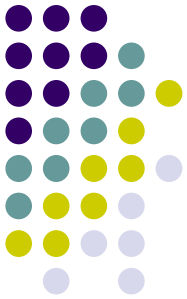
Vektorok

A fizika numerikus módszerei I.

mf1n1a06- mf1n2a06

Csabai István



Octave: alapok

- Az octave mint számológép:
  - `octave:##> 2+2`
    `ans = 4`
- Válasz elrejtése
  - `octave:##> 2+2;`
    `octave:##>`
  - `+ - / * ()`
  - Hatványozás: `^` (Magyar billentyűzeten: Alt Gr 3)
- Program futás megszalítása: Ctrl-C
- Kilépés: `quit`
- Help lapozás, kilépés: space, `q`
- Függvények:
  - `exp(1)`
    `ans = 2.7183`
  - `1.2 * sin(40*pi/180 + log(2.4^2))`
    `ans = 0.76618`

Függvények

`cos`	Cosine of an angle (in radians)
`sin`	Sine of an angle (in radians)
`tan`	Tangent of an angle (in radians)
`exp`	Exponential function (e^x)
`log`	Natural logarithm (NB this is $\log_e$, not $\log_{10}$)
`log10`	Logarithm to base 10
`sinh`	Hyperbolic sine
`cosh`	Hyperbolic cosine
`tanh`	Hyperbolic tangent
`acos`	Inverse cosine
`acosh`	Inverse hyperbolic cosine
`asin`	Inverse sine
`asinh`	Inverse hyperbolic sine
`atan`	Inverse tangent
`atan2`	Two-argument form of inverse tangent
`atanh`	Inverse hyperbolic tangent
`abs`	Absolute value
`sign`	Sign of the number (−1 or +1)
`round`	Round to the nearest integer
`floor`	Round down (towards minus infinity)
`ceil`	Round up (towards plus infinity)
`fix`	Round towards zero
`rem`	Remainder after integer division

Konstansok

- Foglalt változónevek: utasítások, függvények
- Konstansok:
  - Imaginárius egység: `i , j`
    ```
    octave:1> j*j
    ans = -1
    ```
  - Π : `pi`
  - e ,természetes logaritmus alapja: `e`

Skalár változók (számok)

- Változók definiálása:
```
octave:##> deg = pi/180
deg =0.017453
```
- Változók használata:
```
octave:##> 1.2 * sin(40*deg + log(2.4^2))
ans =0.76618
```
- Előző eredmény felhasználása:
```
octave:28> x = 2 * ans
x = 1.5324
```
- Nem kell deklarálni a változókat mint pl. C++ -ban, minden változó „double”
  - Pl. C nyelven így nézne ki:
  - double deg, pi;
    pi = 3.1414;
    deg = pi / 180.0;

A változók felülírhatóak

- Vigyázat! A konstansok felülírhatóak:
```
octave:13> e
e = 2.7183
octave:14> log(e)
ans = 1
octave:15> e=132.6
e = 132.60
octave:16> log(e)
ans = 4.8873
```
- Sőt a függvények is felülírhatóak!!
```
octave:17> sin(pi)
ans =  1.2246e-16
octave:18> sin=12.5
sin = 12.500
octave:19> sin(pi)
error: expecting integer index, found 3.141593
error: Array::Array (const Array&, const
dim_vector&): dimension mismatch
```

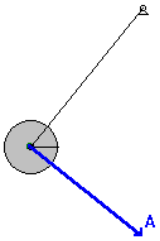
Változók törlése

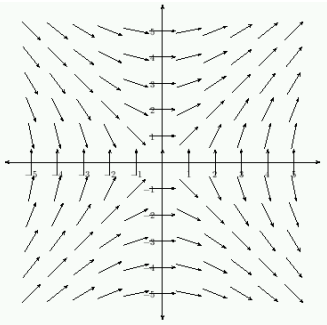
- ```
  octave:22> clear sin
  octave:23> sin(pi)
  ans =  1.2246e-16
  octave:24> log(e)
  ans = 4.8873
  octave:25> clear all
  octave:26> log(e)
  ans = 1
  ```
- Lehetőleg ne írjuk felül a `clear` utasítást!

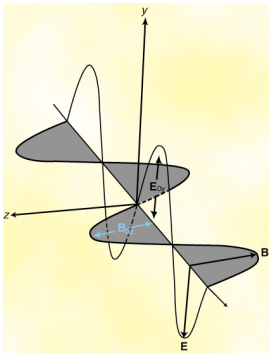
Vektorok és mátrixok az *octave*-ban

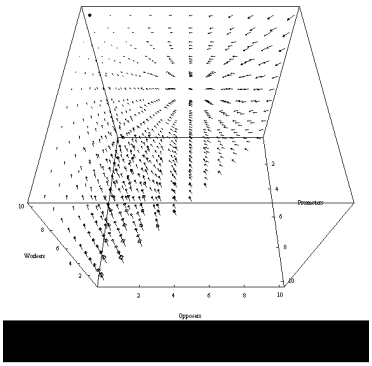
- Az *octave* a változókat mátrixnak tekinti (adattípusok)
- Skalár értékek: *1x1* mátrix
- Vektorok: *1xN* (vagy *Nx1*) mátrix
- Vektorok, mátrixok: más értelemben használjuk mint fizikában
  - 1 és több indexes tömbök
  - adattáblák
  ```

Fizikai vektorok és vektorterek



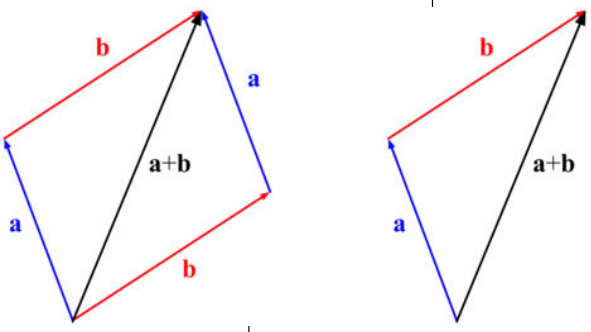




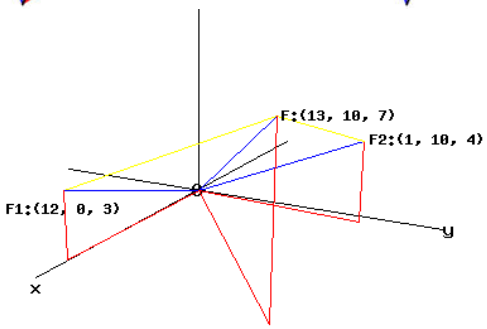


Vektorok

- A vektorokkal műveletek végezhetőek

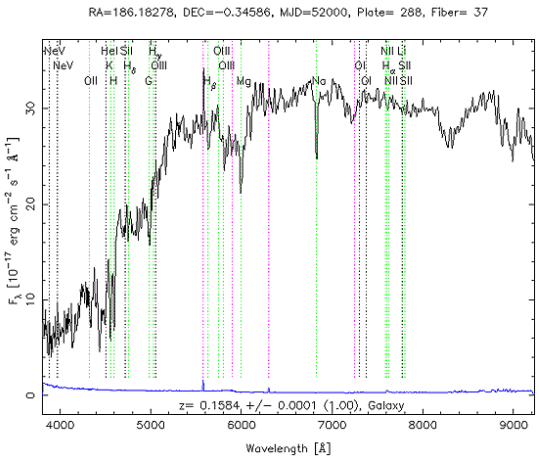


- A vektorok koordináta rendszerben ábrázolhatóak szám-hármasokként



Vektor

- Fizika: 2 dimenziós, 3 dimenziós vektor
- Programok: 1 soros/oszlopos mátrix (tömb)
- N dimenziós vektor: adatsor



Vekotorok az octave-ban

- Sorvektor:

```
octave:##> a=[1 4 5]
a = 1 4 5
octave:##> b=[2,1,0]
b = 2 1 0
```

- Oszlopvektor:

```
octave:##> c=[4;7;10]
c =
4
7
10
```

- A vektorokon végzett műveletekben jelent majd különbséget!

Transzponálás

- Oszlop és sorvektor átalakítás, transzponálás: ’

```
octave:50> a=[1  4  5]
a = 1   4   5
octave:51> b=a’
b =
1
4
5
```

Vektorok: folytatás

- Kiegészítés

```
octave:##> a=[1  4  5]
a = 1  4  5
octave:##> d=[a  6]
d = 1  4  5  6
```

- Automatikus számolás

```
octave:##> e=2:6
e = 2  3  4  5  6
octave:##> e=2:0.3:4
e = 2.0000  2.3000  2.6000  2.9000  3.2000  3.5000  3.8000
```

Speciális vektorok

```
octave:37> zeros(1,5)
ans = 0  0  0  0  0
octave:38> ones(1,5)
ans = 1  1  1  1  1
octave:39> linspace(1,10,5)
ans = 1.0000   3.2500   5.5000   7.7500  10.0000
octave:40> logspace(1,9,5)
ans = 10        1000      100000    10000000  1000000000
```

A vektor elemei

```
octave:##> a=[1:2:6 -1 0]
a = 1 3 5 -1 0
octave:##> a(3)
ans = 5
octave:##> a(3:5)
ans =
5 -1 0
octave:##> a(1:2:5)
ans =
1 5 0
octave:45> p=[1:2:5]
p = 1  3  5
octave:46> a(p)
ans = 1  5  0
```

Műveletek vektorokkal

- Szorzás skalárral

```
octave:##> a * 2
ans = 2 6 10 -2 0
```

- Elemenkénti műveletek: .+, .-, .*, ./

```
octave:##> b=[1 2 3 4 5];
octave:##> a.*b
ans = 1 6 15 -4 0
octave:##> b .^ 2
ans = 1 4 9 16 25
octave:##> 2 .^ b
ans = 2 4 8 16 32
```

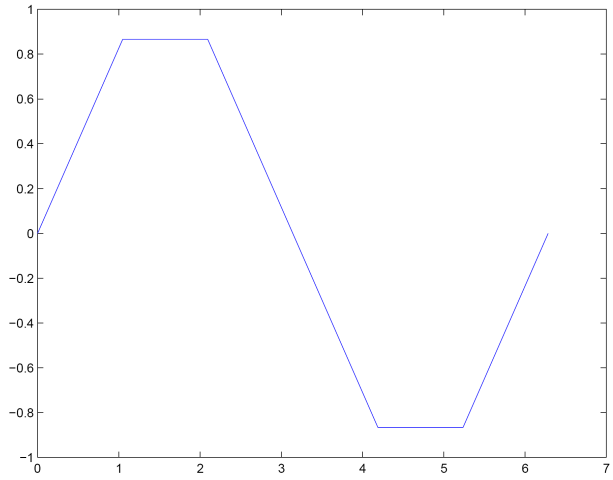
Műveletek vektorokkal

- Függvények is hatnak elemenként

```
octave:##> angles=[0:pi/3:2*pi]
angles = 0 1.0472 2.0944 3.1416
4.1888 5.2360 6.2832
octave:##> y=sin(angles)
y = 0 0.8660 0.8660 0.0000 -
0.8660 -0.8660 -0.0000
```

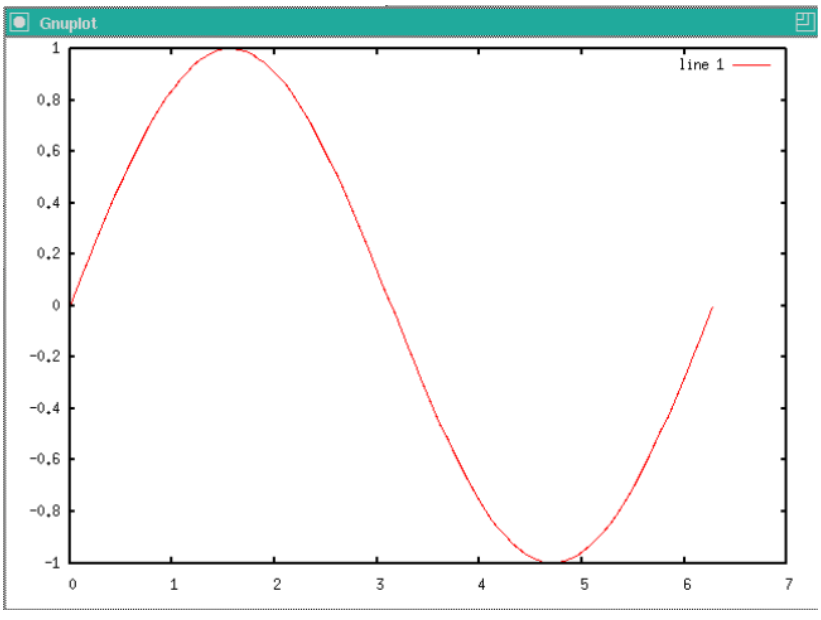
Vektor mint adatsor

- Ábrázolás
  octave:##> plot(angles,y)



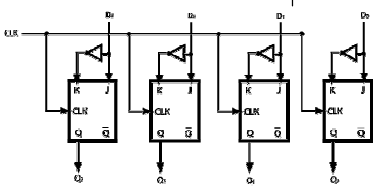
Vektor mint adatsor

```
octave:##> angles=linspace(0,2*pi,100);
octave:##> y=sin(angles);
octave:##> plot(angles, y);
```



Számábrázolás

- Tízes számrendszer:
  $12.25 = 1 \times 10^{1} + 2 \times 10^{0} + 2 \times 10^{-1} + 5 \times 10^{-2}$
- A digitális számítógép áramkörei két állapotúak:
  - van feszültség – nincs feszültség
  - 1 0
- 8 bit = 1 byte : kettes számrendszerben 0-255 egész szám ábrázolható



0	0	0	0	1	0	0	1
7	6	5	4	3	2	1	0

  $= 0\times2^{7}+0\times2^{6}+0\times2^{5}+0\times2^{4}+1\times2^{3}+0\times2^{2}+0\times2^{1}+1\times2^{0} = 8+1 = 9$
- 1 biten jelölhetjük az előjelet, pl. 4 byte = 32 bit egész számok:
  -2147483648 ≤ x ≤ 2147483647
- Egy lehetőség törtek ábrázolására: fixpontos ábrázolás
1	1	0	1	0	1	0	0
3	2	1	0	-1	-2	-3	-4

  $=1\times2^{3}+1\times2^{2}+0\times2^{1}+1\times2^{0}+0\times2^{-1}+1\times2^{-2}+0\times2^{-3}+0\times2^{-4}=8+4+1+1/4=13.25$

$$N_{\text{fix}} = \text{sign} \times (\alpha_n 2^n + \alpha_{n-1} 2^{n-1} + \cdots + \alpha_0 2^0 + \cdots + \alpha_{-m} 2^{-m})$$

Lebegőpontos számok

- Fixpontos számok hátránya
  - Nem ábrázolhatók egyszerre nagyon nagy és nagyon kicsi számok pontosan
  - Bankároknak jó – tudósoknak nem
- „Tudományos” számábrázolás zsebszámológépen
  - *+ 0.299792458 E+09*

előjel | mantissza | kitevő/karakterisztika

- Számítógép lebegőpontos számok

előjel

$(-1)^s \times f \times 2^{e-b}$

mantissza | karakterisztika

- 32 bit esetén pl.:

s	e	f
31	30-23	22-0

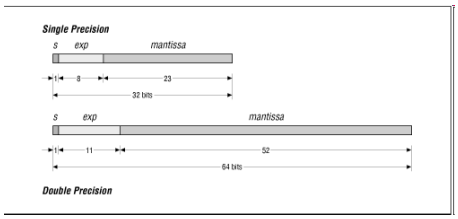
IEEE 754 szabvány

- 32 bit: egyszerű pontosság: *float*

Number Name	*Values of s, e, and f*	*Value of Single*
Normal	$0 < e < 255$	$(-1)^s \times 2^{e-127} \times 1.f$
Subnormal	$e = 0, \quad f \neq 0$	$(-1)^s \times 2^{-126} \times 0.f$
Signed zero (± 0)	$e = 0, \quad f = 0$	$(-1)^s \times 0.0$
$+\infty$	$s = 0, \quad e = 255, \quad f = 0$	`+INF`
$-\infty$	$s = 1, \quad e = 255, \quad f = 0$	`-INF`
Not a number	$s = u, \quad e = 255, \quad f \neq 0$	`NaN`

- 64 bit: dupla pontosság: *double*

Number Name	*Values of s, e, and f*	*Value of Double*
Normal	$0 < e < 2047$	$(-1)^s \times 2^{e-1023} \times 1.f$
Subnormal	$e = 0, \quad f \neq 0$	$(-1)^s \times 2^{-1022} \times 0.f$
Signed zero	$e = 0, \quad f = 0$	$(-1)^s \times 0.0$
$+\infty$	$s = 0, \quad e = 2047, \quad f = 0$	`+INF`
$-\infty$	$s = 1, \quad e = 2047, \quad f = 0$	`-INF`
Not a number	$s = u, \quad e = 2047, \quad f \neq 0$	`NaN`



$(-1)^s \times f \times 2^{e-b}$

Number Name	*Values of s, e, and f*	*Value of Double*
Normal	$0 < e < 2047$	$(-1)^s \times 2^{e-1023} \times 1.f$
Subnormal	$e = 0, \quad f \neq 0$	$(-1)^s \times 2^{-1022} \times 0.f$
Signed zero	$e = 0, \quad f = 0$	$(-1)^s \times 0.0$
$+\infty$	$s = 0, \quad e = 2047, \quad f = 0$	`+INF`
$-\infty$	$s = 1, \quad e = 2047, \quad f = 0$	`-INF`
Not a number	$s = u, \quad e = 2047, \quad f \neq 0$	`NaN`

Példa: 64 bit, b=1023, s:1,e:11,f:23 bit

0 01111111111 0000 (+még 48db 0) = +1 × $2^{1023-1023}$ × 1.0_2 = 1.0
1 01111111111 0000 (+még 48db 0) = +1 × $2^{1023-1023}$ × 1.0_2 = -1.0
0 01111111111 1000 (+még 48db 0) = +1 × $2^{1023-1023}$ × 1.1_2 = 1.5
0 10000000000 0000 (+még 48db 0) = +1 × $2^{1024-1023}$ × 1.0_2 = 2.0
0 10000000001 1010 (+még 48db 0) = +1 × $2^{1025-1023}$ × 1.1010_2 = 6.5

- Normalizált számok: akkor nem veszítünk értékes biteket, ha a mantissza teljesen „ki van töltve” azaz a legnagyobb helyiértéken 1 áll: ezért vegyük ezt fix 1-nek, nem is kell eltárolni, csak a maradék számjegyeket
- Nem normalizált (subnormal) számok: a legkisebb normalizált számoknál kisebb értékeket ábrázolnak, egyre kisebb számokra egyre kisebb relatív pontossággal, hisz a nagyobb helyiértékeken is lehet 0

Egyszerű példa: 8 bit, b=1, s:1, e:3, f:4 bit

$(-1)^s \times f \times 2^{e-b}$

A legnagyobb szám:

0 111 1111 = $+1 \times 2^{7-1} \times 1.1111_2 = 64 \times 1.9375 = 124.0$

A legkisebb normált pozitív szám:

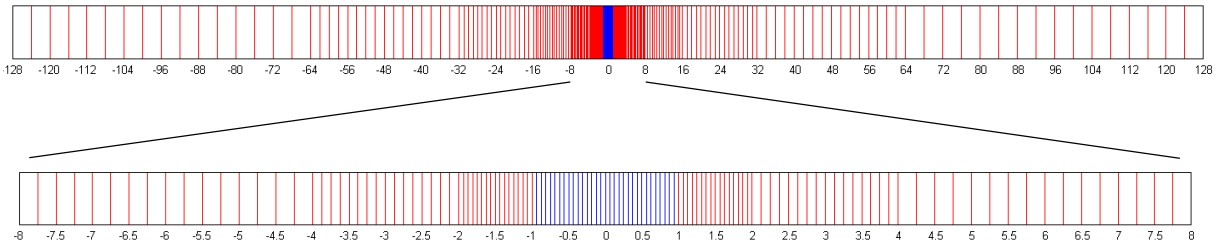
0 001 0000 = $+1 \times 2^{1-1} \times 1.0_2 = 1 \times 1.0 = 1.0$

A legnagyobb nem normált szám:

0 000 1111 = $+1 \times 2^{0} \times 0.1111_2 = 0.9375$

A legkisebb pozitív nem normált szám:

0 000 0001 = $+1 \times 2^{0} \times 0.0001_2 = 0.0625$



Egyszerű példa, jobb felbontás: 8 bit, b=3, s:1, e:3, f:4 bit

$(-1)^s \times f \times 2^{e-b}$

A legnagyobb szám:

0 111 1111 = $+1 \times 2^{7-3} \times 1.1111_2 = 16 \times 1.9375 = 31$

A legkisebb normált pozitív szám:

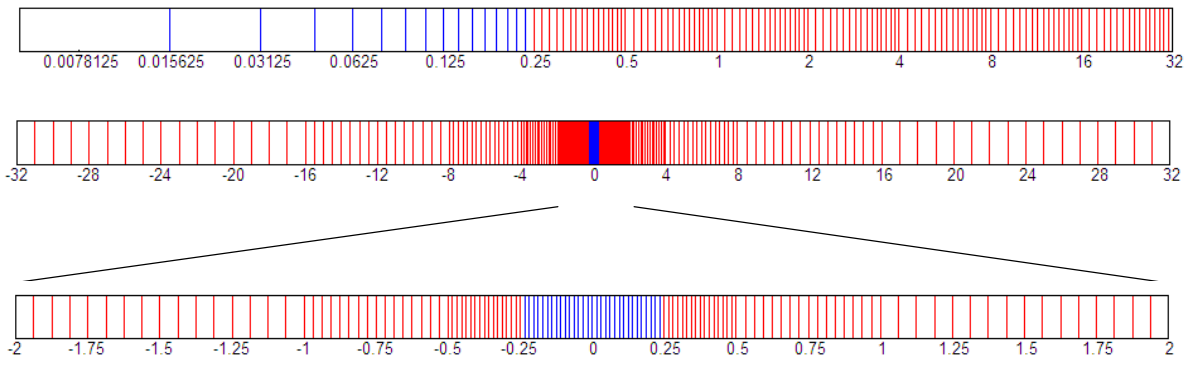
0 001 0000 = $+1 \times 2^{1-3} \times 1.0_2 = 0.25 \times 1.0 = 0.25$

A legnagyobb nem normált szám:

0 000 1111 = $+1 \times 2^{-2} \times 0.1111_2 = 0.234375$

A legkisebb pozitív nem normált szám:

0 000 0001 = $+1 \times 2^{-2} \times 0.0001_2 = 0.015625$



Lebegőpontos számok

- Csak véges értékek ábrázolhatóak
  - 32 bit (float): $1.4\times10^{-45} \le x \le 3.4\times10^{38}$
  - 64 bit (double): $4.9\times10^{-324} \le x \le 1.8\times10^{308}$
  - **túlcsordulás, alulcsordulás**
- Csak véges számú szám ábrázolható
  - véges a „felbontás”
  - 32 bit: 6-7 tizedes jegy
  - 64 bit: 15-16 tizedes jegy
  - **kerekítési hiba**
- Gépi pontosság:

$$1_c+\varepsilon_m = 1_c$$

(jelölés: gép által ábrázolt $1 \equiv 1_c$)

```
octave:##> 1+1e-16
ans = 1
```

  - float: ε_m =`1.19209e-07`
  - double: ε_m = `2.2204e-16`

Számábrázolás: octave

- A véges ábrázolás miatti kerekítési hiba

```
octave:##> 1 - 0.2 - 0.2 - 0.2 - 0.2 - 0.2
ans =  5.5511e-17
```

- Túlcsordulás:

```
octave:##> 1e308+1e308
ans = Inf
```

  - Végtelen: `inf`
  - 0/0: `nan`
- Számábrázolási pontosság: `eps`

```
octave:##> eps
eps =  2.2204e-16
```

- Legnagyobb, és legkisebb szám amit a gép ábrázolni tud:

```
octave:##> realmin, realmax
realmin =  2.2251e-308
realmax =  1.7977e+308
```

Hibák felerősödése: kivonás

- Véges számábrázolás miatt:

$$x_c = x(1+\varepsilon_x)$$

- Különbség hibája:

$$a = b - c \Rightarrow a_c \cong b_c - c_c \cong b(1+\varepsilon_b) - c(1+\varepsilon_c)$$

$$\frac{a_c}{a} \cong 1 + \varepsilon_b \frac{b}{a} - \varepsilon_c \frac{c}{a}$$

- Probléma, ha azonos nagyságrendű számokat vonunk ki, a relatív hibája nagy lesz :

$$\frac{a_c}{a} \equiv 1 + \varepsilon_a \cong 1 + \frac{b}{a}(\varepsilon_b - \varepsilon_c) \cong 1 + \frac{b}{a}\max(|\varepsilon_b|, |\varepsilon_c|)$$

- Mivel $b \approx c \Rightarrow \frac{b}{a}$ nagy, tehát a relatív hiba nagy.

Különbségi hiba: példa

$ax^2 + bx + c = 0$ megoldó-képlet: $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Ha $b^2 >> 4ac \Rightarrow \sqrt{b^2 - 4ac} \approx |b|$ vagyis az első gyök pontatlan!

Szorozzuk meg a számlálót és nevezőt $-b \mp \sqrt{b^2 - 4ac}$ -vel!

$x'_{1,2} = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$ matematikailag ekvivalens megoldó-képlet.

Ekkor, a fenti helyzetben az első gyök pontos lesz, viszont a második pontatlan. Használjuk tehát az elő gyökhöz a 2. képletet, a második gyökhöz pedig az elsőt!
Megj.:

1. Ugyanezt a képletet kapjuk, ha 1/x-re írunk fel 2-od fokú egyenletet.
2. Programban használható praktikus változat:

$$q := -(b + sign(b)\sqrt{b^2 - 4ac})/2, \quad x_1 := q/a, \quad x_2 := c/q$$

3. További példa numerikus hiba stabilizálására

$$\text{ha } x \gg 1, \quad \sqrt{x+1} - \sqrt{x} = \frac{1}{\sqrt{x+1} + \sqrt{x}}$$

```
octave:##> a=1; b=1; c=1e-12;
octave:##> x1=(-b+sqrt(b*b-4*a*c))/2*a
x1 =  -9.99977878279878e-13
octave:##> x2=(-b-sqrt(b*b-4*a*c))/2*a
x2 = -0.999999999999000
octave:##> xx1=-2*c/(b+sqrt(b*b-4*a*c))
xx1 =  -1.00000000000100e-12
octave:##> xx2=-2*c/(b-sqrt(b*b-4*a*c))
xx2 = -1.00002212220950
octave:##> a*x1*x1+b*x1+c
ans =  2.21217211215324e-17
octave:##> a*x2*x2+b*x2+c
ans =  2.21217201214839e-17
octave:##> a*xx1*xx1+b*xx1+c
ans = 0
octave:##> a*xx2*xx2+b*xx2+c
ans =  2.21226998947887e-05
```

```
octave:##> a=1; b=100; c=1e-12;
octave:##> x1=(-b+sqrt(b*b-4*a*c))/2*a
x1 =  -7.10542735760100e-15
octave:##> x2=(-b-sqrt(b*b-4*a*c))/2*a
x2 = -100
octave:##> xx1=-2*c/(b+sqrt(b*b-4*a*c))
xx1 =  -1.00000000000000e-14
octave:##> xx2=-2*c/(b-sqrt(b*b-4*a*c))
xx2 = -140.737488355328
octave:##> a*x1*x1+b*x1+c
ans =  2.89457264239900e-13
octave:##> a*x2*x2+b*x2+c
ans =  1.00000000000000e-12
octave:##> a*xx1*xx1+b*xx1+c
ans = 0
octave:##> a*xx2*xx2+b*xx2+c
ans = 5733.29179303328
```

Olvasmány

- Stoyan G. Numerikus matematika 7-20. o.
- Stoyan G. MATLAB könyv, 1-20.o., 28-38.o. és 373-376.o.
- P.J.G. Long: Octave Tutorial, 4-17. o.