Implementace automatického derivování v systému UFO

Hartman, J.
2007

Institute of Computer Science
Academy of Sciences of the Czech Republic

Implementace automatického derivování v systému UFO

J.Hartman, L.Lukšan

Technical report No. 1002

December 2007

Pod Vodárenskou věží 2, 182 07 Prague 8 phone: +420 2 688 42 44, fax: +420 2 858 57 89,
e-mail:e-mail:ics@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

Implementace automatického derivování v systému UFO

J.Hartman, L.Lukšan [1]

Technical report No. 1002

December 2007

Abstrakt:

Automatic differentiation is an effective method for evaluating derivatives of function, which is defined by a formula or a program. Program for evaluating of value of function is by automatic differentiation modified to program, which also evaluates values of derivatives. Computed values are exact up to computer precision and their evaluation is very quick. In this report, we describe a program realization of automatic differentiation. This implementation is prepared in the system UFO, but its principles can be applied in other systems. We describe, how the operations are stored in the first part of the derivative computation and how this records are effectively used in the second part of the computation.

Keywords:
Automatic Differentiation, Modeling Languages, Systems for Optimization.

[1]This work was supported by the Grant Agency of the Czech Academy of Sciences, project code IAA1030405, and the institutional research plan No. AV0Z10300504. L.Lukšan is also from Technical University of Liberec, Hálkova 6, 461 17 Liberec.

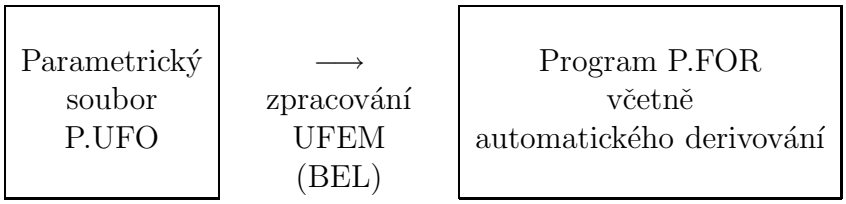
1 Úvod

Automatické derivování je efektivní metoda pro výpočet hodnot derivace funkce, která je zadaná složitým vzorcem nebo programem. Získané hodnoty derivací jsou přesné až na aritmetickou přesnost počítače a výpočet derivací je velmi rychlý. Automatickým derivováním je možné počítat první, druhé nebo vyšší derivace. Při výpočtu Jacobiho nebo Hessovy matice je možné navíc využívat řídkosti matice a rychlost výpočtu tím dále zrychlit. Například v úlohách tvarové optimalizace se setkáváme s úkolem počítat minimum cenového funkcionálu. V průběhu výpočtu se aplikuje Newtonova iterační metoda pro řešení nelineární soustavy, přičemž matice derivací je řídká. Kombinací automatického derivování a řídkosti matice je možné sestavit postup, který několikanásobně urychlí výpočet. V této zprávě budeme na akademickém příkladě demonstrovat až čtyřicetinásobné zkrácení času výpočtu. Program, který vyhodnocuje derivovanou funkci, nemusí být pro výpočet derivace téměř modifikován a celý postup výpočtu derivace je možné automatizovat. Podrobný teoretický popis automatického derivování a jeho aplikace je možné nalézt například v [1], [2] nebo [5].

Cílem této výzkumné zprávy je podat vyčerpávající přehled o tom, jakým způsobem bylo automatické derivování implementováno v interaktivním systému pro univerzální funkcionální optiomalizaci UFO. Je to tedy jednak doklad, který bude sloužit jako manual pro uživatele a jednak základ pro další rozšiřování tohoto systému. Tím jsme zároveň vyjádřili dva základní cíle této výzkumné zprávy. Systém UFO [4], který je vyvíjen v Ústavu informatiky AV ČR, obsahuje přes 2000 fortranských modulů a může být použit nejen k řešení optimalizačních úloh ale i k vývoji a testování optimalizačních metod, případně ke generování samostatných problémově orientovaných optimalizačních programů. S optimalizačními metodami použitými v systému UFO se lze seznámit také v [3] Vzhledem k popularitě a vlastnostem automatického derivování byl systém UFO o tyto postupy rozšířen. Protože se zdaleka nejedná o triviální záležitost, byla mimo jiné vytvořena tato výzkumná zpráva, ve které jsou postup a implementace automatického derivování podrobně popsány. Tato zpráva je také určena pro programátory, kterým by měla usnadnit orientaci v systému UFO a usnadnit používání a rozšíření zabudovaných programů.

V této výzkumné zprávě popíšeme, jak se automatické derivování v systému UFO používá a jak je v tomto systému naimplementováno. V první části stručně shrneme základní teoretické vlastnosti automatického derivování a uvedeme algoritmy a postupy pro výpočet derivací. Ve druhé části popíšeme, jak se automatické derivování v systému UFO používá, vyvolává a jak je začleněno do programu P.FOR, který vznikne po zavolání UFA na vstupní parametrický soubor P.UFO. Těmto souborům odpovídá levá a pravá strana na obrázku 1. Ve třetí, závěrečné části, podrobně uvedeme, jak byl modifikován systém BEL a skupina šablon, aby v systému UFO bylo možné používat automatické derivování, což odpovídá střední části na obrázku 1.

Zpráva je doplněna přílohou ve které jsou podrobně uvedeny výpisy podprogramů, které jsou zabudované v UFU.



Obrázek 1: Schématické znázornění odvození programu systémem UFO.

Druhá část výzkumné zprávy pojednmávající o aplikacích automatického derivování vyžaduje alespoň informativní znalosti o systému UFO, se kterými je možné se seznámit v [4].

2 Základní algoritmy automatického derivování

Aby bylo v následujících kapitolách jasné, jak automatické derivování v systému UFO používáme a implementujeme, uvedeme v této, první části výzkumné zprávy základní teoretické vlastnosti automatického derivování. Popíšeme formule a algoritmy odvozené automatickým derivováním pro výpočet prvních, druhých anebo vyšších derivací.

Podrobnější popis automatického derivování lze najít například v [1], [2] nebo v [5].

2.1 Soupis operací

Nejprve uvedeme formální popis obecné struktury programu, na který se automatické derivování aplikuje.

Program, který chceme transformovat, počítá hodnotu funkce

$$\begin{aligned} f &: D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^m \\ f &: x \longmapsto y = f(x), \end{aligned}$$

Funkce f je složením *základních (elementárních) funkcí* například sčítání, násobení, sinus apod.

Během programu se postupně počítají výsledky jednotlivých základních operací a ty jsou používány jako argumenty dalších základních operací. Každý takový výsledek základní funkce označme v_i.

Všechny číselné hodnoty v_i spočtené během vyhodnocení funkce f v jednom konkrétním bodě označme následujícím způsobem:

$$\underbrace{v_{1-n}, \ldots, v_0}_{x}, v_1, v_2, \ldots, v_{l-m}, \underbrace{v_{l-m+1}, \ldots, v_l}_{y},$$

kde $x = (x_1, \ldots x_n)$ jsou vstupní (nezávislé) proměnné a $y = (y_1, \ldots y_m) = f(x)$ jsou výstupní proměnné.

Řekneme, že $j \prec i$ právě tehdy, když hodnota proměnné v_i závisí *přímo* na proměnné v_j.

Každou hodnotu v_i, $i = 1, \ldots, l$ získáme vyhodnocením *základní funkce* φ_i (elemental function, library function), neboli

$$v_i = \varphi_i(v_j)_{j \prec i}, \tag{1}$$

kde $(v_j)_{j \prec i}$ jsou všechny proměnné, které přímo ovlivňují proměnnou v_i, tj. ty proměnné, ze kterých se počítá hodnota v_i.

Zadaný program lze tedy formálně přepsat do soupisu operací na obrázku 2. V první části jsou proměnným $v_{1-n}, \ldots, v_0$ přiřazeny vstupní hodnoty $x_1, \ldots, x_n$, ve druhé části probíhá výpočet hodnoty $f(x)$ a ve třetí části jsou naplněny výstupní hodnoty $(y_1, \ldots, y_m) = f(x)$.

v_{i-n}	$=$	x_i	pro	$i = 1, \ldots, n$
v_i	$=$	$\varphi_i(v_j)_{j \prec i}$	pro	$i = 1, \ldots, l$
y_{m-i}	$=$	v_{l-i}	pro	$i = m-1, \ldots, 0$

Obrázek 2: Soupis operací.

2.2 Výpočet první derivace - přímý mód

Přímý mód automatického derivování (forward mode, tangent) nám umožní počítat derivace všech výstupních proměnných najednou podle zadaného směru.

Soupis operací na obrázku 2 zderivujeme s pomocí pravidla řetězení. Neboli ke každé základní operaci (1) přidáme novou *sdruženou* operaci

$$\dot{v}_i = \sum_{j \prec i} \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} \cdot \dot{v}_j. \tag{2}$$

která je derivací původní operace (1). Tím jsme ke každé proměnné v_i přiřadili novou *sdruženou* proměnnou $\dot{v}_i$, která je její derivací. Úvodní a závěrečnou část soupisu operací doplníme analogicky. Takto získáme *sdružený* soupis operací, který je na obrázku 3.

Jeho vstupními parametry jsou bod x, ve kterém počítáme derivaci, a vektor $\dot{x}$, který znamená směr požadované derivace. Pokud chceme určit derivaci podle i-té nezávislé proměnné, tj. ve směru i-tého vektoru kartézské báze e_i, vezmeme za $\dot{x}$ právě e_i.

2.3 Výpočet první derivace - zpětný mód

Pokud je f skalární funkce, tj. $m = 1$, pomocí *zpětného* módu automatického derivování (reverse mode, adjoint, gradients) spočítáme její gradient. Pokud je f vektorová funkce, tj. $m > 1$, spočítáme lineární kombinaci gradientů jednotlivých složek f_i, respektive gradient i-té komponenty funkce f při vhodné volbě parametrů.

$$
\left.\begin{array}{lcl}
v_{i-n} & = & x_i \\
\dot{v}_{i-n} & = & \dot{x}_i
\end{array}\right\} \; i = 1, \ldots n
$$

$$
\left.\begin{array}{lcl}
v_i & = & \varphi_i(v_k)_{k \prec i} \\
\dot{v}_i & = & \sum_{j \prec i} \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} \cdot \dot{v}_j
\end{array}\right\} \; i = 1, \ldots l
$$

$$
\left.\begin{array}{lcl}
y_{m-i} & = & v_{l-i} \\
\dot{y}_{m-i} & = & \dot{v}_{l-i}
\end{array}\right\} \; i = m-1, \ldots 0
$$

Obrázek 3: Soupis operací odvozený přímým módem automatického derivování.

Z přímého módu automatického derivování lze Jacobiho matici přepsat jako součin matic, totiž

$$
f'(x) = Q_m A_l A_{l-1} \cdots A_2 A_1 P_n^T. \tag{3}
$$

Nyní však chceme spočítat hodnoty derivace ve tvaru

$$
\bar{x} = \bar{y} \cdot f'(x). \tag{4}
$$

Z těchto vztahů (3) a (4) dostáváme soupis opercí na obrázku 4, ketrý vyhodnocuje $\bar{x} = \bar{y} \cdot f'(x)$. Hodnoty $\bar{v}_j$ lze přitom například interpretovat jako

$$
\bar{v}_j = \sum_{i \; ; \; i \succ j} \bar{v}_i \cdot \frac{\partial \varphi_i}{\partial v_j}. \tag{5}
$$

v_{i-n}	$= x_i$	$i = 1, \ldots, n$
v_i	$= \varphi_i(v_k)_{k \prec i}$	$i = 1, \ldots, l$
y_{m-i}	$= v_{l-i}$	$i = m-1, \ldots, 0$
$\bar{v}_{l-i}$	$= \bar{y}_{m-i}$	$i = 0, \ldots, m-1$
$\bar{v}_i$	$= 0$	$i = l-m, \ldots, 1-n$

for $i = l, \ldots, 1$ do
for $j \prec i$ do
$\bar{v}_j = \bar{v}_j + \bar{v}_i \cdot \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i}$
end for
end for		

$\bar{x}_i$	$= \bar{v}_{i-n}$	$i = n, \ldots, 1$

Obrázek 4: Soupis operací odvozený zpětným módem automatického derivování.

2.4 Výpočet druhých a vyšších derivací

Program pro výpočet druhých (nebo vyšších) derivací je možné odvodit kombinací přímého a zpětného módu. Nejdříve na zadaný program aplikujeme zpětný mód, odkud získáme program pro výpočet gradientu funkce f. Na tento program aplikujeme přímý mód a získáme tak program pro vyhodnocení druhých derivací ve tvaru

$$\bar{y} \cdot f''(x) \cdot \dot{x} + \dot{\bar{y}} \cdot f'(x), \tag{6}$$

kde $\dot{\bar{y}}$ je vstupní parametr, podobně jako $\dot{x}$ nebo $\bar{y}$. Výraz (6) označme jako $\dot{\bar{x}}$. Abychom spočetli pouze druhou derivaci tvaru $\bar{y} \cdot f''(x) \cdot \dot{x}$, je potřeba předem přiřadit $\dot{\bar{y}} = 0$. Výraz $\bar{y} \cdot f''(x) \cdot \dot{x}$ chápeme v této souvislosti jako

$$\bar{y} \cdot f''(x) \cdot \dot{x} = \frac{\partial}{\partial \alpha}\, \bar{y} \cdot f'(x + \alpha \dot{x})\bigg|_{\alpha=0} \in \mathbb{R}^n.$$

Soupis operací získáme tím způsobem, že aplikujeme přímý mód na soupis operací získaný zpětným módem. Z výrazu

$$\bar{v}_j = \bar{v}_j + \bar{v}_i \cdot \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i}$$

přitom odvodíme

$$\dot{\bar{v}}_j = \dot{\bar{v}}_j + \dot{\bar{v}}_i \cdot \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} + \bar{v}_i \cdot \frac{\partial \dot{\varphi}_i}{\partial v_j}(v_k)_{k \prec i}, \tag{7}$$

kde

$$\frac{\partial \dot{\varphi}_i}{\partial v_j}(v_k)_{k \prec i} = \sum_{l \prec i} \frac{\partial^2 \varphi_i}{\partial v_j \partial v_l}(v_k)_{k \prec i} \cdot \dot{v}_l.$$

Výsledný soupis operací je na obrázku 5. Po provedení tohoto soupisu operací jsou požadované hodnoty druhých derivací $\bar{y} \cdot f''(x) \cdot \dot{x}$ uloženy v proměnných

$$\dot{\bar{x}} = (\dot{\bar{x}}_1, \ldots, \dot{\bar{x}}_m).$$

3 Automatické derivování v systému UFO

V předcházející kapitole jsme popsali základní formule a algoritmy automatického derivování. V této části výzkumné zprávy ukážeme, jak se tyto postupy v systému UFO používají, vyvolávaji a jak jsou tyto teoretické algoritmy v systému UFO realizovány. Pro názornost budeme tyto principy demonstrovat na jednoduchém příkladu.

$$
\left.\begin{array}{lcl} v_{i-n} & = & x_i \\ \dot{v}_{i-n} & = & \dot{x}_i \end{array}\right\} \quad i = 1, \ldots, n
$$

$$
\left.\begin{array}{lcl} v_i & = & \varphi_i(v_k)_{k \prec i} \\ \dot{v}_i & = & \sum_{j \prec i} \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} \cdot \dot{v}_j \end{array}\right\} \quad i = 1, \ldots l
$$

$$
\left.\begin{array}{lcl} y_{m-i} & = & v_{l-i} \\ \dot{y}_{m-i} & = & \dot{v}_{l-i} \end{array}\right\} \quad i = m-1, \ldots 0
$$

$$
\left.\begin{array}{lcl} \bar{v}_{l-i} & = & \bar{y}_{m-i} \\ \dot{\bar{v}}_{l-i} & = & 0 \end{array}\right\} \quad i = 0, \ldots, m-1
$$

$$
\left.\begin{array}{lcl} \bar{v}_i & = & 0 \\ \dot{\bar{v}}_i & = & 0 \end{array}\right\} \quad i = 1-n, \ldots, l-m
$$

for $j = l, \ldots, 1$ do
  for $j \prec i$
    $\bar{v}_j = \bar{v}_j + \bar{v}_i \cdot \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i}$
    $\dot{\bar{v}}_j = \dot{\bar{v}}_j + \dot{\bar{v}}_i \cdot \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} + \bar{v}_i \cdot \frac{\partial \dot{\varphi}_i}{\partial v_j}(v_k)_{k \prec i}$
  end for
end for

$$
\left.\begin{array}{lcl} \bar{x}_i & = & \bar{v}_{i-n} \\ \dot{\bar{x}}_i & = & \dot{\bar{v}}_{i-n} \end{array}\right\} \quad i = n, \ldots, 1
$$

Obrázek 5: Soupis operací pro výpočet druhých derivací.

3.1 Vyvolání automatického derivování v systému UFO

Pomocí automatického derivování je možné v systému UFO počítat hodnoty prvních nebo druhých derivací funkcí, které jsou zadány makroproměnnými `FMODELF`, `FMODELA` nebo `FMODELC`.

Zadáním jedné z následujících hodnot do proměnné `$IADF` se určuje, zda se má použít automatické derivování pro výpočet hodnoty derivace funkce `FF` v proměnné `FMODELF`:

- `$IADF`=0 (defaultní hodnota)
  Derivace funkce `FF` zadané v `FMODELF` se nepočítají pomocí automatického derivování.

- `$IADF`=1
  První derivace funkce `FF` zadané proměnnou `FMODELF` se počítají pomocí zpětného módu automatického derivování. Je vytvořena proměnná `FGMODELF` s výpočtem hodnoty funkce `FF` a jejího gradientu `GF`. Proměnná `FMODELF` je poté zrušena.

- `$IADF`=2
  Druhé derivace funkce `FF` zadané proměnnou `FMODELF` se počítají pomocí zpětného

a přímého módu automatického derivování. Jsou vytvořeny proměnné FGMODELF (s výpočtem hodnoty funkce FF a jejího gradientu GF – pomocí zpětného módu) a proměnná HMODELF (s výpočtem hodnoty funkce FF, jejího gradientu GF a její Hessovy matice HF – pomocí zpětného módu a následnou aplikací přímého módu). Pak je proměnná FMODELF zrušena.

Požadování transformace a její provedení pro funkce definované proměnnými FMODELA, resp. FMODELC se provádí analogicky jako pro proměnnou FMODELF, liší se však jméno proměnné určující požadovanou transformaci: místo proměnné $IADF se používá $IADA, resp. $IADC.

V rámci jednoho vstupního souboru je možné nezávisle kombinovat hodnoty proměnných $IADF, $IADA a $IADC, například $IADA=2, $IADC=1 atp.

Všechny popsané transformace automatickým derivováním probíhají na úplném začátku zpracování vstupního souboru *.UFO systémem UFO.

3.2 Omezení při použití automatického derivování v systému UFO

Jak již víme z předchozích kapitol, transformace proměnné FMODELF, FMODELA nebo FMODELC pomocí automatického derivování je transformace programu. Pro to, aby implementace automatického derivování v systému UFO nebyla příliš komplikovaná, položili jsme jistá omezení na obsah těchto proměnných FMODELF ad.

Omezení na výpočet hodnot FF (v makroproměnné FMODELF), FA (v makroproměnné FMODELA) a FC (v makroproměnné FMODELC):

- ve výpočtu je zakázáno voláni subroutin a funkcí, kromě standardních funkcí Fortranu. (Toto omezení neplatí, máme-li k dispozici navíc i subroutinu nebo funkci s vypočtem derivace podle konvencí popsaných v dalším odstavci. V tom případě je možné uživatelsky definované subroutiny nebo funkce ve výpočtu použít.)

- pokud se ve výpočtu používá pole (definované například REAL*8 W(100)) a jeho hodnoty jsou závislé na nezávislých proměnných X(), pak je třeba "ručně" deklarovat pole INTEGER IAD_W(100), tj. pole, jehož jméno je složením řetězce IAD_ a jména původního pole. Toto nově definované pole je typu INTEGER a má stejnou velikost jako původní pole. Toto nové pole bude obsahovat indexy do pole, kde jsou zaznamenávány všechny prováděné operace (příp. proměnné).

- ve jménu proměnné nemůže být obsažena číslice

- v příkazu IF a ELSEIF:

  – lze porovnávat pouze čísla a proměnné, žádné výrazy nelze v podmínkách použít,

  – není možné používat mezery (například, není možné použít podmínku IF (A .LT. B) C=D+1 , lze však použít podmínku IF(A.LT.B)C=D+1 ,

  - indexy polí a argumenty funkcí se netransformují, jsou pouze překopírovány. Toto omezení se týká pouze podmínky, nikoliv příkazu prováděného, když je podmínka splněna.

- v příkazu přiřazení:
  - na rozdíl od příkazu `IF` a `ELSEIF`, mezery v příkazu přiřazení mohou být obsaženy,
  - "iterační přiřazení" (například `F=F*X(I)`) s proměnnými, které jsou závislé na nezávislých proměnných `X()`, je nutné přeformulovat za použití cyklu a nově nadefinovaného pole, například `W(I+1)=W(I)*X(I)`.

- nelze použít tzv. "computed goto statement", tj. příkaz
  `GO TO(`*label1, label2, ... labelN*`)` *INTEGER-EXPRESSION*

Jak vidíme, tyto předpoklady na zápis výpočtu funkcí `FF`, `FA` a `FC` v makroproměnných `FMODELF`, `FMODELA` a `FMODELC` nejsou příliš omezující.

3.3 Implementace automatického derivování v systému UFO

3.3.1 První derivace

Jak jsme již naznačili, při výpočtu hodnot derivace pomocí automatického derivování (`$IADF`=1 nebo 2) jsou všechny nezávislé proměnné a prováděné operace (jejich typ, argumenty a numerické hodnoty výsledků) postupně zaznamenávány do polí o velikosti `$NADARR`. Pro `$IADF`=1 nebo 2 se jedná o pole:

- `REAL*8 V($NADARR)` – je pole, ve kterém jsou uloženy hodnoty $v_i = \varphi_i(v_j)_{j \prec i}$, viz vztah (1)

- `REAL*8 VBAR($NADARR)` – je pole, ve kterém jsou uloženy hodnoty $\bar{v}_i = \sum_{j \succ i} \bar{v}_j \cdot \frac{\partial \varphi_j}{\partial v_i}$, viz vztah (5)

- `INTEGER OPCODE($NADARR)` – je pole, ve kterém je uložena identifikace operace prováděné v i-tém kroku (například, sčítání odpovídá 10, násobení odpovídá 30, sinu odpovídá 60 atp.)

- `INTEGER ARG1($NADARR)` – je pole odkazů do těchto polí pro první argument právě prováděné operace

- `INTEGER ARG2($NADARR)` – je pole odkazů do těchto polí pro druhý argument právě prováděné operace

Pro `$IADF`=2 se jedná navíc o pole:

- `REAL*8 VDOT($NADARR)` – je pole, ve kterém jsou uloženy hodnoty $\dot{v}_i = \sum_{j \prec i} \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} \cdot \dot{v}_j$, viz vztah (2)

- `REAL*8 VBARDT($NADARR)` – je pole, ve kterém jsou uloženy hodnoty $\dot{\bar{v}}_i$, viz vztah (7)

Každá elementární funkce $\varphi_i(v_j)_{j\prec i}$ (například násobení, sinus atd.) v postupu výpočtu hodnoty $y = f(x)$ je v průběhu transformace nahrazena subrutinou, která kromě původní operace $\varphi_i(v_j)_{j\prec i}$ provede i zaznamenání svého zavolání do polí `V`, `VDOT`, `OPCODE`, `ARG1`, `ARG2`, případně také `VDOT`, `VBARDT`. Například, elementární funkce násobení nebo sinus jsou nahrazeny subrutinami `BMULTG`, resp. `SING`:

```
!--- ztransformovaná operace násobení ---
INTEGER FUNCTION BMULTG(IARG1, IARG2) !vstupní parametry: pořadí (index)
                                      !numerických hodnot v polích, které
                                      !máme vynásobit
                                      !výstupní hodnota: pořadí (index)
                                      !v polích pro právě tuto operaci
  COMMON /AD_F1/ V, VBAR, OPCODE, ARG1, ARG2, INDARR
    REAL*8 V($NADARR), VBAR($NADARR)
    INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
    INTEGER INDARR
  INTEGER IARG1, IARG2
  V(INDARR)=V(IARG1)*V(IARG2)         !spočtení elementární operace násobení
  VBAR(INDARR)=0.0D0                  !vynulování proměnné v_i s pruhem
                                      !(bude se do ní postupně přičítat)
  OPCODE(INDARR)=30                   !zaznamenání kódu této operace (30=násobení)
  ARG1(INDARR)=IARG1                  !zaznamenání pořadí (indexu) v polích
                                      !pro první (levý) argument
  ARG2(INDARR)=IARG2                  !zaznamenání pořadí (indexu) v polích
                                      !pro druhý (pravý) argument
  BMULTG=INDARR                       !pořadí této operace
  INDARR=INDARR+1                     !příprava na další elementární operaci -
                                      !- posunutí indexu ("ukazovátka") do polí,
                                      !kam až jsou pole vyplněny
END

!--- ztransformovaná operace sinus ---
INTEGER FUNCTION SING(IARG1)
  COMMON /AD_F1/ V, VBAR, OPCODE, ARG1, ARG2, INDARR
    REAL*8 V($NADARR), VBAR($NADARR)
    INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
    INTEGER INDARR
  INTEGER IARG1
  V(INDARR)=SIN(V(IARG1))
  VBAR(INDARR)=0.0D0
  OPCODE(INDARR)=60                   !zaznamenání kódu této operace (60=sinus)
  ARG1(INDARR)=IARG1                  !sinus má pouze jeden argument
  SING=INDARR
  INDARR=INDARR+1
END
```

V průběhu ztransformovaného výpočtu se na všechny nezávislé proměnné a na proměnné, které na nich závisí, [2] odkazujeme pomocí pořadového čísla prováděné operace, ve které vznikly. Toto pořadí odpovídá indexu v polích, kde jsou informace o této operaci uloženy. Jména proměnných jsou pro tento účel transformována – jsou složením řetězce `IAD_` a jména příslušné proměnné, například `IAD_X(*)` pro nezávislou proměnnou `X(*)` nebo `IAD_PROM` pro proměnnou `PROM`. Na straně 12 uvedeme názorný příklad s vysvětlením, jak se operace a hodnoty do polí ukládají.

Jako poslední fáze výpočtu hodnoty derivace je zavolání subrutiny `RVRSWP`, která projde poly `V`, `VDOT`, `OPCODE`, `ARG1`, `ARG2` odzadu dopředu a provede tak vlastní výpočet hodnot (5), tj.

$$\bar{v}_i = \sum_{j \succ i} \bar{v}_j \cdot \frac{\partial \varphi_j}{\partial v_i}$$

neboli

$$\bar{v}_i = \bar{v}_i + \bar{v}_j \cdot \frac{\partial \varphi_j}{\partial v_i}(v_k)_{k \prec j} \quad \text{pro} \quad i \prec j \qquad j = l, \ldots, 1.$$

Procedura `RVRSWP`:

```
SUBROUTINE RVRSWP()
  COMMON /AD_F1/ V, VBAR, OPCODE, ARG1, ARG2, INDARR
    REAL*8 V($NADARR), VBAR($NADARR)
    INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
    INTEGER INDARR
  REAL*8 DERIV
  INTEGER I
  DO 999, I=INDARR-1, 1, -1            !cyklus přes jednotlivé operace pozpátku,
                                       !tj. průchod polemi odzadu dopředu
    IF(OPCODE(I).EQ.30) THEN           !operace násobení
      VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*V(ARG2(I)) !postupné přičítání hodnot
                                                     !do proměnné v_. s pruhem
      VBAR(ARG2(I))=VBAR(ARG2(I))+VBAR(I)*V(ARG1(I)) !postupné přičítání hodnot
                                                     !do proměnné v_. s pruhem
   .
   .
    ELSEIF(OPCODE(I).EQ.60) THEN       !operace sinus
      DERIV=COS(V(ARG1(I)))                          !pomocná proměnná
      VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV      !postupné přičítání hodnot
                                                     !do proměnné v_. s pruhem
   .
   .
    ELSEIF(OPCODE(I).EQ.2) THEN        !operace nezávislá proměnná
      CONTINUE
   .
   .
    ENDIF
999 CONTINUE
    END
```

[2] A také na konstanty.

Po provedení této subrutiny jsou již spočteny hodnoty derivací v příslušných prvcích pole VBAR, odkud je přiřadíme do proměnných GF(*), jak se předpokládá pro výstup z proměnné FGMODELF – například pro X=(X_1,X_2):

```
   DO 86 IADCOUNT=1,2
   GF(IADCOUNT)=VBAR(IAD_X(IADCOUNT))
86 CONTINUE
```

Pro úplnost bychom měli dodat, že na začátku proměnné FGMODELF je zavolán cyklus pro správnou definici a označení nezávislých proměnných X(*) – například pro X=(X_1,X_2):

```
   DO 85 IADCOUNT=1,2
   IAD_X(IADCOUNT)=MKINDP(X(IADCOUNT))
85 CONTINUE
```

kde MKINDP(X(.)) je subrutina, která uloží hodnotu proměnné X(.) do pole a označí ji jako nezávislou proměnnou:

```
! --- operace definování nezávislé proměnné ---
INTEGER FUNCTION MKINDP(CISLO)
  COMMON /AD_F1/ V, VBAR, OPCODE, ARG1, ARG2, INDARR
    REAL*8 V($NADARR), VBAR($NADARR)
    INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
    INTEGER INDARR
  REAL*8 CISLO
  V(INDARR)=CISLO
  VBAR(INDARR)=0.0D0
  OPCODE(INDARR)=2
  MKINDP=INDARR
  INDARR=INDARR+1
END
```

Konstanty, které se ve výpočtu používají, jsou také uloženy do polí pomocí subroutiny MKCNST(hodnota_konstanty)

```
! --- operace definování konstanty ---
INTEGER FUNCTION MKCNST(CISLO)
  COMMON /AD_F1/ V, VBAR, OPCODE, ARG1, ARG2, INDARR
    REAL*8 V($NADARR), VBAR($NADARR)
    INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
    INTEGER INDARR
  REAL*8 CISLO
  V(INDARR)=CISLO
  VBAR(INDARR)=0.0D0
  OPCODE(INDARR)=1
  MKCNST=INDARR
  INDARR=INDARR+1
END
```

3.3.2 Příklad

Na jednoduchém příkladu ukážeme, jak jsou data v polích `V`, `VBAR`, `OPCODE`, `ARG1` a `ARG2` uložena.

Mějme zadanou funkci $f(x_1, x_2) = x_1 \cdot x_2 + 1$. Ve zdrojovém programu, na který budeme aplikovat automatické derivování, se počítá její hodnota výrazem

```
X(1) * X(2) + 1.
```

Tento výraz se transformuje automatickým derivováním na výraz

```
BPLUSG(BMULTG(IAD_X(1),IAD_X(2)),MKCNST(DBLE(1))),
```

kde `IAD_X(1)` a `IAD_X(2)` jsou odkazy (indexy) do polí, kde jsou uloženy nezávislé proměnné `X(1)` a `X(2)` (pomocí procedur `MKCNST(X(1))` a `MKCNST(X(2))`).

V následující tabulce je ukázáno, jak jsou data u polích `V`, `VBAR`, `OPCODE`, `ARG1` a `ARG2` uložena.

pořadí prováděné operace	1	2	3	4	5	...
prováděná operace	definice nezávislé proměnné $X(1)$	definice nezávislé proměnné $X(2)$	$x_1 \cdot x_2$	definice konstanty 1	$x_1 \cdot x_2 + 1$	... ...
subrutina, odkud záznam operace v polích vznikl	MKINDP (X(1))	MKINDP (X(2))	BMULTG (.,.)	MKCNST (DBLE(1))	BPLUSG (.,.)	... ...
index v polích	1	2	3	4	5	...
pole `V`	hodnota x_1	hodnota x_2	hodnota $x_1 \cdot x_2$	hodnota 1	hodnota $x_1 \cdot x_2 + 1$	... ...
pole `VBAR`	0	0	0	0	0	...
pole `OPCODE`	2	2	30	1	10	...
pole `ARG1`	0	0	1	0	3	...
pole `ARG2`	0	0	2	0	4	...

3.3.3 Druhé derivace

Při výpočtu druhých derivací, kdy se aplikuje nejprve zpětný mód a pak přímý mód, jsou výše uvedené subrutiny složitější:

```
!--- ztransformovaná operace násobení (výpočet druhých derivací) ---
INTEGER FUNCTION BMULTH(IARG1, IARG2) ! vstupní parametry: pořadí (index)
                                      !numerických hodnot v polích, které
                                      !máme vynásobit
                                      !výstupní hodnota: pořadí (index)
                                      !v polích pro právě tuto operaci
COMMON /AD_F2/ V, VBAR, VDOT, VBARDT, OPCODE, ARG1, ARG2, INDARR
```

```
 REAL*8 V($NADARR), VBAR($NADARR), VDOT($NADARR),VBARDT($NADARR)
  INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
  INTEGER INDARR
  INTEGER IARG1, IARG2
  V(INDARR)=V(IARG1)*V(IARG2)         !spočtení elementární operace násobení
  VDOT(INDARR)=V(IARG1)*VDOT(IARG2)+VDOT(IARG1)*V(IARG2) !aplikace přímého módu
                                      !na operaci násobení
  VBAR(INDARR)=0.0D0                  !vynulování proměnné v_i s pruhem
                                      !(bude se do ní postupně přičítat)
  VBARDT(INDARR)=0.0D0                !vynulování proměnné v_i s pruhem a tečkou
                                      !(bude se do ní postupně přičítat)
  OPCODE(INDARR)=30                   !zaznamenání kódu této operace (30=násobení)
  ARG1(INDARR)=IARG1                  !zaznamenání pořadí (indexu) v polích
                                      !pro první (levý) argument
  ARG2(INDARR)=IARG2                  !zaznamenání pořadí (indexu) v polích
                                      !pro druhý (pravý) argument
  BMULTH=INDARR                       !pořadí této operace
  INDARR=INDARR+1                     !příprava na další elementární operaci -
                                      !- posunutí indexu ("ukazovátka") do polí,
                                      !kam až jsou pole vyplněny
END

!--- ztransformovaná operace sinus (výpočet druhých derivací) ---
INTEGER FUNCTION SING2(IARG1)
COMMON /AD_F2/ V, VBAR, VDOT, VBARDT, OPCODE, ARG1, ARG2, INDARR
 REAL*8 V($NADARR), VBAR($NADARR), VDOT($NADARR),VBARDT($NADARR)
  INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
  INTEGER INDARR
  INTEGER IARG1
  V(INDARR)=SIN(V(IARG1))
  VDOT(INDARR)=COS(V(IARG1))*VDOT(IARG1)
  VBAR(INDARR)=0.0D0
  VBARDT(INDARR)=0.0D0
  OPCODE(INDARR)=60                   !zaznamenání kódu této operace (60=sinus)
  ARG1(INDARR)=IARG1                  !sinus má pouze jeden argument
  SING2=INDARR
  INDARR=INDARR+1
END
```

Subrutina, která prochází poly nazpátek a počítá vlastní derivace:

```
 SUBROUTINE RVRSWPH()
 COMMON /AD_F2/ V, VBAR, VDOT, VBARDT, OPCODE, ARG1, ARG2, INDARR
  REAL*8 V($NADARR), VBAR($NADARR), VDOT($NADARR),VBARDT($NADARR)
   INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
   INTEGER INDARR
   REAL*8 DERIV
   INTEGER I
   DO 998, I=INDARR-1, 1, -1          !cyklus přes jednotlivé operace pozpátku,
                                      !tj. průchod polemi odzadu dopředu
     IF(OPCODE(I).EQ.30) THEN         !operace násobení
       VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*V(ARG2(I)) !postupné přičítání hodnot
```

```
                                                    !do proměnné  v_. s pruhem
      VBARDT(ARG1(I))=VBARDT(ARG1(I))+              !postupné přičítání hodnot
&                     VBARDT(I)*V(ARG2(I))+         !do proměnné  v_. s pruhem
&                     VBAR(I)*VDOT(ARG2(I))         !a tečkou
      VBAR(ARG2(I))=VBAR(ARG2(I))+VBAR(I)*V(ARG1(I)) !postupné přičítání hodnot
                                                    !do proměnné  v_. s pruhem
      VBARDT(ARG2(I))=VBARDT(ARG2(I))+              !postupné přičítání hodnot
&                     VBARDT(I)*V(ARG1(I))+         !do proměnné  v_. s pruhem
&                     VBAR(I)*VDOT(ARG1(I))         !a tečkou
    .
    .
     ELSEIF(OPCODE(I).EQ.60) THEN     !operace sinus
      DERIV=COS(V(ARG1(I)))                         !pomocná hodnota
      VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV     !postupné přičítání hodnot
                                                    !do proměnné  v_. s pruhem
      VBARDT(ARG1(I))=VBARDT(ARG1(I))+              !postupné přičítání hodnot
&                     VBARDT(I)*DERIV-              !do proměnné  v_. s pruhem
&                     VBAR(I)*SIN(V(ARG1(I)))*VDOT(ARG1(I)) !a tečkou
    .
    .
     ELSEIF(OPCODE(I).EQ.2) THEN      !operace nezávislá proměnná
      CONTINUE
    .
    .
     ENDIF
998 CONTINUE
    END
```

Protože při výpočtu druhých derivací chceme spočíst celou Hessovu matici, musí být uvnitř proměnné `HMODELF` cyklus – například pro `X=(`X_1`,`X_2`)` –, ve kterém se počítají jednotlivé řádky Hessovy matice:

```
   DO 97 IADN=1,2
   ...
   výpočet IADN-tého řádku Hessovy matice
   ...
97 CONTINUE
```

Cyklus pro přiřazení hodnot spočtených derivací – například pro `X=(`X_1`,`X_2`)`:

```
   DO 99 IADCOUNT=1,IADN
     HF(IADN1+IADCOUNT)=VBARDT(IAD_X(IADCOUNT))
99 CONTINUE
   IADN1=IADN1+IADN
```

Cyklus pro přiřazení hodnot nezávislých proměnných do pole na začátku výpočtu proměnné `HMODELF`, kde se navíc určuje, podle kterých proměnných se bude derivovat v přímém módu:

```
   DO 98 IADCOUNT=1,2
     IF(IADCOUNT.EQ.IADN) THEN
       IAD_X(IADCOUNT)=MKINDPH(X(IADCOUNT),1.0D0)
                          ELSE
       IAD_X(IADCOUNT)=MKINDPH(X(IADCOUNT),0.0D0)
     ENDIF
98 CONTINUE
```

Definice nezávislé proměnné:

```
! --- operace definování nezávislé proměnné (výpočet druhých derivací) ---
INTEGER FUNCTION MKINDPH(CISLO1,CISLO2)
COMMON /AD_F2/ V, VBAR, VDOT, VBARDT, OPCODE, ARG1, ARG2, INDARR
 REAL*8 V($NADARR), VBAR($NADARR), VDOT($NADARR),VBARDT($NADARR)
  INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
  INTEGER INDARR
  REAL*8 CISLO1, CISLO2
  V(INDARR)=CISLO1
  VBAR(INDARR)=0.0D0
  VDOT(INDARR)=CISLO2
  VBARDT(INDARR)=0.0D0
  OPCODE(INDARR)=2
  MKINDPH=INDARR
  INDARR=INDARR+1
END
```

Definice konstanty:

```
! --- operace definování konstanty (výpočet druhých derivací) ---
INTEGER FUNCTION MKCNSTH(CISLO)
COMMON /AD_F2/ V, VBAR, VDOT, VBARDT, OPCODE, ARG1, ARG2, INDARR
 REAL*8 V($NADARR), VBAR($NADARR), VDOT($NADARR),VBARDT($NADARR)
  INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
  INTEGER INDARR
  REAL*8 CISLO
  V(INDARR)=CISLO
  VBAR(INDARR)=0.0D0
  VDOT(INDARR)=0.0D0
  VBARDT(INDARR)=0.0D0
  OPCODE(INDARR)=1
  MKCNSTH=INDARR
  INDARR=INDARR+1
END
```

Shrněme pro přehlednost všechny změny, které při transformaci automatickým derivováním nastávaji:

- transformace výpočtu (tak, aby se kromě spočtení elementárních operací také zaznamenalo jejich pořadí, parametry a výsledné hodnoty) a s tím související přejmenování proměnných (nezávislých proměnných a všech proměnných, které na nich závisí),

- zavolání procedury `RVRSWP` pro `$IADF`=1 (resp. `RVRSWPH` pro `$IADF`=2), která projde seznamem prováděných operací pozpátku a provede vlastní výpočet derivací,
- definice procedur `BPLUSG, BSING, RVRSWP` atd., sloužící pro výpočet a zaznamenání elementárních operací atd. a jejich slinkování s původním programem,

3.4 Příklad

Použití automatického derivování v systému UFO a jeho výhody budeme demonstrovat na následujícím příkladu. Nechť $x \in \mathbb{R}^N$ a nechť funkce $F : \mathbb{R}^N \to \mathbb{R}$ je definována jako

$$F(x) = \sum_{i=1}^{N} (N + i - P_i)^2 , \tag{8}$$

kde

$$P_i = \sum_{j=1}^{N} \left(5\,(1 + mod(i,5) + mod(j,5)) \sin(x_j) + \frac{i+j}{10} \cos(x_j) \right),$$

kde $mod(a,b)$ je zbytek po vydělení "a děleno b". Hledáme lokální minimum funkce F s počáteční iterací

$$x_0 = \left(1, \frac{1}{2}, \ldots, \frac{1}{N}\right).$$

Vstupní soubor pro systém UFO `P.UFO` je na obrázku 6.

Při výpočtu chceme použít automatické derivování pro funkce definované makroproměnnými `FMODELA`, proto přiřadíme `$IADA=1` (řádek 21). Makroproměnná `$NF` (řádek 18) určuje počet nezávislých proměnných `X(*)`, makroproměnné `MODEL` (řádek 17) a `$NA` (řádek 19) specifikují typ optimalizované úlohy (viz [5]). Výpočet hodnoty

$$(N + i - P_i)^2 \tag{9}$$

(viz vztah 8) se definuje proměnnou `FA` v makroproměnné `FMODELA` (řádky 8 až 16). Hodnoty počáteční iterace x_0 se zadávají makroproměnnou `INPUT` (řádky 3 až 7).

Protože makroproměnná `$IADA` je nastavena na hodnotu 1 (řádek 21), provede se na začátku zpracování vstupního souboru `P.UFO` vymazání makroproměnné `FMODELA` a vytvoření makroproměnné `FGMODELA`, která je zobrazena na obrázku 7. Zde jsou na řádcích 2 až 4 označeny proměnné `X(.)` jako nezávislé proměnné. Na řádcích 9 až 10 je ztransformovaný příkaz přiřazení `W(I+1)=W(I)+A*SIN(X(I))+B*COS(X(I))` z řádku 13 (na obrázku 6) a na řádcích 12 a 13 (na obrázku 7) je ztransformovaný řádek 15 (na obrázku 6), totiž `FA=(DBLE($$NF+KA)-W($$NF+1))**2`. Z těchto ztransformovaných řádků jsou volány subroutiny, které provádí nejen původní elementární aritmetické operace, ale také zaznamenání této operace do pole (viz kapitola 2.3). Parametry těchto

```
 ! deklarace pomocných promměnných:
INTEGER IAD_W($$NF+1)                                          !01
REAL*8 W($$NF+1)                                               !02
 ! počáteční iterace:
$SET(INPUT)                                                    !03
      DO 80 I=1, $$NF                                          !04
      X(I)=1.0D0/I                                             !05
   80 CONTINUE                                                 !06
$ENDSET                                                        !07
 ! složky sumy optimalizované funkce:
$SET(FMODELA)                                                  !08
      W(1)=0.0D0                                               !09
      DO 81 I=1, $$NF                                          !10
        A=5.0D 0*(1.0D 0+MOD(I,5)+MOD(KA,5))                   !11
        B=DBLE(I+KA)/1.0D1                                     !12
        W(I+1)=W(I)+A*SIN(X(I))+B*COS(X(I))                    !13
   81 CONTINUE                                                 !14
      FA=(DBLE($$NF+KA)-W($$NF+1))**2                          !15
$ENDSET                                                        !16
 ! typ optimalizované úlohy:
$MODEL='AF'                                                    !17
 ! počet nezávislých proměnných:
$NF=50                                                         !18
$NA=50                                                         !19
$NOUT=1                                                        !20
 ! automatické derivování pro FMODELFA:
$IADA=1                                                        !21
$BATCH                                                         !22
$STANDARD                                                      !23
```

Obrázek 6: Vstupní soubor `P.UFO` pro příklad – automatické derivování v systému UFO.

subroutin jsou odkazy do polí (indexy polí) – jméno proměnné je vždy spojení řetězce `IAD_` a jména původní proměnné. [3]

Na řádku 14 je přiřazena spočtená hodnota `FA`, tj. hodnota

$$(N + i - P_i)^2 .$$

Na řádku 15 se provádí

$$\bar{v}_l = 1.$$

Průchod poly nazpět se zaznamenanými operacemi se vyvolá z řádku 16 subroutinou `RVRSWP()`. Po jejím provedení jsou spočtené hodnoty derivací přiřazeny do proměnných `GA` (řádky 17 až 19).

[3] Tyto proměnné-odkazy do polí není třeba zvlášť deklarovat, s výjimkou případu, kdy původní proměnná je pole, jak je popsáno v kapitole 2.3. Proto na řádku 1 na obrázku 6 deklarujeme pole `INTEGER IAD_W($$NF+1)`.

```
   INDARR=1                                                          !01
    ! označení nezávislých proměnných:
   DO 85 IADCOUNT=1,50                                               !02
   IAD_X(IADCOUNT)=MKINDP(X(IADCOUNT))                               !03
85 CONTINUE                                                          !04
    ! ztransformovaný výpočet FA:
   IAD_W(1)=MKCNST(DBLE(0.0D0))                                      !05
   DO 81 I=1, 50                                                     !06
   A=5.0D 0*(1.0D 0+MOD(I,5)+MOD(KA,5))                              !07
   B=DBLE(I+KA)/1.0D1                                                !08
   IAD_W(I+1)=BPLUSG(BPLUSG(IAD_W(I),BMULTG(MKCNST(DBLE(A)),SING(IAD_  !09
  &    X(I)))),BMULTG(MKCNST(DBLE(B)),COSG(IAD_X(I))))              !10
81 CONTINUE                                                          !11
   IAD_FA=BEXPG(BMINUG(MKCNST(DBLE(DBLE(50+KA))),IAD_W(50+1)),MKCNST(  !12
  &    DBLE(2)))                                                    !13
    ! vypočtená hodnota FA:
   FA=V(IAD_FA)                                                      !14
    ! přiřazení váhy:
   VBAR(IAD_FA)=1.0D0                                                !15
    ! zavolání zpětného průchodu polem:
   CALL RVRSWP()                                                     !16
    ! přiřazení vypočtených hodnot derivací:
   DO 86 IADCOUNT=1,50                                               !17
   GA(IADCOUNT)=VBAR(IAD_X(IADCOUNT))                                !18
86 CONTINUE                                                          !19
```

Obrázek 7: Hodnota proměnné `FGMODELF` pro příklad – automatické derivování v systému UFO

Řádky 6 až 9 (na obrázku 7) jsou ponechány beze změny, protože nejsou závislé na nezávislých proměnných `X(.)`. Na řádku 1 (na obrázku 6) je počáteční nastavení indexu do polí na ukládání provedených operací.

Pro porovnání doby výpočtu jsme tuto úlohu provedli také s výpočtem derivací pomocí poměrných diferencí (ve vstupním souboru `P.UFO` jsme přiřazení `$IADA=1` nahradili přiřazením `$IADA=0`). Doby trvání výpočtů jsou uvedeny v tabulce 1. Při výpočtu pomocí automatického derivování spočtené hodnoty konvergují k řešení rychleji a výpočet trvá kratší dobu. □

Počet proměnných	automatické derivování	poměrné diference
$N = 10$	0.11 s	0.16 s
$N = 20$	0.49 s	0.94 s
$N = 50$	1.37 s	6.97 s
$N = 100$	3.36 s	44.93 s

Tabulka 1: Porovnání doby trvání výpočtu – Příklad UFO.

4 Modifikace BELu pro použití automatického derivování v UFU

Z předchozích kapitol této výzkumné zprávy již víme, jaké jsou základní postupy automatického derivování a jak se automatické derivování používá v programech vytvořených systémem UFO.

V této části podrobně uvedeme, jak byl modifikován systém BEL a systém šablon, aby systém UFO *vytvářel* programy používající automatické derivování ve tvaru popsaném ve druhé kapitole.

4.1 Základní principy implementace automatického derivování v systému UFO

V tomto odstavci popíšeme základní principy implementace automatického derivování pro proměnnou `FMODELF`. Pro proměnné `FMODELA` a `FMODELC` je postup implementace úplně analogický.

Jak již víme z odstavce 3.1, při zadání hodnoty 1 nebo 2 do proměnné `$IADF` dojde na začátku zpracování vstupního souboru systémem UFO k vytvoření proměnné `$FGMODELF` (a pro `$IADF`=2 také k vytvoření proměnné `$HMODELF`) a smazání proměnné `$FMODELF`.

Do proměnné `FGMODELF` se přitom uloží postup výpočtu hodnoty funkce `FF` a jejího gradientu `GF` pomocí zpětného módu automatického derivování. Používáme zpětný mód, protože v proměnné `FMODELF` se počítá skalární hodnota `FF` a pro výpočet jejího gradientu je zpětný mód výhodnější.

Proměnnou `FGMODELF` získáme z proměnné `FMODELF`. Zhruba řečeno, všechny výpočty a přiřazení ztransformujeme, aby se kromě provedení tohoto výpočtu a přiřazení zaznamenala provedená operace společně s argumenty do polí, kterými ve druhé části výpočtu projdeme odzadu a vykonáme patřičné operace pro výpočet derivace. Přednastavené délky těchto polí jsou `$NADARR=1000`. Je-li potřeba tato hodnota zvětšit, pak se do vstupního souboru vloží řádek, na kterém přiřadíme potřebnou hodnotu, například `$NADARR=2000`. Každá elementární operace přitom obsadí v každém z těchto polí jeden prvek.

Pro druhé derivace a proměnnou `HMODELF` platí výše uvedená fakta analogicky jako pro proměnnou `FGMODELF`, ale výpočet derivace probíhá pomocí aplikace nejprve zpětného módu a následně přímého módu.

Zároveň je potřeba k výslednému programu `P.FOR` připojit deklarace použitých procedur a funkcí (pro zaznamenání zavolání operace a pro průchod seznamem operací nazpět atd.) a deklarace příslušných datových typů (pole atd.).

4.2 Implementace automatického derivování v systému UFO

V tomto odstavci popíšeme, jak jsou do výsledného programu `P.FOR` připojeny deklarace použitých podprogramů a datových typů pro automatické derivování. Dále uvedeme,

jakým způsobem je změněn vlastní obsah proměnné FMODELF na FGMODELF, resp. na HMODELF.

4.2.1 Šablony UZDECL.I, UZADFD1.I a UZADFD2.I

Šablona UZDECL.I slouží k deklaraci proměnných, používaných ve výsledném programu P.UFO. Proto jsme na její konec přidali část kódu, kterým se deklarují proměnné používané při automatickém derivování. Část šablony UZDECL.I, příslušná automatickému derivování, je tedy:

```
$REM --- BEGIN AD ---
$IF(IADF=2)              !jestliže počítáme nějaké DRUHÉ derivace automaticky,
  $SUBST('UZADFD2')      ! deklarujeme proměnné ze šablony UZADFD2.I
$ELSEIF(IADA=2)          !jestliže počítáme nějaké DRUHÉ derivace automaticky,
  $SUBST('UZADFD2')      ! deklarujeme proměnné ze šablony UZADFD2.I
$ELSEIF(IADC=2)          !jestliže počítáme nějaké DRUHÉ derivace automaticky,
  $SUBST('UZADFD2')      ! deklarujeme proměnné ze šablony UZADFD2.I
$ELSEIF(IADF=1)          !jestliže počítáme nějaké PRVNÍ derivace automaticky,
  $SUBST('UZADFD1')      ! deklarujeme proměnné ze šablony UZADFD1.I
$ELSEIF(IADA=1)          !jestliže počítáme nějaké PRVNÍ derivace automaticky,
  $SUBST('UZADFD1')      ! deklarujeme proměnné ze šablony UZADFD1.I
$ELSEIF(IADC=1)          !jestliže počítáme nějaké PRVNÍ derivace automaticky,
  $SUBST('UZADFD1')      ! deklarujeme proměnné ze šablony UZADFD1.I
$ELSE
$ENDIF
$REM --- END AD ---
```

Pokud je tedy některá z proměnných IADA, IADC nebo IADF rovna 2 (to znamená, že některé druhé derivace se budou počítat automaticky), do programu P.FOR se nadeklarují proměnné uvedené v šabloně UZADFD2.I:

```
! jména procedur-elementárních funkcí, které zaznamenávají své zavolání
INTEGER BPLUSG, BMINUG, BMULTG, BDIVG, EXPG, LOGG, COSG, SING
INTEGER SQRTG, ABSG, MKINDP, MKCNST, BEXPG, LOG10G
INTEGER TANG, SINHG, COSHG, TANHG, ASING, ACOSG, ATANG
! pole na zaznamenávání elementárních operací
COMMON /AD_F2/ V, VBAR, VDOT, VBARDT, OPCODE, ARG1, ARG2, INDARR
 REAL*8 V($NADARR), VBAR($NADARR), VDOT($NADARR),VBARDT($NADARR)
 INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
 INTEGER INDARR
! jména procedur-elementárních funkcí, které zaznamenávají své zavolání
INTEGER BPLUSH, BMINUH, BMULTH, BDIVH, COSG2, SING2, EXPG2, LOGG2
INTEGER SQRTG2, ABSG2, MKINDPH, MKCNSTH, BEXPG2, LOG10H
INTEGER TANG2, SINHG2, COSHG2, TANHG2, ASING2, ACOSG2, ATANG2
REAL*8 SGN
! nezávislé a závislé proměnné
INTEGER IAD_X($NF), IAD_Y(1)
```

A pokud je tedy některá z proměnných IADA, IADC nebo IADF rovna 1 (to znamená, že některé první derivace se budou počítat automaticky), a zároveň žádná z proměnných IADA, IADC nebo IADF není rovna 2, do programu P.FOR se nadeklarují proměnné uvedené v šabloně UZADFD1.I:

```
! pole na zaznamenávání elementárních operací
COMMON /AD_F1/ V, VBAR, OPCODE, ARG1, ARG2, INDARR
 REAL*8 V($NADARR), VBAR($NADARR)
 INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
 INTEGER INDARR
! jména procedur-elementárních funkcí, které zaznamenávají své zavolání
INTEGER BPLUSG, BMINUG, BMULTG, BDIVG, EXPG, LOGG, COSG, SING
INTEGER SQRTG, ABSG, MKINDP, MKCNST, BEXPG, LOG10G
INTEGER TANG, SINHG, COSHG, TANHG, ASING, ACOSG, ATANG
REAL*8 SGN
! nezávislé a závislé proměnné
INTEGER IAD_X($NF), IAD_Y(1)
```

Makroproměnné $NF a $NADARR budou při substituci nahrazeny hodnotami:

- $NF je počet nezávislých proměnných optimalizované úlohy,

- $NADARR je délka polí, do kterých se zaznamenávají jednotlivé elementární operace. Tato hodnota je definovana v šabloně UZAD1.I, resp. UZAD2.I, o kterých se zmíníme v následujícím odstavci.

Všimněme si, že šabloha UZADFD2.I obsahuje přesně to, co šablona UZADFD1.I, ale ještě něco málo navíc. Díky tomu, pokud se požadují počítat automaticky derivace prvního i druhého řádu, stačí použít deklaraci proměnných v šabloně UZADFD2.I.

4.2.2 Šablony UZINIT.I a UZAD1.I, UZAD2.I, UZADF1.I, UZADF2.I a další

Šablona UZINIT.I se používá pro inicializaci numerických metod. V případě automatického derivování zde probíhají následující kroky:

- Přiřazení $IADF=0 (resp. $IADA=0 nebo $IADC=0), pokud v souboru P.UFO není definovaná makroproměnná $IADF (resp. $IADA nebo $IADC).

- Přiřazení $NADARR=1000, pokud již makroproměnná $NADARR není v souboru P.UFO definovaná.

- Deklarace procedur, které se používají při volání automatického derivování. Tyto procedury jsou vloženy do makroproměnné SUBROUTINES.

- Vlastní transformace makroproměnné FMODELF (resp. FMODELA nebo FMODELC) na makroproměnnou FGMODELF (resp. FGMODELA nebo FGMODELC) a případně také HMODELF (resp. HMODELA nebo HMODELC), aby tyto upravené makroproměnné počítaly derivace automatickým derivováním.
- Vymazání makroproměnné FMODELF (resp. FMODELA nebo FMODELC).

O těchto jednotlivých krocích se nyní zmíníme podobněji.

4.2.3 Přiřazení do makroproměnných $IADF, $IADA a $IADC

Pokud ve vstupním souboru P.UFO není definovaná hodnota proměnné $IADF, resp. $IADC, resp. $IADA, pak je hodnota těchto proměnných definována v šabloně UZINIT.I takto:

```
$REM --- NEJAKE AD?
$IF($DEF(IADF))   ! je definována hodnota proměnné IADF?
$ELSE             ! pokud ano, pak neděláme nic
  $IADF=0         ! pokud ne, pak do ní dosadíme nulu,
$ENDIF            !  tj. derivovat se nebude automaticky
$IF($DEF(IADA))   ! je definována hodnota proměnné IADA?
$ELSE             ! pokud ano, pak neděláme nic
  $IADA=0         ! pokud ne, pak do ní dosadíme nulu,
$ENDIF            !  tj. derivovat se nebude automaticky
$IF($DEF(IADC))   ! je definována hodnota proměnné IADC?
$ELSE             ! pokud ano, pak neděláme nic
  $IADC=0         ! pokud ne, pak do ní dosadíme nulu,
$ENDIF            !  tj. derivovat se nebude automaticky
```

4.2.4 Přiřazení do makroproměnné $NADARR

Dále se ze šablony UZINIT.I vyvolá zpracování šablony UZAD1.I, resp. UZAD2.I, a to podle hodnot proměnných $IADF, resp. $IADC, resp. $IADA:

```
$IF(IADF=2)         ! pokud IADF=2,
 $SUBST('UZAD2')    !  pak vyvolej zpracovani sablony UZAD2.I
$ELSEIF(IADA=2)     ! jinak: pokud IADA=2,
 $SUBST('UZAD2')    !  pak vyvolej zpracovani sablony UZAD2.I
$ELSEIF(IADC=2)     ! jinak: pokud IADC=2,
 $SUBST('UZAD2')    !  pak vyvolej zpracovani sablony UZAD2.I
$ELSEIF(IADF=1)     ! jinak: pokud IADF=1,
 $SUBST('UZAD1')    !  vyvolej zpracovani sablony UZAD1.I
$ELSEIF(IADA=1)     ! jinak: pokud IADA=1,
 $SUBST('UZAD1')    !  vyvolej zpracovani sablony UZAD1.I
$ELSEIF(IADC=1)     ! jinak: pokud IADC=1,
 $SUBST('UZAD1')    !  vyvolej zpracovani sablony UZAD1.I
```

```
$ELSE
$ENDIF
```

V šablonách UZAD1.I, resp. UZAD2.I se přiřadí hodnota do proměnné $NADARR, pokud ještě není definována:

```
$REM --- DELKA POLE PRO UCHOVAVANI OPERACI
$IF($DEF(NADARR))        ! pokud neni definovana promenna $NADARR,
$ELSE
  $NADARR=1000           ! pak prirad implicitni hodnotu 1000
$ENDIF
```

Hodnota $NADARR je délka pole, ve kterém se zaznamenávají provedené elementární operace.

4.2.5 Vlastní transformace makroproměnné FMODELF na FGMODELF a případně HMODELF (Nové OPTION parametry pro BEL: ADF, ADA, ADC)

Dále je ze šablony UZINIT.I zavoláno zpracování šablony UZADF1.I, resp. UZADF2.I takto:

```
$REM --- AD PRO FMODELF
$IF(IADF=1)             ! má-li se odvodit 1.derivace z FMODELF,
  $SUBST('UZADF1')      ! pak zpracuj šablonu UZADF1.I
$ENDIF
$IF(IADF=2)             ! má-li se odvodit 2.derivace z FMODELF,
  $SUBST('UZADF2')      ! pak zpracuj šablonu UZADF2.I
$ENDIF

$REM --- AD PRO FMODELA
$IF(IADA=1)             ! má-li se odvodit 1.derivace z FMODELA,
  $SUBST('UZADA1')      ! pak zpracuj šablonu UZADA1.I
$ENDIF
$IF(IADA=2)             ! má-li se odvodit 2.derivace z FMODELA,
  $SUBST('UZADA2')      ! pak zpracuj šablonu UZADA2.I
$ENDIF

$REM --- AD PRO FMODELC
$IF(IADC=1)             ! má-li se odvodit 1.derivace z FMODELC,
  $SUBST('UZADC1')      ! pak zpracuj šablonu UZADC1.I
$ENDIF
$IF(IADC=2)             ! má-li se odvodit 2.derivace z FMODELC,
  $SUBST('UZADC2')      ! pak zpracuj šablonu UZADC2.I
$ENDIF
$REM --- END AD ---
```

V těchto šablonách UZADF1.I, resp. UZADF2.I, probíhá vlastní transformace makroproměnné FMODELF na FGMODELF, a případně HMODELF.

Šablona UZADF1.I obsahuje:

```
$REM --- TOHLE UVNITR (BEL) PRETRANSFORMUJE VYTVORI FGMODELF POMOCI AD
$OPTION(ADF=1)
```

A šablona UZADF2.I obsahuje:

```
$REM --- TOHLE UVNITR (BEL) PRETRANSFORMUJE VYTVORI HMODELF POMOCI AD
$OPTION(ADF=2)
```

Při implementaci automatického derivování do systému UFO jsme tedy zavedli nové parametry pro preprocesor BEL, tzv. OPTION-parametry. Tyto nové parametry se jmenují ADF, ADA, ADC. Při nastavování hodnoty těchto parametrů $OPTION(ADF=1), resp. $OPTION(ADF=2) dojde k vlastní transformaci proměnné FMODELF na FGMODELF a případně HMODELF.

Kvůli zavedení těchto nových OPTION-parametrů a uchování jejich hodnot, jsme změnili všechny subroutiny, které obsahují datovou strukturu

```
      COMMON/UQINT2/ LENB,LENST,LP,ILNLEN,OLNLEN,MODERW,DIALOG,IUNIT,
     *        OUNIT,INUNIT,IIUNIT,OIUNIT,NOPT,LENOPT,NSUB,MNSUB,
     *        NDIR,LENDIR,NMSG,MNIF,NIF1,NIF2,TABCH,ADF,ADA,ADC
```

kam jsme přidali proměnné ADF,ADA,ADC typu INTEGER. Jedná se o subroutiny: Uqcode, Uqichr, Uqnext, Uqlogm, Uqlexp, Uqceol, Uqsetv, Uqsifd, Uqstri, Uqsubs, Uquchr, Uqaddv a také Uqabel, Uqgval, Uqopti.

Aby se přiřazení $OPTION(ADF=1), resp. $OPTION(ADF=2) správně vyhodnotilo, změnili jsme i subroutinu Uqgval.for, která z příkazu $OPTION(ADF=n) naplní proměnnou ADF datového bloku UQINT2.

Dále jsme modifikovali subroutinu Uqabel, která obsahuje definice všech parametrů typu OPTION. V této subroutině jsme také zavedli nový typ chyby, a to

```
     * '48 AUTOMATICAL DIFFERENTIATION ERROR'/
```

Touto chybou skončí běh BELu, pokud dojde k nějaké chybě v průběhu automatického derivování.

Další změněná subroutina je Uqopti, která na základě hodnoty v OPTION-proměnné ADF, (ADA nebo ADC) provede zavolání vlastní transformace proměnné FMODELF na FGMODELF, a případně HMODELF, a to subroutinu UQADF1, nebo UQADF2 a UQADF3:

```
   39 ADF=IV
      IF (ADF.EQ.1) THEN                          ! vlastni transformace
      CALL UQADF1(SYMTAB,MLENST,IPOINT,MNP,IER)  ! FMODELF -> FGMODELF
      ENDIF
      IF (ADF.EQ.2) THEN                          ! vlastni transformace
      CALL UQADF2(SYMTAB,MLENST,IPOINT,MNP,IER)  ! FMODELF -> FGMODELF
      CALL UQADF3(SYMTAB,MLENST,IPOINT,MNP,IER)  ! FMODELF -> HMODELF
      ENDIF
      RETURN
```

Subroutina UQADF1, resp. UQADF2 a UQADF3, se skládá ze dvou kroků:

- **První průchod proměnnou** FMODELF
  V tomto kroku se zjistí, které proměnné jsou závislé na nezávislých proměnných a tedy které proměnné je potřeba transformovat.

- **Druhý průchod proměnnou** FMODELF
  V tomto kroku se provede vlastní transformace proměnné FMODELF na FGMODELF, resp. HMODELF.

O těchto krocích se nyní zmíníme podrobněji.

První průchod FMODELF Abychom věděli, které proměnné se přejmenují a které elementární operace se budou kvůli automatickému derivování zaznamenávat do polí, je potřeba nejdříve projít celý postup výpočtu v proměnné FMODELF, a při tomto průchodu najít ty proměnné, které závisí na nezávislých proměnných, tj. X(*). Takto nalezené proměnné se přejmenují, například z Y (například typu REAL) na AD_Y (typu INTEGER, jako index do polí s uchovanými elementárními operacemi). Seznam nalezených proměnných, které závisí na nezávislých proměnných, se uloží do proměnné ADVARIA.

Druhý průchod FMODELF Zde proběhne řádek po řádku, vlastní transformace proměnné FMODELF na FGMODELF, resp. HMODELF. Výsledný tvar proměnné FGMODELF, resp. HMODELF, je popsán v kapitole 3.4.

Hodnota proměnné FMODELF je v subroutine UQADF1.FOR uložena v proměnné FORMF (typu CHARACTER*5000) a hodnota proměnné FGMODELF je uložena v proměnné ADFORMF (typu CHARACTER*10000).

Dále následuje cyklus přes jednotlivé řádky proměnné FMODELF a každý řádek je ztransformován – například přiřazení, cyklus, podmínka, ...

4.2.6 Vymazání makroproměnné FMODELF

Závěrem je ze šablon UZADF1.I, resp. UZADF2.I zavoláno vymazání proměnné FMODELF. Obě šablony tedy UZADF1.I i UZADF2.I obsahují:

```
$REM --- A VYMAZEME FMODELF
$ERASE(FMODELF)      ! vymaž hodnotu proměnné FMODELF
```

Po provedení šablon UZINIT.I a UZDECL.I je tedy proměnná FMODELF nahrazena proměnnou FGMODELF, resp. HMODELF. Schématicky lze tento postup znázornit takto:

Příloha

A Deklarace procedur používaných v automatickém derivování

Dále jsou v šablonách UZAD1.I, resp. UZAD2.I uvedeny deklarace procedur, které se používají pri automatickém derivování. Šablona UZAD1.I obsahuje procedury pro výpočet první derivace, šablona UZAD2.I obsahuje procedury pro výpočet druhé derivace.

Deklarace těchto procedur se přiřadí do makroproměnné SUBROUTINES, jejíž obsah je při běhu BELu připojen k výslednému souboru P.FOR, řešící zadaný optimalizační problém.

Šablona UZAD1.I obsahuje procedury pro výpočet první derivace, jako BMULTG, SING a RVRSWP, jak jsme je popsali v kapitole 2.3.

```
$ADD(SUBROUTINES)
      INTEGER FUNCTION BMULTG(IARG1, IARG2)
        COMMON /AD_F1/ V, VBAR, OPCODE, ARG1, ARG2, INDARR
          REAL*8 V($NADARR), VBAR($NADARR)
          INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
          INTEGER INDARR
        INTEGER IARG1, IARG2
        V(INDARR)=V(IARG1)*V(IARG2)
        VBAR(INDARR)=0.0$P 0
        OPCODE(INDARR)=30
        ARG1(INDARR)=IARG1
        ARG2(INDARR)=IARG2
        BMULTG=INDARR
        INDARR=INDARR+1
      END

...

      INTEGER FUNCTION SING(IARG1)
        COMMON /AD_F1/ V, VBAR, OPCODE, ARG1, ARG2, INDARR
          REAL*8 V($NADARR), VBAR($NADARR)
          INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
          INTEGER INDARR
        INTEGER IARG1
        V(INDARR)=SIN(V(IARG1))
        VBAR(INDARR)=0.0$P 0
        OPCODE(INDARR)=60
        ARG1(INDARR)=IARG1
        SING=INDARR
        INDARR=INDARR+1
      END
...
```

```
 SUBROUTINE RVRSWP()
   COMMON /AD_F1/ V, VBAR, OPCODE, ARG1, ARG2, INDARR
     REAL*8 V($NADARR), VBAR($NADARR)
     INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
     INTEGER INDARR
   REAL*8 DERIV, SGN
   INTEGER I, NINTV
   DO 999, I=INDARR-1, 1, -1
     IF(OPCODE(I).EQ.10) THEN
       VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)
       VBAR(ARG2(I))=VBAR(ARG2(I))+VBAR(I)
     ELSEIF(OPCODE(I).EQ.20) THEN
       VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)
       VBAR(ARG2(I))=VBAR(ARG2(I))-VBAR(I)
     ELSEIF(OPCODE(I).EQ.30) THEN
       VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*V(ARG2(I))
       VBAR(ARG2(I))=VBAR(ARG2(I))+VBAR(I)*V(ARG1(I))
     ELSEIF(OPCODE(I).EQ.40) THEN
       VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)/V(ARG2(I))
       VBAR(ARG2(I))=VBAR(ARG2(I))-VBAR(I)*V(I)/V(ARG2(I))
     ELSEIF(OPCODE(I).EQ.50) THEN
       VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*V(I)
     ELSEIF(OPCODE(I).EQ.51) THEN
       VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)/V(ARG1(I))
     ELSEIF(OPCODE(I).EQ.52) THEN
       IF (V(ARG1(I)).LE.0) THEN
         NINTV=NINT(V(ARG2(I)))
         IF (ABS(NINTV-V(ARG2(I)))/MAX(1,ABS(NINTV)).LE.1.0$P-14)
&                 THEN
           VBAR(ARG1(I))=VBAR(ARG1(I))+
&               VBAR(I)*NINTV*V(ARG1(I))**(NINTV-1)
         ELSE
           WRITE(*,*) '--- RVRSWP: POWER ARG ERROR! ---'
         ENDIF
       ELSE
         VBAR(ARG1(I))=VBAR(ARG1(I))+
&             VBAR(I)*V(ARG2(I))*V(ARG1(I))**(V(ARG2(I))-1.0$P 0)
       ENDIF
       IF(V(ARG1(I)).NE.0) THEN
         VBAR(ARG2(I))=VBAR(ARG2(I))+VBAR(I)*V(I)*
&                       LOG(ABS(V(ARG1(I))))
       ELSE
         WRITE(*,*) '--- RVRSWP: LOG ARG ERROR! ---'
       ENDIF
     ELSEIF(OPCODE(I).EQ.53) THEN
```

```
          NINTV=NINT(V(ARG2(I)))
          VBAR(ARG1(I))=VBAR(ARG1(I))+
     &               VBAR(I)*NINTV*V(ARG1(I))**(NINTV-1)
        ELSEIF(OPCODE(I).EQ.54) THEN
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)/V(ARG1(I))/LOG(10D0)
        ELSEIF(OPCODE(I).EQ.60) THEN
          DERIV=COS(V(ARG1(I)))
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.61) THEN
          DERIV=-SIN(V(ARG1(I)))
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.62) THEN
          DERIV=1/(COS(V(ARG1(I))))**2
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.64) THEN
          DERIV=COSH(V(ARG1(I)))
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.65) THEN
          DERIV=SINH(V(ARG1(I)))
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.66) THEN
          DERIV=1/(COSH(V(ARG1(I))))**2
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.70) THEN
          DERIV=1/SQRT(1-V(ARG1(I))**2)
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.71) THEN
          DERIV=-1/SQRT(1-V(ARG1(I))**2)
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.72) THEN
          DERIV=1/(1+V(ARG1(I))**2)
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.80) THEN
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*SGN(V(ARG1(I)))
        ELSEIF(OPCODE(I).EQ.90) THEN
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)/(2.0$P 0*V(I))
        ELSEIF(OPCODE(I).EQ.1) THEN
          CONTINUE
        ELSEIF(OPCODE(I).EQ.2) THEN
       CONTINUE
        ENDIF
 999    CONTINUE
      END
$ENDADD
```

Šablona `UZAD2.I` obsahuje procedury pro výpočet první derivace, jako BMULTG,

SING, RVRSWP a zároveň procedury pro výpočet druhé derivace, jako BMULTH, SING2, RVRSWPH, jak jsme je popsali v kapitole 2.3,

```
$ADD(SUBROUTINES)
      INTEGER FUNCTION BMULTG(IARG1, IARG2)
      COMMON /AD_F2/ V, VBAR, VDOT, VBARDT, OPCODE, ARG1, ARG2, INDARR
       REAL*8 V($NADARR), VBAR($NADARR), VDOT($NADARR),VBARDT($NADARR)
       INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
          INTEGER INDARR
        INTEGER IARG1, IARG2
        V(INDARR)=V(IARG1)*V(IARG2)
        VBAR(INDARR)=0.0$P 0
        OPCODE(INDARR)=30
        ARG1(INDARR)=IARG1
        ARG2(INDARR)=IARG2
        BMULTG=INDARR
        INDARR=INDARR+1
      END

...

      INTEGER FUNCTION SING(IARG1)
      COMMON /AD_F2/ V, VBAR, VDOT, VBARDT, OPCODE, ARG1, ARG2, INDARR
       REAL*8 V($NADARR), VBAR($NADARR), VDOT($NADARR),VBARDT($NADARR)
       INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
          INTEGER INDARR
        INTEGER IARG1
        V(INDARR)=SIN(V(IARG1))
        VBAR(INDARR)=0.0$P 0
        OPCODE(INDARR)=60
        ARG1(INDARR)=IARG1
        SING=INDARR
        INDARR=INDARR+1
      END

...

      SUBROUTINE RVRSWP()
      COMMON /AD_F2/ V, VBAR, VDOT, VBARDT, OPCODE, ARG1, ARG2, INDARR
       REAL*8 V($NADARR), VBAR($NADARR), VDOT($NADARR),VBARDT($NADARR)
       INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
          INTEGER INDARR
        REAL*8 DERIV, SGN
        INTEGER I
        INTEGER NINTV
        DO 999, I=INDARR-1, 1, -1
```

```
      IF(OPCODE(I).EQ.10) THEN
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)
        VBAR(ARG2(I))=VBAR(ARG2(I))+VBAR(I)
      ELSEIF(OPCODE(I).EQ.20) THEN
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)
        VBAR(ARG2(I))=VBAR(ARG2(I))-VBAR(I)
      ELSEIF(OPCODE(I).EQ.30) THEN
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*V(ARG2(I))
        VBAR(ARG2(I))=VBAR(ARG2(I))+VBAR(I)*V(ARG1(I))
      ELSEIF(OPCODE(I).EQ.40) THEN
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)/V(ARG2(I))
        VBAR(ARG2(I))=VBAR(ARG2(I))-VBAR(I)*V(I)/V(ARG2(I))
      ELSEIF(OPCODE(I).EQ.50) THEN
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*V(I)
      ELSEIF(OPCODE(I).EQ.51) THEN
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)/V(ARG1(I))
      ELSEIF(OPCODE(I).EQ.52) THEN
        IF (V(ARG1(I)).LE.0) THEN
          NINTV=NINT(V(ARG2(I)))
          IF (ABS(NINTV-V(ARG2(I)))/MAX(1,ABS(NINTV)).LE.1.0$P-14)
&                 THEN
            VBAR(ARG1(I))=VBAR(ARG1(I))+
&                VBAR(I)*V(ARG2(I))*V(ARG1(I))**(NINTV-1)
          ELSE
            WRITE(*,*) '--- RVRSWP: POWER ARG ERROR! ---'
          ENDIF
        ELSE
          VBAR(ARG1(I))=VBAR(ARG1(I))+
&              VBAR(I)*V(ARG2(I))*V(ARG1(I))**(V(ARG2(I))-1.0$P 0)
        ENDIF
        IF(V(ARG1(I)).NE.0.0$P 0) THEN
          VBAR(ARG2(I))=VBAR(ARG2(I))+VBAR(I)*V(I)*
&                       LOG(ABS(V(ARG1(I))))
        ELSE
          WRITE(*,*) '--- RVRSWP: LOG ARG ERROR! ---'
        ENDIF
      ELSEIF(OPCODE(I).EQ.53) THEN
        NINTV=NINT(V(ARG2(I)))
        VBAR(ARG1(I))=VBAR(ARG1(I))+
&                VBAR(I)*NINTV*V(ARG1(I))**(NINTV-1)
      ELSEIF(OPCODE(I).EQ.54) THEN
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)/V(ARG1(I))/LOG(10D0)
      ELSEIF(OPCODE(I).EQ.60) THEN
        DERIV=COS(V(ARG1(I)))
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
      ELSEIF(OPCODE(I).EQ.61) THEN
```

```
          DERIV=-SIN(V(ARG1(I)))
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.62) THEN
          DERIV=1/(COS(V(ARG1(I))))**2
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.64) THEN
          DERIV=COSH(V(ARG1(I)))
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.65) THEN
          DERIV=SINH(V(ARG1(I)))
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.66) THEN
          DERIV=1/(COSH(V(ARG1(I))))**2
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.70) THEN
          DERIV=1/SQRT(1-V(ARG1(I))**2)
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.71) THEN
          DERIV=-1/SQRT(1-V(ARG1(I))**2)
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.72) THEN
          DERIV=1/(1+V(ARG1(I))**2)
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        ELSEIF(OPCODE(I).EQ.80) THEN
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*SGN(V(ARG1(I)))
        ELSEIF(OPCODE(I).EQ.90) THEN
          VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)/(2.0$P 0*V(I))
        ELSEIF(OPCODE(I).EQ.1) THEN
          CONTINUE
        ELSEIF(OPCODE(I).EQ.2) THEN
       CONTINUE
        ENDIF
 999    CONTINUE
      END

...

      INTEGER FUNCTION BMULTH(IARG1, IARG2)
      COMMON /AD_F2/ V, VBAR, VDOT, VBARDT, OPCODE, ARG1, ARG2, INDARR
       REAL*8 V($NADARR), VBAR($NADARR), VDOT($NADARR),VBARDT($NADARR)
        INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
        INTEGER INDARR
        INTEGER IARG1, IARG2
        V(INDARR)=V(IARG1)*V(IARG2)
        VDOT(INDARR)=V(IARG1)*VDOT(IARG2)+VDOT(IARG1)*V(IARG2)
        VBAR(INDARR)=0.0$P 0
```

```
        VBARDT(INDARR)=0.0$P 0
        OPCODE(INDARR)=30
        ARG1(INDARR)=IARG1
        ARG2(INDARR)=IARG2
        BMULTH=INDARR
        INDARR=INDARR+1
      END

...

      INTEGER FUNCTION SING2(IARG1)
      COMMON /AD_F2/ V, VBAR, VDOT, VBARDT, OPCODE, ARG1, ARG2, INDARR
       REAL*8 V($NADARR), VBAR($NADARR), VDOT($NADARR),VBARDT($NADARR)
        INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
        INTEGER INDARR
        INTEGER IARG1
        V(INDARR)=SIN(V(IARG1))
        VDOT(INDARR)=COS(V(IARG1))*VDOT(IARG1)
        VBAR(INDARR)=0.0$P 0
        VBARDT(INDARR)=0.0$P 0
        OPCODE(INDARR)=60
        ARG1(INDARR)=IARG1
        SING2=INDARR
        INDARR=INDARR+1
      END

...

      SUBROUTINE RVRSWPH()
      COMMON /AD_F2/ V, VBAR, VDOT, VBARDT, OPCODE, ARG1, ARG2, INDARR
       REAL*8 V($NADARR), VBAR($NADARR), VDOT($NADARR),VBARDT($NADARR)
        INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
        INTEGER INDARR
        REAL*8 DERIV,SGN
        INTEGER I, ZAP_KOEF
        INTEGER NINTV
        REAL*8 VEXPI
        DO 998, I=INDARR-1, 1, -1
          IF(OPCODE(I).EQ.10) THEN
            VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)
            VBARDT(ARG1(I))=VBARDT(ARG1(I))+
     &                      VBARDT(I)
            VBAR(ARG2(I))=VBAR(ARG2(I))+VBAR(I)
            VBARDT(ARG2(I))=VBARDT(ARG2(I))+
     &                      VBARDT(I)
          ELSEIF(OPCODE(I).EQ.20) THEN
```

```
      VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)
      VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                     VBARDT(I)
      VBAR(ARG2(I))=VBAR(ARG2(I))-VBAR(I)
      VBARDT(ARG2(I))=VBARDT(ARG2(I))-
&                     VBARDT(I)
    ELSEIF(OPCODE(I).EQ.30) THEN
      VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*V(ARG2(I))
      VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                     VBARDT(I)*V(ARG2(I))+
&                     VBAR(I)*VDOT(ARG2(I))
      VBAR(ARG2(I))=VBAR(ARG2(I))+VBAR(I)*V(ARG1(I))
      VBARDT(ARG2(I))=VBARDT(ARG2(I))+
&                     VBARDT(I)*V(ARG1(I))+
&                     VBAR(I)*VDOT(ARG1(I))
    ELSEIF(OPCODE(I).EQ.40) THEN
      VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)/V(ARG2(I))
      VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                     (VBARDT(I)*V(ARG2(I))-
&                      VBAR(I)*VDOT(ARG2(I)))/
&                      (V(ARG2(I))*V(ARG2(I)))

      VBAR(ARG2(I))=VBAR(ARG2(I))-VBAR(I)*V(I)/V(ARG2(I))
      VBARDT(ARG2(I))=VBARDT(ARG2(I))-
&                     VBARDT(I)*V(I)/V(ARG2(I))-
&                     VBAR(I)*VDOT(I)/V(ARG2(I))+
&                     VBAR(I)*V(I)*VDOT(ARG2(I))/
&                     (V(ARG2(I))*V(ARG2(I)))
    ELSEIF(OPCODE(I).EQ.50) THEN
      VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*V(I)
      VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                     VBARDT(I)*V(I)+
&                     VBAR(I)*VDOT(I)
    ELSEIF(OPCODE(I).EQ.51) THEN
      VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)/V(ARG1(I))
      VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                     VBARDT(I)/V(ARG1(I))-
&                     VBAR(I)*VDOT(ARG1(I))/
&                     (V(ARG1(I))*V(ARG1(I)))
    ELSEIF(OPCODE(I).EQ.52) THEN
      IF(
&       (ABS(-2*INT((V(ARG2(I))-1.0$P 0)/2)+
&         V(ARG2(I))-1.0$P 0-1).LT.1.0$P-4)
&       .AND.
&       (V(ARG1(I)).LT.0)
&       ) THEN
```

```
          ZAP_KOEF=-1
        ELSE
          ZAP_KOEF= 1
        ENDIF
        IF (V(ARG1(I)).LE.0.0$P 0) THEN
          IF (ABS(V(ARG1(I))).LT.1.0$P-60) V(ARG1(I))=1.0$P-60
          NINTV=NINT(V(ARG2(I)))
          IF (ABS(NINTV-V(ARG2(I)))/MAX(1,ABS(NINTV)).LE.1.0$P-14)
&             THEN
            VBAR(ARG1(I))=VBAR(ARG1(I))+
&                VBAR(I)*V(ARG2(I))*V(ARG1(I))**(NINTV-1)
            VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                VBARDT(I)*V(ARG2(I))*V(ARG1(I))**(NINTV-1)+
&                VBAR(I)*VDOT(ARG2(I))*V(ARG1(I))**(NINTV-1)+
&                VBAR(I)*V(ARG2(I))*V(ARG1(I))**(NINTV-1)*
&                (VDOT(ARG2(I))*LOG(ABS(V(ARG1(I))))+
&                (V(ARG2(I))-1.0$P 0)*VDOT(ARG1(I))/V(ARG1(I)))*
&                ZAP_KOEF
          ELSE
            WRITE(*,*) '--- RVRSWPH: POWER ARG ERROR! ---'
          ENDIF
        ELSE
          VEXPI=V(ARG2(I))-1.0$P 0
          VBAR(ARG1(I))=VBAR(ARG1(I))+
&             VBAR(I)*V(ARG2(I))*V(ARG1(I))**VEXPI
          VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&             VBARDT(I)*V(ARG2(I))*V(ARG1(I))**VEXPI+
&             VBAR(I)*VDOT(ARG2(I))*V(ARG1(I))**VEXPI+
&             VBAR(I)*V(ARG2(I))*V(ARG1(I))**VEXPI*
&             (VDOT(ARG2(I))*LOG(ABS(V(ARG1(I))))+
&             VEXPI*VDOT(ARG1(I))/V(ARG1(I)))*ZAP_KOEF
        ENDIF
        IF(V(ARG1(I)).GT.0) THEN
          VBAR(ARG2(I))=VBAR(ARG2(I))+VBAR(I)*V(I)*LOG(V(ARG1(I)))
          VBARDT(ARG2(I))=VBARDT(ARG2(I))+
&              VBARDT(I)*V(I)*LOG(V(ARG1(I)))+
&              VBAR(I)*VDOT(I)*LOG(V(ARG1(I)))+
&              VBAR(I)*V(I)*VDOT(ARG1(I))/V(ARG1(I))
        ENDIF

      ELSEIF(OPCODE(I).EQ.53) THEN
        NINTV=NINT(V(ARG2(I)))
        VBAR(ARG1(I))=VBAR(ARG1(I))+
&                VBAR(I)*NINTV*V(ARG1(I))**(NINTV-1)
        VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                VBARDT(I)*NINTV*V(ARG1(I))**(NINTV-1)+
```

```
&               VBAR(I)  *NINTV*V(ARG1(I))**(NINTV-2)*
&               VDOT(ARG1(I))*(NINTV-1)
      ELSEIF(OPCODE(I).EQ.54) THEN
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)/V(ARG1(I))/LOG(10D0)
        VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                     VBARDT(I)/V(ARG1(I))/LOG(10D0)-
&                     VBAR(I)*VDOT(ARG1(I))/
&                     (V(ARG1(I))*V(ARG1(I)))/LOG(10D0)
      ELSEIF(OPCODE(I).EQ.60) THEN
        DERIV=COS(V(ARG1(I)))
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                     VBARDT(I)*DERIV-
&                     VBAR(I)*SIN(V(ARG1(I)))*VDOT(ARG1(I))
      ELSEIF(OPCODE(I).EQ.61) THEN
        DERIV=-SIN(V(ARG1(I)))
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                     VBARDT(I)*DERIV-
&                     VBAR(I)*COS(V(ARG1(I)))*VDOT(ARG1(I))
      ELSEIF(OPCODE(I).EQ.62) THEN
        DERIV=1/(COS(V(ARG1(I))))**2
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                     VBARDT(I)*DERIV+
&                     VBAR(I)*VDOT(ARG1(I))*2*SIN(V(ARG1(I)))*
&                     DERIV/COS(V(ARG1(I)))
      ELSEIF(OPCODE(I).EQ.64) THEN
        DERIV=COSH(V(ARG1(I)))
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                     VBARDT(I)*DERIV+
&                     VBAR(I)*VDOT(ARG1(I))*SINH(V(ARG1(I)))
      ELSEIF(OPCODE(I).EQ.65) THEN
        DERIV=SINH(V(ARG1(I)))
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                     VBARDT(I)*DERIV+
&                     VBAR(I)*VDOT(ARG1(I))*COSH(V(ARG1(I)))
      ELSEIF(OPCODE(I).EQ.66) THEN
        DERIV=1/(COSH(V(ARG1(I))))**2
        VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
        VBARDT(ARG1(I))=VBARDT(ARG1(I))+
&                     VBARDT(I)*DERIV-
&                     2*VBAR(I)*VDOT(ARG1(I))*SINH(V(ARG1(I)))*
&                     DERIV/COSH(V(ARG1(I)))
```

```
          ELSEIF(OPCODE(I).EQ.70) THEN
            DERIV=1/SQRT(1-V(ARG1(I))**2)
            VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
            VBARDT(ARG1(I))=VBARDT(ARG1(I))+
     &                      VBARDT(I)*DERIV+
     &                      VBAR(I)*VDOT(ARG1(I))*V(ARG1(I))*
     &                      (-DERIV)**3
          ELSEIF(OPCODE(I).EQ.71) THEN
            DERIV=-1/SQRT(1-V(ARG1(I))**2)
            VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
            VBARDT(ARG1(I))=VBARDT(ARG1(I))+
     &                      VBARDT(I)*DERIV-
     &                      VBAR(I)*VDOT(ARG1(I))*V(ARG1(I))*
     &                      (-DERIV)**3
          ELSEIF(OPCODE(I).EQ.72) THEN
            DERIV=1/(1+V(ARG1(I))**2)
            VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV
            VBARDT(ARG1(I))=VBARDT(ARG1(I))+
     &                      VBARDT(I)*DERIV-
     &                      2*VBAR(I)*VDOT(ARG1(I))*V(ARG1(I))*
     &                      DERIV**2
          ELSEIF(OPCODE(I).EQ.80) THEN
            VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*SGN(V(ARG1(I)))
            VBARDT(ARG1(I))=VBARDT(ARG1(I))+
     &                      VBARDT(I)*SGN(V(ARG1(I)))
          ELSEIF(OPCODE(I).EQ.90) THEN
            VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)/(2.0*V(I))
            VBARDT(ARG1(I))=VBARDT(ARG1(I))+
     &                      VBARDT(I)/(2.0*V(I))-
     &                     VBAR(I)*VDOT(ARG1(I))/(4.0*V(I)*V(ARG1(I)))
          ELSEIF(OPCODE(I).EQ.2) THEN
            CONTINUE
          ELSEIF(OPCODE(I).EQ.1) THEN
            CONTINUE
          ELSE
            WRITE(*,*) '--- NEDEFINOVANA OPERACE! ---'
          ENDIF
 998    CONTINUE
      END
$ENDADD
```

Reference

[1] A. Griewank: *Evaluation Derivatives: Principles and Techniques of Algorithmic Differentiation.* SIAM, Philadelphia, 2000.

[2] J. Hartman: *Realizace metod pro automatické derivování.* Diplomová práce, MFF UK, Praha, 2001.

[3] L. Lukšan: *Numerické optimalizační metody.* Technická zpráva V-930, ICS AS CR, Praha, 2005.

[4] L. Lukšan, M. Tůma, J. Hartman, J. Vlček, N. Ramešová, M. Šiška, C. Matonoha: *UFO 2006 - Interactive system for universal functional optimization.* Technická zpráva V-977, ICS AS CR, Praha, 2006.

[5] J. Hartman, J. Zítko: *Principy automatického derivování.* Výzkumná zpráva, Katedra numerické matematiky, MFF UK, Praha, 2006.