Egy intervallum alapú globális optimalizálási módszer és alkalmazása szenzor lokalizálási feladatra

Pál László és Csendes Tibor

Kivonat

A cikkben egy intervallum alapú optimalizálási módszer egy új implementációját mutatjuk be. Az algoritmust MATLAB környezetben valósítottuk meg az INTLAB csomagot használva, amely tartalmazza a szükséges intervallum aritmetikai műveleteket és az automatikus differenciálást is. A numerikus eredmények alapján az új INTLAB alapú implementáció hasonlóan hatékony, mint a C-XSC alapú eredeti algoritmus – eltekintve a számítási időtől. A program más hasonló programokkal összevetve könnyen telepíthető és használható. A cikkben vizsgálunk továbbá egy új feltételt a Newton lépés bekapcsolására, valamint egy szenzorhálózati alkalmazást mutatunk be, és tanulmányozunk az INTLAB alapú algoritmussal.

1. Bevezető

A megoldandó feladat az intervallum korlátos globális optimalizálási probléma a következő formában:

$$\min_{x \in X} f(x)$$

ahol $f : \mathbb{R}^n \to \mathbb{R}$, $X = \{x_i \in [\underline{x}_i, \overline{x}_i],\ i = 1, \ldots, n\}$, és $\underline{x}_i, \overline{x}_i \in \mathbb{R},\ i = 1, \ldots, n$. Az f függvényről általában feltételezzük, hogy sima. A mi algoritmusunk is erre támaszkodik, ennek ellenére nem sima függvények optimalizálására is használható, ha elhagyjuk belőle a Newton lépést és a konkavitási tesztet.

Globális optimalizálási módszerek segítségével nehéz matematikai feladatokat sikerült megoldani az elmúlt időszakban. Ezek a problémák a dinamikus rendszerek [3, 9, 10], a diszkrét geometria, illetve az optimális körpakolás [15, 20] témakörébe tartoznak. Sikeresen alkalmaztuk ezen módszereket továbbá elméleti kémiai feladatokra [2], valamint üzemelhelyezési feladatokban [22].

Célunk volt egy könnyen használható, megbízható globális optimalizálási módszer implementálása és tesztelése MATLAB környezetben. Egy korábbi sikeres implementációnk MATLAB környezetben a GLOBAL [11] nevű sztochasztikus globális optimalizálási algoritmus.

A jelen cikkben vizsgált eljárás csak egy, a célfüggvény kiszámítására szolgáló szubrutinra támaszkodik. Más információt nem használ a globális optimalizálási problémára vonatkozóan. Az eljárás maga határozza meg a célfüggvény gradiensét és Hesse mátrixát, felhasználva az INTLAB csomag automatikus differenciáló eljárásait.

2. Az algoritmus és implementálása

Az implementált algoritmus egy korlátozás és szétválasztás típusú módszer, amelynek pszeudokódját az 1. Algoritmusban adtuk meg. A módszer elődje a Numerical Toolbox for Verified Computing [12] csomagban volt implementálva. Jelenleg a következő gyorsító teszteket tartalmazza: középponti teszt, konkavitási teszt, monotonitási teszt, és intervallumos Newton lépés. A természetes befoglaláson kívül a középponti formulákat is alkalmazzuk mint befoglaló függvényt. Ha a

Algorithm 1: A tanulmányozott intervallum korlátos globális optimalizálási algoritmus

1. **function** GlobalOptimize (f, X, ϵ)
2. $L_{work} := \emptyset; L_{temp} := \emptyset$
3. $Y := X$; $\widetilde{f} := \overline{F}(\text{mid}(X))$
4. **repeat**
5. OptimalComponents(Y, k_1, k_2)
6. Trisection$(Y, k_1, k_2, U^1, U^2, U^3)$
7. **for** $i := 1$ **to** 3 **do**
8. **if** MonotonicityTest$(\nabla F(U^i))$ **then next** i
9. $f_U := F(U^i)$
10. **if** $\widetilde{f} < \underline{f_U}$ **then next** i
11. $f_U := f_U \cap \text{CenteredForm}(U^i, \nabla F(U^i))$
12. **if** $\overline{F}(\text{mid}(U^i)) < \widetilde{f}$ **then**
13. $\widetilde{f} := \overline{F}(\text{mid}(U^i))$
14. $L_{work} := \text{CutOffTest}(L_{work}, \widetilde{f})$
15. **if** $\widetilde{f} >= \underline{f_U}$ **then** $L_{temp} := L_{temp} \cup (U^i, \underline{f_U})$
16. **if** length$(L_{temp}) = 1$ **then**
17. $U := \text{Head}(L_{temp})$
18. **if** not ConcavityTest$(\nabla^2 F(U))$ **then**
19. NewtonStep$(f, U, \nabla^2 F(U), V, p)$
20. **for** $i := 1$ **to** p **do**
21. **if** MonotonicityTest$(\nabla F(V^i))$ **then next** i
22. $f_V := F(V^i) \cap \text{CenteredForm}(V^i, \nabla F(V^i))$
23. **if** $\overline{F}(\text{mid}(V^i)) < \widetilde{f}$ **then**
24. $\widetilde{f} := \overline{F}(\text{mid}(V^i))$
25. $L_{work} := \text{CutOffTest}(L_{work}, \widetilde{f})$
26. **if** $\widetilde{f} >= \underline{f_V}$ **then** $L_{work} := L_{work} \cup (V^i, \underline{f_V})$
27. **else**
28. **while** $L_{temp} \neq \emptyset$ **do**
29. $U := \text{Head}(L_{temp})$
30. $L_{work} := L_{work} \cup (U, \underline{f_U})$
31. **if** $L_{work} \neq \emptyset$ **then**
32. $Y := \text{Head}(L_{work})$
33. **if** $w(f_Y) < \epsilon$ **then**
34. $f^* := [\underline{f_Y}, \widetilde{f}]$
35. **return** Y, f^*
36. **until** $L_{work} = \emptyset$

gradiens befoglalása ismert, akkor az előbbi két befoglaló függvény metszete általában jó megközelítést ad a függvény értékkészletére.

Az algoritmusban használt jelölések: $w(X) = b - a$, ahol $X = [a, b]$, $\mathrm{mid}(X) = (a + b)/2$, és F az f célfüggvénynek megfelelő befoglaló függvény.

A keresési tartományok felosztására szeletelést, illetve fejlett felosztási irányt választó [14] szabályokat használunk. A szeletelés azt jelenti, hogy az intervallumot felosztjuk három másik intervallumra a két legmegfelelőbb irányt használva. A felosztási irány megválasztása a C típusú szabály ([7, 14]) alapján történik. Az algoritmus egydimenziós feladatok megoldására is alkalmazható. Ebben az esetben a szeletelés helyett egy sima ketté osztást használunk.

A felosztás során keletkezett intervallumokat rendezett listában tároljuk valamilyen rendezési szabály alapján. Az algoritmusban a pf^* heurisztikus paramétert [7] használjuk a lista rendezésére. Pontosabban, mindig a lista legnagyobb pf^* értékkel rendelkező intervallumát osztjuk fel. Egy előző cikkben [18] a befoglalások legkisebb alsó korlátja szerinti rendezést is vizsgáltunk.

Az algoritmus nem keresi meg az összes globális minimum pontot, hanem az elsőnél leáll, ha az azt tartalmazó intervallum befoglalásának szélessége kisebb, mint 10^{-8}.

Az implementálás során követtük a Markót Mihály Csaba által készített C-XSC alapú program [16] szerkezetét. Természetesen ahol lehetett, használtuk a MATLAB vektor struktúráit. Ez utóbbi programozás technika a processzor csővezetékének (pipeline) jobb kihasználása révén az eljárás gyorsítását eredményezi.

Az algoritmus tartalmaz egy fő iterációs struktúrát a 4. sortól a 36. sorig. Az iterációk során két listát kezelünk: egy munka-, valamint egy temporális listát. Minden iteráció elején meghatározzuk az optimális vágási irányokat (5. sor), majd ezek alapján felosztjuk az aktuális intervallumot három másik intervallumra (6. sor). A 8. és 15. sorok között rendre a kapott intervallumokra végrehajtunk egy monotonitási- (8. sor), egy kivágási tesztet (10. sor), valamint alkalmazzuk a középponti formulát (11. sor). A 14. sorban a munka listát aktualizáljuk az új felső korlát alapján. Ha az aktuális intervallumot nem sikerült eldobni az előbbi lépések során, akkor tároljuk a temporális listában (15. sor). A három intervallum feldolgozása után, ha a temporális lista csak egy elemet tartalmaz (16. sor), akkor erre végrehajtunk egy konkavitási tesztet (18. sor) valamint egy Newton lépést (19. sor). A Newton lépés eredményeként kapott intervallumokra a korábban leírt lépéseket hajtjuk végre. Ha a lista több mint egy elemet tartalmaz, akkor ezeket tároljuk a munka listában (30. sor). Az iteráció végén (31. és 35. sorok között), ha a munka lista nem üres, akkor kivesszük annak első elemét, és megvizsgáljuk, hogy az intervallum befoglalásának szélessége kisebb-e, mint egy előre megadott érték (33. sor). Ha ez a feltétel teljesül, akkor az intervallumot és a optimum értéket tartalmazó intervallummal együtt visszaadjuk eredményként (35. sor), különben folytatjuk a következő iterációval.

Az új algoritmus használatához szükség van az INTLAB csomag [19] telepítésére. Az INTLAB csomagot a Hamburgi Egyetemen fejlesztette ki Siegfried M. Rump és ingyenesen használható nem kereskedelmi céllal, illetve a kutatásban is. A program számos MATLAB verzió alatt volt tesztelve egészen az R2009b változatig. A csomag a

`http://www.ti3.tu-harburg.de/rump/intlab/`

helyről tölthető le, ahol található egy telepítési útmutató is. Letöltés után a programot ki kell csomagolni, majd MATLAB környezetben futtatni kell a `startintlab.m` szkriptet, amely tulajdonképpen telepíti a csomagot. Az INTLAB 5.5 és korábbi verziói esetén előfordulhat, hogy nem sikerül a telepítés. Ennek az lehet az oka, hogy a MATLAB az Intel Math Kernel Library (IMKL) csomagot használja a numerikus műveletekre. A megoldás az, hogy az Atlas könyvtárat használjuk az IMKL helyett. A Windows rendszer alatt a BLAS_VERSION környezeti változót kell beállítani az atlas***.dll értékre, ahol a '***' a processzor típusát jelenti (Pentium 4 esetén 'atlas_P4.dll'). A szükséges fájl a ...\MATLAB\bin\win32\ mappában található. Az új 6.0 verzió esetén a környezeti változó beállítása nélkül is problémamentesen települ az INTLAB.

Linux alatt a következő utasítással állíthatjuk be a megfelelő könyvtárat:

```
export BLAS_VERSION="atlas_P4.so"
```

Lényeges, hogy telepítés után valahányszor szükség van az INTLAB csomagra, mindig el kell indítani azt a `startintlab` paranccsal.

Az INTLAB elindítása után új intervallumot az alábbi utasítás segítségével hozhatunk létre:

```
infsup(a,b).
```

Az intervallumok alapértelmezett megjelenítési formája a bizonytalanságot megjelenítő mód. Például

```
a = infsup(3.14, 3.15)
```

eredménye

```
intval a = 3.15_.
```

A hagyományos intervallum megjelenítési formát a

```
intvalinit('displayinfsup')
```

paranccsal állíthatjuk be, amelynek eredményeképpen az intervallum kiírási alakja:

```
intval a = [3.14, 3.15].
```

Intervallumokkal persze különböző műveleteket is végezhetünk. Például az `x=infsup(0,1);` és `y=infsup(2,3);` intervallumok osztásának eredménye (`x/y`):

```
intval ans = [0.0000, 0.5000]
```

Míg `sin(x)` az

```
intval ans = [0.0000, 0.8415]
```

intervallumot eredményezi.

Számos más példát és bemutató programot találhatunk az INTLAB [19] leírásában. Az új MATLAB/INTLAB alapú intervallumos globális optimalizálási program letölthető a

```
www.inf.u-szeged.hu/∼csendes/Reg/regform.php
```

címről. A letöltött csomag tartalmazza a szükséges állományokat és a teszt környezetet is.

3. Az intervallumos globális optimalizáló program használata

Az intervallumos globális optimalizáló programot viszonylag egyszerű használni, ugyanis a felhasználónak csak az optimalizálandó feladatot kell elhelyeznie a TestFunctions mappába, majd futtatni a `MainTester` programot. A mappába egyszerre több feladatot is be lehet tenni, és megoldani.

Az optimalizálandó problémát két állomány segítségével kell megadni. Az egyik (ennek neve *.bnd lesz) tartalmazza a feladat olyan adatait, mint például a feladat neve, dimenziója, intervallum korlátok a változókra, és a megkövetelt pontosság. A másik fájl pedig a célfüggvényt adja meg (a neve *.m). Például a Shekel-5 standard globális optimalizálási tesztfeladat esetén a két állomány `sh5.bnd` és `sh5.m`. Az `sh5.bnd` fájl tartalma:

```
S5
4
0   10
0   10
0   10
0   10
1e-8
```

A célfüggvényt tartalmazó `sh5.m` állomány tartalma pedig:

```
function y = sh5(x)

m = 5;
a = ones(10,4);
a(1,:) = 4.0*a(1,:);
a(2,:) = 1.0*a(2,:);
a(3,:) = 8.0*a(3,:);
a(4,:) = 6.0*a(4,:);
for j = 1:2;
   a(5,2*j-1) = 3.0; a(5,2*j) = 7.0;
   a(6,2*j-1) = 2.0; a(6,2*j) = 9.0;
   a(7,j)     = 5.0; a(7,j+2) = 3.0;
   a(8,2*j-1) = 8.0; a(8,2*j) = 1.0;
   a(9,2*j-1) = 6.0; a(9,2*j) = 2.0;
   a(10,2*j-1)= 7.0; a(10,2*j)= 3.6;
end
c(1) = 0.1; c(2) = 0.2; c(3) = 0.2; c(4) = 0.4; c(5) = 0.4;
c(6) = 0.6; c(7) = 0.3; c(8) = 0.7; c(9) = 0.5; c(10)= 0.5;
s = 0.0;
for j = 1:m;
   p = 0.0;
   for i = 1:4
      p = p+(x(i)-a(j,i))^2;
   end
   s = s+1.0/(p+c(j));
end y = -s;
```

A MainTester futtatása után az eredményt az alábbi formában kapjuk meg:

```
Function name: S5

The set of global minimizers is located in the union of the
following boxes:

c1: [4.00003713662883, 4.00003718945147]
    [4.00013323800906, 4.00013329348396]
    [4.00003713910016, 4.00003717168197]
    [4.00013326916774, 4.00013328566954]

The global minimum is enclosed in:

[-10.153199679058694, -10.153199679058199]

Statistics:

   Iter   Feval   Geval   Heval     MLL  CPUt(sec)
     16     126      86       7      10       6.69
```

Előfordulhat természetesen, hogy eredményként több intervallumot is kapunk. A globális minimum pontok ezen intervallumok egyesítésében találhatók. Az eredmény kiírt alakja tartalmazza a feladat megoldása során mért szokásos hatékonysági mutatókat: az iteráció számot, a függvényhívások számát, a gradienshívások számát, a Hesse mátrix kiértékelések számát, a maximális lista hosszat és a CPU időt.

Az optimalizáló eljárás közvetlenül, parancssorból is használható a következő módon:

```
>> addpath('./','./Utils','TestFunctions')
>> amin = [0; 0; 0; 0]
>> amax = [10; 10; 10; 10]
>> b = infsup(amin,amax)
>> [intv, min, stats] = GOP(@sh5, b, 1e-8)
```

Lehetőség van arra is, hogy úgy használjuk az opimalizáló eljárást, hogy a szükséges két állományt nem adjuk meg, hanem a célfüggvényt inline módon definiáljuk. Erre tekintsük az alábbi példát:

```
>> f = inline('x(1)^2+x(2)^2+1')
>> amin = [-2; -2]
>> amax = [1; 1]
>> int = infsup(amin,amax)
>> [intv, min, stat] = GOP(f, int, 1e-8)
```

4. A C-XSC alapú algoritmus és az INTLAB alapú módszer összehasonlítása

Az INTLAB alapú algoritmust numerikusan teszteltük. A célunk az volt, hogy megvizsgáljuk ennek hatékonyságát a szokásos hatékonysági mutatókkal, valamint összehasonlítani a kapott eredményeket a C-XSC alapú algoritmus megfelelő mutatóival.

A teszteket MATLAB R2008a környezetben, INTLAB 5.5-ös verzió alatt végeztük, Pentium 4-es (2 Gbyte RAM memóriával ellátott, Core 2 Duo, 2 GHz-es processzorú) számítógépen. A tesztfeladatok halmaza tartalmazta az összes standard globális optimalizálási függvényt is. Hasonló feladat halmazon végeztünk tesztelést a [7, 8] cikkekben.

A numerikus eredményeket az 1. és 2. Táblázatokban összegeztük. A táblázatok első két oszlopa a probléma nevét és dimenzióját tartalmazza. A függvény nevek helyett azok rövidítéseit használtuk, például Shekel-5 helyett S5, Schwefel 3.2 helyett Sch3.2, Ratz-4 helyett R4 szerepel stb. A többi oszlop a szokásos hatékonysági mutatókat tartalmazza, mint amilyen az iteráció szám (ITSz), a függvényhívások száma (FHSz), a gradienshívások száma (GHSz), a Hesse mátrix kiértékelések száma (HHSz), a maximális munka lista hossza (MLH) és a szükséges CPU idő másodpercben (CPU).

A két implementáció esetén a hatékonysági mutatók (egy, a CPU idő kivételével) megegyeznek vagy hasonlók. Ez nagyjából annak tudható be, hogy a két algoritmus struktúrája közel megegyezik. A CPU idő esetén nagy eltérés figyelhető meg. Az INTLAB alapú implementáció átlagosan 442-szer több időt igényel ugyanannak a feladatnak a megoldásához. Az arányok az egyes feladatok esetén 16 és 1251 között változnak, és a médián értéke erre az arányra 345. A nagyobb számok azon feladatok esetén figyelhetők meg, ahol sok számításra volt szükség. A nagy futásidőbeli különbségek a MATLAB interpreter módban való működésének tudhatók be.

Az összehasonlítás eredményeképpen elmondható, hogy MATLAB környezetben az INTLAB alapú algoritmus egyszerűen használható, viszont hátránya a sebesség visszaesése. Ennek ellenére az új globális optimalizálási módszer hasznos modellező eszköz lehet optimalizálási feladatok kezdeti tanulmányozására. Különösen igaz ez olyan feladatok esetén, amelyekre a CPU idő a növekedés ellenére is elfogadható mértékű (legföljebb pár perc). A számítási környezet interaktivitása előnyös olyan esetekben, amikor kezdeti lehetőségek közötti választás céljából nagy számú kísérletet végzünk eltérő beállításokkal, illetve modellekkel.

5. A Newton lépés bekapcsolásának vizsgálata

A Newton lépést az algoritmus gyorsítására használjuk. Tulajdonképpen egy intervallumos Newton-Seidel lépést alkalmazunk a célfüggvény gradiensére, ezáltal megpróbálunk közeli korlátokat

1. Táblázat. A C-XSC- és az INTLAB alapú algoritmusok eredményeinek összehasonlítása. A táblázatban Dim a feladat dimenzióját jelenti, ITSz az iteráció számot, FHSz a függvényhívások számát, és GHSz a gradiens hívások számát.

		A C-XSC kód			Az INTLAB kód		
Feladat	Dim	ITSz	FHSz	GHSz	ITSz	FHSz	GHSz
S5	4	16	126	86	16	126	86
S7	4	18	129	84	17	121	78
S10	4	17	122	78	17	123	78
H3	3	38	256	187	23	147	99
H6	6	191	1 505	1 167	191	1 505	1 167
GP	2	76	458	229	76	458	229
SHCB	2	17	103	60	17	103	60
THCB	2	44	274	189	44	274	189
BR	2	44	250	177	44	250	177
RB	2	38	238	151	38	238	151
RB5	5	389	3 584	2 713	396	3 660	2 758
L3	2	47	293	170	47	293	170
L5	2	86	593	406	86	593	406
L8	3	11	80	55	11	80	55
L9	4	13	107	73	13	107	73
L10	5	15	125	86	15	125	86
L11	8	23	189	128	23	189	128
L12	10	30	254	175	30	254	175
L13	2	10	74	47	10	74	47
L14	3	15	120	77	15	120	77
L15	4	17	136	87	18	146	94
L16	5	19	142	88	19	142	88
L18	7	27	206	130	27	206	130
Schw2.1	2	113	803	580	113	804	580
Schw3.1	3	14	96	64	14	96	64
Schw2.5	2	50	293	205	50	293	205
Schw2.14	4	353	3 146	2 263	356	3 242	2 337
Schw2.18	2	3	21	13	3	21	13
Schw3.2	3	20	144	98	20	144	98
Schw3.7_5	5	45	309	208	45	309	208
Schw3.7_10	10	696	4 371	2 665	696	4 371	2 665
Griew5	5	25	190	117	25	190	117
Griew7	7	40	297	173	40	297	173
R4	2	35	210	125	35	210	125
R5	3	107	996	748	107	996	748
R6	5	140	1 516	1 221	140	1 516	1 221
R7	7	204	2 728	2 293	204	2 728	2 293
R8	9	310	4 723	4 063	320	4 881	4 201
EX2	5	5 975	37 816	27 834	9 279	59 605	44 126

2. Táblázat. A C-XSC -és az INTLAB alapú algoritmusok eredményeinek összehasonlítása. A táblázatban Dim a feladat dimenzióját jelenti, HHSz a Hesse mátrix kiértékelések számát, MLH a maximális lista hosszát, valamint CPU a futási időt másodpercben.

		A C-XSC kód			Az INTLAB kód		
Feladat	Dim	HHSz	MLH	CPU	HHSz	MLH	CPU
S5	4	7	10	0.02	7	10	5.66
S7	4	7	14	0.02	6	14	7.27
S10	4	6	16	0.03	6	17	10.36
H3	3	22	21	0.00	11	16	5.09
H6	6	86	64	0.43	86	64	97.45
GP	2	0	153	0.00	0	153	9.25
SHCB	2	3	22	0.00	3	22	1.70
THCB	2	21	24	0.00	21	24	3.75
BR	2	18	10	0.00	18	10	3.44
RB	2	11	11	0.00	11	11	1.59
RB5	5	309	74	0.15	317	79	93.13
L3	2	8	57	0.01	8	57	10.06
L5	2	31	32	0.05	31	32	26.25
L8	3	5	9	0.01	5	9	2.34
L9	4	7	13	0.01	7	13	4.08
L10	5	8	15	0.03	8	15	5.95
L11	8	9	28	0.07	9	28	14.11
L12	10	11	36	0.09	11	36	23.89
L13	2	4	9	0.01	4	9	1.45
L14	3	7	12	0.00	7	12	3.16
L15	4	7	19	0.00	8	19	4.84
L16	5	6	20	0.03	6	20	5.56
L18	7	8	26	0.01	8	26	10.80
Schw2.1	2	53	25	0.01	53	25	12.50
Schw3.1	3	5	6	0.00	5	6	1.58
Schw2.5	2	27	4	0.00	27	4	2.13
Schw2.14	4	209	119	0.10	216	123	47.98
Schw2.18	2	1	4	0.01	1	4	0.16
Schw3.2	3	11	7	0.00	11	7	1.66
Schw3.7_5	5	24	32	0.03	24	32	7.13
Schw3.7_10	10	192	818	0.48	192	818	183.59
Griew5	5	7	28	0.02	7	28	5.94
Griew7	7	8	58	0.06	8	58	12.48
R4	2	6	36	0.00	6	36	2.23
R5	3	71	57	0.05	71	57	27.20
R6	5	100	30	0.25	100	30	71.72
R7	7	168	41	0.35	168	41	184.78
R8	9	261	59	1.17	270	59	429.48
EX2	5	3 149	534	3.51	5 020	576	4 390.09

adni a minimum helyekre. A teljes Newton algoritmust nem futtatjuk le, mert az túl költséges lenne, ezért alkalmazunk csak iterációnként egy-egy lépést.

A Newton lépés alkalmazása minden egyes részintervallumra szintén költséges lenne, ezért célszerű valamilyen feltételt használni ennek bekapcsolására. Egy korábbi cikk [16] alapján algoritmusunkban a Newton lépést csak abban az esetben használtuk, ha a felosztás során keletkezett három részintervallumból a többi gyorsító lépés végrehajtása után már csak egy részintervallum maradt.

A továbbiakban egy új bekapcsolási feltételt vizsgálunk. Ennek az a lényege, hogy minden olyan részintervallumra alkalmazzuk a Newton lépést, amelynek szélessége kisebb, mint egy előre megadott érték, ugyanis a Newton lépés igazából akkor hatékony, ha az adott argumentum intervallum már elég kicsi. Az új feltételben 0.1 értéket használtunk az intervallum szélessége küszöbértékének.

5.1. Elméleti vizsgálat

Vannak esetek, amikor az előbbi feltételt alkalmazva a Newton lépés nem eredményes abban az értelemben, hogy vagy nem is csökken az intervallum mérete, vagy sok darabra osztja fel az aktuális intervallumot. A gyakorlatban többnyire az utóbbi eset szokott előfordulni, amely azért nem előnyös, mert hasonló eredményt érhetünk el egyszerű kettéosztással is, de jóval kisebb költséggel. A Newton lépés hatékonysága több-dimenziós esetben méginkább romolhat, ugyanis a dimenzió növekedéssel az új részintervallumok száma is növekszik.

A továbbiakban feltételezzük, hogy a másodrendű derivált befoglalása, amelyre a Newton lépés nem eredményes, az alábbi alakú:

$$F''_{uj}(X) = \left[\underline{F}''(X) - D \cdot w(F''(X)), \overline{F}''(X) + D \cdot w(F''(X))\right], \tag{1}$$

ahol $F''(X)$ a másodrendű derivált egy befoglalása az X intervallumon, és D egy pozitív valós szám. $F''(X)$ lehet a másodrendű derivált értékkészlete is X-en. Hasonló alakú befoglalófüggvényt feltételeztek a [5, 6] cikkekben is, a gyakorlatban használt befoglaló függvények körében a szimmetrikus túlbecslés gyakori.

A következő két tétel arra akar rávilágítani, hogy milyen esetekben nem lesz a Newton lépés hatékony.

1. Tétel. *Adott f egyváltozós függvény és X, $w(X) < \epsilon$ intervallum esetén létezik olyan befoglaló függvény, amelyre a Newton lépés nem eredményes abban az értelemben, hogy vagy nem is csökken az intervallum mérete, vagy sok darabra osztja fel az X intervallumot.*

Bizonyítás. Az egydimenziós $f(x)$ függvény esetén a Newton iteráció az alábbi összefüggésekkel írható le:

$$N(X^{(k)}, \widetilde{x}^{(k)}) = \widetilde{x}^{(k)} - \frac{f'(\widetilde{x}^{(k)})}{F''(X^{(k)})}, \tag{2}$$

$$X^{(k+1)} = X^{(k)} \cap N(X^{(k)}, \widetilde{x}^{(k)}),\ k = 1, 2, \ldots \tag{3}$$

ahol $\widetilde{x}^{(k)} = \mathrm{mid}(X^{(k)})$, f' az elsőrendű derivált, F'' a másodrendű derivált befoglalása.

Legyen egy, a $w(X) < \epsilon$ feltételnek megfelelő $X = [a, b]$ intervallum és $F''(X) = [c, d]$. A Newton lépés akkor nem eredményes, ha a (2) egyenletben a másodrendű derivált tartalmazza a nullát, és a metszetképzés (3) után eredményként két intervallumot kapunk. A Newton lépés ezen eredménye azért nem hasznos, mert ezt elérhetjük egy egyszerű kettéosztással is, amely olcsóbb művelet, mint a Newton lépés. A kérdés az, hogy milyen túlbecslésre van szükség ahhoz, hogy a másodrendű derivált befoglalása tartalmazza a nullát és a metszetképzés után két intervallumot kapjunk eredményül.

A továbbiakban feltételezzük, hogy $0 \notin F''(X)$ és $F''_{uj}(X) = [c', d']$. A feladat c' és d' értékeit úgy meghatározni, hogy $0 \in F''_{uj}(X)$ teljesüljön.

A korábbi jelölésekkel a Newton operátor a következő alakú lesz:

$$N(X, \widetilde{x}) = \widetilde{x} - \frac{f'(\widetilde{x})}{F''_{uj}(X)} = \widetilde{x} - \frac{f'(\widetilde{x})}{[c', d']}. \tag{4}$$

1. Eset: amikor $f'(\widetilde{x}) > 0$. Ekkor $N(X, \widetilde{x}) = \left(-\infty, \widetilde{x} - \frac{f'(\widetilde{x})}{d'}\right] \cup \left[\widetilde{x} - \frac{f'(\widetilde{x})}{c'}, \infty\right)$. Az $X \cap N(X, \widetilde{x})$ művelet eredményeként akkor kapunk két intervallumot, ha teljesül

$$a < \widetilde{x} - \frac{f'(\widetilde{x})}{d'}, \tag{5}$$

$$b > \widetilde{x} - \frac{f'(\widetilde{x})}{c'}. \tag{6}$$

Az (5) és (6) egyenlőtlenségekből rendre azt kapjuk, hogy:

$$d' > 2 \cdot \frac{f'(\widetilde{x})}{w(X)}, \tag{7}$$

$$c' < -2 \cdot \frac{f'(\widetilde{x})}{w(X)}. \tag{8}$$

A (7) és (8) valamint az (1) összefüggésekből következik:

$$c - D \cdot w(F''(X)) < -2 \cdot \frac{f'(\widetilde{x})}{w(X)}, \tag{9}$$

$$d + D \cdot w(F''(X)) > 2 \cdot \frac{f'(\widetilde{x})}{w(X)}. \tag{10}$$

Innen D-re a következő egyenlőtlenségek adódnak:

$$D > D_1 = \frac{2 \cdot f'(\widetilde{x}) + c \cdot w(X)}{w(X) \cdot w(F''(X))}, \tag{11}$$

$$D > D_2 = \frac{2 \cdot f'(\widetilde{x}) - d \cdot w(X)}{w(X) \cdot w(F''(X))}. \tag{12}$$

Tehát, ha $D > \max(D_1, D_2)$, akkor a Newton lépés eredményeként két intervallumot kapunk.

2. Eset: amikor $f'(\widetilde{x}) < 0$. Ekkor $N(X, \widetilde{x}) = \left(-\infty, \widetilde{x} - \frac{f'(\widetilde{x})}{c'}\right] \cup \left[\widetilde{x} - \frac{f'(\widetilde{x})}{d'}, \infty\right)$.

Az első esethez hasonló gondolatmenettel azt kapjuk D-re, hogy:

$$D > D_1 = \frac{-2 \cdot f'(\widetilde{x}) + c \cdot w(X)}{w(X) \cdot w(F''(X))}, \tag{13}$$

$$D > D_2 = \frac{-2 \cdot f'(\widetilde{x}) - d \cdot w(X)}{w(X) \cdot w(F''(X))}. \tag{14}$$

Tehát ebben az esetben ha $D > \max(D_1, D_2)$, akkor a Newton lépés nem lesz eredményes.

□

Az előbbi bizonyításban a D_1 és D_2 értékek segítségével olyan befoglalását kapjuk a másodrendű deriváltnak, amelyre a Newton módszer két intervallumot ad eredményül. Mivel szimmetrikus túlbecslést feltételeztünk a másodrendű derivált befoglalására, ezért D értékéül a D_1 és D_2 közül nyilván a nagyobbikat választjuk.

Az 1. Esetnek megfelelően, ha most azt vizsgáljuk, hogy a Newton lépés mikor hatékony, akkor a következő összefüggést kapjuk:

$$D \leq D_1 = \frac{2 \cdot f'(\widetilde{x}) + c \cdot w(X)}{w(X) \cdot w(F''(X))}. \tag{15}$$

Ezen összefüggés alapján ismert D túlbecslési paraméter mellett megadható olyan ϵ küszöbérték a Newton lépés bekapcsolására, amely biztosítani tudja, hogy a Newton lépés hatékony legyen abban az értelemben, hogy eredményül egy szűkebb intervallumot kapjunk, mint az argumentum intervallum. Az (15) összefüggés alapján

$$w(X) \leq \frac{2 \cdot f'(\widetilde{x})}{D \cdot w(F''(X)) - c}.$$

Tehát, ha

$$\epsilon = \frac{2 \cdot f'(\widetilde{x})}{D \cdot w(F''(X)) - c},$$

akkor a Newton lépés eredményeként szűkebb intervallumot kapunk, mint az argumentum intervallum.

Hasonló állítás vezethető le arra az esetre is, ha $f'(\widetilde{x}) < 0$ (ez a 2. Eset az 1. Tétel bizonyításában).

Egy $F(X)$ befoglaló függvényt akkor mondunk izotonnak, ha $X \subseteq Y$-ból $F(X) \subseteq F(Y)$ következik. Ez a tulajdonsága csaknem minden, a számítógépes eljárásokban használatos befoglalófüggvénynek megvan. Az (1)-ben definiált F''_{uj} függvény nyilvánvalóan minden rögzített D értékre izoton függvény. A következő állítás mégis azt mondja ki, hogy amennyiben a D konstans értékét úgy kell megválasztani, hogy a Newton lépés ne legyen eredményes, akkor az így adódó befoglaló függvény nem lesz izoton.

1. Állítás. *Az 1. Tétel bizonyításában megkonstruált F''_{uj} befoglalófüggvény nem izoton.*

Bizonyítás. Ennek belátására elegendő ellenpéldát mutatni.

Az $f(x) = \frac{1}{20}(x+4)(x+2)(x+1)(x-1)(x-3)+2$ függvény esetén az (1) összefüggéssel és az 1. Tétel bizonyításában kidolgozott konstrukcióval megadott befoglaló függvény nem izoton az alábbi intervallumokra.

Ha $X_1 = [-1.60, -1.5]$ akkor $D > 3.7569$. Ha $D = 3.76$ akkor $F''_{uj}(X_1) = [-1.41, 7.46]$.
Ha $X_2 = [-1.65, -1.45]$ akkor $D > 1.3002$. Ha $D = 1.31$ akkor $F''_{uj}(X_2) = [-0.70, 6.81]$.
Ha $X_3 = [-1.7, -1.4]$ akkor $D > 0.6353$. Ha $D = 0.64$ akkor $F''_{uj}(X_3) = [-0.46, 6.67]$.

A példa esetén tehát $X_1 \subset X_2 \subset X_3$ és $F''_{uj}(X_3) \subset F''_{uj}(X_2) \subset F''_{uj}(X_1)$, azaz az előírt tulajdonsággal rendelkező F''_{uj} nem izoton.

□

Az 1. Állítás lényegi jelentése tehát az, hogy elegendően kicsi ϵ döntési paraméter választása esetén a Newton lépés eredményessége esélye növelhető.

2. Tétel. *Adott f többváltozós függvény és X, $w(X) < \epsilon$ intervallum esetén létezik olyan befoglaló függvény, amelyre a Newton lépés nem lesz hatékony, abban az értelemben, hogy vagy nem is csökken az intervallum mérete, vagy sok darabra osztja fel az X intervallumot.*

Bizonyítás. Többváltozós esetben a Newton módszer iterációs sémája a következő formában írható:

$$G(\widetilde{x}^{(k)}) + H(X^{(k)})(N(X^{(k)}, \widetilde{x}^{(k)}) - \widetilde{x}^{(k)}) = 0 \tag{16}$$

$$X^{(k+1)} = X^{(k)} \cap N(X^{(k)}, \widetilde{x}^{(k)}),\ k = 1, 2, \ldots, \tag{17}$$

ahol $\widetilde{x}^{(k)} = \mathrm{mid}(X^{(k)})$, G a gradiens, és H a Hesse mátrix befoglalása.

Az N érték meghatározására a Hansen-Sengupta operátort használjuk. Ez az operátor a Gauss-Seidel eljárást alkalmazza. A (16) egyenletet $H(X)$ középponti inverzével, C-vel prekondicionálva kapjuk, hogy

$$C \cdot H(X)(N(X, \widetilde{x}) - \widetilde{x}) = -C \cdot G(\widetilde{x}).$$

Bevezetve az

$$M = C \cdot H(X), \quad b = C \cdot G(\widetilde{x})$$

mennyiségeket, a komponensenkénti intervallumos Gauss-Seidel eljárás

$$N(X^{(k)}, \widetilde{x}^{(k)}) = \widetilde{x}_i^{(k)} - \frac{b_i + \sum_{j=1}^{i-1} M_{ij}(X_j^{(k+1)} - \widetilde{x}_j^{(k+1)}) + \sum_{j=i+1}^{n} M_{ij}(X_j^{(k)} - \widetilde{x}_j^{(k)})}{M_{ii}}, \tag{18}$$

$$X_i^{(k+1)} = X_i^{(k)} \cap N(X^{(k)}, \widetilde{x}^{(k)}) \tag{19}$$

alakú lesz, ahol $\widetilde{x}^{(k)} = \mathrm{mid}(X^{(k)})$ és $k = 1, 2, \ldots$. Az iterációban, ha az $N(X^{(k)}, \widetilde{x}^{(k)})$ i-dik komponensét kiszámoltuk, akkor elvégezzük a metszetképzést.

A (18) összefüggésből látszik, hogy minden iterációs lépésben az M mátrix egy főátlóbeli elemével osztunk. Az egyváltozós esettől eltérően, itt nem a másodrendű derivált befoglalását használjuk, hanem a Hesse mátrix egy prekondicionált változatát. A prekondicionálás célja, hogy az $M \cdot H(X)$ szorzás eredményeképpen olyan mátrixot kapjunk, amelynek együtthatói jobban kezelhetők a rendszer megoldása során, mint az eredeti mátrix esetén. A (18) egyenlet hasonló a (2) egyenlethez, így a bizonyítás is hasonlóan történik, mint az egyváltozós esetben. A különbség itt az, hogy úgy kell meghatározni a befoglalást, hogy a Hesse mátrix középponti inverzével való szorzás után tartalmazza a nullát, és a metszetképzés során két intervallumot kapjunk.

Az adott iteráció i. belső lépésében, ha $0 \in M_{ii}$, $i = 1, \ldots, n$ és a Newton lépés eredményeként az i. komponens két intervallum egyesítéseként írható fel, akkor a Newton lépés két külön részfeladatra osztható. Ezekben a részfeladatokban a kapott i. komponenst helyettesítjük az egyesítésben szereplő egyik intervallummal. Tehát, ha egy komponens esetén igaz a fenti állítás, akkor a Newton lépés már nem lesz eredményes. □

A teljes elméleti gondolatmenet hasonló formában megismételhető olyan befoglaló függvényekre is, amelyek túlbecslése nem szimmetrikus.

5.2. Numerikus vizsgálat

Az INTLAB alapú algoritmust teszteltük az új feltétellel és a kapott eredményeket összehasonlítottuk az eredeti feltétellel elért eredményekkel. A tesztkörnyezet beállításai megegyeznek a korábban leírt beállításokkal. A 3. és 4. Táblázatok tartalmazzák a függvényekre az eredményeket, valamint az egyes mutatók összesített és átlag értékeit a teljes célfüggvény halmazra vonatkozóan. A két feltételt a kapott hatékonysági mutatókkal nehéz összehasonlítani, ugyanis ezek értékei lényegesen, és persze érthető módon különböznek az új feltételnek köszönhetően. Ezért célszerűbb a két feltételnek megfelelő CPU időket összehasonlítani a teljes függvény halmazra.

A teljes futásidőt tekintve megállapítható, hogy az új feltétel esetén ez 14 perccel (15%-al) kevesebb, mint a régi esetén. Figyelembe véve azt, hogy az EX2 és a Schw3.7.10 függvények

együttesen a teljes futásidő 80 százalékát igénylik, érdemes megvizsgálni az eredményeket e két feladat elhagyásával. Ebben az esetben azt találjuk, hogy az új feltétel esetén a teljes futásidő 6 perccel (30%-al) lett kevesebb, mint a régi feltétellel. Az eredmények alapján elmondható, hogy az új feltétel segítségével sikerült a szükséges számítási időt lényegesen csökkenteni. Ez a javulás mégis nem minden függvényen figyelhető meg, sőt egyes estekben (pl. Schw3.7.10) rosszabb értéket kaptunk. Ez azt jelenti, hogy nehéz az új feltétel számára olyan értéket beállítani, amely minden vizsgált feladatra javulást eredményez. A jövőben ennek az értéknek az adaptív beállítását fogjuk megvizsgálni.

6. Alkalmazás

6.1. Lokalizálás szenzorhálózatokban

Szenzorhálózati alkalmazásokban nagyon gyakran szükség van az egyes csomópontok földrajzi pozícióinak az ismeretére. Ezért az egyik legfontosabb elvárás az ilyen hálózatokban a csomópontok helymeghatározó képessége. Az elmúlt időszakban számos módszert [1, 4, 13, 17] dolgoztak ki szenzorok lokalizálására.

Az általános szenzor lokalizálási feladat a következőképpen fogalmazható meg. Adottak m darab szenzor csomópont ismert pozíciókkal: $a_k \in \mathbb{R}^l$, $k = 1, \ldots, m$, valamint n darab csomópont, ismeretlen pozíciókkal $x_j \in \mathbb{R}^l$, $j = 1, \ldots, n$. Két tetszőleges csomópont esetén bevezetjük a következő Euklideszi távolságokat: $d_{kj} = ||a_k - x_j||$, egy ismert és egy ismeretlen pozíciójú csomópont esetén, valamint $d_{ij} = ||x_i - x_j||$, két ismeretlen pozíciójú csomópont esetén, ahol $j = 1, \ldots, n$ és $i \neq j$.

Távolságalapú alkalmazásoknál a csomópontok jeleket bocsátanak ki, amelyek segítségével a szomszédos csomópontok képesek az egymás közötti távolságokat becsülni. Egy csomópont szomszédai azon csomópontok, amelyek az adott csomópont hatókörén belől vannak. Az összes rögzített és nem rögzített csomópontokra értelmezzük a következő halmazokat:

$$N_k = \{(k, j) : d_{kj} \leq r_k\},\ j = 1, \ldots, n,$$
$$N_i = \{(i, j) : d_{ij} \leq r_i\},\ j = 1, \ldots, n,$$

ahol az r_k és r_i paraméterek a maximális hatókörök sugarai.

A d_{kj} és d_{ij} valós távolságok helyett általában a csomópontok által mért $\widetilde{d}_{kj}$, $\widetilde{d}_{ij}$ zajos távolságokat szokták használni. Az utóbbiak a valós távolságoknak egy zajjal módosított változatai.

Tehát a lokalizálási feladat: adottak a $\widetilde{d}_{kj}$, $\widetilde{d}_{ij}$ zajos távolságok, és az ismert csomópontok koordinátái $a_k \in \mathbb{R}^l$, $k = 1, \ldots, m$ felhasználásával határozzuk meg a többi csomópont $x_j \in \mathbb{R}^l$, $j = 1, \ldots, n$ pozícióit. A feladat megfogalmazható szemidefinit programozásként [4], vagy nemlineáris optimalizálási feladatként [13]. Mi az utóbbi lehetőséget választottuk, amelyben a cél egy nemlineáris hibafüggvény minimalizálása:

$$\min_{\widehat{x}} \left\{ \sum_{k=1}^{m} \sum_{j \in N_k} \left(||a_k - \widehat{x}_j|| - \widetilde{d}_{kj} \right)^2 + \sum_{i=1}^{n} \sum_{j \in N_i} \left(||\widehat{x}_i - \widehat{x}_j|| - \widetilde{d}_{ij} \right)^2 \right\}, \tag{20}$$

ahol $\widehat{x}_i$, $\widehat{x}_j$ az i és j csomópontok becsült pozíciói, $\widetilde{d}_{kj}$ és $\widetilde{d}_{ij}$ a mért távolságok a (k, j) és (i, j) csomópont párok között és N_i, N_k a szomszédos csomópont halmazok.

6.2. A feladat megoldása

A korábban megfogalmazott lokalizálási problémát számos sztochasztikus módszerrel sikerült közelítőleg megoldani. A leggyakrabban használt módszer a szimulált hűtés [1, 13, 17] és a genetikus algoritmus [23]. A (20) célfüggvénnyel megadott feladat egy globális optimalizálási feladat, amelyet mi intervallum aritmetikán alapuló módszerrel próbáltunk megoldani. A feladatban minden ismeretlen pozíciójú csomópont mindkét koordinátájához egy intervallum

3. Táblázat. A Newton lépés bekapcsolásának vizsgálata az INTLAB alapú algoritmus esetén. A táblázatban Dim a feladat dimenzióját jelenti, ITSz az iteráció számot, FHSz a függvényhívások számát, GHSz pedig a gradiens hívások számát.

		Eredeti feltétel			Új feltétel		
Feladat	Dim	ITSz	FHSz	GHSz	ITSz	FHSz	GHSz
S5	4	16	126	86	22	117	76
S7	4	17	121	78	22	120	76
S10	4	17	123	78	22	122	76
H3	3	23	147	99	14	82	51
H6	6	191	1 505	1 167	112	560	363
GP	2	76	458	229	53	717	415
SHCB	2	17	103	60	16	105	63
THCB	2	44	274	189	59	284	187
BR	2	44	250	177	71	360	256
RB	2	38	238	151	17	174	117
RB5	5	396	3 660	2 758	608	3 511	2 568
L3	2	47	293	170	32	191	103
L5	2	86	593	406	22	131	73
L8	3	11	80	55	20	98	67
L9	4	13	107	73	26	129	85
L10	5	15	125	86	33	161	106
L11	8	23	189	128	52	253	163
L12	10	30	254	175	65	315	202
L13	2	10	74	47	13	76	49
L14	3	15	120	77	22	121	76
L15	4	18	146	94	28	150	94
L16	5	19	142	88	29	162	97
L18	7	27	206	130	41	226	136
Schw2.1	2	113	804	580	168	758	557
Schw3.1	3	14	96	64	21	122	81
Schw2.5	2	50	293	205	34	161	114
Schw2.14	4	356	3 242	2 337	527	5 914	4 160
Schw2.18	2	3	21	13	19	95	63
Schw3.2	3	20	144	98	25	149	99
Schw3.7_5	5	45	309	208	108	517	364
Schw3.7_10	10	696	4 371	2 665	5 232	22 065	15 781
Griew5	5	25	190	117	53	263	163
Griew7	7	40	297	173	73	363	223
R4	2	35	210	125	21	134	80
R5	3	107	996	748	181	760	544
R6	5	140	1 516	1 221	339	1 409	1 018
R7	7	204	2 728	2 293	500	2 086	1 501
R8	9	320	4 881	4 201	651	2 713	1 954
EX2	5	9 279	59 605	44 126	5 774	42 338	32 161
Összeg		12 640	89 037	65 775	15 125	88 012	64 362
Átlag		324	2 283	1 687	388	2 257	1 650

4. Táblázat. A Newton lépés bekapcsolásának vizsgálata az INTLAB alapú algoritmus esetén. A táblázatban Dim a feladat dimenzióját jelenti, HHSz a Hesse mátrix kiértékelések számát, MLH a maximális lista hosszát, valamint CPU a futási időt másodpercben.

		Eredeti feltétel			Új feltétel		
Feladat	Dim	HHSz	MLH	CPU	HHSz	MLH	CPU
S5	4	7	10	5.66	3	10	5.02
S7	4	6	14	7.27	3	14	7.00
S10	4	6	17	10.36	3	17	9.95
H3	3	11	16	5.09	3	13	2.69
H6	6	86	64	97.45	10	55	32.77
GP	2	0	153	9.25	56	175	16.63
SHCB	2	3	22	1.70	6	19	1.80
THCB	2	21	24	3.75	3	26	3.59
BR	2	18	10	3.44	18	12	4.91
RB	2	11	11	1.59	19	9	1.25
RB5	5	317	79	93.13	162	79	83.20
L3	2	8	57	10.06	2	53	6.22
L5	2	31	32	26.25	2	32	5.25
L8	3	5	9	2.34	2	9	2.72
L9	4	7	13	4.08	2	14	4.67
L10	5	8	15	5.95	2	17	7.13
L11	8	9	28	14.11	2	30	17.42
L12	10	11	36	23.89	2	39	27.03
L13	2	4	9	1.45	3	9	1.47
L14	3	7	12	3.16	3	12	3.08
L15	4	8	19	4.84	3	18	4.73
L16	5	6	20	5.56	3	20	6.03
L18	7	8	26	10.80	4	26	11.31
Schw2.1	2	53	25	12.50	17	31	11.36
Schw3.1	3	5	6	1.58	7	6	1.95
Schw2.5	2	27	4	2.13	5	7	1.13
Schw2.14	4	216	123	47.98	455	442	88.22
Schw2.18	2	1	4	0.16	2	11	0.64
Schw3.2	3	11	7	1.66	9	8	1.66
Schw3.7_5	5	24	32	7.13	13	32	10.52
Schw3.7_10	10	192	818	183.59	28	1 024	873.70
Griew5	5	7	28	5.94	1	28	7.95
Griew7	7	8	58	12.48	1	51	15.02
R4	2	6	36	2.23	6	24	1.44
R5	3	71	57	27.20	0	23	18.97
R6	5	100	30	71.72	0	25	59.06
R7	7	168	41	184.78	0	47	122.38
R8	9	270	59	429.48	0	65	204.88
EX2	5	3 802	388	4 390.09	4 284	145	3 182.41
Összeg		6 777	2 600	5 732	5 111	2 677	4 867
Átlag		174	67	147	132	69	125

változót rendeltünk. A csomópontok a sík $[0,1] \times [0,1]$ tartományában helyezkednek el, így az intervallum változók is ezen a tartományon belül változnak.

A (20) célfüggvény minimumának a megkeresése az INTLAB alapú optimalizálóval nagyon időigényes művelet, ezért egy közelítő megoldás megtalálására az ívmetszés [17, 21] technikáját alkalmazzuk használva az INTLAB alapú programot. Az ívmetszés lényege, hogy ha egy ismeretlen pozíciójú csomópontnak van három ismert helyzetű szomszédja, akkor az előbbi helyzete egyértelműen meghatározható, amennyiben ismertek a csomópontok közötti valós távolságok. Mivel a távolságok zajosak, ezért általában nem létezik egyértelmű megoldás, így azt a pontot keressük meg, amely minimalizálja az ismert helyzetű csomópontoktól vett távolságok összegét. A feladat megoldása során a csomópontokat két halmazba csoportosítjuk: az ismert és az ismeretlen helyzetű csomópontok halmazába. Minden ismeretlen helyzetű csomópont esetén meghatározzuk a szomszédait, majd az ívmetszés segítségével kiszámoljuk a pozícióját. A meghatározott helyzetű csomópontot áthelyezzük az ismert pozíciójú csomópontok halmazába és felhasználjuk más ismeretlen helyzetű csomópont helyzetének meghatározására. A mi esetünkben, ahol lehetséges, ott négy szomszédos csomópontot választunk a pontosabb meghatározás érdekében.

A lokalizálási feladat megoldása nagymértékben függ a különböző csomópontok számától, a rögzített szenzorok pozíciójától, a szomszédos csomópontok számától, a mért távolságok hibájától, és a hatókör sugarától. Az általunk vizsgált feladatokban 5 ismert, valamint 50 ismeretlen pozíciójú csomópontot választottunk véletlenszerű elhelyezéssel a $[0,1] \times [0,1]$ tartományban. Valamennyi csomópont esetén a hatókör sugara $r = 0.3$. A szimuláció elvégzésére szükség van a szomszédos csomópontok közötti mért távolságokra, amelyek hibáját normál eloszlás segítségével modellezzük. Hasonló eloszlást feltételeztek az [1, 17] cikkekben is. Tehát a mért távolságok és a valós távolságok közötti összefüggések:

$$\widetilde{d}_{kj} = d_{kj}(1 + randn() \cdot nf),$$

$$\widetilde{d}_{ij} = d_{ij}(1 + randn() \cdot nf),$$

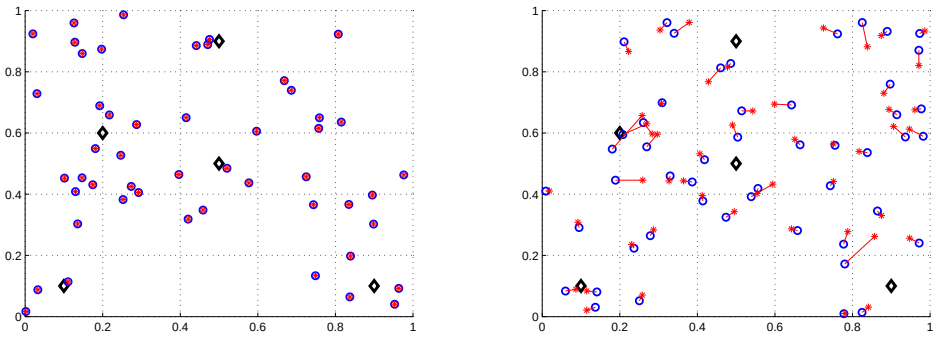
ahol $randn()$ a normál eloszlás alapján generált szám, nf pedig a hiba faktor, amelynek értéke 10%.

Az előbbi beállítások mellett a lokalizálási feladatot az INTLAB alapú globális optimalizáló eljárás segítségével oldottuk meg. Az eredmény intervallumra 10^{-3} pontosságot követeltünk meg, és ezen intervallum középpontját tekintettük a keresett szenzorok becsült pozíciójának. Abban az esetben, ha nincs zaj a mért távolságokban, a becsült pozíciók megegyeznek az eredeti pozíciókkal (lásd 1(a) ábra). Zajos távolságok esetén pedig a közelítő megoldások az 1(b) ábrán láthatók. Az algoritmus teljes futási ideje a zajos távolságok esetén 261 másodperc. Egy csomópont esetén az átlagos függvény-, gradiens-, illetve Hesse mátrix kiértékelések száma rendre 194, 139 és 10. A maximális munkalista hossza 10 volt. Az ábrákon látható a csomópontok eredeti pozíciója (karika), a becsült pozíciók (csillag), az eredeti pozíció és a becsült pozíció közötti távolság (vonal) valamint az ismert pozíciójú csomópontok (négyszög). A kapott közelítő megoldások javítását valamint a feladat részletesebb tanulmányozását az X-CSC alapú intervallumos globális optimalizálóval tervezzük.

7. Következtetések

A cikkben egy intervallumos globális optimalizáló algoritmus új implementációját mutattuk be, és teszteltük azt MATLAB/INTLAB környezetben. Az új programot összehasonlítottuk a hasonló C-XSC alapú eljárással. A teszt eredménye azt mutatta, hogy az új módszer hasonlóan hatékony, mint az előbbi – eltekintve a CPU időtől.

Megvizsgáltunk továbbá egy új feltételt a Newton lépés bekapcsolására. Az eredmények alapján ez az új feltétel csökkenti a teljes szükséges számítási időt a régi megoldáshoz viszonyítva. Az INTLAB alapú algoritmus segítségével sikerült jó közelítő megoldást találni a szenzor lokalizálási problémára.



(a) Zaj nélküli távolságok (b) Zajos távolságok

1. ábra. A szenzor lokalizációs feladat közelítő megoldása zaj nélküli és zajos távolságok esetén

Köszönetnyilvánítás

Az elvégzett kutatást részben a Nemzeti Fejlesztési Ügynökség TÁMOP-4.2.2/08/1/2008-0008 és TÁMOP-4.2.1/B-09/1/KONV-2010-0005 pályázatai, valamint az MTA Határon Túli Magyar Tudományos Ösztöndíjprogramja támogatták.

Hivatkozások

[1] Anderson, B.D.O., Mao, G., and Fidan, B.: *Wireless sensor network localization techniques.* Computer Networks 51, (2007) 2529–2553.

[2] Balogh, J., Csendes, T., and Stateva, R.P.: *Application of a stochastic method to the solution of the phase stability problem: cubic equations of state.* Fluid Phase Equilibria 212, (2003) 257–267.

[3] Bánhelyi, B., Csendes, T., and Garay, B.M.: *Optimization and the Miranda approach in detecting horseshoe-type chaos by computer.* Int. J. Bifurcation and Chaos 17, (2007) 735–747.

[4] Biswas, P. and Ye, Y.: *"Semidefinite programming for ad hoc wireless sensor network localization"*, in: Proceedings of the 3-rd International Symposium on Information Processing in Sensor Networks, Berkeley, CA, USA, ACM Press, New York, (2004) 46–54.

[5] Casado, L.G., García, I., and Csendes, T.: *A heuristic rejection criterion in interval global optimization algorithms.* BIT 41, (2001) 683–692.

[6] Casado, L.G., García, I., Csendes, T., and Ruiz, V.G.: *Heuristic Rejection in Interval Global Optimization.* J. Optimization Theory and Applications 118, (2003) 27–43.

[7] Csendes, T.: *New subinterval selection criteria for interval global optimization.* J. Global Optimization 19, (2001) 307–327.

[8] Csendes, T.: *Numerical experiences with a new generalized subinterval selection criterion for interval global optimization.* Reliable Computing 9, (2003) 109–125.

[9] Csendes, T., Bánhelyi, B., and Hatvani, L.: *Towards a computer-assisted proof for chaos in a forced damped pendulum equation.* J. Computational and Applied Mathematics 199, (2007) 378–383.

[10] Csendes, T., Garay, B.M., and Bánhelyi, B.: *A verified optimization technique to locate chaotic regions of Hénon systems.* J. of Global Optimization 35, (2006) 145–160.

[11] Csendes, T., Pál, L., Sendín, J.O.H., and Banga, J.R.: *The GLOBAL Optimization Method Revisited.* Optimization Letters 2, (2008) 445–454.

[12] Hammer, R., Hocks, M., Kulisch, U., and Ratz, D.: *Numerical Toolbox for Verified Computing I.* (Springer-Verlag, Berlin, 1993).

[13] Kannan, A.A., Mao, G., and Vucetic, B.: *Simulated Annealing based Wireless Sensor Network Localization*, Journal of Computers 1, (2006) 15–22.

[14] Kearfott, R.B.: *Rigorous global search: continuous problems* (Kluwer, Dordrecht, 1996).

[15] Markót, M.C. and Csendes, T.: *A new verified optimization technique for the "packing circles in a unit square" problems.* SIAM J. on Optimization 16, (2005) 193–219.

[16] Markót, M.C., Fernández, J., Casado, L.G., and Csendes, T.: *New interval methods for constrained global optimization.* Mathematical Programming 106, (2006) 287–318.

[17] Niewiadomska-Szynkiewicz, E. and Marks, M.: *Optimization Schemes For Wireless Sensor Network Localization.* International Journal of Applied Mathematics and Computer Science, 19, (2009) 291–302.

[18] Pál, L. and Csendes, T.: *INTLAB implementation of an interval global optimization algorithm.* Optimization Methods and Software 24, (2009) 749–759.

[19] Rump, S.M.: *INTLAB – Interval Laboratory.* In: T. Csendes (ed.): Developments in Reliable Computing, Kluwer, Dordrecht, (1999) 77–104.

[20] Szabó, P.G., Markót, M.C., Csendes, T., Specht, E., Casado, L.G., and García, I.: *New Approaches to Circle Packing in a Square – With Program Codes* (Springer-Verlag, Berlin, 2007).

[21] Takács, G.: *Helymeghatározás mobiltelefonnal és mobil hálózattal.* Híradástechnika 8, (2008) 20–27.

[22] Tóth, B., Fernández, J., and Csendes, T.: *Empirical convergence speed of inclusion functions for facility location problems.* J. of Computational and Applied Mathematics 199, (2007) 384–389.

[23] Zhang, Q., Wang, J., Jin, C., Ye, J., Ma, C., and Zhang, W.: *"Genetic Algorithm Based Wireless Sensor Network Localization"*, in: Proceedings of the 2008 Fourth International Conference on Natural Computation, (2008) 608–613.

Pál László
Sapientia Erdélyi Magyar Tudományegyetem
Gazdaság- és Humántudományok Kar
530104, Csíkszereda, Szabadság tér 1, Románia

Csendes Tibor
Szegedi Tudományegyetem
Informatikai Intézet
6720 Szeged, Árpád tér 2.