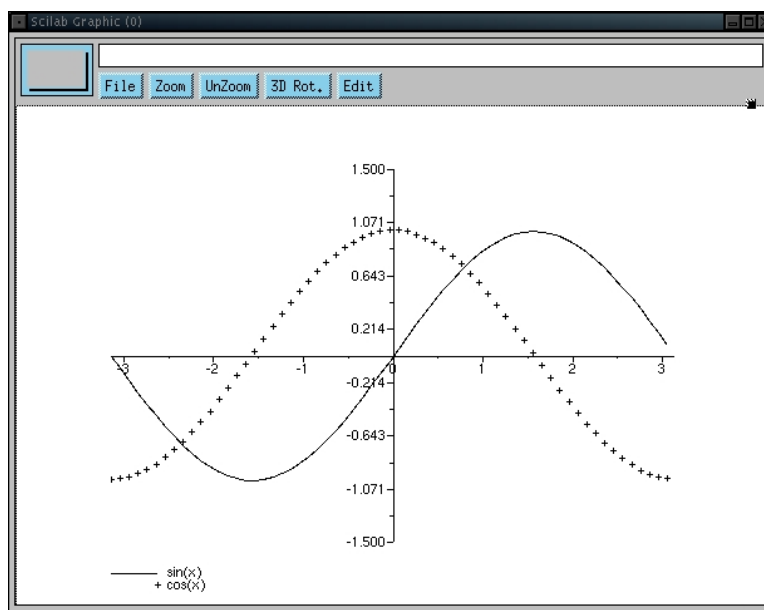


Figura 6.3: Saídas para a função `plot2d([x], y, <opt_args>)`.

Observar que a primeira curva,  $\sin(x)$ , é contínua já que o valor do parâmetro `style` associado a ela é positivo enquanto a segunda curva,  $\cos(x)$ , está desenhada com marcas já que o parâmetro `style` associado a ela é negativo. A opção `rect` estabelece os limites dos eixos `x` e `y`, enquanto a opção `axesflag=5` traça os eixos na forma apresentada. Finalmente, a opção `leg` associa uma legenda a cada curva desenhada.

O comando `plot2d()` apresenta algumas variações<sup>3</sup>, como mostrado na Tabela 6.1.

Comando	Tipo de Gráfico
<code>plot2d2()</code>	gráficos 2-D linearizados
<code>plot2d3()</code>	gráficos 2-D com barras verticais
<code>plot2d4()</code>	gráficos 2-D com setas

Tabela 6.1: Variações do comando `plot2d()`

A sub-opção `Graphics` da opção `Demos` apresenta exemplos de utilização da função `plot2d()` e de suas variações. É importante lembrar que o `demo` de uma função gráfica também pode ser ativado através da chamada da função. Por exemplo, para ativar o `demo` da função gráfica `histplot`, que plota um histograma, basta fazer:

```
-->histplot()
```

### 6.2.1 Outros Comandos

Existem comandos que podem ser utilizados para melhorar a apresentação de um gráfico. Dentre eles, destacamos:

- `xgrid` - coloca uma grade em um gráfico bi-dimensional.

<sup>3</sup>O comando `plot2d1()`, para traçar gráficos bi-dimensionais em escala logarítmica está obsoleto. Utilizar `plot2d()` com a opção `logflag` adequada

- `xtitle` - coloca títulos em gráficos 2-D ou 3-D;
- `titlepage` - coloca um título no meio de uma janela gráfica.

Os elementos de um gráfico são controlados por parâmetros globais. Estes parâmetros definem um contexto no qual o gráfico está inserido. Outros parâmetros dos gráficos são controlados através de argumentos dos próprios comandos usados para traça-los. O comando `xset`, usado sem argumento, `xset()`, apresenta um conjunto de opções, chamadas de “Contexto gráfico da janela gráfica 0”, (*Graphic context of graphic window 0*), que permitem alterar parâmetros através da utilização do *mouse*.

O comando

```
subplot(m,n,p)
```

permite dividir a janela gráfica do Scilab em uma matriz  $m \times n$ . Em cada um dos elementos da “matriz”, especificado por `p`, pode ser colocado um gráfico. A sessão a seguir ilustra os procedimentos para utilização do comando `subplot()` usando os próprios *demos* de funções gráficas do Scilab.

```
-->// Demonstracao do comando subplot

-->subplot(221)

-->champ // chamada do demo da funcao champ
Demo of champ
champ(1:10,1:10,rand(10,10),rand(10,10),1.0);

-->subplot(222)

-->histplot // chamada do demo da funcao histplot
histplot([-6:0.2:6],rand(1,2000,'n'),[1,-1], '011', ' ', [-6,0,6,0.5], [2,12,2,11]);
deff(' [y]=f(x) ', 'y=exp(-x.*x/2)/sqrt(2*%pi); ');
x=-6:0.1:6;x=x';plot2d(x,f(x),1,"000");
titre= 'macro histplot : Histogram plot';
xtitle(titre,'Classes','N(C)/Nmax');

-->subplot(223)

-->errbar // chamada do demo da funcao errbar
x=0:0.1:2*%pi;
y=[sin(x);cos(x)]';x=[x;x]';
plot2d(x,y);
errbar(x,y,0.05*ones(x),0.03*ones(x));

-->subplot(224)

-->grayplot // chamada do demo da funcao grayplot
Demo of grayplot
t=-%pi:0.1:%pi;m=sin(t)*cos(t);grayplot(t,t,m);

-->
```

A Figura 6.4 o resultado dos comandos da sessão anterior.

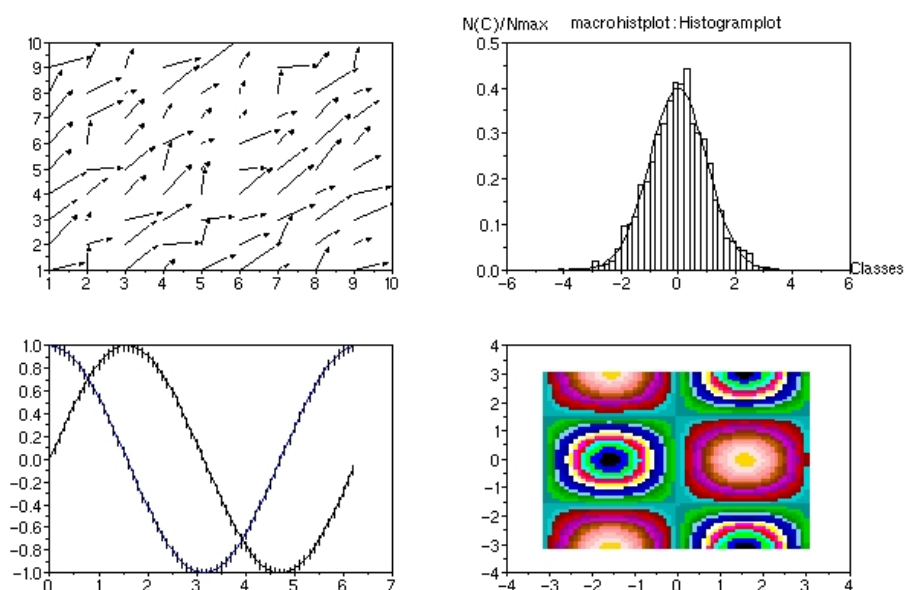


Figura 6.4: Saídas para a função subplot().

Observar que essa a Figura 6.4 não foi capturada pelo GIMP. Ela foi armazenada em um arquivo através da sub-opção Export da opção **File** da janela gráfica na qual o gráfico foi gerado. A sub-opção Export possui as opções mostradas na Figura 6.5.

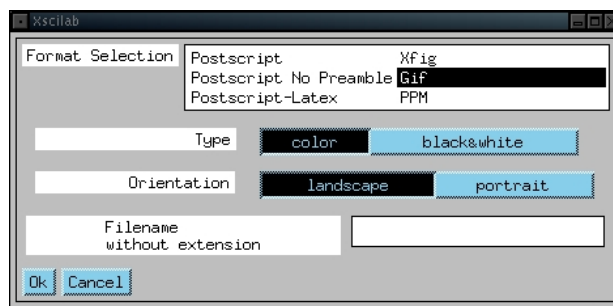


Figura 6.5: Exportando gráficos para o  $\text{\LaTeX}$ .

Como o arquivo deste trabalho é gerado diretamente em pdf, o arquivo gráfico foi salvo no formato gif e transformado em jpg através do GIMP.

### 6.2.2 Gráficos 2D Especiais

Scilab dispõe de alguns comandos que permitem traçar gráficos bi-dimensionais especiais. Por exemplo, na área de controle de processos, temos:

- `bode` - permite traçar o gráfico de módulo e fase da resposta em frequência de um sistema linear;
- `gainplot` - permite traçar o gráfico do módulo da resposta em frequência de um sistema linear;

- `nyquist` - permite traçar o gráfico da parte imaginária versus parte real da resposta em frequência de um sistema linear;
- `m_cicle` - gráfico M-círculo usado com o gráfico de Nyquist;
- `chart` - permite traçar a diagrama de Nichols;
- `black` - permite traçar o diagrama de Black para um sistema linear;
- `evans` - permite traçar o o lugar das raízes pelo método de Evans;
- `plzr` - permite traçar o diagrama de polos e zeros.

O *help* do Scilab fornece informações mais detalhadas sobre a utilização dessas funções.

### 6.3 Gráficos Tri-dimensionais

O comando `plot3d()` permite traçar gráficos de superfícies,

$$z = f(x, y)$$

Na notação Scilab, as variáveis independentes  $x$  e  $y$  são vetores de dimensões  $n1$  e  $n2$ , respectivamente, e a variável dependente  $z$  é uma matriz de dimensão  $n1 \times n2$ . Então, por essa definição, o elemento  $z(i, j)$  é o valor da superfície no ponto  $(x(i), y(j))$ .

Para exemplificar o uso de `plot3d()`, vamos considerar a função

$$\cos(x) * \sin(y)$$

no intervalo  $[0, 2\pi]$  com incremento igual a 0.1. A sessão Scilab,

```
-->x = [0:0.1:2*%pi]';
-->y = x;
-->z = cos(x) * sin(x');
-->plot3d(x, x, z)
-->
```

gera o gráfico mostrado na Figura 6.6.

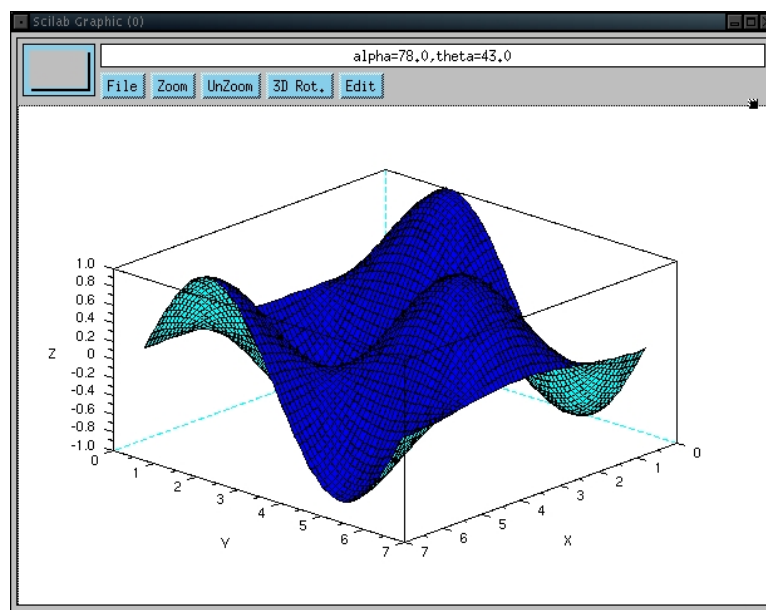


Figura 6.6: Exemplo de saída gráfica 3-D.

Além da função `plot3d()`, e de suas variações, Scilab implementa outras funções que permitem traçar gráficos tri-dimensionais. Dentre elas, destacamos:

- `fplot3d` - que permite traçar gráficos de superfícies definidas por funções, como no *script* mostrado no exemplo:

```
def('z=f(x,y)', 'z=x^4-y^4')
x=-3:0.2:3 ;y=x ;
clf() ;fplot3d(x,y,f,alpha=5,theta=31)
```

- `fplot3d1` - que permite traçar gráficos de superfícies definidas por funções, com o no caso anterior. As superfícies são apresentadas em escala cinza ou com uma graduação de cores.

### 6.3.1 Gráficos 3-D Especiais

As seguintes funções permitem traçar gráficos tri-dimensionais especiais:

- `param3d` - permite traçar curvas paramétricas;
- `hist3d` - permite traçar histogramas 3-D;
- `contour` - permite traçar curvas de nível para uma função 3-D.

Na Figura 6.7, utilizamos o comando `subplot()` para dividir a janela gráfica do Scilab e mostrar exemplos de utilização de algumas funções que geram gráficos 3-D especiais.



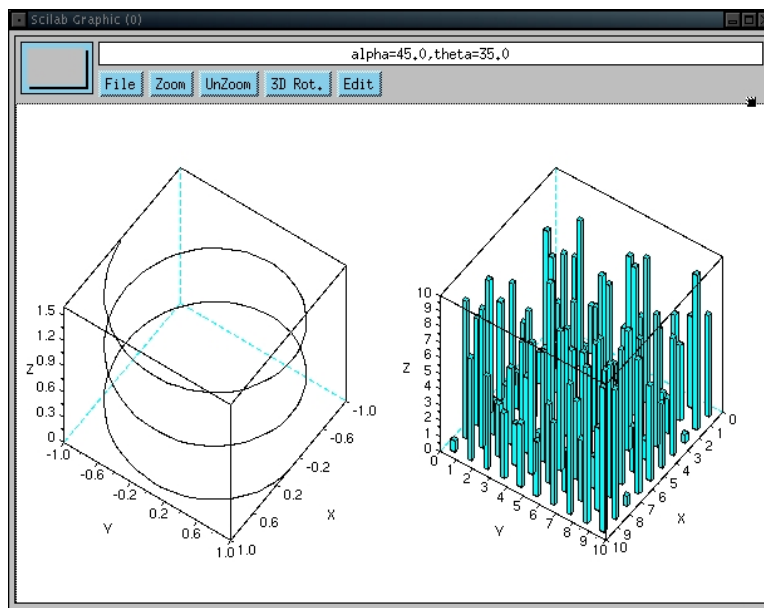


Figura 6.7: Exemplos de gráficos 3-D especiais.

Para gerar os gráficos apresentados na Figura 6.7, utilizamos os comandos:

```
-->// Exemplos de graficos 3-D especiais

-->subplot(121);

-->param3d
Demo of param3d
t=0:0.1:5*%pi;param3d(sin(t),cos(t),t/10,35,45,'X@Y@Z',[2,4]);

-->subplot(122)

-->hist3d
hist3d(10*rand(10,10));

-->
```

Ressaltamos que a sintaxe de todos esses comandos pode ser verificada usando o `help` do Scilab.

# Apêndice A

## Instalação do Scilab

O objetivo deste Apêndice é apresentar os procedimentos necessários à instalação do Scilab a partir de seu código fonte. A instalação é feita no ambiente Linux distribuição Slackware 9.1, com kernel 2.4.24. Apesar de apresentarmos os procedimentos de instalação em uma plataforma específica, eles são válidos para qualquer ambiente operacional que disponha dos requisitos mínimos exigidos, apresentados na seção A.1.

Os procedimentos necessários à instalação de outras distribuições, binárias ou pré-compiladas, podem ser encontrados na *homepage* do Scilab<sup>1</sup>. Na data em que este documento foi escrito, estão disponíveis, na *homepage* do Scilab, arquivos para as seguintes plataformas:

- Plataformas UNIX/Linux:
  - Scilab 3.0 - arquivo binário para Linux (scilab-3.0.bin.linux-i686.tar.gz);
  - Scilab 3.0 - arquivo com o código fonte do Scilab (scilab-3.0.src.tar.gz).
- Plataformas Windows 9X/NT/2000/XP
  - Scilab 3.0 - instalador da versão binária do Scilab (scilab3.0.exe);
  - Scilab 3.0 - código fonte do Scilab (scilab-3.0.src.zip)

### A.1 Instalação no Linux Slackware - Código Fonte

O código fonte para a versão 3.0 do software Scilab é disponibilizado através do arquivo `scilab-3.0-src.tar.gz` na homepage do Scilab. Para instalar o software a partir do código fonte, devemos garantir que:

- O X-Window esteja instalado (X11R4, X11R5 ou X11R6), e
- O sistema disponha de compiladores C e FORTRAN (cc e g77).

Para que o *help* do Scilab seja instalado, é necessário que o sistema operacional disponha do software Sablotron, disponível em [http://www.gingerall.com/charlie/ga/xml/p\\_sab.xml](http://www.gingerall.com/charlie/ga/xml/p_sab.xml).

Tendo garantido os requisitos mínimos, escolhemos o diretório no qual o software será instalado. Geralmente, utilizamos o diretório `/usr/local`. Copiamos, então, a distribuição fonte do Scilab para este diretório. Descompactamos o arquivo através do comando `tar -zxvf scilab-3.0-src.tar.gz`. Será criado um diretório `/usr/local/scilab-3.0` onde estarão colocados todos os arquivos que compõem o software. O diretório `/usr/local/scilab-3.0` é chamado de diretório principal do Scilab, referenciado pela variável de ambiente `SCIDIR`. Estes procedimentos devem ser realizados como `root`.

---

<sup>1</sup><http://scilabsoft.inria.fr/>

A instalação do software propriamente dita é bastante simples. No diretório SCIDIR digita-se `.\configure` e, depois, `make all`. É interessante executar, nesse momento, um teste da instalação. Isso pode ser feito através do comando `make tests` executando dentro do diretório SCIDIR/`tests`. Para complementar a instalação, o usuário deve, também, acrescentar à variável PATH o caminho `/usr/local/scilab-3.0/bin` para que o Scilab possa ser executado de qualquer diretório<sup>2</sup>. Mais detalhes sobre opções de configuração podem ser encontrados no arquivo `/usr/local/scilab-3.0/README_Unix`

Para executar o programa, basta digitar `scilab` no ambiente gráfico ou `scilab -nw` no ambiente texto do Linux.

No diretório SCIDIR=`/usr/local/scilab-3.0` estão localizados, entre outros, os seguintes arquivos :

- `scilab.star` - arquivo de inicialização do Scilab. As instruções deste arquivo são executadas quando Scilab é ativado. O usuário também pode ter seu próprio arquivo de inicialização, `.scilab`, dentro do seu diretório;
- `license.txt` - arquivo contendo informações relativas ao uso do software, e
- `configure` - arquivo de preparação para a instalação. Este arquivo modificará outros arquivos do Scilab, os `Makefile`, de acordo com a configuração do Sistema Operacional no qual Scilab será instalado.

Em SCIDIR são criados os seguintes diretórios :

- `bin/` - onde estão localizados os arquivos executáveis. Nas distribuições Unix/Linux o script `scilab` aciona o executável `scilex`. Neste diretório estão, também, localizados programas para manipulação de arquivos Postscript e  $\LaTeX$  gerados pelo Scilab;
- `config/` - onde estão localizados os arquivos utilizados pelo arquivo de configuração `configure`;
- `contrib/` - onde estão colocados os arquivos correspondentes aos *toolboxes* instalados;
- `demos/` - onde estão localizados os arquivos de demonstração;
- `examples/` - onde estão localizados arquivos com exemplos de como ligar o Scilab com programas externos;
- `imp/` - onde estão localizados os códigos fontes dos programas que permitem a manipulação de arquivos Postscript;
- `intersi/` - onde estão localizados os arquivos que permitem a construção das interfaces necessárias para adicionar novas primitivas escritas em FORTRAN ou C ao Scilab;
- `libs/` - onde estão localizados os códigos objeto das bibliotecas do Scilab;
- `macros/` - onde estão localizadas as funções do Scilab. Este diretório contém diversos sub-diretórios correspondendo, cada um, a um tópico específico. Por exemplo, no diretório `signal` estão localizadas as funções que permitem trabalhar com aspectos relacionados com a área de processamento de sinais. Cada sub-diretório contém o código fonte das funções (arquivos com extensão `.sci`);
- `man/` - onde estão localizados diversos sub-diretórios contendo arquivos man (help *on-line*) dos comandos e funções que compõem o Scilab;

---

<sup>2</sup>No Linux distribuição Slackware, editar o arquivo `/etc/profile` adicionando o caminho `/usr/local/scilab-3.0/bin` à variável de ambiente PATH

- `maple/` - onde estão localizados arquivos escritos na linguagem Maple que permitem a ligação do software de computação simbólica Maple com Scilab;
- `pvm3/` - implementação da versão 3.4 do PVM (*Parallel Virtual Machine System*) para Scilab;
- `routines/` - onde estão as rotinas numéricas em C e em FORTRAN utilizadas pelo Scilab, divididas por sub-diretórios;
- `scripts/` - onde estão localizados os fontes de alguns *script* utilizados pelo Scilab;
- `tcl/` - implementação TCL/TK para Scilab;
- `tests/` - onde estão localizados arquivos para a realização de testes da instalação do Scilab, e
- `util/` - onde estão localizadas rotinas e arquivos para gerenciamento do Scilab.

## Apêndice B

# Ligação do Scilab com Programas em C

Os programas escritos na linguagem Scilab são interpretados. Estes programas, principalmente os que realizam cálculos numéricos utilizados em simulações ou otimizações, podem ser lentos em comparação com os programas compilados escritos em linguagens convencionais. Uma maneira de acelerar a execução desses programas, sem perder a flexibilidade disponibilizada pelo Scilab, é escrever o código lento em uma linguagem convencional e utilizar este código dentro do ambiente Scilab.

O Scilab permite que rotinas ou funções escritos em FORTRAN ou C sejam utilizados dentro de seu ambiente. Neste Apêndice, apresentamos os procedimentos necessários à ligação de programas Scilab com funções escritas na linguagem C.

Uma função escrita na linguagem C pode ser ligada ao Scilab de três maneiras distintas :

- através do comando `link`, em um processo chamado de ligação dinâmica;
- através de programas de interface, os chamados *gateways*, escritos pelo usuário ou gerados por `intersi`, ou
- através da adição de uma nova função ao código do Scilab.

Apenas a primeira maneira será apresentada. Os demais casos podem ser verificados em [1].

### B.1 A Ligação Dinâmica

O comando

```
--> link('foo.o', 'foo', 'c')
```

```
-->
```

liga o arquivo com o código objeto da função `foo`, `foo.o`, escrita na linguagem C, indicado pelo terceiro argumento, `c`, ao Scilab. O segundo argumento de `link`, `foo`, é o nome da função a ser executada. Um arquivo objeto pode conter várias funções. O nome de cada uma delas, se necessário, deve ser indicado como segundo argumento de `link` na forma de um vetor de *strings*, `['prog1', 'prog2']`.

Para exemplificar a ligação de uma função escrita em C com um programa em Scilab, vamos re-escrever em C o programa Scilab que implementa o método de Runge-Kutta de 4<sup>a</sup> ordem apresentado no Capítulo 5. Algumas observações devem ser feitas :

- O programa transforma-se em uma função;

- Além do intervalo de integração,  $[a, b]$ , do passo de integração,  $h$  e da condição inicial em  $y$ ,  $y_0$ , que são os argumentos de entrada da função, os vetores  $x$  e  $y$  são explicitados na chamada da função e são os seus argumentos de retorno ou de saída;
- As variáveis de entrada da função principal, `rk4`, são passadas como apontadores;
- A ordem dos argumentos na chamada da função é relevante;
- A função a ser integrada faz parte do arquivo que contém a função principal;

O programa, transformado em uma função C, e a função com a equação a ser integrada, são mostrados em Código 7.

```

1  /*
2      Exemplo de utilizacao do comando link.
3      Resolucao de equacoes diferenciais ordinarias por Runge-Kutta
4      de 4a. ordem.
5
6      Entrada : [a,b] - intervalo de integracao
7                h - passo da integracao
8                y0 - condicao inicial em x0
9  */
10
11 double rk4(x, y, a, b, h, y0)
12 double x[], y[], *a, *b, *h, *y0;
13 {
14     int n, k;
15     double hf1, hf2, hf3, hf4;
16     double f();
17
18     n = (*b - *a) / (*h);
19
20     x[0] = *a;
21     y[0] = *y0;
22
23     for (k = 0; k < n; ++k)
24     {
25         hf1 = (*h) * f(x[k], y[k]);
26         hf2 = (*h) * f(x[k] + (*h)/2, y[k] + hf1/2);
27         hf3 = (*h) * f(x[k] + (*h)/2, y[k] + hf2/2);
28         hf4 = (*h) * f(x[k] + (*h), y[k] + hf3);
29
30         y[k+1] = y[k] + (hf1 + 2*hf2 + 2*hf3 + hf4)/6;
31         x[k+1] = x[k] + (*h);
32     }
33 }
34
35 /*
36     Funcao a ser integrada
37 */
38
39 double f(x,y)
40 double x, y;
41 {
42     return( (x - y)/2 );
43 }

```

Código 7: Função Runge-Kutta escrita em C.

Usando o compilador `gcc`, geramos o código objeto `runge.o` para o programa `runge.c` através da linha de comando:

```
paulo@pmotta:~$ gcc -c runge.c -o runge.o
```

Em seguida, no ambiente Scilab, o código objeto é ligado ao Scilab através do comando `link`,

```
-->link('runge.o','rk4','c')
linking files runge.o to create a shared executable
shared archive loaded
Linking rk4
Link done
ans =

    0.

-->
```

Observar que `rk4`, segundo argumento de `link` é o nome da rotina que resolve o problema.

Para acessar a função `rk4`, devemos inicializar seus parâmetros de entrada. A inicialização desses parâmetros é feita através dos comandos,

```
->a = 0          // Valor inicial do intervalo

    0.

-->b = 3          // Valor final do intervalo

    3.

-->h = 1/8       // Passo da integracao

    0.125

-->y0 = 1         // Valor da condicao inicial em y

    1.

-->
```

Em seguida, usamos a função `call` para rodar a função `rk4`,

```
-->[X,Y]=call('rk4',a,3,'d',b,4,'d',h,5,'d',y0,6,'d','out', ...
-->[25,1],1,'d',[25,1],2,'d');
```

O primeiro argumento da função `call` é o nome da função `rk4`. Cada argumento de entrada da função `rk4` deve ser acompanhado da posição em que ele ocupa na lista de argumentos da função chamada, `rk4`, e o seu tipo de dado. Assim, considerando

```
double rk4(x, y, a, b, h, y0),
```

vemos que `a` é o terceiro argumento na lista dos parâmetros de `rk4` e seu tipo de dado é `double`. Este fato é indicado na função `call` por `a,3,'d'`. Do mesmo modo, `b` é o quarto argumento, tipo de dado<sup>1</sup> `double`, indicado em `call` por `b,4,'d'`, e assim por diante.

Os argumentos de saída, especificados após `'out'`, são vetores do tipo `double`, indicado por `'d'`, com vinte e cinco linhas e uma coluna, indicados por `[25,1]`, ocupando as posições 1 e 2 da lista de argumentos da função `rk4`<sup>2</sup>.

Finalmente, os valores de retorno de `rk4` são associados às variáveis `[X, Y]` do ambiente Scilab. A resposta, então, é

```
-->[X Y]
ans =

!  0.      1.      !
!  0.125   0.9432392 !
!  0.25    0.8974908 !
!  0.375   0.8620874 !
!  0.5     0.8364024 !
!  0.625   0.8198470 !
!  0.75    0.8118679 !
!  0.875   0.8119457 !
!  1.      0.8195921 !
!  1.125   0.8343486 !
!  1.25    0.8557844 !
!  1.375   0.8834949 !
!  1.5     0.9170998 !
!  1.625   0.9562421 !
!  1.75    1.0005862 !
!  1.875   1.049817  !
!  2.      1.1036385 !
!  2.125   1.1617724 !
!  2.25    1.2239575 !
!  2.375   1.2899485 !
!  2.5     1.3595145 !
!  2.625   1.4324392 !
!  2.75    1.5085189 !
!  2.875   1.5875626 !
!  3.      1.6693906 !

-->
```

Observar que a função `call` foi chamada com um ponto-e-vírgula no final, suprimindo a apresentação imediata dos resultados. Só depois, os vetores `X` e `Y` foram mostrados. Neste caso, o ponto-e-vírgula evitou que primeiro fosse mostrado o vetor `Y` e, depois, o vetor `X`.

<sup>1</sup>Os outros possíveis tipos de dados são real, `r` e inteiro, `i`

<sup>2</sup>Observar que, na resposta, são  $n = (b - a)/h$  iterações mais o valor da condição inicial,  $(x_0, y_0)$



# Apêndice C

## Instalação de *Toolboxes*

O objetivo deste Apêndice é apresentar os procedimentos necessários à instalação de *toolboxes* no Scilab 3.0. A instalação é feita no ambiente Linux distribuição Slackware 9.1, com kernel 2.4.24. Os *toolboxes* são conjuntos de funções desenvolvidas com uma finalidade específica. Os *toolboxes* para Scilab estão disponíveis em <http://scilabsoft.inria.fr/contributions.html/>.

### C.1 Instalação

Scilab tem *toolboxes* disponíveis nas seguintes áreas:

- Interface gráfica com o usuário;
- Processamento e manipulação de imagens;
- Elementos finitos;
- Otimização;
- Programação linear;
- Simulação;
- Modelagem e controle, incluindo redes neurais, wavelets e aprendizagem com reforço;
- Estatística e análise de dados;
- Reconhecimento de padrões;

entre outras.

Para exemplificar os procedimentos de instalação de um *toolbox* no Scilab, vamos utilizar o *toolbox* ANN que implementa funções para análise de sinais utilizando redes neurais. A instalação será realizada pelo usuário *root*.

O *toolbox* ANN está disponível no arquivo ANN\_Toolbox\_0.4.2.tar.gz que pode ser obtido em <ftp://ftp.inria.fr/INRIA/Scilab/contrib/ANN/>. A primeira providência, então, é abrir este arquivo em um diretório adequado. Através do comando

```
tar -zxvf ANN_Toolbox_0.4.2.tar.gz,
```

executado como *root* dentro do diretório `/usr/local`, criamos o diretório

```
/usr/local/ANN_Toolbox_0.4.2.
```

Neste diretório estão localizados os códigos fontes das funções que compõem o *toolbox*.

O próximo passo é executar o *script* `builder.sce`,

```
--> exec /usr/local/ANN_Toolbox_0.4.2/builder.sce
```

para instalar os comandos e funções do *toolbox*. Este procedimento pode, por exemplo, ser executado utilizando-se o Scilab em modo texto.

```
root@none:/usr/local/ANN_Toolbox_0.4.2# scilab -nw
```

```
-----  
Scilab-3.0
```

```
Copyright (c) 1989-2004  
Consortium Scilab (INRIA, ENPC)  
-----
```

Startup execution:

```
loading initial environment
```

```
-->exec /usr/local/ANN_Toolbox_0.4.2/builder.sce
```

```
-->mode(-1)
```

```
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ANN.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ANN_FF.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ANN_GEN.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_ConjugGrad.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_Hess.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_INT.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_Jacobian.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_Jacobian_BP.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_Mom_batch.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_Mom_batch_nb.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_Mom_online.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_Mom_online_nb.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_SSAB_batch.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_SSAB_batch_nb.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_SSAB_online.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_SSAB_online_nb.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_Std_batch.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_Std_batch_nb.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_Std_online.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_Std_online_nb.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_VHess.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_grad.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_grad_BP.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_grad_BP_nb.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_init.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_init_nb.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_run.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_FF_run_nb.man to ascii  
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_d_log_activ.man to ascii
```

```
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_d_sum_of_sqr.man to ascii
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_log_activ.man to ascii
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_pat_shuffle.man to ascii
Processing /usr/local/ANN_Toolbox_0.4.2/man/ann/ann_sum_of_sqr.man to ascii
```

```
-->
```

Após ser instalado, os comandos e funções do *toolbox* poderão ser usados por qualquer usuário bastando que ele execute o *script loader.sce*,

```
-->exec /usr/local/ANN_Toolbox_0.4.2/loader.sce
```

para carregar os comandos e funções do *toolbox* ANN no ambiente do Scilab.

Na Figura C.1, mostramos o procedimento para a utilização do *toolbox*. Observar que as funções do *toolbox* instalado já aparecem no help.

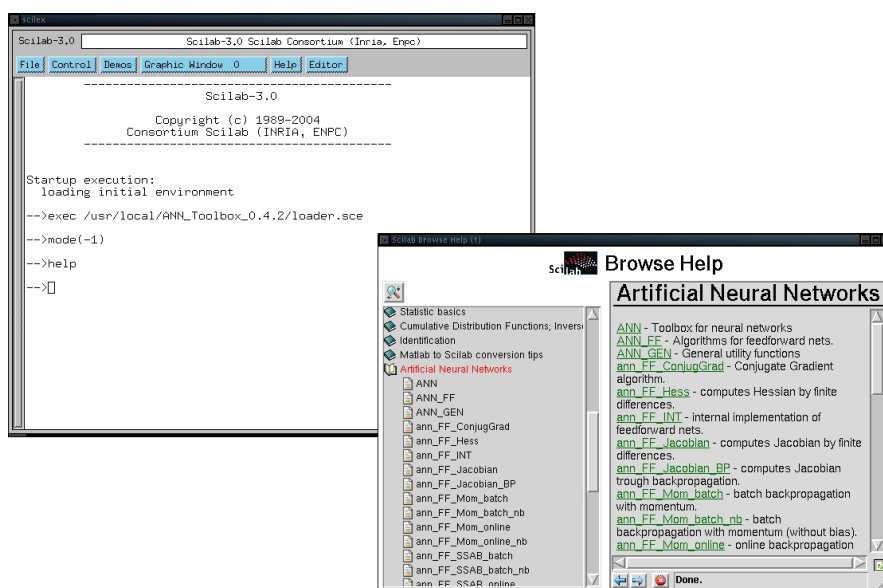


Figura C.1: Procedimentos para a utilização do *toolbox* ANN e help com as funções disponíveis no *toolbox*.

A instalação e o acesso a outros *toolboxes* do Scilab é feita a partir de procedimentos semelhantes aos apresentados neste Apêndice.

# Apêndice D

## Funções Pré-definidas - Scilab 3.0

Apresentamos a relação de todas as funções disponíveis no Scilab 3.0. Esta relação foi obtida a partir de uma busca automática realizada no diretório `SCI/man/eng/`. Entretanto, alguns caracteres foram ajustados (ainda) manualmente à sintaxe do `LATEX`.

### D.1 Programming

- `abort` - interrupt evaluation.
- `ans` - answer
- `backslash` - (`\`) left matrix division.
- `bool2s` - convert boolean matrix to a zero one matrix.
- `boolean` - Scilab Objects, boolean variables and operators `&`, `|`, `~`
- `brackets` - (`[,]`) left and right brackets
- `break` - keyword to interrupt loops
- `call` - Fortran or C user routines call
- `case` - keyword used in select
- `clear` - kills variables
- `clearglobal` - kills global variables
- `colon` - (`:`) colon operator
- `comma` - (`,`) column, instruction, argument separator
- `comments` - comments
- `continue` - keyword to pass control to the next iteration of a loop
- `date` - Current date as date string
- `debug` - debugging level
- `definedfields` - return index of list's defined fields
- `dot` - (`.`) symbol
- `else` - keyword in if-then-else
- `elseif` - keyword in if-then-else
- `empty` - (`[]`) empty matrix
- `end` - end keyword
- `equal` - (`=`) affectation, comparison equal sign
- `errcatch` - error trapping
- `errclear` - error clearing
- `error` - error messages
- `etime` - Elapsed time
- `evstr` - evaluation of expressions
- `exec` - script file execution
- `execstr` - execute Scilab code in strings
- `exists` - checks variable existence
- `exit` - Ends the current Scilab session
- `external` - Scilab Object, external function or routine
- `extraction` - matrix and list entry extraction
- `feval` - multiple evaluation
- `find` - find indices of boolean vector or matrix true elements
- `for` - language keyword for loops
- `format` - number printing and display format
- `fort` - Fortran or C user routines call
- `funptr` - coding of primitives ( wizard stuff )
- `getdate` - get date and time information
- `getenv` - get the value of an environment variable
- `getfield` - list field extraction
- `getpid` - get Scilab process identificator
- `getversion` - get Scilab version name
- `global` - Define global variable
- `gstacksize` - set/get scilab global stack size
- `hat` - (`^`) exponentiation

- **host** - Unix or DOS command execution
- **hypermat** - initialize an N dimensional matrices
- **hypermatrices** - Scilab object, N dimensional matrices in Scilab
- **iconvert** - conversion to 1 or 4 byte integer representation
- **ieee** - set floating point exception mode
- **if** - conditional execution
- **assignation** - partial variable assignation
- **insertion** - partial variable assignation or modification
- **intppty** - set interface argument passing properties
- **inttype** - type integers used in integer data types
- **inv\_coeff** - build a polynomial matrix from its coefficients
- **iserror** - error occurrence test
- **isglobal** - check if a variable is global
- **lasterror** - get last recorded error message
- **left** - ([]) left bracket
- **less** - (<) lower than comparison
- **list** - Scilab object and list function definition
- **lsslist** - Scilab linear state space function definition
- **lstcat** - list concatenation
- **matrices** - Scilab object, matrices in Scilab
- **matrix** - reshape a vector or a matrix to a different size matrix
- **mlist** - Scilab object, matrix oriented typed list definition.
- **mode** - select a mode in exec file
- **mtlb\_mode** - switch Matlab like operations
- **names** - scilab names syntax
- **null** - delete an element in a list
- **overloading** - display, functions and operators overloading capabilities
- **parents** - ( ) left and right parenthesis
- **pause** - pause mode, invoke keyboard
- **percent** - (%) special character
- **plus** - (+) addition operator
- **poly** - polynomial definition
- **power** - power operation (^, .^)
- **predef** - variable protection
- **getcwd** - get Scilab current directory
- **pwd** - print Scilab current directory
- **quit** - decrease the pause level or exit
- **quote** - (') transpose operator, string delimiter
- **rational** - Scilab objects, rational in Scilab
- **resume** - return or resume execution and copy some local variables
- **return** - return or resume execution and copy some local variables
- **rlist** - Scilab rational fraction function definition
- **sciargs** - scilab command line arguments
- **select** - select keyword
- **semi** - (;) instruction and row separator
- **semicolon** - (;) ending expression and row separator
- **setfield** - list field insertion
- **slash** - (/) right division and feed back
- **stacksize** - set scilab stack size
- **star** - (\*) multiplication operator
- **symbols** - scilab operator names
- **testmatrix** - generate some particular matrices
- **then** - keyword in if-then-else
- **tic** - start a stopwatch timer
- **tilda** - (~) logical not
- **tlist** - Scilab object and typed list definition.
- **toc** - Read the stopwatch timer
- **type** - variable type
- **typename** - associates a name to variable type
- **user** - interfacing a Fortran or C routine
- **varn** - symbolic variable of a polynomial
- **what** - list the Scilab primitives
- **where** - get current instruction calling tree
- **whereami** - display current instruction calling tree
- **whereis** - name of library containing a function
- **while** - while keyword
- **who** - listing of variables
- **who\_user** - listing of user's variables
- **whos** - listing of variables in long form

## D.2 Graphics Library

- **Graphics** - graphics library overview
- **Matplot** - 2D plot of a matrix using colors
- **Matplot1** - 2D plot of a matrix using colors
- **Sfgrayplot** - smooth 2D plot of a surface defined by a function using colors
- **Sgrayplot** - smooth 2D plot of a surface using colors
- **addcolor** - add new colors to the current colormap
- **agregation\_properties** - description of the Agregation entity properties
- **alufunctions** - pixel drawing functions
- **arc\_properties** - description of the Arc entity properties
- **axes\_properties** - description of the axes entity properties
- **axis\_properties** - description of the axis entity properties
- **black** - Black's diagram (Nichols chart)
- **bode** - Bode plot
- **champ** - 2D vector field plot
- **champ1** - 2D vector field plot with colored arrows
- **champ\_properties** - description of the 2D vector field entity properties
- **chart** - Nichols chart
- **clear\_pixmap** - erase the pixmap buffer
- **clf** - clear or reset the current graphic figure (window) to default values
- **color** - returns the color id of a color
- **color\_list** - list of named colors
- **colorbar** - draw a colorbar
- **colormap** - using colormaps
- **contour** - level curves on a 3D surface
- **contour2d** - level curves of a surface on a 2D plot
- **contour2di** - compute level curves of a surface on a 2D plot
- **contourf** - filled level curves of a surface on a 2D plot
- **copy** - copy a graphics entity.
- **delete** - delete a graphic entity and its children.
- **dragrect** - Drag rectangle(s) with mouse
- **draw** - draw an entity.
- **drawaxis** - draw an axis
- **drawlater** - makes axes children invisible.
- **drawnow** - draw hidden graphics entities.
- **driver** - select a graphics driver
- **edit\_curv** - interactive graphic curve editor
- **errbar** - add vertical error bars on a 2D plot
- **eval3d** - values of a function on a grid
- **eval3dp** - compute facets of a 3D parametric surface
- **evans** - Evans root locus
- **fac3d** - 3D plot of a surface (obsolete)
- **fchamp** - direction field of a 2D first order ODE
- **fcontour** - level curves on a 3D surface defined by a function
- **fcontour2d** - level curves of a surface defined by a function on a 2D plot
- **fec** - pseudo-color plot of a function defined on a triangular mesh
- **fec\_properties** - description of the fec entities properties
- **fgrayplot** - 2D plot of a surface defined by a function using colors
- **figure\_properties** - description of the graphics figure entity properties
- **fplot2d** - 2D plot of a curve defined by a function
- **fplot3d** - 3D plot of a surface defined by a function
- **fplot3d1** - 3D gray or color level plot of a surface defined by a function
- **gainplot** - magnitude plot
- **gca** - Return handle of current axes.
- **gce** - Get current entity handle.
- **gcf** - Return handle of current graphic or GUI window.
- **gda** - Return handle of default axes.
- **gdf** - Return handle of default figure.
- **genfac3d** - compute facets of a 3D surface
- **geom3d** - projection from 3D on 2D after a 3D plot
- **get** - Retrieve a property value from a graphics entity or an User Interface object.
- **getcolor** - opens a dialog to show colors in the current colormap
- **getfont** - dialog to select font
- **getlinestyle** - dialog to select linestyle
- **getmark** - dialog to select mark (symbol)
- **getsymbol** - dialog to select a symbol and its size
- **glue** - glue a set of graphics entities into an aggregation.

- `gr_menu` - simple interactive graphic editor
- `graduate` - pretty axis graduations
- `graphics_entities` - description of the graphics entities data structures
- `graycolormap` - linear gray colormap
- `grayplot` - 2D plot of a surface using colors
- `grayplot_properties` - description of the grayplot entities properties
- `graypolarplot` - Polar 2D plot of a surface using colors
- `hist3d` - 3D representation of a histogram
- `histplot` - plot a histogram
- `hotcolormap` - red to yellow colormap
- `isoview` - set scales for isometric plot (do not change the size of the window)
- `jetcolormap` - blue to red colormap
- `label_properties` - description of the Label entity properties
- `legend` - draw graph legend
- `legend_properties` - description of the Legend entity properties
- `legends` - draw graph legend
- `loadplots` - loads and formats saved plots
- `locate` - mouse selection of a set of points
- `m_circle` - M-circle plot
- `milk_drop` - milk drop 3D function
- `move` - move, translate, a graphic entity and its children.
- `name2rgb` - returns the RGB values of a named color
- `nf3d` - rectangular facets to plot3d parameters
- `nyquist` - nyquist plot
- `param3d` - 3D plot of a parametric curve
- `param3d1` - 3D plot of parametric curves
- `param3d_properties` - description of the 3D curves entities properties
- `paramfplot2d` - animated 2D plot, curve defined by a function
- `plot` - simple plot
- `plot2d` - 2D plot
- `plot2d1` - 2D plot (logarithmic axes) (obsolete)
- `plot2d2` - 2D plot (step function)
- `plot2d3` - 2D plot (vertical bars)
- `plot2d4` - 2D plot (arrows style)
- `plot2d_old_version` - 2D plot reference
- `plot3d` - 3D plot of a surface
- `plot3d1` - 3D gray or color level plot of a surface
- `plot3d2` - plot surface defined by rectangular facets
- `plot3d3` - mesh plot surface defined by rectangular facets
- `plot3d_old_version` - 3D plot of a surface
- `plotframe` - plot a frame with scaling and grids
- `plzr` - pole-zero plot
- `polarplot` - Plot polar coordinates
- `polyline_properties` - description of the Polyline entity properties
- `printing` - printing scilab graphics
- `rectangle_properties` - description of the Rectangle entity properties
- `replot` - redraw the current graphics window with new boundaries
- `rgb2name` - returns the name of a color
- `rotate` - rotation of a set of points
- `rubberbox` - Rubberband box for rectangle selection
- `scaling` - affine transformation of a set of points
- `scf` - set the current graphic figure (window)
- `sd2sci` - gr\_menu structure to scilab instruction convertor
- `sda` - Set default axes.
- `sdf` - Set default figure.
- `secto3d` - 3D surfaces conversion
- `segs_properties` - description of the Segments entity properties
- `set` - set a property value of a graphic entity object or of a User Interface object.
- `sgrid` - s-plane grid lines.
- `show_pixmap` - send the pixmap buffer to the screen
- `square` - set scales for isometric plot (change the size of the window)
- `subplot` - divide a graphics window into a matrix of sub-windows
- `surface_properties` - description of the 3D entities properties
- `text_properties` - description of the Text entity properties
- `title_properties` - description of the Title entity properties

- **titlepage** - add a title in the middle of a graphics window
- **twinkle** - is used to have a graphics entity twinkle
- **unglue** - unglue an agragation and replace it by individual children.
- **winsid** - return the list of graphics windows
- **xarc** - draw a part of an ellipse
- **xarcs** - draw parts of a set of ellipses
- **xarrows** - draw a set of arrows
- **xaxis** - draw an axis
- **xbasc** - clear a graphics window and erase the associated recorded graphics
- **xbasimp** - send graphics to a Postscript printer or in a file
- **xbasr** - redraw a graphics window
- **xchange** - transform real to pixel coordinates
- **xclea** - erase a rectangle
- **xclear** - clear a graphics window
- **xclick** - wait for a mouse click
- **xclip** - set a clipping zone
- **xdel** - delete a graphics window
- **xend** - close a graphics session
- **xfarc** - fill a part of an ellipse
- **xfarcs** - fill parts of a set of ellipses
- **xfpoly** - fill a polygon
- **xfpolys** - fill a set of polygons
- **xfrect** - fill a rectangle
- **xget** - get current values of the graphics context
- **xgetech** - get the current graphics scale
- **xgetmouse** - get the mouse events and current position
- **xgraduate** - axis graduation
- **xgrid** - add a grid on a 2D plot
- **xinfo** - draw an info string in the message subwindow
- **xinit** - initialisation of a graphics driver
- **xlfont** - load a font in the graphic context or query loaded font
- **xload** - load a saved graphics
- **xname** - change the name of the current graphics window
- **xnumb** - draw numbers
- **xpause** - suspend Scilab
- **xpoly** - draw a polyline or a polygon
- **xpolys** - draw a set of polylines or polygons
- **xrect** - draw a rectangle
- **xrects** - draw or fill a set of rectangles
- **xrpoly** - draw a regular polygon
- **xs2fig** - send graphics to a file in Xfig syntax
- **xs2gif** - send graphics to a file in GIF syntax
- **xs2ppm** - send graphics to a file in PPM syntax
- **xs2ps** - send graphics to a file in PS syntax
- **xsave** - save graphics into a file
- **xsegs** - draw unconnected segments
- **xselect** - raise the current graphics window
- **xset** - set values of the graphics context
- **xsetech** - set the sub-window of a graphics window for plotting
- **xsetm** - dialog to set values of the graphics context
- **xstring** - draw strings
- **xstringb** - draw strings into a box
- **xstringl** - compute a box which surrounds strings
- **xtape** - set up the record process of graphics
- **xtitle** - add titles on a graphics window
- **zgrid** - zgrid plot



## D.3 Elementary Functions

- **abs** - absolute value, magnitude
- **acos** - element wise cosine inverse
- **acosh** - hyperbolic cosine inverse
- **acoshm** - matrix hyperbolic inverse cosine
- **acosm** - matrix wise cosine inverse
- **addf** - symbolic addition
- **adj2sp** - converts adjacency form into sparse matrix.
- **amell** - Jacobi's am function
- **and** - (&) logical and
- **asin** - sine inverse
- **asinh** - hyperbolic sine inverse
- **asinhm** - matrix hyperbolic inverse sine
- **asinm** - matrix wise sine inverse
- **atan** - 2-quadrant and 4-quadrant inverse tangent
- **atanh** - hyperbolic tangent inverse
- **atanhm** - matrix hyperbolic tangent inverse
- **atanm** - square matrix tangent inverse
- **besseli** - Modified Bessel functions of the first kind (I sub alpha).
- **besselj** - Bessel functions of the first kind (J sub alpha).
- **besselk** - Modified Bessel functions of the second kind (K sub alpha).
- **bessely** - Bessel functions of the second kind (Y sub alpha).
- **beta** - beta function
- **binomial** - binomial distribution probabilities
- **bloc2exp** - block-diagram to symbolic expression
- **bloc2ss** - block-diagram to state-space conversion
- **bsplin3val** - 3d spline arbitrary derivative evaluation function
- **calerf** - computes error functions.
- **ceil** - rounding up
- **cmb\_lin** - symbolic linear combination
- **conj** - conjugate
- **cos** - cosine function
- **cosh** - hyperbolic cosine
- **coshm** - matrix hyperbolic cosine
- **cosm** - matrix cosine function
- **cotg** - cotangent
- **coth** - hyperbolic cotangent
- **cothm** - matrix hyperbolic cotangent
- **cshep2d** - bidimensional cubic shepard (scattered) interpolation
- **cumprod** - cumulative product
- **cumsum** - cumulative sum
- **delip** - elliptic integral
- **diag** - diagonal including or extracting
- **diff** - Difference and discrete derivative
- **dlgamma** - derivative of gammaln function, psi function
- **double** - conversion from integer to double precision representation
- **dsearch** - binary search (aka dichotomous search in french)
- **erf** - The error function.
- **erfc** - The complementary error function.
- **erfcx** - scaled complementary error function.
- **eval** - evaluation of a matrix of strings
- **eval\_cshep2d** - bidimensional cubic shepard interpolation evaluation
- **eye** - identity matrix
- **fix** - rounding towards zero
- **floor** - rounding down
- **frexp** - dissect floating-point numbers into base 2 exponent and mantissa
- **full** - sparse to full matrix conversion
- **gamma** - The gamma function.
- **gammaln** - The logarithm of gamma function.
- **gsort** - decreasing order sorting
- **imag** - imaginary part
- **imult** - multiplication by i the imaginary unitary
- **ind2sub** - linear index to matrix subscript values
- **int** - integer part
- **int16** - conversion to 2 bytes integer representation
- **int32** - conversion to 4 bytes integer representation
- **int8** - conversion to one byte integer representation
- **uint16** - conversion to 2 bytes unsigned integer representation
- **uint32** - conversion to 4 bytes unsigned integer representation
- **uint8** - conversion to one byte unsigned integer representation

- **integrate** - integration by quadrature
- **interp** - cubic spline evaluation function
- **interp2d** - bicubic spline (2d) evaluation function
- **interp3d** - 3d spline evaluation function
- **interp1n** - linear interpolation
- **intersect** - returns the vector of common values of two vectors
- **intsplin** - integration of experimental data by spline interpolation
- **inttrap** - integration of experimental data by trapezoidal interpolation
- **isdef** - check variable existence
- **isempty** - check if a variable is an empty matrix or an empty list
- **isequal** - objects comparison
- **isinf** - check for infinite entries
- **isnan** - check for "Not a Number" entries
- **isreal** - check if a variable as real or complex entries
- **kron** - Kronecker product (.\*.)
- **ldivf** - left symbolic division
- **legendre** - associated Legendre functions
- **lex\_sort** - lexicographic matrix rows sorting
- **linear\_interp\_n** - n dimensional linear interpolation
- **linspace** - linearly spaced vector
- **log** - natural logarithm
- **log10** - logarithm
- **log1p** - computes with accuracy the natural logarithm of its argument added by one
- **log2** - base 2 logarithm
- **logm** - square matrix logarithm
- **logspace** - logarithmically spaced vector
- **lsq\_splin** - weighted least squares cubic spline fitting
- **lstsize** - list, tlist, mlist numbers of entries
- **max** - maximum
- **maxi** - maximum
- **min** - minimum
- **mini** - minimum
- **minus** - (-) subtraction operator, sign changes
- **modulo** - symmetric arithmetic remainder modulo m
- **pmodulo** - positive arithmetic remainder modulo m
- **mps2linpro** - convert lp problem given in MPS format to linpro format
- **mtlb\_sparse** - convert sparse matrix
- **mulf** - symbolic multiplication
- **ndgrid** - arrays for multidimensional function evaluation on grid
- **ndims** - number of dimensions of an array
- **nearfloat** - get previous or next floating-point number
- **nextpow2** - next higher power of 2.
- **nnz** - number of non zero entries in a matrix
- **norm** - matrix norms
- **not** - (~) logical not
- **number\_properties** - determine floating-point parameters
- **ones** - matrix made of ones
- **or** - (|) logical or
- **pen2ea** - pencil to E,A conversion
- **pertrans** - pertranspose
- **prod** - product
- **rand** - random number generator
- **rat** - Floating point rational approximation
- **rdivf** - right symbolic division
- **real** - real part
- **round** - rounding
- **setdiff** - returns components of a vector which do not belong to another one
- **sign** - sign function
- **signm** - matrix sign function
- **sin** - sine function
- **sinc** - sinc function
- **sinh** - hyperbolic sine
- **sinhm** - matrix hyperbolic sine
- **sinm** - matrix sine function
- **size** - size of objects
- **smooth** - smoothing by spline functions
- **solve** - symbolic linear system solver
- **sort** - decreasing order sorting
- **sp2adj** - converts sparse matrix into adjacency form
- **sparse** - sparse matrix definition
- **spcompact** - converts a compressed adjacency representation

- `speye` - sparse identity matrix
- `spget` - retrieves entries of sparse matrix
- `splin` - cubic spline interpolation
- `splin2d` - bicubic spline gridded 2d interpolation
- `splin3d` - spline gridded 3d interpolation
- `spones` - sparse matrix
- `sprand` - sparse random matrix
- `spzeros` - sparse zero matrix
- `sqrt` - square root
- `sqrtn` - matrix square root
- `squarewave` - generates a square wave with period  $2^*$
- `ssprint` - pretty print for linear system
- `ssrand` - random system generator
- `sub2ind` - matrix subscript values to linear index
- `subf` - symbolic subtraction
- `sum` - sum (row sum, column sum) of vector/matrix entries
- `sysconv` - system conversion
- `sysdiag` - block diagonal system connection
- `syslin` - linear system definition
- `tan` - tangent
- `tanh` - hyperbolic tangent
- `tanhm` - matrix hyperbolic tangent
- `tanm` - matrix tangent
- `toeplitz` - toeplitz matrix
- `trfmod` - poles and zeros display
- `trianfml` - symbolic triangularization
- `tril` - lower triangular part of matrix
- `trisolve` - symbolic linear system solver
- `triu` - upper triangle
- `typeof` - object type
- `union` - extract union components of a vector
- `unique` - extract unique components of a vector
- `vectorfind` - finds in a matrix rows or columns matching a vector
- `zeros` - matrix made of zeros

## D.4 Input/Output Functions

- `diary` - diary of session
- `dir` - get file list
- `disp` - displays variables
- `dispfiles` - display opened files properties
- `file` - file management
- `fileinfo` - Provides information about a file
- `fileparts` - returns the path, filename and extension for a file path
- `fprintf` - Emulator of C language fprintf function
- `fprintfMat` - print a matrix in a file.
- `fscanf` - Converts formatted input read on a file
- `fscanfMat` - Reads a Matrix from a text file.
- `getio` - get Scilab input/output logical units
- `input` - prompt for user input
- `isdir` - checks if argument is a directory path
- `lines` - rows and columns used for display
- `load` - load saved variable
- `loadmatfile` - loads a Matlab MAT-file into Scilab
- `ls` - show files
- `manedit` - editing a manual item
- `matfile2sci` - converts a Matlab 5 MAT-file into a Scilab binary file
- `mclearerr` - reset binary file access errors
- `mclose` - close an opened file
- `mdelete` - Delete file(s)
- `meof` - check if end of file has been reached
- `merror` - tests the file access errors indicator
- `mfscanf` - interface to the C fscanf function
- `mscanf` - interface to the C scanf function
- `msscanf` - interface to the C sscanf function
- `mget` - reads byte or word in a given binary format and convert to double
- `mgeti` - reads byte or word in a given binary format return an int type
- `mgetl` - read lines from an ascii file
- `mgetstr` - read a character string
- `mopen` - open a file
- `mfprintf` - converts, formats, and writes data to a file
- `mprintf` - converts, formats, and writes data to the main scilab window
- `msprintf` - converts, formats, and writes data in a string
- `mput` - writes byte or word in a given binary format

- `mputl` - writes strings in an ascii file
- `mputstr` - write a character string in a file
- `mseek` - set current position in binary file.
- `mtell` - binary file management
- `newest` - returns newest file of a set of files
- `oldload` - load saved variable in 2.4.1 and previous formats
- `oldsave` - saving variables in 2.4.1 and previous format
- `print` - prints variables in a file
- `printf` - Emulator of C language printf function
- `printf_conversion` - printf, sprintf, fprintf conversion specifications
- `read` - matrices read
- `read4b` - fortran file binary read
- `readb` - fortran file binary read
- `readc_` - read a character string
- `readmps` - reads a file in MPS format
- `save` - saving variables in binary files
- `scanf` - Converts formatted input on standard input
- `scanf_conversion` - scanf, sscanf, fscanf conversion specifications
- `sprintf` - Emulator of C language sprintf function
- `sscanf` - Converts formatted input given by a string
- `startup` - startup file
- `tk_getdir` - dialog to get a directory path
- `tk_getfile` - dialog to get a file path
- `tk_savefile` - dialog to get a file path for writing
- `warning` - warning messages
- `writb` - fortran file binary write
- `write` - write in a formatted file
- `write4b` - fortran file binary write
- `xgetfile` - dialog to get a file path

## D.5 Handling of functions and libraries

- `addinter` - new functions interface incremental linking at run time
- `argn` - number of arguments in a function call
- `clearfun` - remove primitive.
- `comp` - scilab function compilation
- `deff` - on-line definition of function
- `delbpt` - delete breakpoint
- `dispbpt` - display breakpoints
- `edit` - function editing
- `funcprot` - switch scilab functions protection mode
- `endfunction` - closes a function definition
- `function` - opens a function definition
- `functions` - Scilab procedures and Scilab objects
- `genlib` - build library from all functions in given directory
- `get_function_path` - get source file path of a library function
- `getd` - getting all functions defined in a directory
- `getf` - defining a function from a file
- `lib` - library definition
- `library` - library datatype description
- `macr2lst` - function to list conversion
- `macr2tree` - function to tree conversion
- `macro` - Scilab procedure and Scilab object
- `macrovar` - variables of function
- `newfun` - add a name in the table of functions
- `plotprofile` - extracts and displays execution profiles of a Scilab function
- `profile` - extract execution profiles of a Scilab function
- `setbpt` - setting breakpoints
- `showprofile` - extracts and displays execution profiles of a Scilab function
- `varargin` - variable numbers of arguments in an input argument list
- `varargout` - variable numbers of arguments in an output argument list

## D.6 Character string manipulations

- **code2str** - returns character string associated with Scilab integer codes.
- **convstr** - case conversion
- **emptystr** - zero length string
- **grep** - find matches of a string in a vector of strings
- **justify** - Justify character array.
- **length** - length of object
- **part** - extraction of strings
- **str2code** - return scilab integer codes associated with a character string
- **strcat** - concatenate character strings
- **strindex** - search position of a character string in an other string.
- **string** - conversion to string
- **strings** - Scilab Object, character strings
- **stripblanks** - strips leading and trailing blanks of strings
- **strsubst** - substitute a character string by another in a character string.
- **tokenpos** - returns the tokens positions in a character string.
- **tokens** - returns the tokens of a character string.

## D.7 GUI and Dialogs

- **addmenu** - interactive button or menu definition
- **browsevar** - Scilab variable browser
- **buttondialog** - Create a simple button dialog
- **config** - Scilab general configuration.
- **delmenu** - interactive button or menu deletion
- **demoplay** - interactive demo player.
- **editvar** - Scilab variable editor
- **getvalue** - xwindow dialog for data acquisition
- **halt** - stop execution
- **havewindow** - return scilab window mode
- **keyboard** - keyboard commands
- **progressionbar** - Draw a progression bar
- **seteventhandler** - set an event handler for the current graphic window
- **setmenu** - interactive button or menu activation
- **unsetmenu** - interactive button or menu or submenu de-activation

- **waitbar** - Draw a waitbar
- **winclose** - close windows created by sciGUI
- **winlist** - Return the winId of current window created by sciGUI
- **x\_choices** - interactive Xwindow choices through toggle buttons
- **x\_choose** - interactive Xwindow choice
- **x\_dialog** - Xwindow dialog
- **x\_matrix** - Xwindow editing of matrix
- **x\_mdialog** - Xwindow dialog
- **x\_message** - X window message
- **x\_message\_modeless** - X window modeless message

## D.8 Utilities

- **G\_make** - call make or nmake
- **add\_demo** - Add an entry in the demos list
- **add\_help\_chapter** - Add an entry in the helps list
- **add\_palette** - Add an entry in the Scicos palettes list
- **apropos** - searches keywords in Scilab help
- **basename** - strip directory and suffix from filenames
- **c\_link** - check dynamic link
- **cd** - changes Scilab current directory
- **chdir** - changes Scilab current directory
- **c1c** - Clear Command Window
- **dec2hex** - hexadecimal representation of integers
- **dirname** - get directory from filenames
- **foo** - foo short description
- **head\_comments** - display scilab function header comments
- **help** - on-line help command
- **help\_skeleton** - build the skeleton of the xml help file associated to a Scilab function
- **hex2dec** - conversion from hexadecimal representation to integers
- **ilib\_build** - utility for shared library management
- **ilib\_compile** - ilib\_build utility: executes the makefile produced by ilib\_gen\_Make
- **ilib\_for\_link** - utility for shared library management with link
- **ilib\_gen\_Make** - utility for ilib\_build: produces a makefile for building shared libraries

- `ilib_gen_gateway` - utility for `ilib_build`, generates a gateway file.
- `ilib_gen_loader` - utility for `ilib_build`: generates a loader file
- `intersci` - scilab tool to interface C of Fortran functions with scilab
- `LANGUAGE` - Variable defining the language
- `link` - dynamic link
- `listfiles` - list files
- `make_index` - creates a new index file for on-line help
- `man` - on line help XML file description format
- `pathconvert` - pathnames conversion between posix and windows.
- `%helps` - Variable defining the path of help directories
- `realtime` - set dates origin or waits until date
- `realtimeinit` - set time unit
- `sci2exp` - converts an expression to a string
- `sci2map` - Scilab to Maple variable conversion
- `scilab` - Major unix script to execute Scilab and miscellaneous tools
- `scilink` - Unix script to relink Scilab
- `scipad` - Embedded Scilab text editor
- `timer` - cpu time
- `tohome` - Move the cursor to the upper left corner of the Command Window
- `unlink` - unlink a dynamically linked shared object
- `unix` - shell (sh) command execution
- `unix_g` - shell (sh) command execution, output redirected to a variable
- `unix_s` - shell (sh) command execution, no output
- `unix_w` - shell (sh) command execution, output redirected to scilab window
- `unix_x` - shell (sh) command execution, output redirected to a window
- `with_gtk` - Checks if Scilab has been built with the "GIMP Toolkit" library
- `with_pvm` - Checks if Scilab has been built with the "Parallel Virtual Machine" interface
- `with_texmacs` - Checks if Scilab has been called by texmacs
- `with_tk` - Checks if Scilab has been built with TCL/TK
- `xmltohtml` - converts xml Scilab help files to html

## D.9 Linear Algebra

- `aff2ab` - linear (affine) function to A,b conversion
- `balanc` - matrix or pencil balancing
- `bdiag` - block diagonalization, generalized eigenvectors
- `chfact` - sparse Cholesky factorization
- `chol` - Cholesky factorization
- `chsolve` - sparse Cholesky solver
- `classmarkov` - recurrent and transient classes of Markov matrix
- `coff` - resolvent (cofactor method)
- `colcomp` - column compression, kernel, nullspace
- `companion` - companion matrix
- `cond` - condition number
- `det` - determinant
- `eigenmarkov` - normalized left and right Markov eigenvectors
- `ereduc` - computes matrix column echelon form by qz transformations
- `exp` - element-wise exponential
- `expm` - square matrix exponential
- `fstair` - computes pencil column echelon form by qz transformations
- `fullrf` - full rank factorization
- `fullrfk` - full rank factorization of  $A^k$
- `genmarkov` - generates random markov matrix with recurrent and transient classes
- `givens` - Givens transformation
- `glever` - inverse of matrix pencil
- `gschur` - generalized Schur form (obsolete).
- `gspec` - eigenvalues of matrix pencil (obsolete)
- `hess` - Hessenberg form
- `householder` - Householder orthogonal reflexion matrix
- `im_inv` - inverse image
- `inv` - matrix inverse
- `kernel` - kernel, nullspace
- `kroneck` - Kronecker form of matrix pencil
- `linsolve` - linear equation solver
- `lsq` - linear least square problems.
- `lu` - LU factors of Gaussian elimination
- `ludel` - utility function used with `lufact`
- `lufact` - sparse lu factorization

- **luget** - extraction of sparse LU factors
- **lusolve** - sparse linear system solver
- **lyap** - Lyapunov equation
- **nlev** - Leverrier's algorithm
- **orth** - orthogonal basis
- **pbig** - eigen-projection
- **pencan** - canonical form of matrix pencil
- **penlaur** - Laurent coefficients of matrix pencil
- **pinv** - pseudoinverse
- **polar** - polar form
- **proj** - projection
- **projspec** - spectral operators
- **psmall** - spectral projection
- **qr** - QR decomposition
- **quaskro** - quasi-Kronecker form
- **randpencil** - random pencil
- **range** - range (span) of  $A^k$
- **rank** - rank
- **rankqr** - rank revealing QR factorization
- **rcond** - inverse condition number
- **rowcomp** - row compression, range
- **rowshuff** - shuffle algorithm
- **rref** - computes matrix row echelon form by lu transformations
- **schur** - [ordered] Schur decomposition of matrix and pencils
- **spaninter** - subspace intersection
- **spanplus** - sum of subspaces
- **spantwo** - sum and intersection of subspaces
- **spchol** - sparse cholesky factorization
- **spec** - eigenvalues of matrices and pencils
- **sqroot** -  $W^*W'$  hermitian factorization
- **sva** - singular value approximation
- **svd** - singular value decomposition
- **sylv** - Sylvester equation.
- **trace** - trace

## D.10 Polynomial calculations

- **bezout** - Bezout equation for polynomials or integers
- **clean** - cleans matrices (round to zero small entries)
- **cmdred** - common denominator form
- **coeff** - coefficients of matrix polynomial
- **coffg** - inverse of polynomial matrix
- **colcompr** - column compression of polynomial matrix
- **degree** - degree of polynomial matrix
- **denom** - denominator
- **derivat** - rational matrix derivative
- **determ** - determinant of polynomial matrix
- **detr** - polynomial determinant
- **diophant** - diophantine (Bezout) equation
- **factors** - numeric real factorization
- **gcd** - gcd calculation
- **hermit** - Hermite form
- **horner** - polynomial/rational evaluation
- **hrmt** - gcd of polynomials
- **htrianr** - triangularization of polynomial matrix
- **invr** - inversion of (rational) matrix
- **lcm** - least common multiple
- **lcmdiag** - least common multiple diagonal factorization
- **ldiv** - polynomial matrix long division
- **numer** - numerator
- **pdiv** - polynomial division
- **pol2des** - polynomial matrix to descriptor form
- **pol2str** - polynomial to string conversion
- **polfact** - minimal factors
- **residu** - residue
- **roots** - roots of polynomials
- **routh\_t** - Routh's table
- **rowcompr** - row compression of polynomial matrix
- **sfact** - discrete time spectral factorization
- **simp** - rational simplification
- **simp\_mode** - toggle rational simplification
- **sylv** - Sylvester matrix
- **systemat** - system matrix

## D.11 General System and Control

- **abcd** - state-space matrices
- **abinv** - AB invariant subspace
- **arhnk** - Hankel norm approximant
- **ar12** - SISO model realization by L2 transfer approximation
- **balreal** - balanced realization
- **bilin** - general bilinear transform
- **cainv** - Dual of abinv
- **calfrq** - frequency response discretization
- **canon** - canonical controllable form
- **cls2dls** - bilinear transform
- **colregul** - removing poles and zeros at infinity
- **cont\_frm** - transfer to controllable state-space
- **cont\_mat** - controllability matrix
- **contr** - controllability, controllable subspace, staircase
- **contrss** - controllable part
- **csim** - simulation (time response) of linear system
- **ctr\_gram** - controllability gramian
- **dbphi** - frequency response to phase and magnitude representation
- **ddp** - disturbance decoupling
- **des2tf** - descriptor to transfer function conversion
- **dscr** - discretization of linear system
- **dsimul** - state space discrete time simulation
- **dt\_ility** - detectability test
- **equil** - balancing of pair of symmetric matrices
- **equil1** - balancing (nonnegative) pair of matrices
- **feedback** - feedback operation
- **flts** - time response (discrete time, sampled system)
- **frep2tf** - transfer function realization from frequency response
- **freq** - frequency response
- **freson** - peak frequencies
- **g\_margin** - gain margin
- **gfrancis** - Francis equations for tracking
- **imrep2ss** - state-space realization of an impulse response
- **invsyslin** - system inversion
- **kpure** - continuous SISO system limit feedback gain
- **krac2** - continuous SISO system limit feedback gain
- **lin** - linearization
- **linmeq** - Sylvester and Lyapunov equations solver
- **lqe** - linear quadratic estimator (Kalman Filter)
- **lqg** - LQG compensator
- **lqg2stan** - LQG to standard problem
- **lqr** - LQ compensator (full state)
- **ltitr** - discrete time response (state space)
- **markp2ss** - Markov parameters to state-space
- **minreal** - minimal balanced realization
- **minss** - minimal realization
- **obs\_gram** - observability gramian
- **obscont** - observer based controller
- **observer** - observer design
- **obsv\_mat** - observability matrix
- **obsvss** - observable part
- **p\_margin** - phase margin
- **pfss** - partial fraction decomposition
- **phasemag** - phase and magnitude computation
- **ppol** - pole placement
- **projsl** - linear system projection
- **repfreq** - frequency response
- **ricc** - Riccati equation
- **riccsl** - Riccati equation solver
- **rowregul** - removing poles and zeros at infinity
- **rtitr** - discrete time response (transfer matrix)
- **sm2des** - system matrix to descriptor
- **sm2ss** - system matrix to state-space
- **specfact** - spectral factor
- **ss2des** - (polynomial) state-space to descriptor form
- **ss2ss** - state-space to state-space conversion, feedback, injection
- **ss2tf** - conversion from state-space to transfer function
- **st\_ility** - stabilizability test
- **stabil** - stabilization
- **svplot** - singular-value sigma-plot
- **sysfact** - system factorization



- `syssize` - size of state-space system
- `tf2ss` - transfer to state-space
- `time_id` - SISO least square identification
- `trzeros` - transmission zeros and normal rank
- `ui_observer` - unknown input observer
- `unobs` - unobservable subspace
- `zeropen` - zero pencil

## D.12 Robust control toolbox

- `augment` - augmented plant
- `bstap` - hankel approximant
- `ccontrg` - central H-infinity controller
- `colinout` - inner-outer factorization
- `copfac` - right coprime factorization
- `dcf` - double coprime factorization
- `des2ss` - descriptor to state-space
- `dhinf` - H\_infinity design of discrete-time systems
- `dhnorm` - discrete H-infinity norm
- `dtsi` - stable anti-stable decomposition
- `fourplan` - augmented plant to four plants
- `fspecg` - stable factorization
- `fstabst` - Youla's parametrization
- `gamitg` - H-infinity gamma iterations
- `gcare` - control Riccati equation
- `gfare` - filter Riccati equation
- `gtild` - tilde operation
- `h2norm` - H2 norm
- `h_cl` - closed loop matrix
- `h_inf` - H-infinity (central) controller
- `h_inf_st` - static H\_infinity problem
- `h_norm` - H-infinity norm
- `hankelsv` - Hankel singular values
- `hinf` - H\_infinity design of continuous-time systems
- `lcf` - normalized coprime factorization
- `leqr` - H-infinity LQ gain (full state)
- `lft` - linear fractional transformation
- `linf` - infinity norm
- `linfn` - infinity norm
- `lqg_ltr` - LQG with loop transform recovery

- `macglov` - Mac Farlane Glover problem
- `mucomp` - mu (structured singular value) calculation
- `nehari` - Nehari approximant
- `parrot` - Parrot's problem
- `ric_desc` - Riccati equation
- `riccati` - Riccati equation
- `rowinout` - inner-outer factorization
- `sensi` - sensitivity functions
- `tf2des` - transfer function to descriptor

## D.13 Optimization and simulation

- `NDcost` - generic external for optim computing gradient using finite differences
- `bvode` - boundary value problems for ODE
- `dasrt` - DAE solver with zero crossing
- `dassl` - differential algebraic equation
- `datafit` - Parameter identification based on measured data
- `derivative` - approximate derivatives of a function
- `fit_dat` - Parameter identification based on measured data
- `fsolve` - find a zero of a system of n nonlinear functions
- `impl` - differential algebraic equation
- `int2d` - definite 2D integral by quadrature and cubature method
- `int3d` - definite 3D integral by quadrature and cubature method
- `intc` - Cauchy integral
- `intg` - definite integral
- `intl` - Cauchy integral
- `karmarkar` - karmarkar algorithm
- `leastsq` - Solves non-linear least squares problems
- `linpro` - linear programming solver
- `lmisolver` - linear matrix inequation solver
- `lmitool` - tool for solving linear matrix inequations
- `lsqrsolve` - minimize the sum of the squares of nonlinear functions, levenberg-marquardt algorithm
- `numdiff` - numerical gradient estimation
- `ode` - ordinary differential equation solver
- `ode_discrete` - ordinary differential equation solver, discrete time simulation

- `ode_root` - ordinary differential equation solver with root finding
- `odedc` - discrete/continuous ode solver
- `odeoptions` - set options for ode solvers
- `optim` - non-linear optimization routine
- `quapro` - linear quadratic programming solver
- `semidef` - semidefinite programming

## D.14 Signal Processing toolbox

- `Signal` - Signal manual description
- `analpf` - create analog low-pass filter
- `buttmag` - response of Butterworth filter
- `casc` - cascade realization of filter from coefficients
- `cepstrum` - cepstrum calculation
- `cheb1mag` - response of Chebyshev type 1 filter
- `cheb2mag` - response of type 2 Chebyshev filter
- `chepol` - Chebychev polynomial
- `convol` - convolution
- `corr` - correlation, covariance
- `cspect` - spectral estimation (correlation method)
- `czt` - chirp z-transform algorithm
- `dft` - discrete Fourier transform
- `ell1mag` - magnitude of elliptic filter
- `eqfir` - minimax approximation of FIR filter
- `eqiir` - Design of iir filters
- `faurre` - filter computation by simple Faurre algorithm
- `ffilt` - coefficients of FIR low-pass
- `fft` - fast Fourier transform.
- `fftshift` - rearranges the fft output, moving the zero frequency to the center of the spectrum
- `filter` - modelling filter
- `find_freq` - parameter compatibility for elliptic filter design
- `findm` - for elliptic filter design
- `frfit` - frequency response fit
- `frmag` - magnitude of FIR and IIR filters
- `fsfirlin` - design of FIR, linear phase filters, frequency sampling technique
- `group` - group delay for digital filter
- `hank` - covariance to hankel matrix
- `hilb` - Hilbert transform
- `iir` - iir digital filter
- `iirgroup` - group delay Lp IIR filter optimization
- `iirlp` - Lp IIR filter optimization
- `intdec` - Changes sampling rate of a signal
- `jmat` - row or column block permutation
- `kalm` - Kalman update
- `lattn` - recursive solution of normal equations
- `lattp` - lattp
- `lev` - Yule-Walker equations (Levinson's algorithm)
- `levin` - Toeplitz system solver by Levinson algorithm (multidimensional)
- `lgfft` - utility for fft
- `lindquist` - Lindquist's algorithm
- `mese` - maximum entropy spectral estimation
- `mfft` - multi-dimensional fft
- `mrfit` - frequency response fit
- `%asn` - elliptic integral
- `%k` - Jacobi's complete elliptic integral
- `%sn` - Jacobi 's elliptic function
- `phc` - Markovian representation
- `pspect` - cross-spectral estimate between 2 series
- `remez` - Remez's algorithm
- `remezb` - Minimax approximation of magnitude response
- `rpem` - RPEM estimation
- `sinc` - samples of sinc function
- `sincd` - digital sinc function or Dirichlet kernel
- `srfaur` - square-root algorithm
- `srkf` - square root Kalman filter
- `sskf` - steady-state Kalman filter
- `system` - observation update
- `trans` - low-pass to other filter transform
- `wfir` - linear-phase FIR filters
- `wiener` - Wiener estimate
- `wigner` - 'time-frequency' wigner spectrum
- `window` - symmetric window
- `yulewalk` - least-square filter design
- `zbutt` - Butterworth analog filter
- `zpch1` - Chebyshev analog filter
- `zpch2` - Chebyshev analog filter
- `zpell` - lowpass elliptic filter

## D.15 Arma modelisation and simulation toolbox

- **arma** - Scilab arma library
- **arma2p** - extract polynomial matrices from ar representation
- **armac** - Scilab description of an armax process
- **armax** - armax identification
- **armax1** - armax identification
- **arsimul** - armax simulation
- **narsimul** - armax simulation ( using rtitr)
- **noisegen** - noise generation
- **odedi** - test of ode
- **prbs\_a** - pseudo random binary sequences generation
- **reglin** - Linear regression

## D.16 Metanet: graph and network toolbox

- **add\_edge** - adds an edge or an arc between two nodes
- **add\_node** - adds a disconnected node to a graph
- **adj\_lists** - computes adjacency lists
- **arc\_graph** - graph with nodes corresponding to arcs
- **arc\_number** - number of arcs of a graph
- **articul** - finds one or more articulation points
- **bandwr** - bandwidth reduction for a sparse matrix
- **best\_match** - best matching of a graph
- **chain\_struct** - chained structure from adjacency lists of a graph
- **check\_graph** - checks a Scilab graph list
- **circuit** - finds a circuit or the rank function in a directed graph
- **con\_nodes** - set of nodes of a connected component
- **connex** - connected components
- **contract\_edge** - contracts edges between two nodes
- **convex\_hull** - convex hull of a set of points in the plane
- **cycle\_basis** - basis of cycle of a simple undirected graph
- **delete\_arcs** - deletes all the arcs or edges between a set of nodes
- **delete\_nodes** - deletes nodes
- **edge\_number** - number of edges of a graph
- **edit\_graph** - graph and network graphical editor

- **edit\_graph\_menus** - edit\_graph menus description
- **find\_path** - finds a path between two nodes
- **gen\_net** - interactive or random generation of a network
- **girth** - girth of a directed graph
- **glist** - graph list creation
- **graph-list** - description of graph list
- **graph\_2\_mat** - node-arc or node-node incidence matrix of a graph
- **graph\_center** - center of a graph
- **graph\_complement** - complement of a graph
- **graph\_diameter** - diameter of a graph
- **graph\_power** - kth power of a directed 1-graph
- **graph\_simp** - converts a graph to a simple undirected graph
- **graph\_sum** - sum of two graphs
- **graph\_union** - union of two graphs
- **hamilton** - hamiltonian circuit of a graph
- **is\_connex** - connectivity test
- **knapsack** - solves a 0-1 multiple knapsack problem
- **line\_graph** - graph with nodes corresponding to edges
- **load\_graph** - loads a graph
- **make\_graph** - makes a graph list
- **mat\_2\_graph** - graph from node-arc or node-node incidence matrix
- **max\_cap\_path** - maximum capacity path
- **max\_clique** - maximum clique of a graph
- **max\_flow** - maximum flow between two nodes
- **mesh2d** - triangulation of n points in the plane
- **min\_lcost\_cflow** - minimum linear cost constrained flow
- **min\_lcost\_flow1** - minimum linear cost flow
- **min\_lcost\_flow2** - minimum linear cost flow
- **min\_qcost\_flow** - minimum quadratic cost flow
- **min\_weight\_tree** - minimum weight spanning tree
- **neighbors** - nodes connected to a node
- **netclose** - closes an edit\_graph window
- **netwindow** - selects the current edit\_graph window
- **netwindows** - gets the numbers of edit\_graph windows
- **node\_number** - number of nodes of a graph

- `nodes_2_path` - path from a set of nodes
- `nodes_degrees` - degrees of the nodes of a graph
- `path_2_nodes` - set of nodes from a path
- `perfect_match` - min-cost perfect matching
- `pipe_network` - solves the pipe network problem
- `plot_graph` - general plot of a graph
- `predecessors` - tail nodes of incoming arcs of a node
- `qassign` - solves a quadratic assignment problem
- `salesman` - solves the travelling salesman problem
- `save_graph` - saves a graph
- `shortest_path` - shortest path
- `show_arcs` - highlights a set of arcs
- `show_graph` - displays a graph
- `show_nodes` - highlights a set of nodes
- `split_edge` - splits an edge by inserting a node
- `strong_con_nodes` - set of nodes of a strong connected component
- `strong_connex` - strong connected components
- `subgraph` - subgraph of a graph
- `successors` - head nodes of outgoing arcs of a node
- `supernode` - replaces a group of nodes with a single node
- `trans_closure` - transitive closure

## D.17 Sound file handling

- `analyze` - frequency plot of a sound signal
- `auread` - load .au sound file
- `auwrite` - writes .au sound file
- `lin2mu` - linear signal to mu-law encoding
- `loadwave` - load a sound <<wav>> file into scilab
- `mapsound` - Plots a sound map
- `mu2lin` - mu-law encoding to linear signal
- `playsnd` - sound player facility
- `savewave` - save data into a sound <<wav>> file.
- `sound` - sound player facility
- `wavread` - load .wav sound file
- `wavwrite` - writes .wav sound file

## D.18 Language or data translations

- `ascii` - string ascii conversions
- `excel2sci` - reads ascii Excel files
- `formatman` - translates old NROFF man files in a directory into ascii, tex, html or xml
- `fun2string` - generates ascii definition of a scilab function
- `mfile2sci` - Matlab M-file to Scilab conversion function
- `mtlb_load` - load variables from file with Matlab4 format.
- `mtlb_save` - save variables on file with matlab4 format.
- `pol2tex` - convert polynomial to TeX format
- `sci2for` - scilab function to Fortran routine conversion
- `texprint` - TeX output of Scilab object
- `translatepaths` - convert a set of Matlab M-files directories to Scilab
- `tree2code` - generates ascii definition of a Scilab function

## D.19 PVM parallel toolbox

- `AdCommunications` - advanced communication toolbox for parallel programming
- `Example` - just to test the environment
- `pvm` - communications with other applications using Parallel Virtual Machine
- `pvm_addhosts` - add hosts to the virtual machine.
- `pvm_barrier` - blocks the calling process until all processes in a group have called it.
- `pvm_bcast` - broadcasts a message to all members of a group
- `pvm_bufinfo` - Returns information about a message buffer.
- `pvm_config` - sends a message
- `pvm_delhosts` - deletes hosts from the virtual machine.
- `pvm_error` - Prints message describing an error returned by a PVM call
- `pvm_exit` - tells the local pvmd that this process is leaving PVM.
- `pvm_f772sci` - Convert a F77 complex into a complex scalar
- `pvm_get_timer` - Gets the system's notion of the current time.
- `pvm_getinst` - returns the instance number in a group of a PVM process.
- `pvm_gettid` - returns the tid of the process identified by a group name and instance number

- **pvm\_gsize** - returns the number of members presently in the named group.
- **pvm\_halt** - stops the PVM daemon
- **pvm\_joiningroup** - enrolls the calling process in a named group.
- **pvm\_kill** - Terminates a specified PVM process.
- **pvm\_lvgroup** - Unenrolls the calling process from a named group.
- **pvm\_mytid** - returns the tid of the calling process.
- **pvm\_parent** - tid of the process that spawned the calling process.
- **pvm\_probe** - Check if message has arrived.
- **pvm\_recv** - receive a message.
- **pvm\_reduce** - Performs a reduce operation over members of the specified group
- **pvm\_sci2f77** - Convert complex scalar into F77
- **pvm\_send** - immediately sends (or multicast) data.
- **pvm\_set\_timer** - Sets the system's notion of the current time.
- **pvm\_spawn** - Starts new Scilab processes.
- **pvm\_spawn\_independent** - Starts new PVM processes.
- **pvm\_start** - Start the PVM daemon
- **pvm\_tasks** - information about the tasks running on the virtual machine.
- **pvm\_tidtohost** - returns the host of the specified PVM process.
- **pvm3** - PVM daemon

## D.20 TdCs

- **artest** - arnold dynamical system
- **bifish** - shows a bifurcation diagram in a fish population discrete time model
- **boucle** - phase portrait of a dynamical system with observer
- **chaintest** - a three-species food chain model
- **gpeche** - a fishing program
- **fusee** - a set of Scilab macro for a landing rocket problem
- **lotest** - demo of the Lorenz attractor
- **mine** - a mining problem
- **obscont1** - a controlled-observed system
- **portr3d** - 3 dimensional phase portrait.
- **portrait** - 2 dimensional phase portrait.
- **recur** - a bilinear recurrent equation

- **systems** - a collection of dynamical system
- **tangent** - linearization of a dynamical system at an equilibrium point
- **tdinit** - interactive initialisation of the tdc's dynamical systems

## D.21 TCL/Tk interface

- **ScilabEval** - tcl instruction : Evaluate a string with scilab interpreter
- **TK\_EvalFile** - Reads and evaluate a tcl/tk file
- **TK\_EvalStr** - Evaluate a string within the tcl/tk interpreter
- **TK\_GetVar** - Get a tcl/tk variable value
- **TK\_SetVar** - Set a tcl/tk variable value
- **close** - close a figure
- **figure** - create a figure
- **findobj** - find an object with specified property
- **uicontrol** - create a Graphic User Interface object
- **uimenu** - Create a menu or a submenu in a figure

## D.22 Statistic basics

- **center** - center
- **wcenter** - center and weight
- **cmoment** - central moments of all orders
- **correl** - correlation of two variables
- **covar** - covariance of two variables
- **ftest** - Fischer ratio
- **ftuneq** - Fischer ratio for samples of unequal size.
- **geomean** - geometric mean
- **harmean** - harmonic mean
- **iqr** - interquartile range
- **labostat** - Statistical toolbox for Scilab
- **mad** - mean absolute deviation
- **mean** - mean (row mean, column mean) of vector/matrix entries
- **meanf** - weighted mean of a vector or a matrix
- **median** - median (row median, column median) of vector/matrix entries
- **moment** - non central moments of all orders
- **msd** - mean squared deviation
- **mvvacov** - computes variance-covariance matrix

- **nancumsum** - This function returns the cumulative sum of the values of a matrix
- **nand2mean** - difference of the means of two independent samples
- **nanmax** - max (ignoring Nan's)
- **nanmean** - mean (ignoring Nan's)
- **nanmeanf** - mean (ignoring Nan's) with a given frequency.
- **nanmedian** - median of the values of a numerical vector or matrix
- **nanmin** - min (ignoring Nan's)
- **nanstdev** - standard deviation (ignoring the NANs).
- **nansum** - Sum of values ignoring NAN's
- **nfreq** - frequency of the values in a vector or matrix
- **pca** - Principal components analysis
- **perctl** - computation of percentils
- **quart** - computation of quartiles
- **regress** - regression coefficients of two variables
- **sample** - Sampling with replacement
- **samplef** - sample with replacement from a population and frequencies of his values.
- **samwr** - Sampling without replacement
- **st\_deviation** - standard deviation (row or column-wise) of vector/matrix entries
- **stdev** - standard deviation (row or column-wise) of vector/matrix entries
- **stdevf** - standard deviation
- **strange** - range
- **tabul** - frequency of values of a matrix or vector
- **thrownan** - eliminates nan values
- **trimmean** - trimmed mean of a vector or a matrix
- **variance** - variance of the values of a vector or matrix
- **variancef** - standard deviation of the values of a vector or matrix
- **cdfchn** - cumulative distribution function non-central chi-square distribution
- **cdfF** - cumulative distribution function F distribution
- **cdfnc** - cumulative distribution function non-central f-distribution
- **cdfgam** - cumulative distribution function gamma distribution
- **cdfnbn** - cumulative distribution function negative binomial distribution
- **cdfnor** - cumulative distribution function normal distribution
- **cdfpoi** - cumulative distribution function poisson distribution
- **cdft** - cumulative distribution function Student's T distribution
- **grand** - Random number generator(s)

## D.23 Cumulative Distribution Functions; Inverses, grand

- **cdfbet** - cumulative distribution function Beta distribution
- **cdfbin** - cumulative distribution function Binomial distribution
- **cdfchi** - cumulative distribution function chi-square distribution

## D.24 Identification

- **findABCD** - discrete-time system subspace identification
- **findAC** - discrete-time system subspace identification
- **findBD** - initial state and system matrices B and D of a discrete-time system
- **findBDK** - Kalman gain and B D system matrices of a discrete-time system
- **findR** - Preprocessor for estimating the matrices of a linear time-invariant dynamical system
- **findxOBD** - Estimates state and B and D matrices of a discrete-time linear system
- **inistate** - Estimates the initial state of a discrete-time system
- **sident** - discrete-time state-space realization and Kalman gain
- **sorder** - computing the order of a discrete-time system

## D.25 Matlab to Scilab conversion tips

- `About_M2SCI_tools` - Generally speaking about tools to convert Matlab files to Scilab...
- `Cste` - Create a tree representing a constant
- `Equal` - Create a tree representing an instruction
- `Funcall` - Create a tree representing a function call
- `Infer` - Create a tree containing inference data
- `Matlab-Scilab_character_strings` - Generally speaking about...
- `Operation` - Create a tree representing an operation
- `Type` - Create a tree containing type inference data
- `Variable` - Create a tree representing a variable
- `asciimat` - string matrix to ASCII conversion
- `firstnonsingleton` - Finds first dimension which is not 1
- `m2scideclare` - Giving tips to help M2SCI...
- `mstr2sci` - character string matrix to character matrix conversion
- `mtlb_0` - Matlab non-conjugate transposition emulation function
- `mtlb_a` - Matlab addition emulation function
- `mtlb_all` - Matlab all emulation function
- `mtlb_any` - Matlab any emulation function
- `mtlb_beta` - Matlab beta emulation function
- `mtlb_box` - Matlab box emulation function
- `mtlb_close` - Matlab close emulation function
- `mtlb_colordef` - Matlab colordef emulation function
- `mtlb_conv` - Matlab conv emulation function
- `mtlb_cumprod` - Matlab cumprod emulation function
- `mtlb_cumsum` - Matlab cumsum emulation function
- `mtlb_dec2hex` - Matlab dec2hex emulation function
- `mtlb_delete` - Matlab delete emulation function
- `mtlb_diag` - Matlab diag emulation function
- `mtlb_diff` - Matlab diff emulation function
- `mtlb_dir` - Matlab dir emulation function
- `mtlb_double` - Matlab double emulation function
- `mtlb_e` - Matlab extraction emulation function
- `mtlb_eig` - Matlab eig emulation function
- `mtlb_eval` - Matlab eval emulation function
- `mtlb_exist` - Matlab exist emulation function
- `mtlb_eye` - Matlab eye emulation function
- `mtlb_false` - Matlab false emulation function
- `mtlb_fft` - Matlab fft emulation function
- `mtlb_find` - Matlab find emulation function
- `mtlb_full` - Matlab full emulation function
- `mtlb_hold` - Matlab hold emulation function
- `mtlb_i` - Matlab insertion emulation function
- `mtlb_imp` - Matlab colon emulation function
- `mtlb_int16` - Matlab int16 emulation function
- `mtlb_int32` - Matlab int32 emulation function
- `mtlb_int8` - Matlab int8 emulation function
- `mtlb_is` - Matlab string insertion emulation function
- `mtlb_isa` - Matlab isa emulation function
- `mtlb_isspace` - Matlab isspace emulation function
- `mtlb_l` - Matlab left division emulation function
- `mtlb_logic` - Matlab logical operators emulation function
- `mtlb_logical` - Matlab logical emulation function
- `mtlb_lower` - Matlab lower emulation function
- `mtlb_max` - Matlab max emulation function
- `mtlb_min` - Matlab min emulation function
- `mtlb_more` - Matlab more emulation function
- `mtlb_norm` - Matlab norm emulation function
- `mtlb_num2str` - Matlab num2str emulation function
- `mtlb_ones` - Matlab ones emulation function
- `mtlb_plot` - Matlab plot emulation function
- `mtlb_prod` - Matlab prod emulation function
- `mtlb_rand` - Matlab rand emulation function
- `mtlb_randn` - Matlab randn emulation function
- `mtlb_rcond` - Matlab rcond emulation function
- `mtlb_s` - Matlab subtraction emulation function
- `mtlb_setstr` - Matlab setstr emulation function
- `mtlb_size` - Matlab size emulation function
- `mtlb_strcmp` - Matlab strcmp emulation function
- `mtlb_strcmppi` - Matlab strcmppi emulation function
- `mtlb_strfind` - Matlab strfind emulation function
- `mtlb_strep` - Matlab strep emulation function
- `mtlb_sum` - Matlab sum emulation function
- `mtlb_t` - Matlab transposition emulation function

- `mtlb_toeplitz` - Matlab toeplitz emulation function
- `mtlb_tril` - Matlab tril emulation function
- `mtlb_triu` - Matlab triu emulation function
- `mtlb_true` - Matlab true emulation function
- `mtlb_uint16` - Matlab uint16 emulation function
- `mtlb_uint32` - Matlab uint32 emulation function
- `mtlb_uint8` - Matlab uint8 emulation function
- `mtlb_upper` - Matlab upper emulation function
- `mtlb_zeros` - Matlab zeros emulation function
- `sci_files` - How to write conversion functions



# Referências Bibliográficas

- [1] Scilab Group, *Introduction to Scilab - User's Guide*. Esta referência, e outras escritas pelo Scilab Group, podem ser obtidas em <http://scilabsoft.inria.fr/doc.html>, acessada em julho de 2004.
- [2] Jesus Olivan Palacios, *An Introduction to the Treatment of Neurophysiological Signals using Scilab - version 0.02*, Maio de 2001, <http://www.neurotraces.com/scilab/scilab2/node2.html>, acessada em julho de 2004.
- [3] Bruno Piçon, *Une introduction à Scilab, version 0.996*, disponível em <http://www-rocq.inria.fr/scilab/books>, acessada em maio de 2004.
- [4] L.E. van Dijk, C.L. Spiel, *Scilab Bag of Tricks, The Scilab-2.5 IAQ (Infrequently Asked Questions)*, disponível em <http://kiwi.emse.fr/SCILAB/sci-bot/sci-bot.pdf>, acessada em julho de 2004.
- [5] Gilberto E. Urroz, *Scilab*, disponível em <http://www.engineering.usu.edu/cee/faculty/gurro/Scilab.html>, acessada em junho de 2004.
- [6] Pramode C.E, *Mathematical Explorations with Scilab/Linux*, Linux Gazette, Issue 98, Janeiro de 2004, disponível em <http://linuxgazette.net/issue98/pramode.html>, acessada em julho de 2004.
- [7] Paulo S. Motta Pires, David A. Rogers, *Free/Open Source Software: An Alternative for Engineering Students*, 32nd ASEE/IEEE Frontiers in Education Conference, Nov. 6-9, 2002, Boston, MA, USA, disponível em <http://fie.engrng.pitt.edu/fie2002/papers/1355.pdf>. Uma cópia deste trabalho também está disponível em <http://www.dca.ufrn.br/~pmotta>, acessada em julho de 2004.
- [8] Scilab//, <http://www.ens-lyon.fr/~desprez/FILES/RESEARCH/SOFT/SCILAB>, acessada em julho de 2004.
- [9] Paulo S. Motta Pires, *Métodos Computacionais - Notas de Aula*, disponível em <http://www.dca.ufrn.br/~pmotta>, acessada em julho de 2004.