UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

SCILAB: INTRODUÇÃO E APLICAÇÕES

XXVII SEMANA DA MATEMÁTICA

Profa. Dra. Heloisa Helena Marino Silva

Depto. de Matemática Aplicada - IBILCE/UNESP

e-mail: hsilva@ibilce.unesp.br

Diego Ferracini Bando

Pedro Vitor Jhum Haramoto

Sumário

1. Introdução

1.1. Apresentação

Scilab é um software gratuito opensource (código aberto) de programação numérica, oferece interface para as linguagens FORTRAN e C, além de ser um ambiente poderoso que permite a solução de problemas numéricos e a geração de gráficos bi e tridimensionais devido à sua rica coleção de funções matemáticas que estão em contínuo aperfeiçoamento em virtude da flexibilidade da comunidade opensource e suas contribuições para o meio acadêmico e científico.

Há inúmeras funções que são pequenos programas chamados de "*functions"*, agrupados em "*toolboxes"* que designaremos por bibliotecas. Além das bibliotecas nativas, há outras disponíveis gratuitamente, com destaque para ANN (Artificial Neural Network Toolbox), para redes neurais artificiais e aprendizado de máquinas, FISLAB(Fuzzy Logic Inference Toolbox), para lógica difusa, e o FRACLAB(Fractal, Multifractal and Wavelet Analysis Toolbox), para análise com fractais e wavelets.
O software dispõe de um manual em ficheiro PDF (Scilab Reference Manual) onde são identificadas as bibliotecas com suas respectivas funções.

As bibliotecas do Scilab podem ser vistas, com o software aberto, pressionando a tecla F1 ou digitando *help* no *prompt*, abrindo a ajuda do Scilab, que exibe as bibliotecas no painel de navegação situado à esquerda. Por exemplo, as funções de Álgebra Linear estão nas bibliotecas LINSPACK, EISPACK, LAPACK e BLAS; as funções para resolução de Equações Diferenciais nas bibliotecas ODEPACK e SLATEC; as funções de otimização na biblioteca MINPACK.

1.2. Ambiente Scilab

A tela inicial do Scilab é apresentada na Figura 1.

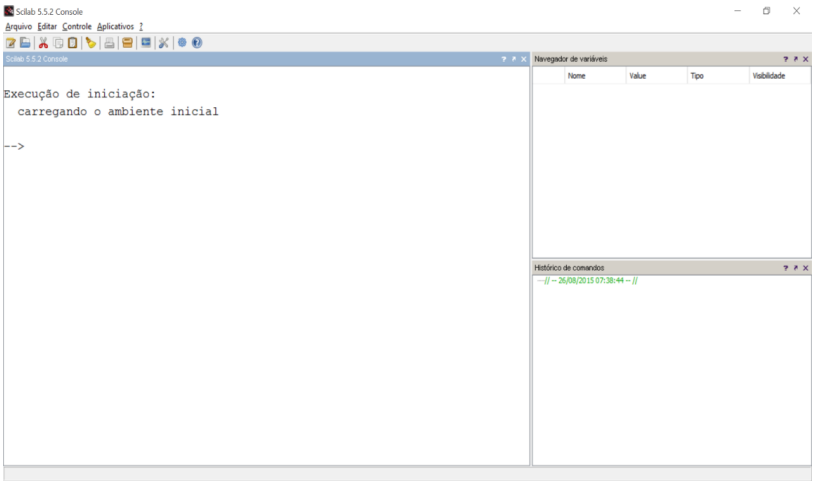

Figura 1: Tela inicial do Scilab

Observamos que o *prompt* do Scilab, na Figura 1, é representado por uma seta, -->. Este *prompt* é chamado de *prompt* de primeiro nível.

1.3. Matemática elementar

1.3.1. Expressões aritméticas

É disponivel no Scilab as operações aritméticas básicas vide a Tabela 1.

Operação	Operador	Exemplo
Adição	**+**	a + b
Subtração	**-**	a - b
Multiplicação	*****	a * b
Divisão	**/**	a / b

Potenciação	^	a ^ b

Tabela 1: Expressões aritméticas.

1.3.2. Ordem de precedência das operações aritméticas e lógicas

As operações matemáticas são realizadas de acordo com a prioridade, caso tenham a mesma prioridade irá valer a regra da associatividade. Podemos alterar a ordem de prioridade ao adicionar os parênteses, ou seja, o que estiver dentro do parênteses será executado primeiro, porém as expressões internas respeitam as regras de prioridade. Vejamos a Tabela 2 a seguir:

Prioridade	Operação	Associatividade
1ª	Potenciação	Da direita para a esquerda
2ª	Multiplicação e divisão	Da esquerda para a direita
3ª	Adição e subtração	Da esquerda para a direita

Tabela 2: Ordem de precedência.

1.4. Variáveis

1.4.1. Regras para a formação de nomes de variáveis

Em programação, define-se variável como uma posição na memória do computador. Podemos armazenar valores ou expressões em variáveis. Identificaremos as variáveis usando os caracteres alfanúmericos.

As variáveis devem seguir as seguintes regras:

a) As variáveis começam com letras, seguidas por letras, números ou sublinhados;
b) Caracteres especiais não são permitidos;
c) Caracteres acentuados não são permitidos;
d) As variáveis se diferem entre letras maiúsculas e minúsculas, ou seja X ≠ x.

Usaremos o operador = para atribuir valores às variáveis. Exemplo:

```
-->X=5;
-->Y=9;
-->Z=X+Y;
-->A=(Z+X)*5
A  =   95.
```

Obs.: O operador == verifica a igualdade entre 2 variáveis, retornando T para verdadeiro ou F para falso. O operador ; (ponto e vírgula) no final de um comando, não mostra o resultado.

1.4.2. Funções matemáticas comuns

Na tabela 3, vemos algumas das funções matemáticas do Scilab:

Função no Scilab	Função
abs(x)	Valor absoluto.
cos(x)	Cosseno.
acos(x)	Arco cosseno
acosh(x)	Arco cosseno hiperbólico.
sin(x)	Seno.
asin(x)	Arco seno.

asinh(x)	Arco seno hiperbólico.
tan(x)	Tangente.
atan(x)	Arco tangente.
atanh(x)	Arco tangente hiperbólico.
exp(x)	Exponencial e^x.
real(x)	Parte real de um número complexo.
imag(x)	Parte imaginária de um número complexo.
log(x)	Logaritmo natural.
log10(x)	Logaritmo na base 10.
sign(x)	Retorna o valor −1 ou +1 ou zero conforme o argumento de x, seja negativo, nulo ou positivo, respectivamente.
sqrt(x)	Raiz quadrada.
modulo(x,y)	Resto da divisão de x por y.

Tabela 3: Funções matemáticas.

1.4.3. Números complexos

Para usar variáveis complexas usaremos %i para diferenciar uma variável real de uma variável complexa. Veja os exemplos a seguir:

a)
```
-->z=5 -5*%i
z  =   5. -5.i
```
Logo, a parte real é 5 e a parte imaginária é -5.

b)
```
-->z=5 + 8*%i
z  =   5. + 8.i
```
Logo, a parte real é 5 e a parte imaginária é 8.

1.5. Exercícios

1. Resolva, tomando x=10, y=5, z=6, a=4, b=6 e c=x + bi:
   - **a)** x+y
   - **b)** (x+y)*z
   - **c)** modulo(2,3)
   - **d)** $\sqrt{a}$
   - **e)** cos(0) + sen(0)
   - **f)** Qual é a parte real e a parte imaginária de c?

2. Matrizes

2.1. Definição

Matrizes são variáveis bidimensionais, ou seja, são variáveis que possuem um único nome, porém dois índices e são individualizadas por tais índices.

2.2. Vetores

Assim como as matrizes, os vetores são variáveis de um único nome, porém estes são unidimensionais, ou seja, possuem apenas um índice.

Para criar vetores, faremos V=[5 3 7 4] ou V=[5,3,7,4] . Para acessar os elementos de um vetor faremos V(1), que é o elemento 5,V(2), que é o elemento 3.

Obs.: Podemos criar vetores usando operadores que criam sequências, como o operador dois pontos, por exemplo:

Início:Incremento:Fim, usando números, temos V=1:1:10, ou seja, a sequência terá o número 1 como primeiro elemento e o número 10 como último elemento do vetor V.

```
-->V=1:1:10
 V =  1.  2.  3.  4.  5.  6.  7.  8.  9.  10.
```

2.3. Construção de matrizes

Utilizaremos os colchetes para marcar o início e o fim de uma matriz, separaremos os elementos por vírgulas ou espaços e para adicionar uma nova linha usaremos o ponto e vírgula. Exemplos:

1)
```
-->M=[5 8 9 ; 4, 0, 6 ; 1 5 9] //neste caso, a matriz M será 3x3.
M  =
   5.  8.  9.
   4.  0.  6.
   1.  5.  9.
```

2)
```
-->N=[0:5:10 ; 5 9 7; 1:1:3] //observe que podemos utilizar o operador dois pontos também.
N  =
   0.  5.  10.
   5.  9.  7.
   1.  2.  3.
```

2.4. Função aplicada à matrizes

As funções (sin(), cos(), etc) são definidas para receber valores e retornar valores, mas podem receber matrizes ou vetores como parâmetros, assim a função irá retornar um valor para cada elemento da matriz ou vetor. Por exemplo:

Vamos criar a matriz A=[0 30 60 90; 45 180 270 360] e usar a função *cos()* e como parâmetro a matriz A, ou seja, *cos(A) e*ssa função retornará, para cada elemento da matriz, um valor. Veja a Figura 2.

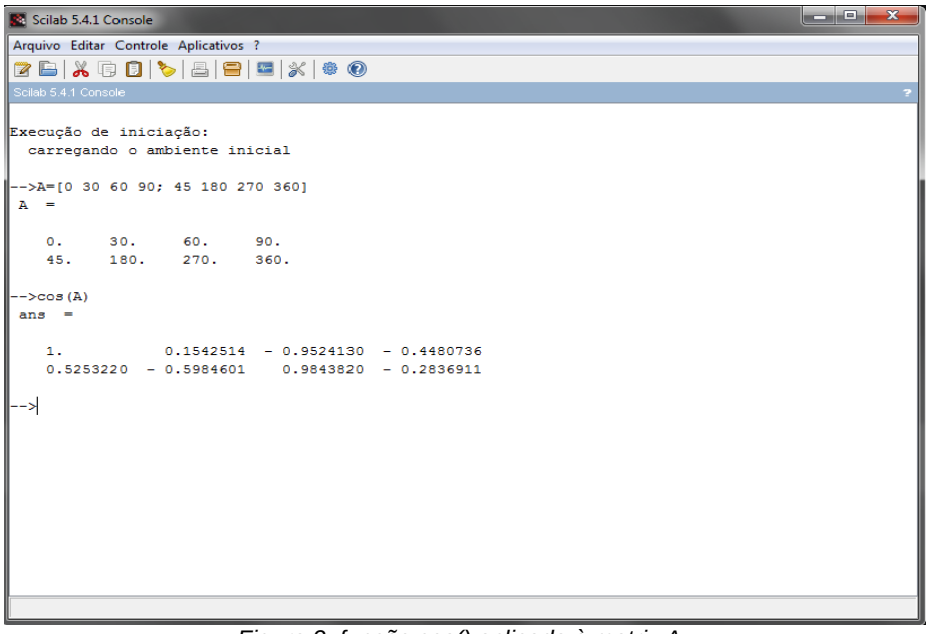

Figura 2: função cos() aplicada à matriz A.

Obs.: O *Scilab* está configurado para trabalhar em radianos e não em graus.

2.5. Operações com matrizes

Podemos operar com as matrizes como, por exemplo, somar, concatenar duas matrizes, etc. Na tabela 4, a seguir, apresentamos algumas destas operações:

Operador	**Operação**	**Uso**
'(apóstrofo)	O operador ' (apóstrofo) irá retornar a matriz transposta.	A'
[a ; b] ou [a b]	Intuitivamente, podemos criar 2 matrizes(de mesmas dimensões) e concatená-las, como se fossem elementos.	[A ; B] ou [A B]
: (dois pontos)	Retorna um vetor construído pelas colunas da matriz.	A(:)
$ (cifrão)	O operador *$* (cifrão) retorna o último elemento da matriz.	A($)
+	O operador + irá somar as matrizes.	A+B
.*	O operador .* irá multiplicar elemento por elemento.	A.*B
./	O operador ./ irá dividir elemento por elemento.	A./B
.^	O operador .^ irá realizar a potenciação elemento por elemento.	A.^B
det()	Retorna o determinante da matriz.	det(a)
eye()	Retorna uma matriz identidade.	eye(a)
inv()	Retorna a matriz inversa.	inv(a)
ones()	Retorna uma matriz cujos elementos são o número 1.	ones(a)
zeros()	Retorna uma matriz nula.	zeros(a)
size()	Retorna a quantidade de linhas e colunas da matriz. O parâmetro <*M*> é a matriz propriamente dita, já o parâmetro <*o*> é adicionado se quisermos que seja exibido apenas a quantidade de linhas ou colunas, usaremos *"r"* para linhas ou *"c"* para colunas.	size*(<M>,<o>)*
diag()	Retorna a diagonal de uma matriz	diag(a)

tril()	Retorna uma matriz triangular inferior	tril(a)
triu()	Retorna uma matriz triangular superior	triu(a)

Tabela 4: Funções para matrizes.

Obs.: Lembre-se que as regras matemáticas devem ser respeitadas, por exemplo, as matrizes A e B devem ter as mesmas dimensões para que sejam somadas,subtraídas, etc.

2.6. Hipermatrizes

Hipermatrizes são matrizes com mais de duas dimensões. Iremos apenas dar um exemplo de como criá-las. A função *hypermat(D,V)* irá nos retornar uma matriz cujas dimensões são dadas pelo vetor D e as entradas pelo argumento V. Exemplo de uma hipermatriz M com dimensões 2x3x4, isto é, composta por 4 submatrizes de dimensões 2x3:

```
-->M = hypermat([2 3 4],1:24)
 M  =
(:,:,1)
   1.   3.   5.
   2.   4.   6.

(:,:,2)
   7.   9.   11.
   8.   10.  12.

(:,:,3)
   13.  15.  17.
   14.  16.  18.

(:,:,4)
   19.  21.  23.
   20.  22.  24.
```

Obs.: O produto dos elementos do vetor *D* deverá ser igual ao número de elementos da matriz *V*.

2.7. Exercícios

1) Dadas as matrizes A = $\begin{pmatrix} 0 & 3 \\ 2 & -5 \end{pmatrix}$, B = $\begin{pmatrix} 0 & 3 \\ 2 & -5 \end{pmatrix}$ e C = $\begin{pmatrix} 1 & 4 \\ 2 & 6 \end{pmatrix}$, executar as seguintes operações:

- **a)** (A+B)+C.
- **b)** (A+C)+B.
- **c)** A*B.
- **d)** 2*A.
- **e)** A^t.
- **f)** $A + A^t$.
- **g)** Determine a inversa de C.
- **h)** Calcule o determinante de C.

3. Polinômios

3.1. Definição

Usando a função *poly(<parâmetros>,<variável>,<modo>)*, podemos criar polinômios de duas formas: especificando suas raízes ou coeficientes. Vejamos as sintaxes:

a) Especificando as raízes:

Ao definir o *<modo>='roots'* ou deixando esse parâmetro em branco, *<parâmetros>=V*, onde V é um vetor, o polinômio será construído através de suas raízes. Exemplo:

```
-->V=[-1 1] //vetor para as raízes do polinômio
        V  =
        - 1.    1.
-->R=poly(V,'x','roots')
        R  = x^2  - 1
```

O polinômio R(x) = $x^2 - 1$ será criado. Lembrando que em <*variável*> colocamos apenas a variável que o polinômio irá mostrar, no caso x.

b) Especificando os coeficientes:

Ao definir *<modo>='coeff'*, iremos criar um polinômio através de seus coeficientes. Exemplo:

Utilizaremos o mesmo vetor V, para ver a diferença.

```
-->P=poly(V,'s','coef')
        P  =  - 1 + s
```

O polinômio P(s)=-1 +s será criado. Observe que mudamos para a variável *s*.

3.2. Avaliação de polinômios

A função *horner(<polinômio>,<pontos>)* avalia o polinômio, onde *<polinômio>* é o polinômio em questão e *<pontos>* são os pontos nos quais queremos avaliar o polinômio. Exemplo:

Utilizando o polinômio R criado anteriormente e criemos o vetor T=[0 5]. A função *horner()* irá avaliar o polinômio nos pontos 0 e 5, veja:

```
-->T=[0 5];
-->horner(R,T)
 ans  =
- 1.    24.
```

3.3. Operações com polinômios

Podemos operar polinômios subtraíndo-os, somando-os, etc. Vejamos com exemplos:

Criemos os polinômios:

```
-->A=poly([-1 0 1 4 5],'x','coef')
 A  = -1 + x^2  + 4x^3  + 5x^4
-->B=poly([1 4 0 5],'x','coef')
 B  = 1 + 4x + 5x^3
```

- **Adição e subtração**

Basta usar os operadores de adição(+) ou subtração(-) para somar ou subtrair os polinômios A(x) e B(x). Sendo assim, se C = A + B e D = A - B, obtemos:

C(x)= $5x^4 + 9x^3 + x^2 + 4x$ e D(x)= $5x^4 - 3x^3 + x^2 - 4x - 2$.

- **Multiplicação**

A multiplicação é feita utilizando o operador de multiplicação(*). Seja M = A*B, então:

M(x) = $25x^7 + 20x^6 + 25x^5 + 21x^4 + 3x^3 + x^2 - 4x - 1$.

- **Divisão**

Iremos utilizar a função *pdiv(<p1>,<p2)* para dividir os polinômios p1 e p2. Vamos criar os polinômios p1(x) e p2(x). Sabemos que ao dividir p1(x) por p2(x), teremos o polinômio quociente (iremos usar a variável q) e o polinômio resto (iremos usar a variável r) da divisão. Sendo assim, p1=p2*q + r. Vejamos essa divisão:

```
-->p1=poly([3 -4 4 5 2],'x','coef');
-->p2=poly([-3 2 1],'x','coef');
-->[r,q] = pdiv(p1,p2);
```

Ou seja, a divisão do polinômio p1(x) por p2(x), resultará no quociente q(x)= $2x^2 + x + 8$ com resto r(x) = $-17x + 27$.

Obs.: Se utilizarmos o operador da divisão(*/*), iremos criar um polinômio racional. Exemplo:

```
-->k=p1/p2;
```

k(x) = $\frac{(2x^4 + 5x^3 + 4x^2 - 4x + 3)}{x^2 + 2x - 3}$

- **Derivação**

Uma das funções mais úteis, para polinômios, é a função *derivat(P),* na qual P é um polinômio ou uma matriz polinomial. Essa função realiza a derivação de um polinômio. Vejamos um exemplo.

Vamos utilizar o polinômio p1(x) do item acima:

```
-->dx=derivat(p1);
```

Ou seja, ao derivar o p1(x), temos dx(x) = $8x^3 + 15x^2 + 8x - 4$.

- **Cálculo de raízes**

Para calcular as raízes de um polinômio, usaremos a função *roots(P*), na qual P é um polinômio. Vejamos um exemplo:

```
-->pr=poly([-4 0 1],'x','coef');
-->roots(pr)
 ans  =
 2.
  - 2.
```

Logo, 2 e -2 são raízes de pr(x)=$-x^2 + 4$.

3.4. Exercícios

1) Forme um polinômio com as raízes 1, -3, i e -1

2) Determine as raizes dos seguintes polinômios:

a) A(x) = $3x^3 + 2x^2 - 7x + 2$
b) B(x) = $x^3 + 5x^2 - 18x - 72$
c) C(x) = $x^4 - 8x^3 - 32x + 15$
d) D(x) = $x^5 + 5x^4 + 6x^3 - 2x^2 - 7x - 3$

3) Crie uma matriz polinomial com os polinômios do exercício 2 e calcule o determinante.

4. Sistemas de equações algébricas lineares

4.1. Introdução

No Scilab, podemos resolver sistemas de equações algébricas lineares do tipo Ax=b, em que A é a matriz dos coeficientes das equações, x o vetor coluna das incógnitas e b o vetor dos termos independentes.

Considere o sistema:

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

Então,

$$A = \begin{pmatrix} 2 & 3 & -5 \\ 6 & -2 & 1 \\ 1 & 3 & -1 \end{pmatrix}$$

$$x = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$b = \begin{pmatrix} -7 \\ 5 \\ 4 \end{pmatrix}$$

4.2. Métodos para solução de sistemas lineares

- **Matriz inversa**

Vamos resolver o sistema acima utilizando a matriz inversa de A, para isso devemos:

1. Calcular o determinante da matriz A, se det(A) ≠ 0, então, o sistema é possível e determinado e, portanto, podemos resolver esse sistema.
2. Usando a função *Inv()*, faremos x = inv(A)*B para achar o vetor coluna x.

Então, vamos resolver o sistema S:

```
-->A=[2 3 -5;6 -2 1;1 3 -1];
-->b=[-7;5;4]; //observe que b deve ser um vetor coluna
-->det(A)
 ans  =  - 81.
-->x=inv(A)*b
 x  =
   1.
   2.
   3.
```

Logo, $x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$

- **Operador **

O operador (\) denota a divisão matricial direita-esquerda. Usaremos x=A\b, que é equivalente ao método anterior(x = inv(A)*b).

Considerando o mesmo sistema S, vamos resolver utilizando o operador \:

```
-->x1=A\b
 x1  =
          1.
          2.
          3.
```

Logo, $x1 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$, dessa maneira, não precisamos usar a função *inv()*.

- **Eliminação de Gauss - Jordan**

Usaremos a função *rref()*, para resolver o sistema pelo método de eliminação de Gauss - Jordan. A função aplica a decomposição LU à esquerda. Ou seja, precisamos concatenar o vetor coluna b a matriz A dos coeficientes.

Vamos usar o mesmo sistema S:

```
-->A=[2 3 -5; 6 -2 1; 1 3 -1];
-->b=[-7; 5 ; 4];
-->A2=[A b]
A2  =
2.    3.  - 5.  - 7.
6.  - 2.    1.    5.
1.    3.  - 1.    4.
```

Agora que temos a matriz A2, podemos usar a função *rref()*, que nos dará a matriz identidade concatenada à matriz solução. Vejamos:

```
-->x3=rref(A2)
x3  =
1.    0.    0.    1.
0.    1.    0.    2.
0.    0.    1.    3.
```

A última coluna é a solução do sistema.

4.3. Exercícios

1) Resolva abaixo o sistema utilizando o método da matriz inversa.

$$\begin{cases} 2x + y = 5 \\ x - 3y = 0 \end{cases}$$

2) Resolva abaixo o sistema utilizando o operador \.

$$\begin{cases} x + 2y - z = 2 \\ 2x - y + 3z = 9 \\ 3x + 3y - 2z = 3 \end{cases}$$

3) Resolva abaixo o sistema usando o método de eliminação de Gauss-Jordan.

$$\begin{cases} x + y - 10 = 0 \\ x - y - 5 \ = 0 \\ y - \ z - 3 \ = 0 \end{cases}$$

5. Programação

Programar é a arte de produzir códigos numa linguagem de programação como Fortran, C, C++, Java etc., a fim de controlar o computador para efetuar cálculos, processar dados ou controlar processos. Neste minicurso abordaremos um estudo básico sobre programação.

5.1. Funções de entrada e saída de dados

A entrada e saída de dados permite a interação entre o usuário e o Scilab. Usaremos as duas funções básicas *input()* e *printf()* para este fim.

- **Saída de dados com o *printf()***

Usamos o comando printf() para exibir dados e sua forma básica é: *printf(<texto>, <dados>)*, onde *<texto>* é uma string que contém o texto a ser exibido e, se necessário, descrevendo a forma como a lista de dados *<dados>* será exibida. Veja um exemplo simples:

```
-->printf( “Olá, mundo!” );
 Olá, mundo!
```

Usando agora a formatação de dados:

```
-->r = 1.45;
-->A = %pi * r ^ 2;
-->printf( “Raio = %g“ , r );
Raio = 1.45
```

Podemos ainda compor da seguinte forma:

```
-->printf( “A circunferência de raio igual a %g tem area igual a %g” , r , A );
A circunferência de raio igual a 1.45 tem area igual a 6.6052
```

Obs.:

a) O caractere %g indica para a função exibir um valor real.
b) Usaremos a função *disp()* para exibir matrizes de constantes, strings, etc.

- **Entrada de dados com o *input()***

Para a inserção de dados, usamos a forma básica do comando *input(): <variável> = input(<texto> , “string”)*, onde *<variável>* receberá um valor numérico digitado pelo usuário e *<texto>* uma string com a mensagem a ser apresentada na tela. *“string”* é opcional e informa que *<texto>* é uma cadeia de caracteres. Por exemplo:

```
 -->a = input( “a = “ );
-->b = input( “b = “ );
-->printf( “%g x %g = %g” , a , b , a * b );
```

5.2. SciNotes

O Scilab fornece o seu próprio editor de texto, o *SciNotes*, para criar ou editar arquivos de script e de funções. O editor é similar ao bloco de notas do Windows, porém com pequenos recursos de identação que facilitam a legibilidade do código. Para ativá-lo, clique nos respectivos menus da janela principal do Scilab: *Aplicativos->SciNotes* ou simplesmente clique no ícone  .

Usaremos esse editor para criar os nossos comandos de programação.

5.3. Arquivos de script

Um script é uma série de comandos interativos do Scilab listados juntos em um arquivo. Os comandos no Script são executados da mesma forma que seriam executados no console de primeiro nível. É importante salientar que um script não pode ser chamado de “programa”. Já as funções, por

outro lado, podem ser programas autosuficientes que requerem argumentos (variáveis de entradas) e, muitas vezes, uma atribuição a variáveis de saída.

Segue como um exercício prático:

```
printf( “Entre com as medidas de um trapézio\n” );
b = input( “Base menor = “ );
m = input( “Base maior = “ );
h = input( “Altura = “ );
A = 1/2 * ( m + b ) * h;
printf( “Área = %g” , A );
```

Salve como *“script1.sce”* e execute-o. A extensão padrão dos arquivos de script é *.sce*. Note que as barras duplas “//” possibilitam o incremento de comentários em uma linha. Os comentários são apenas úteis para documentar o código e facilitar o seu entendimento, portanto não interferem na execução do script.

Para executar um arquivo de script use o comando exec(arquivo_esp), onde o parâmetro *arquivo_esp* deve receber uma string descrevendo a localização do arquivo a ser aberto. Por exemplo, “c:\script1.sce”. Caso o diretório de trabalho atual seja o mesmo onde o arquivo de script está localizado, pode-se especificar na string apenas o nome do arquivo, por exemplo, “script1.sce”. Para saber a localização do diretório de trabalho atual, use o comando pwd().

5.4. Estruturas condicionais e de repetições

As estruturas condicionais permitem a tomada de decisão a partir de expressões lógicas, estabelecendo condições que controlam se um determinado trecho de código é executado ou não. As estruturas de repetição executam uma série de instruções repetidamente em loop e utilizam expressões lógicas como critério de parada e podem também ser chamadas de iterações. Vejamos as estruturas, mas antes observe a tabela abaixo com as expressões lógicas que serão muito úteis:

Expressão	Função
==	Verifica a igualdade.
~=	Verifica a desigualdade.
>=	Verifica se é maior ou igual.
<=	Verifica se é menor ou igual
>	Verifica se é maior.
<	Verifica se é menor
~	Negação de uma sentença.
& ou and	"E" lógico.
| ou or	"OU" lógico.

Tabela 5: Expressões lógicas.

- **Estrutura if ... else**

A estrutura *if … else* executa uma sentença somente se uma determinada condição for verdadeira, caso contrário, irá ignorar a sentença ou executará uma outra setença. A construção básica da estrutura é dada na forma:

```
if( <condição> )
   <comandos_1>
else
   <comandos_2>
end;
```

onde <condição> é uma expressão lógica e, se for verdadeira, a lista <comandos_1> será executada, caso contrário a lista <comandos_2> que será executada. Se não houver uma segunda lista de comandos a ser executada, a estrutura pode ser escrita da mesma forma sem a instrução *else*.

Exemplo:

```
nota1=input("digite a nota1:");
```

```
nota2=input("digite a nota 2:");
media=(nota1+nota2)/2;
if(media>=5)
      printf("aluno aprovado com media %g",media);
else
      printf("aluno reprovado com media %g",media);

end;
```

- **Estrutura while**

A estrutura *while* irá executar um certo bloco de comandos, enquanto uma condição for verdadeira. Exemplo:

```
A=0;
V=1;
while((A<10) & (V==1)) then
   A=A+1;
   if(A==4) then V=0; end;
end;
printf("A=%g",A);
```

- **Estrutura for**

A estrutura *for* é parecida com a *while*, o que difere entre elas é o critério de parada. A estrutura *while* é executa enquanto a condição posta for verdadeira, já a estrutura *for* é executada até a condição ser atingida. Exemplo:

```
-->A=0;
-->for(i=0:2:100) A=A+1; end;
-->A
A  =    50.
```

Ou seja, será acrescentado em A 1 unidade, até que i seja igual a 100.

5.5. Funções

No Scilab, temos um ambiente que nos permite criar funções. Isto pode ser feito utilizando o SciNotes.

Um arquivo de função é um arquivo de texto iniciado com uma instrução de função da forma:

function [<*variaveis de saída*>] = nome_da_funcao(<*argumentos da função*>)

onde <*variaveis de saída*> e <*argumentos da função*> podem ser uma lista com os nomes das variáveis representando valores escalares, vetores, matrizes ou strings. Na sequência da declaração de função, um arquivo de função pode conter inúmeras instruções do Scilab que executam operações nos argumentos da função ou nas variáveis de saída depois de terem sido avaliadas. Para carregar um arquivo de funções, use o comando exec().

Exemplo:

```
function y=fat(n)
   p = 1;
   for (i = n:-1:2)
         p = p*i;
   end
   y = p;
endfunction
```

5.6. Manipulando arquivos

No Scilab, podemos manipular arquivos, ou seja, podemos criar, deletar, modificar arquivos. É importante dizer que serão apresentados apenas os arquivos ASCII. Vejamos algumas funções para a manipulação de arquivos:

Função no Scilab	Função
uigetfile()	Permite a seleção de um arquivo. Abre uma janela para que o arquivo seja escolhido pelo usuário.
uigetdir()	Permite a seleção de um diretório. Abre uma janela para que o diretório seja escolhido pelo usuário.
mopen()	Esta função permite a abertura de um arquivo, caso o arquivo não exista, será criado.
mclose()	Esta função permite o fechamento de um arquivo.
mfscanf()	Esta função permite a leitura de informações contidas em um arquivo.
mfprintf()	Esta função permite a gravação de informações em um arquivo.
meof()	Esta função verifica se o fim do arquivo foi atingido.

Tabela 6: Funções para manipulação de arquivos.

Exemplo:

Temos o arquivo *notas.txt*(figura 3), no qual aparecem a notas e o RA dos alunos de uma classe, vamos abrir o arquivo, no Scilab, ler as informações, guardá-las em variáveis e escrever em um outro arquivo as informações lidas, porém iremos trocar a ordem, ou seja, vamos digitar o RA seguido da nota. Veja as imagens a seguir:

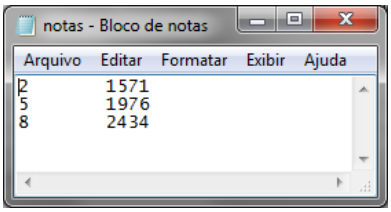

Figura 3: Arquivo a ser lido.

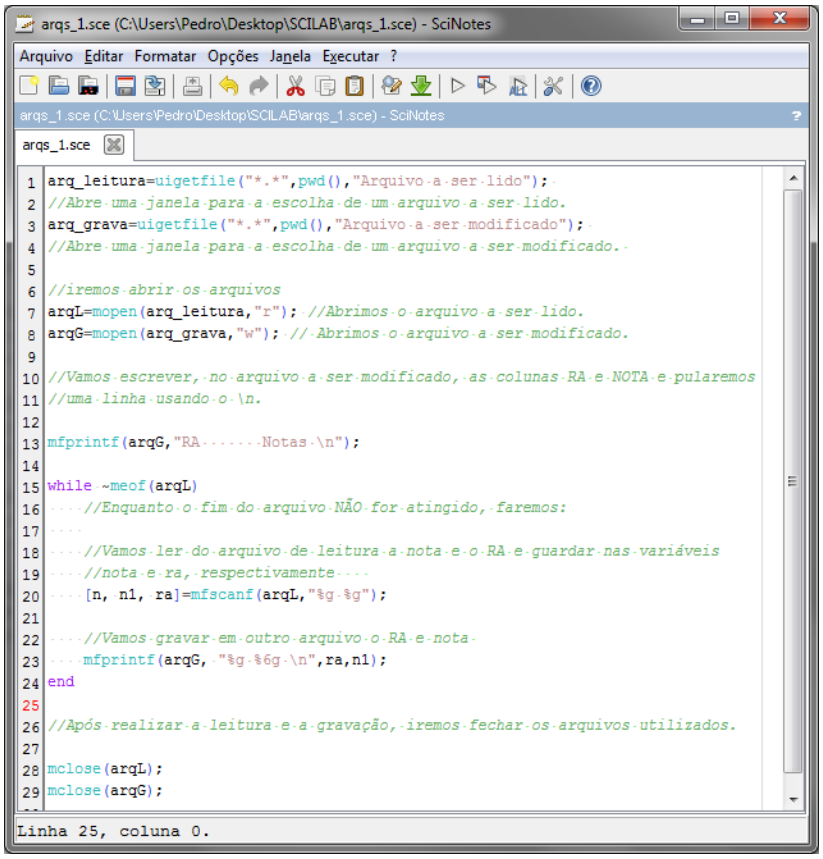

Figura 4: Algoritmo para o exemplo.

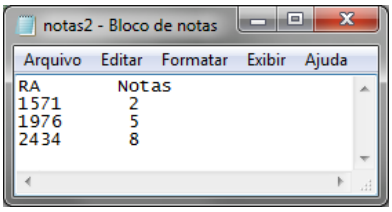

Figura 5: Arquivo modificado.

5.7. Exercícios

1) Faça um algoritmo, no SciNotes, para resolver o sistema abaixo. Use a função *disp()*, para exibir a solução.

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

2) Faça um algoritmo, no SciNotes, para calcular e gravar as raízes dos polinômios abaixo em um arquivo.

P(x) = $2x^2 - x - 1$
Q(x) = $x^2 + 1$

6. Gráficos

6.1. Gráficos bidimensionais

O Scilab fornece algumas funções para a produção de uma variedade de gráficos de duas ou três dimensões. Alguns exemplos de gráficos e suas aplicações são fornecidas neste capítulo, bem como alguns exemplos de animações com gráficos.

6.1.1. Função plot(): plot simples

Nas soluções de equações não lineares na forma f(x) = 0, muitas vezes é conveniente poder representar graficamente a função y = f(x) para visualizar a localização das raízes da função. O Scilab oferece o comando *plot* para obter tais gráficos. A forma básica do comando é:

plot(x , y)

em que x é um vetor que contém os valores para a coordenada x do gráfico e y é o vetor que contém os valores para a coordenada y do gráfico.

- **Ajustando títulos e eixos**

O Scilab permite que se crie títulos e adicione rótulos nos eixos dos gráficos. Veja os comandos:

- ***title()*** adiciona um título na parte superior do gráfico;
- ***xlabel()*** adiciona um rótulo no eixo das abscissas;
- ***ylabel()*** adiciona um rótulo no eixo das ordenadas.

Mais adiante estudaremos esses comandos detalhadamente.

No seguinte exemplo, nós primeiramente criaremos um vetor t com os valores (0.00 , 0.25 , … , 10.0):

```
-->t = ( 0 : 0.25 : 10 );
```

Em seguida, nós geramos o polinômio s = 2t2-2t+1 da seguinte forma:

```
-->s = 2 * t .^2 - 2 * t + 1;
```

Finalmente usamos o comando *plot* para traçar o gráfico de *s* em função de *t*, *xlabel* para escrever "***tempo (s)"*** no eixo das abscissas, *ylabel* para escrever ***"posição (m)"*** no das ordenadas e *title* para escrever "Movimento Linear" no título, da seguinte forma:

```
-->plot( t , s );
-->xlabel( "tempo (s)" );
-->ylabel( "posição (m)" );
-->title( "Movimento Linear" );
```

O Scilab irá gerar uma janela gráfica chamada ‘***"Janela gráfica número 0"*** mostrando o gráfico (veja na próxima página).

6.1.2. A janela gráfica do Scilab

Os seguintes menus estão disponíveis na janela gráfica do Scilab:

- *Arquivo*
  - *Nova figura:* abre uma nova janela gráfica em branco.
  - *Carregar...:* abre um ficheiro de um gráfico armazenado.
  - *Salvar...:* armazena o ficheiro do gráfico.
  - *Exportar para…:* armazena o gráfico em formatos de imagem e vetor gráfico.
  - *Copiar para a área de transferência:* copia a imagem do gráfico para uma memória temporária para ser, por exemplo, usado num editor gráfico ou de texto.
  - *Configurações de página:* acesso às opções de ajustes para impressão

- *Imprimir:* imprime o gráfico
- *Fechar:* fecha a janela gráfica do Scilab.

- *Ferramentas*
  - *Mostrar/esconder a barra de ferramentas:*
  - *Ampliar:*
  - *Ver original:*
  - *Rotação 2D/3D:*
- *Editar*
  - *Selecionar como figura atual:*
  - *Limpar figura:*
  - *Propriedades da figura:*
  - *Propriedades dos eixos:*
  - *Iniciar apanhador de entidades:*
  - *Terminar apanhador de entidades:*
  - *Iniciar gerenciador datatip:*
  - *Parar gerenciador datatip:*
  - *Start curve data modification:*
  - *Stop curve data modification:*
- *?*
  - *Ajuda do Scilab:*
  - *Sobre o Scilab:*

O título “Janela gráfica 0” indica que a janela gráfica criada tem o *número identificador (window-id)* igual a 0. Veremos que é possível trabalhar com duas ou mais janelas gráficas simultaneamente, e isso só será possível porque o Scilab permite que sejam manipuladas através de seus *números identificadores* conforme a necessidade.

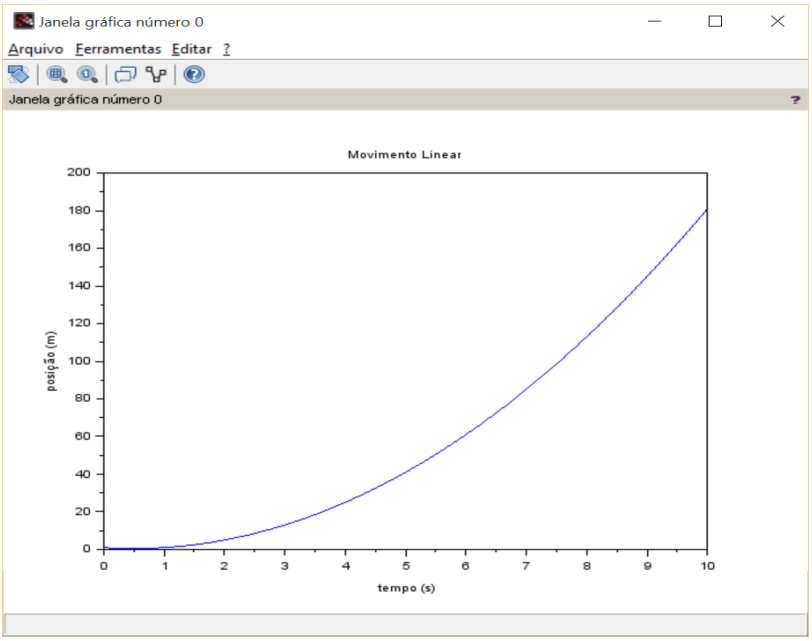

Figura 6: janela gráfica do Scilab

6.1.3. Ajustando as legendas com os comandos xlabel(), ylabel() e title()

Foi visto anteriormente o uso dos comandos básicos de xlabel, ylabel e title. Agora estudaremos um pouco mais desses comandos.

```
xlabel(, leg , [ nome_da_propriedade , valor_da_propriedade , ... ] )
ylabel( leg , [ nome_da_propriedade , valor_da_propriedade , ... ] )
title( leg , [ nome_da_propriedade , valor_da_propriedade , ... ] )
```

Os comandos xlabel e ylabel têm os mesmos parâmetros e propriedades. O parâmetro *leg* deve receber uma string contendo o texto e os parâmetros *nome_da_propriedade* e *valor_da_propriedade*, sendos estes dois últimos opcionais, devem receber uma string com nome da propriedade e o valor dela a ser alterado, respectivamente. O comando title possui todas as propriedades de ambos, porém com uma delas de nome distinto (*backgroundcolor*), como será visto adiante.

Segue um exemplo:

```
-->x = ( 0 : 0.1 : 7 ); y = sin( x );
-->xlabel( “x” , “font_size” , 5 );
-->ylabel( “y” , “font_size” , 5 );
-->title( “Senoide” , “font_size” , 5 );
```

A propriedade *font_size* dos comandos recebeu um atributo igual a 5, e isso fez com que o tamanho dos textos exibidos pelas funções ficassem com o tamanho igual a 5.

Segue abaixo todas as propriedades disponíveis para x-ylabel e title:

- “visible”: visibilidade da legenda. Recebe uma string “on” para deixar o texto visível ou “off” para invisível. O padrão desta propriedade é “on”.
- “font_size” ou” fontsize”: altera tamanho da fonte do texto (como visto no exemplo anterior). Recebe um valor inteiro positivo para definir o seu tamanho.
- “fontname”: altera o nome da fonte do texto. Recebe uma string com o nome da fonte(case unsensitive) ou um *id-number* (número identificador) da fonte. A saber:

Courier	0
Symbol	1
Times	2
Times Italic	3
Times Bold	4
Times Bold Italic	5
Helvetica	6
Helvetica Italic	7
Helvetica Bold	8
Helvetica Bold Italic	9

- “rotation”: rotaciona o texto para um valor em graus recebido.
- “position”: altera a posição do texto. Recebe um vetor de duas dimensões com as coordenadas da posição a ser alterada.
- “color”: altera a cor do texto. Recebe uma string (“red”, "green","blue","cyan","magenta","yellow","black","white") ou uma matriz 1x3 RGB.
- “edgecolor”: altera a cor da linha em torno do texto. Recebe os mesmos argumentos que a propriedade *color*
- “background” (apenas para x-ylabel): altera a cor do fundo em torno do texto.
- “backgroundcolor” (apenas para title): altera a cor de fundo em torno do texto.

6.1.4. Outros comandos para ajustes gráficos

No Scilab há alguns comandos que fornecem resultados distintos para criar a janela gráfica, mas para questões didáticas exemplificamos o comando *show_window, que cria uma janela gráfica “em branco”:*

```
-->show_window();
```

Se nenhuma janela gráfica estiver aberta, será criado uma nova janela gráfica com o menor número identificador *(window-id)* não criado. Caso contrário, o comando apenas restaura a janela gráfica.

Caso seja conveniente criar janelas gráficas com números identificadores *(window-id)* específicos, o parâmetro da função *show_window* recebe um número inteiro para especificar o número da janela gráfica. Também é possível fazer o mesmo com a função scf.

Segue um exemplo:

```
-->show_window( 5 );
-->scf( 8 );
```

Será criada uma janela gráfica com o número igual a 5 e outra de número igual a 8.

Este exercício mostra como produzir 4 gráficos diferentes em 4 janelas diferentes:

```
-->x = ( -10 : 0.1 : 10 ); y = sin( 2 .* x );
-->z = cos( 2 .* x ) - 1; w = 1 / ( x + 1 ); t = abs( x );
-->scf(1); plot( x , y );
-->xlabel( “x” ); ylabel( “y” ); title( “Primeiro gráfico” );
-->scf(2); plot( x , z );
-->xlabel( “x” ); ylabel( “z” ); title( “Segundo gráfico” );
-->scf(3); plot( x , w );
-->xlabel( “x” ); ylabel( “w” ); title( “Terceiro gráfico” );
-->scf(4); plot( x , t );
-->xlabel( “x” ); ylabel( “t” ); title( “Quarto gráfico” );
```

Segue alguns comandos básicos para manipular a janela gráfica do Scilab:

- *clf(n)* ou *clf(n , “clear”):* limpa a janela gráfica, onde n é um valor inteiro associado ao número da janela (window-id). Se o parâmetro n não receber um argumento, o comando limpa a janela corrente.
- *clf(“reset” , n):* reseta as configurações da janela gráfica para os valores padrões, onde *n* é um valor inteiro associado ao número da janela (window-id). Caso não seja usado um parâmetro *n*, será resetado as configurações da janela gráfica corrente;
- *xdel(n): deleta as janelas gráficas indicadas pelo parâmetro n,* onde *n* é um vetor de inteiros ou um escalar inteiro cujo o valor deverá estar associado ao *número da janela (window-id)* ;
- *xsave(filename , n):* salva o gráfico de uma janela gráfica de número (inteiro) *n*, em um arquivo designado pela string filename.
- xload(filename , n): carrega os gráficos salvos num arquivo cujo o nome deverá ser o argumento da string *filename, e exibe numa janela gráfica de número (inteiro) n . Se n não for fornecido, a janela gráfica corrente será aberta.*
- *winsid()*: retorna a lista de janelas gráficas abertas como um vetor de inteiros, isto é, retorna os *números das janelas (window-id)*.

É interessante notar que, se usarmos o comando:

```
xdel( winsid() )
```

Isso fará com que todas as janelas gráficas do Scilab abertas sejam fechadas.

6.1.5. Modificando parâmetros locais e globais de um plot

Parâmetros globais de um gráfico referem-se às opções a serem usadas num gráfico tais como, por exemplo, o tipo e o tamanho da fonte usadas nas descrições, o tipo de marcas para plotar pontos, o mapa de cores, o número da janela a ser aberto ou criado, a posição de uma janela gráfica etc. Muitas dessas propriedades podem ser alteradas usando a função *set*, que recebe como parâmetros: manipulador da entidade, uma string da propriedade a ser ajustada e o valor a ser ajustado.

1) Mudando a cor de fundo

A cor de fundo da janela gráfica do Scilab é branca por padrão. É possível mudá-la através do comando:

```
set( gca() , "background" , color )
```

O parâmetro *color* deve ser um número inteiro não negativo.

Segue agora um exemplo para mudar a cor de fundo da figura de uma janela gráfica. Primeiramente, criaremos um vetor x com os valores (0, 0.1 , … , 10), geramos o polinômio y = x2, e então plotamos o gráfico de y em função de x:

```
-->x = ( 0 : 0.1 : 20 ); y = x .^ 2;
-->plot( x , y );
-->set( gca() , "background" , 12 );
```

gca() retorna como parâmetro o manipulador de eixos da janela gráfica corrente, sendo "background" a propriedade alterada para o valor 12, isto é, a cor de fundo *(background)* do gráfico foi alterada para um tom de azul claro. As cores disponíveis seguem na tabela abaixo com seus respectivos *color-id* (números identificadores):

0	preto	**11**	azul puro	**22**	roxo escuro
1	preto	**12**	azul bebê	**23**	roxo claro
2	azul escuro	**13**	verde escuro	**24**	marrom avermelhado escuro
3	verde claro	**14**	verde primavera	**25**	marrom avermelhado escuro
4	ciano	**15**	verde lima	**26**	marrom avermelhado
5	vermelho Claro	**16**	verde esmeralda	**27**	laranja escuro
6	magenta	**17**	azul esverdeado escuro	**28**	rosa claro
7	amarelo	**18**	azul esverdeado	**29**	rosa claro
8	branco	**19**	vermelho escuro	**30**	rosa
9	azul safira	**20**	vermelho escuro	**31**	rosa
10	azul royal	**21**	vermelho	**32**	amarelo escuro

Tabela : números identificadores

Essas são as cores básicas disponíveis no mapa de cores padrão. Mais adiante veremos como criar outro mapa de cores personalizado.

2) Mudando a cor de primeiro plano:

A cor de primeiro plano de um gráfico corresponde à cor das linhas das bordas do quadro (incluindo a dos eixos) do gráfico. A cor de primeiro plano pode ser mudada usando:

```
set( gca() , "foreground" , color )
```

Onde o parâmetro *color* recebe valores inteiros não negativos (cores básicas).
Para ver a mudança de cor de fundo e a de primeiro plano simultaneamente, tente o seguinte exercício:

```
-->x = (  -%pi : %pi / 100 : %pi ); y = sin( x );
-->set( gca() , "background" , 32 );
-->set( gca() , "foreground" , 13 );
-->plot( x , y  ); xlabel( "x" ); ylabel( "y" ); title( "Seno" );
```

Para listar as propriedades da janela gráfica atual, que podem ser alteradas, como o tamanho da fonte (*"font_size"*), estilo da fonte ("font_style"), espessura das linhas (*"thickness"*) e, como a cor de fundo (*"background"*) vista anteriormente, dentre outras, basta usar o comando:

```
-->gca();
```

Este comando retorna as propriedades da estrutura de dados *"Axes"* do gráfico corrente que definem as coordenadas, o esboço do eixo dos eixos do gráfico e os valores que permitem ajustar o gráfico.

3) Alterando o tamanho e estilo de fonte:

Você pode usar a função *set* para alterar o tamanho e o estilo da fonte da janela gráfica do Scilab através das propriedades *font_size* e *font_style*, respectivamente.
font_size corresponde aos tamanhos típicos usados em textos, enquanto *font_style* refere-se aos seguintes estilos de fontes:

0 Courier

1 Symbol

2 Times

3 *Times Italic*

4 **Times Bold**

5 ***Times Bold Italic***

Tente o seguinte exercício para ver as diferentes fontes:

```
-->x = ( 0.1 ; 0.1 : 20 ); y = sqrt( 1 + x .^ 2 );
-->for j = 0 : 5, set( gca() , “font_style” , j ), plot( x , y ), end;
```

Ao fazê-lo, será exibido 6 janelas gráficas com os 6 estilos de fontes diferentes.

4) Adicionando grades

A forma geral do comando para adicionar grades é dada por:

xgrid(color, thickness , style)

onde os parâmetros:

- ***color****:* ajusta a cor da grade e deve receber o número da cor (color-id);
- ***thickness****:* ajusta a espessura da grade e recebe um real ou uma matriz linha;
- ***style****:* ajusta o estilo da grade e recebe um inteiro ou uma matriz linha.

Segue o exercício:

```
-->x = ( 0 : 0.1 : 10 )';
-->y = 1.5 + 0.2 .* sin( x );
-->plot2d( x, y )
-->xgrid( 5 , 1 , 3 );
```

6.1.6. Função plot2d()

Esta função pode ser usada se você quiser plotar um ou mais gráficos bidimensionais. O comando básico plot2d é da forma:

plot2d(x , y)

onde x e y são duas matrizes de mesmo tamanho.

Segue alguns exemplos de simples aplicações do comando plot2d:

Primeiro, nós criamos os seguintes vetores linhas:

```
-->x = ( -2  .* %pi  : %pi / 100 : 2 .* %pi )
-->y = sin( 2 .* x );
-->z = cos( x );
-->w = exp( -abs( 0.1 .* x ) ) .* sin( x );
```

Em seguida usamos uma simples chamada da função *plot2d* para ver os resultados:

```
-->plot2d( x , y );
-->plot2d( x , z );
-->plot2d( x , w );
```

A janela gráfica padrão é a *Janela gráfica número 0*. A função *plot2d* coloca um plot em cima do outro na janela gráfica padrão. A projeção gráfica não é muito útil porque as escalas verticais são recalculadas quando um novo gráfico é adicionado. Mais adiante mostraremos como colocar mais de um plot em um único gráfico com uma escala vertical em comum.

- **Criando multiplos plots com o plot2d()**

Pode-se criar plots individuais em diferentes janelas gráficas usando, por exemplo, os seguintes comandos:

```
-->scf( 1 ); plot2d( x' , y' ); xtitle( "Gráfico 1" , "eixo x" , "eixo y" );
-->scf( 2 ); plot2d( x' , z' ); xtitle( "Gráfico 2" , "eixo x" , "eixo y" );
-->scf( 3 ); plot2d( x' , w' ); xtitle( "Gráfico 3" , "eixo x" , "eixo y" );
```

E você pode combinar os três gráficos em um único conjunto de eixos da seguinte forma:

```
-->scf( 4 ); plot2d( [ x' , x' , x' ] , [ y' , z' , w' ] );
-->xtitle( "Múltiplos esboços 1" , "eixo x" , "eixo y" );
```

O comando plot2d([x' , x' , x'] , [y' , z' , w']) indica ao Scilab que três gráficos serão criados juntos com os valores da abscissa do gráfico dado pelo conjunto [x ', x' , x'] e com valores das ordenadas pelo conjunto [y' , z' , w']. Quando produz-se um múltiplo plot dessa forma, os vetores devem ser vetores colunas, isto é, uma matriz com m elementos e uma única coluna. No caso desse exemplo, nós definimos os vetores x, y, z e w como vetores linhas, então foi preciso adicionar apóstofres (isto é, x', y', z', w') nas funções.
De forma alternativa, podemos criar vetores como vetores colunas do início:

```
-->x = ( -2  .* %pi  : %pi / 100 : 2 .* %pi )';
-->y = sin( 2 .* x ); z = cos( x ); w = exp( -abs( 0.1 .* x ) ) .* sin( x );
```

e produzir um plot combinado:

```
-->scf( 5 ); plot2d( [ x , x , x ] , [ y , z , w ] ); xtitle( "Gráfico 5" , …
-->"eixo x" , "eixo y" );
```

Note que cada um dos três esboços é mostrado com cores diferentes nos exemplos acima. A propósito, o comando xtitle foi usado para adicionar um título e rótulos aos eixos do gráfico. Esse comando pode ser usado por qualquer plot bi ou tridimensional do Scilab.

O próximo exemplo mostra como plotar mais de uma função usando uma mesma independente variável:

```
-->x = ( 0 : 0.1 : 10 )';
-->plot2d( [ x , x , x ] , [ sin( x ) cos( x ) abs( sin( x ) - cos( x ) ) ] );
```

Nesse caso, ao invés de criarmos vetores y, z e w, usados anteriormente, nós definimos as funções a serem plotadas passando o argumento dentro da função plot2d.

- **Alterando os estilos de linhas em múltiplos plots**

Até agora nós usamos o comando *plot2d* com apenas dois argumentos, isto é, os vetores das abscissas e das ordenadas dos esboços. Podemos usar argumentos adicionais na função que nos dê mais controle do resultado final do esboço. Por exemplo, o seguinte comando é usado para plotar três curvas simultaneamente:

```
-->plot2d( [ x , x , x ] , [ sin( x ) cos( x ) sin( x ) + cos( x ) ] , …
-->[ 10 20 30 ] );
```

O comando acima inclui três argumentos, além dos que descrevem os vetores que produzem o esboço. O vetor style = [10 20 30] é um argumento que deve representar uma cor ou estilo. Nesse exemplo foi atribuido aos gráficos as cores identificadas como 10, 20 e 30. Esses valores no vetor em questão, quando positivos de 1 a 32, faz a curva ser contínua com uma cor associada ao número. Quando os valores são negativos de -1 a -14, faz a curva ser traçada com marcadores associados ao seu número, como mostrado na figura abaixo:

Figura 7: Marcadores

- **Adicionando legendas ao gráfico**

O seguinte comando plota as mesmas funções anteriores, mas adiciona legenda ao gráfico para identificar as curvas:

```
-->plot2d( [ x , x , x ] , [ sin( x ) cos( x ) sin( x ) + cos( x ) ] , …
-->[ 10 , 20 , 30 ] , “100” , leg = “posição@velocidade@aceleração” );
```

O comando acima inclui o vetor [10 20 30] representando as cores (*style*) das três curvas, e a atribuição à string *leg* a *“posição@velocidade@aceleração”* que associará uma legenda a cada uma das três curvas, isto é: posição, velocidade e aceleração. O caractere @ usado nessa função separa as legendas para seus respectivos gráficos. As legendas adicionadas com esse comando são colocadas abaixo do eixo das abscissas.
Legendar o gráfico desta forma não é tão flexível e, caso seja necessário, use o comando *legend()*. De forma alternativa, usariamos da seguinte forma:

```
-->legend( “posição” , “velocidade” , “aceleração” , 3 );
```

Os parâmetros relativos às legendas das curvas recebem strings e o último parâmetro é relativo à posição da legenda que recebe um valor inteiro. As posições variam de 1 a 4 e -1 a -6. Ficará como exercício testar as posições das legendas. Caso não seja passado nenhum argumento para a posição, é necessário clicar com o mouse em cima do gráfico para indicar a posição.

- **Modificando as dimensões do quadro de plot**

Para alterar as dimensões do gráfico para esboçar a curva apenas no intervalo 0<x< e 0<y<1, tente o seguinte comando:

```
-->plot2d( [ x , x , x ] , [ sin( x ) cos( x ) sin( x ) + cos( x ) ] , …
-->[ 10 , 20 , 30 ] , “100” , leg = “posição@velocidade@aceleração” , …
--> rect = [ 0 0 %pi 1 );
```

O último argumento do comando mostrado acima inclui o as dimensões do gráfico como [xmin ymin xmax ymax] = [0 0 %pi 1], isto é, o esboço será mostrado nos intervalos entre os pontos (0,0) e (,1).

- **Personalizando os tiques do gráfico**

Para ajustar os tiques, usamos o comando da seguinte forma:

```
-->plot2d( [ x , x , x ] , [ sin( x ) cos( x ) sin( x ) + cos( x ) ] , …
-->[ 10 , 20 , 30 ] , “100” , leg = “posição@velocidade@aceleração” , …
-->rect = [ 0 0 %pi 1 ) , nax = [ 2 10 2 10 ];
```

Aqui o último argumento inclui a relação de tiques para os eixos como um vetor [nx Nx ny Ny] = [2 10 2 10], em que nx e ny fornecem o número de tiques principais, Nx e Ny subtiques entre dois tiques principais, relativos ao eixo x e y, respectivamente.

6.1.7. Outras funções de plots bidimensionais

Os comandos *plot* e *plot2d* produzem uma série de linhas contínuas (curva linear por partes) para que as curvas sejam plotadas. Se for preciso usar outros tipos de curvas, é possível usar os comandos *plot2d1, plot2d2, plot2d3* e o *plot2d4.* Os tipos de curvas produzidas por esses comandos são as seguintes:

plot2d1: curva linear por partes, mas com escalas logarítmicas (estará obsoleta nas próximas versões);
plot2d2: curva constante por partes (escalonada), isto é, plota em degraus;
plot2d3: barras verticais;

plot2d4: estilo de setas (usado com equações diferenciais ordinárias em um espaço de fase);

6.1.8. Criando sub-janelas com os comandos xsetech() e subplot()

Os comandos *xsetech()* e *subplot()* permitem mostrar sub-janelas de gráficos dentro de uma mesma janela.

- **Sub-janelas com o xsetech()**

A forma geral do comando é:

xsetech(wrect , [frect , logflag]);

onde *wrect = [x y w h]*, tal que (x,y) são coordenadas do canto superior do plot, sendo (0,0) o canto superior da janela, w é a largura do plot e h a altura. Os valores x, y, w, h são especificados usando proporção de altura ou largura da janela gráfica corrente, como na figura abaixo:

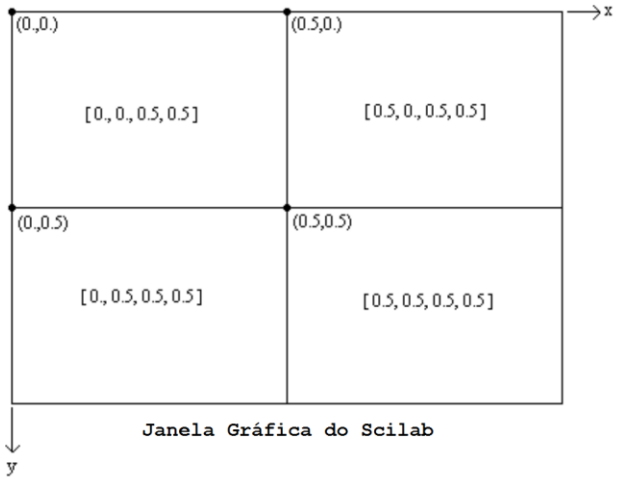

Figura 8: Janela gráfica do Scilab

frect é um vetor com 4 elementos e tem o mesmo fim do parâmetro *rect = [xmin ymin xmax ymax]* do comando *plot2d(); logfalg = "xy"* uma string, onde x e y podem ser "n" ou "l", isto é, escala normal ou escala logarítmica atribuidas aos eixos x e y, respectivamente.

O seguinte exemplo irá colocar quatro versões diferentes do mesmo plot numa única janela, copie em um arquivo de texto e execute-o como script:

```
// Script para plotar quatro versões diferentes de um gráfico numa única janela
x = ( 1 : 0.1 : 10 )'; z1 = 2.5 + 0.2 .* sin( x ); z2 = 3 + cos( x );
set( gca() , "font_size" , 3 );
clf();
xsetech( [ 0 , 0 , 0.5 , 0.5 ] , [ -1 , 1 , -1 , 1 ] );  //Quadrante superior esquerdo
plot2d( [ x, x ],[ z1 , z2 ],[ 4 10 ] );
xtitle( "Plot2d: linear por partes" );
xsetech( [ 0.5 , 0 , 0.5 , 0.5 ] );              //Quadrante superior direito
plot2d1( "oll" , x , [z1 , z2 ] , [ 3 11 ] );
xtitle( "Plot2d1: com escala logarítmica" );
xsetech( [0 , 0.5 , 0.5 , 0.5 ] , [ -1 , 1 , -1 , 1 ] ); //Quadrante inferior esquerdo
plot2d2( "onn" , x , [ z1 , z2 ] , [ 2 , 17 ] );
xtitle( "Plot2d2: partes constantes" );
xsetech( [ 0.5 , 0.5 , 0.5 , 0.5 ] , [ -1 , 1 , -1 , 1 ] ); //Inferior direito
plot2d3( "onn" , x , [ z1 , z2 ] , [ 12 22 ] );
xtitle( "Plot2d3: plot com barras verticais" );
```

Um segundo script exemplificando o uso do *xsetech()* para dividir o gráfico:

```
clf();
t = ( 0 : 0.05 : 1 )'; st = sin( 2 * %pi .* t );
xsetech( [ 0 , 0 , 1 , 0.5 ] );
```

```
plot2d2( “onn” , t , st );
xsetech( [ 0 , 0.5 , 1 , 0.5 );
plot2d3( “onn” , t , st );
```

- **Sub-janelas com o subplot()**

O *subplot()* é um comando mais simples que *xsetech()*, no entanto menos flexível. A grande vantagem em usá-lo ao invés do *xsetech()* é a possibilidade de separar os gráficos em matrizes de sub-janelas sem a necessidade de passar os valores das proporções das janelas para os parâmetros.

A forma geral do comando è:

```
-->subplot( mni );
```

onde *m* e *n* são os números de linhas e colunas, respectivamente, de uma matriz m por n, e *i* é a i-ésima sub-janela.

Segue abaixo um script que gera, usando o *subplot(),* o mesmo resultado reproduzido pelo pelo *xsetech()* no script anterior :

```
//Exibindo 4 gráficos distintos dentro de uma mesma janela com o subplot()
x = ( 1 : 0.1 : 10 )'; z1 = 2.5 + 0.2 .* sin( x ); z2 = 3 + cos( x );
set( gca() , “font_size” , 3 );
clf();
subplot( 221 );                 //Matriz 2x2, primeiro elemento: a1x1
plot2d( [ x, x ],[ z1 , z2 ],[ 4 10 ] );
xtitle( “Plot2d: linear por partes” );
subplot( 222 );                 //Matriz 2x2, segundo elemento: a1x2
plot2d1( “oll” , x , [z1 , z2 ] , [ 3 11 ] );
xtitle( “Plot2d1: com escala logarítmica” );
subplot( 221 );                 //Matriz 2x2, terceiro elemento: a2x1
plot2d2( “onn” , x , [ z1 , z2 ] , [ 2 , 17 ] );
xtitle( “Plot2d2: partes constantes” );
subplot( 221 );                 //Matriz 2x2, quarto elemento: a2x2
plot2d3( “onn” , x , [ z1 , z2 ] , [ 12 22 ] );
xtitle( “Plot2d3: plot com barras verticais” );
```

6.1.9. Criando animações

Para prosseguir é necessário apenas conhecer a ideia básica de animação. Considere um caso simples unidimensional para ilustrar a ideia do fundamento básico de uma animação. Se as posições do ponto são dadas por um intervalo de tempo fixo como $x = x_1, x_2, x_3, ...$ então a animação do ponto móvel pode ser feita da seguinte forma:

1. Apaga o ponto anterior (se existir);
2. Desenha o novo ponto em $x = x_i$;
3. i = i + 1.

Estendendo este conceito para um plano bidimensional temos de forma análoga, além de x, os pontos em y.

As imagens geradas na janela gráfica podem ser manipuladas usando o conceito acima mas, antes, faremos alguns exercícios para entender a aplicação desse conceito no Scilab. Por exemplo, tente este exercício:

```
-->x = ( 0 : 0.1 : 10 )’; y = sin( x ); z = cos( x );
-->plot( x , y );
```

Isso fará com que o gráfico de y seja plotado normalmente.
Então nós usamos o comando drawlater() que indicará ao Scilab para não processar os próximos comandos aos gráficos e armazená-los numa memória temporária:

```
-->drawlater();
-->clf();
-->plot( x , z );
```

Note que o comando *clf()* não limpou a janela gráfica e o gráfico de z não foi plotado.

Agora, com o comando *drawnow()* nós indicamos ao Scilab que processe esses comandos:

```
-->drawnow();
```

E assim a janela gráfica com o plot anterior foi limpa e o gráfico de z foi exibido, de forma que esses dois processos aconteceram quase que instantaneamente sob a ótica do usuário. Agora aplicaremos o que vimos até aqui para simular uma animação.

- **Animando um gráfico**

O script do seguinte exercício é um script que produz a animação de uma onda senoidal:

```
1.  xdel( winsid() ); x = ( 0 : 0.1 : 10 )';
2.  for i = 1 : 600
3.      drawlater();
4.      clf();
5.      y = sin( x + i/10);
6.      plot( x , y );
7.      drawnow();
8.  end;
```

Observe que y é recalculado para plotar uma sequência de 600 gráficos que formarão a animação através do laço de repetição *for*. Note também que usamos o comando *xdel(winsid())* neste exemplo por questões didáticas porque isso faz com que todas as janelas gráficas abertas sejam fechadas antes de executar o restante do script, a fim de evitar possíveis erros.

Este segundo exercício é um script que mostra animação do movimento de uma onda senoidal sendo achatada com o *plot2d()*:

```
1.  xdel( winsid() );
2.  x = ( 0 : 0.1 : 10 )';
3.  for i = 0 : 600
4.      drawlater();
5.      clf();
6.      y = exp( - 0.01 .* i ) .* sin( x + i/10 );
7.      plot2d( x , y , rect = [ 1 -1 10 1 ] );
8.      drawnow();
9.  end;
```

6.1.10. Plotando funções na forma y=f(x)

O comando *fplot2d()* pode ser usado para plotar uma função de maneira similar à produzida pelo comando *plot()*, exceto que na função *fplot2d()* a função é definida como uma função externa do tipo *y = f(x)*. Por exemplo:

```
-->x = ( 0 : 0.025 : 10 );
-->deff( "[ y ] = f( x )" , "y = sin( x ) + sin( 2 .* x )" );
-->fplot2d( x , f );
```

Para adicionar uma grade e rótulos aos eixos:

```
-->xgrid();
-->xtitle( "Sinal da onda" , "tempo(s)" , "sinal(J)" );
```

6.2. Gráficos tridimensionais

No Scilab, podemos *plotar* gráficos tridimensionais. Neste tópico, iremos ver algumas dessas funções.

6.2.1. Função plot3d()

A função *plot3d(<x>,<y>,<z>)* exibe um gráfico tridimensional, onde <*x*>, <*y*> são vetores linhas com n1 e n2 elementos, respectivamente, e <*z*> uma matriz n1xn2, onde *z(i,j)* é o valor da altura da superfície no ponto *(x(i), y(j))*.

Exemplo:

Vamos digitar no SciNotes o seguinte código,

```
x=[0:0.1:2*%pi]';
disp(x);
z = cos(x) * sin(x');
plot3d(x,x,z);
```

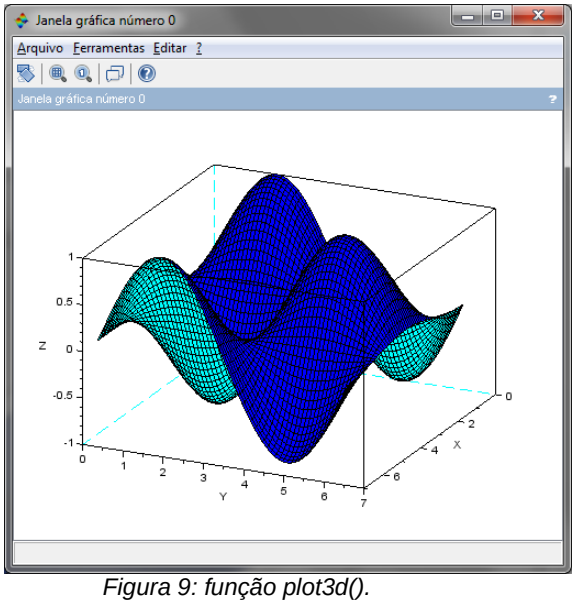

Figura 9: função plot3d().

6.3. Exercícios

1) *Plote* a reta $2x + 1$ no intervalo [0,10], usando a função *plot()*.

2) *Plote* a função f(x,y) = $(sen(x) * e^x) * \cos(y)$, onde x=([0:0.1:2*%pi])' e y=x'.

7. Referências bibliográficas

1) Motta, P.S.P., Introduçãoo ao Scilab, versão 3.0, 2004.
2) Guidorizzi, H.L., Um curso de Cálculo, Volume 1, 5.ª edição, 2001.
3) Bucchi, P., Matemática, Volume Único, 1.ª edição, 1992.
4) Giovanni, J.R. ; Bonjorno, J.R. ; Giovanni, J.R., Matemática Fundamental, 1.ª Edição, 1994.
5) Danusio Gadelha Filho. Scilab 5.x. Disponível em: < http://euler.mat.ufrgs.br/~giacomo/Manuais-softw/SCILAB/Apostila%20de%20Scilab%20-%20atualizada.pdf>. Data de acesso: 26/10/2015.
6) Anderson Almeira Ferreira. Disponível em: < http://www.decom.ufop.br/anderson/BCC701/scilab_arquivo.pdf>. Data de acesso: 26/10/2015.
7) Daniel Norberto Kozakevich. Disponível em: < http://mtm.ufsc.br/~daniel/amcom/scilab/IntroaoScilab.html>. Data de acesso: 26/10/2015.