

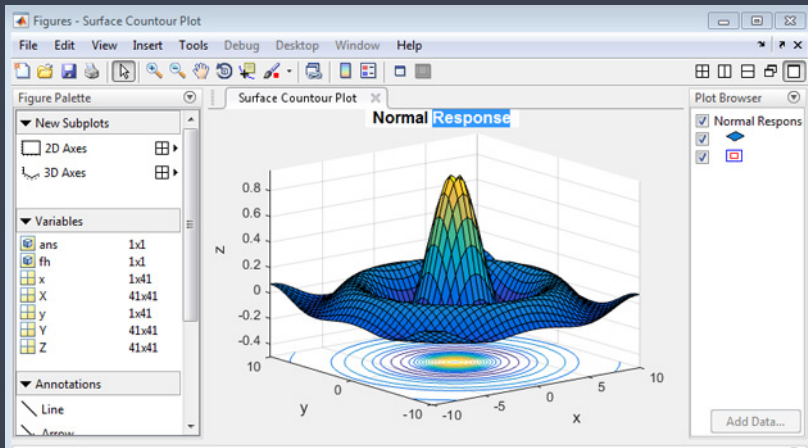
Introdução à Programação Científica

Conhecendo o Scilab e compreendendo os conceitos fundamentais em lógica de programação

by João Paulo Carvalho (Universidade Federal do Piauí)
on 26 de setembro de 2020

» Motivação inicial

Ser capazes de implementar métodos matemáticos de otimização no **Matlab**.



» Obstáculo iminente

Ao tentar adquirir uma licença básica para estudantes, nos deparamos com o seguinte valor.

New License for MATLAB Student R2020b	
Product	Price
New Products	
MATLAB and Simulink Student Suite	USD 55.00
Subtotal:	USD 55.00

Tomando por base a cotação do dólar de **16/09/2020** logo abaixo, temos um preço de aproximadamente **R\$ 288,75**, sem incluir taxas de câmbio e compra.

USD 1,00 = BRL 5,25

» Alternativas

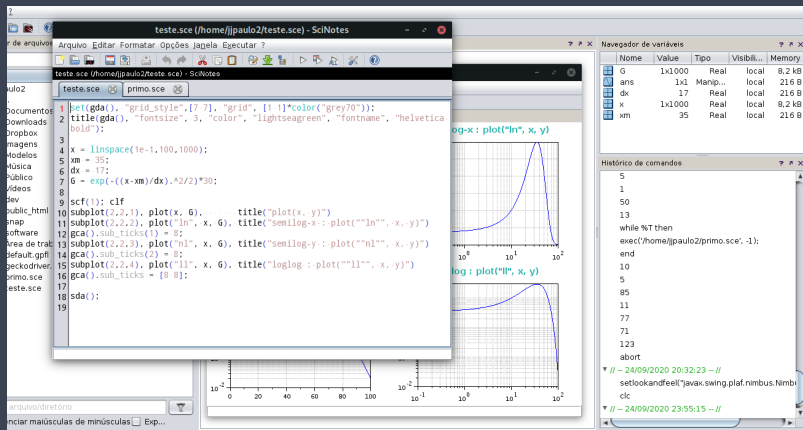




Recursos do Scilab

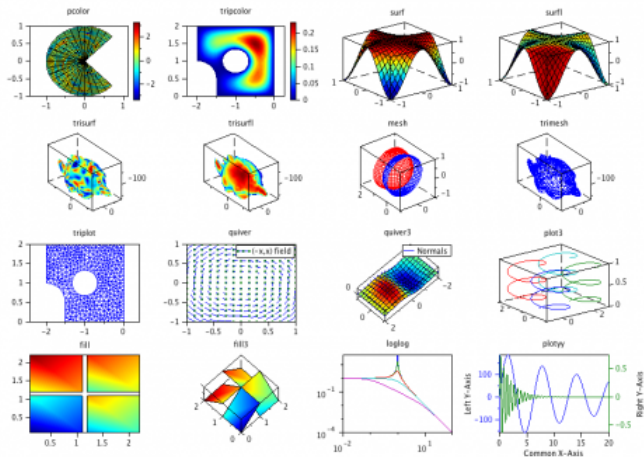
» Programação

Interface intuitiva que possibilita a implementação dos seus próprios algoritmos.



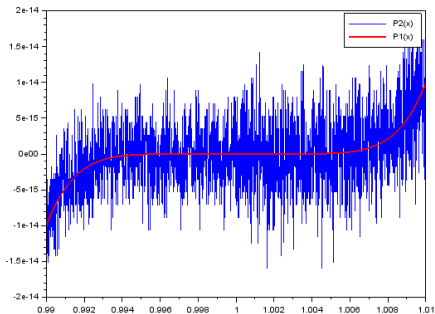
» Plotagem de gráficos

Comandos que facilitam a plotagem de gráficos 2D e 3D, dos mais diversos tipos.



» Análise numérica

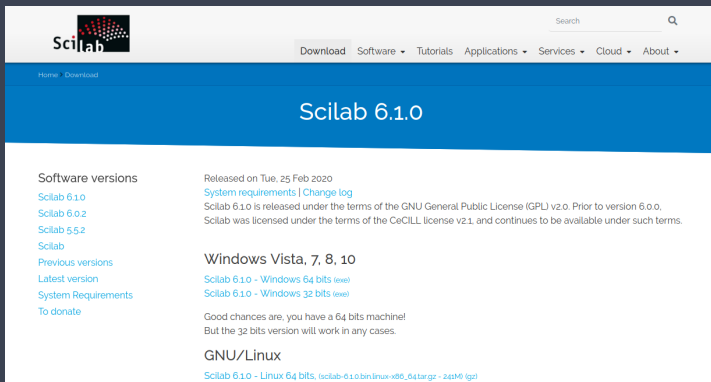
Ferramentas que facilitam a implementação de análises numéricas com constantes, funções matemáticas e métodos pré-implementados que te deixam preocupado apenas com a parte essencial do seu trabalho.



Instalação e uso do Scilab

» Download e instalação

<https://www.scilab.org/download>



The screenshot shows the Scilab website's download page for version 6.1.0. The page has a blue header with the Scilab logo and navigation links. The main content area is white and features a blue banner with the text 'Scilab 6.1.0'. Below the banner, there are two columns of text. The left column lists software versions and links to previous versions, the latest version, system requirements, and a donation link. The right column contains release information, including the date (Tue, 25 Feb 2020), system requirements, and a link to the change log. It also provides download links for Windows (64 bits and 32 bits) and GNU/Linux (64 bits), along with a note about the 32 bits version.

Scilab 6.1.0

Software versions

- [Scilab 6.1.0](#)
- [Scilab 6.0.2](#)
- [Scilab 5.5.2](#)
- [Scilab](#)
- [Previous versions](#)
- [Latest version](#)
- [System Requirements](#)
- [To donate](#)

Released on Tue, 25 Feb 2020
[System requirements](#) | [Change log](#)

Scilab 6.1.0 is released under the terms of the GNU General Public License (GPL) v2.0. Prior to version 6.0.0, Scilab was licensed under the terms of the CeCILL license v2.1, and continues to be available under such terms.

Windows Vista, 7, 8, 10

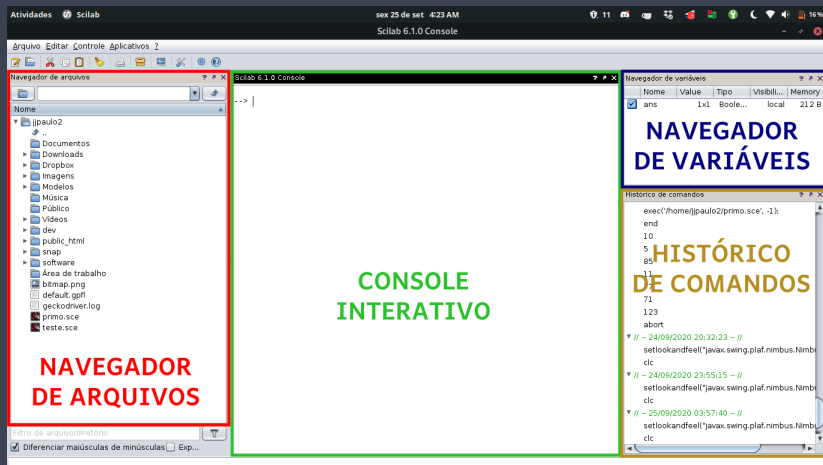
[Scilab 6.1.0 - Windows 64 bits \(exe\)](#)
[Scilab 6.1.0 - Windows 32 bits \(exe\)](#)

Good chances are, you have a 64 bits machine!
But the 32 bits version will work in any cases.

GNU/Linux

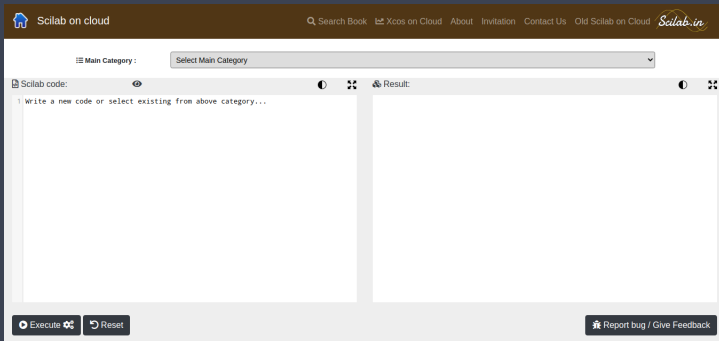
[Scilab 6.1.0 - Linux 64 bits, \(scilab-6.1.0.binlinux-x86_64.tar.gz - 241M\) \(gz\)](#)

» Tour pela interface



» Use online

<https://cloud.scilab.in/>



Lógica de programação

» Breve explicação sobre a arquitetura dos computadores



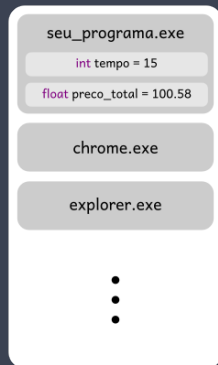
» Variáveis

Variáveis

guardam valores que são essenciais para a execução e lógica do programa.

Elas ficam disponíveis e podem ser chamadas e alteradas à qualquer momento na execução do programa.

Elas ficam armazenadas na memória RAM do computador.



» Variáveis na prática

Abaixo veja o código contendo alguns tipos primitivos de dados encontrados em qualquer linguagem de programação.

```
1 // variáveis do tipo `string`  
2 hello = "Olá, mundo!";  
3 palavra = 'teste';  
4  
5 // variáveis do tipo `int`  
6 numero = 5;  
7 soma = 5 + 10;  
8  
9 // variável do tipo `float`  
10 pi = 3.14;  
11  
12 // variáveis do tipo `array`  
13 lista = [1, 2, 3];  
14 lista3 = [2.5, 88, 9.7];  
15
```

» Constantes

Constantes também guardam valores essenciais para a lógica do programa.

Entretanto, **diferente das variáveis, elas ficam disponíveis para serem chamadas, mas não podem ser alteradas.**

» Constantes na prática

Constantes no **Scilab** costumam ter seu nome antecedido pelo caractere % (porcento). Existe uma tabela com algumas constantes matemáticas pré-definidas para auxiliar com alguns cálculos.

Nome da constante	Valor
%pi	3.1415927
%e	2.7182818
%eps	2.2×10^{-16}
%i	$0.0 + i$
%inf	$1/0$
%nan	$\infty - \infty$

» Entrada e saída

Insira um número para
verificar se é primo.



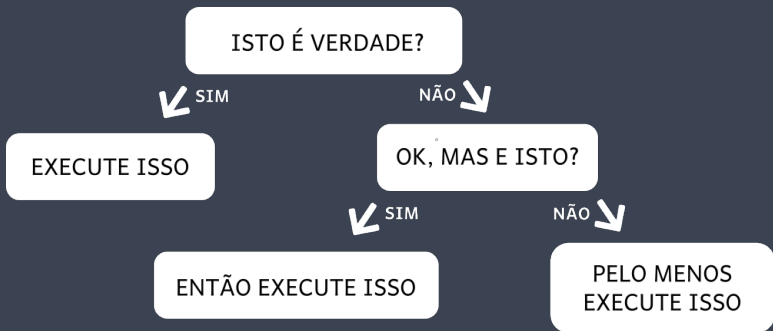
Em geral, um programa deve ser capaz de receber dados de entrada, executar um algoritmo utilizando os mesmos e então retornar um dado de saída após isso tudo.

» Entrada e saída na prática

O código abaixo ilustra os comandos padrão para entrada e saída de dados no **Scilab**.

```
1 // recebe uma entrada
2 numero = input("Insira um número...");
3
4 // algoritmo que trata a entrada
5 ...
6
7 // exibe a saída
8 disp('O número inserido é válido!');
9
```

» Estruturas condicionais



Em quase 100% do tempo programando é necessário tomar algum tipo de decisão condicional. Esta é uma das estruturas mais fundamentais de todas.

» Condições e valores booleanos

É costume encontrar o tipo de dado bool ou boolean em linguagens de programação. Eles representam **valores booleanos**.

Definição

***Valor booleano:** Representa um valor que só pode ser verdadeiro ou falso.*

Em geral, operações lógicas retornam valores booleanos. O código abaixo ilustra bem isto.

```
1 // definindo variáveis booleanas
2 condicao_falsa = %F;
3 condicao_verdadeira = %T;
4
5 // condicao vai ser verdadeiro e vai receber o valor T
6 condicao = ((2 + 2) == 4);
7
```

» Estruturas condicionais na prática

O seguinte código mostra os comandos padrão para tomada de decisões no **Scilab**.

```
1 // se a condição for verdadeira
2 if condicao1 then
3     // código
4     ...
5
6 elseif condicao2 then
7     // código
8     ...
9
10 else
11     // código
12     ...
13
14 end
15
```


» Operadores lógicos

São operadores que trabalham na manipulação de valores booleanos. São os mesmos da matemática.

Nome do operador	Simbolo Scilab	Exemplo
OR (OU)		$\%T \mid \%F = T$
AND (E)	&	$\%T \& \%F = F$
NOT (NÃO)	~	$\sim \%T = F$

» Operadores de comparação

São operadores que servem para comparar dois valores.
Também são os mesmos da matemática.

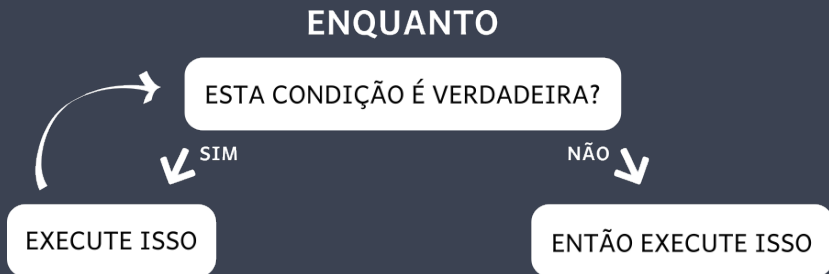
Nome do operador	Simbolo Scilab	Exemplo
MENOR QUE	<	$3 < 3 = F$
MAIOR QUE	>	$1 > 3 = F$
MENOR OU IGUAL QUE	<=	$3 <= 3 = T$
MAIOR OU IGUAL QUE	>=	$1 <= 3 = T$
IGUAL A	==	$3 == 3 = T$
DIFERENTE DE	~=	$1 \sim = 1 = F$

» Operadores aritméticos

São os operadores usados para representar operações matemáticas básicas. São os mesmos da matemática (com exceção do operador de divisão à esquerda).

Nome da operação	Simbolo Scilab	Exemplo
ADIÇÃO	+	$1 + 3 = 4$
SUBTRAÇÃO	-	$1 - 3 = -2$
MULTIPLICAÇÃO	*	$3 * 3 = 9$
DIVISÃO À ESQUERDA	\	$8 \setminus 4 = 0.5$
DIVISÃO À DIREITA	/	$8 / 4 = 2$
EXPONENCIAÇÃO	^	$1^3 = 1$

» Estruturas de repetição



Outras estruturas de controle fundamentais são as estruturas ou laços de repetição. Normalmente se tem dois tipos de estruturas: **while** e **for**. Acima está ilustrado o esquema de um laço do tipo **while**.

» Estruturas de repetição

CONSIDERE i VARIANDO NO
SEGUINTE INTERVALO NATURAL

$$0 < i < n$$



EXECUTE ISSO À CADA ITERAÇÃO

Acima está ilustrado o esquema de um laço do tipo **for**.

» Estruturas de repetição na prática

O código abaixo exemplifica a implementação do loop while no **Scilab**.

```
1 while condicao then
2     // código que será executado enquanto a condição for
    verdadeira
3     ...
4
5 else
6     // código que será executado assim que a condição se
    tornar falsa
7     ...
8
9 end
10
```

» Estruturas de repetição na prática

O código abaixo exemplifica a implementação do loop for no **Scilab**.

```
1  for i=inicio:salto:fim
2      // código
3      ...
4
5  end
6
7  for j=inicio:fim
8      // por padrão salto=1 caso seu valor seja omitido
9      ...
10
11 end
12
```

» Forçando a parada de loops

Você pode invocar o comando **break** a qualquer momento para forçar o encerramento de um laço de repetição. Veja o exemplo abaixo.

```
1  for i=1:100
2      // código
3      ...
4
5      if i == 25 then
6          break;
7      end
8
9  end
10
```


» Funções

Acabamos repetindo um mesmo procedimento várias vezes dentro de um programa. A fim de tornar o código mais enxuto, legível e fácil de corrigir erros e bugs, devemos criar funções para trechos que irão se repetir constantemente.

Em geral, funções devem receber valores como parâmetro, executar seu procedimento e por fim retornar ou não algum valor.

```
1 retorno = funcao(parametro1, parametro2);  
2
```

» Funções na prática

Veja um exemplo de código com muita repetição e sem a utilização de funções.

```
1 nome1 = input('Informe o 1º nome: ');
2 disp("Olá " + nome1);
3
4 nome2 = input('Informe o 2º nome: ');
5 disp("Olá " + nome2);
6
7 nome3 = input('Informe o 3º nome: ');
8 disp("Olá " + nome3);
9
10 nome4 = input('Informe o 4º nome: ');
11 disp("Olá " + nome4);
12
13 nome5 = input('Informe o 5º nome: ');
14 disp("Olá " + nome5);
15
```

» Funções na prática

Veja o mesmo código, entretanto, com o problema de repetições resolvido.

```
1 function string_formatada=ola(nome)
2     string_formatada = "Olá " + nome;
3 endfunction
4
5 for i=1:5
6     nome = input('Informe um nome: ', 's');
7     ola_nome = ola(nome);
8     disp(ola_nome);
9 end
10
```

Até o próximo encontro!