Automatikus differenciálás

Csendes Tibor

A differenciálhányadosoknak fontos szerepük van a numerikus matematikában, számos területen szinte elengedhetetlen a használatuk. Ilyen problémák találhatók például az optimalizálásban, a nemlineáris egyenletmegoldásban, az irányításelméletben és az érzékenységvizsgálatban is. Talán a legismertebb eset a függvények zérushelyének megkeresése, itt a derivált használatával működő Newton-Rawson eljárás konvergenciasebessége lényegesen jobb, mint a deriváltakat nem használó szelő- vagy húrmódszeré. Egy példa saját tapasztalatból [1]: bizonyos globális optimalizálási feladatok megoldása a deriváltak használata nélkül egyszerűen lehetetlen volt (a fellépő hatalmas tárigény miatt) — míg a gradiens alkalmazásával hatékony algoritmust lehetett megadni. Ma már egyes programozási nyelvek (pl. a PASCAL-XSC [7]) is támogatják az automatikus differenciálást megfelelő adattípussal és műveletekkel, és számos szoftver is használja ezt a deriválási módszert (pl. [10]-ban leírt optimalizálási eljárás, amely csak a célfüggvény megadását igényli).

1. Hogyan szokás a deriváltakat számítógépen előállítani, és miért nem igazán jók ezek a megoldások? A leggyakrabban használt két módszer a deriváltértékek előállítására azok numerikus közelítése és a "kézzel" való deriváltmeghatározás a deriválási szabályok alkalmazásával.

Tekintsük például az $f(x) = (x-1)^2$ függvényt az $x = 2$ pontban. A numerikus differenciálás a derivált értékét (az egyik szokásos módszerrel) az $(f(x) - f(x-\delta))/\delta$ becsléssel közelíti, ahol δ egy a számábrázolásnak megfelelően választott 0 és 1 közötti kis szám. $\delta = 10^{-8}$ esetén $f'(2)$-re így (Fortranban, dupla pontossággal) 1.99999997674282781190 adódik. A deriválási szabályok alkalmazásával az $f'(x) = 2x - 2$ függvényt kapjuk, ennek számítógépes kiértékelése (lényegében) 2-t ad.

A legtöbb numerikus matematikai monográfia és a professzionális numerikus programcsomagok többsége is ezt a két utat javasolja, bár mindkét

módszernek vannak olyan gyengéi, amelyek számos feladatban lehetetlenné vagy értelmetlenné teszik alkalmazásukat. A ritka kivételek egyike Skeel és Keiper könyve [11], amely a szimbolikus differenciálással szemben is az automatikus deriválást javasolja.

Érdekes összefüggések vannak a deriválás és az integrálás analitikus, illetve numerikus meghatározása között is. Az analitikus deriválás képlettel adott, elemi függvényekből felépített függvényekre könnyen, csaknem mechanikusan végrehajtható, míg az analitikus integrálás nehéz vagy akár lehetetlen is lehet. Ezzel szemben a numerikus közelítés a deriváltra gyakran pontatlan, míg az integrálra általában pontosabb.

A numerikus deriváltak viszonylag könnyen programozhatók, sokszor a könyvtári program maga állítja őket elő, ha a felhasználó nem adott meg szubrutint az analitikus deriváltak kiszámítására. A numerikus derivált használatának előnye, hogy

+ nincs előzetes munkaráfordítás a deriváltak "kézzel" történő előállítására,

+ emiatt javítani sem kell az azok programozása során elkövetett hibákat, és

+ akkor is működik, ha az illető függvény képletét nem ismerjük, csak a kiszámolására szolgáló szubrutin adott.

Hátránya viszont, hogy

– a levágási hiba miatt sok értékes jegy veszik el. Ez a jelenség csak bonyolult, és nem is minden számítógépes környezetben rendelkezésre álló eszközökkel csökkenthető (változó méretű számábrázolás, racionális aritmetika stb.).

– a gyorsan változó deriváltak becslésére alkalmatlan.

Az optimalizálásban általánosan elfogadott hüvelykszabály szerint (hacsak lehetséges) érdemes előállítani a deriváltakat számító szubrutinokat. Ezen eljárás előnye, hogy

+ a levágási hiba nem jelentkezik, a kiszámított deriváltértékek általában csak nagyon kis kerekítési hibával terheltek, és

+ a gyorsan változó deriváltértékek is jól meghatározhatók.

A hátránya ezzel szemben, hogy

– a deriváltak képletének meghatározása munkaigényes, és a "kézzel" való előállítás esetén gyakran komoly hibaforrás, valamint

– csak a képlettel adott függvények deriváltja határozható meg ilyen módon, tehát a kizárólag algoritmussal adottakat általában nem lehet így deriválni.

Itt kell megjegyezni, hogy a számítógépes algebrarendszerek (mint például a Mathematica, a Maple vagy a Derive) szimbolikus manipulációval elő tudják állítani a képlettel adott függvények deriváltjait. Így ez az előkészítő munka legalább számítógépesíthető, tehát nem feltétlenül kell "kézzel" végrehajtani. Az ilyen szimbolikus deriválás, a vele járó egyszerűsítés és a programozási nyelvre való alakítás időigénye nagyon változó, mindenesetre a számítógépes algebrarendszerek sokat fejlődtek ezen a téren az utóbbi időben [5].

Az automatikus differenciálás egyszerűen abból az igényből fakadt, hogy az előző módszerek előnyeit kell egyesíteni a hátrányok elhagyásával. Olyan eljárást kerestek tehát, amely

+ lényegében nem igényel előzetes ráfordítást a deriváltak "kézzel-" vagy akár számítógépes algebrarendszerrel, szimbolikus manipulációval való meghatározására,

+ emiatt nem is kell a megfelelő szubrutinokat programozni és javítani,

+ akkor is működik, ha csak az illető függvény kiszámolására szolgáló szubrutin adott, de a függvény képlete nem ismert,

+ a levágási hiba miatt nem vesznek el értékes jegyek,

+ a gyorsan változó deriváltak meghatározására is alkalmas, és

+ a deriváltak kiszámításának műveletigénye általában kisebb, mint a numerikus deriválásé, illetve az analitikus deriváltakat kiszámító szubrutinoké.

1. Táblázat. Néhány alapművelet és elemi függvény differenciálása

$y = f(x)$	$a \pm x$	$a * x$	a/x	$\sqrt{x}$	$\log(x)$	$\exp(x)$	$\cos(x)$
$f'(x)$	± 1	a	$-y/x$	$0.5/y$	$1/x$	y	$-\sin(x)$

Maga az ötlet nem nagyon bonyolult, és jellemző módon többen egymástól függetlenül rátaláltak [5, 8, 12]. Ha valaki kedvet érez hozzá, maga is megpróbálhatja az automatikus differenciálást újra felfedezni: az előző feltételeket teljesítő eljárást kell megadni (eddig nem árultunk el semmi lényegeset a trükkből).

2. Az ötlet. A trükk mindössze annyi, hogy használjuk az adott függvényre ismert kiszámítási eljárást az egyes műveletekhez tartozó deriválási szabályokkal együtt. Például ha $f(x) = f_1(x) * f_2(x)$, akkor legyen $f'(x)$ értéke $f_1'(x) * f_2(x) + f_1(x) * f_2'(x)$, ahol $f_1'(x)$ és $f_2'(x)$ értéke már ismert. Minden egyes részletszámítással együtt tehát a rá vonatkozó, az aktuális változó- és konstansértékekhez tartozó deriváltértéket is meghatározzuk, és tároljuk. A kiinduláshoz a változó deriváltja természetesen 1, a konstansé nulla. Az 1. Táblázat egyes alapműveletek és elemi függvények differenciálásának formális leírását tartalmazza, itt x változó, a pedig konstans.

Az automatikus differenciálás implementálása során célszerű olyan adatszerkezetet választani, hogy minden, az illető függvény kiszámításában szerepet játszó változó és konstans számára egy rendezett párt használunk, amelynek első tagja a szokásos értéket tartalmazza majd, a második tag pedig a hozzá tartozó deriváltértéket. Ilyen adatstruktúrával az új műveleteket egyszerű felírni szubrutinok segítségével, vagy egyes újabb programozási nyelvekben (pl. C++ vagy FORTRAN-90) az eredeti műveletek és standard függvények definíciójának az új adatszerkezetre való kiterjesztésével (operation overloading). Az utóbbi esetben a már működő, az eredeti függvényt kiszámító programban csak az adattípust kell kicserélni (pl. "real" helyett "derivative" vagy "gradient"), és máris rendelkezésre állnak a kívánt deriváltértékek.

Tekintsünk egy egyszerű példát az automatikus differenciálás használatára: határozzuk meg ismét az $f(x) = (x-1)^2$ függvény deriváltját az $x = 2$ pontban! A differenciálhányados-függvény tehát $f'(x) = 2x - 2$, a keresett

deriváltérték pedig 2.

A változónkhoz tartozó pár $(2,1)$, a függvényben szereplő konstanshoz tartozó pedig $(1,0)$. A zárójelen belüli kifejezés $f(x)$ képletében a $(2,1) - (1,0) = (1,1)$ párt eredményezi. A négyzetreemelést szorzással értelmezve az $(1,1) * (1,1) = (1,2)$ párt kapjuk, amelyből kiolvasható, hogy $f(2) = 1$, és $f'(2) = 2$.

3. Kiterjesztések. A képlettel megadott függvények differenciálásával szemben szokás kiemelni az "algoritmusok differenciálását". Ezen az automatikus differenciálás egyszerű kiterjesztését értik feltételes utasításokat is tartalmazó eljárásokkal megadott függvények deriválására. Az utóbbiakkal kapcsolatban persze felvetődik, hogy differenciálhatók-e ezek egyáltalán. Szerencsére ez a probléma inkább elméleti jellegű, és a technikai megoldást nem nagyon befolyásolja [3].

A magasabbrendű deriváltak előállításához két út között választhatunk: vagy közvetlenül az egyes műveletekhez tartozó magasabbrendű deriválási képleteket használjuk (például, ha $f(x) = g(x) + h(x)$, akkor $f''(x) = g''(x) + h''(x)$), vagy az alacsonyabbrendű deriváltak kiszámítására már meglévő algoritmusra alkalmazzuk ismételten az algoritmusok differenciálását.

A többváltozós függvények differenciálására a bevezetett automatikus differenciálási módszer minden további nélkül alkalmazható, az egyes parciális deriváltak meghatározásakor csak a változó-konstans viszonyt kell mindig megfelelően tisztázni. Ez is könnyen programozható, és így a gradiens, a Hesse- és a Jacobi-mátrix kiszámítása is nagyon kényelmessé tehető.

4. Az automatikus differenciálás két változata. Az automatikus differenciálás legegyszerűbb megvalósítása az, amikor a különben már rendelkezésre álló, az adott függvényt kiszámító programot kibővítjük az egyes műveletekhez tartozó elemi deriválási lépésekkel - megtartva az eredeti algoritmus szerkezetét. Ezt a módszert a továbbiakban sima algoritmusnak fogjuk nevezni. Az angol nyelvű szakirodalomban nincs még kialakult egységes elnevezése, a "forward", "contravariant" vagy "bottom-up" jelzőkkel szokás megkülönböztetni (a másik, fordított eljárás angolul "reverse", "backward", "covariant" vagy "top-down"). A két eljárás lényegében az összetett függvények deriválásához használatos láncszabály végrehajtási irányában különbözik.

A sima eljárás például az $y = f(g(h(x), k(x)))$ függvény automatikus differenciálása során a

$$\begin{aligned} \mathrm{d}u &= h'(x)\mathrm{d}x, \\ \mathrm{d}v &= k'(x)\mathrm{d}x, \\ \mathrm{d}w &= [g_u(u,v)h'(x) + g_v(u,v)k'(x)]\mathrm{d}x, \\ \mathrm{d}y &= f'(w)[g_u(u,v)h'(x) + g_v(u,v)k'(x)]\mathrm{d}x \end{aligned}$$

sorrendet követi.

A fordított eljárás az ellentétes irányban alkalmazza a láncszabályt:

$$\begin{aligned} \mathrm{d}y &= f'(w)\mathrm{d}w, \\ \mathrm{d}y &= f'(w)[g_u(u,v)\mathrm{d}u + g_v(u,v)\mathrm{d}v], \\ \mathrm{d}y &= f'(w)[g_u(u,v)\mathrm{d}u + g_v(u,v)k'(x)\mathrm{d}x], \\ \mathrm{d}y &= f'(w)[g_u(u,v)\mathrm{d}h'(x) + g_v(u,v)k'(x)]\mathrm{d}x. \end{aligned}$$

A fordított algoritmus előnye abban van, hogy ez a végrehajtási sorrend lehetővé teszi többváltozós függvények differenciálása során bizonyos szükségtelen műveletek elhagyását. Ennek az az ára (amit a következő szakasz adatai is alátámasztanak), hogy a fordított algoritmus tárigénye magasabb, és a sima algoritmus egymenetes végrehajtásával szemben két menetet igényel.

A két változat közötti különbség megvilágítása céljából tekintsük most az $f(x) = x_1(1 - x_2)^2$ függvényt az $x = [2, 1]^T$ pontban. A síma eljárás az egyes végrehajtott műveletekkel együtt a megfelelő deriváltértékeket is meghatározza:

$$\begin{aligned} f_1 &= x_1 = 2 & d_1 &= (1, 0), \\ f_2 &= x_2 = 1 & d_2 &= (0, 1), \\ f_3 &= 1 & d_3 &= (0, 0), \\ f_4 &= f_3 - f_2 = 0 & d_4 &= d_3 - d_2 = (0, -1), \\ f_5 &= f_4^2 = 0 & d_5 &= 2f_4 d_4 = (0, 0), \\ f_6 &= f_1 f_5 = 2 & d_6 &= f_1 d_5 + d_1 f_5 = (0, 0). \end{aligned}$$

A síma eljárás eközben esetleg többször is végrehajtja ugyanazt a műveletet, viszont nem igényli a kiszámítási fa létrehozását és tárolását. A fordított

algoritmus ezzel szemben először meghatározza az f_i értékeket és a kiszámítási fát, majd ennek segítségével előállítja a $d_i = \partial f / \partial f_i$ értékeket:

$$
\begin{aligned}
d_6 &= 1 \\
d_5 &= d_6 \tfrac{\partial f_6}{\partial f_5} = d_6 f_1 = 2, \\
d_4 &= d_5 \tfrac{\partial f_5}{\partial f_4} = d_5 2 f_4 = 0, \\
d_3 &= d_4 \tfrac{\partial f_4}{\partial f_3} = d_4 1 = 0, \\
d_2 &= d_4 \tfrac{\partial f_4}{\partial f_2} = d_4 (-1) = 0, \\
d_1 &= d_6 \tfrac{\partial f_6}{\partial f_1} = d_6 f_5 = 0.
\end{aligned}
$$

A gradiens értékét a $[d_1, d_2]^T$ vektor adja.

5. Művelet- és tárigény. A 2. Táblázat az automatikus differenciálás két változatának művelet- és tárigényét adja meg néhány gyakori deriválási feladatra. A legmeglepőbb adat talán az, hogy egy többváltozós függvénynek és gradiensének meghatározása a fordított algoritmussal legfeljebb négyszerannyi műveletet igényel mint az illető függvény kiszámítása. A felső korlát tehát nem is függ közvetlenül az illető függvény változóinak számától.

Az automatikus differenciálás műveletigénye nagyjából megfeleltethető egy ciklusmentes gráfban a legrövidebb út megkeresése műveletigényének, hozzáadva a kiszámítási gráf létrehozásának műveletigényét. A tárigény nagy részét a kiszámítási gráf tárolása okozza. A tár- és műveletigény javítása terén még várhatók további eredmények, de az is látszik, hogy a tárigény inkább csak a műveletigény rovására csökkenthető (és viszont).

6. Az automatikus differenciálás veszélyei. Az előzőek alapján úgy tűnhet, hogy az automatikus differenciálás számítógépes megvalósítása problémamentes. Sajnos nem egészen ez a helyzet, íme néhány példa:

1. A zérus gyökök esete. Tekintsük az $f(x) = \sqrt{x_1^4 + x_2^4}$ függvényt. Ez differenciálható, és a gradiense a $(0.,0.)^T$ pontban $(0.,0.)^T$. Az automatikus differenciálás a négyzetgyök művelethez azonban nem tud értéket rendelni, ha a gyök argumentuma nulla. A felhasználó számára ilyen esetekben az a leghasznosabb, ha az illető implementáció felhívja a figyelmet erre a hibalehetőségre, pl. az IEEE aritmetikát támogató számítógépekben a NaN (Not

2. Táblázat. A fontosabb automatikus differenciálási feladatok művelet- és tárigénye. Magyarázat: f: egy n-változós függvény, $\mathbf{f}$: m darab n-változós függvény, ∇f: az f gradiense, H: az f Hesse-mátrixa, J: az $\mathbf{f}$ Jacobi-mátrixa, $L(.)$: az argumentumok meghatározásának műveletigénye a $\{+, -, *, /, \sqrt{}, \log, \exp, \sin, \cos\}$ alapműveletek felett, és $S(.)$: az argumentumok meghatározásának tárigénye.

Feladat	Algoritmus	
	sima	fordított
$L(f, \nabla f)$	$\leq 4nL(f)$	$\leq 4L(f)$
$L(f, \nabla f, H)$	$\mathrm{O}(n^2 L(f))$	$\leq (10n+4)L(f)$
$L(\mathbf{f}, J)$	$\mathrm{O}(nL(\mathbf{f}))$	$\leq (3m+1)L(\mathbf{f})$
$S(f, \nabla f)$	$\mathrm{O}(S(f))$	$\mathrm{O}(S(f)+L(f))$
$S(f, \nabla f, H)$	$\mathrm{O}(S(f))$	$\mathrm{O}(S(f)+L(f))$
$S(\mathbf{f}, J)$	$\mathrm{O}(S(\mathbf{f}))$	$\mathrm{O}(S(\mathbf{f})+L(\mathbf{f}))$

a Number) érték hozzárendelésével.

2. A programelágazás esete. Tekintsük az alábbi utasítást:

$$\text{if } x = 1 \text{ then } f(x) = 1 \text{ else } f(x) = x^2$$

Világos, hogy az így definiált függvény folytonosan differenciálható, mégis az automatikus differenciálás a hamis $f'(1) = 0$ értéket adja. A példa kicsit erőltetettnek tűnik, de viszonylag gyakran előfordul, hogy adott függvény kiszámítására hasonló módon az argumentumok értékétől függően más és más eljárást adunk meg. Valódi megoldást erre a problémára nem lehet javasolni, legfeljebb azt, hogy a jelenség tudatában (különösen az egyenlőség-feltétellel adott programelágazás esetén) a felhasználó ellenőrizze, hogy ilyen hiba felléphet-e.

3. A határértékkel adott függvény esete. Eddig a függvények megadására mindig véges eljárást használtunk. Mi történik akkor, ha ez a leírás végtelen? Könnyű olyan alkalmazási példát mutatni, ahol a differenciálni kívánt függvényt csak egy iteratív sorozattal tudjuk jellemezni. A klasszikus analízis szerint viszont a differenciálás és a határértékképzés nem cserélhetők fel. Tekintsük a következő egyszerű függvénysorozatot:

$$f_1(x) = xe^{-x^2}, f_2(x) = xe^{-x^2}e^{-x^2}, \ldots, f_k = x(e^{-x^2})^k, \ldots$$

Automatikus differenciálással (is) $\lim_{k\to\infty} f_k'(0) = 1$, habár a valódi $f(x)$ határfüggvényre $f'(0) = 0$. Ebben az esetben is csak azt lehet tanácsolni, hogy a jelenség ismeretében az automatikus differenciálással nyert értékeket ellenőrizni kell. Ehhez viszonylag kényelmesen használható elméleti eredmények is rendelkezésre állnak [2].

7. Az automatikus differenciálás implementálása. A már említett PASCAL-XSC beépített adattípusainak és kiterjesztett alapműveleteinek a használata a legegyszerűbb. A felhasználónak mindössze a megfelelő adattípusokat kell megváltoztatnia. A FORTRAN-90 és C++ nyelvekben ezek az új adattípusok és a kiterjesztett műveletek megvalósítása után ugyanolyan kényelmesen lehet az automatikus differenciálás sima algoritmusát alkalmazni, mint a PASCAL-XSC támogatásával. Anonymous ftp-vel elérhető egy C++ bővítés az `iamk4515.mathematik.uni-karlsruhe.de` címen, a `/pub/toolbox/cxsc` könyvtárban. Ismertetése a [4] kötetben található.

A következő egyszerű példában az $f(x) = 25(x-1)/(x+2)$ függvény és deriváltja értékét határozzuk meg automatikus differenciálással az $x = 2$ pontban. A PASCAL-XSC implementáció érdekesebb részleteit adjuk csak meg.

```
program pelda (input,output);
type df_type = record f,df: real; end;

operator + (u,v: df_type) res: df_type;
begin res.f:=u.f+v.f; res.df:=u.df+v.df; end;

⋮

operator * (u,v: df_type) res: df_type;
begin res.f:=u.f*v.f; res.df:=u.df*v.f+u.f*v.df; end;

⋮
```

```
function df_var (h: real) : df_type;
begin df_var.f:=h; df_var.df:=1.0; end;

var x,f: df_type;
    h:   real;
begin
  h:=2.0;
  x:=df_var(h);
  f:=25*(x-1)/(x+2);
  writeln('f, df:',f.f,f.df);
end.
```

Számos kevésbé elegáns, de annál hatásosabb számítógépes eszköz (preprocesszor, precompiler, keresztfordító és más programcsomag) érhető el az automatikus differenciálás megvalósítására. Mintaként néhány híresebbnek az adatai:

- A JAKEF egy FORTRAN precompiler, amit az Argonne National Laboratory fejlesztett ki. Inputként egy skalár vagy vektorfüggvényt kiszámító szubrutint használ, és eredményként egy a gradienst, illetve a Jacobi-mátrixot előállító szubrutint ad. A fordított algoritmusra épül. A dokumentációt és a forrásszöveget is meg lehet kapni. A NETLIB nevű adatbázisban található, bővebb információt úgy kaphatunk, hogy a `netlib@research.att.com` E-mail címre egy "send index" üzenetet küldünk.

- A FORTRAN programok sima algoritmussal való automatikus differenciálására szolgáló GRAD programcsomag a következő címen érhető el: Larry Husch, Dept. Mathematics, University of Tenessee, Knoxville TN, USA, illetve `husch@wuarchive.wustl.edu` az elektronikus postával.

- Az ADOL-C egy C++ nyelven írt rendszer, amely C vagy C++ nyelvű algoritmusok differenciálására alkalmas sima és fordított eljárással is. A forráskód és a dokumentáció Andreas Griewank címén érhető el (Argonne National Labs, Argonne, IL 60439, USA, illetve elektronikus postával `griewank@antares.mcs.anl.gov`).

- A MAPLE nevű számítógépes algebrarendszer az 5.1-es változatától kezdve a szimbolikus deriválás mellett képes az automatikus differenciálásra is (a sima algoritmussal). Az "optimize" rutinja csökkentheti a műveletigényt, és az eredményt FORTRAN vagy C nyelven is ki tudja adni.

Meg kell még említeni, hogy az automatikus differenciáláshoz természetes módon kapcsolható a kerekítési hibák becslése és a számított deriváltak alsó- és felsőkorlátjainak meghatározása is. Az utóbbi feladatok (részben az intervallum-aritmetikára támaszkodva) szintén kényelmesen megoldhatók számítógépen [4, 7, 10].

IRODALOM

[1] Csendes, T., J. Pintér: The impact of accelerating tools on the interval subdivision algorithm for global optimization, European J. of Operational Research 65(1993) 314-320.

[2] Fischer, H.: Special problems in automatic differentiation, in: [Griewank – Corliss, 1991] 43-50.

[3] Griewank, A., G. Corliss (Eds.): Automatic Differentiation of Algorithms: Theory, Implementation, and Application. SIAM, Philadelphia, 1991.

[4] Hammer, R., M. Hocks, U. Kulisch, D. Ratz: C++ Toolbox for Verified Computing, Springer-Verlag, Berlin, 1995.

[5] Iri, M.: History of automatic differentiation and rounding error estimation, in: [Griewank – Corliss, 1991] 3-16.

[6] Kedem, G.: Automatic differentiation of computer programs, ACM Transactions on Mathematical Software 6(1980) 150-165.

[7] Klatte, R., U. Kulisch, M. Neaga, D. Ratz, Ch. Ullrich: PASCAL-XSC, Springer-Verlag, Berlin, 1991.

[8] Ostrovskij, G.M., Ju.M. Wolin, W.W. Borisov: Über die Berechnung von Ableitungen, Wissenschaftliche Zeitschrift der Technischen Hochschule für Chemie, Leuna-Merseburg 13(1971) 382-384.

[9] Rall, L.B.: Automatic Differentiation: Techniques and Applications. Lecture Notes In Computer Science, Vol. 120, Springer-Verlag, Berlin, 1981.

[10] Ratz, D.: Automatische Ergebnisverifikation bei globalen Optimierungsproblemen. Doktori értekezés, Karlsruhei Egyetem, 1992.

[11] Skeel, R.D., J.B. Keiper: Elementary Numerical Computing with MATHEMATICA. McGraw-Hill Inc., New York, 1993.

[12] Wengert, R.E.: A simple automatic derivative evaluation program, Communications of the ACM 7(1964) 463-464.

Csendes Tibor, JATE Kalmár Laboratórium, Szeged, Árpád tér 2. A cikk írása idején a Bázeli Egyetem Informatikai Intézetében volt ösztöndíjas a Svájci Szövetségi Ösztöndíjbizottság Különprogramja támogatásával.

Automatic Differentiation

(Summary)

Derivatives are required for optimization, integration of stiff differential equations, parameter estimation, sensitivity analysis, and in many other problems. Often, hand-coding of analytical derivative computations is too laborious and error-prone, and the use of finite difference approximations is too expensive and/or inaccurate. Sometimes symbolic algebra packages can be useful, but these are generally inadequate when the functions to be differentiated are defined by computer programs containing intermediate variables, loops, and conditionals. This is where automatic differentiation comes in. The article gives a review of the state of the art in automatic differentiation.