KIV/ZEP - 2012

Robustnost a matematické výpočty

- robustnost algoritmů
  - numerická vs. singulární případy
- numerická robustnost
  - výsledek výpočtu na počítači je výrazně odlišný od skutečného výsledku
  - hlavní důvod: omezená přesnost reprezentace
    - zaokrouhlení vstupních hodnot $\rightarrow$ malá chyba
    - chyba narůstá během výpočtu
      - číslo 0.1 je periodické – viz *float x double x DECIMAL*

```
Float sum: 615798996992.000000
Double sum: 615852916691.270870
Decimal sum: 615852916691.4
Chyba float -> double: 53919699.270874
double -> decimal: -0.129
```

Numerická robustnost

- proslulé „nehody“
  - zaokrouhlování na burze ve Vancouver (1982)
    - opakované zaokrouhlování na 3 desetinná místa
    - po 22 měsících burzovní index 524.881 namísto skutečného 1009.811
  - násobení číslem 0.1 u obranného protiraketového systému USA (1992)
    - celočíselný čítač inkrementující se každých 0.1s
    - pro výpočet času čítač násoben čítač 0.1 a na základě toho vypočteno zaměřování rakety
    - chyba 0.343 s, když systém online po dobu 100h
    - výsledek: raketa minula nepřátelskou raketu mířící na US základnu $\rightarrow$ 28 mrtvých, 100 zraněných

Numerická robustnost

- proslulé „nehody“
  - přetečení čísla u rakety Ariane (1996)
    - rychlost zkonvertována z 64-bitového čísla na 16-bitové
    - výsledkem nesmyslný údaj použitý pro řízení a shoření rakety
    - škoda 500 millionů USD

Numerická robustnost

- je-li numerika takový problém, tak proč vůbec používat jednoduchou přesnost?

Numerická robustnost

- reálná naměřená data často veliká
  - např. volumetrická data (CT), vektorová pole
- reálná data již zatížena chybou měření
  - např. tisícina mm
- $\rightarrow$ jednoduchá přesnost
    - poloviční velikost
    - rychlejší (Intel preferuje)
    - 7 desetiných míst je pro uchování OK
      - často jiné jednotky než SI (např. ms, mm)
      - POZOR na fyzikální jednotky ve výpočtech!

Numerická robustnost

- proslulé „nehody“
  - Mars Climate Orbiter (1998)
    - některé algoritmy vyvinuté v UK jiné v US
    - UK kód pracuje v palcích, apod. US kód v SI jednotkách
    - sonda se zřítila na Mars
    - škoda 125 millionů USD

Numerická robustnost

```
void ComputePoints(int N, double* x, double* y)
{
        double delta_fi = 2*M_PI / N;

        N = 0;
        double fi = 0.0;
        while (fi != 2*M_PI)
        {
                x[N] = cos(fi);
                y[N] = sin(fi);
                fi += delta_fi;
                N++;
        }
}
```

Numerická robustnost

- porovnávání dvou reálných čísel
  - porovnávat jen intervalově (<, >)
  - použití ε okolí:
    - $x = y \leftrightarrow |x - y| < \varepsilon$
    - jak volit ε, pokud nic nevím o možném rozsahu vstupních dat?
      - ε dáno počtem platných míst (v mantise)

```
//------------------------------------------------------------
bool mafEquals(double x, double y)
//------------------------------------------------------------
{
   double diff=fabs(x - y);
   double max_err=fabs(x / pow((double)10,(double)15));
   if (diff > max_err)
     return false;
   return  ( diff <= max_err);
}
```

Numerická robustnost

- kumulace chyby při výpočtech
  - příklad (v desítkové reprezentaci): $x = 10.12$, $y = 9.933$, přesnost na 2 desetinná místa
  - čísla se uloží normalizovaně jako: $\tilde{x} = 1.01 \times 10^1$, $\tilde{y} = 9.93 \times 10^0$
  - počítejme: $\tilde{z} = \tilde{x} - \tilde{y}$
  - nejprve musíme sjednotit exponenty, abychom to mohli spočíst: $\tilde{z} = 1.01 \times 10^1 - 0.993 \times 10^1$, ale druhé čílo by již vyžadovalo 3 místa! $\rightarrow$ musíme provést další zaokrouhlení $\tilde{y}$ v kontextu $\tilde{x}$, tedy $\tilde{z} = 1.01 \times 10^1 - \ 0.99 \times 10^1 = 0.02 \times 10^1$,
  - po normalizaci: $\tilde{z} = 2.00 \times 10^{-1}$
  - ve skutečnosti $1.01 \times 10^1 - 0.993 \times 10^1 = 1.70 \times 10^{-1}$
  - v rámci přesnosti na 2 desetinná místa tedy máme chybu: $2.00 - 1.70 = 0.30 \ \rightarrow 30\,ulps$

Numerická robustnost

- problémy vznikají zejména při
  - odečítání dvou blízkých čísel
    - $x^2 - y^2$ méně přesné než $(x + y) \cdot (x - y)$
      - pokud však $x \gg y$ nebo $x \ll y$, pak je tomu naopak
    - příklad: obsah plochého trojúhelníka přes Heronův vzorec: $s = \frac{(a+b+c)}{2}, A = \sqrt{s \cdot (s-a) \cdot (s-b) \cdot (s-c)}$
    - $a = 9, b = c = 4.53$
    - $a + b = 13.53 \rightarrow po\ normalizaci\ a\ zaokr. \rightarrow 1.35 \times 10^1$
    - $1.35 \times 10^1 + c = 1.35 \times 10^1 + 4.53 \times 10^0 = 1.8 \times 10^1$
    - $s = 9 \rightarrow A = 0$, ve skutečnosti $s = 9.03, A = 2.342 \ldots$, takže zatímco $s_{err} = 1\,ulps, A_{err} = 234.2 \ldots ulps$

Numerická robustnost

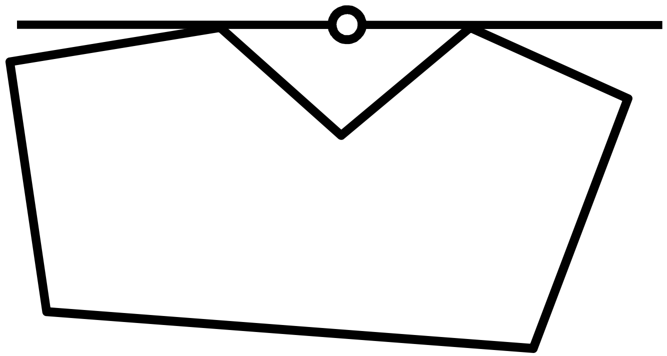
- problémy vznikají zejména při
  - odečítání dvou blízkých čísel
    - řešení: upravený Heronův vzorec:
    - $A = \frac{\sqrt{(a+(b+c))\cdot(c-(a-b))\cdot(c+(a-b))\cdot(a+(b-c))}}{4}, a \geq b \geq c$
    - výsledek: $A = 2.35$, tj. $A_{err} \cong 0.8\, ulps$
  - dělení číslem blízkým nule
    - příklad: Gauss-Seidlova eliminační metoda
    - řešení: pivotace řádků, eliminuji podobné

Numerická robustnost

- základní pravidla pro robustní kód
  - čím méně operací tím lépe
  - vyhnout se používání složitých nepřesných funkcích jako je sin, cos, ln, apod., lze-li výsledek stanovit jiným způsobem
  - Hornerovo schéma
    - polynom: $a_n x^n + \dots a_1 x + a_0$ počítán jako: $a_0 + x(a_1 + x(\dots x(a_{n-1} + a_n x)\dots)$
    - složitost?
  - využít robustnější metody
    - *příklad: kružnice opsaná trojúhelníku*
  - ukládat ve float (je-li vhodnější než double), ale počítat v double nebo vlastní spec. aritmetice

Numerická robustnost

- program spadne pro neočekávaný vstup ☺
  - velmi často: jednoprvkové pole, něco je *null*
  - řešení (ne vždy): výjimky a testování
- algoritmus v některých (neočekávaných) případech poskytuje chybné výsledky
    - např. ray-crossing lokace bodu
  - možné řešení: modlit se
    - často nemá smysl řešit, pokud pravděpodobnost výskytu takového vstupu je téměř nulová
  - jiné řešení: jiný (často složitější) algoritmus



Singulární případy

- citlivost na vstupních datech
  - algoritmus se nechová dobře pro určitý typ dat
    - např. body v clusterech vs. mřížka, seřazená posloupnost čísel (pro strom)
  - možné řešení: randomizace

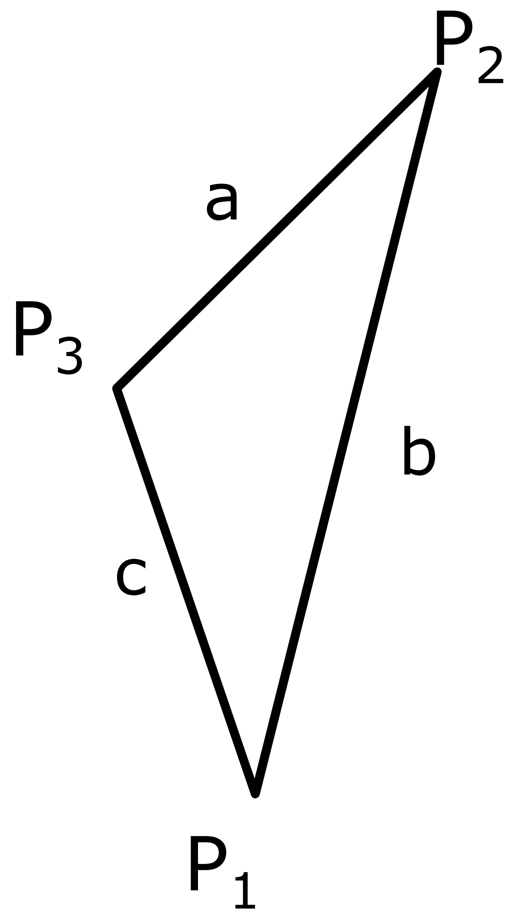
Singulární případy

- obsah trojúhelníka

$$A = \sqrt{s \cdot (s-a) \cdot (s-b) \cdot (s-c)}$$

$$s = \tfrac{1}{2} \cdot (a+b+c)$$

$$A = \frac{1}{2} \cdot \left| \det \begin{pmatrix} P_1^x & P_2^x & P_3^x \\ P_1^y & P_2^y & P_3^y \\ 1 & 1 & 1 \end{pmatrix} \right|$$

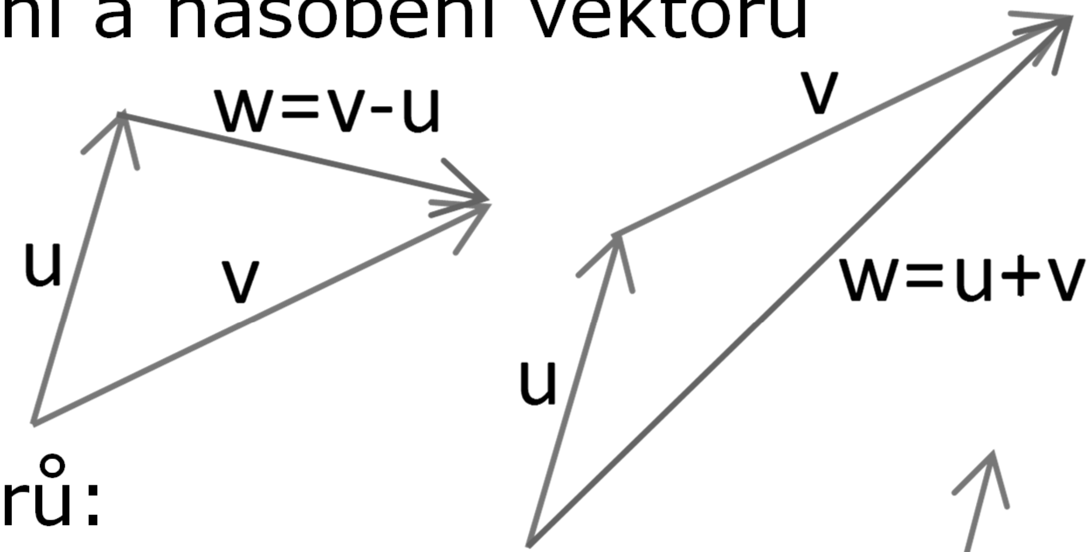


Vektorový počet

Operace	Instrukce	Latence
sčítání / odčítání	FADD / FSUB	5
násobení	FMUL	7
dělení (float)	FDIV	23
dělení (double)	FDIV	38
absolutní hodnota	FABS	1
druhá odmocnina	FSQRT	58
cos / sin	FCOS / FSIN	119
$\tan^{-1}$	FPATAN	147
$\cos^{-1}$ / $\sin^{-1}$	neexistuje, počítá se přes $\tan^{-1}$ a druhou odmocninu	> 220

Vektorový počet

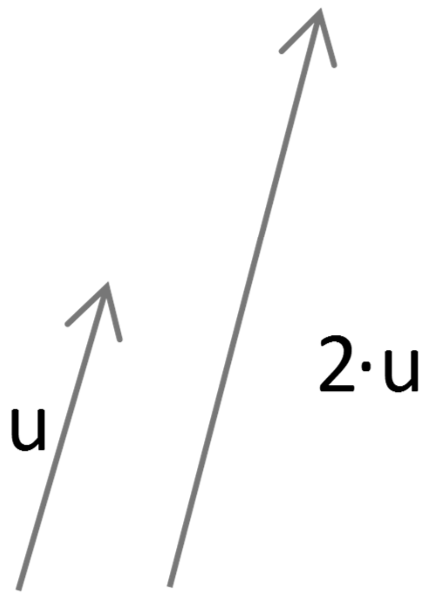
- sčítání, odčítání a násobení vektorů



- velikost vektorů:

$$\left|\vec{u}\right| = \sqrt{u_x \cdot u_x + u_y \cdot u_y + u_z \cdot u_z}$$

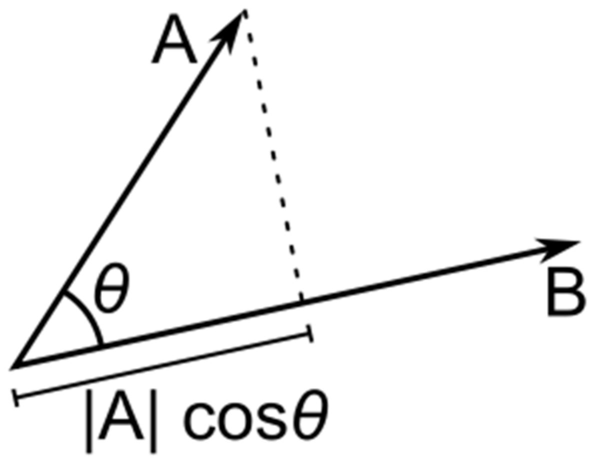
- normalizace vektorů: $\hat{u} = \dfrac{\vec{u}}{\left|\vec{u}\right|}$



Vektorový počet

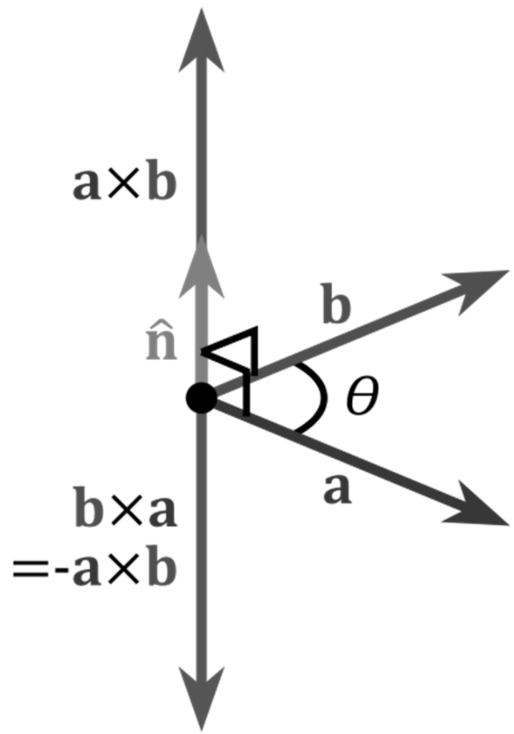
- skalární součin

$$\vec{a}\cdot\vec{b} = \left|\vec{a}\right|\cdot\left|\vec{b}\right|\cdot\cos\theta$$



- vektorový součin

$$\vec{a}\times\vec{b} = \left|\vec{a}\right|\cdot\left|\vec{b}\right|\cdot\vec{n}\cdot\sin\theta$$



Vektorový počet

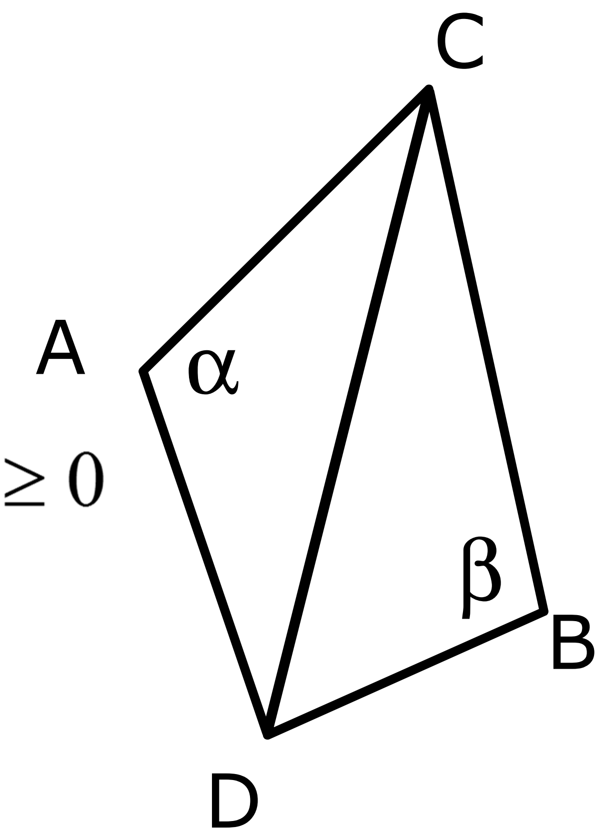
- příklad: *„ošklivý" pár trojúhelníků*
  - α + β ≥ 180°

$$a^2 = b^2 + c^2 - 2bc \cdot \cos\alpha$$

$$\Rightarrow \alpha = \cos^{-1}\left(\frac{b^2 + c^2 - a^2}{2bc}\right)$$

$$\alpha + \beta \geq 180^\circ \Leftrightarrow \cot\beta + \cot\alpha \geq 0$$

$$\cot\alpha = \frac{\vec{u} \cdot \vec{v}}{\left|\vec{u} \times \vec{v}\right|}$$



Vektorový počet