

# PA081: Programování numerických výpočtů

## 12. Automatické vyhodnocení derivací

jaro 2012

- ▶ kromě funkce dokážeme vyhodnotit i její derivace
  - ▶ první, druhé, ...
  - ▶ parciální
- ▶ mnohé numerické problémy jsou lépe řešitelné metodami s dostupnou derivací
- ▶ metody s derivací zpravidla rychleji konvergují
  - ▶ Newton vs. metoda sečen

- ▶ kromě funkce dokážeme vyhodnotit i její derivace
  - ▶ první, druhé, ...
  - ▶ parciální
- ▶ mnohé numerické problémy jsou lépe řešitelné metodami s dostupnou derivací
- ▶ metody s derivací zpravidla rychleji konvergují
  - ▶ Newton vs. metoda sečen
- ▶ aproximace konečnou diferencí nestačí
- ▶ zpracovávaná funkce nebývá triviální
- ▶ analytické vyjádření derivace je náročné
  - ▶ zdroj nebezpečných chyb
  - ▶ komplikace při změnách

# Automatické derivování

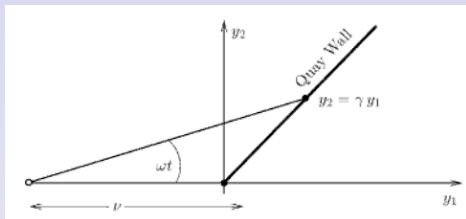
- ▶ funkce je implementována jako program
- ▶ hledáme program, který vedle funkční hodnoty spočte i derivace
- ▶ <http://www.autodiff.org>

Automatické  
derivování

Obrácené  
vyhodnocení

# Automatické derivování

- ▶ funkce je implementována jako program
- ▶ hledáme program, který vedle funkční hodnoty spočte i derivace
- ▶ <http://www.autodiff.org>
- ▶ příklad s majákem



- ▶ bod dopadu paprsku na nábřeží

$$y_1 = \frac{\nu \tan \omega t}{\gamma - \tan \omega t} \quad y_2 = \gamma y_1$$

- ▶ vyjádřeno jako jednoduchý program

$$a := \tan \omega t; \quad y_1 := va/(y - a); \quad y_2 := \gamma y_1;$$

- ▶ chceme spočítat i rychlost pohybu, tj. derivaci v čase
- ▶ výpočet současně s původní funkcí

$$\begin{array}{ll} \dot{t} := 1 & \\ a_1 := \omega t & \dot{a}_1 := \omega \dot{t} \\ a := \tan a_1 & \dot{a} := \dot{a}_1 / (\cos a_1)^2 \\ a_2 := y - a & \dot{a}_2 := -\dot{a} \\ y_1 := va/a_2 & \dot{y}_1 := v(\dot{a}a_2 - a\dot{a}_2)/a_2^2 \\ y_2 := \gamma y_1 & \dot{y}_2 := \gamma \dot{y}_1 \end{array}$$

- ▶ pouze mechanická aplikace základních pravidel

# Automatické derivování

- ▶ transformace zdrojového kódu (např. ADIC)
- ▶ přetížení operátorů a funkcí (ADOL-C)
- ▶ zjednodušený příklad

Automatické  
derivováníObrácené  
vyhodnocení

```
class adouble { double x; double dx; ... }

adouble operator * (adouble a, adouble b) {
    adouble r;
    r.x = a.x * b.x;
    r.dx = a.x * b.dx + a.dx * b.x;
    return r;
}

adouble v1, v2, v3;
...
v1 = v2 * v3;
```

- ▶ aktivní proměnná
  - ▶ nezávislé proměnné, podle kterých chceme derivovat
  - ▶ všechny další proměnné na nich během výpočtu závisející
- ▶ vyhrazený typ `adouble`
- ▶ přiřazení `adouble` → `double` je chybné
  - ▶ ztrácíme informaci o derivacích tam, kde je potřebná
  - ▶ explicitní přístup metodou `value()`
- ▶ více nezávislých proměnných
  - ▶ s každou aktivní proměnnou udržujeme vektor parciálních derivací



# Typický program

```
#define NUMBER_DIRECTIONS 3
#define ADOLC_TAPELESS
#include <adolc/adouble.h>

func(adouble[], const adouble[]);

adouble x[3];
adouble y[4];
...
for (i=0; i<3; i++)
    for (j=0; j<3; j++)
        x[i].setADValue(j,i==j ? 1.0 : 0.0);

func(y,x);

for (i=0; i<4; i++) {
    cout << y.getValue() << ": ";
    for (j=0; j<3; j++) cout << y.getADValue(j) << ", ";
    cout << endl;
}
```

Automatické  
derivování

Obrácené  
vyhodnocení

# Není to tak jednoduché

- ▶ nediferencovatelné funkce (**fmod**)
  - ▶ nelze použít
- ▶ absolutní hodnota (**fabs**)
  - ▶ v 0 pravá nebo levá derivace  $\pm 1$
  - ▶ případně položíme uměle rovnu 0
- ▶ minimum a maximum (**fmin**, **fmax**)
  - ▶ lze nahradit  $(a + b \pm |a - b|)/2$
- ▶ nedefinované na celém  $\mathbb{R}$  (**sqrt**, **pow**, **tan**, ...)
  - ▶ derivaci položíme uměle **NaN**, resp. **Inf**

# Obrácené vyhodnocení

- ▶ množství potřebných operací  $O(mk)$  pro  $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$ 
  - ▶  $m$  počet nezávislých proměnných
  - ▶  $n$  počet závislých proměnných
  - ▶  $k$  počet kroků výpočtu
- ▶ nepříjemné s rostoucím  $m$
- ▶ velká třída problémů velkým  $m$  ale malým  $n$ 
  - ▶ optimalizace funkce více proměnných ( $n = 1$ )

Automatické  
derivováníObrácené  
vyhodnocení

- ▶ množství potřebných operací  $O(mk)$  pro  $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$ 
  - ▶  $m$  počet nezávislých proměnných
  - ▶  $n$  počet závislých proměnných
  - ▶  $k$  počet kroků výpočtu
- ▶ nepříjemné s rostoucím  $m$
- ▶ velká třída problémů velkým  $m$  ale malým  $n$ 
  - ▶ optimalizace funkce více proměnných ( $n = 1$ )
- ▶ počítali jsme citlivost pomocných proměnných na změny vstupu
- ▶ obrácený postup, citlivost výstupu (závislých proměnných) na změny pomocných
- ▶ výpočet od konce

# Obrácené vyhodnocení

- ▶ např. funkce  $y = \sin(x_1/x_2)e^{x_3}$
- ▶ označujeme  $\bar{a} = \partial y / \partial a$

$$a_1 := x_1/x_2$$

$$a_2 := \sin a_1$$

$$a_3 := e^{x_3}$$

$$y := a_2 a_3$$

$$\bar{y} := 1$$

$$\bar{a}_2 := a_3 \bar{y}$$

$$\bar{a}_3 := a_2 \bar{y}$$

$$\bar{x}_3 := a_3 \bar{a}_3$$

$$\bar{a}_1 := \bar{a}_2 \cos a_1$$

$$\bar{x}_2 := -\bar{a}_1/x_2$$

$$\bar{x}_1 := \bar{a}_2/x_2$$

- ▶ v  $O(nk)$  operacích máme všechny  $\partial y / \partial x_i$

Automatické  
derivováníObrácené  
vyhodnocení

- ▶ derivace se vyhodnocují od konce
  - ▶ tj. až poté, co proběhl výpočet funkce
  - ▶ proto „obrácené“ (reverse) vyhodnocení
- ▶ výpočet funkce je nutné nějak zaznamenat
- ▶ ADOL-C: datová struktura (i soubor) **páska** (tape)
- ▶ záznam posloupnosti operací, nikoli konkrétních hodnot
  - ▶ lze recyklovat pro jiný vstup
  - ▶ případné rozdílnosti vyhodnocení ( $a > b ? c : d$ ) jsou detekovány
- ▶ použije se pro vyhodnocení funkce, gradientu/Jakobiánu, Hessiánu, ...

# Instrumentace pro zpětné vyhodnocení

```
#include <adolc/adouble.h>

void func(
    double px[4],    /* vstup */
    double py[4],    /* vystup */
) {
    adouble x[4],y[4],aux;
    trace_on(0);      /* paska 0 */
    for (i=0; i<4; i++) x[i] <<= px[i];
    aux = x[1] * x[4] + exp(x[2]);    /* vlastni vypocet */
    ...
    y[4] = sin(aux);
    for (i=0; i<4; i++) y[i] >>= py[i];
    trace_off();
}
```

- ▶ ADOL-C poskytuje celou řadu „driver“ funkcí
  - ▶ vstupem je vždy páska a konkrétní vstupní hodnoty
  - ▶ driver provede výpočet na základě záznamu na pásce
- ▶ funkční hodnota funkce  $\mathbb{R}^n \rightarrow \mathbb{R}^m$ 
  - ▶ `function(short tag, int m, int n, double x[n],double y[m])`
- ▶ gradient funkce  $\mathbb{R}^n \rightarrow \mathbb{R}$ 
  - ▶ `gradient(short tag, int n, double x[n],double g[n])`
- ▶ Jakobián funkce  $\mathbb{R}^n \rightarrow \mathbb{R}^m$ 
  - ▶ `jacobian(short tag, int m, int n, double x[n],double J[m][n])`
- ▶ Hessián (matice druhých derivací) funkce  $\mathbb{R}^n \rightarrow \mathbb{R}$ 
  - ▶ `hessian(short tag, int m, int n, double x[n],double H[n][n])`



- ▶ optimalizované varianty pro Newtonovu metodu
- ▶ řešení různých typů diferenciálních rovnic
- ▶ speciální varianty pro řídké Jakobiány a Hessiány
- ▶ tenzory vyšších derivací
- ▶ implicitně definované funkce

$$G(\mathbf{x}, \mathbf{y}) = 0$$

- ▶ obecné - plná kontrola nad zpracováním pásky

# Rekapitulace obráceného vyhodnocení

- ▶ aktivní proměnné deklarovat jako `adouble`
- ▶ výpočet označit `trace_on()` ... `trace_off()`
- ▶ zavolat instrumentovanou funkci `jednou`
  - ▶ s typickými hodnotami vstupů
  - ▶ vyprodukuje záznam výpočtu - pásku
- ▶ opakovaně volat potřebné drivery
  - ▶ případně na různé vstupy
- ▶ pro jinou posloupnost výpočtu zavolat instrumentovanou funkci s jinou páskou

- ▶ rozsáhlá problematika, řada implementací
- ▶ včetně obsáhlé teorie
- ▶ <http://www.autodiff.org>
  - ▶ přehled nástrojů, FAQ, ...
- ▶ <https://projects.coin-or.org/ADOL-C>
  - ▶ implementace a dokumentace ADOL-C
  - ▶ včetně řady příkladů
- ▶ Griewank, Walter. *Evaluating Derivatives*. 2008