Utilizando o **SCILAB**

na Resolução de Problemas da **Engenharia Química**

Versão: 0.1

Luís Cláudio Oliveira Lopes

Curitiba - Paraná - Brasil

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

SCILAB (**Sci**entific **Lab**oratory)[1] é um ambiente gráfico para cálculo científico disponível gratuitamente[2] desde 1994 e desenvolvido desde 1990 por pesquisadores do "Institut Nationale de Recherche en Informatique et en Automatique - (INRIA)" e "Ecole Nationale des Ponts et Chaussée" (ENPC) na França[3]. O Scilab foi desenvolvido para ser um sistema *aberto* onde o usuário pode definir novos tipos de dados e operações; possui centenas de funções matemáticas com a possibilidade de interação com programs em várias linguagens com o C e Fortran; tem uma sofisticada estrutura de dados que inclui objetos como funções racionais, polinômios, listas, sistemas lineares, etc., possui um interpretador e uma linguagem de programação (estruturada) própria.

A utilização do Scilab dá-se internacionalmente nos ambientes acadêmicos e industriais, assim o Scilab é uma plataforma em constante atualização e aperfeiçoamento. Ele possui várias bibliotecas de funções, destacando-se:

- Biblioteca Gráfica 2-D e 3-D e Animação
- Álgebra Linear
- Polinômios e Funções Racionais
- Integração: Equações Diferencias Ordinárias (ODEPACK) e Equações Algébrico-Diferenciais (DASSL)
- Modelagem e Simulação (Scicos)
- Controle Clássico e Robusto
- Otimização (Inequações Matriciais Lineares -LMI, Otimização Diferenciável e Não Diferenciável
- Processamento de Sinais
- Processamento de Imagens
- Grafos e Redes (Metanet)
- Scilab para Arquitetura Paralela
- Estatística
- Rede Neuronal
- Lógica Nebulosa (*Fuzzy Logic*)
- Controle Ótimo Discreto
- Interfaces com *Softwares* de Computação Simbólica (Maple®, MuPAD)

[1] Pronúncia em sintaxe fonética internacional é "sailæb".

[2] Veja detalhes da licença no Apêndice I.

[3] Desde 16 de maio 2003, com um time de especialistas dedicados pertencentes a um consórcio de instituições e empresas que será responsável pelo desenvolvimento, evolução e promoção do Scilab. Informações adicionais estão disponíveis em http://www-rocq.inria.fr/scilab/.

- Interface com Tck/Tk
- E muitas outras contribuições.

Existem distribuições Scilab com código fonte disponível para a maioria das plataformas computacionais. O Scilab possui recursos similares àqueles existentes no MATLAB® e outros ambientes para cálculo científico. Esse texto se refere a versão 3.0 do Scilab. A base eletrônica do projeto Scilab encontra-se em http://www.scilab.org, nesse endereço pode-se encontrar atualizações relativas ao Scilab, informações, documentação e um conjunto de endereços relativos à utilização do Scilab em várias áreas de conhecimento. Esse documento não tem a ambição de ser um documento completo sobre a utilização do Scilab na resolução de problemas da Engenharia Química, seu objetivo principal é apresentar o potencial dessa plataforma como alternativa ao Matlab® nas avaliações numéricas usuais nas atividades relacionadas à Engenharia Química.

Essa apostila é a primeira parte de material mais amplo que tenho em desenvolvimento sobre o Scilab. Existem outros materiais disponíveis sobre o assunto e podem ser encontrados no site do Scilab. O *Introduction to Scilab: User´s Guide* (Scilab.Group, 1998) é uma fonte para o primeiro contato com esse ambiente, esse material assim, procurará ser uma versão pessoal dos conhecimentos básicos do Scilab. Acredito no princípio de que há grandes campos para aperfeiçoamento quando se compartilha informações, assim, forneço meu endereço eletrônico (lcol@ufu.br) para que você possa me ajudar, enviando críticas, elogios ou sugestões que servirão para o aprimoramento de versões posteriores desse material.

2 Introdução a Programação Computacional

A automatização de tarefas, procedimentos e processos é um aspecto importante da sociedade moderna. Na Engenharia Química, o aperfeiçoamento tecnológico alcançado tem em alguma parte do seu processo de desenvolvimento elementos fundamentais de análise e da obtenção de descrições e execução de tarefas feitas seja com extrema rapidez, complexidade, repetição eficiente ou precisão. Situações propícias para a sua automatização, ou seja, realizadas por uma máquina especialmente desenvolvida para este fim e conhecida como **computador** (Guimarães & Lages, 1994).

As última décadas presenciou um processo de desenvolvimento simultâneo e interativo de máquinas (*hardware*) e dos elementos que gerenciam a sua execução automática (*software*). A parcela embrionária mais elementar de execução de tarefas é o que se chama de algoritmo. Antes de se iniciar o estudo é interessante se avaliar a trajetória de desenvolvimento das ferramentas de mesma natureza que aquela que compõe o objetivo desse mini-curso.

2.1 Pequena História do *Hardware*

A lista a seguir apresenta uma relação dos acontecimentos principais que resultaram no desenvolvimento das plataformas de análise que conhecemos hoje.

- 1700 ac Povos Mesopotâmicos (primeiras tabuadas, base 60)
- 1200 dc Ábaco chinês 1614 Bastões de Napier (logaritmos)
- 1633 Régua de Cálculo (*Oughtred*)

- 1642 Máquina de Calcular Mecânica (Blaise Pascal)
- 1822 Máquina de Diferenças (Charles Babbage)
- 1833 Máquina Analítica (programável)
- 1880 Perfuradora de Cartões (Herman Hollerith)
  - Criou a Tabulating Machine Company (futura IBM)
- 1939 Computadores Bell à relé (encomenda do exército americano)
- 1941 Z3 (máquina que usa sistema binário) construída por Konrad Zuse
- 1944 Calculadora Automática de Seqüência Controlada (MARK I)
- 1946 ENIAC (*Electronic Numerical Integrator and Calculator*)
- 1949 EDSAC (*Electronic Delay Storage Automatic Calculator*)
- 1951 UNIVAC I (Computador Automático Universal)
- 1953 1^o IBM (IBM 701)
- 1955 IBM 705 (memória de núcleos de ferrite)
- 1958 IBM 709 (entrada e saída de dados paralelamente aos cálculos)
- 1959 IBM 7090 (transistorizado compatível com o IBM 709)
- 1961 IBM 360 (modular)
- 1971 Intel lança primeiro microprocessador
- anos 80 - computador pessoal - até 1 milhão de transistores
-

Para se entrar nos aspectos centrais da utilização do Scilab na resolução de problemas, apresenta-se nas próxima seções uma base para que se possa desenvolver tarefas automatizadas utilizando a plataforma de avaliações do Scilab ou de uma linguagem de programação.

2.2 Algoritmos

O computador é capaz de coletar informações, processá-las e fornecer um resultado com extrema rapidez e eficiência, mas não passam de máquinas de processamento, assim, para que seja capaz de realizar essas tarefas é necessário que desenvolvamos programas, *softwares* capazes de utilizar os recursos da máquina para a computação e fornecimento destes resultados. Uma das etapas fundamentais para o desenvolvimento dos *softwares* é a construção de uma representação lógica a ser instruída ao computador. Essa representação lógica é conhecida como **algoritmo**. Assim, programar é basicamente o ato de construir algoritmos. Existem várias definições sobre o tema, para o nosso interesse, algoritmo é: "a descrição, de forma lógica, dos passos a serem executados no cumprimento de determinada tarefa", assim, o algoritmo constitui-se numa ferramenta genérica para representar a solução de tarefas, ou seja "é uma receita para um processo computacional e

consiste de uma série de operações primitivas, convenientemente interconectadas, sobre um conjunto de objetos". Como o algoritmo pode ter vários níveis de abstrações de acordo com a necessidade de representar detalhes inerentes às linguagens de programação, então eles devem seguir as regras básicas de programação para que sejam compatíveis com as linguagens de programação. A seção a seguir apresenta um pequeno esboço do histórico de desenvolvimento das linguagens de programação nos últimos 50 anos.

2.3 Pequena história da Linguagem de Programação: últimos 50 anos

- ENIAC - programado em linguagem de máquina!
- 1957 - FORTRAN - (***FOR****mula* ***TRAN****slator*) (desenvolvido por pesquisadores da IBM) - potencial para problemas numéricos
- 1958 - ALGOL - (***ALGO****rithmic* ***L****anguage*) potencial para processamento científico
- 1959 - COBOL - (***CO****mmon* ***B****usiness* ***O****riented* ***L****anguage*) potencial para aplicações comerciais
- 1963 - BASIC (***B****eginners* ***A****ll-purpose* ***S****ymbolic* ***I****nstruction* ***C****ode*)- Desenvolvido por Thomas Kurtz e John Kemeny - recursos básicos e facilidade para aprendizado
- 1970
  - PASCAL - (Niklaus Wirth) programação estruturada
  - Linguagem B, desenvolvida por Ken Thompson, nos Laboratórios Bell
- 1973 - Linguagem C, início de desenvolvimento por Denis Ritchie
- 1977
  - FORTRAN 77 - incorporação de conceitos de programação estruturada
  - MODULA 2 - Linguagem também introduzida por Niklaus Wirth
  - Surgimento do Apple II (Basic), TRS-80 (Radio Shack)
- 1981
  - Primeiro PC, lançado pela Acorn (na verdade, a IBM parecia ter cautela de colocar a marca IBM nos computadores pessoais)
  - SMALLTALK
- 1983
  - ADA, Surgimento de Lisa - Apple (primeira máquina com mouse e interface gráfica para usuário)
  - C++, desenvolvido por Bjarne Sroustrup nos Laboratórios Bell
- 1984 - IBM PC (286-AT) e Macintosh. Vários Aplicativos disponíveis- Ex: Lotus 1-2-3, Microsoft Word e Matlab
- ...

- 1989 - Scilab[4] - Biblioteca de Ferramentas para Aplicação em Engenharia.
- 1993 - GNU Octave - potencial para cálculo numérico[5].

O leitor é encorajado a ler várias obras sobre a história do desenvolvimento das linguagens e máquinas. Existe vasto material disponível livremente na WEB (hitmill.com, 2004).

2.4 Construção de um Algoritmo

Quando tem-se um problema e se deseja utilizar um computador para resolvê-lo inevitavelmente tem-se que passar pelas seguintes etapas:

1. Definição do problema e objetivos.
2. Compreensão do problema e objetivos e realização de um estudo da situação atual com verificação de qua(l,is) a(s) forma(s) de resolver o problema.
3. Construção de um algoritmo para a resolução do problema
4. Verificação do algoritmo
5. Utilização de uma linguagem de programação para escrever o programa que deverá resolver o problema.
6. Analisar junto aos usuários se o problema foi resolvido e os objetivos atingidos. Se a solução não foi encontrada, deverá ser retornado para a fase de estudo para descobrir onde está a falha.

As etapas acima são, de forma bem geral, as etapas que um especialista passa, desde a apresentação do problema até a sua efetiva solução (Guimarães & Lages, 1994). Essa apostila tem como objetivo a aplicação do Scilab na etapa de programação. Mas antes de introduzir os aspectos de programação em Scilab é importante introduzir o seguinte conceito: Programar um computador consiste em elaborar um conjunto finito de instruções, reconhecidas pela máquina, de forma que o computador execute estas instruções. Estas instruções possuem regras e uma sintaxe própria, como uma linguagem tipo português ou inglês, sendo isto chamadas de linguagem de computador ou linguagem de programação (Forbellone, 2000).

Na estruturação de um algoritmo, a etapa de sistematização da solução do problema pode ser feita utilizando a filosofia descrita na Figura 1. No mundo computacional existe uma grande variedade de linguagens Pascal, C, C++, Cobol, Fortran, etc. Existe também um conjunto de ambientes desenvolvidos para a facilitação de implementação de algoritmos de determinadas áreas de interesse. Nesse curso enfoca-se o Scilab, que tem como objetivo a simplificação da implementação de algoritmos para a área de análise e controle de sistemas.

2.5 Propriedades de um Algoritmo

- Cada operação deve ser bem definida. Deve ser perfeitamente claro o que deve ser feito.

[4] Criação de Consórcio para Desenvolvimento em 2003.
[5] http://www.octave.org

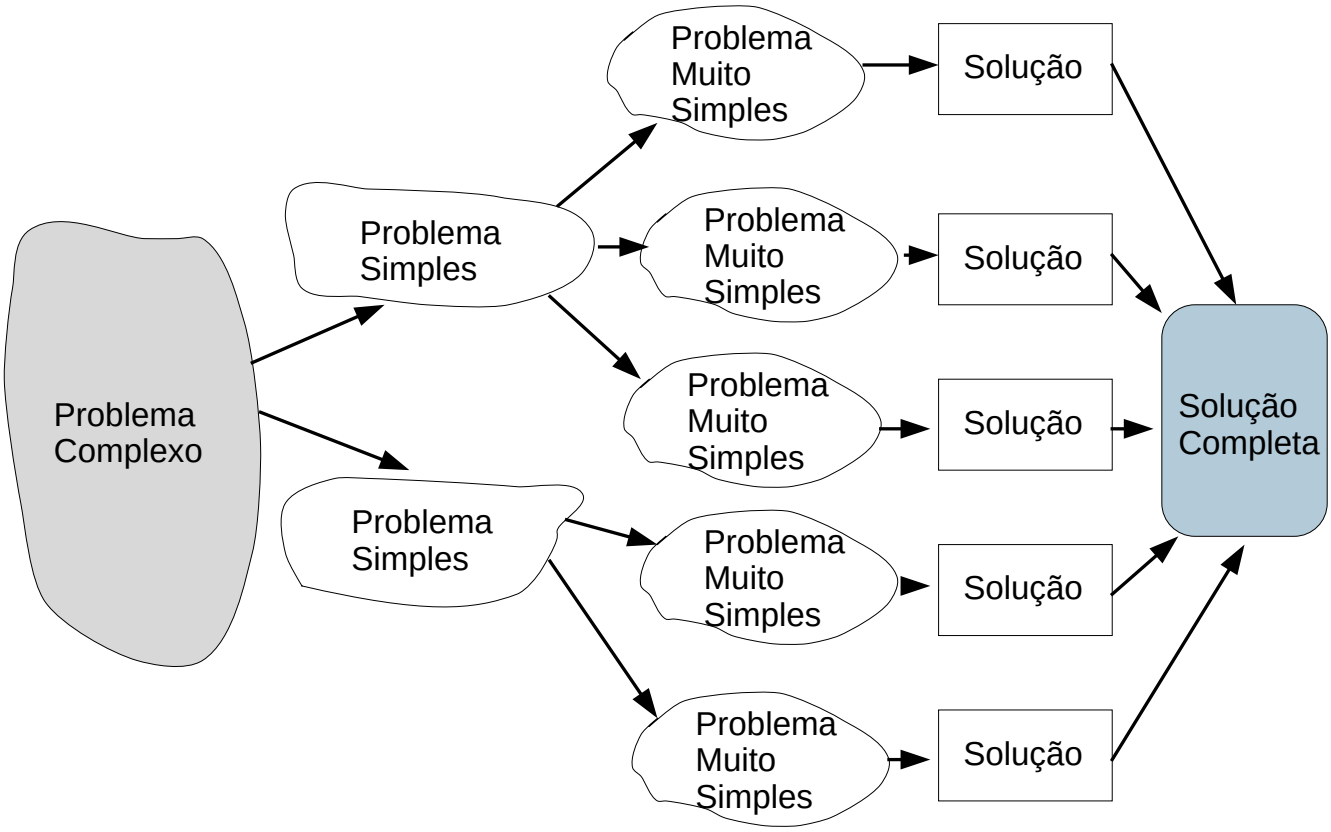

Figura 1: Estruturação para a Solução de Problemas

- Cada operação deve ser efetiva.
- Cada etapa deve ser tal que, pelo menos em princípio, uma pessoa munida apenas de papel e lápis possa executá-la em um tempo finito.
- O algoritmo deve terminar após um número finito de passos.

Um algoritmo que apresenta todas as propriedades anteriores, salvo a de terminação é chamado de procedimento computacional. Ex.: Sistema operacional. Para que um algoritmo seja implementado num computador ele deve ser codificado numa linguagem de programação.

Dado um algoritmo suficientemente preciso, a codificação como um programa de computador é direta. Entre as técnicas atuais de programação, encontram-se:

- Programação Seqüencial
- Programação Estruturada
  - Facilita a escrita de programas
  - Facilita a leitura e o entendimento
  - Antecipa a correção
  - Facilita a manutenção e modificação

  - Possibilita o desenvolvimento em equipe
  - Reduz complexidade

- Programação Orientada a Eventos e Objetos

Existem duas formas principais de representação das tarefas de um algoritmo: o fluxograma simbólico e a apresentação em pseudo-linguagem[6]

2.5.1 A Estrutura Geral de um Algoritmo

A estrutura geral de um algoritmo pode ser representada por:

Algoritmo Nome_do_Algoritmo
Variáveis
    Declaração das variáveis
Procedimentos
    Declaração dos procedimentos
Funções
    Declaração das funções
Início
    Corpo do algoritmo
Fim

Nesse curso, quando se fizer necessário, a representação de um algoritmo, será feita utilizando-se de pseudo-linguagem, no caso utiliza-se aquela conhecida como "PORTUGOL". O "Portugol" ou Português Estruturado é derivado da aglutinação de Português + Algol.

O Scilab é uma linguagem interpretada que pode interagir com módulos compilados e devido a sua natureza, e porque reside num ambiente previamente construído para a sua interpretação, alguns aspectos regulares dos algoritmos não são utilizados. Entre eles destacam-se: (a) Não há denominação interna de nomes para códigos de processamento global, ou seja, apenas os procedimentos em forma de funções recebem nomes, aqueles globais serão referenciados pelo nome do arquivo em disco; (b) Seguindo a estrutura das linguagens interpretadas com base em *scripts*[7], não existe declaração de variáveis no código. Essas determinações são feitas automaticamente ao tempo de interpretação;(c) Os únicos procedimentos de avaliação são as funções e para que estejam disponíveis ao código devem ser carregadas previamente na memória.

A estrutura de um código implementado no Scilab possui então a seguinte forma.

[6] A linguagem natural é prolixa, imprecisa, e pode ser interpretada incorretamente.

[7] Um *script* em Scilab é simplesmente uma série de comandos Scilab em um arquivo. A execução do arquivo faz com que os comandos sejam executados na sua ordem seqüencial como se eles fossem individualmente digitados na linha de comandos do Scilab.

```
[// Apresentação_do_Algoritmo]
[// informações sobre o código e sua utilização]
[// Detalhes do seu desenvolvimento]
[//Pré-processamento]
    Especificação de variáveis internas do Scilab
    Carregamento de bibliotecas para a memória
[//Funções]
    Declaração das funções
[ // Início]
    Corpo do algoritmo
[//Fim]
```

É importante salientar que as instruções apresentadas entre `[]` são opcionais. A nomenclatura adotada é consistente com àquela do próprio Scilab que usa (//) para denominação de comentários. Veja como ficaria por exemplo um algoritmo para a avaliação da média final e do resultado para um aluno em uma disciplina com duas avaliações de igual peso. Considere como resultado aprovado se a média for maior ou igual a $6,0$.

Etapa	Comentário	*Status*
1	Definição do problema e dos objetivos	OK!
2	Compreensão do Problema	Media = (Nota1+Nota2)/2 Média $\geq 6,0 \mapsto$ Aprovado!
3	Construção do Algoritmo	Veja Figura 2 e Tabela 1

A Figura 2 apresenta o fluxograma para o problema acima e a Tabela 1 apresenta o pseudo-código para o mesmo problema.

```
Início
        Ler Nota1,Nota2
        Média ← (Nota1+Nota2)/2
        Se (Média < 6) Então
            Imprime "Não Passou"
        Senão
            Imprime "Passou"
        Fim Se
Fim.
```

Tabela 1: Problema 1: Pseudo-código

2.6 Estruturas de Controle de Fluxo para a Programação em Scilab

As estruturas de processamento lógico de interesse são:

- **Estruturas Condicionais**: As estruturas condicionais, com a sua representação no Scilab tem a seguinte forma:

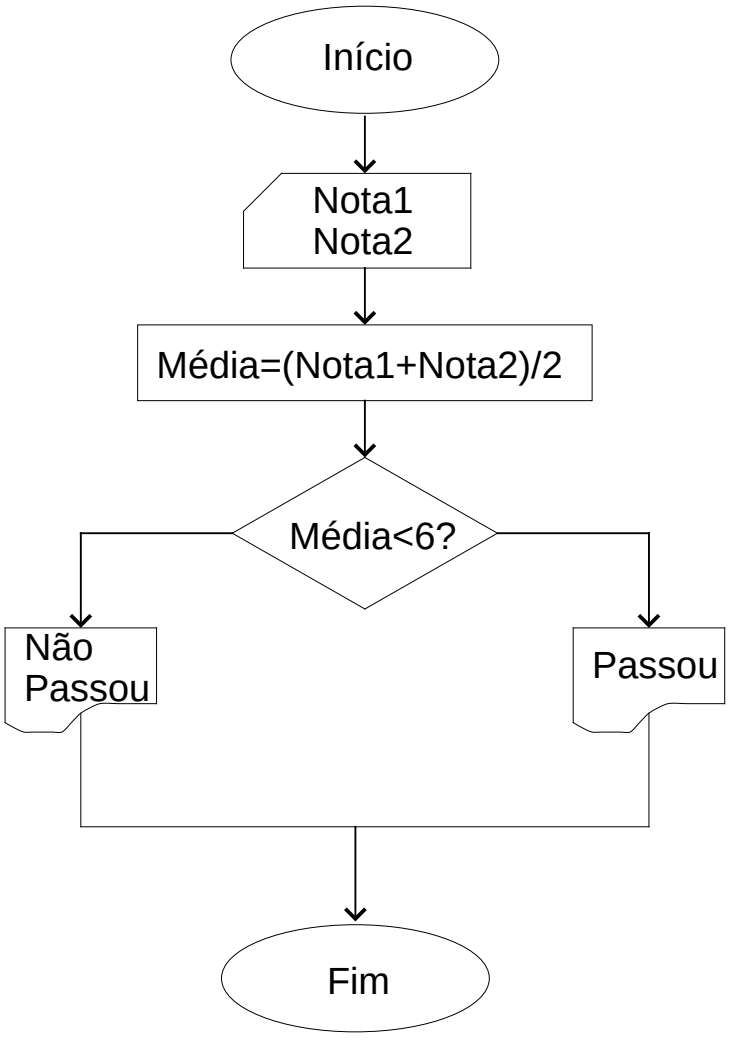

Figura 2: Problema 1: Fluxograma

Estrutura lógica	Comando Scilab
Se (Condição) Então	**if** (Condição) **then**
Executar comandos	Executar comandos
da alternativa verdadeira	da alternativa verdadeira
Senão	**else**
Executar comandos	Executar comandos
da alternativa falsa	da alternativa falsa
Fim Se	**end**

No esquema acima, a condição representa a expressão lógica (booleana) e alternativa representa a seqüência de comandos (vários comandos são possíveis).

Além disso, as estruturas condicionais podem ocorrer em ninhos (*nest*). Observe a Figura 3 para a verificação das estruturas válidas e inválidas: A representação de estruturas condicionais encadeadas é dada por:

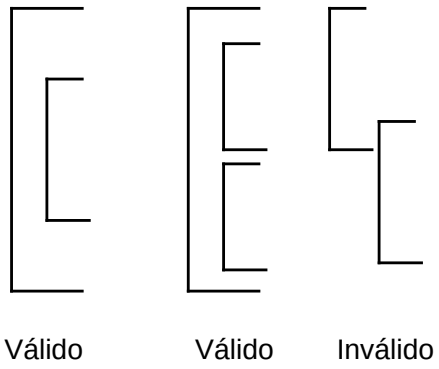

Figura 3: Estruturas Condicionais.

Estrutura lógica	Comando Scilab
Se (Condição 1) Então	**if** (Condição 1) **then**
...	...
Se (Condição 2) Então	**if** (Condição 2) **then**
...	...
Senão	**else**
...	...
Fim Se	**end**
...	...
Senão	**else**
...	...
Se (Condição 3) Então	**if** (Condição 3) **then**
...	...
Senão	**else**
...	...
Fim Se	**end**
...	...
Fim Se	**end**

Ou podem-se ter estruturas dadas por:

Estrutura lógica	Comando Scilab
Se (Condição 1) Então	**if** (Condição 1) **then**
...	...
Senão Se (Condição 2) Então	**elseif** (Condição 2) **then**
...	...
Senão	**else**
...	...
Fim Se	**end**

- **Estruturas Repetitivas**

  As estruturas repetitivas, laços ou *loops* podem ser de vários tipos:

  – Controlados por contador: Uma variável é fornecida com o no. de vezes que será repetido o laço.

– Controlados por sentinela: Ao final da entrada de dados é fornecido um dado especial que sinaliza o fim do processamento.

– Laços contados: O laço é executado um no. fixo de vezes, utilizando uma variável para controle do laço.

– Laços com condição no final: Faz o teste no final da estrutura.

Para propósitos do Scilab, as estruturas de repetição podem ser representadas por:

1. O laço **enquanto**: A estrutura do laço enquanto é dada por[8]:

Estrutura lógica	Comando Scilab
Enquanto (Condição) Então ... (Comandos) ... [Senão] [(Comandos)] [...] Fim Enquanto	**while** (Condição) **then** ... (Comandos) ... [**else**] [(Comandos)] [...] **end**

O comando **while** no Scilab possui as seguintes variações[9]:

while condição **,** instruções,...[,**else** instructions], **end**
while condição **do** instruções,...[,**else** instructions], **end**
while condição **then** instruções,...[,**else** instructions], **end**

2. O laço **for**: A estrutura de repetição **for** pode ser utilizada no Scilab com a seguinte sintaxe:

for variável=expressão **do** instrucão1,...,instruçãoN,**end**
for variável=expressão**,** instrucão1,...,instruçãoN,**end**

Se **expressão** é uma matriz ou um vetor linha, a variável recebe os valores de cada coluna da matriz[10]. Se a **expressão** é uma lista a **variável** recebe os valores das sucessivas entradas da lista. Exemplos de utilização

Exemplo 1	Exemplo 2	Exemplo 3
for i=1:2:10**,** ... (Comandos) ... **end**	**for** a=[7, 9, -1, 8, 12] **do** ... (Comandos) ... **end**	**for** k=list(1,2,'exemplo') **do** ... (Comandos) ... **end**

- **Estrutura de Condição**

A estrutura de condição é equivalente a um ninho de estruturas condicionais (Se-Então-Senão), a sua sintaxe é dada por:

[8] Os parte indicadas entre colchetes [ ] são opcionais.

[9] Nota-se que o comando possui três diferentes sintaxes: usa-se then ou do ou (,). Além disso, independentemente da estrutura opcional **else**, sempre finaliza-se o comando **while** com **end**.

[10] Um expressão muito usual é a dada por [**n1:incremento:n2**], onde a variável receberá todos os valores de n1 até n2 de incremento a incremento. Ex.: [3:-0.1:2.5]=[3, 2.9, 2.8, 2.7, 2.6, 2.5]. Quando não indicado o valor do **incremento** é dado por 1.

```
select expressão,
      case expressão1 then instrucões1,
      case expressão2 then instrucões2,
      ...
      case expressãoN then instrucõesN,
[else instruções],
end
```

2.7 Aspectos Básicos para a Programação em Scilab

- **Operadores Lógicos para Condições Compostas**

  As condições compostas utilizam os conectivos **E**(&), **OU**(|) e **NÃO**(∼).

  – Conjunção **E**: (condição 1 & condição 2): retorna verdade (V) quando as duas condições são verdadeiras.

  – Disjuncão **OU** :(condição 1 | condição 2): Basta que uma condição seja verdadeira para a expressão ser verdadeira.

  – Negação **NÃO**: (∼(condição)): nega a condição, de verdadeiro passa para falso (F) e vice-versa.

  Para ilustrar a utilização dos operadores lógicos acima, apresenta-se a tabela verdade das suas operações. Sejam as proposições lógicas p e q, então sabe-se que:

p	q	$p\&q$	$p\|q$	$\sim p$
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

  As expressões lógicas podem ainda utilizar os seguintes operadores relacionais:

Operadores relacionais	
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual
==	Igual
∼ = ou <>	Diferente

- **Comentários**

  A utilização de comentários faz parte do conjunto de boas práticas na programação computacional. Os comentários devem ser descritivos e no Scilab devem ser colocados após o sinal de barras duplas (//), objetivando ajudar o leitor a entender melhor um trecho de algoritmo, e não para indicar qualquer ação a ser executada. É considerado boa prática a utilização de comentários no início de cada *script*, para:

  – Uma descrição, sucinta, do que faz o algoritmo

– Se necessário, como utilizar o algoritmo
– Explicação do significado das variáveis mais importantes
– A estrutura de dados utilizada
– Autor e datas de escrita e última revisão.

2.8 Definições Básicas

1. **Dados**: Um algoritmo manipula dados. Os dados podem ser de vários tipos:

   (a) Numéricos
   - inteiros: 1; 13; -45
   - reais: 34.7; -8.76; 0.23
   - ponto flutuante: 0.456xE9

   (b) Não numéricos
   - caracteres: 'L'; 'S'
   - cadeia de caracteres (*strings*):'SCILAB'; '1990'

   (c) Lógicos
   - VERDADEIRO (*TRUE*): $(6 > 3)$
   - FALSO (*FALSE*): $(4 < 8)$

   (d) Objetos: Existem vários tipos de objetos, um exemplo importante são as LISTAS: $(dias.da.semana = [segunda, quarta, sexta, ...])$.

   No Scilab o usuário poderá definir outras outras estruturas de dados. Existem dois tipos de estruturas de dados:

   - Estruturas primitivas: inteiro, caracter, real etc. A maioria dos computadores têm instruções para manipulá-las.
   - Estruturas não-primitivas: vetores, matrizes etc. Vetor - É um conjunto que contém um número fixo de elementos de um mesmo tipo. Cada elemento do vetor está associado a um índice único.

   A operação dos dados no Scilab dá-se através de operadores:

operador	sinal	exemplo
Soma	$+$	$2+3$
Subtração	$-$	$3-4$
Multiplicação	$*$	$2*5$
Divisão	$/$	$5/2$
Potência	^	2^3
	$**$	$2**3$
Conjugado transposto	'	a'

Tabela 2: Operadores de Uso Comum

2. **Constantes**: Entidades que não mudam de valor durante a execução do algoritmo. O SCILAB possui um número de constantes especiais, tais como:

Constante	representa
%i	$\sqrt{-1}$
%pi	$\pi = 3.1415927.....$
%e	constante $e = 2,7182818$, base dos logaritmos naturais
%eps	constante representando a precisão da máquina, 1 + %eps = 1
%nan	não é um número (*not a number*)
%inf	infinito
%s	é o polinômio $y(s) = s$, s=poly(0,'s').
%t	constante lógica para VERDADEIRO (*true*), ex. 1 == 1
%f	constante lógica para FALSO (*false*), ex. $\sim \%t$.

3. **Variável**: Entidade que armazena dados e pode mudar de valor durante a execução de um algoritmo. Possui apenas um valor num determinado instante. Possui um nome e um tipo. Constitui boa prática a escolha de nomes que indicam a função da variável no algoritmo[11].

   Ao invés de $S = (a * b)/2.0$ recomenda-se: $Area = (Base * Altura)/2.0$, pois é bem mais compreensível.

   A escolha dos nomes de variáveis deve utilizar dos seguintes critérios:

   - Deve começar com uma letra.
   - O restante pode ser letra ou dígito ou algum caracter especial permitido.

   Variável Global e Local: Variáveis locais são conhecidas apenas no subalgoritmo que as define, portanto desconhecidas para o algoritmo principal. As variáveis globais são conhecidas por todos as funções do algoritmo[12].

4. **Expressões**: Forma de manipular variáveis, definindo operações sobre elas.

5. **Operação de Atribuição**: Especifica que uma variável recebe determinado valor. Em pseudo-linguagem é usualmente indicada pelos símbolos: $\leftarrow$ ou =.

   Observação 1: Uma variável só pode receber valores do tipo em que ela foi definida. Exemplo:

   - Variáveis reais só recebem valores reais: $X \leftarrow 5.6$;
   - Variáveis Tipo Caracter: $Nome \leftarrow' Maria'$;
   - Tipo Booleano: Condição $\leftarrow Falso$

   Forma geral:

   variável ←expressão

   ou

   variável = expressão.

   Observação 2: Variáveis do lado direito não sofrem alteração. Toda variável usada no lado direito deve possuir um valor antes da expressão ser avaliada.

[11] Embora algumas linguagens aceitem nomes de variáveis com acentos, constitui prática comum não se acentuar as palavras que nomeiam as variáveis nos códigos em linguagem de programação. No caso do Scilab, a acentuação não é reconhecida e não deve ser usada na definição de variáveis ou nomes de funções.

[12] É considerada boa prática na programação computacional a utilização de variáveis globais apenas quando extremamente necessária.

6. Subalgoritmos: Como apresentado na Figura 1 vê-se que a construção de um algoritmo pode ser dividida em vários subalgoritmos (subproblemas), esses subproblemas[13] podem ser utilizados modularmente em várias partes do problema principal. A Figura 4 ilustra esse fato. A

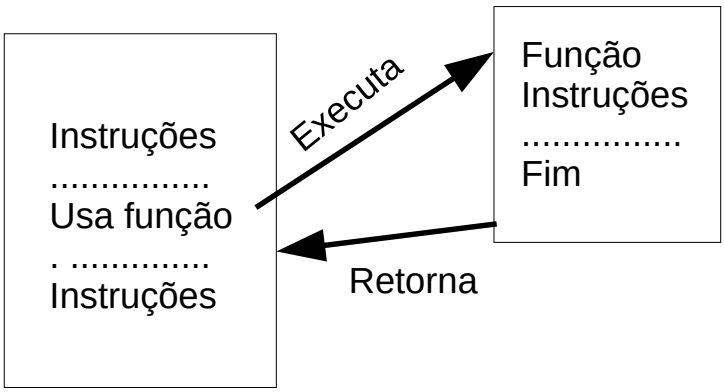

Figura 4: Estruturas de códigos usando funções

passagem de argumentos para uma função pode ser de vários tipos:

- Por referência (Pascal, Algol): O endereço do argumento é passado para o algoritmo chamado. Tudo que é feito com a variável que recebeu o argumento também tem efeito na variável de origem. Também chamada "por variável" ou "por endereço".
- Por valor (C, PLM, ADA): O valor, e apenas o valor, do argumento é copiado para o parâmetro da chamada. Qualquer ação sobre esta variável no procedimento chamado não tem efeito na variável ou expressão de origem.
- Por resultado (ADA): Nada é passado na chamada ao procedimento. Somente quando a execução do procedimento termina é que o valor da variável usada como parâmetro é que é copiado para a variável do procedimento principal.
- Por valor-resultado (ADA): Combina os dois tipos anteriores.
- Por nome (Algol): O parâmetro recebe o NOME da variável do argumento da chamada.

7. Recursividade: Um subalgoritmo pode chamar a si próprio.

3 O Ambiente do SCILAB

3.1 Interface Gráfica do Ambiente Scilab

O Scilab possui uma interface gráfica facilmente personalizada. A Figura 5 apresenta o janela de interface do Scilab com o usuário.

O *menu* "**File**": As seguintes opções estão disponíveis:

- **New Scilab**: use-o para abrir outro ambiente do Scilab.
- **Exec ...**: use-o para executar um arquivo (*script*) do Scilab.

[13] No Scilab os subalgoritmos são chamados de funções.

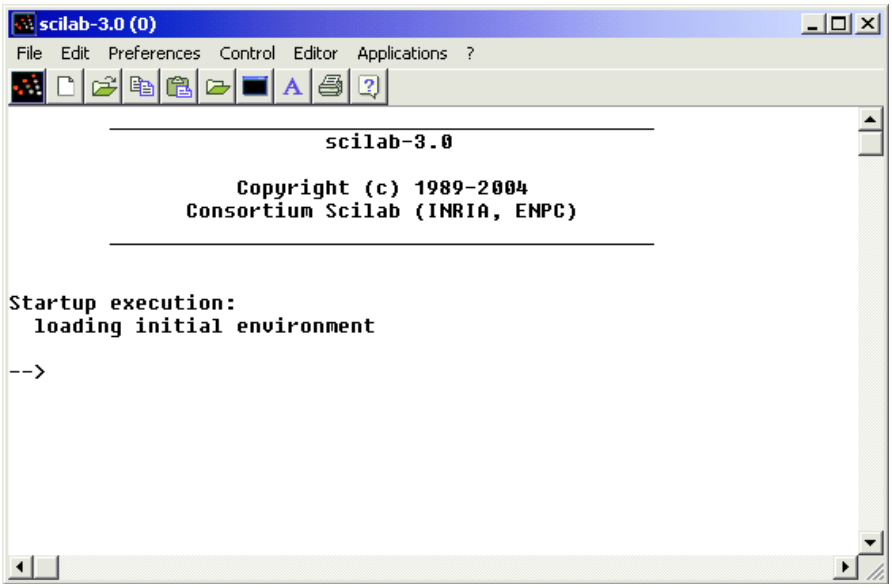

Figura 5: SCILAB: Ambiente de interação com usuário.

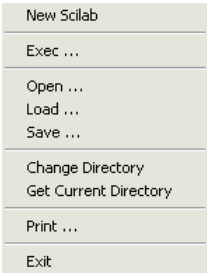

Figura 6: SCILAB: Item "File"do Menu.

- **Open ...**: use-o para carregar um *script* no editor padrão do Scilab (Scipad).
- **Load ...**: use-o para carregar para a memória variáveis que tenham sido anteriormente salvas através de **Save**.
- **Save ...**: use-o para salvar variáveis.
- **Change Directory**: Mudar diretório de trabalho.
- **Get Current Directory**: Verificar diretório de trabalho.
- **Print ...**: use-o para imprimir relatório de comandos executados e ainda disponíveis no ambiente Scilab.
- **Exit**: use-o para sair so Scilab

O *menu* "**Edit**": As seguintes opções estão disponíveis:

- **Select All**: Seleciona texto.

- **Copy**: copia texto selecionado para *clipboard*.
- **Paste**: cola texto em *clipboard* para posição atual do cursor no Scilab.
- **Empty Clipboard**: Esvazia *clipboard* de textos anteriormente copiado para o mesmo.
- **History**: acessa um *menu* de comandos para edição de comandos do Scilab.

O *menu* "**Preferences**": As seguintes opções estão disponíveis:

- **Language**: seleciona opção de língua para interface do Scilab. Opções: Francês e Inglês.
- **Toolbar (F3)**: ativa/Desativa barra de ferramentas do Scilab.
- **Choose Font**...: permite escolha de fontes para janela principal do Scilab.
- **Clear History**: limpa histórico de comandos da seção em andamento no Scilab.
- **Clear Command Window (F2)**: limpa comandos processados. Equivalente ao comando **clc**.
- **Console (F12)**: Apresenta/Oculta o console de saída do Scilab.

O *menu* "**Control**": As seguintes opções estão disponíveis:

- **Resume**: retorna ao modo de execução após modo de pausa[14].
- **Abort**: finaliza execução do programa corrente.
- **Interrupt**: interrompe a execução do programa corrente. Essa opção permite entrar no modo de pausa, contudo ela interromperá qualquer operação sendo executada por Scilab.

O *menu* "**Editor**": Abre editor SciPad para desenvolvimentos de *scripts*, vide Figura 7.

O *menu* "**Applications**": As seguintes opções estão disponíveis:

- **Scicos**: abre ambiente do Scicos.
- **EditGraph**: abre janela de editor gráfico.
- **m2sci**: conversor de *scripts* de Matlab para Scilab (vide Figura (8)).
- **Browser Variables**: Sistema que apresenta variáveis correntes na memória do Scilab (vide Figura (9)).

O *menu* "**?**": As seguintes opções estão disponíveis:

- **Scilab Help (F1)**: disponibiliza janela com telas de ajuda de comandos e funções do Scilab. Pode ser acessado diretamante digitando-se **help <comando>** ou simplesmente **help**.

[14]No modo de pausa o Scilab cria um novo *prompt*, nesse modo pode-se entrar comandos do Scilab sem afetar os cálculos do modo principal do Scilab, indicado pelo prompt `-n->`. Pode-se ter vários níveis do modo de pausa, para sair de um modo de pausa e retornar para o superior, entre <resume>, para subir dois níveis use o comando <abort> ou <quit>, esse último comando quando executado fora do modo de pausa forçará a finalização da seção do Scilab.

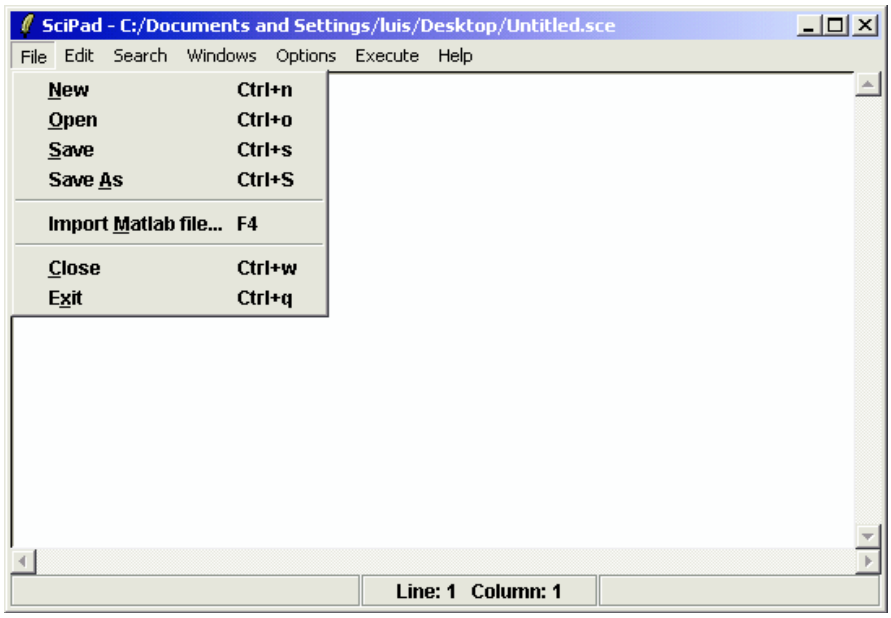

Figura 7: SCIPAD: Editor para construção de *scripts* no Scilab. Disponibilidade de conversão de scripts Matlab em Scilab.

- **Configure**: seleção de *browser* a ser utilizado pelo Scilab.
- **Scilab Demos**: apresenta uma tela com um conjunto de programas demonstrativos de várias áreas de conhecimento.
- **Report a bug or a request**: interage com time de desenvolvimento do Scilab através do *Scilab tracking system*.
- **Scilab Web Site**: visita a página do Scilab.
- **Scilab Newsgroup**: visita a página de discussões dos usuários do Scilab.
- **About**: Informações sobre o Scilab, incluindo a licença.

3.2 Iniciando o Uso do Scilab

Existem duas maneiras de se utilizar o Scilab. na forma de calculadora (ambiente de linhas de comandos) e na execução de *scripts*.

Para se carregar na memória o valor de uma variável usa-se uma atribuição (com o sinal de =), por exemplo:

$-->a = 4.5;$ <return>

A finalização do lado direito da atribuição pode conter o ";" (ponto e vírgula) ou não , caso a atribuição não tenha o ";" o Scilab reportará o valor da variável fornecida[15].

$-->a = 4.5$ <return>

[15]Essa declaração supõe que o ambiente Scilab esteja funcionando com **mode**(0), caso esteja em **mode**(-1) o Scilab omitirá o eco dos comandos. Para detalhes veja a função **mode**.

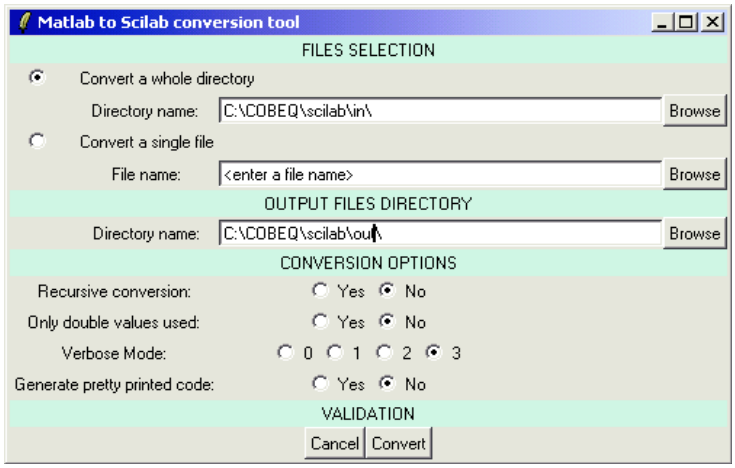

Figura 8: Janela do conversor de scripts Matlab para Scilab.

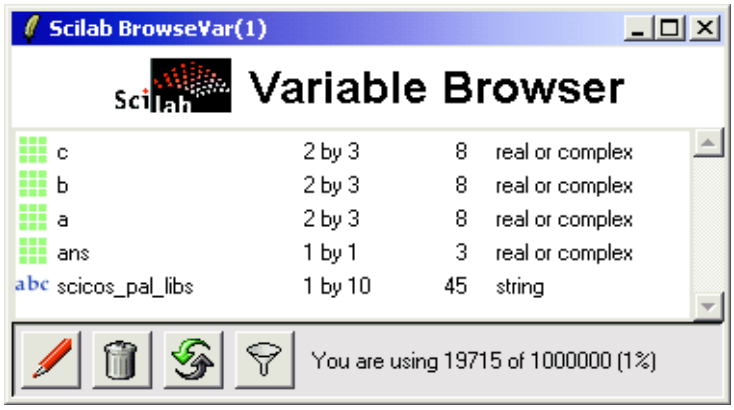

Figura 9: Janela do *browser* de variáveis do Scilab.

a= 4.5

Assim, após proceder as seguintes operações:

a=4.6;<return>
b=2.3;<return>
c=a/b;<return>
d=c^c <return>

O Scilab reportará, $d = 4.$, pois é o único comando que não apresentou a finalização de atribuição com o ";".

3.3 Aspectos Básicos

3.3.1 Comentários e escalares

Como anteriormente apresentado, os comentários para o Scilab são representados por duas barras, //. Para os Scilab, qualquer objeto (incluindo, número real, string, variável lógica, polinômio, função racional) que não estiver entre colchetes, será considerada um objeto escalar.

Exemplos :

```
a = 1 // constante real <return>
2>1 // constante lógica <return>
'minha casa' // string de caracter <return>
r = poly(1.,'x') // polinômio com variável 'x' e raiz em 1,0 <return>
```

3.3.2 Expressões e Variáveis

Analogamente a grande maioria das linguagens de programação, o Scilab possui expressões matemáticas como atribuição, mas diferentemente da maioria das linguagens, o Scilab trabalha com objetos de várias formas, inclusive matrizes.

Um aspecto importante para se lembrar na preparação do código em Scilab é que o mesmo não necessita de qualquer tipo de declaração ou definição de dimensão para as variáveis usadas no código. Quando Scilab encontra um novo nome de uma variável, ele automaticamente cria a variável e aloca o tamanho apropriado para o seu armazenamento na memória. Se a variável já existe, Scilab muda o seu conteúdo e se necessário, aloca novo espaço para o seu armazenamento na memória.

Scilab é sensível ao caso, ou seja, uma variável, A é diferente da variável a. O nome da variável em Scilab é dado por uma letra seguida de um número de letras, dígitos ou sublinhado. Para se ter acesso ao valor contido em uma variável basta fornecer o seu nome.

3.3.3 Dados do tipo *list*

A lista (**list**) é uma coleção de objetos não necessariamente do mesmo tipo, podendo conter outros objetos (inclusive funções e outras listas). As listas são muito úteis para a definição de outros estruturas de dados. Existem dois tipos de listas: (a) listas ordinárias e (b) listas com tipo definido (**typed-list**).

1. A lista **list**: lista ordinária é definida no Scilab através do comando **list**, com sintaxe:

   ```
   list(a1,....an)
   ```

   que cria uma lista com elementos arbitrários. list() define uma lista vazia. As operações mais comuns com as listas são:

   - extração $[x, y, z...] = L(v)$, onde v é o vetor de índices; $[x, y, z] = L(:)$ extrai todos os elementos.

- inserção: $L(i) = a$
- remoção: $L(i) = null()$ remove o e-ésimo elemento da lista L.

Exemplos para listas ordinárias (**list**) :

−− >`L=list(1,'w',ones(2,2));`<return>

A lista acima possui três elementos: $L(1) = 1$, $L(2) =' w'$ e $L(3) = ones(2, 2)$, sendo que cada elemento da lista pode ser acessado diretamente através da invocação da sua posição na lista, exemplo:

−− >L(1)<return>
ans =
1.
−− >L(3)<return>
ans =
! 1. 1. !
! 1. 1. !
Pode-se ter acesso direto aos componentes de um elemento de uma lista, exemplo:
−− >L(3)(2,2)<return>
ans =
1.

2. A lista **tlist**: As listas **tlist** possuem um primeiro elemento específico. Esse primeiro elemento deve ser uma *string* de caracter (o tipo) ou um vetor de caracteres (com o primeiro elemento definindo o tipo e os elementos seguintes os nomes dados às entradas da lista). A lista **tlist** é definida no Scilab através do comando **tlist**, com sintaxe:

tlist(tipo,a1,....an)

que cria uma lista com elementos arbitrários após o primeiro elemento, tipo, que é uma *string* de caracter ou vetor de *string* de caracter. As operações de extração, inserção e remoção de **list** também são válidas para **tlist**, além disso se o tipo especificar os nomes dos campos, então pode-se referir diretamente aos mesmos. Um importante aspecto de listas **tlist** é que pode-se definir operadores que agem nos mesmos (*overloading*), assim pode-se definir por exemplo a multiplicação $L1 * L2$ de duas listas **tlist** $L1$ e $L2$.

Exemplos para listas (**typed-list**) :

−− >L=tlist(['Nome';'Professor';'Vetor'],'Luís Cláudio',[2 3]);<return>
L =
L(1)
! *Nome* !
! !
! *Professor* !
! !
! *Vetor* !
L(2)
Luís Cláudio
L(3)

```
! 2. 3. !
-- >L.Professor
ans =
Luís Cláudio
-- >L(2)
ans =
Luís Cláudio
-- >L.Vetor(2) // Equivalente a L(3)(1,2)
ans =
    3.
```

3.3.4 Arquivo *diary*

Um arquivo diary é simplesmente um arquivo texto que contém os comandos do Scilab bem como as suas respostas de uma seção do Scilab. Para se ativar o arquivo diary utiliza-se o comando **diary**. Conforme indicado abaixo:

comando	resultado
`diary(''c:\arquivo.txt'')`	inicia arquivo *diary* para arquivo `c:\arquivo.txt`
`diary(0)`	finaliza arquivo *diary*

Após a execução do comando **diary** com uma especificação de arquivo, todo o comando do usuário e os resultados do Scilab na janela do Scilab serão arquivados em arquivo texto até que o comando **diary(0)** seja executado.

3.3.5 Operadores para Matrizes

Além dos operadores já definidos na Tabela (2), o Scilab possui os seguintes operadores para matrizes:

Operador	Função	
[]	definição de matriz e concatenação	
;	separador de linhas	
()	extração de elemento	$m = a(k)$
()	inserção de elemento	$a(k) = m$
′	transposta	
+	adição	
−	subtração	
∗	multiplicação	
\	divisão a esquerda	
/	divisão a direita	
^	potenciação	
.∗	multiplicação elemento a elemento	
.\	divisão à esquerda elemento a elemento	
./	divisão à direita elemento a elemento	
.^	potência elemento a elemento	
. ∗ .	produto de kronecker	
./.	divisão de kronecker a direita	
.\.	divisão de kronecker a esquerda	

3.3.6 Funções

Scilab proporciona um número bastante grande de funções elementares. Algumas funções, para aumentar eficiência, são construídas internamente (*built-in*) ao Scilab. Mas, funções inexistente podem ser adicionadas através de programação da mesma como *scripts*, que quando carregados na memória podem ser diretamente executados. A Tabela abaixo apresenta as funções básicas mais simples disponíveis no Scilab.

Funções Básicas do SCILAB

clc - limpa tela de comandos
chdir - muda diretório padrão
exec - executa função
getf - carrega funções na memória
help - ajuda
spec - calcula valores característicos
rank - calcula posto de matriz
trace - calcula traço de matriz
abs - valor absoluto
conj - conjugado
diag - matriz diagonal
eye - matriz identidade
floor - arredondamento para baixo
imag - parte imaginária
integrate - integração por quadratura
interp - interpolação
interpln - interpolação linear
intersect - retorna um vetor com valores comuns entre dois vetores
linspace - vetor linearmente espaçado

log - logaritmo natural
log10 - logaritmo em base 10
max - máximo
min - mínimo
modulo - calcula o resto de uma divisão
ndims - número de dimensões de uma matriz
norm - norma de matriz
ones - matriz de 1's
rand - gerador de números randômicos
real - parte real
solve - resolve sistema linear simbólico
sort - ordenamento decrescente
sqrt - raiz quadrada
sum - soma elementos de matrizes
syslin - definição de sistema linear
trisolve - resolve simbolicamente sistema linear
diary - diário da seção
disp - mostra variáveis
fprintf - emula a função fprintf da linguagem C
fscanf - leitura formatada de um arquivo
fscanfMat - ler uma matriz de um arquivo texto
getio - unidade lógica de entrada/saída para Scilab
input - solicita entrada de dados para o usuário
load - carrega variáveis salvas na memória
printf - Emula a função printf do C
scanf - entrada formatada de dados plot - faz gráfico
plot2d - faz gráfico 2D
xsave - salva gráfico em arquivo
xtitle - adiciona título em janela gráfica
det - determinante
inv - matriz inversa
linsolve - resolve sistema linear
lsq - problema de mínimos quadrados
clear - limpa variáveis da memória
file - função para manipulação de arquivos. Ex.: `u = file(‘‘open’’,‘‘c:\dados.txt’’,‘‘new’’)`
who - apresenta variáveis carregadas na memória do Scilab

3.4 Scilab: Primeiros Passos

Nessa seção apresentam-se algumas atividades no ambiente calculadora, na próxima seção aspectos de programação serão introduzidos.

3.4.1 Carregando variáveis

Faça o seguinte exercício no Scilab:

−− >a = 1.4 <return>
−− >b = 1.6; <return>
−− >a+b <return>
−− >a-b <return>
−− >a*b <return>
−− >a/b <return>
−− >a^b <return>
−− >who <return>
−− >save a <return>
−− >clear a <return>
−− >a <return>
−− >b <return>
−− >load a <return>
−− >a <return>
−− >sin(a*%pi/b) <return>
−− >c = 3.2; <return>
−− >v=[a, b, c] <return>
−− >w=[a; b; c] <return>
−− >A = [1. 2. 3.; 4. 5. -6.; 1. -1. 0.] <return>
−− >B = [1. 1. 1. <return>
−− >2. 3. -1. <return>
−− >5. 7. -2.] <return>

Para se acessar os elementos de uma matriz pode-se usar o comando (i, j), assim para se acessar ao elemento na segunda linha e 3 coluna da matriz B acima, faz-se:

−− > B(2,3)<return>

que fornecerá o valor -1. Este elemento também pode ser acessado através do comando $B(8)$, pois ocupa a posição 8 da matriz B que é 3×3, com armazenamento por coluna[16].

−− > B(2,1)<return>
ans =
2.

Pode-se apagar linhas ou colunas de uma matriz utilizando-se colchetes vazios. Por exemplo:

−− > S = [1 2 3 4; 5 6 7 8; 9 10 11 12];<return>
−− > S(:,2) =[] <return>
S =
! 1. 3. 4. !
! 5. 7. 8. !
! 9. 11. 12. !

Existem várias funções básicas para a manipulação e operação de matrizes[17].

[16]Se o usuário tenta usar um número em uma posição fora da matriz, o Scilab fornecerá um erro. Se ao contrário deseja-se especificar um número para uma posição fora da matriz o Scilab automaticamente aumentará a dimensão da mesma para acomodá-lo. Ex:
−− > B(4,1)<return> fornecerá um erro (erro 21), por outro lado,
−− > B(4,1)=2<return> aumentará a matriz B com esse valor e os outros elementos para completar a dimensão da matriz serão nulos.

[17]Pode-se definir na Scilab uma matriz multidimensional. Seja o exemplo de acesso ao elemento da matriz multidimensional $i \times j \times k \times l \times m \times n$, é $A(i, j, k, l, m, n)$. Pode-se também utilizar o comando com sintaxe:

M=hypermat(dims [,v])

com dims sendo o vetor of dimensões e v (opcional) o vector elementos (valor *default* é dado por zeros(prod(dims),1))

Exemplos:
A função **zeros** cria matriz de elementos nulos,
−− >A = zeros(3,2)<return>
A =
! 0. 0. !
! 0. 0. !
! 0. 0. !
A função **ones** cria matriz com todos elementos iguais a 1,
−− >B = ones(3,2) <return>
B =
! 1. 1. !
! 1. 1. !
! 1. 1. !
A função **eye** cria matriz identidade. A função **eye** utiliza as seguintes sintaxes:

X=eye(m,n)	retorna matriz identidade de dimensão* $m \times n$
X=eye(A)	retorna matriz identidade com mesma dimensão* que A
X=eye( )	produz uma matriz identidade com dimensões indefinidas$^{\diamond}$

*O comando é aplicável mesmo quando a matriz desejada não é uma matriz quadrada.
*Atenção: eye(10) é interpretado como eye(A) e A=10, ou seja, retorna uma matriz identidade de dimensão 1.
$^{\diamond}$As dimensões serão definidas quando a matriz for adicionada a uma matriz com dimensões fixas.

A função **ceil** produz um valor inteiro imediatamente superior ao argumento fornecido, por exemplo,
−− >ceil(4.35) <return>
ans =
5.
Por outro lado a função **floor** fornece o inteiro imediatamente menor que o argumento fornecido,
−− >floor(3.15)<return>
ans =
3.
A função **round** é uma função que arredonda os números do argumento para o seu inteiro mais próximo, assim, round(3.15) fornecerá a resposta igual a 3. e round(3.65) fornecerá a resposta igual a 4.
Para arredondar para baixo pequenos números, por exemplo o caso da matriz **A** abaixo utiliza-se a função **clean**,
−− >A=[1 1e-15; 1e-10 2];<return>
−− >B=clean(A)<return>
B =
! 1. 0. !
! 0. 2. !
A função **rand** é usada para gerar vetores de números randômicos igualmente espaçados na faixa de $[0, 1]$. Para exemplificar, a geração de um vetor linha com 5 números randômicos é dada por,
−− >rand(1,5)<return>
ans =
! .2113249 .7560439 .0002211 .3303271 .6653811 !
O Scilab possui uma biblioteca significativa para o cálculo polinomial. Veja por exemplo como se define um polinômio usando a função **poly**,

−− >p=poly([1 2 3],'z','c') <return>
p = 1 + 2z + 3z^2
Esse polinômio pode ser definido também, por exemplo na variável s, fazendo-se:
−− >s=poly(0,'s');<return>
−− >p=1+2*s+3*s^2
p = 1 + 2s + 3s^2
E suas raízes extraídas através do comando **roots**:
−− >r=roots(p)<return>
r =
! − .3333333 + .4714045i!
! − .3333333 − .4714045i!
A função **poly** com dois argumentos é interpretada como se o primeiro argumento seja dado pelas raízes do polinômio,
−− >q=poly(r,'x')<return>
q =
.3333333 + .6666667x + x^2
−− >3*q <return>
ans =
1 + 2x + 3x^2

Operadores úteis para manipulação de matrizes:

Operador	função
$a : b : c$ ou $a : c$	gera faixa de números de a até c de b em b
:	refere-se a "toda linha" ou "toda coluna"
$	último índice de matriz ou vetor
'	operador transposto conjugado
$C * D$	operador para multiplicação de matrizes C e D
$C. * D$	operador para multiplicação de matrizes C e D, elemento a elemento.
$C\backslash D$	divisão à esquerda. Equivale a $inv(C) * D$

Exemplos:
−− > [1:2:6]<return>
ans =
! 1 3 5 !
−− > A(:,$) <return>
ans =
! 3. !
! − 6. !
! 0. !
−− >D=B' <return>
D =
! 1. 2. 5. 2. !
! 1. 3. 7. 0. !
! 1. − 1. − 2. 0. !
−− >A*D<return>
ans =

```
    !  6.      5.    13.   2. !
    !  15.    17.    43.   8. !
    !  0.    − 1.    − 2.   2. !
−− >D.*D <return>
ans =
    ! 1.    4.    25.    4. !
    ! 1.    9.    49.    0. !
    ! 1.    1.    4.     0. !
```

É importante ressaltar que o operador (') refere-se a operação transposta conjugada,

```
−− >A=[1+2*%i 2; 0 3+4*%i] <return>
A =
    ! 1. + 2.i    2. !
    ! 0    3. + 4.i !
−− >A' <return>
ans =
    ! 1. − 2.i    0 !
    ! 2.    3. − 4.i !
```

3.4.2 Operações de entrada/leitura de dados

Os comandos abaixo são muito úteis para a leitura de dados, principalmente em modo de programação:

Leitura via teclado

Função	Sintaxe
input	[x]=input(mensagem,[”string”])
scanf(formato);	ler dados formatados via teclado
read(%io(1),...)	leitura de dados formatados via teclado
x_dialog	r=x_dialog(labels,valor_inicial)
getvalue	[ok,x1,..,x14]=getvalue(Título,labels,tipos,inicial)
x_mdialog	r=x_mdialog(titulo,labels,valor_default_inicial)
x_message	[n]=x_message(strings [,botões])
x_choose	[n]=x_choose(items,titulo [,botão])
x_choices	r=x_choices(titulo,items)
x_matrix	[r]=x_matrix(label,matriz-inicial)

Exemplo:

```
−− >x=input(”Forneça número de elementos?”)<return>
Forneça número de elementos?−− >2
x =
    2.
```

No exemplo anterior ao se fornecer o valor 2, ele é atribuído a variável x[18].

Exemplo:

```
−− >x=input('Forneça seu nome?','s')<return>
```

[18] Para a versão 2.7 do Scilab, a função **input** apresenta uma mensagem de *warning*, para se retirar a mesma deve-se executar o comando **funcprot(0)** na seção do Scilab ou acrescentar o comando funcprot(0) no arquivo **.scilab** para processamento automático do comando. Esse problema já foi resolvido na versão Scilab 3.0.

Forneça seu nome?−− >Maria
x =
Maria
Através da função: [H,ierr]=evstr(Z), que transforma *string* em valores numéricos, podem-se utilizar as funções x_dialog para a leitura de valores numéricos de programas. Assim,
Exemplo x_dialog/evstr:
−− >v=evstr(x_dialog('Valor ?','0.2')) <return>
que gera uma figura contendo o valor 0.2 e com as opções de entrada de **OK** ou **Cancel**. A variável v receberá o valor contido na janela se o botão selecionado for o OK, se outra forma a variável v será igual a [ ];
Exemplo de getvalue:
−− > labels=['magnitude';'frequência';'fase ']; <return>
−− > [ok,mag,freq,ph]=getvalue('Defina sinal senoidal',labels,... <return>
−− > list('vec',1,'vec',1,'vec',1),['0.85';'10^2';'%pi/3']) <return>
ph =
1.0471976
freq =
100.
mag =
.85
ok =
T
O exemplo acima gera a janela conforme a Figura 10:

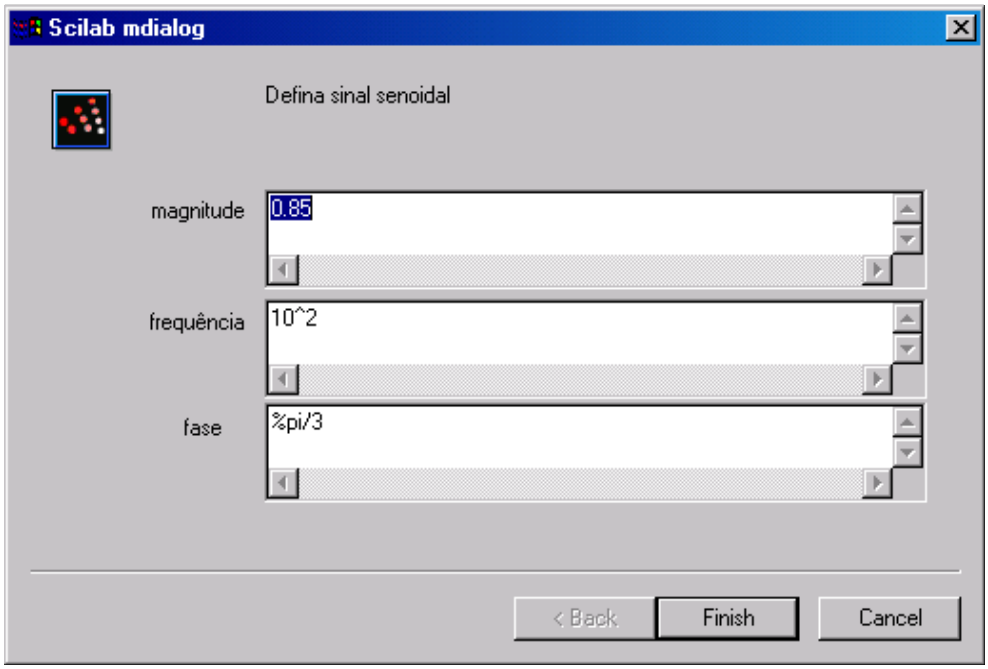

Figura 10: Exemplo de comando **getvalue**

Exemplo de x_mdialog/evstr:
−− >txt=['magnitude';'frequência';'fase '];<return>
−− >s=x_mdialog('Forneça sinal senoidal',txt,['1';'10';'0']) <return>
−− >mag=evstr(s(1));<return>
−− >freq=evstr(s(2));<return>
−− >fase=evstr(s(3));<return>
Exemplo de x_message:
−− >r=x_message(['Sistema Quadrático'; <return>
−− > 'Continua?'],['Sim','Não'])<return>
Exemplo de x_choose:
−− >n=x_choose(['item1';'item2';'item3'],['that is a comment'],'Return') <return>
Exemplo de x_choices:
−− >l1=list('Opção 1',1,['c1','c2','c3']);<return>
−− >l2=list('Opção 2',2,['d1','d2','d3']); <return>
−− >l3=list('Opção 3',3,['e1','e2']); <return>
−− >r=x_choices('Menu de Escolha',list(l1,l2,l3)); <return>
Que retornará o vetor r com o item de escolha das opções, Ex:
r =
! 1. 2. 2.!
Exemplo de x_matrix/evstr:
−− >m=evstr(x_matrix('Forneça matriz 3x3',zeros(3,3))) <return>
No exemplo acima a matriz é inicializada com valores zeros através da função **zeros(3,3)**.

Leitura de arquivo
A tabela a seguir apresenta um conjunto de funções essenciais para a leitura de dados de arquivos:

Sintaxe	Função
[fd,err]=**mopen**(file [, mode, swap])	abre arquivo, compatível com fopen da linguagem C
err=**mclose**([fd]) ou **mclose**('all')	fecha arquivo aberto com mopen
read(arquivo,m,n,[formato])	leitura de matriz de dados linha após linha
fscanf(arquivo,formato)	leitura de dados de arquivo

3.4.3 Operações de saída/escrita de dados

O Scilab possui várias formas de apresentar resultados na janela de comandos. O forma mais simples é a execução de comando sem o ponto-e-vírgula (;). Além dessas formas pode-se apresentar resultados com as funções:

Comando	finalidade
;	inibe apresentação de resultado após execução de instrução
disp	mostra objeto na tela do Scilab
file	seleciona unidade lógica e gerencia arquivo
write(%io(2),....)	escreve resultados formatados na janela do Scilab
print(%io(2),....)	escreve resultados formatados na janela do Scilab
printf(formato,valor1,..,valorn)	emula a função **printf** da linguagem C
mprintf(formato,a1,...,an);	converte, formata e escreve dados na janela do Scilab
mfprintf(fd,formato,a1,...,an);	converte, formata e escreve dados para arquivo
str=**msprintf**(formato,a1,...,an);	converte, formata e escreve dados em uma *string*

As funções **mprintf** e **msprintf** são interfaces para as funções printf e sprintf da linguagem C.

Escrita em arquivo

Comando	finalidade
file	seleciona unidade lógica e gerencia arquivo
write(fd,....)	escreve resultados formatados na janela do Scilab
print(fd,....)	escreve resultados formatados na janela do Scilab
fprintf(fd,formato,a1,...,an);	converte, formata e escreve dados para arquivo
mfprintf(fd,formato,a1,...,an);	converte, formata e escreve dados para arquivo

A função **mfprintf** é uma interface para a função fprintf da linguagem C. A string de *status* do comando **file** possui as seguintes opções:

string	significado
"new"	arquivo não pode existir (*default*)
"old"	arquivo deve existir
"unknown"	*status* desconhecido
"scratch"	arquivo a ser apagado no final da seção

As ações possíveis seguem as atribuições da *string* correspondente:

string	significado
"close"	fecha o(s) arquivo(s) representado(s) pela unidade lógica associada
"rewind"	coloca o ponteiro para o início do arquivo
"backspace"	coloca o ponteiro para o início do último registro
"last"	coloca o ponteiro após o último registro

O Scilab possui dois tipos de acesso a um arquivo:

acesso	significado
"sequential"	acesso seqüencial (*default*)
"direct"	acesso direto

Saída em forma gráfica
O Scilab possibilita apresentar resultados em vários formatos de gráficos, imprimir, formatar, salvar figuras em vários formatos gráficos (Xfig, GIF, PS, EPS). As funções gráficas básicas na Scilab[19] estão apresentadas na Tabela 3.

Exemplos 1
−− >x=0:0.1:2*%pi; <return>
−− >plot(x,sin(x),'seno','tempo','faz gráfico de seno')<return>
−− >xbasc()<return>
−− >plot([sin(x);cos(x)]) <return>
−− >// Faz campo de direções de ODE

[19]Existe uma biblioteca de funções gráficas, PLOTLIB, disponivel em:
http://www.engineering.usu.edu/cee/faculty/gurro/plotlib.html, com várias funções similares as funções do MATLAB®. Entre elas, destacam-se:

```
-- >deff("[xdot] = derpol(t,x)",..
-- >['xd1 =x(2)';..
-- > 'xd2 = -x(1) + (1 - x(1)**2)*x(2)';..
-- >'xdot = [ xd1 ; xd2 ]'])
-- >xf= -1:0.1:1;
-- >yf= -1:0.1:1;
-- >fchamp(derpol,0,xf,yf)
```

Exemplos 2

```
-- >deff('[y]=f(x)','y=sin(x)+cos(x)')
-- >x=[0:0.1:10]*%pi/10;
-- >fplot2d(x,f)
```

Exemplos 3

```
-- >// cria figura número 3
-- >h=figure(3);
-- >// cria figura número 1 e coloca texto
-- >uicontrol( h, 'style','text', ...
-- > 'string','Isto é uma figura', ...
-- > 'position',[50 70 100 100], ...
-- > 'fontsize',15);
-- >// Cria figura 1 e coloca texto
-- >figure();
-- >uicontrol( 'style','text', ...
-- > 'string','Outra figura', ...
```

plot(y)	Apresenta gráfico do vetor y contra seus índices
plot(x,y)	Apresenta gráfico de valores do vetor x e y, ambos do mesmo comprimento
plot(x,y,'+')	Apresenta gráfico de x contra y usando + como símbolo
plot(x,y,'o')	Apresenta gráfico de x contra y usando o como símbolo
plot(x,y,'-')	Apresenta gráfico de x contra y usando linha contínua
plot(x,y,'–')	Apresenta gráfico de x contra y usando linha tracejada (*dashed*)
plot(x,y,'-.')	Apresenta gráfico de x contra y usando linha tracejada-ponto (*dash-dot*)
hold()	Mantém ou libera janela gráfica ativa para gráficos adicionais
semilogx(x,y)	Gráfico com escala logarítmica no eixo-x
semilogy(x,y)	Gráfico com escala logarítmica no eixo-y
loglog(x,y)	Gráfico com escala logarítmica em ambos eixos
plot(x1,y1,x2,y2,x3,y3)	Gráfico de conjunto de dados (x1,y1), (x2,y2) etc.
semilogx(x1,y1,x2,y2,x3,y3)	Gráfico de conjunto de dados com escala logarítmica no eixo-x
semilogy(x1,y1,x2,y2,x3,y3)	Gráfico de conjunto de dados com escala logarítmica no eixo-y
loglog(x1,y1,x2,y2,x3,y3)	Gráfico de conjunto de dados com escala logarítmica em ambos eixos
legend('leg1','leg2','leg3')	Apresenta caixa de legendas
subplot(m,n,k)	Define janela matricial e seleciona gráfico número k na janela de m linhas e n colunas
plot3(x,y,z)	Apresenta gráfico 3D dos vetores x,y,z com coordenadas de pontos
z=feval(x,y)	Gera matriz de z(i,j) = f(x(i),y(j)), x,y são vetores
mesh(x,y,z')	Superfície (*mesh*) definida pelos vetores x,y e matriz z
mesh(x,y,f)	Superfície (*mesh*) definida pelos vetores x,y e função f [f(x,y)]
surf(x,y,z')	Superfície sólida definida pelos vetores x,y e matriz z
surf(x,y,f)	Superfície sólida definida pelos vetores x,y e função f [f(x,y)]
surfl(x,y,z')	Superfície com sombra (*lightened*) definida pelos vetores x,y e matriz z
surfl(x,y,f)	Superfície com sombra (*lightened*) definida pelos vetores x,y e função f [f(x,y)]

−− > 'position',[50 70 100 100], ...
−− > 'fontsize',15);
−− >// fecha janela gráfica corrente (fig. 1)
−− >close();
−− >// fecha figura 3
−− >close(h);

Exemplos 4

−− >// cria figura
−− >f=figure('position', [10 10 300 200]);
−− >// cria item na barra de menu
−− >m=uimenu(f,'label', 'Janela');
−− >//cria dois ítens no menu "janela"
−− >m1=uimenu(m,'label', 'Operações');
−− >m2=uimenu(m,'label', 'Sai Scilab', 'callback', "exit");
−− >// cria submenu para o item "operações"
−− >m11=uimenu(m1,'label', 'Nova janela', 'callback',"xselect()");
−− >m12=uimenu(m1,'label', 'Limpa janela', 'callback',"xbasc()");
−− >// fecha figura
−− >close(f);

Os comandos de gráficos possuem um argumento que classifica o estilo das curvas quanto a forma (contínua ou discreta) e cor da curva. Para se ter acesso à janela de configuração basta usar as funções: −− > `xset()`

Que apresenta janela para configuração de aspectos gráficos, ou,

−− > `getcolor()`

O argumento de estilo pode ser fornecido diretamente na chamada da função. Para exemplificar:

−− >plot2d(x,y,arg)

onde arg é um vetor real de tamanho igual a número de curvas, com default dado por [1:ncurvas], (1 para primeira curva, 2 para a segunda e assim sucessivamente). Se arg é negativo ou zero a curva usa forma discreta e se positivo é a curva continua conforme mapa de cores.

argumento	atributo	argumento	atributo
-9	círculo vazio, ∘	1	curva preta
-8	sinal, ♣	2	curva azul
-7	triângulo, ▽	3	curva verde claro
-6	sinal, ⊗	4	curva azul claro
-5	losango vazio, ◊	5	curva vermelha tonalidade 1
-4	losango preenchido, ♦	6	curva magenta
-3	asterisco, *	7	curva amarela tonalidade 2■
-2	sinal, x	8	curva branca
-1	sinal, +	9	curva azul escuro
0	pontilhado, .	⋮	⋮
		32	curva dourada

■Algumas versões anteriores do Scilab apresentam essa cor como vermelha.

Observações:

1. Para se fazer uma curva com uma cor diferente da cor preta, deve-se mudar a cor padrão da paleta de cores do Scilab usando o comando **xset("color",valor)**, porém, para se mudar apenas a cor dos pontos pode ser usar as funções **get**() e **set**().
   Exemplo: Deseja-se fazer uma curva com os losangos preenchidos na cor magenta (cor=6).
   `-->clf()// limpa janela gráfica`
   `-->x=[-.2:0.1:2*%pi]'; // cria vetor x`
   `-->plot2d(x-.3,sin(x-1),[6] ); // Faz gráfico`
   `-->a=get('current_axes'); // Ler dados do eixo`
   `-->p1=a.children.children(1);`
   `-->set(p1,'mark_style',4);`

2. Para se exportar uma figura em formato para inclusão em Latex, basta proceder conforme indicado na Figura (11).

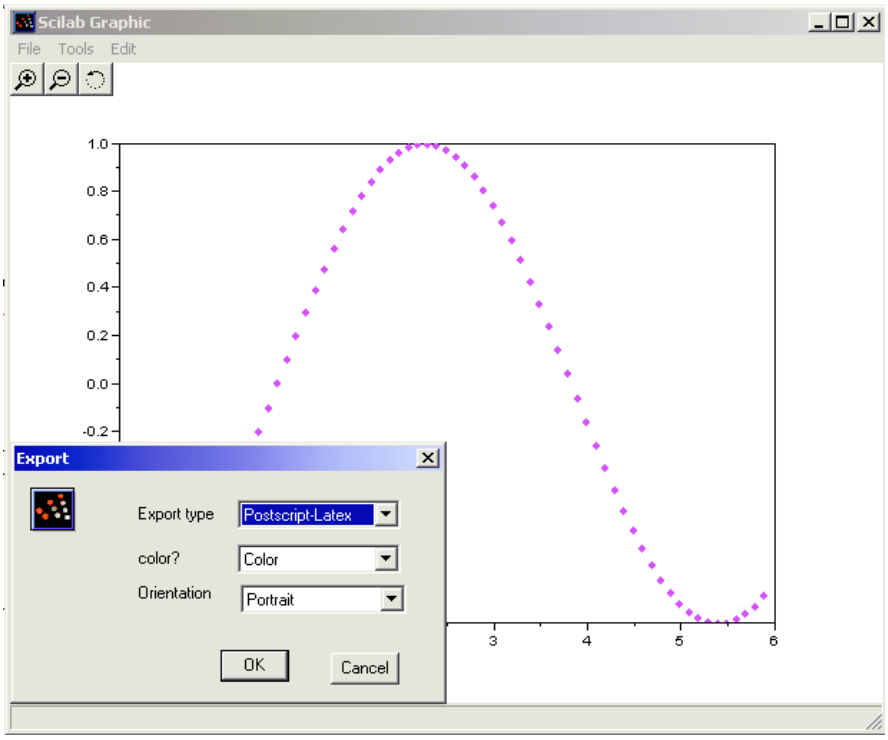

Figura 11: Geração de figura para inclusão em arquivos Latex.

3. A configuração padrão do modo gráfico no Scilab não encontra-se com limpeza automática. Assim, caso não queira fazer gráficos superpostos deve-se abrir outra janela gráfica com o comando **xset**, (`xset('window',numerojanela)`), ou proceder a limpeza da janela ativa com os comandos **xbasc**() ou **clf**().

4. A utilização das funções do novo estilo gráfico flexibilizam e facilitam a especificação dos gráficos necessários. As funções **get**, **gda**() e **set** possuem muitos recursos. O *script* a seguir apresenta aa demonstração de alguns recursos para as funções gráficas (vide Figura (12)).

```
clf()
set('figure_style','new') //cria uma figura
a=get('current_axes')//recebe manipulador (handler) do novo eixo criado
a.axes_visible='on'; // torna os eixos visíveis
a.font_size=3; //configura o tamanho da fonte para os labels dos tics
a.x_location='top'; //configura a posição do eixo x
a.data_bounds=[-7,-2;7,2]; //configura os tamanhos limites dos eixos x, y
a.sub_tics=[5,0];
a.labels_font_color=5;
a.grid=[2,2];
a.box='off';
x=-2*%pi:0.1:2*%pi;
plot2d(x-.3,sin(x)*7+.2,16);
da=gda(); // recebe manipulador do modelo do eixo
da.thickness = 2; // Espessura da linha dos eixos
da.foreground = 6; // cor das linhas dos eixos
// título
da.title.text='Meu Titulo@Principal' // Título em múltiplas linhas
da.title.foreground = 12;
da.title.font_size = 4;
// x labels default
da.x_label.text='x';
da.x_label.font_style = 8;
da.x_label.font_size = 2;
da.x_label.foreground = 5;
da.x_location = 'middle';
// y labels default
da.y_label.text='y';
da.y_label.font_style = 3;
da.y_label.font_size = 5;
da.y_label.foreground = 3;
da.y_location = 'right';
```

Para que se entenda o potencial dessas funções apresenta-se a seguir as propriedades padrões de um eixo. E que são portanto factíveis a configuração.

```
-->set('figure_style','new')
-->a=get('current_axes')
a=
Handle of type 'Axes' with properties:
======================================
parent: Figure
children: []
visible = 'on'
axes_visible = 'off'
grid = [-1,-1]
x_location = 'bottom'
y_location = 'left'
title: 'Label'
```

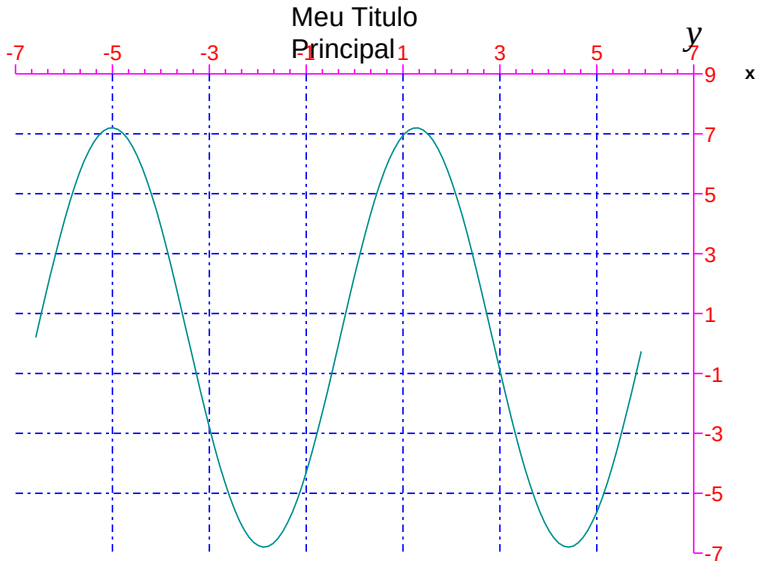

Figura 12: Exemplo de figura usando o estilo gráfico novo.

```
x_label: 'Label'
y_label: 'Label'
z_label: 'Label'
box = 'on'
sub_tics = [1,1]
tics_color = -1
font_style = 6
font_size = 1
font_color = -1
isoview = 'off'
cube_scaling = 'off'
view = '2d'
rotation_angles = [0,270]
log_flags = 'nn'
tight_limits = 'off'
data_bounds = [0,0;1,1]
margins = [0.125,0.125,0.125,0.125]
axes_bounds = [0,0,1,1]
auto_clear = 'off'
auto_scale = 'on'
hiddencolor = 4
line_style = 0
thickness = 1
mark_mode = 'off'
mark_style = 0
mark_size = 1
```

```
background = -2
foreground = -1
clip_state = 'off'
clip_box = []
```

Comando	utilização
driver	seleciona driver gráfico
xclear	limpa janela gráfica
xpause	pausa em milisegundos
xbasc	limpa janela gráfica
clf	limpa janela gráfica
xclick	aguarda click de mouse
xbasr	refaz gráfico de janela gráfica
xinit	inicializa *device* gráfico (arquivo)
xend	encerra seção gráfica
xset	define propriedades de gráfico
xget	recebe propriedades do gráfico corrente
plot2d	faz gráfico linear por partes(*)
plot2d2	faz gráfico constante por parte (degrau)(*)
plot2d3	faz gráfico de barras verticais(*)
plot2d4	faz gráfico com setas (flechas)(*)
subplot	divide janela gráfica em matriz de sub-janelas
xtitle	adiciona título à janela gráfica e eixos X e Y
xgrid	adiciona *grid* em gráfico 2D
champ	faz gráfico de campo vetorial 2D
fchamp	faz campo de direções de uma ODE 2D de 1a. ordem
xsave	salva gráfico para arquivo
xload	leitura de gráfico salvo com **xsave**
figure	cria pagina gráfica tksci
close	fecha janela gráfica tksci
uicontrol	cria objeto GUI
uimenu	cria menu ou submenu em figura
xselect	seleciona (se existente) ou cria janela gráfica

(*)Para verificar demonstrativo execute a função sem argumentos, Ex.: **plot2d3()**.

Tabela 3: Funções gráficas básicas do Scilab

3.4.4 Operações Simbólicas no Scilab

O Scilab não é um ambiente simbólico, porém possui algumas funções para manipulação de expressões simbólicas. Entre as funções mais usuais destacam-se:

addf	adição simbólica
subf	subtração simbólica
mulf	multiplicação simbólica
ldivf	divisão simbólica à esquerda
rdivf	divisão simbólica à direita
cmb_lin	com 4 argumentos, a, x, b e y, cria combinação linear $a * x + b * y$
eval	avalia expressão simbólica
evstr	avalia expressão simbólica
trianfml	produz matriz triangular superior de uma matriz simbólica
solve	obtém solução de sistema simbólico $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A}$ é matriz triangular superior

Para resolver simbolicamente o sistema linear $\mathbf{Ax} = \mathbf{b}$ em que a matriz $\mathbf{A}$ não é uma matriz triangular superior pode-se seguir o seguinte procedimento:

1. Cria-se uma matriz aumentada $\mathbf{Aaum} = [\mathbf{A}|\mathbf{b}]$
2. Usa-se a função **trainfml** na matriz aumentada para se produzir uma matriz triangular superior, **Aaum2**
3. Extrai-se uma matriz triangular superior com as mesmas dimensões de $\mathbf{A}$ de **Aaum2**, **A1**
4. Faça a última coluna em **Aaum2** igual a um vetor **b1**
5. A solução do sistema $\mathbf{Ax} = \mathbf{b}$ é obtida utilizando-se a função **solve** com argumentos **A1** e **b1**

Além dessas funções o Scilab possui um conjunto de funções para manipulação simbólica de expressões polinomiais, conforme apresentado no Apêndice.

clean	limpa matrizes (arredonda para zero valores muito pequenos)
coeff	coeficientes de matriz polinomial
coffg	inversa de matriz polinomial
degree	grau de matriz polinomial
denom	denominador
derivat	derivada de matriz racional
determ	determinante de matriz polinomial
detr	determinante polinomial
factors	fatoração numérica real
horner	avaliação de polinômio ou matriz racional
invr	inversão de matriz (racional)
numer	numerador
pdiv	divisão polinomial
residu	resíduo
roots	raízes de polinômios

3.5 Programação

O Scilab possui uma série de ferramentas para desenvolvimento de programas incluindo operadores de comparação, repetições condicionais, funções e criação de novos ambientes.

3.5.1 Funções

Funções são coleções de comandos que podem ser criadas e executadas de várias formas no Scilab. Elas podem passar argumentos e podem ser elementos em listas. O formato geral de uma função é:

```
function [y1,y2,...,yn]=nome_func(x1,x2,...,xm)
          .........
endfunction
```

onde x_i são os m argumentos de entrada para a função nome_func e y_j são os argumentos de saída. Os parâmetros de entrada e saída podem ser qualquer objeto Scilab, inclusive funções. Cada função deve ser finalizada com o comando **endfunction**. Quando as funções são definidas em arquivo como *scripts*, elas devem ser carregadas no Scilab através dos comandos **getf** (carrega função na memória) e **exec** (carrega para a memória e executa). A boa prática de programação em Scilab é usar as extensões:

extensão	uso
.sce	arquivo para execução direta
.sci	arquivo de funções para carregar em memória

No caso de arquivos, cada um pode conter mais de um função. A forma mais útil de se criar funções é através de um editor, porém elas podem ser criadas diretamente no ambiente do Scilab conforme indicado a seguir:

Exemplo

```
--> function [x]=teste(y) <return>
-->     if y>0 then, <return>
-->         x=1; <return>
-->     else, <return>
-->         x=-1; <return>
-->     end <return>
--> endfunction <return>
```

ou, pode-se utilizar a definição em linha

```
--> deff('[x]=teste(y)','if y>0 then, x=1; else, x=-1; end') <return>
```

A execução em ambos os casos da-se conforme o exemplo abaixo:

```
--> a=teste(5) <return>
a=
    1.
```

A tabela abaixo apresenta alguns comandos úteis para o desenvolvimento de funções:

Comando	utilidade para programação
global	definição de variável global
argn	número de argumentos de uma função
error	imprime *string* em uma mensagem de erro e para a instrução corrente
warning	imprime *string* em uma mensagem de atenção
pause	muda para o modo pause. Útil para depuração de programa
break	interrompe instrução de repetição
return ou **resume**	retorna ou resume execução

Exemplo de leitura de matriz de arquivo texto:
Esse exemplo usa o comando **file** para abrir um arquivo para leitura, e a função **read** para a leitura de uma matriz de um arquivo texto. Essa é uma forma adequada de fornecer uma tabela ao Scilab. Para efeito de exemplificação, suponha que a tabela encontra-se no arquivo `c:\tabela1.dat` e possui os seguintes dados:

1.21	3.14	4.56
2.13	3.56	7.89
10.0	0.23	5.43
2.98	8.41	6.23

O comando **file** usa três argumentos: o primeiro é a opção "open" para indicar que o arquivo está sendo aberto, o segundo argumento é o **nome** do arquivo e o terceiro é um qualificado que indica que o arquivo já existe (necessário para arquivos com dados para leitura). Do lado esquerdo da função tem-se a variável **u** que receberá um valor inteiro representando a unidade lógica de entrada-saída atribuída para o arquivo **nome**. O comando **read** possui um segundo argumento, -1, que permite não se conhecer o número de linhas a ser lida e o Scilab lerá todas as linhas da matriz. O número de colunas deverá ser conhecido, no exemplo é dado por 3. A função a seguir lê a tabela do arquivo `c:\tabela1.dat` e separa as 3 colunas em um número de variáveis x1, x2, x3:

```
function [x1,x2,x3] = le_tabela()
printf(‘‘ Lendo tabela de arquivo texto \n’’)
printf(‘‘==============================\n’’)
printf(‘‘  ’’)
nome = input(‘‘Entre nome do arquivo entre aspas:\n’’)
u = file(‘‘open’’,nome,‘‘old’’)
Table = read(u,-1,3)
[n m] = size(Table)
printf(‘‘ ’’)
printf(‘‘Existem %d colunas na tabela’’,m)
for j = 1:m
   execstr(‘‘x’’+string(j)+‘‘ = Table(:,j) ’’)
end
close(u)
endfunction
```

Existem várias formas alternativas de se processar a leitura acima, por. ex.:

```
function [x1,x2,x3] = le_tabela2()
printf(‘‘ Lendo tabela de arquivo texto \n’’)
printf(‘‘==============================\n’’)
printf(‘‘  ’’)
nome = xgetfile()
u = file(‘‘open’’,nome,‘‘old’’)
x1=[]; x2=[]; x3=[];
for j = 1:4
   [x y z] = fscanf(u,’%f %f %f’);
    x1 = [x1;x];
    x2 = [x2;y];
    x3 = [x3;z];
end
close(u)
endfunction
```

Observe que nesse caso precisa-se saber também quantas linhas devem ser lidas do arquivo. Ou, simplesmente, pode-se utilizar o comando **fscanfMat** para a leitura da matriz, ou seja:
−− > `X = fscanfMat(‘‘c:\tabela1.dat’’)`
O comando correspondente para a escrita de uma matriz é dado pelo comando **fprintfMat**:

Exemplo de escrita de matriz em arquivo texto:

```
−− >A = [ 1 2 3; 4 5 6];
−− >arq = ’c:\matriz.txt’;
−− >u = file(’open’,arq,’new’);
−− >fprintfMat(arq,A,’%10.6f’)
−− >file(’close’,u)
```

4 Utilizando o SCILAB na Engenharia Química

Nessa seção apresenta-se a utilização o Scilab para o estudo de problemas da Engenharia Química dentro das seguintes categorias:

1. Sistemas de Equações Algébricas Lineares
2. Problemas de Valor Característico
3. Sistemas de Equações Algébricas Não Lineares
4. Sistemas de Equações Diferencias Ordinárias: PVI e PVC
5. Otimização
6. Solução de equações algébrico-diferenciais
7. Solução de Equações Diferenciais Parciais (EDPs) por diferenças finitas

4.1 Sistemas de Equações Algébricas Lineares

Como ilustração, seja um sistema de equações algébricas lineares dado por:

$$\begin{cases} 2x + 3y - 5z = -7 \\ 6x - 2y + z = 5 \\ x + 3y - z = 4 \end{cases}$$

Esse sistema pode ser escrito na forma matricial como $\mathbf{A} * \mathbf{w} = \mathbf{b}$, com a matriz $\mathbf{A}$ definida por:

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

A =

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

e o vetor $\mathbf{b}$ dado por:

```
-- >b=[-7;5;4]
```

b =

```
!  - 7. !
!    5. !
!    4. !
```

A solução do sistema é o vetor coluna $\mathbf{w} = [x; y; z]$, que pode ser calculado nas formas abaixo:

1. Através de $w = A^{-1} * b$, função inv

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

w =

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

Para se verificar a solução pode-se usar:

```
-- >A*w
```

ans =

```
!  - 7.!
!    5.!
!    4.!
```

que é igual ao vetor $\mathbf{b}$. Assim a solução está correta.

2. Através do operador de divisão à esquerda (\)

```
-- >w = A\b
```

w =

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

3. Utilizando-se a função linsolve

A função **linsolve** calcula todas as soluções do problema $\mathbf{A} * \mathbf{x} + \mathbf{c} = \mathbf{0}$. Observe que o vetor c é igual ao vetor $-\mathbf{b}$. Assim, pode-se resolver o problema acima fazendo-se,

```
-- >w=linsolve(A,-b)
```

w =

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

A função **linsolve** possui outros argumentos, a sintaxe completa da função é dada por:

[x0,kerA]=linsolve(A,c [,x0])

Nesse caso:

x0	solução particular do sistema (se existir)
kerA	*nullspace* de **A**.

Qualquer $x = x0 + kerA * q$ com q arbitrário satisfaz $A * x + c = 0$. Se x0 compatível é fornecido na entrada, x0 é retornado, senão um x0 compatível, se existir, será retornado.

4. Utilizando-se da Eliminação de Gauss-Jordan, *row-reduced echelon form*

A determinação da solução do sistema através da Eliminação de Gauss-Jordan utiliza a função **rref** que aplica a decomposição LU à esquerda. A função **rref** possui sintaxe: $R = rref(A)$. Assim, para se resolver o sistema deve-se gerar a matriz aumentada $Aum = [A|b]$ e aplicar-se a função **rref**,

```
-->Aum = [A b]
```
Aum =
```
! 2.    3.   - 5.   - 7. !
! 6.   - 2.    1.    5. !
! 1.    3.   - 1.    4. !
```
```
-->rref(Aum)
```
ans =
```
! 1.   0.   0.   1. !
! 0.   1.   0.   2. !
! 0.   0.   1.   3. !
```
Essa matriz fornece a solução de x,y e z na quarta coluna.

4.2 Problemas de Valor Característico

O estudo do problema de valor característico, e conseqüentemente, dos valores característicos (autovalores) e vetores característicos (autovetores), é de grande aplicabilidade na análise de sistemas da Engenharia Química. O Scilab baseia-se nas rotinas do Lapack para essas determinações.

Sejam os exemplos ilustrativos abaixo em que deseja-se estudar as características de estabilidade do sistema representado pelas equações:

$$\frac{d\mathbf{x}}{dt} = \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \mathbf{x}$$

com as condições iniciais: $\mathbf{x}(0) = \mathbf{x}_o$. Esse estudo pode ser feito análise do problema de valor característico associado, assim determinando-se os valores característicos λ do problema associado e pode-se, pela Teoria de Lyapunov, avaliar a estabilidade do sistema linear conforme o sinal da parte real dos mesmos:

$\Re(\forall\lambda) < 0$ sistema estável

$\Re(\forall\lambda) > 0$ sistema instável
Esse estudo pode ser feito de várias formas:

1. Utilizando-se a função spec
A função **spec** possui as seguintes sintaxes:

```
evals=spec(A)
[X,diagevals]=spec(A)
evals=spec(A,E)
[al,be]=spec(A,E)
[al,be,Z]=spec(A,E)
[al,be]=spec(A,E)
[al,be,Q,Z]=spec(A,E)
```

Com os parâmetros:

A	matriz quadrada real ou complexa
E	matriz quadrada real ou complexa com mesma dimensão de **A**
evals	vetor de valores característicos (autovalores)
diagevals	matriz diagonal com valores característicos na diagonal
al	vetor, al./be fornece os valores característicos
be	vetor, al./be fornece os valores característicos
X	matriz quadrada inversível, matriz dos vetores característicos
Q	matriz quadrada inversível, *pencil left eigenvectors*
Z	matriz quadrada inversível, *pencil right eigenvectors*

Utilização:

`evals=spec(A)`	retorna em vetor evals os valores característicos de **A**
`[evals,X] =spec(A)`	retorna os valores (evals) e vetores (X) característicos de **A**
`evals=spec(A,E)`	retorna espectro de $s\mathbf{E} - \mathbf{A}$, raízes de $s\mathbf{E} - \mathbf{A}$
`[al,be] = spec(A,E)`	retorna espectro de $s\mathbf{E} - \mathbf{A}$(*)
`[al,be,Z] = spec(A,E)`	retorna al, be e a matriz Z
`[al,be,Q,Z] = spec(A,E)`	retorna al, be, Q e a matriz Z

(*)Os valores característicos são dados por al./be. Para E = eye(A), têm-se que al./be é spec(A)

2. Utilizando-se as funções para cálculo simbólico
A determinação dos valores característicos do problema de valor característico: $\mathbf{Ax} = \lambda\mathbf{x}$, exige-se que se encontre as raízes da equação característica: $det(\mathbf{A} - \lambda\mathbf{I}) = det(\lambda\mathbf{I} - \mathbf{A}) = 0$.

3. Utilizando-se a função bdiag
A função **bdiag** implementa uma diagonalização em bloco da matriz **A**, a sua sintaxe é dada por:

```
[Ab [,X [,bs]]]=bdiag(A [,rmax])
```

Com os parâmetros:

A	matriz quadrada
rmax	controla o condicionamento de X, *default* é norma l_1 de **A**(*)
Ab	matriz quadrada
bs	fornece a estrutura de blocos (dimensão dos blocos)
X	é a mudança de base, matriz não singular

(*) Para se obter uma forma diagonal (se existente) escolhe-se elevados valores para rmax, ex. `rmax=1/%eps`.

A função **bdiag** pode ser utilizada diretamente para a determinação dos valores característicos de sistemas que possuem valores característicos reais.

Exemplo: Usando spec

```
-- >A=[1,2;-2,1]; <return>
-- >spec(A) <return>
```

ans =

$! 1. + 2.i !$

$! 1. - 2.i !$

Assim, como $\Re(\forall\lambda) > 0$ o sistema é instável.

Exemplo: Usando funções simbólicas

```
-- >x=poly(0,'x'); <return>
-- >pol=det(x*eye()-A); <return>
-- >roots(pol) <return>
```

ans =

$! 1. + 2.i !$

$! 1. - 2.i !$

o que conforma que o sistema é instável.

Exemplo: Usando a função bdiag

```
-- >A=[1 2 3; 3 2 1; 2 1 3];
-- >spec(A)
```

ans =

$!$ 6. $!$

$! -1.4142136 !$

$!$ 1.4142136 $!$

```
-- >[S,X]=bdiag(A);
-- >clean(inv(X)*A*X)
```

ans =

$! -1.4142136$ 0. 0. $!$

$!$ 0. 6. 0. $!$

$!$ 0. 0. 1.4142136 $!$

Com os valores característicos da matriz **A** na diagonal.

4.3 Sistemas de Equações Algébricas Não Lineares

A resolução de sistemas de equações algébrica não lineares pode ser feita através da utilização da função **fsolve**. A função **fsolve** utiliza uma modificação do método híbrido de powell e a fornecimento do jacobiano é opcional. A sintaxe da função **fsolve** é dada por:

[x [,v [,info]]]=**fsolve**(x0,fct [,fjac] [,tol])

Os termos entre colchetes são opcionais. A tabela abaixo descreve os parâmetros sa função **fsolve**.

parâmetros		descrição
x0		vetor de estimativa inicial
fct		função ou lista de *string* (external)
fjac		função da matriz jacobiana ou lista de *string* (external)
tol		tolerância, *default*: tol=1.e-10
x		vetor com valor final estimado
v		vetor com valor da função em x
info		indicador de término
	0	parâmetros de entrada inadequados
	1	erro relativo é no máximo igual a tol
	2	número de chamadas a função excedido
	3	tol é muito pequeno. Não se pode melhorar solução
	4	interação não leva a convergência

Exemplo 1
−− >a=[1,7;2,8];b=[10;11];
−− >deff('[y]=fsol1(x)','y=a*x+b');
−− >deff('[y]=fsolj1(x)','y=a');
−− >[xres]=fsolve([100;100],fsol1,fsolj1,1.e-7);
−− >//verificação da solução
−− >xres
−− >a*xres+b

Exemplo 2
Para a solução do sistema:

$$\begin{cases} f_1(x_1,x_2) = \frac{1}{2}\sin(x_1x_2) - \frac{x_2}{4\pi} - \frac{x_1}{2} = 0 \\ f_2(x_1,x_2) = (1-\frac{1}{4\pi})(e^{2x_1}-e) + \frac{sx_2}{\pi} - 2ex_1 = 0 \end{cases}$$

Para a solução do sistema pode-se preparar o **script** abaixo e salvá-lo no arquivo **nssim.sci**:

```
function [f]=fun(x)
  f(1)=1/2*sin(x(1)*x(2))-x(2)/4/%pi-x(1)/2;
  f(2)=(1-1/4/%pi)*(exp(2*x(1))-%e)+%e*x(2)/%pi-2*%e*x(1);
endfunction
```

Para a execução da função **fsolve** pode-se fazer:
−− >[xres,v,info]=fsolve([0.4;3],fun)

info =
1.
v =
1.0E-15 *
! − .6383782!
! − .4440892!
xres =
!.2994487!
!2.8369278!
ou,
−− >[xres]=fsolve([-0.2;0.6],fun)
xres =
! − .2605993!
!.6225309!
mudando a estimativa inicial para a solução:
−− >[xres]=fsolve([0.6;3],fun)
xres =
!.5!
!3.1415927!
Observa-se que o problema possui várias soluções (multiplicidade de soluções).

Exemplo 3

Solução do sistema de reação na forma adimensionalizada:

$$\begin{cases} f_1(x_1, x_2) = 1 - x_1\left(1 + \theta \exp(\frac{-\Delta E}{x_2})\right) = 0 \\ f_2(x_1, x_2) = (1 - x_2) + \beta(\gamma_c - x_2) + \Delta h x_1 \theta \exp(\frac{-\Delta E}{x_2}) = 0 \end{cases}$$

com $\beta = 1; \gamma_c = 1, \theta = 50, \Delta E = 10, \Delta h = 10$. Para a solução do sistema pode-se preparar o **script** abaixo e salvá-lo no arquivo **nssim2.sci**:

```
function [f]=fun(x)
  B=1; g=1; t=50; de=10; dh=10;
  f(1)=1-x(1)*(1+t*exp(-de/x(2)));
  f(2)=(1-x(2))+B*(g-x(2))+dh*x(1)*t*exp(-de/x(2));
endfunction
```

Assim, executando-se:
−− > [xres]=fsolve([0.;6],fun)
xres =
!.1116045!
!5.4419775!
−− > [xres]=fsolve([1;1],fun)
xres =
!.9974295!
!1.0128525!
Execute a função para uma estimativa inicial de $[0.5; 2]$. Que conclusão pode tirar desse problema?

Observação: O comando **exec** pode ser usado também para carregar funções para a memória do Scilab, neste caso pode ser necessário a utilização do comando com a sua formatação geral, que

inclui opções:

```
ierr=exec(path,'errcatch' [,mode])
ou
ierr=exec(fun,'errcatch'[,mode])
```

com os parâmetros:

path		o caminho de um arquivo *script*.
mode		escalar que indica modo de execução.
	0	valor *default*.
	-1	nada é impresso.
	1	apresenta cada linha de comando.
	2	o *prompt* $-->$ é impresso.
	3	apresenta cada linha de comando e o *prompt*.
	4	para antes de cada *prompt*. A execução continua após <return>.
	7	útil para demonstração contempla as opções 3 e 4, simultaneamente.
fun		uma função Scilab
ierr		inteiro, 0 ou número do erro.

A forma adequada de utilização da função **fsolve** é apresentada no exemplo da Tabela 4:

Quando a função a ser resolvida necessitar de parâmetros pode-se passar esses parâmetros para a função **fsolve** através da definição de uma lista, ou seja:

```
function [f]=funcao(x,a,b,c)
     .......
endfunction
.............................
// Programa
.......
flist=list(funcao,a,b,c);
[x,fv,iflag]=fsolve(x0,flist);
```

4.3.1 Aplicações à Engenharia Química

4.3.2 Cálculo do Volume pela Equação de Estado de Redlich-Kwong

A equação de Redlich-Kwong é dada por:

$$P = \frac{RT}{V-b} - \frac{a}{V(V+b)\sqrt{T}} \tag{4.1}$$

com:

Variável	Significado	Unidade
P	pressão	atm
V	volume molar	L/g-mol
T	temperatura	K
R	Constante Universal dos gases	R=0.08206 atm L/(g-mol K)
T_c	temperatura crítica	K
P_c	pressão crítica	atm

```
mode(-1);
// Template para resolução de Funções no SCILAB
// Exemplo de solução de sistema não linear
// Formato de utilização da função fsolve:
// [x [,v [,info]]]=fsolve(x0,fct [,fjac] [,tol])
// info  : indicador de final de execução
//          =   0  : parâmetros de entrada não adequados
//          =   1  : erro relativo entre x e a solução é no máximo igual a tol
//          =   2  : número de chamadas da função foi atingido
//          =   3  : tolerância, tol, é muito pequena
//          =   4  : Não converge
// Valor default para tol é tol=1.e-10
//
// Definições das funções
function [f]=fun(x)
  f(1)=1/2*sin(x(1)*x(2))-x(2)/4/%pi-x(1)/2;
  f(2)=(1-1/4/%pi)*(exp(2*x(1))-%e)+%e*x(2)/%pi-2*%e*x(1);
endfunction

// Programa principal
txt=['x(1)';'x(2)'];
valor=x_mdialog('Forneça estimativa inicial',txt,[' ';' ']);
x0(1)=evstr(valor(1));
x0(2)=evstr(valor(2));
[x,fv, iflag]=fsolve(x0,fun);
if iflag==1 then
  printf('Solução:\n');
  disp(x);
end
```

Tabela 4: Exemplo de utilização da função **fsolve**

e,

$$a = 0.42747 \left(\frac{R^2 T_c^{5/2}}{P_c} \right) \tag{4.2}$$

$$b = 0.08664 \left(\frac{RT_c}{P_c} \right) \tag{4.3}$$

1. Calcule o volume molar e fator de compressibilidade para amônia gasosa à uma pressão de $P = 56atm$ e temperatura $T = 450K$

2. Como o fator de compressibilidade varia com a $P_r = P/P_c$ na faixa de $0,1$ até 10.

Sabe-se que $z = \frac{PV}{RT}$, $T_c = 405,5K$ e $p_c = 111.3atm$. Solução: A equação (4.1) pode ser resolvida como uma equação não linear no volume. A utilização da função **fsolve** pode ser aplicada definindo-

se:

$$f = \frac{RT}{V-b} - \frac{a}{V(V+b)\sqrt{T}} - P \tag{4.4}$$

O código abaixo apresenta a função que faz o gráfico dessa função indicando a solução do problema:

```
// Equação de Estado de RK
mode(-1);
// Definição da função
function [f]=fun(V,P,T,Tc,Pc)
  R=0.08206;
  a=0.42747*R^2*Tc^(5/2)/Pc;
  b=0.08664*R*Tc/Pc;
  f=R*T/(V-b)-a/V/(V+b)/sqrt(T)-P;
endfunction
//-----------------------------------
//       Programa principal
//-----------------------------------
// Dados
Tc=405.5; Pc=111.3; P=56; T=450;
// Estimativa para a solução
x0=1; flist=list(fun,P,T,Tc,Pc);
[x,fv,iflag]=fsolve(x0,flist);
select iflag,
case 0 then
      printf('Parâmetros de entrada não adequados!\n'), abort
case 1 then
        printf('Solução:\n');
        printf(' V  = %f\n',x);
case 2 then
      printf('Número máximo de chamadas da função atingido!\n'), abort
case 3 then
      printf('Valor da variável tol é muito pequeno!\n'), abort
case 4 then
      printf('Não converge!\n'),abort
end
```

Salvando o código no arquivo EQ_RK.sce na unidade C: pode-se executá-lo conforme indicado:

```
--> exec('C:/EQ_RK.sce');
Solução:
 V  = 0.569804
```

A comparação desse valor com aquele usado como estimativa da solução (x0=RT/P), o volume molar considerando gás ideal, indica as características do problema estudado. A resposta da segunda parte do problema exige a determinação do fator de compressibilidade para várias condições de pressão reduzida, P_r. A tabela a seguir apresenta a modificação do código acima para esse estudo.

```
// Equação de Estado de RK
clear
clc
mode(-1);
// Definição da função
function [f]=fun(V,P,T,Tc,Pc)
  R=0.08206;
  a=0.42747*R^2*Tc^(5/2)/Pc;
  b=0.08664*R*Tc/Pc;
  f=R*T/(V-b)-a/V/(V+b)/sqrt(T)-P;
endfunction
//---------------------------------
// Programa principal
//---------------------------------
// Dados
Tc=405.5; Pc=111.3; T=450;Pr=0.1:0.1:10;Pt=Pr.*Pc;sol=[];
h=figure(1);
 uicontrol( h, 'style','text',...
 'string','Favor aguardar. Calculando...', ...
 'position',[1 180 200 20], ...
 'fontsize',15);
x0=0.08206*T/Pt(1);
for i=1:length(Pt)
//Estimativa para a solução
flist=list(fun,Pt(i),T,Tc,Pc);
[x,fv,iflag]=fsolve(x0,flist);
select iflag,
 case 0 then
     printf('Parâmetros de entrada não adequados!\n'), abort
 case 1 then
       z=Pt(i)*x/0.08206/T;
       sol=[sol;x z Pr(i)];
       x0=x;
 case 2 then
     printf('Número máximo de chamadas da função atingido!\n'),abort
 case 3 then
     printf('Valor da variável tol é muito pequeno!\n'), abort
 case 4 then
     printf('Não converge para P=%f!\n',Pt(i)), abort
 end
end
close(h);
// Gráfico com algumas funções do novo estilo
xset('window',0);
clf() // limpa janela gráfica
xset('font size',12);xset('thickness',2);
xset('mark size',1);
plot2d(sol(:,3),sol(:,2),5);
a=get('current_axes'); // Ler dados do eixo
p=a.children.children;
set(p,'mark_style',5);
xtitle('z x Pr', 'Pr', 'z'); xselect();
```

A Figura apresenta o comportamento alcançado para o fator de compressibilidade variando-se a pressão relativa para a amônia[20]:

A estrutura do Scilab é também um conveniente meio para o estudo e desenvolvimento de scripts

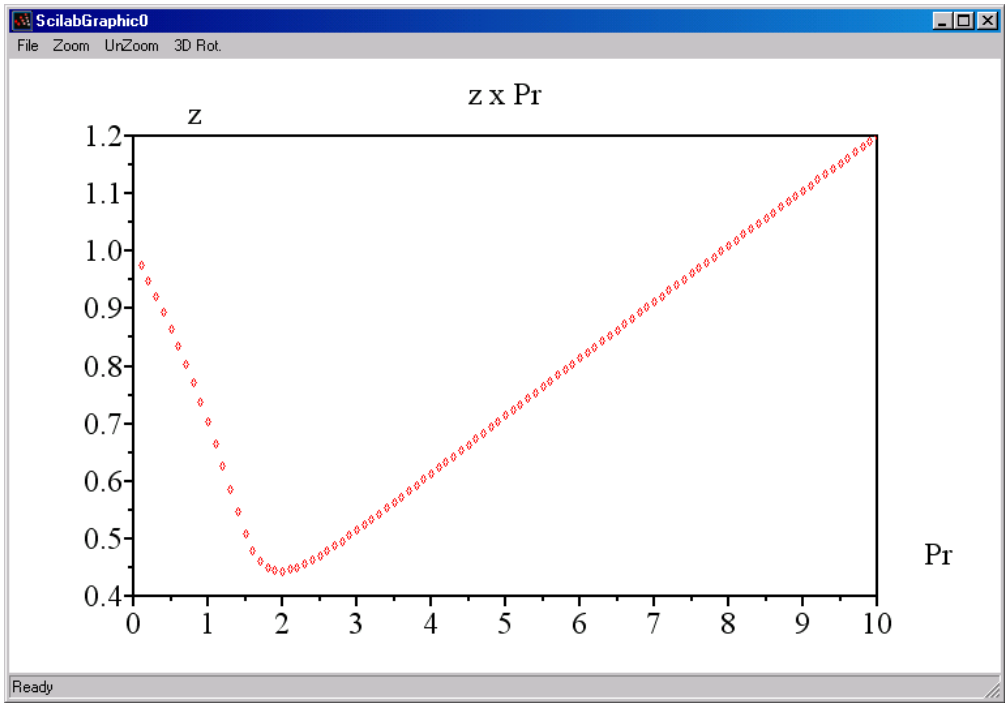

Figura 13: Comportamento do fator de compressibilidade (z) com a Pressão reduzida (P_r)

para as várias áreas dos método numéricos. Para exemplificar, apresenta-se a seguir alguns *scripts* simples para a solução de equações não lineares:

```
// Código que implementa o método da bissecção
// LCOL, Ag/2004
// UFU/FEQUI/NUCOP
// Pré-processamento
clear
clc
mode(-1)
//
function [sol,erro,fc,kit]=bisseccao(fun,a,b,delta)
//
ya=feval(a,fun);
yb=feval(b,fun);
```

[20] Para pessoas utilizando o estilo antigo para as funções gráficas, ter-se-ia:

```
// Estilo antigo para funções gráficas
xset('font size',12);xset('thickness',2);
xset('color',5); xset('mark size',1); xbasc();
plot2d(sol(:,3),sol(:,2),-5); xset('color',1);
xtitle('z x Pr', 'Pr', 'z'); xselect();
```

```
if ya*yb > 0, break, end
max1=1+round((log(b-a)-log(delta))/log(2));
kit=0;
for k=1:max1
  kit=kit+1;
  c=(b+a)/2;
  yc=feval(c,fun);
  if yc==0
    a=c;
    b=c;
  elseif yb*yc>0
    b=c;
    yb=yc;
   else
    a=c;
    ya=yc;
   end
   if b-a < delta, break, end
end
sol=(a+b)/2;
erro=abs(b-a);
fc=feval(c,fun);
endfunction
// Código que implementa o método de regula-falsi
// LCOL, Ag/2004
// UFU/FEQUI/NUCOP
function [sol,erro,fc,kit]=regula(fun,a,b,delta,epsilon,itmax)
//---------------------------------------------------------------
// sol    = solução encontrada
// erro   = estimativa de erro para a solução sol
// fc     = valor da função na solução encontrada
// a      = ponto esquerdo no intervalo onde situa-se a solução
// b      = ponto direito no intervalo onde situa-se a solução
// fun    = nome da função a ser resolvida
// delta  = tolerância para a solução
// epsilon= tolerância para valor da função no ponto da solução
// itmax  = número máximo de iterações
//---------------------------------------------------------------
ya=feval(a,fun);
yb=feval(b,fun);
if ya*yb > 0,
  disp( Intevalo [a,b] fornecido é inadequado para o método "),"
  break,
end
kit=0;
for k=1:itmax
  kit=kit+1;
  dx=yb*(b-a)/(yb-ya);
```

```
  c=b-dx;
  ac=c-a;
  yc=feval(c,fun);
  if yc==0, break;
   elseif yb*yc>0
    b=c;
    yb=yc;
   else
    a=c;
    ya=yc;
   end
   dx=min(abs(dx),ac);
   if abs(dx) < delta,   break, end
   if abs(yc) < epsilon, break, end
end
sol=c;
erro=abs(b-a)/2;
fc=feval(c,fun);
endfunction
// Método de Newton -------------------------------------
function [x,erro,kit,fc]=newton(fun,dfun,x0,delta,epsilon,itmax)
 kit=0;
 for k=1:itmax
  kit=kit+1;
  x=x0-feval(x0,fun)/feval(x0,dfun);
  erro=abs(x-x0);
  erro_rel=2*erro/(abs(x)+delta);
  x0=x;
  fc=feval(x0,fun);
  if (erro <delta) | (erro_rel < delta) | (abs(fc)<epsilon), break,end
 end
endfunction
//-----------------------------------------------------------
// Programa Principal
// Uso da função bissecção
deff('[f]=funcao(x)','f=x*sin(x)-1');
[sol,erro,fc,it]=bisseccao(funcao,0,2,1e-7);
printf('    Solução pelo Método da bissecção de Bolzano:\n');
printf(' x = %f\n',sol);
printf(' Erro = %e\n',erro);
printf(' Função = %e\n',fc);
printf(' Iterações = %d\n',it);
// Uso da função regula
[sol,erro,fc,it]=regula(funcao,0,2,1e-7,1e-8,100);
printf('    Solução pelo Método Regula-Falsi: \n');
printf(' x = %f\n',sol);
printf(' Erro = %e\n',erro);
printf(' Função = %e\n',fc);
```

```
printf(' Iterações = %d\n',it);
// Uso da função newton
deff('[f]=funcao2(x)','f=x^3-3*x+2');
deff('[df]=deriva(x)','df=3*x^2-3');
x0=1.2;
delta=1e-8;
epsilon=1e-10;
itmax=40;
[x,erro,it,fc]=newton(funcao2,deriva,x0,delta,epsilon,itmax)
printf('    Solução pelo Método de Newton: \n');
printf(' x = %f\n',x);
printf(' Erro = %e\n',erro);
printf(' Função = %e\n',fc);
printf(' Iterações = %d\n',it);
```

4.4 Sistemas de Equações Diferenciais Ordinárias(EDO)

Uma grande variedade de problemas da Engenharia Química pode ser formulada em termos de equações diferencias ordinárias (EDOs). A equação diferencial ordinárias é a equação envolvendo uma relação entre uma função desconhecida e e uma ou mais de suas derivadas. Equações envolvendo derivadas com somente uma variáveis independente são chamadas de equações diferencias ordinárias. As EDOs podem ser classificadas como problema de valor inicial (PVI) e problema de valor no contorno (PVC). A diferenciação do PVI e do PVC deve-se a localização das condições extras na formulação do problema e seguem a seguinte especificação:

- PVI: as condições são dadas para o mesmo valor da variável independente.
- PVC: as condições são dadas para valores distintos da variável independente.

A forma genérica de expressar um sistema de EDOs PVI é:

$$\frac{d\mathbf{y}}{dx} = \mathbf{f}(x, \mathbf{y}), \quad \mathbf{y}(x_o) = \mathbf{y}_o \tag{4.5}$$

4.4.1 EDO: Problema de Valor Inicial (PVI)

A resolução de EDOs, PVI no Scilab é feita através da função **ode** que é uma interface para várias funções de integração de EDOs pertencentes à biblioteca numérica ODEPACK.

A estrutura completa do comando ode é dada por:

[y,rd,w,iw]=ode(tipo,y0,t0,t [,rtol [,atol]],f [,jac],ng,g [,w,iw])

Com os parâmetros:

y0	vetor ou matriz real com condições iniciais
t0	valor do tempo inicial
t	vetor real com os instantes em que a solução é calculada
f	função, **list** ou *string* de caracteres
tipo	uma das seguintes opções: "adams", "stiff", "rk","rkf","fix","discrete","roots" Obs: **lsoda** é o pacote de integração *default*
rtol,atol	constantes ou vetor com tolerância relativa e absoluta de mesma dimensão de y. valores *default*: rtol=1.e-5 e atol=1.e-7 valores *default* para "rfk"e "fix": rtol=1.e-3 e atol=1.e-4
jac	função, **list** ou *string* de caracteres
w,iw	vetores.
ng	inteiro.
g	função, **list** ou *string* de caracteres
k0	inteiro, tempo inicial
kvect	inteiro (vetor)

A função **ode** usa a função interativa **odeoptions** para especificação de opções, ou pode-se usar a variável %ODEOPTIONS como um vetor conforme abaixo:

[itask,tcrit,h0,hmax,hmin,jactyp,mxstep,maxordn,maxords,ixpr,ml,mu] valor *default*, `[1,0,0,%inf,0,2,500,12,5,0,-1,-1]`

Exemplo 1: Simples EDO.

$$\frac{dy}{dx} = -21.6y, \quad y(0) = 1; \tag{4.6}$$

A equação acima é bem simples e possui solução analítica dada por: $y = exp(-21.6x)$. A implementação com a função **ode** pode ser feita conforme a Tabela 5.

Exemplo 2: Seja o sistema reacional.

$$\begin{aligned} A &\overset{k_1}{\rightarrow} B \\ B + C &\overset{k_2}{\rightarrow} A + C \\ 2B &\overset{k_3}{\rightarrow} C + B \end{aligned} \tag{4.7}$$

que resulta no sistema de EDOs,

$$\begin{bmatrix} \frac{dC_A}{dt} \\ \frac{dC_B}{dt} \\ \frac{dC_C}{dt} \end{bmatrix} = \begin{bmatrix} -k_1 C_A + k_2 C_B C_C \\ k_1 C_A - k_2 C_B C_C - k_3 C_B^2 \\ k_3 C_B^2 \end{bmatrix}$$

com as condições iniciais: $\mathbf{C_A}(0) = 1$, $\mathbf{C_B}(0) = 0$, $\mathbf{C_C}(0) = 0$ e os parâmetros $k_1 = 0.08$; $k_2 = 2 \times 10^4$; e $k_3 = 6 \times 10^7$
Um script que implementa a integração desse sistema encontra-se na Tabela 6.

Com o Scilab pode-se também estudar o retrato de fase de sistemas bidimensionais de duas maneiras. Seja o sistema de EDOs formado pelas equações:

$$\frac{d\mathbf{x}}{dt} = \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \mathbf{x}$$

com as condições iniciais: $\mathbf{x}(0) = \mathbf{x}_o$.

```
mode(-1);
// Template para resolução de EDOs PVI no SCILAB
// Formato de utilização da função ode:
// [y,rd,w,iw]=ode('root',y0,t0,t [,rtol [,atol]],f [,jac],ng,g [,w,iw])
//     Valor default:rtol=1.e-5 e atol=1.e-7
//     Valor default para 'rfk' e 'fix': rtol=1.e-3 e atol=1.e-4
// Definições das funções
// dy/dt=-21.6*y, y(0)=1
//
function [f]=fun1(x,y)
 f=-21.6*y;
endfunction
function [f]=fun2(x)
  f=exp(-21.6*x);
endfunction
// Programa principal
txt=['xo=';'yo=';'x='];
valor=x_mdialog('Forneça informações',txt,['0';'1';'0:0.01:1'])
x0=evstr(valor(1)); y0=evstr(valor(2)); x=evstr(valor(3));
y=ode(y0,x0,x,fun1);
xbasc();
subplot(211),plot2d(x,y), xtitle('Solução Numérica');
subplot(212),fplot2d(x,fun2), xtitle('Solução Analítica');
xselect()
```

Tabela 5: Exemplo de utilização da função **ode**

1. Integrando o sistema com a função **ode**. Analogamente ao apresentado no exemplo anterior, pode-se integrar esse sistema de equações diferencias ordinárias utilizando a função **ode** com várias condições iniciais, e se observar as trajetórias no plano de fases $x_1 \times x_2$. A Tabela 7 apresenta o script em Scilab e a Figura 14 representa o resultado do código.

2. Integrando o sistema com a função **fchamp**. A implementação em Scilab utilizando a função **fchamp** é dado pela Tabela 8.

4.4.2 EDO: Problema de Valor no Contorno (PVC)

Uma possibilidade para resolver problemas do tipo (de Assis, 2003):

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

$$y(x_c) = y_c \text{ com } x_c \neq 0$$

é considerar um valor para y em $x = 0$ e aplicar os métodos de integração para encontrar y em $x = x_c$, o local conhecido. Se y calculado for igual ao conhecido, ou seja, se $y = y_c$ em $x = x_c$ o

```
mode(-1);
// Template para resolução de EDOs PVI no SCILAB
// Formato de utilização da função ode:
// [y,rd,w,iw]=ode('root',y0,t0,t [,rtol [,atol]],f [,jac],ng,g [,w,iw])
//     Valor default:rtol=1.e-5 e atol=1.e-7
//     Valor default para 'rfk' e 'fix': rtol=1.e-3 e atol=1.e-4
// Definições das funções
//
function [f]=fun(t,y)
k=[0.08;2e4; 6e7];
  f(1)=-k(1)*y(1)+k(2)*y(2)*y(3);
  f(2)=k(1)*y(1)-k(2)*y(2)*y(3)-k(3)*y(2)^2;
  f(3)=k(3)*y(2)^2;
endfunction

// Programa principal
txt=['to=';'yo=';'t='];
valor=x_mdialog('Forneça informações',txt,['0';'[1;0;0]';'[0:0.1:10]'])
t0=evstr(valor(1));
y0=evstr(valor(2)); t=evstr(valor(3));
//%ODEOPTIONS=[itask,tcrit,h0,hmax,hmin,jactyp,mxstep,maxordn,maxords,ixpr,ml,mu]
%ODEOPTIONS=[1,0,0,%inf,0,2,500,12,5,1,-1,-1]; // printevel=1;
y=ode(y0,t0,t,fun);
xbasc();
subplot(311),plot2d(t,y(1,:));
subplot(312),plot2d(t,y(2,:));
subplot(313),plot2d(t,y(3,:));
xselect()
```

Tabela 6: Exemplo 2 de utilização da função **ode**

problema foi resolvido com sucesso. Caso contrário, necessita-se realizar uma nova estimativa para y(0) e repetir o procecimento. Este método de tentativa e erro, pouco eficiente e consumidor de tempo considerável pode ser melhorado se aplicarmos o método da bisseção do seguinte modo:

- Escolhe-se $y_1(0)$ de tal modo que ao integrar f(x,y), o valor de y_1 em $x = x_c$ seja maior que y_c;
- Escolhe-se $y_2(0)$ de tal modo que ao integrar f(x,y), o valor de y_2 em $x = x_c$ seja menor que y_c;
- Escolhe-se o novo $y_3(0)$ como a média aritmética entre $y_1(0)$ e $y_2(0)$ e realiza-se a integração até $x = x_c$, encontrando y_3;
- Se y_3 for maior que y_c, o novo intervalo será entre $y_3(0)$ e $y_2(0)$, desprezando-se $y_1(0)$; caso contrário, despreza-se $y_2(0)$ e o novo intervalo será entre $y_1(0)$ e $y_3(0)$;

```
mode(-1);
// Análise de diagrama de fase
function fun=funcao(t,x)
  fun(1)=x(1)+2*x(2);
  fun(2)=-2*x(1)+x(2);
endfunction
// Programa Principal
t=[0:0.01:20]; t0=0;
x0=[-3;3];[x1] = ode(x0,t0,t,funcao);
x0=[-2;3];[x2] = ode(x0,t0,t,funcao);
x0=[-1; 3];[x3] = ode(x0,t0,t,funcao);
x0=[1; 3];[x4] = ode(x0,t0,t,funcao);
x0=[2; 3];[x5] = ode(x0,t0,t,funcao);
x0=[3; 3];[x6] = ode(x0,t0,t,funcao);
x0=[-3; -3];[x7] = ode(x0,t0,t,funcao);
x0=[-2; -3];[x8] = ode(x0,t0,t,funcao);
x0=[-1; -3];[x9] = ode(x0,t0,t,funcao);
x0=[1; -3];[x10] = ode(x0,t0,t,funcao);
x0=[2; -3];[x11] = ode(x0,t0,t,funcao);
x0=[3; -3];[x12]=ode(x0,t0,t,funcao);
xbasc();xset('font size',12)
plot2d(x1(1,:),x1(2,:));plot2d(x2(1,:),x2(2,:));
plot2d(x3(1,:),x3(2,:));plot2d(x4(1,:),x4(2,:));
plot2d(x5(1,:),x5(2,:));plot2d(x6(1,:),x6(2,:));
plot2d(x7(1,:),x7(2,:));plot2d(x8(1,:),x8(2,:));
plot2d(x9(1,:),x9(2,:));plot2d(x10(1,:),x10(2,:));
plot2d(x11(1,:),x11(2,:));plot2d(x12(1,:),x12(2,:));
xtitle('Plano de Fase');xselect();
```

Tabela 7: Exemplo 3 de utilização da função **ode**: Retrato de fase

O Scilab possui rotinas especializadas para essa função. A função **bvode**() implementa código com esse objetivo.

```
[z]=bvode(points,ncomp,m,aleft,aright,zeta,ipar,ltol,tol,fixpnt,fsub1,dfsub1,gsub1,
         dgsub1,guess1)
 z     : A solução da ODE determinada sobre a malha fornecida por points
 points: Vetor que fornece os pontos onde se deseja a solução
 ncomp : Número de EDOs (ncomp <= 20)
 m     : Um vetor de dimensão ncomp.  m(j)  fornece a ordem da j-ésima EDO
 aleft : Lado esquerdo do intervalo
 aright: Lado direito do intervalo
 zeta  : zeta(j)  fornece j-ésimo ponto do contorno (boundary point). Deve ter
         zeta(j) <= zeta(j+1)
       Obs:
```

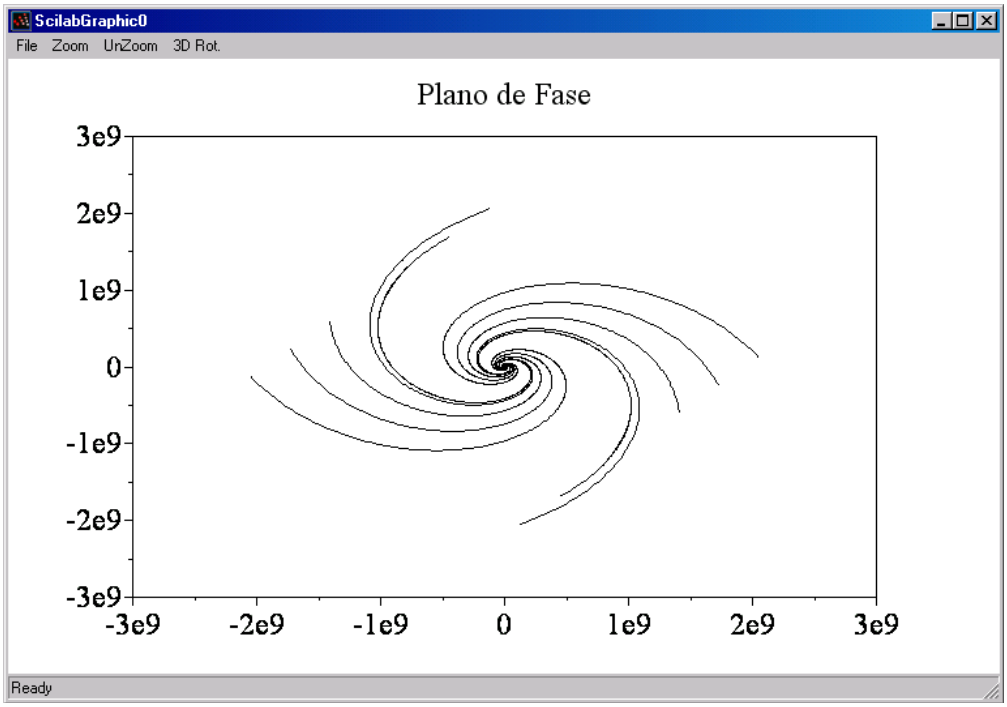

Figura 14: Diagrama de fases: Exemplo 3

```
mode(-1);
// Uso de fchamp
function fun=funcao(t,x)
  fun(1)=x(1)+2*x(2);
  fun(2)=-2*x(1)+x(2);
endfunction
// Programa Principal
xbasc();xset('font size',12)
xf= -20:1:20; yf= -20:1:20; t=0;
fchamp(funcao,t,xf,yf)
xtitle('Direção do campo vetorial');
xselect();
```

Tabela 8: Exemplo 4 de utilização da função **fchamp**

.

```
        todos pontos no contorno devem ser pontos na malha em todas malhas usadas,
        veja descrição de ipar(11) e fixpnt abaixo.
ipar  : Um vetor de inteiros com dimensão de no mínimo 11. A lista de parâmetros
        em ipar e seus significados são renomeados em bvode; seus novos nome são
        dados em parênteses.
       ipar(1)   = 0 se o problema é linear, 1 se o problema é não linear
```

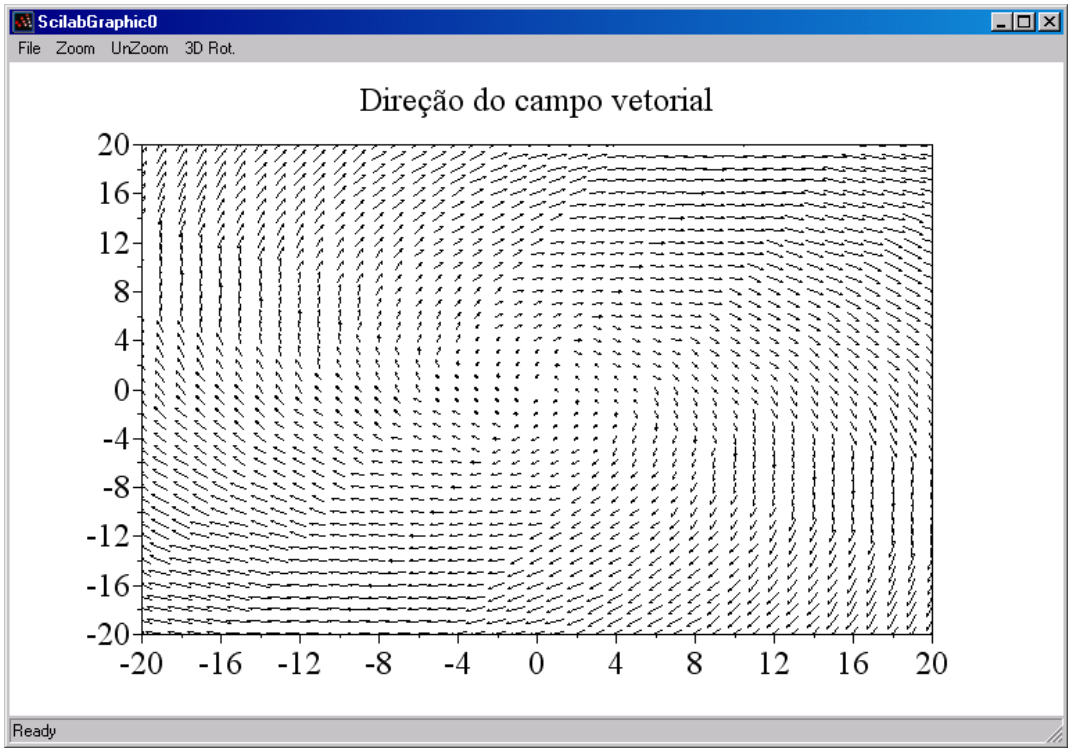

Figura 15: Campo de Direção: Exemplo 4

```
ipar(2)   = número de pontos de colocação por subintervalo (= k) onde
            max m(i) <= k <= 7 .
  Se ipar(2)=0 então bvode faz
      k = max ( max m(i)+1, 5-max m(i) )
ipar(3)   = número de subintervalos na malha inicial (= n).
  Se ipar(3) = 0  então bvode arbitrariamente faz n = 5.
ipar(4)   = número de tolerâncias para soluções e derivadas.
            (= ntol) exige-se 0 < ntol <= mstar.
ipar(5)   = dimensão de fspace (= ndimf) um vetor real de trabalho.
            Sua dimensão representa uma restrição sobre nmax.
            escolha ipar(5) de acordo com a fórmula: ipar(5)>=nmax*nsizef
      onde:
      nsizef=4+3*mstar+(5+kd)*kdm+(2*mstar-nrec)*2*mstar .
ipar(6)   = dimensão de ispace (= ndimi) um vetor inteiro de trabalho.
            Sua dimensão representa restrição sobre nmax, o número máximo de
            subintervalos. Escolha ipar(6) de acordo coma formula:
            ipar(6)>=nmax*nsizei
      onde:
      nsizei=3+kdm  com  kdm=kd+mstar; kd=k*ncomp;
      nrec=número de condições de contorno a direita.
ipar(7)  controle de saída de resultados (= iprint)
       = -1  para impressão completa de diagnóstico
        = 0  para impressão restrita
        = 1  para nenhuma impressão
```

```
       ipar(8)   (= iread)
               = 0  faz bvode gerar malha inicial uniforme.
               = xx  Outros valores ainda não implementados no Scilab
               = 1  Se a malha inicial é fornecida pelo usuário. Ela é definida em
                    fspace como segue: a malha ocupará  fspace(1), ..., fspace(n+1).
                    O usuário necessitará suprir apenas os pontos interiores
                    fspace(j) = x(j), j = 2, ..., n.
               = 2  se a malha inicial é fornecida pelo usuário com ipar(8)=1, e nenhuma
                    seleção de malha adaptativa é feita.
       ipar(9)   (= iguess )
               = 0  se nenhuma estimativa para a solução é fornecida.
               = 1  se estimativa inicial é fornecida pelo usuário na subrotina guess.
               = 2  se uma malha inicial e coeficientes de solução aproximados são
                    fornecidos pelo usuário em fspace. (o primeiro e novo mesh são
                    os mesmos).
               = 3  se a malha inicial e os coeficientes de solução aproximada são
                    fornecidos pelo usuário em fspace, e a nova malha é para ser
                    tomada duas vezes mais robusta; i.e., a cada segunda ponto da
                    malha inicial.
               = 4  se além da malha inicial e dos coeficientes de solução aproximada,
                    uma nova malha é fornecida em fspace. (veja descrição de saída
                    para outros detalhes sobre iguess = 2, 3, e 4.)
      ipar(10)
               = 0  se o problema é regular
               = 1  se o primeiro fator de relaxamento é =rstart, e a interação não
                    linear não se baseia em convergência passada (use somente para
                    problemas não lineares extra sensíveis).
               = 2  se deseja-se retornar imediatamente após (a) duas situações de
                    não convergência sucessivas, ou (b) após obter estimativa de erro
                    pela primeira vez.
      ipar(11)  = número de pontos fixos na malha além de aleft e aright.
                  (= nfxpnt, a dimensão de fixpnt)
                  o código requer que todas as outras condições de contorno, além de
                  aleft e aright,
                  (veja descrição de zeta) sejam incluídas como pontos fixos em fixpnt.
ltol  um vetor de dimensão ipar(4). ltol(j)=l  especifica que a j-ésima tolerância
      em tol controla o erro no l-ésimo componente de z(u). Também requer que:
             1 <= ltol(1) < ltol(2) < ... < ltol(ntol) <= mstar
tol   um vetor de dimensão ipar(4). tol(j) é a tolerância do erro no ltol(j)-ésimo
      componente de z(u).
      Assim, o código tenta satisfazer y para j=1:ntol em cada subintervalo
        abs(z(v)-z(u))       <=     tol(j)*abs(z(u))     +tol(j)
                     ltol(j)                        ltol(j)
      se  v(x) é o vetor de solução aproximada.
fixpnt  um vetor de dimensão ipar(11). Ele contém os outros pontos além de aleft e
        aright, que devem ser incluídos em cada mesh.
externals  as funções fsub,dfsub,gsub,dgsub,guess  são externos ao Scilab
           (veja sintaxe abaixo) ou o nome de uma subrotina Fortran
```

```
          A interface da função em Fortran com bvode são especificadas no
          arquivo fcol.f, disponível na distribuição Scilab.
fsub   nome de uma subrotina para avaliação. [f]=fsub(x,z)  onde f é o vetor
          contendo o valor de fi(x,z(u)) no i-ésimo componente
dfsub  nome da subrotina para avaliação do Jacobiano de f(x,z(u)) no ponto x.
       [df]=dfsub (x, z)  onde z(u(x)) é definida analogamente aquele para
       fsub e a matriz df com dimensão (ncomp ) por (mstar) deve conter as
       derivadas parciais de f, para uma chamada calcula-se
                          df(i,j) = dfi / dzj, i=1,...,ncomp
                                                j=1,...,mstar.
gsub  nome de subrotina para avaliação do i-ésimo componente de
      g(x,z(u(x)))=g(zeta(i),z(u(zeta(i)))) no ponto x=zeta(i) onde 1<=i<=mstar.
       [g]=gsub(i,z) onde z(u) é idêntico aquele de fsub, e i e g=gi são como acima.
       Note que diferentemente de f em fsub, aqui somente um valor por chamada
       retorna em g.
dgsub  nome da subrotina para avaliação da i-ésima linha do Jacobiano de g(x,u(x)).
       [dg]=dgsub (i, z)  onde  z(u) é o memso de fsub, i é idêntico a gsub e o vetor
       dg de dimensão mstar possui derivadas parciais de g.
guess  nome de subrotina para avaliar aproximação inicial para z(u(x)) e para
       dmval(u(x))= vetor de mj-ésima derivadas de u(x). [z,dmval]= guess(x).
       Note que essa subrotina é usada somente se ipar(9) = 1 então todos os mstar
       componentes de z e ncomp componentes de dmval devem ser especificados para
       qualquer x, aleft <= x <= aright.
```

O script a seguir aplica esse código para a resolução do problema:

```
deff('df=dfsub(x,z)','df=[0,0,-6/x**2,-6/x]')
deff('f=fsub(x,z)','f=(1 -6*x**2*z(4)-6*x*z(3))/x**3')
deff('g=gsub(i,z)','g=[z(1),z(3),z(1),z(3)];g=g(i)')
deff('dg=dgsub(i,z)',['dg=[1,0,0,0;0,0,1,0;1,0,0,0;0,0,1,0]';
                      'dg=dg(i,:)'])
deff('[z,mpar]=guess(x)','z=0;mpar=0')
deff('u=trusol(x)',[
   'u=0*ones(4,1)';
   'u(1) =  0.25*(10*log(2)-3)*(1-x) + 0.5 *( 1/x   + (3+x)*log(x) - x)'
   'u(2) = -0.25*(10*log(2)-3)+ 0.5 *(-1/x^2 + (3+x)/x      + log(x) - 1)'
   'u(3) = 0.5*( 2/x^3 + 1/x   - 3/x^2)'
   'u(4) = 0.5*(-6/x^4 - 1/x/x + 6/x^3)'])
fixpnt=0;m=4;
ncomp=1;aleft=1;aright=2;
zeta=[1,1,2,2];
ipar=zeros(1,11);
ipar(3)=1;ipar(4)=2;ipar(5)=2000;ipar(6)=200;ipar(7)=1;
ltol=[1,3];tol=[1.e-11,1.e-11];
res=aleft:0.1:aright;
```

```
z=bvode(res,ncomp,m,aleft,aright,zeta,ipar,ltol,tol,fixpnt,...
 fsub,dfsub,gsub,dgsub,guess)
z1=[];for x=res,z1=[z1,trusol(x)]; end;
z-z1
```

Pode-se verificar que o código **bvode**(), embora poderoso, possui complexidade de parâmetros para a sua utilização mais abrangente. Uma outra possibilidade para a solução de PVC é aplicar o método das diferenças finitas, como exposto a seguir.
Considere a EDO de 2^a ordem mostrada na Equação 4.8 que descreve a variação da temperatura ao longo de um trocador de calor tubular, com o fluido escoando da direita para a esquerda, entrando no trocador a uma temperatura conhecida e saindo na mesma temperatura da parede do tubo:

$$\begin{gathered} Pe\frac{d\phi}{dx} + \frac{d^2\phi}{dx^2} - 2Nu\phi = 0 \\ x = 0; \phi = 0 \\ x = X; \phi = 1 \end{gathered} \tag{4.8}$$

Aplicando diferenças finitas centrais:

$$\dot{y}_j = \frac{y_{j+1} - y_{j-1}}{2h} \tag{4.9}$$

$$\ddot{y}_j = \frac{y_{j+1} - 2y_j + y_{j-1}}{h^2} \tag{4.10}$$

Tem-se:

$$(2 - Peh)\phi_{j-1} - 4(Nuh^2 + 1)\phi_j + (2 + Peh)\phi_{j+1} = 0, \quad j = 1, \ldots, J-1 \tag{4.11}$$

onde N é o número de subintervalos criados na discretização, cada qual com comprimento h dado por $h = X/N$, neste caso. A Equação 4.11 é aplicada para j=1,...,N-1 pois conhece-se o valor da variável dependente ϕ para o primeiro (x=0) e para o último ponto (x=1). Obtem-se, deste processo, um sistema de equações algébricas lineares neste caso, podendo aplicar-se os métodos já estudados anteriormente na sua solução. Empregaremos o **método de Gauss-Seidel** na solução deste problema, cujo script de implementação no Scilab é mostrado a seguir:

```
clear;
//
// Implementado por Adilson J. de Assis
//
function [fi,kk]=pvc(xrange,fi0bound,fiLbound,epsilon,nmax)

n = length(xrange);
h = (xrange(n)-xrange(1))/(n-1);
//estimativa inicial de fi e fik (var. auxiliar)
fi =ones(n);
fik=ones(n)
//condições de contorno
```

```
fi(1) = fi0bound;
fi(n) = fiLbound;

printf('iterações iniciadas...aguarde!\n\n');
for k=1:nmax
    ke = 0; kk = k;
    //imprime a iteração se for múltipla de 10, inclusive.
    if modulo(kk,10) == 0 then
       printf('A iteração atual é %g.\n',k);
    end;
    for j = 2:n-1
        fik(j) = ((2-Pe*h)*fi(j-1)+(2+Pe*h)*fi(j+1))/(4*(Nu*h^2+1));
        if abs(fik(j)-fi(j)) > epsilon then
           ke = ke + 1;
        end
    fi(j)=fik(j);
    end;
    if ke == 0 then
       break
    end
end
if kk == nmax then
   printf('O número máximo de iterações %g foi alcançado \n\n',kk);
end
endfunction

Pe = 1;
Nu = 1;
u0 = 0;
uL = 1;
nmax = 10000;
epsilon = 1e-6;
J  = 4;
X  = 4;
h  = X/J;
x1  = [0:h:X];
x2  = [0:h/2:X];
x3  = [0:h/4:X];
x4  = [0:h/8:X];
x5  = [0:h/16:X];
x6  = [0:h/32:X];
[u1,k1] = pvc(x1,u0,uL,epsilon,nmax);
[u2,k2] = pvc(x2,u0,uL,epsilon,nmax);
[u3,k3] = pvc(x3,u0,uL,epsilon,nmax);
[u4,k4] = pvc(x4,u0,uL,epsilon,nmax);
[u5,k5] = pvc(x5,u0,uL,epsilon,nmax);
```

```
[u6,k6] = pvc(x6,u0,uL,epsilon,nmax);
ktot = [k1 k2 k3 k4 k5 k6];
xbasc();
plot2d(x1,u1,-1);
plot2d(x2,u2,-1);
plot2d(x3,u3,1);
plot2d(x4,u4,1);
plot2d(x5,u5,1);
plot2d(x6,u6,1);
xtitle('Solução Numérica da Eq. do calor');
```

Na Figura 16 mostra-se o perfil de temperatura para N=4,8,16,32,64,128, sendo que para os dois primeiros valores, são os pontos + e os demais estão na forma de linha contínua. Visualmente quase não há diferença[21] para $N > 16$.

4.5 Introdução à Otimização

As técnicas de otimização constituem uma das mais importantes ferramentas no processo de tomada de decisões. Essas técnicas tem como finalidade a busca de uma melhor solução para um determinado problema (máximos ou mínimos), e fazem-se necessárias em muitas áreas da engenharia, tais como:

- pesquisa operacional: otimização de sistemas técnico-econômicos, controle de estoques, planejamento de produção etc.;
- projeto de processo: dimensionamento e otimização de processos, estimação de parâmetros, reconciliação de dados, análise de flexibilidade etc.;
- controle de processo: identificação de sistemas, controle ótimo, controle adaptativo, controle preditivo etc.;
- análise numérica: aproximações, regressão, solução de sistemas lineares e não-lineares etc.

4.5.1 Ajuste de Modelos: Método dos Mínimos Quadrados

Um modelo relaciona as saídas (variáveis dependentes) como as entradas (variáveis independentes). As equações de um modelo em geral inclui também coeficientes que são presumidos constantes. Nesse livro, esses coeficientes serão referidos como parâmetros. Com a informação de dados experimentais pode-se determinar não apenas a forma matemática do modelo, mas também esses coeficientes. A determinação dos parâmetros se dá através de procedimentos conhecidos como determinação de parâmetros, regressão ou identificação de modelos. O ajuste de uma modelo (empírico ou teórico) necessita de pelo menos tantos dados experimentais quantos sejam os parâmetros a

[21] deve-se dedicar especial atenção para a tolerância adotada (neste caso: 1×10^{-6}), pois há que se considerar cuidadosamente este valor juntamente com o valor de ϕ. Para este problema uma tolerância menor pode levar a conclusões errôneas acerca do comportamento da temperatura.

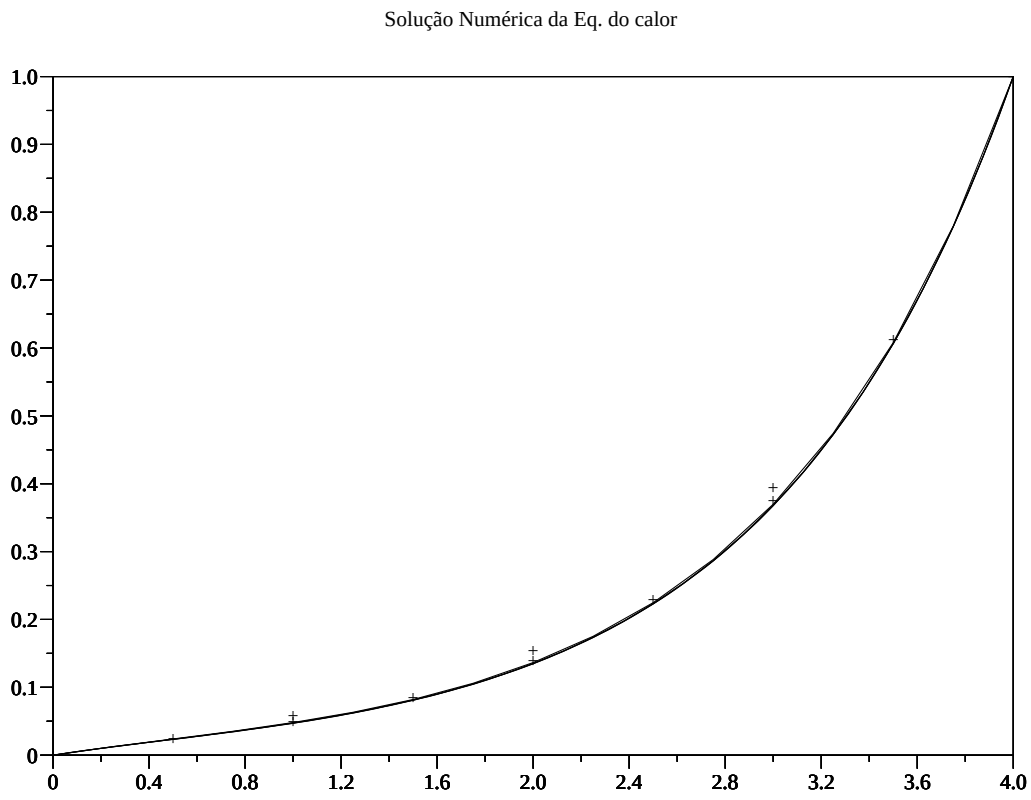

Figura 16: Solução de PVC usando diferenças finitas

serem determinados. Ou seja, com três pontos experimentais para y e x pode-se estimar um modelo que tenha no máximo três parâmetros. Existem vários critérios que podem ser usados para a estimativa dos parâmetros de um modelo com a utilização de dados experimentais, eles baseiam-se na definição de um erro, ε_j, para cada ponto experimental. Assim, em um conjunto de p pontos experimentais pode-se definir o erro ε_j, $j = 1, \ldots p.$ como sendo a diferença entre os dados experimentais Y_j e os valores preditos pelo modelo, $y_j(\mathbf{x})$,

$$\varepsilon_j = Y_j - \hat{y}_j, \quad j = 1 \ldots p$$

Dentre os vários critérios de ajuste, o problema de minimização do somatório do quadrado dos erros, é um dos mais utilizados pois amplifica erros grandes muito mais do que os erros pequenos. Esse critério baseia-se na função objetivo,

$$f = \sum_{j=1}^{p} {\varepsilon_j}^2 \tag{4.12}$$

Considerando um modelo que é linear nos parâmetros e tem a forma:

$$y = \sum_{i=0}^{n} \beta_i x_i, \quad x_o = 1 \tag{4.13}$$

No modelo acima existem n variáveis independentes x_i, $i = 1, \ldots n$. A equação 4.13 é linear nos parâmetros β_i, mas x_i pode ser não linear de forma que uma equação quadrática em x pode ser escrita nessa forma. Assim, Assim,

$$y = \beta_o + \beta_1 x + \beta_2 x^2 \tag{4.14}$$

pode ser representada por

$$y = \beta_o x_o + \beta_1 x_1 + \beta_2 x_2, \quad \text{com } x_o = 1, x_1 = x, x_2 = x^2 \tag{4.15}$$

A solução do problema de minimização da função 4.12 pode ser escrito na forma:

$$\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

com:

$$\mathbf{b} = \begin{bmatrix} \beta_o \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}, \qquad \mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_p \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1n} \\ 1 & x_{21} & x_{22} & \ldots & x_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{p1} & x_{p2} & \ldots & x_{pn} \end{bmatrix} \tag{4.16}$$

Exemplo : Aplicação do Método dos Mínimos quadrados
Deseja-se ajustar um modelo quadrático $y = \beta_0 + \beta_1 x + \beta_2 x^2$ utilizando-se o conjunto de dados experimentais abaixo:

x	20	20	30	40	40	50	50	50	60	70
Y	73	78	85	90	91	87	86	91	75	65

O *script* a seguir implementa o método apresentado na equação 4.16 para esse exemplo. Nesse caso precisa-se montar a matriz $\mathbf{X}$,

$$\begin{matrix} x_o & x & x^2 \end{matrix} \tag{4.17}$$

$$\mathbf{X} = \begin{bmatrix} 1 & 20 & 20^2 \\ 1 & 20 & 20^2 \\ 1 & 30 & 30^2 \\ 1 & 40 & 40^2 \\ 1 & 40 & 40^2 \\ 1 & 50 & 50^2 \\ 1 & 50 & 50^2 \\ 1 & 50 & 50^2 \\ 1 & 60 & 60^2 \\ 1 & 70 & 70^2 \end{bmatrix} \tag{4.18}$$

e determinar-se o vetor $\mathbf{b}$.

```
//Exemplo de ajuste de Dados
x = [20 ; 20 ; 30 ; 40 ; 40 ; 50 ; 50 ; 50 ; 60 ; 70 ];
Y = [73 ; 78 ; 85 ; 90 ; 91 ; 87 ; 86 ; 91 ; 75 ; 65 ];
// Modelo y=bo+b1*x+b2*x^2  -> 3 parâmetros
n=3;
// Dados experimentais
p=length(Y);
// Montagem de X
X=[ones(p,1) x x.^2];
if p >= n then
  if p==n then
    b=inv(X)*Y;
  else
    b=inv(X'*X)*X'*Y;
  end
else
  printf(' Conjunto de Dados Insuficiente para Modelo \n');
end
disp(b);
```

que fornece o resultado[22] para **b**:

! 35.657437 !
! 2.6314478 !
! – .0319185 !

4.5.2 Ajuste de Modelos a Dados Experimentais

O ajuste de modelos a dados experimentais, ou identificação pode ser usado através da função **datafit**[23]. Para uma dada função G(p,z), **datafit** determina o melhor vetor de parâmetros **p** que aproximando $G(p, z_i) = 0$ para um conjunto de dados experimentais z_i. O vetor p é calculado através da resolução do problema:

$$\min_p G(p, z_1)'WG(p, z_1) + G(p, z_2)'WG(p, z_2) + ... + G(p, z_n)'WG(p, z_n) \tag{4.19}$$

A sintaxe de **datafit** é dada por:

```
[p,err]=datafit([imp,] G [,DG],Z [,W],[contr],p0,[algo],[df0,[mem]],[work],[stop],['in'])
```

Com os parâmetros:

[22] O Scilab apresenta um alerta "*warning*" de matriz próximo a condição de matriz singular e calcula a solução pelo métodos dos mínimos quadrados.

[23] datafit é uma versão aperfeiçoada de fit_dat.

[p,err]=datafit([imp,] G [,DG],Z [,W],[contr],p0,[algo],[df0,[mem]],[work],[stop],['in'])	
imp	argumento escalar usado para definir modo de relatório: imp=0 nada é reportado (exceto erros) imp=1 relatório inicial e final imp=2 um relatório por interação imp>2 adiciona relatório na busca linear **Atenção**: A maioria desses relatórios é escrita na unidade padrão de saída do Scilab
G	função erro (e=G(p,z), e: ne x 1, p: np x 1, z: nz x 1)
DG	matriz das derivadas parciais de G em relação a p, (opcional), S=DG(p,z), S: ne x np)
Z	matriz $[z_1, z_2, ...z_n]$, onde z_i (nz x 1) é o i-ésimo dado experimental
W	matriz ne x ne de pesos (opcional), o *default* é nenhuma ponderação
contr	'b',binf,bsup com binf e bsup vetores com mesma dimensão que p0. binf e bsup são limites (*bounds*) inferiores e superiores para p.
p0	Estimativa inicial para p (dimensão np x 1)
algo	Algoritmo a ser usado: 'qn' é o quasi-Newton (*default*) 'gc' é o algoritmo de gradiente conjugado e 'nd' é o não diferenciável (não aceita limites para x)
df0	número escalar representando o decréscimo de f na primeira interação (df0=1 é o valor *default*)
mem	número de variáveis usado para aproximar a Hessian nos algoritmos (algo='gc' ou 'nd') valor *default* é 6
stop	seqüencia de parâmetros opcionais que controlam a convergência do algoritmo stop= 'ar',nap, [iter [,epsg [,epsf [,epsx]]]] "ar": palavra reservada para critério de parada definida conforme: nap: número máximo de chamadas a função permitido iter: número máximo de iterações permitido epsg: *threshold* da norma do gradiente epsf: *threshold* controlando decrescimento de f epsx: *threshold* controlando variação de x. Esse vetor (possivelmente matriz) de mesma dimensão de x0 pode ser usado para colocar x em escala.
"in"	palavra reservada para inicialização de parâmetros usados quando função é dada como uma rotina Fortran
p	vetor coluna, solução encontrada
err	escalar, mínimo erro quadrado

Nas próximas tabelas, vários exemplos de ajustes são feitos. Os exemplos foram preparados para execução de *script*. O leitor pode então transcrever os exemplos para um único arquivo **ajuste.sce** e executá-lo para análise dos resultados.

Exemplo: Gerando dado para ajuste:

```
//Exemplos de Aplicação para a função 'datafit'
X = [0:0.1:10]; Y = 20+30*X+50*X^2;
E = 1000*(rand(1,length(X))-0.5);Y1 = Y + E; Z = [X;Y1];
xset('window',0); xset('font size',14);xset('thickness',2);
plot2d(X,Y,5);plot2d(X,Y1,-1);
xtitle('Função Quadrática com Erro Randômico','x','y');
x_message('Pressione OK para continuar...'); xselect();
```

Exemplo 1: Ajuste Quadrático:

```
//Ajuste quadrático com 'datafit'
deff('[e]=G(a,z)','e=z(2)-a(1)-a(2)*z(1)-a(3)*z(1)^2')
a0 = [1;1;1];
h=figure(1);
uicontrol( h, 'style','text',...
         'string','Favor aguardar. Calculando...', ...
        'position',[1 180 200 20], ...
        'fontsize',15);
[aa,er] = datafit(G,Z,a0)
close(h);
deff('[y]=f(x)','y=aa(1)+aa(2)*x+aa(3)*x^2')
YY = f(X);
xset('window',1);xbasc(); xset('font size',14);xset('thickness',2);
plot2d(X,YY,5);plot2d(X,Y1,-1);
xtitle('Ajuste Quadrático e Dados','x','y')
x_message('Pressione OK para continuar'); xselect();
```

Exemplo 2: Ajuste Cúbico:

```
//Ajuste Cúbico com 'datafit'
deff('[e]=G1(a,z)','e=z(2)-a(1)-a(2)*z(1)-a(3)*z(1)^2-a(4)*z(1)^3')
a0 = [15;25;49;10];
h=figure(1);
uicontrol( h,'style','text', ...
 'string','Favor aguardar. Calculando...', ...
         'position',[1 180 200 20], ...
         'fontsize',15);
[aa,er] = datafit(G1,Z,a0)
close(h);
deff('[y]=f1(x)','y=aa(1)+aa(2)*x+aa(3)*x^2+aa(4)*x^3')
YYY = f1(X);
xset('window',2);xbasc();
xset('font size',14);xset('thickness',2);
plot2d(X,YYY,5);plot2d(X,Y1,-1);
xtitle('Ajuste Cúbico e Dados','x','y')
x_message('Pressone OK para continuar...'); xselect();
```

Exemplo 3: Ajuste Exponencial:

```
//Ajuste Exponencial com 'datafit'
deff('[e]=G2(a,z)','e=z(2)-a(1)-a(2)*exp(a(3)*z(1))')
a0 = [5;1;2];
h=figure(1);
uicontrol( h, 'style','text', ...
         'string','Favor aguardar. Calculando...', ...
         'position',[1 180 200 20], ...
         'fontsize',15);
[aa,er] = datafit(G2,Z,a0)
close(h);
deff('[y]=f2(x)','y=aa(1)+aa(2)*exp(aa(3)*x)')
YYYY = f2(X);
xset('window',3);xbasc();
xset('font size',14);xset('thickness',2);
plot2d(X,YYYY,5);plot2d(X,Y1,-1);
xtitle('Ajuste Exponencial e Dados','x','y')
x_message('Pressione OK para continuar...'); xselect();
```

Exemplo: Comparação Gráfica dos Ajustes:

```
xset('window',4);xbasc();
xset('font size',14);xset('thickness',2);
plot2d(X,YYY,5);plot2d(X,Y1,-1);plot2d(X,YY,2);plot2d(X,YYYY,6);
xtitle('Ajuste Quadrático, Cúbico, Exponencial e Dados','x','y')
x_message('Pressione OK para finalizar.'); xselect();
```

A execução fornece os seguintes resultados:

- Ajuste quadrático ($y = a_1 + a_2x + a_3x^2$):
  erro = 8024369.8 e parâmetros a=[-74.616712, 101.18464, 41.974133].
- Ajuste cúbico ($y = a_1 + a_2x + a_3x^2 + a_4x^3$):
  erro =7802128.6 e parâmetros a =[-191.566, 245.115, 5.8119456, 2.4108133].
- Ajuste exponencial ($y = a_1 + a_2exp(a_3x)$):
  erro =13002784 e parâmetros: a=[6.7950864, 331.63134, .2834938].

Dentre as funções mais importantes para a otimização de problemas, destacam-se:

Função **datafit**: Determinação de parâmetros
[p,err]=datafit([imp,] G [,DG],Z [,W],[contr],p0,[algo],[df0,[mem]],[work],[stop],['in'])
Função **linpro**: Resolução de Problemas LP
[x,lagr,f]=linpro(p,C,b,ci,cs,me,x0 [,imp])
Função **quapro**: Resolução de Problemas QP
[x,lagr,f]=quapro(Q,p,C,b,ci,cs,me,x0 [,imp])
Função **leastsq**: Resolução de Problemas de Mínimos Quadrados não lineares
[f,xopt]=leastsq([imp,] fun [,Dfun],x0) [f,[xopt,[gradopt]]]=leastsq(fun[,Dfun],[contr],x0,['algo'],[df0,[mem]],[stop],['in'])
Função **optim**: Resolução de Problemas NLP
[f,xopt]=optim(costf,x0) [f,[xopt,[gradopt,[work]]]]=optim(costf,[contr],x0,['algo'],[df0,[mem]],[work],[stop],['in'],[imp=iflag])
Função **semidef**: Resolução de Problemas de Prohramação Semidefinida
[x,Z,ul,info]=semidef(x0,Z0,F,blck_szs,c,options)

4.6 Solução de equações algébrico-diferenciais

Seja o sistema descrito pela seguintes equações algébrico-diferenciais(de Assis, 2003):

$$\begin{aligned} -0,04y_1 + 1\times 10^4 y_2 y_3 - \frac{dy_1}{dt} &= 0 \\ 0,04y_1 - 1\times 10^4 y_2 y_3 - 3\times 10^7 y_2^2 - \frac{dy_2}{dt} &= 0 \\ y_1 + y_2 + y_3 - 1 &= 0 \end{aligned} \tag{4.20}$$

que está na forma (método BDF):

$$F(t, y, \dot{y}) = 0$$

e cuja matriz de iteração

$$\frac{\partial F}{\partial y} + cj\frac{\partial F}{\partial \dot{y}}$$

é dada por:

$$\begin{bmatrix} -0,04 - cj & 1\times 10^4 y_3 & 1\times 10^4 y_2 \\ 0,04 & -1\times 10^4 y_3 - 2\times 3\times 10^7 y_2 - cj & -1\times 10^4 y_2 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.21}$$

As condições inicias são:

$$y(0) = [1 \quad 0 \quad 0]^T$$

$$\dot{y}(0) = [-0,04 \quad 0,04 \quad 0]^T$$

O *script* que resolve este sistema de EADs através da função **dassl()** no Scilab é:

```
clc
clf()
function [r,ires]=chemres(t,y,yd)
         r(1)=-0.04*y(1)+1d4*y(2)*y(3)-yd(1);
```

```
        r(2)=0.04*y(1)-1d4*y(2)*y(3)-3d7*y(2)*y(2)-yd(2);
        r(3)=y(1)+y(2)+y(3)-1;
        ires=0;
endfunction

y0=[1;0;0];
yd0=[-0.04;0.04;0];
t=[1.d-5:0.02:.4,0.41:.1:4,4.1:1:15];
y=dassl([y0,yd0],0,t,chemres);
xbasc();
subplot(131),
plot2d(y(1,:)',y(2,:),16,'021');
xtitle('Comportamento - y(1)',   ' ',    ' ');
subplot(132),
plot2d(y(1,:)',y(3,:),-6,'021');
xtitle('Comportamento - y(2)',   ' ',    ' ');
subplot(133),
plot2d(y(1,:)',y(4,:),9,'021');
xtitle('Comportamento - y(3)',   ' ',    ' ');
```

A função **dassl()** também poderia ser chamada com a opção de chamada à função externa que calcula o jacobiano, neste caso chamada de chemjac:

```
function [pd]=chemjac(x,y,yd,cj)
        pd=[-0.04-cj , 1d4*y(3)               , 1d4*y(2);
        0.04    ,-1d4*y(3)-2*3d7*y(2)-cj ,-1d4*y(2);
        1       , 1                      , 1       ];
endfunction
```

sendo o integrador chamado através do comando:

```
y=dassl([y0,yd0],0,t,chemres,chemjac);
```

4.7 Solução de Equações Diferenciais Parciais (EDPs) por diferenças finitas

Exemplo: Equações Elípticas: Equação de Laplace Consideremos a solução de uma EDP elíptica, a Equação de Laplace, mostrada a seguir juntamente com as condições de contorno

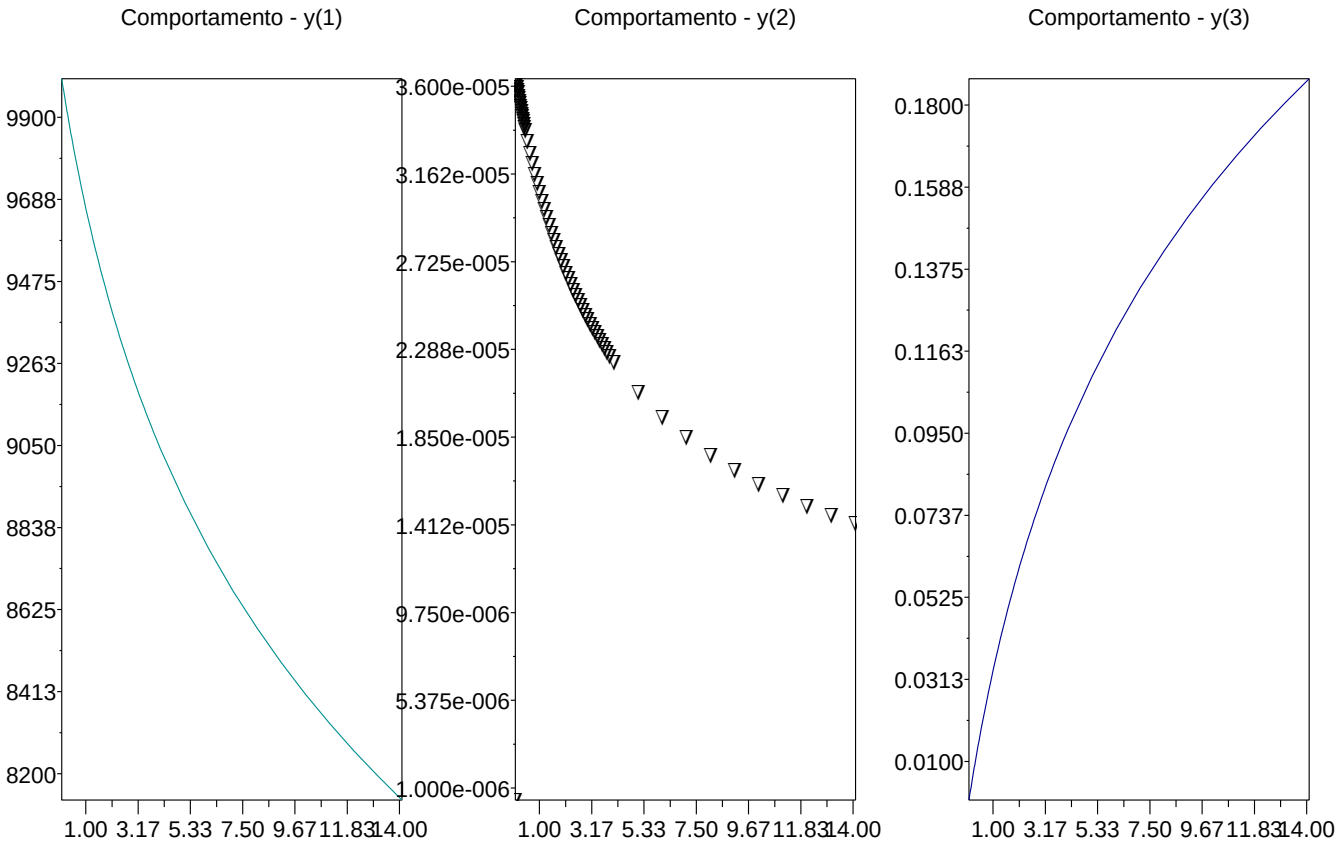

Figura 17: Solução de EADs no Scilab

consideradas (de Assis, 2003):

$$\begin{aligned} &\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \\ &u(0, y) = 0 \\ &u(1, y) = 1 \\ &u(x, 0) = 1 - x^2 \\ &u(x, 1) = 0 \end{aligned} \tag{4.22}$$

A célula de discretização para o domínio considerado está mostrada na Figura 18, onde se consideraram divisões iguais para o domínio em x (I=5) e em y (J=5). Lembrar que conforme nossa nomenclatura, a variação na direção x é representada pelo índice i e a variação em y pelo índice j. Os pontos marcados com $\times$ são aqueles cuja solução é conhecida dada pelas condições de contorno.

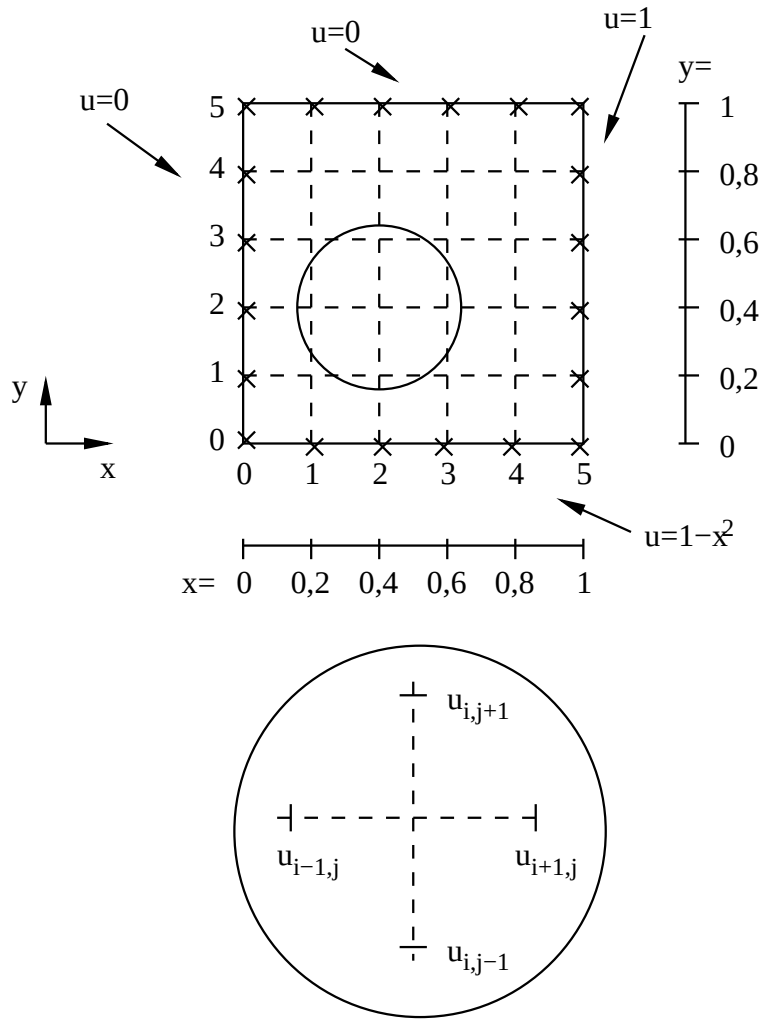

Figura 18: Célula de discretização usada para resolver a Eq. de Laplace

Aplicando diferenças finitas centrais nas duas direções, temos:

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^2} = 0 \qquad (4.23)$$
$$i = 1, ..., I-1 \text{ e } j = 1, ..., J-1$$

Neste exemplo u é conhecido em todos os pontos de fronteira, não aparecendo u fictício. O *script* (de Assis, 2003) a seguir implementa no Scilab a solução deste problema, mostrada na forma gráfica 3D na Figura 19 e as linhas de contorno na Figura 20[24].

[24]Como o Scilab não permite indexar vetores e matrizes a partir do zero, ou seja, fazer *u(0,j)*, que aparecerá na Equação 4.23 quando se aplicar *i=1*, deve-se incrementar tais índices, iniciando os laços a partir de *i=2* e *j=2* e terminando em *i=I* e *j=J*. Do mesmo modo, como o I e o J estão sendo definidos a partir dos intervalos de x e y, sendo calculados a partir do comando **length()**, que fornece a quantidade de elementos em um vetor, ou a dimensão do vetor (ou matriz). Como I = n-1 e J = m -1, o fim dos laços estão definidos por estes valores.

```
clear;
function [u,kk]=laplacepde(xrange,yrange,ux1,uxn,uy1,uym,epsilon,nmax)

n = length(xrange); m = length(yrange);
Dx = (xrange(n)-xrange(1))/(n-1);
Dy = (yrange(m)-yrange(1))/(m-1);
//estimativa inicial de u e uk (var. auxiliar)
u = ones(n,m); uk = ones(n,m);
//condições de contorno
u(:,1) = uy1'; u(:,m) = uym';
u(1,:) = ux1;  u(n,:) = uxn;

printf('iterações iniciadas...aguarde!\n\n');
for k=1:nmax
    ke = 0; kk = k;
    //imprime a iteração se for múltipla de 10, inclusive.
    if modulo(kk,10) == 0 then
       printf('A iteração atual é %g.\n',k);
    end;
    for i = 2:n-1
        for j = 2:m-1
        uk(i,j) = ...
        (Dx^2*(u(i,j-1)+u(i,j+1))+Dy^2*(u(i-1,j)+u(i+1,j)))/(2*(Dx^2+Dy^2));
            if abs(uk(i,j)-u(i,j)) > epsilon then
               ke = ke + 1;
            end
            u(i,j)=uk(i,j);
        end;
    end;
    if ke == 0 then
       break
    end
end
if kk == nmax then
   printf('O número máximo de iterações %g foi alcançado \n\n',kk);
end

endfunction

x = [0:0.05:1]; y = [0:0.05:1];
ux1 = zeros(y); uxn = ones(y); uy1 = 1-x^2; uym = zeros(x);
nmax = 1000;
epsilon = 1e-4;
[u,k] = laplacepde(x,y,ux1,uxn,uy1,uym,epsilon,nmax);
k
```

```
plot3d(x,y,u,45,45,'x@y@u(x,y)')
xtitle('Solução Numérica da Eq. de Laplace');
pause

contour(x,y,u,10);
xtitle('Solução numérica para a Eq. de Laplace','x','y');
```

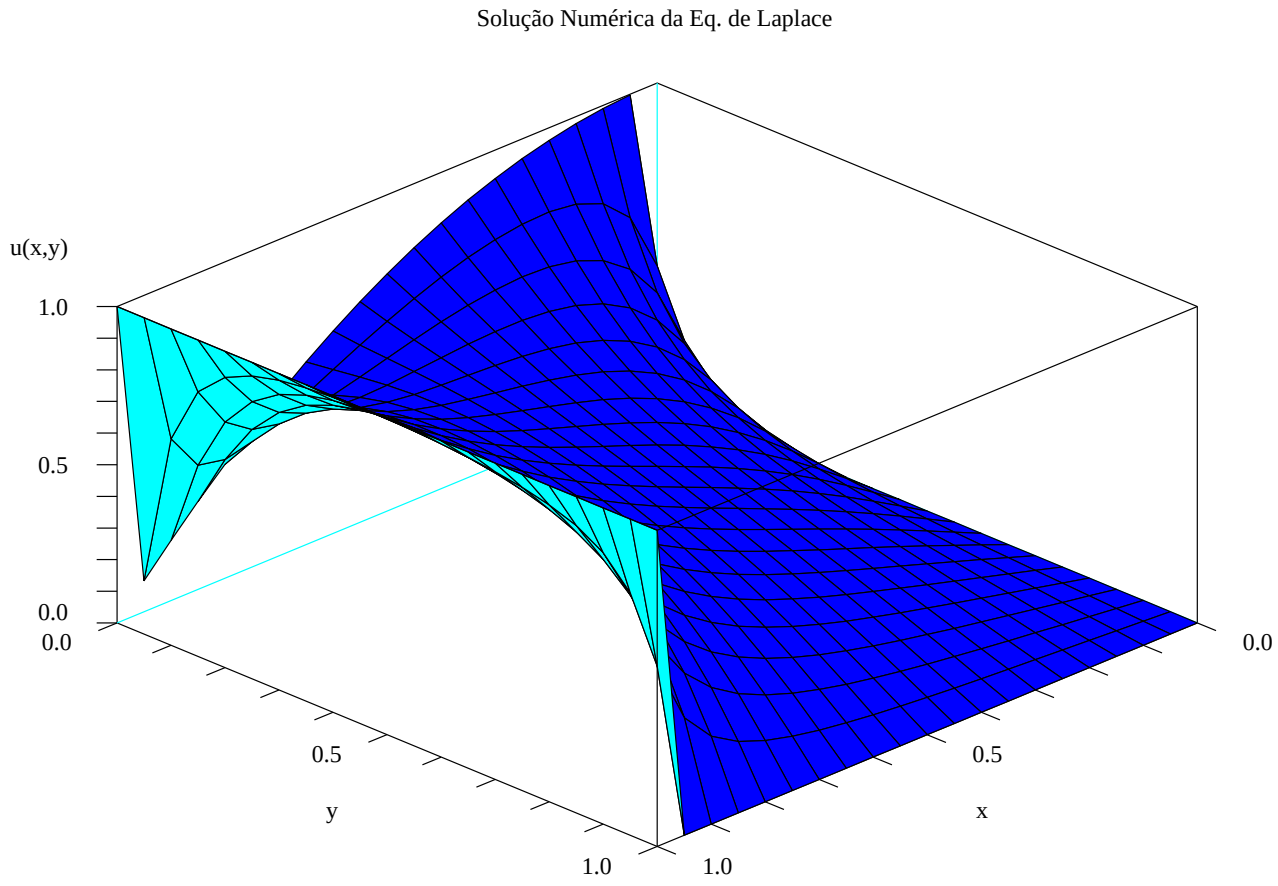

Figura 19: Solução da Eq. de Laplace por diferenças finitas - 3D

5 Aspectos Complementares

Nessa seção apresenta-se a utilização o Scilab para o estudo de inicial de problemas de Controle na Engenharia Química.

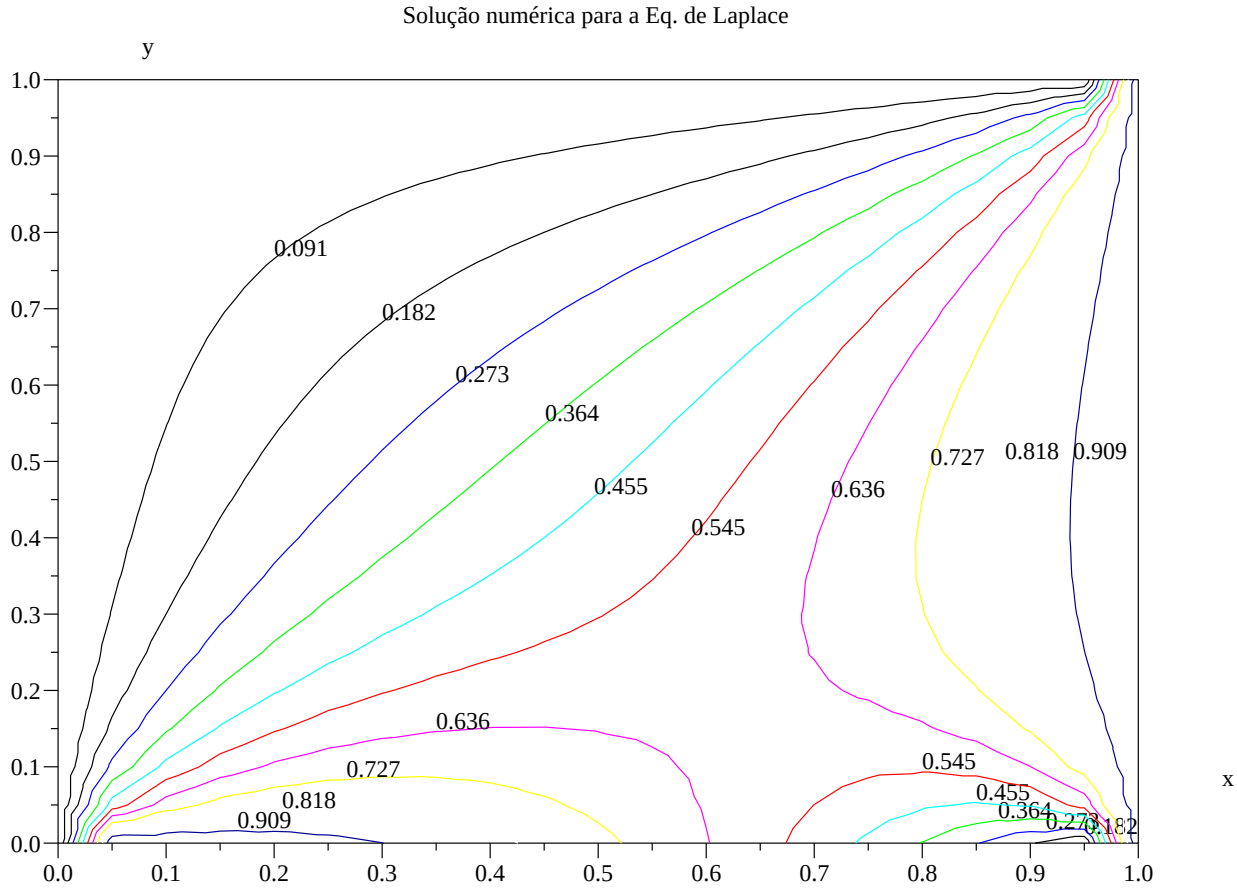

Figura 20: Solução da Eq. de Laplace por diferenças finitas - contornos

5.1 Sistemas de Controle

5.1.1 Representação de Modelos Lineares no Scilab

Em geral o especialista em controle de processos necessita desenvolver uma representação dinâmica do processo que se deseja controlar. Esses modelos representam a planta a ser controlada em uma ambiente de simulação de estratégias de controle. Por outro lado, uma grande categoria de algoritmos de controle também se baseiam em modelos do processo a ser controlado, são os controladores com modelo interno. Para a especificação de um problema mais realista o modelo utilizado no controle possui algumas diferenças em relação àquele que representa a planta real. Assim, faz parte do estudo de sistemas controlados a sua representação na forma de modelos. Esta seção apresenta o conjunto de funções disponíveis no Scilab para representar modelos lineares SISO (single-input single-output) e MIMO (multiple-input multiple-output) nas suas várias formas de representação.

O Scilab suporta as seguintes representações de modelos:

1. Espaço de Estados
   - Sistemas contínuos no tempo
   - Sistemas discretos no tempo
2. Funções de Transferências
   - Domínio de Laplace
   - Domínio da Transformada Z
3. Polos.
4. freqüência.

A definição de um sistema linear no Scilab é feita com a função **syslin**, que define um sistema linear como uma lista (**list**) e verifica a consistência de dados. Sistemas lineares definidos como **syslin** podem ser manipulados com as operações usuais existentes para as matrizes (concatenação, extração, transposta, multiplicação etc) sejam nas representações no espaço de estado, sejam na representação de funções de transferência. A maioria das funções no espaço de estado recebem uma lista **syslin** como entrada ao invés das matrizes A, B, C e D que definem o sistema.

$$\frac{d\mathbf{x}}{dt} = \mathbf{Ax} + \mathbf{Bu} \tag{5.1}$$

$$\mathbf{y} = \mathbf{Cx} + \mathbf{Du} \tag{5.2}$$

$$\mathbf{x}(0) = \mathbf{x}_0 \tag{5.3}$$

A sintaxe da função **syslin** é dada por:

chamada da função	forma da lista de saída
`[sl]=syslin(dom,A,B,C [,D [,x0]])`	`sl=tlist(['lss','A','B','C','D','X0','dt'],A, B,C,D,x0,dom)`
`[sl]=syslin(dom,N,D)`	`sl=tlist(['r','num','den','dt'],N,D,dom)`
`[sl]=syslin(dom,H)`	`sl=tlist(['r','num','den','dt'],H(2),H(3),dom)`

Com os parâmetros:

dom	string de caracter ('c', 'd'), [] or ainda um escalar dom='c' representa sistemas contínuos no tempo dom='d' representa sistemas discretos no tempo dom=n para sistema em tempo amostrado, período de amostragem igual a n dom=[ ] se o domínio do tempo é indefinido
A,B,C,D	matrizes da representação no espaço de estados D é opcional tendo como valor *default* a matriz nula D para sistemas impróprios é uma matriz polinomial
x0	vetor de estado inicial (*default* tem valor 0)
N, D	matrizes polinomiais
H	matriz racional ou representação linear no espaço de estados
sl	tlist ('syslin' list) representando o sistema linear

Exemplos

```
-- >A=[0,1;0,0];
-- >B=[1;1];
-- >C=[1,1];
-- >S1=syslin('c',A,B,C)
S1 =


    S1(1) (state-space system:)
!lss A B C D X0 dt !
S1(2) = A matrix =
! 0.   1. !
! 0.   0. !
S1(3) = B matrix =
! 1. !
! 1. !
S1(4) = C matrix =
! 1.   1. !
S1(5) = D matrix =
0.
S1(6) = X0 (initial state) =
! 0. !
! 0. !
S1(7) = Time domain =
c
-- >S1('A')
ans =
! 0. 1. !
! 0. 0. !
-- >S1('X0'), S1('dt')
ans =
! 0. !
! 0. !
ans =
c
-- >s=poly(0,'s');
-- >D=s;
-- >S2=syslin('c',A,B,C,D)
S1 =


    S1(1) (state-space system:)
!lss A B C D X0 dt !
S1(2) = A matrix =
! 0.   1. !
! 0.   0. !
S1(3) = B matrix =
! 1. !
! 1. !
```

```
S1(4) = C matrix =
! 1.   1. !
S1(5) = D matrix =
s
S1(6) = X0 (initial state) =
! 0. !
! 0. !
S1(7) = Time domain =
c
-- >H1=(1+2*s)/s^2,
H1 =
   1 + 2s
   ------
      2
     s
-- >S1bis=syslin('c',H1)
S1bis =
   1 + 2s
   ------
      2
     s
-- >H2=(1+2*s+s^3)/s^2,
H2 =
              3
    1 + 2s + s
    ----------
         2
        s
-- >S2bis=syslin('c',H2);
-- >S1+S2; // Soma os sistemas S1 e S2
-- >[S1,S2]; // Concatenação
-- >ss2tf(S1)-S1bis; // Transforma S1 e subtrai S1bis
-- >S1bis+S2bis;
-- >S1*S2bis;
-- >size(S1)
ans =
! 1. 1. !
```

Conforme visto no exemplo acima, a conversão de um sistema com representação no espaço de estado para função de transferência foi feita com a função **ss2tf**. O Scilab possui um conjunto de funções para conversão entre modelos na forma **syslin**(SS), função de transferência (TF), Matriz de FT (POL), matriz de sistema[25](SM) e forma descriptor (DES) entre outros.
As principais são:

[25] A Matriz de Sistema (SM) é definida por:

$$Sm = \begin{bmatrix} -sE + A & B \\ C & D \end{bmatrix}$$

ss2tf	SS para TF
tf2ss	TF para SS
tf2des	TF para DES
pol2des	POL para DES
ss2des	SS para DES
des2tf	DES para TF
frep2tf	Resposta frequencial para TF
imrep2ss	Resposta impulso para SS
markp2ss	Parâmetros de Markov para SS
sm2des	SM para DES
sm2ss	SM para SS
ss2ss	SS para SS, *feedback*, injeção

A tabela abaixo apresenta um conjunto de funções de conversão para manuseio de modelos de sistemas no Scilab que possuem uso freqüente,

[A,B,C,D]=abcd(sl)	retorna matrizes no espaço de estado para **syslin** sl
`[Ds,NUM,chi]=ss2tf(sl)`	retorna matriz polinomial do Numerador, polinômio característico chi e a parte polinomial Ds;
`h=ss2tf(sl)`	neste caso h=NUM/chi + Ds
`sl=tf2ss(h [,tol])`	conversão de função de transferência para syslin `h=C*(s*eye()-A)^-1*B+D(s)`

Só como ilustração, a discretização de sistemas contínuos para gerar os sistemas discretos no tempo pode ser implementada através das funções:

dscr	discretização de sistema linear
bilin	transformação bilinear

Mais detalhes sobre outras funções do Scilab e aspectos avançados da sua utilização em Controle de Processos, favor contactar o autor desse material para solicitar os próximos capítulos e as atualizações desse texto. Analogamente ao apresentado para as outras áreas de aplicação do Scilab, para os sistemas de controle existem uma grande variedade de pacotes já desenvolvidos e de funções. Entretanto, sempre será possível, o desenvolvimento de funções de uso pessoal para a criação de pacotes personalidades de funções (sejam pacotes de *scripts*, ou de códigos já compilados em C ou Fortran). O *script* a seguir é uma demonstração de como se pode avaliar a matriz de ganho de um sistema linear[26].

```
//-------------------------------------------------
// Compute low frequency (DC) gain of LTI systems
//-------------------------------------------------
// Luis Cláudio Oliveira Lopes,
// Uberlândia, UFU, Abril 2004
// version 0
function [K]=dcgain(varargin)
// K=dcgain(dom,G)
// K=dcgain(dom,A,B,C,D)
```

[26]Esses scripts apresentados aqui visam mais o fornecimento de um padrão de desenvolvimento dos *scripts* para o iniciante em Scilab que o desenvolvimento do assunto técnico a que se refere.

```
// Testar dom='d' com G, LCOL 20/04/2004 → comentários para posterior estudo
  dom=varargin(1);
  tm=length(varargin);
  if tm == 2 then
     sys=tf2ss(varargin(2));
    [A,B,C,D]=abcd(sys);
  end
  select dom
   case 'c' then
    K=D-C*inv(A)*B;
   case 'd' then
    K=D+C*inv(eye()-A)*B;
   else
    printf(' Error: domain must be ''c'' (continuous) or ''d'' (discrete).\n')
    abort
  end
endfunction
```

6 Conclusão

Espero que o leitor tenha aproveitado esse material para a formação dos aspectos básicos de programação em Scilab. Essa primeira parte abordou a criação de scripts utilizando um elenco básico de funções. A segunda parte desse material, ainda em desenvolvimento, conterá:

- **Introdução ao uso do Scicos**
- **Otimização Utilizando o Scilab**
- **Controle de Processos Utilizando o Scilab**
- **Utilizando C e Fortran no Scilab**
- **O Scilab em Ambiente Linux**
- **Criando Bibliotecas no Scilab**
- **Pacotes Específicos no Scilab**
    1. Controle Robusto
    2. Controle Nebuloso
    3. Controle Preditivo
    4. Identificação de Processos
    5. Redes Neuronais no Scilab
    6. Inequações de Matrizes Lineares (LMIs)
    7. Processamento de Sinais
    8. Introdução ao uso de Metanet

9. Arma e Simulação de Processos

- **Trabalhando com Sons no Scilab**
- **Traduções de linguagem e de Dados para e do Scilab**
- **Desenvolvendo GUI no Scilab: TCL/Tk**
- **Análise Estatística no Scilab**

Até a próxima!

RESUMO INCOMPLETO DAS FUNÇÕES DO SCILAB

Resumo Incompleto das Categorias de Funções disponíveis na distribuição padrão do Scilab 3.0.

1. Funções Elementares

abs - valor absoluto
addf - adição simbólica
acos - arco cosseno
acosm - matriz do arco cosseno
acosh - arco cosseno hiperbólico
acoshm - matriz do arco cosseno hiperbólico
asin - arco seno
asinh - arco seno hiperbólico
asinhm - matriz do arco seno hiperbólico
asinm - matriz do arco seno
atan - arco tangente no 2o. e 4o quadrantes
atanh - arco tangente hiperbólico
atanhm - matriz do arco tangente hiperbólico
atanm - matriz quadrada do arco tangente
besseli - função de Bessel Modificada do Primeiro Tipo e ordem α, **I**

besselj - função de Bessel Modificada do Primeiro Tipo e ordem α, **J**
besselk - função de Bessel Modificada do Segundo Tipo e ordem α, **K**
bessely - função de Bessel Modificada do Primeiro Tipo e ordem α, **Y**
binomial - probabilidades de distribuição binomial
bloc2exp - conversão de diagrama de bloco para expressão simbólica
bloc2ss - conversão de diagrama de bloco para espaço de estados
calerf - calcula função erro
ceil - arredondamento para cima
cmb_lin - combinação linear simbólica
cos - cosseno
cosh - cosseno hiperbólico
coshm - cosseno hiperbólico de matriz
cosm - cosseno de matriz
cotg - cotangente
coth - cotangente hiperbólico
cothm - cotangente hiperbólico de matriz
conj - conjugado
cumprod - produto acumulativo
cumsum - soma acumulativa
delip - integral elíptica
diag - matriz diagonal
diff - diferença e derivada discreta
dlgamma - derivada da função gammaln, função Ψ
double - conversão de inteiro para representação em dupla precisão
dsearch - busca dicotômica (binário)
erf - a função erro
erfc - a função erro complementar
erfcx - a função erro complementar escalonada
eval - avaliação de uma matriz de *strings*
eye - matriz identidade
fix - arredondamento para inteiro imediatamente menor
floor - arredondamento para baixo
frexp - separa números em ponto flutuante em expoente na base 2 e mantissa
full - conversão de matriz esparsa para matriz cheia
gamma - a função Γ
gammaln - o logaritmo da função Γ
gsort - ordenamento em ordem decrescente
imag - parte imaginária
imult - multiplicação de um número por i, $i = \sqrt{-1}$
int - parte inteira
integrate - integração por quadratura
interp - interpolação
interpln - interpolação linear
intersect - retorna um vetor com valores comuns entre dois vetores
intsplin - integração de dados experimentais por interpolação *spline*
inttrap - integração de dados experimentais por interpolação trapezoidal
isdef - verifica a existência de uma variável
isequal - comparação de objetos

isinf - verifica se possui valor "infinito"
isnan - verifica se existe valor "Not a Number"
isreal - verifica se uma variável é real ou complexa
kron - produto de Kronecker, (.*.)
ldivf - divisão simbólica a esquerda
lex_sort - ordenamento de linha de matriz lexicográfica
linspace - vetor linearmente espaçado
log - logaritmo natural
log10 - logaritmo em base 10
log2 - logaritmo em base 2
logm - logaritmo de matriz quadrada
logspace - vetor logaritimicamente espaçado
lstsize - retorna o número de elementos para *list*, *tlist* e *mlist*
max - máximo
maxi - máximo
min - mínimo
mini - mínimo
modulo - calcula o resto de uma divisão
pmodulo - calcula o resto positivo de uma divisão
mps2linpro - converte problema lp dado em formato MPS para o formato *linpro*
mtlb_sparse - converte matriz esparsa
mulf - multiplicação simbólica
ndims - número de dimensões de uma matriz
nearfloat - calcula o mais próximo sucessor ou antecessor de um número em ponto flutuante
nextpow2 - próxima maior potência de 2
nnz - número de elementos não nulos em uma matriz
norm - norma de matriz
number_properties - determina parâmetros de ponto flutuante
ones - matriz de 1's
pen2ea - conversão de *pencil* para **E**, **A**
pertrans - pertransposta
prod - produto
rand - gerador de números randômicos
rat - aproximação racional de ponto flutuante
rdivf - divisão simbólica a direita
real - parte real
round - arrendondamento
sign - função sinal
signm - função sinal de matriz
sin - seno
sinc - função seno complexa (sinc)
sinh - seno hiperbólico
sinhm - seno hiperbólico de matriz
sinm - seno de matriz
size - tamanho de objetos
smooth - suavização (smoothing) por funções *spline*
solve - resolve sistema linear simbólico
sort - ordenamento decrescente

sp2adj - converte matriz esparsa
sparse - definição de matriz esparsa
spcompack - converte um representação *compressed adjacency* para uma *standard adjacency*
speye - matriz identidade esparsa
spget - recebe elementos de matriz esparsa
splin - função *spline*
spones - matriz esparsa com 1 nas posições não nulas
sprand - matriz esparsa randômica
spzeros - matriz esparsa
sqrt - raiz quadrada
sqrtm - raiz quadrada de matriz
squarewave - gera uma onda quadrada com período 2π
ssprint - impressão elegante para sistemas lineares
ssrand - gerador randômico de sistemas
subf - subtração simbólica
sum - soma elementos de matrizes
sysconv - conversão de sistemas
sysdiag - conexão de sistema bloco-diagonal
syslin - definição de sistema linear
tan - tangente
tanh - tangente hiperbólico
tanhm - tangente hiperbólico de matriz
tanm - tangente de matriz
toeplitz - matriz toeplitz
trfmod - mostra pólos and zeros
trianfml - triangularização simbólica
tril - parte triangular inferior de matriz
trisolve - resolve simbolicamente sistema linear
triu - triangular superior
typeof - tipo do objeto
union - extrai união dos componentes de um vetor
unique - extrai componentes únicos de um vetor

2. Funções de Entrada-Saída

diary - diário da seção
disp - mostra variáveis
dispfiles - mostra propriedades de arquivos abertos
file - gerenciamento de arquivo
fileinfo - fornece informações sobre um arquivo
fprintf - emula a função fprintf da linguagem C
fprintfMat - imprime uma matriz em um arquivo
fscanf - leitura formatada de um arquivo
fscanfMat - ler uma matriz de um arquivo texto
getio - unidade lógica de entrada/saída para Scilab
input - pede entrada de informações via teclado
isdir - verifica se o argumento é é um diretório existente

lines - linhas e colunas usadas para apresentar na tela
load - carregar variáveis na memória
loadmatfile - carrega arquivo MAT do Matlab® 5 no Scilab
manedit - edita um item do manual
matfile2sci - converte um arquivo MAT do Matlab® 5 em um arquivo binário do Scilab
mclearerr - reinicializa erro de acesso em arquivo binário
mclose - fecha um arquivo aberto
meof - verifica se o final de um arquivo foi atingido
mfscanf - interface para a função fscanf do C
mscanf - interface para a função scanf do C
msscanf - interface para a função sscanf do C
mget - leitura de byte ou palavra em um dado formato binário e conversão para *double*
mgeti - leitura de byte ou palavra em um dado formato binário e conversão para *int*
mgetl - leitura de linhas de um arquivo ascii
mgetstr - leitura de uma *string* de caracteres
mopen - abre um arquivo
mfprintf - converte, formata e escreve dados em um arquivo
mprintf - converte, formata e escreve dados para a janela principal do Scilab
msprintf - converte, formata e escreve dados em uma *string*
mput - escreve byte ou palavra em um dado formato binário
mputl - escreve *strings* em um arquivo *ascii*
mputstr - escreve uma *string* de caracteres em um arquivo
mseek - fixa posição corrente em arquivo binário
mtell - gerenciamento de arquivo binário
newest - retorna mais recente arquivo de um conjunto de arquivos
oldload - carrega variáveis salvas na versão 2.4.1 ou versões anteriores do Scilab
oldsave - salva variáveis em formato da versão 2.4.1 ou versões anteriores do Scilab
print - imprime variáveis em um arquivo
printf - emula a função printf da linguagem C
printf_conversion - conversões de especificações das funções printf, sprintf e fprintf
read - leitura de matrices
read4b - leitura de arquivos binários do fortran
readb - leitura de arquivos binários do fortran
readc_ - leitura de uma *string* de caracteres
readmps - leitura de um arquivo em formato MPS
save - salva variáveis em arquivo binário
scanf - converte entrada formatada na entrada padrão
scanf_conversion - conversões de especificações das funções scanf, sscanf e fscanf
sprintf - emulador da função sprintf da linguagem C
sscanf - converte entrada formatada dado por uma *string*
startup - arquivo de inicialização. O arquivo **scilab.star** é arquivo padrão do Scilab e não deve ser modificado, para personalizar o Scilab pode-se criar um arquivo suplementar **.scilab** no diretório padrão. Esse arquivo (se existente) é automaticamente executado.
tk_getdir - janela para leitura de caminho de diretório
tk_getfile - janela para leitura de caminho de arquivo
warning - mensagens de aviso de atenção (*warning messages*)
writb - escreve arquivo binário do fortran
write - escreve em um arquivo formatado

write4b - escreve arquivo binário do fortran

3. Funções Gráficas

Graphics - apresentação da biblioteca gráfico; uso: **help Graphics**
Matplot - apresenta gráfico de uma matriz 2D usando cores
Matplot1 - apresenta gráfico de uma matriz 2D usando cores
Sfgrayplot - apresenta gráfico 2D suave de superfície definida por função usando cores
Sgrayplot - apresenta gráfico 2D suave de superfície definida por função usando cores
addcolor - adiciona novas cores ao mapa de cores corrente
agregation_properties - descrição de propriedades de entidade de agregação (*Agregation entity*)
alufunctions - descrição e número de pixels (modos gráficos)
arc_properties - descrição das propriedades da entidade *Arc*
axes_properties - descrição das propriedades dos eixos
axis_properties - descrição das propriedades dos eixos
black - Diagrama de Nichols
bode - Diagrama de Bode
champ - gráfico de campo vetorial 2D
champ1 - gráfico de campo vetorial 2D com vetores coloridos
champ_properties - descrição das propriedades de campo vetorial 2D
chart - Diagrama de Nichols
colormap - usando mapa de cores
contour - curvas de nível de uma superfície em um gráfico 3D
contour2d - curvas de nível de uma superfície em um gráfico 2D
contour2di - calcula as curvas de nível de uma superfície em um gráfico 2D
contourf - curvas de nível preenchidas de uma superfície em um gráfico 2D
copy - copia um gráfico
delete - apaga um gráfico e seus "filhos"
dragrect - arraste retângulo(s) com mouse
draw - faça um gráfico
drawaxis - trace um eixo
drawlater - faça eixos dos "filhos" invisíveis
drawnow - faça gráfico de entidades escondidas
driver - selecione um *driver* gráfico
edit_curv- editor interativo de curvas em gráficos
errbar - adicione barras de erros verticais em gráfico 2D
eval3d - valores de uma função em uma malha
eval3dp - calcula facetas de uma superfície paramétrica 3D
evans - Lugar das raízes (*Evans root locus*)
fac3d - gráfico 3D de uma superfície (**obsoleto**)
fchamp - campo de direção de uma Equação Diferencial Ordinária (EDO) 2D de primeira ordem
fcontour - curvas de nível de uma superfície 3D definida por uma função
fcontour2d - curvas de nível de uma superfície 2D definida por uma função
fec - gráfico pseudo-cor de uma função definida em uma malha triangular
fec_properties - descrição das propriedades da entidade **fec**
fgrayplot - gráfico 2D de uma superfície definida por uma função usando cores
figure_properties - descrição das propriedades da entidade gráfica *figure*
fplot2d - gráfico 2D de uma curva definida por uma função

fplot3d - gráfico 3D de uma superfície definida por uma função
fplot3d1 - curva de nível 3D cinza ou colorida de uma superfície
gainplot - gráfico de magnitude
genfac3d - calcula facetas de uma superfície 3D
geom3d - projeção de 3D em 2D após existência de gráfico 3D
get - recebe um valor da propriedade de um gráfico ou um objeto de interface com o usuário
getcolor - janela para selecionar cores no mapa de cors corrente
getfont - janela para selecionar fonte
getlinestyle - janela para selecionar estilo de linha
getmark - janela para selecionar símbolo
getsymbol - janela para selecionar um símbolo e seu tamanho
glue - junte um conjunto de gráficos em uma agregação
gr_menu - simples editor gráfico interativo
graduate - graduações de eixo (*pretty*)
graphics_entities - descrição das estruturas de dados de entidades gráficas
graycolormap - mapa de cores cinza linear
grayplot - gráfico 2D de uma superfície usando cores
grayplot_properties - descrição das propriedades da entidade **grayplot**
graypolarplot - gráfico polar 2D de uma superfície usando cores
hist3d - representação 3D de um histograma
histplot - construção de um histograma
hotcolormap - mapa de cores vermelho a amarelo
soview - fixa escala para gráfico isométrico (não muda o tamanho da janela)
legend_properties - descrição das propriedades de legendas
legends - construção de legendas para gráficos
loadplots - carrega e formata gráficos salvos
locate - seleção pelo *mouse* de um conjunto de pontos
m_circle - gráfico circular para ser usado em diagrama de Nyquist
milk_drop - função 3D (*milk drop*)
move - move, translaciona um gráfico e seus "filhos"
nf3d - facetas retangulares para parâmetros de **plot3d**
nyquist - diagrama de Nyquist
param3d - gráfico 3D de uma curva paramétrica
param3d1 - gráfico 3D de curvas paramétricas
param3d_properties - descrição das propriedades de curvas 3D
paramfplot2d - gráfico 2D animados, curva definida por uma função
patch_properties - descrição das propriedades do **Patch**
plot - faz gráfico
plot2d - faz gráfico 2D
plot2d1 - gráfico 2D (escala logarítmica) (obsoleto)
plot2d2 - gráfico 2D (função degrau)
plot2d3 - gráfico 2D (varras verticais)
plot2d4 - gráfico 2D (com estilo de setas)
plot3d - gráfico 3D de uma superfície
plot3d1 - gráfico 3D cinza ou colorido de curvas de nível de uma superfície
plot3d2 - gráfico de superfície definido por facetas retangulares
plot3d3 - gráfico com malhas (mesh) de superfícies definidas por facetas retangulares
plotframe - faz gráfico de *frame* com escala e *grids*

plzr - gráfico de pole-zero de sistema linear
polarplot - gráfico em coordenadas polares
polylin_properties - descrição de propriedades de **Polyline**
printing - imprime gráficos do Scilab
rectangle_properties - descrição das propriedades do **Rectangle**
replot - refaça gráfico da janela gráfica corrente com novas fronteiras
rotate - rotação de um conjunto de pontos
rubberbox - caixa Rubberband para seleção de retângulos
scaling - transformação *affine* de um conjunto de pontos
sd2sci - conversor de estrutura **gr_menu** para instruções do Scilab
secto3d - conversão de superfícies 3D
segs_properties - descrição das propriedades dos **Segments**
set - define um valor de propriedades de um objeto gráfico ou de interface
sgrid - linhas (grid) no plano-s
square - define escalas para gráficos isométricos (muda o tamanho da janela)
subplot - divide uma janela gráfica em uma matriz de sub-janelas
surface_properties - descrição das propriedades de entidades 3D
text_properties - descrição das propriedades de **Text**
title_properties - descrição das propriedades de **Title**
titlepage - adiciona uma título no meio da janela gráfica
unglue - desfaz uma agregação e as substitui por "filhos" individuais
winsid - retorna a lista de janelas gráficas
xarc - faça o gráfico de uma parte de uma elipse
xarcs - faça o gráfico de uma parte de um conjunto de elipses
xarrows - faça o gráfico de um conjunto de setas
xaxis - faça um eixo
xbasc - limpe uma janela gráfica e apague os gráficos associados gravados
xbasimp - mande gráficos para uma impressora Postscript ou para um arquivo
xbasr - refaça o gráfico de uma janela gráfica
xchange - transforme número real para coordenadas de pixel
xclea - apague um retângulo
xclear - limpe uma janela gráfica
xclick - aguarde por um *click* de *mouse*
xclip - defina uma zona de *clipping*
xdel - apague uma janela gráfica
xend - feche uma sessão gráfica
xfarc - preencha uma parte de uma elipse
xfarcs - preencha partes de um conjunto de elipses
xfpoly - preencha um polígono
xfpolys - preencha um conjunto de polígonos
xfrect - preencha um retângulo
xget - recebe valores correntes do contexto gráfico
xgetech - recebe a escala gráfica corrente
xgetmouse - recebe os eventos do *mouse* e posição corrente
xgraduate - graduação dos eixos
xgrid - adiciona um *grid* em um gráfico 2D
xinfo - faz uma *string* de informação na sub-janela de mensagem
xinit - inicialização de *driver* gráficos

xlfont - carrega uma fonte no contexto gráfico e coloca na fila a fonte carregada
xload - carrega uma gráfico salvo
xname - muda o nome da janela gráfica corrente
xnumb - traça números
xpause - suspende Scilab
xpoly - faz o gráfico de um polinômio (*polyline*) ou de um polígono
xpolys - faz o gráfico de um conjunto de polinômios (*polylines*) ou de polígonos
xrect - faz um retângulo
xrects - faz ou preenche um conjunto de retângulos
xrpoly - traça um polígono regular
xs2fig - manda gráfico para arquivo em formato do Xfig
xs2gif - manda gráfico para arquivo em formato GIF
xs2ppm - manda gráfico para arquivo em formato PPM
xs2ps - manda gráfico para arquivo em formato PS
xsave - salva gráfico para um arquivo
xsegs - traça segmentos desconectados
xselect - seleciona janela gráfica corrente
xset - define valores do contexto gráfico
xsetech - define a sub-janela de uma janela para fazer o gráfico
xsetm - janela para definir valores do contexto gráfico
xstring - apresenta *strings* em janela gráfica
xstringb - apresenta *strings* em uma caixa
xstringl - calcula caixa que englobe *strings*
xtape - fixa o processo de gravação para gráficos
xtitle - adiciona títulos em uma janela gráfica a e aos eixos X e Y
zgrid - linhas (grid) no plano-z

4. Funções da Álgebra Linear

aff2ab - função linear (*affine*) para conversão para **A**, **b**
balanc - balanceie (melhora condicionamento) de matriz ou *pencil*
bdiag - diagonalização em bloco, vetor característico (autovetor) generalizado
chfact - fatoração esparsa de Cholesky
chol - fatoração de Cholesky
chsolve - *solver* esparso de Cholesky
classmarkov - classes recorrente e transiente de matriz de Markov
coff - resolvente (método do cofator)
colcomp - compressão de coluna, núcleo (*kernel*) e *nullspace*
companion - matriz *companion*
cond - número de condicionamento
det - determinante
eigenmarkov - vetor característico (autovetor) normalizado de Markov a esquerda e direita
ereduc - calcula a forma de matriz *column echelon* por transformações QZ
exp - calcula exponencial de elemento
expm - calcula exponencial de matriz quadrada
fstair - calcula a forma de *pencil column echelon* por transformações QZ
fullrf - fatoração de posto completo (*full rank factorization*)

fullrfk - fatoração de posto completo de $A\hat{}k$
genmarkov - gera matriz randômica de Markov com classes recorrente e transiente
givens - retorna matriz unitária 2×2 que segue a transformação $U * xy = [r; 0] = c$, com $xy = [x; y]$
glever - inversa da matriz *pencil*
gschur - forma generalizada de Schur (obsoleto).
gspec - valor característico (autovalor) de matriz *pencil* (obsoleto)
hess - forma de Hessenberg
householder - matriz ortogonal de Householder
im_inv - inversa de imagem
inv - matriz inversa
kernel - kernel ou nullspace
kroneck - forma de matriz *pencil* de Kronecker
linsolve - resolve sistema de equações lineares
lsq - problemas de mínimos quadrados lineares
lu - fator LU da eliminação gaussiana
ludel - função de utilidade usada com **lufact**
lufact - fatoração esparsa LU
luget - extração de fatores LU esparsos
lusolve - resolve sistemas lineares esparsos
lyap - resolve equação de Lyapunov
nlev - algoritmo de Leverrier
orth - base ortogonal
pbig - projeção de subespaço associado com os valores característicos (*eigen-projection*)
pencan - forma canônica de matriz *pencil*
penlaur - coeficientes de Laurent de matriz *pencil*
pinv - calcula pseudo-inversa
polar - forma polar
proj - projeção
projspec - operadores espectrais
psmall - projeção espectral
qr - decomposição QR
quaskro - forma quasi-Kronecker
randpencil - *pencil* randômico
range - *range* (*span*) de $A\hat{}k$
rank - posto de matriz (*rank*)
rankqr - calcula fatoração QR que revela posto (*rank revealing QR factorization*)
rcond - número de condicionamento inverso (*inverse condition number*)
rowcomp - compressão de linha, *range*
rowshuff - algoritmo *shuffle*
rref - calcula a forma de matriz *row echelon* por transformação LU
schur - decomposição Schur (*ordered*) de matrizes e *pencils*
spaninter - intersecção de subespaços
spanplus - soma de subespaços
spantwo - soma e intersecção de subespaços
spchol - fatoração esparsa de Cholesky
spec - valores característicos (autovalores) de matrizes e *pencils*
sqroot - fatoração hermitiana W*W´
sva - aproximação de valor singular (*singular value approximation*)

svd - decomposição de valor singular (*singular value decomposition*)
sylv - equação de Sylvester

5. Funções para Desenvolvimento de Programas

abort - interrompe avaliação
ans - answer
\ - divisão de matriz à esquerda
bool2s - converte matriz booleana para matriz de zeros (*false*) e uns (*true*)
boolean - objetos do Scilab Objects, variáveis booleanas e operadores: $\&, |, \sim$
break - palavra para interromper *loops*
call - chamadas de rotinas em Fortran ou C
case - palavra usada em **select**
clear - apaga variável da memória
clearglobal - apaga variáveis globais da memória
// - comentários no Scilab, não são executados
date - data corrente em formato de *string*
debug - nível de *debugging*
definedfields - retorna índice campos definidos em **list**
else - palavra reservada em **if-then-else**
elseif - palavra reservada em **if-then-else**
[] - matriz vazia
end - palavra chave da linguagem do Scilab
errcatch - *trapping* erro
errclear - limpando erro
error - mensagens de erro
evstr - avaliação de expressões
exec - executa arquivo com *script* do Scilab
execstr - executa código Scilab em *strings*
exists - verifica a existência de variáveis
exit - finaliza a sessão corrente do Scilab
feval - avaliação múltipla
find - ache indices de matrizes ou vetores booleanos para elementos verdadeiros (*true*)
for - palavra reservada na linguagem para *loops*
format - formato de número para impressão e apresentação na tela
fort - chama rotinas de Fortran ou C
funptr - codificação de primitivos (*wizard*)
getdate - fornece informações de data e hora
getenv - fornece o valor de uma variável de ambiente
getfield - extração de campo de **list**
getpid - fornece identificador de processo do Scilab
getversion - fornece versão do Scilab
global - define variável global
gstacksize - fixa/recebe tamanho do *stack* global do Scilab
^ - exponenciação, também $**$
host - execução de comandos do Sistema Operacional
hypermat - inicializa uma matriz de dimensão N, ex. a(1,1,1,1:2)=[1 2]

iconvert - conversão para representação de inteiro de 1 a 4 bytes
ieee - define modo de exceção de ponto flutuante
intppty - define propriedades de argumentos de interface
inttype - retorna o tipo de representação de inteiro utilizada
inv_coeff - construa uma matriz polinomial como os seus coeficientes
iserror - testa ocorrência de erros
isglobal - verifica se uma variável é global
lasterror - fornece última mensagem de erro registrada
list - definição de objeto Scilab **list**
lsslist - definição de função Scilab linear no espaço de estados Scilab
lstcat - concatenação de lista
matrix - reforma um vetor ou matriz para uma dimensão diferente
mlist - objeto Scilab, definição de lista orientada para matriz
mode - seleciona um modo em arquivo de execução
mtlb_mode - troca para operadores análogos àqueles do Matlab®
null - apaga um elemento de uma lista
overloading - apresenta funções e operadores sobrecarregando capacidades
pause - modo de pausa, invoca teclado
poly - definição de polinômio
predef - proteção de variável
getcwd - recebe diretório corrente do Scilab
pwd - imprime diretório corrente do Scilab
quit - diminui um nível de pausa ou sai
′ - operador conjugado transposto, delimitador de *string*
resume - retorna ou volta a execução e copia algumas variáveis locais
return - retorna ou volta a execução e copia algumas variáveis locais
rlist - definição de fração de função racional no Scilab
sciargs - argumento de comando de linha do scilab
select - função para múltiplas opções de seleção, select-case, que é equivalente ao switch-case da linguagem C
setfield - inserção de campo em **list**
/ - divisão a direita e *feed back*
stacksize - define tamanho de pilha (*stack size*)para o Scilab
testmatrix - gera alguma matriz particular
then - palavra chave no comando **if-then-else**
tlist - definição de objeto *typed list* no Scilab
type - variável **type**
typename - associa um nome a variável **type**
user - faz interface com rotinas em Fortran ou C, verifique **intersci** para complementação
varn - variável simbólica de um polinômio
what - lista os primitivos no Scilab, ex. while, if, clear etc.
where - recebe instruções correntes da posição de execução
whereami - apresenta instruções correntes da posição de execução
whereis - nome de biblioteca contendo uma função
while - palavra chave de comando Scilab, para *loop* lógico
who - lista as variáveis na memória do Scilab
who_user - lista as variáveis do usuário

6. Funções para Otimização e Simulação

NDcost - função externa genérica para determinação do gradiente da função **optim** usando diferenças finitas
bvode - problema de valor no contorno para equações diferenciais ordinárias
dasrt - resolução de equações algébrico-diferenciais com determinação de superfície de cruzamento (*zero crossing*)
dassl - resolução de equações algébrico-diferenciais
datafit - identificação de parâmetros baseados em dados medidos
derivative - aproxima derivadas de uma função
fit_dat - identificação de parâmetros baseados em dados medidos
fsolve - determinação de solução de sistema de equações algébricas não lineares
impl - resolução de equações diferenciais lineares implícitas
int2d - integral definida 2D pelo método de quadratura e *cubature*
int3d - integral definida 3D pelo método de quadratura e *cubature*
intc - integral de Cauchy
intg - integral definida
intl - integral de Cauchy
karmarkar - algoritmo de Karmarkar
leastsq - resolução de problemas de mínimos quadrados não lineares
linpro - resolução de problemas de programação linear (LP)
lmisolver - resolução de LMI (linear matrix inequality)
lmitool - ferramentas para resolução de LMIs
numdiff - estimação numérica de gradiente
ode - resolução de equações diferenciais ordinárias (EDOs), problema de valor inicial (PVI)
ode_discrete - resolução de EDOs, simulação no domínio do tempo discreto
ode_root - resolução de EDOs com solução de raízes
odedc - resolução de EDOs discretas/contínuas
odeoptions - define opções para integradores de EDOs
optim - resolução do problema de otimização não linear
quapro - resolução do problema de programação linear quadrático
semidef - resolução do problema de programação semi-definida

7. Funções para Cálculo Polinomial

bezout - equação de Bezout para polinômios
clean - limpa matrizes (arredonda para zero valores muito pequenos)
cmndred - forma de denominador comum
coeff - coeficientes de matriz polinomial
coffg - inversa de matriz polinomial
colcompr - compressão de coluna de matriz polinomial
degree - grau de matriz polinomial
denom - denominador
derivat - derivada de matriz racional
determ - determinante de matriz polinomial
detr - determinante polinomial

diophant - equação de Diophantine (Bezout)
factors - fatoração numérica real
gcd - cálculo de gcd
hermit - forma de Hermite
horner - avaliação de polinômio ou matriz racional
hrmt - gcd de polinômios
htrianr - triangularização de matriz polinomial
invr - inversão de matriz (racional)
lcm - mínimo múltiplo comum (lcm, *least common multiple*)
lcmdiag - fatoração diagonal de lcm (*least common multiple*)
ldiv - *long division* para matriz polinomial
numer - numerador
pdiv - divisão polinomial
pol2des - matriz polinomial para forma *descriptor*
pol2str - conversão polinomial para *string*
polfact - fatores mínimos
residu - resíduo
roots - raízes de polinômios
routh_t - Tabela de Routh
rowcompr - compressão de linha de matriz polinomial
sfact - fatoração espectral no domínio do tempo discreto
simp - simplificação racional
simp_mode - modo de simplificação racional
sylm - matriz de Sylvester

8. Funções de Menus

addmenu - definição de botão interativo ou *menu*
delmenu - apaga botão interativo ou *menu*
getvalue - janela para aquisição de dados
halt - interrompe execução
havewindow - retorna ao modo de janela do Scilab
keyboard - comandos do teclado
seteventhandler - define um manipulador de evento (*event handler*) para a janela gráfica corrente
setmenu - ativação de botão interativo ou *menu*
unsetmenu - desativação de botão interativo ou *menu/submenu*
x_choices - janela de escolha interativa através de botões
x_choose - janela de escolha interativa
x_dialog - janela de menu
x_matrix - janela de menu de matrizes
x_mdialog - janela de menu com várias linhas
x_message - janela gráfica com mensagem
x_message_modeless - apresenta janela de mensagem

Cópia da Licença do Scilab

SCILAB License
****

1- Preface
****

The aim of this license is to lay down the conditions enabling you to use, modify and circulate the SOFTWARE. However, INRIA and ENPC remain the authors of the SOFTWARE and so retain property rights and the use of all ancillary rights.

2- Definitions
****

The SOFTWARE is defined as all successive versions of SCILAB software and their documentation that have been developed by INRIA and ENPC.

SCILAB DERIVED SOFTWARE is defined as all or part of the SOFTWARE that you have modified and/or translated and/or adapted.

SCILAB COMPOSITE SOFTWARE is defined as all or a part of the SOFTWARE that you have interfaced with a software, an application package or a toolbox of which you are owner or entitled beneficiary.

3- Object and conditions of the SOFTWARE license
****

a) INRIA and ENPC authorize you free of charge, to reproduce the SOFTWARE source and/or object code on any present and future support, without restriction, providing the following reference appears in all the copies: Scilab (c)INRIA-ENPC.

b) INRIA and ENPC authorize you free of charge to correct any bugs, carry out any modifications required for the porting of the SOFTWARE and to carry out any usual functional modification or correction, providing you insert a patch file or you indicate by any other equivalent means the nature and date of the modification or the correction, on the corresponding file(s) of the SOFTWARE.

c) INRIA and ENPC authorize you free of charge to use the SOFTWARE source and/or object code, without restriction, providing the following reference appears in all the copies: Scilab (c)INRIA-ENPC.

d) INRIA and ENPC authorize you free of charge to circulate and distribute, free of charge or for a fee, the SOFTWARE source and/or object code, including the SOFTWARE modified in accordance with

```
above-mentioned article 3 b), on any present and future support,
providing:

- the following reference appears in all the copies: Scilab
  (c)INRIA-ENPC.

- the SOFTWARE is circulated or distributed under the present license.

- patch files or files containing equivalent means indicating the
  nature and the date of the modification or the correction to the
  SOFTWARE file(s) concerned are freely circulated.

4- Object and conditions of the DERIVED SOFTWARE license
*******************************************************

a) INRIA and ENPC authorize you free of charge to reproduce and modify
and/or translate and/or adapt all or part of the source and/or the
object code of the SOFTWARE, providing a patch file indicating the
date and the nature of the modification and/or the translation and/or
the adaptation and the name of their author in the SOFTWARE file(s)
concerned is inserted. The SOFTWARE thus modified is defined as
DERIVED SOFTWARE. The INRIA authorizes you free of charge to use the
source and/or object code of the SOFTWARE, without restriction,
providing the following reference appears in all the copies: Scilab
(c)INRIA-ENPC.

b) INRIA and ENPC authorize you free of charge to use the SOFTWARE
source and/or object code modified according to article 4-a) above,
without restriction, providing the following reference appears in all
the copies: "Scilab inside (c)INRIA-ENPC".

c) The INRIA and the ENPC authorize you free of charge to circulate
and distribute for no charge, for non-commercial purposes the source
and/or object code of DERIVED SOFTWARE on any present and future
support, providing:

- the reference " Scilab inside (c)INRIA-ENPC  " is prominently
  mentioned;

- the DERIVED SOFTWARE is distributed under the present license;

- the recipients of the distribution can access the SOFTWARE  code
  source;

- the DERIVED SOFTWARE is distributed under a name other than SCILAB.

d) Any commercial use or circulation of the DERIVED SOFTWARE shall
have been previously authorized by INRIA and ENPC.
```

```
5- Object and conditions of the license concerning COMPOSITE SOFTWARE
**********************************************************************

a) INRIA and ENPC authorize you to reproduce and interface all or part
of the SOFTWARE with all or part of other software, application
packages or toolboxes of which you are owner or entitled beneficiary
in order to obtain COMPOSITE SOFTWARE.

b) INRIA and ENPC authorize you free, of charge, to use the SOFTWARE
source and/or object code included in the COMPOSITE SOFTWARE, without
restriction, providing the following statement appears in all the
copies: "composite software using Scilab (c)INRIA-ENPC functionality".

c) INRIA and ENPC authorize you, free of charge, to circulate and
distribute for no charge, for purposes other than commercial, the
source and/or object code of COMPOSITE SOFTWARE on any present and
future support, providing:

- the following reference is prominently mentioned: "composite
  software using Scilab (c)INRIA-ENPC functionality ";

- the SOFTWARE included in COMPOSITE SOFTWARE is distributed under the
  present license ;

- recipients of the distribution have access to the SOFTWARE  source
  code;

- the COMPOSITE SOFTWARE is distributed under a name other than
  SCILAB.

e) Any commercial use or distribution of COMPOSITE SOFTWARE shall have
been previously authorized by INRIA and ENPC.

6- Limitation of the warranty
*****************************

Except when mentioned otherwise in writing, the SOFTWARE is supplied
as is, with no explicit or implicit warranty, including warranties of
commercialization or adaptation. You assume all risks concerning the
quality or the effects of the SOFTWARE and its use. If the SOFTWARE is
defective, you will bear the costs of all required services,
corrections or repairs.

7- Consent
**********

When you access and use the SOFTWARE, you are presumed to be aware of
```

```
and to have accepted all the rights and obligations of the present
license.

8- Binding effect
*****************

This license has the binding value of a contract.
You are not responsible for respect of the license by a third party.

9- Applicable law
*****************

The present license and its effects are subject to French law and the
competent French courts.
```

Referências

de Assis, A. J. (2003). Métodos numéricos em engenharia química. Apostila de Curso de Métodos Numéricos. UFU/FEQUI/NUCOP.

Forbellone, A. L. V. (2000). *Lógica de programação : a construção de algoritmos e estruturas de dados*. Makron.

Guimarães, A. M. & Lages, N. A. C. (1994). *Algoritmos e Estruturas de Dados*. Livros Técnicos e Científicos Editora.

hitmill.com (2004). History of computers. http://www.hitmill.com/computers/computerhx1.html, visitado em 25 de setembro de 2004.

Scilab.Group (1998). Introduction to scilab: User´s guide. Disponível em http://scilabsoft.inria.fr/product/index_product.php?page=old_documentation.html.