Questão: Como armazenar números em caixas?

Representação finita => erros de arredondamento

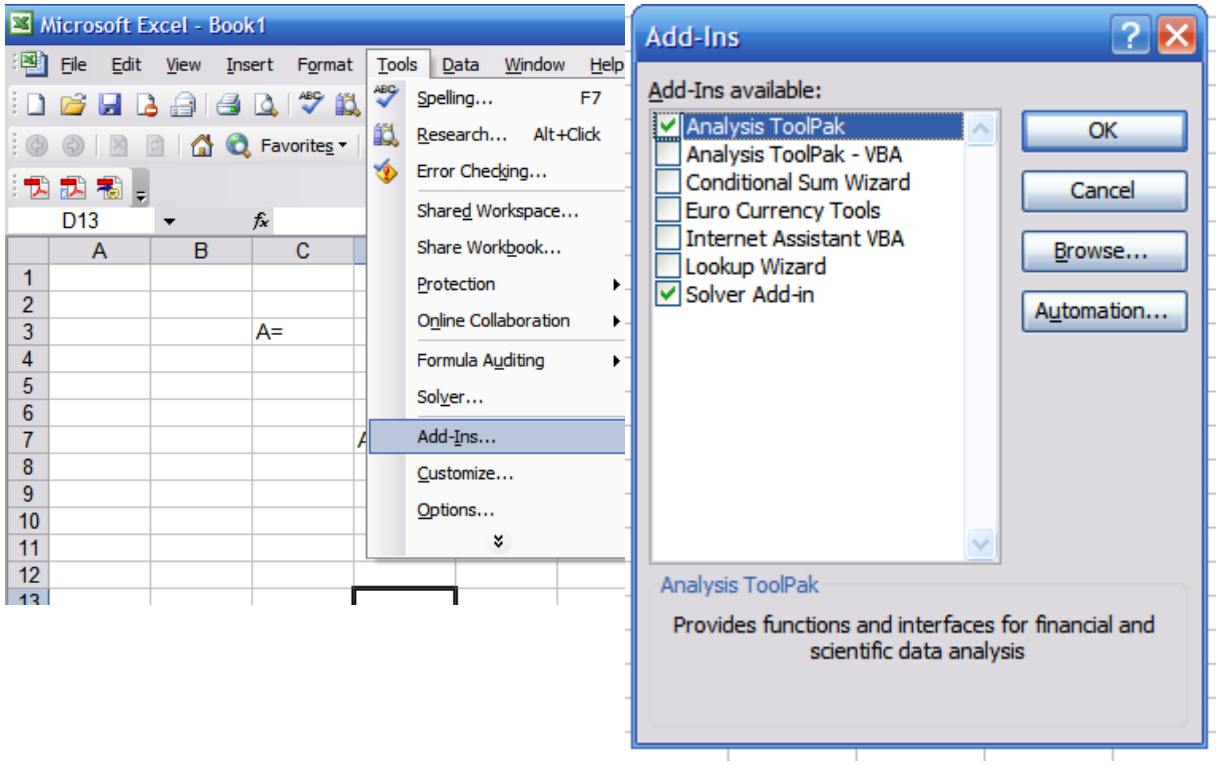
$423_{10} = 4*100 + 2*10 + 3*1 = 4*10^2 + 2*10^1 + 3*10^0$

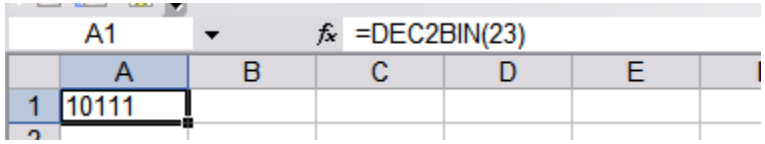
$35.23_{10} = 3*10 + 5*1 + 2*0.1 + 3*0.01 = 3*10^1 + 5*10^0 + 2*10^{-1} + 3*10^{-2}$

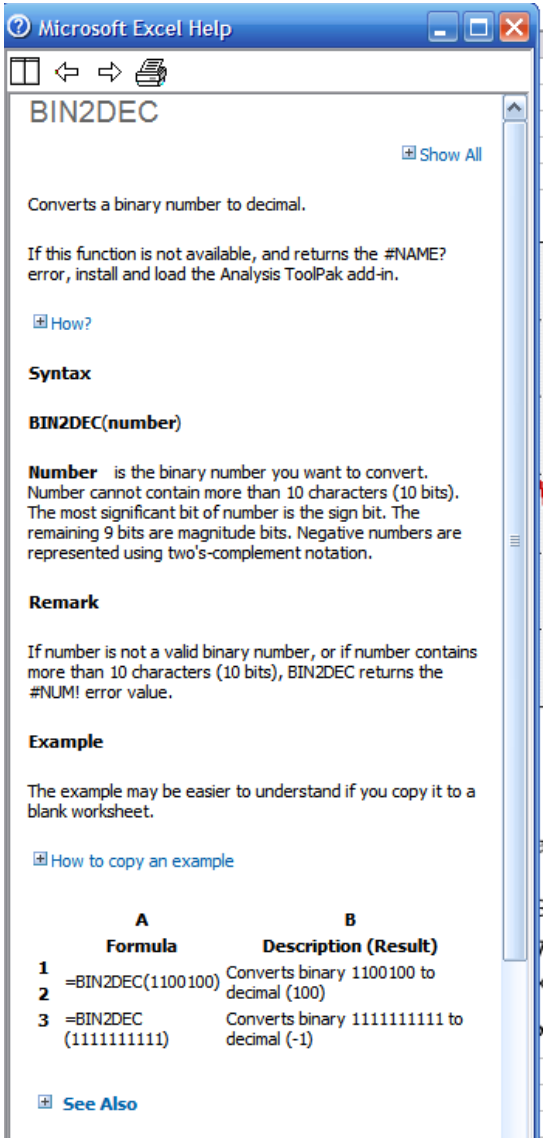
```
>> dec2bin(87)

ans=  1010111
```

87=1x 2^6+1x 2^4+ 1 x2^2+1x 2^1+1x 2^0

No Excel há que introduzir o add-in para incluir a instrução de conversão







Sistemas de codifição

Decimal	Hexadecimal	Binário
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Problema de Arredondamentos

Exemplo simples: aplicar uma sucessão de operações e depois a sucessão inversa de operações

Aplicar a raiz quadrada a um número, N vezes. Depois aplicar N vezes o quadrado. Conseguirei recuperar o número inicial?

Fazer em excel...

Exemplo 2: funções

Fazer no EXCEL:

sin(PI())

Quanto dará?

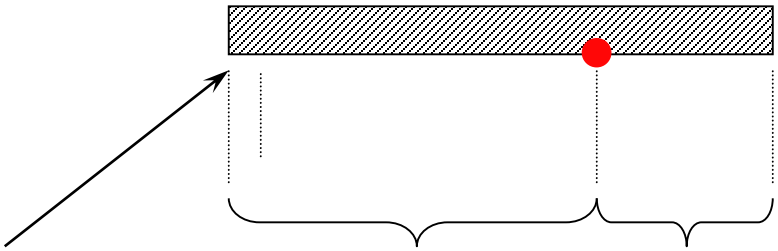
Fazer no EXCEL:

cos(PI())

Quanto dará?

E NO MATLAB DARÁ O MESMO?

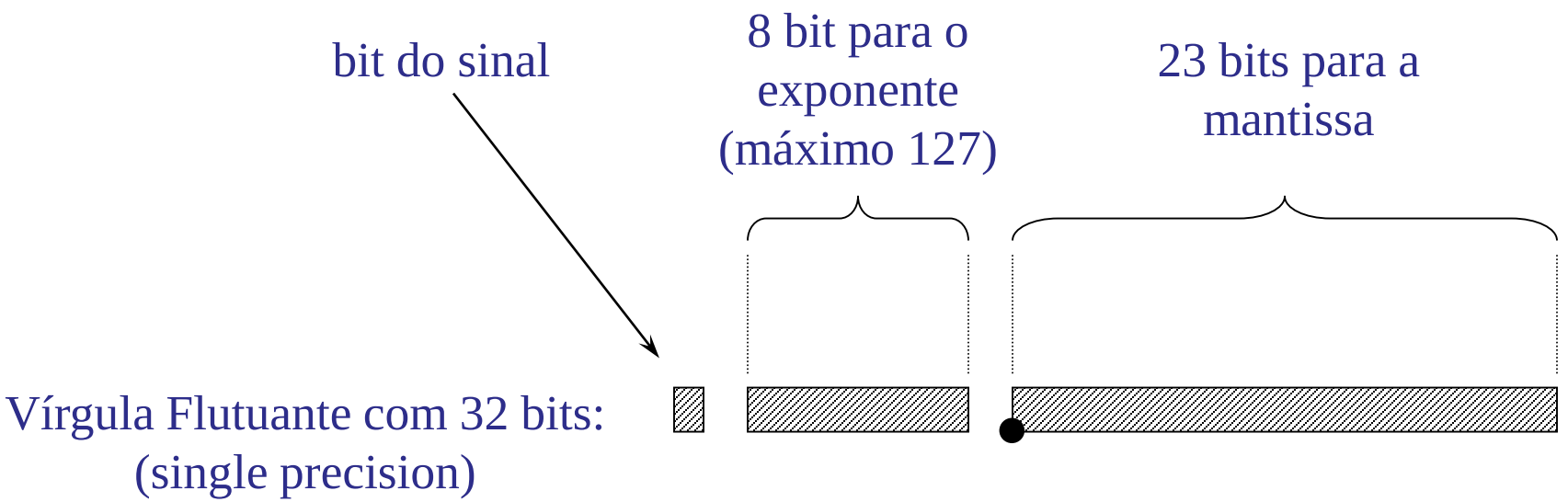
Representação em Vírgula Fixa



bit do sinal Parte Inteira Parte Fracionária

Exemplo: 51.23, 0.23, 0.99 estão todos representados com 2 dígitos na parte fracionária.

Representação em Vírgula Flutuante



$$x = (-1)^s \times b^e \times (0.a_1 a_2 \ldots a_t)$$

Representação em vírgula flutuante normalizada (**binário**):

$$x = (1 + f) \times 2^{e} \qquad 0 \leq f < 1$$

Os números em dupla precisão dedicam 52 bits para a fracção e 11 bits para o expoente.

Variação mínima na precisão: $2^{-52} = eps$

Name	Binary	Decimal
eps	2^(-52)	2.2204e-16
realmin	2^(-1022)	2.2251e-308
realmax	(2-eps)*2^1023	1.7977e+308

Ver help do matlab com
floating point precision

Cleve's Corner

Floating points

IEEE Standard unifies arithmetic model

by Cleve Moler

If you look carefully at the definition of fundamental arithmetic operations like addition and multiplication, you soon encounter the mathematical abstraction known as the *real numbers*. But actual computation with real numbers is not very practical because it involves limits and infinities. Instead, MATLAB and most other technical computing environments use *floating-point* arithmetic, which involves a finite set of numbers with finite precision. This leads to phenomena like *roundoff error*, *underflow*, and *overflow*. Most of the time, MATLAB can be effectively used without worrying

where f is the fraction or mantissa and e is the exponent. The fraction must satisfy

$$0 \le f < 1$$

and must be representable in binary using at most 52 bits. In other words, $2^{52}f$ must be an integer in the interval

$$0 \le 2^{52}f < 2^{53}$$

The exponent must be an integer in the interval

$$-1022 \le e \le 1023$$

What is the output?

```
a = 4/3
b = a - 1
c = b + b + b
e = 1 - c
```

Conversão pode levar a séries infinitas

A very important example occurs with the simple MATLAB statement

```
t = 0.1
```

The value stored in `t` is not exactly 0.1 because expressing the decimal fraction $1/10$ in binary requires an infinite series. In fact,

$$1/10 = 1/2^4 + 1/2^5 + 0/2^6 + 0/2^7 + 1/2^8 + 1/2^9 + 0/2^{10} + 0/2^{11} + 1/2^{12} + \ldots$$

After the first term, the sequence of coefficients 1, 0, 0, 1 is repeated infinitely often. The floating-point number nearest 0.1 is obtained by rounding this series to 53 terms, including rounding the last four coefficients to binary 1010. Grouping the resulting terms together four at a time expresses the approximation as a base 16, or *hexadecimal*, series. So the resulting value of `t` is actually

$$t = (1 + 9/16 + 9/16^2 + 9/16^3 + \ldots + 9/16^{12} + 10/16^{13}) \cdot 2^{-4}$$

Cleve Moler

Ou seja:

Há números racionais que só admitem uma representação aproximada no computador.

Mas de notar que o mesmo já acontecia na representação decimal de 1/3 ...

De forma mais geral ainda: qualquer número irracional tem de ser representado de forma aproximada num computador...

Acidentes causados por tratamento incorrecto dos erros de arredondamento

Ver a página na internet http://ta.twi.tudelft.nl/nw/users/vuik/wi211/disasters.html#patriot

Guerra do Golfo (1991):

24 bits => 2^24=16777216 números

Os relógios internos do bateria Patriot tinham registos de 24 bits. Além disso utilizavam a unidade da décima do segundo. A conversão para binário produz erros de truncatura da ordem de 0.0000000000000000000000011001100... ou seja 0.000000095 decimal. Estes erros podiam-se acumular ao longo do tempo, de tal maneira que se a bateria estivesse em funcionamento há 100 horas, os erros acumulados poderiam ser da ordem de 0.000000095×100×60×60×10=0.34. Como um missil Scud viaja 1,676 metros por segundo, a imprecisão gerada pode ser superior ao raio de correcção/detecção do antimissil.

The range gate's prediction of where the Scud will next appear is a function of the Scud's known velocity and the time of the last radar detection. Velocity is a real number that can be expressed as a whole number and a decimal (e.g., 3750.2563...miles per hour). Time is kept continuously by the system's internal clock in tenths of seconds but is expressed as an integer or whole number (e.g., 32, 33, 34...). The longer the system has been running, the larger the number representing time. To predict where the Scud will next appear, both time and velocity must be expressed as real numbers. Because of the way the Patriot computer performs its calculations and the fact that ***its registers are only 24 bits long****, the conversion of time from an integer to a real number cannot be any more precise than 24 bits. This conversion results in a loss of precision causing a less accurate time calculation. The effect of this inaccuracy on the range gate's calculation is directly proportional to the target's velocity and the length of the system has been running. Consequently, performing the conversion after the Patriot has been running continuously for extended periods causes the range gate to shift away from the center of the target, making it less likely that the target, in this case a Scud, will be successfully intercepted.*

GAO/IMTEC-92-26

The Vancouver Stock Exchange

In 1982 the Vancouver Stock Exchange instituted a new index initialized to a value of 1000.000. The index was updated after each transaction. Twenty two months later it had fallen to 520. The cause was that the updated value was truncated rather than rounded. The rounded calculation gave a value of 1098.892.

Sources

The Wall Street Journal November 8, 1983, p.37.
The Toronto Star, November 19, 1983.
B.D. McCullough and H.D. Vinod, Journal of Economic Literature, Vol XXXVII (June 1999), pp. 633-665.

Como se faz uma soma?

```
>> x=     1.1*2^30   +    9*2^(-30)
x = 1.1811e+009
```

Os dois números são convertidos em números com igual expoente, e depois as mantissas são somadas.

```
>> y=1.1*2^30
y = 1.1811e+009
```

Por isso podem surgir arredondamentos importantes. Porém hoje a precisão é grande, chegando-se a utilizar 52 bits na parte da mantissa. Por isso neste exemplo não se nota o efeito do arrendondamento.

```
>> x==y
ans = 1
```

Mas o mesmo não sucede em pequenos processadores (microcontroladores) utilizados em múltiplos equipamentos de electrónica.

Mais uma consequência de haver arredondamentos:

instruções `if x==y` são problemáticas sempre que x e y não forem inteiros.

Cancelamento Catastrófico

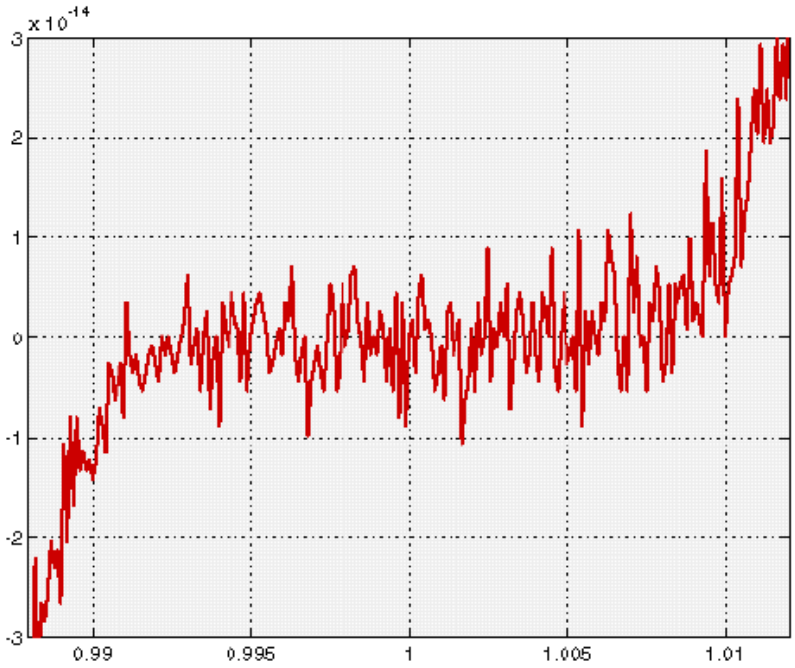
Se x e y forem números grandes, e os substrairmos podemos gerar erros de arredondamentos grandes em termos de erro relativo no resultado final:

$$f(x, y) = x - y$$

pode ser grande se os números forem grandes

$$\delta_{x-y} = \frac{\varepsilon_{x-y}}{Diferença} = \frac{\varepsilon_{x-y}}{x - y}$$

é pequeno quando há cancelamento



But the resulting plot doesn't look anything like a polynomial. It isn't smooth. You are seeing roundoff error in action. The y-axis scale factor is tiny, 10^{-14}. The tiny values of y are being computed by taking sums and differences of numbers as large as $35 \cdot 1.012^4$. There is severe subtractive cancellation. The example was contrived by using the Symbolic Toolbox to expand $(x - 1)^7$ and carefully choosing the range for the x-axis to be near $x = 1$. If the values of y are computed instead by

```
y = (x-1).^7;
```

then a smooth (but very flat) plot results. ■

Our final example plots a seventh degree polynomial.

```
x = 0.988:.0001:1.012;
y = x.^7-7*x.^6+21*x.^5-35*x.^4+35*x.^3-...
    21*x.^2+7*x-1;
plot(x,y)
```

Cleve Moler is chairm
and co-founder of
The MathWorks.
His e-mail address is
moler@mathworks.com

Como analisar facilmente o computador a cometer erros de arredondamento?

```
>> 1-0.9999
ans = 1.0000e-004

>> format long
>> 1-0.9999
ans = 9.99999999998899e-005

>> 1-0.99999
ans = 9.99999999954490e-006

>> 1-0.999999
ans = 1.00000000028756e-006
```

não altera a precisão com que são feitos os cálculos (só a apresentação)

Default: short

Como estudar de uma forma analítica o condicionamento de um algoritmo?

Objectivo: Calcular xy/z

Algoritmo 1

1) x y
2) res/z

Algoritmo 2

1) x / z
2) res * y

Algoritmo 1

1) x y $\Rightarrow xy + \varepsilon_{xy}$

2) res/z $\Rightarrow \left(xy + \varepsilon_{xy}\right)/z + \varepsilon_{xy/z}$

$$\frac{xy}{z} \Rightarrow \frac{xy}{z} + \frac{\varepsilon_{xy}}{z} + \varepsilon_{xy/z}$$

Algoritmo 2

1) x / z $\Rightarrow x/z + \varepsilon_{x/z}$

2) res * y $\Rightarrow \left(x/z + \varepsilon_{x/z}\right) \times y + \varepsilon_{xy/z}$

$$\frac{xy}{z} \Rightarrow \frac{xy}{z} + y\varepsilon_{x/z} + \varepsilon_{xy/z}$$

Se x e y forem muito diferentes em magnitude, a magnitude do erro intermédio não diverge usando o primeiro algoritmo. Se x/z tiverem magnitudes semelhantes, pode-se utilizar o segundo algoritmo pois o erro intermédio também não diverge.

Dicas para evitar grandes erros

1) Devemos evitar que os cálculos intermédios tenham grandes números (por exemplo, usem expressões com singularidades) pois os erros nos cálculos intermédios podem tornar-se grandes, ainda que o resultado final possa não o vir a ser.

Por exemplo: para calcular x y / z devemos:

a) calcular primeiro xy e depois /z se x e y forem muito diferentes em magnitude.

b) devemos efectuar primeiro x/z ou y/z se estes números forem de magnitudes semelhantes.

Assim, numéricamente certas equações não são equivalentes porque conduzem a erros muito diferentes

Exemplo:

$$f_1(x) = \sqrt{x}(\sqrt{x+1} - \sqrt{x}), \qquad f_2(x) = \frac{\sqrt{x}}{\sqrt{x+1} + \sqrt{x}}$$

Na primeira expressão, podemos ter erros importantes se $x=10^{15}$ pois o erro será dessa ordem.

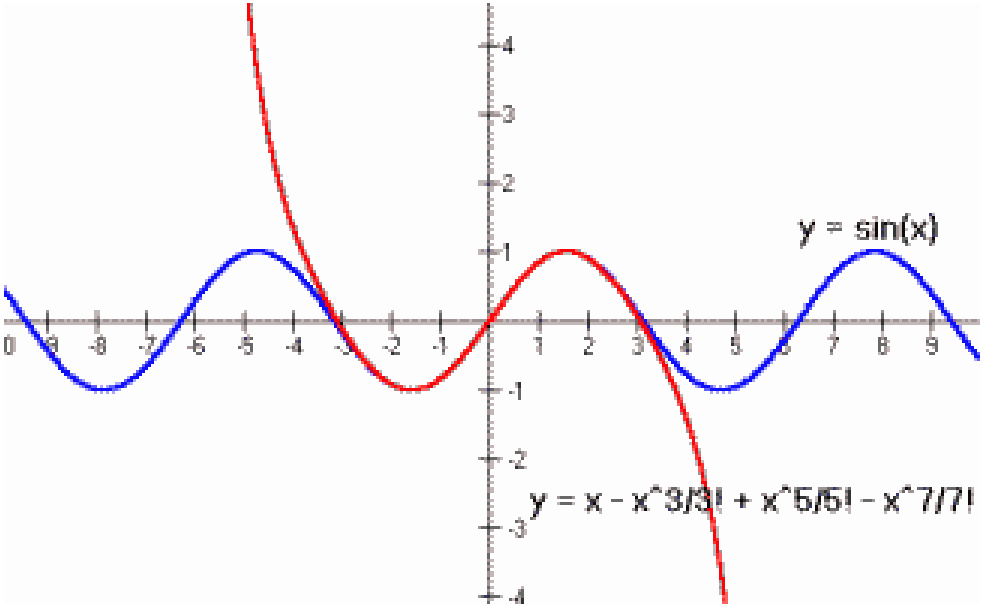
$$f_1(x) = \frac{1 - \cos x}{x^2}, \qquad f_2(x) = \frac{\sin^2 x}{x^2(1 + \cos x)}$$

f_1 deve ser utilizada para calcular a expressão em x~ π;
neste caso a expressão f_2 tem uma singularidade e diverge.

No entanto, f_2 deve ser utilizada para calcular a expressão em x~ 0;
nesse caso a expressão f_1 tem uma singularidade e diverge.

TPC: Usar o MATLAB para representar a função entre - 2π e 2π.

Como é que o computador calcula o seno de um número

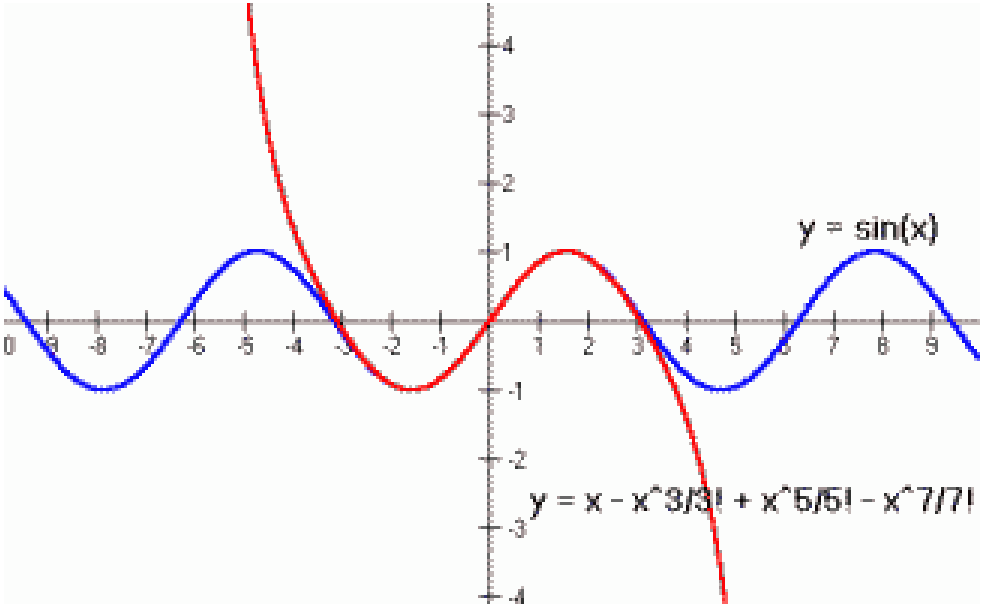


Questão: Quanto vale sin(2π)?

E sin(20π)? E sin(2000π)?

TPC: Usar o Excel para fazer o gráfico de sin(2n π) entre n=0 e 300.

Como é que o computador calcula o seno de um número



O uso de séries de Taylor não é apropriado, pelo menos em todo o domínio. Os algoritmos usam diversas técnicas (mapeamentos, aproximações com polinómios de Chebyshev).

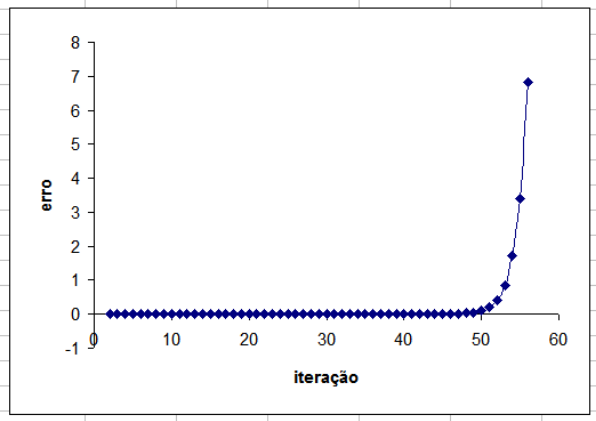
Exemplo: cálculos exactos versus numéricos

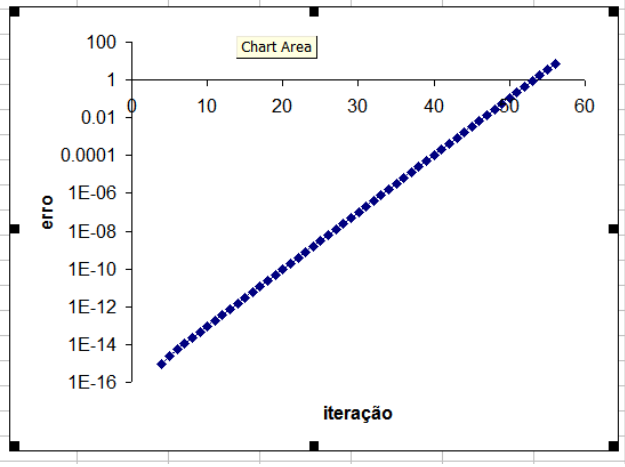
Sequência: $f_{n+1} = 2.9\, f_n - 1.8\, f_{n-1}$

Solução exacta: $f_n = (0.9)^{n-1}$

Ainda que seja desejável conhecer soluções exactas, isso nem sempre é possível, sendo essencial compreender em que medida a natureza dos resultados numéricos não são um artefacto do método numérico.

	A	B	C	D
1				
2		exacto	recurrencia	erro
3	1	1	1	
4	2	0.9	0.9	0
5	3	0.81	0.81	0
6	4	0.729	0.729	8.88E-16
7	5	0.6561	0.6561	2.44E-15
8	6	0.59049	0.59049	5.55E-15
9	7	0.531441	0.531441	1.17E-14
10	8	0.478297	0.4782969	2.38E-14
11	9	0.430467	0.43046721	4.8E-14
12	10	0.38742	0.387420489	9.65E-14
13	11	0.348678	0.34867844	1.94E-13
14	12	0.313811	0.313810596	3.88E-13
15	13	0.28243	0.282429536	7.76E-13
16	14	0.254187	0.254186583	1.55E-12
17	15	0.228768	0.228767925	3.11E-12
18	16	0.205891	0.205891132	6.21E-12
19	17	0.185302	0.185302019	1.24E-11
20	18	0.166772	0.166771817	2.48E-11
21	19	0.150095	0.150094635	4.97E-11
22	20	0.135085	0.135085172	9.94E-11
23	21	0.121577	0.121576654	1.99E-10
24	22	0.109419	0.109418989	3.98E-10
25	23	0.098477	0.098477089	7.95E-10
26	24	0.088629	0.08862938	1.59E-09
27	25	0.079766	0.07976644	3.18E-09
28	26	0.07179	0.071789792	6.36E-09
29	27	0.064611	0.064610806	1.27E-08
30	28	0.05815	0.058149712	2.54E-08
31	29	0.052335	0.052334712	5.09E-08
32	30	0.047101	0.047101185	1.02E-07
33	31	0.042391	0.042390955	2.04E-07
34	32	0.038152	0.038151635	4.07E-07
35	33	0.034337	0.034336024	8.14E-07
36	34	0.030903	0.030901526	1.63E-06





Eficiência Computacional
(em termos de número de operações)

$$p_4(x) = a_1x^4 + a_2x^3 + a_3x^2 + a_4x + a_5$$

$$p_{4n}(x) = (((a_1x + a_2)x + a_3)x + a_4)x + a_5$$

A multiplicações são operações que requerem mais operações elementares que as adições. Por isso, uma computação eficiente deve evitá-las...
No primeiro caso gastamos 4 multiplicações para calcular a_1x^4,
tantas quantas as multiplicações usadas no segundo método.