Egy hatékony számítógépes eszköz matematikai problémák megoldására

Tóth Boglárka és Vinkó Tamás

1. Bevezető

Tegyük fel, hogy a következő feladatot kaptuk. Számítsuk ki két egyváltozós polinom legnagyobb közös osztóját: azt a legmagasabb fokú polinomot, amely maradék nélkül osztja a megadott két polinomot. Abban az esetben, ha a polinomok együtthatói pontosan ismertek, a jól ismert Euklideszi algoritmus gyorsan megadja a választ. Az algoritmus programozása is meglehetősen egyszerű, de használhatjuk valamely jól bevált számítógép-algebrai rendszert is, például a Maple-t.

Ha azonban a két polinom együtthatói valamilyen mérési eredményből származnak, akkor azok nem mindig egészek. Tekintsük például a következő két polinomot [4]:

$$\begin{aligned} p(x) &= x^5 + 5.503x^4 + 9.765x^3 + 7.647x^2 + 2.762x + 0.37725, \\ q(x) &= x^4 - 2.993x^3 - 0.7745x^2 + 2.007x + 0.7605. \end{aligned}$$

Erre a két polinomra az Euklideszi algoritmus azt az eredményt adja, hogy a polinomok relatív prímek, azaz a legnagyobb közös osztójuk 1.

A kapott eredmény pontos, de használhatatlan. Ilyen esetekben szerencsésebb azt a lehető legmagasabb fokú polinomot keresni, amely a $p(x)$ és $q(x)$ polinomokhoz "legközelebbi" polinompár legnagyobb közös osztója. Ezt az ötletet alkalmazva kapjuk a polinomok nem pontos legnagyobb közös osztóját, amely a gyakorlati alkalmazások során kitűnően használható. A probléma megfogalmazható optimalizálási feladatként [2, 3]. Legyenek adottak a $p(x)$ n-ed fokú és $q(x)$ m-ed fokú egyváltozós polinomok úgy, hogy együtthatóik értékeit nem ismerjük pontosan. Akkor egy $\varepsilon \geq 0$-ra a $p(x)$ és $q(x)$ lnko-ja az a lehető legnagyobb fokú $d(x)$ polinom, amelyre $d(x)$ pontosan osztja $p(x) + \Delta p(x)$-et és $q(x) + \Delta q(x)$-et egyaránt valamely Δp-re és Δq-ra, melyre $\|\Delta \mathbf{p}\| \leq \varepsilon$ és $\|\Delta \mathbf{q}\| \leq \varepsilon$, ahol $\mathbf{p}$ jelöli a $p(x)$ polinom $\mathbf{q}$ pedig a $q(x)$ polinom együtthatóvektorát. Ezek alapján az optimalizálási feladatunk célfüggvényét

$$\min f(\mathbf{d}, \mathbf{u}, \mathbf{v}) = \|\Delta \mathbf{p}\| + \|\Delta \mathbf{q}\|$$

alakban írhatjuk kiegészítve a

$$d(x)u(x) = p(x) + \Delta p(x)$$

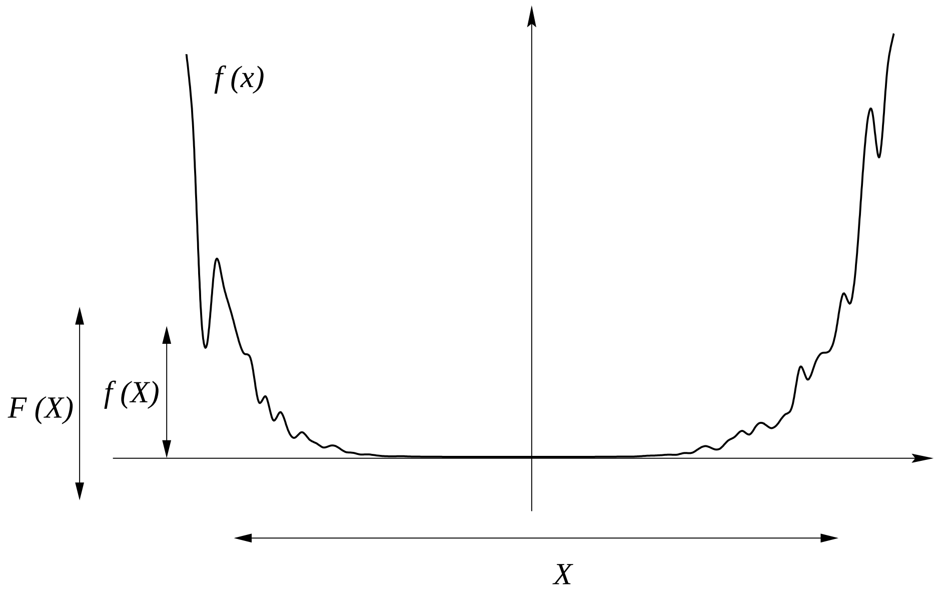
$$d(x)v(x) = q(x) + \Delta q(x)$$

korlátozó feltételekkel. A célfüggvényben használt $\|.\|$ norma lehet a jól ismert 2-es norma, de más normák használatával jobb közelítéseket érhetünk el.

Erre a feladatra és más hasonló problémákra az alábbiakban ismertetésre kerülő intervallum-aritmetikát és más kifinomult számítógépes technikát használó GlobSol nevű optimalizáló és nemlineáris egyenletrendszer megoldó programmal adhatunk megbízható megoldást. Így a fent megadott problémára a program az

$$x^2 + [1.0069986925, 1.0069986978]\, x + [0.253494953, 0.253494961]$$

polinomot adja megoldásként, amelyre $\|\Delta p(x)\|^2 + \|\Delta q(x)\|^2 \approx 8 \times 10^{-7}$. Mint láthatjuk, a program az együtthatókat intervallumokként adja meg, amelyek az alkalmazott módszerek miatt garantáltan tartalmazzák a helyes értéket.



1. ábra. A befoglaló függvény elvének bemutatása az $f(x) = x^6(\sin\frac{1}{x} + 2)$ függvényen.

2. Alapfogalmak

Ebben a fejezetben először alapvető fogalmakat fogunk bevezetni, valamint megemlítjük a globális optimalizálás néhány általános problémáját. Röviden tárgyaljuk az ismertetésre kerülő program által használt eljárásokat és technikákat. Az itt felsoroltak csak egy részét képezik a GlobSol-ban megvalósított eljárásoknak, az érdeklődő olvasónak mindenekelőtt a [11] könyvet ajánljuk, ahol további részleteket és hivatkozásokat találhat.

2.1. Intervallumos globális optimalizálás és a GlobSol

Az ismertetésre kerülő program segítségével többek között olyan optimalizálási problémákat oldhatunk meg, amelyek az ún. folytonos problémák közé tartoznak. Ez azt jelenti, hogy az optimalizálandó függvényünk folytonos, de nem feltétlen differenciálható. A lehetséges megoldásokat egy többdimenziós valós tér S korlátos és zárt (kompakt) részhalmazában keressük. Ezt a következőképpen fogalmazhatjuk meg. Legyen $f : S \subset \mathbb{R}^n \to \mathbb{R}$ folytonos célfüggvény. A

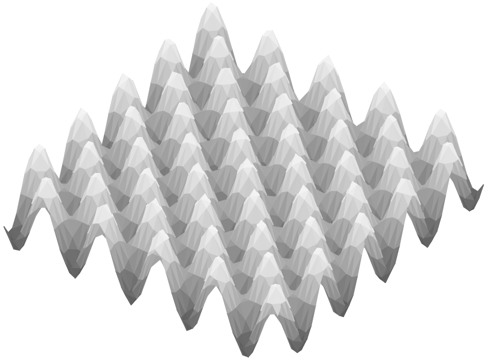
$$\min_{x \in S} f(x)$$

globális optimalizálási problémát vizsgáljuk, esetleg lineáris vagy nemlineáris korlátozó feltételekkel kiegészítve, amelyek a lehetséges megoldások halmazát szűkítik. A minimalizálási feladatokon kívül a program segítségével $Ax = b$ alakú lineáris- vagy $g(x) = 0$, ahol $g : \mathbb{R}^n \to \mathbb{R}^n$ alakú, nemlineáris egyenletrendszereket is megoldhatunk.

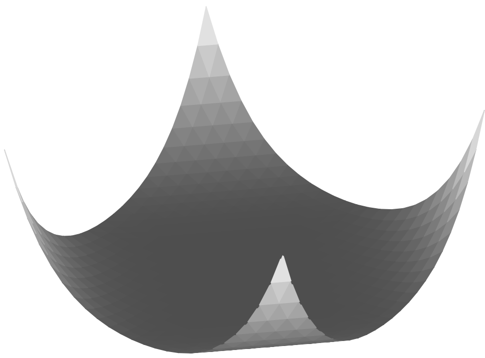
Egy megoldást globálisnak nevezünk, ha nem adható nála jobb, az esetleges korlátozási feltételeket is kielégítő megoldás a célfüggvény értelmezési tartományán. Ha ez a megoldásnak csak egy szűk környezetére teljesül, akkor a megoldást lokálisnak nevezzük.

A továbbiakban az $X = [a, b] = \{x \in \mathbb{R}^n : a \leq x \leq b\}$ valós kompakt halmazt (n-dimenziós) intervallumnak hívjuk. Amennyiben a dimenziószám a szövegkörnyezetből egyértelmű, egyszerűen intervallumnak nevezzük. Az f befoglalófüggvénye egy olyan F függvény, amely D minden X részintervallumához hozzárendel egy olyan $\mathbb{R}$-beli intervallumot, amelyre igaz, hogy $f(X) \subseteq F(X)$, ahol $f(X)$ az f függvény X feletti értékkészlete. Ilyen befoglaló függvény adható intervallum-aritmetika segítségével és szemléletesen az 1. ábrán látható egydimenziós esetben.

Megemlítjük, hogy a négy alapműveletet értelmezhetjük intervallumokra, valamint a standard függvények kiterjesztése is könnyen megvalósítható. Ezekkel a kiterjesztésekkel pont egy befoglaló függvényét adhatjuk



(a) Sok helyi minimumponttal rendelkező függvény.



(b) Nagy kiterjedésű minimumhely.

2. ábra. Kritikus függvények, amelyek minimalizálása valamely szempontból nehéz.

meg a standard függvényekből és alapműveletekből felépíthető függvényeknek. További részleteket az [1] könyvben találhatunk.

A következőkben néhány pontban áttekintjük az általános globális optimalizálás nehézségeit. Ezekre a program és az általa használt technikák segítségével megoldásokat találhatunk.

Lehetséges megoldás létezése. A korlátozó feltételeket tartalmazó feladatoknál problémát okozhat, hogy van-e lehetséges megoldás. Az intervallumos technikával ki tudjuk zárni azokat a részintervallumokat, amelyek garantáltan nem tartalmaznak megoldást, így szűkítve a keresés alaphalmazát.

Több kritikus lokális pont. Hogyan állapítsuk meg, hogy az a helyi minimum, amelyet megtaláltunk valójában a globális minimum? Egy olyan célfüggvényre, amelynek sok helyi minimumpontja van (lásd például a 2(a) ábrát), az intervallum-aritmetika segítségével garantáltan el tudjuk dönteni, hogy egy helyi minimum valójában globális minimum-e.

Szinguláris, vagy közel szinguláris megoldások. Az olyan célfüggvények, amelyek nagyon laposak vagy tartalmaznak ilyen részeket, gyakran szinguláris viselkedést mutatnak (lásd a 2(b) ábrát). Az intervallumos technika képes arra, hogy a megoldás pontok teljes halmazát befoglalja, ahol azok a globális minimum értékét elérik.

Nem pontos paraméterek. A valós életből vett problémák általában méréseken alapulnak. Ezen paraméter értékek alapján felállított modellek már eleve hibával terheltek, mivel nem tudunk pontosan mérni. Gyakran a pontos érték nem is adható meg. Ilyenkor ha a paramétereket befoglaló intervallumokkal adjuk meg, akkor a program ezek figyelembevételével a lehető legszűkebb befoglalását adja a lehetséges eredményeknek.

Magas dimenziószám. Ahhoz, hogy garantáljuk, hogy a globális optimum meghatározható lesz, a korlátozás és szétválasztás módszerét használjuk. A magas dimenziószám azonban nehézségeket okozhat. A legrosszabb esetben az algoritmus $\mathcal{O}(2^n)$ műveletigényű is lehet, ha a probléma n darab független változót tartalmaz. Számos esetben azonban lineáris algebrai eszközök segítségével ez a mérték csökkenthető. A GlobSol-lal eddig megoldott legnagyobb méretű probléma 128 dimenziós volt, itt azonban erősen kihasználták a rendszer ritkaságát, azaz azt a tényt, hogy a feladat nagyon sok paraméterének az értéke nulla volt. További megoldást jelenthet a párhuzamosítás, amelyen a program írói dolgoznak, sőt egy korai verzió már el is érhető az Interneten keresztül:
`http://www.mscs.mu.edu/~globsol/ParaGlobSol.html`.

műv.kód	p	q	r	a	b
5	3	1	0	-	-
5	4	2	0	-	-
20	5	3	4	-	-
22	6	1	0	3	-
20	7	6	0	-	-
23	8	7	0	-	$-6\sqrt{3}$
18	5	0	0	-	-

(a) kódlista

műv.kód	p	q	r	a	b
5	3	1	0	-	-
5	4	2	0	-	-
20	5	3	4	-	-
22	6	1	0	3	-
20	7	6	2	-	-
23	8	7	0	-	$-6\sqrt{3}$
22	9	1	0	2	-
26	10	0	0	3	-
22	11	2	0	2	-
26	12	0	0	1	-
18	5	0	0	-	-
18	9	0	0	-	-
18	11	0	0	-	-

(b) derivált kódlista

műv.kód	művelet
5	$x_p = x_q^2$
18	x_p függő vált.
19	$x_p = x_q^r$
20	$x_p = x_q + x_r$
22	$x_p = ax_q$
23	$x_p = x_q + b$
26	$x_p = a$

(c) műveletek

3. ábra.
Az $f(x,y) = x^2 + y^2$ függvényhez és a $3x + y \leq 6\sqrt{3}$ korlátozó feltételhez tartozó kódlista és derivált kódlista a műveletek kódjával.

2.2. Kódlista

A minimalizálandó függvényről a program készít egy úgynevezett kódlistát, amelynek lényege, hogy a megadott függvényt alaputasításokra bontva szimbólikusan tárolja. Ez nagyon fontos, mert így a kódlista segítségével a már említett intervallum-aritmetika szabályai szerint kiszámíthatjuk a függvény befoglalófüggvényét vagy szimbólikus manipulációval hasznosíthatjuk az ismert képleteket. Másrészt, ez alapján automatikus deriválással készül a derivált kódlista is, amire például az intervallumos Newton módszer támaszkodik.

A kódlistára a 3. ábrán láthatunk egy példát az $f(x,y) = x^2 + y^2$ függvény esetére kiegészítve a $3x + y \leq 6\sqrt{3}$ korlátozó feltétellel. Itt először feltüntettük a függvényből kapott kódlistát valamint a derivált kódlistát, illetve az ebben a példában szereplő összes művelethez tartozó kódszámot. Az (a) és (b) táblázatokban az utolsó két oszlop tartalmazza az egyes konstansok értékeit, amiket a GlobSol intervallumokként kezel. Tehát az itt feltüntetett $-6\sqrt{3}$ értéket nem ebben a formában, hanem a megfelelő lebegőpontos szám egy intervallumos befoglalásával kezeli.

A kódlista előállításáról, az ezzel kapcsolatos további lehetőségekről részletesebben a [11] könyv 78–102. oldalán olvashatunk.

2.3. Korlátozás és szétválasztás

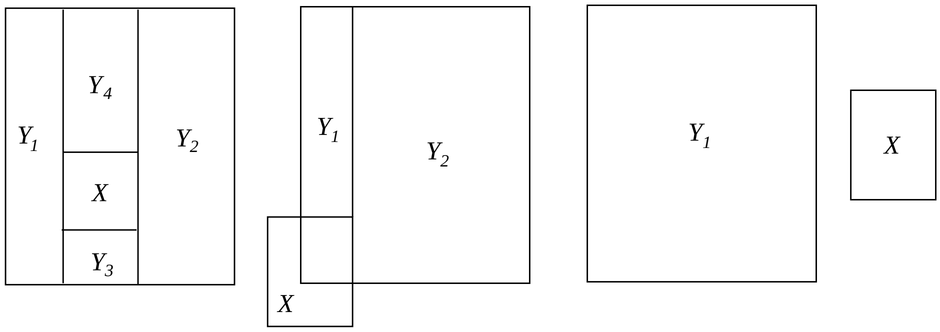
A korlátozás és szétválasztás módszerét alkalmazzuk, hogy pontos és szűk korlátot adjunk az optimumra, és kizárjuk az optimumot nem tartalmazó részeket. Ehhez három intervallumlistát használunk:

$\mathcal{L}$ a még nem vizsgált intervallumok listája;

$\mathcal{R}$ olyan részintervallumok, amelyek bizonyítottan tartalmaznak lokális minimumot;

$\mathcal{U}$ olyan részintervallumok, amelyekről nem tudjuk biztosan, hogy tartalmaznak-e lokális minimumot, vagy sem;

Hogy találjunk egy nagyon szűk intervallumot az x^* minimumpontra és az f^* minimumértékre, amelyekre $f^* = f(x^*) \leq f(x)$ minden $x \in X_0 \subset \mathbb{R}^n$, hajtsuk végre a következő algoritmust:



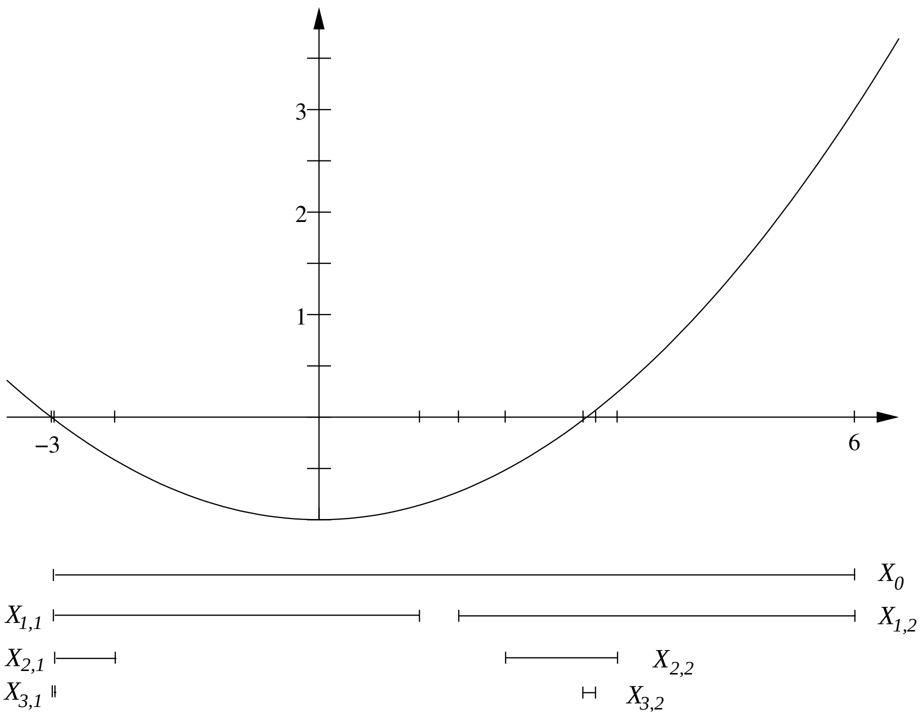
4. ábra. A komplementerképzés három esete.

Inicializáljuk az $\mathcal{L} = \{X_0\}$ intervallumlistát
Legyen $\bar{f} = +\infty$ // ebben tároljuk az aktuális felső korlátot; $f^* \leq \bar{f}$
while $\mathcal{L}$ nem üres **do**
  $X^{(akt)} = \{$a következő intervallum az $\mathcal{L}$ listában$\}$
  töröljük $X^{(akt)}$-ot az $\mathcal{L}$ listából
  if $\bar{f} < F(X^{(akt)})$ **then**
    dobjuk el az $X^{(akt)}$ intervallumot
  else
    próbálunk javítani az $\bar{f}$ korláton
    if tudjuk máshogyan bizonyítani, hogy nem tartalmaz globális optimumot **then**
      dobjuk el az $X^{(akt)}$ intervallumot
    else if bizonyítani tudjuk egyetlen helyi minimum létezését az $X^{(akt)}$ intervallumban **then**
      összehúzzuk az $X^{(akt)}$ intervallumot a helyi minimum körül, és hozzáadjuk az $\mathcal{R}$ listához
    else if az $X^{(akt)}$ intervallum elég kicsi **then**
      hozzáadjuk az $X^{(akt)}$ intervallumot az $\mathcal{U}$ listához
    else
      felosztjuk az $X^{(akt)}$ intervallumot részintervallumokra, és az $\mathcal{L}$ listához fűzzük azokat
    end if
  end if
end while

A tényleges kód ennél több részletben bonyolultabb, de a fő elv ezzel megegyező. További részleteket a [11] könyv 145–149. oldalán találhat az érdeklődő olvasó.

2.4. Epszilon-kiterjesztés

Akkor beszélünk epszilon-kiterjesztésről, amikor létrehozunk egy olyan, lehetőleg nagy intervallumot egy közelítő gyök körül, hogy a gyök létezését és egyértelműségét meg tudjuk mutatni az intervallumon belül [12]. Tulajdonképpen ez egy olyan iteratív eljárás, amely a közelítő gyök egy kis környezetével kezdve, lépésenként növelve az intervallum szélességét, igazolja a megkövetelt létezést és egyértelműséget addig, ameddig ez lehetséges. Természetesen, ha a létezés egyszer már igazolt, akkor az újabb megnövelt intervallumra elegendő megmutatni, hogy másik gyök nem létezhet ezen belül. Vagyis az intervallum szélességének ε-nal való növelését addig folytatjuk, amíg az egyértelműség megmutatható. Ha már nem tudjuk tovább növelni az intervallumot, akkor vegyük ki ezt a keresési térből, megkönnyítve a megmaradó intervallumok eliminálását. Az adott intervallumot úgy vetjük el, hogy képezzük a komplementerét a keresési térre nézve. A komplementerképzés három alapesetét a 4. ábrán láthatjuk. A komplementerképzés egy intervallum listát hoz létre, aminek az uniója adja ki a megmaradt keresési tartományt. A használt $\mathcal{O}(n)$ műveletigényű algoritmus $2n$ darab új intervallumot hozhat be legfeljebb. Amennyiben a keresési tér már egy intervallum listával volt megadva, ez az intervallumszám-növekedés nem jelentős.



5. ábra. Az $f(x) = \frac{1}{9}x^2 - 1$ függvényre az intervallumos Newton-módszer eredménye.

2.5. Intervallumos Newton-módszer

A numerikus matematikából jól ismert Newton-módszer – amelyet nemlineáris egyenletek, illetve egyenletrendszerek megoldására használhatunk – az intervallumos világba is átvihető. A hagyományos algoritmusra jellemző erősség, a négyzetes helyi konvergencia, itt is érvényben marad. Egyváltozós esetben az átírás meglehetősen egyszerű, itt is a középérték tételből indulunk ki, azaz ha f kétszer folytonosan deriválható egy X intervallumon, és létezik olyan $x^* \in X$, hogy $f(x^*) = 0$, akkor bármely $\check{x} \in X$-re

$$0 = f(x^*) = f(\check{x}) + f'(\xi)\,(x^* - \check{x})$$

valamely $\xi \in X$-re. Vagyis ha $f'(\xi)$ helyett a derivált befoglaló függvényével számolunk az X intervallumon (ezt többféleképpen megtehetjük, lásd a [11] könyv 1.3 fejezetét), akkor az $\check{x} - f(\check{x})/F'(X)$ intervallum tartalmazni fogja a x^* minimumpontot. Ez az intervallum nem feltétlenül része a kiinduló intervallumnak, ezért úgy kapjuk meg az iteráció következő elemét, ha vesszük ezek metszetét. Tehát számoljunk a következő iterációs képlettel:

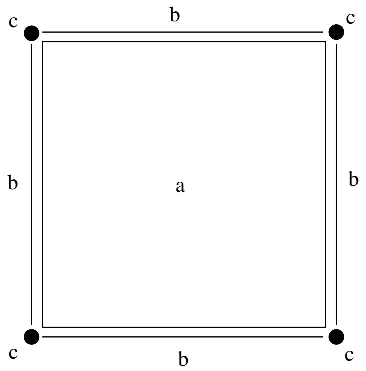
$$N(f; X_k, \check{x}_k) = \check{x}_k - \frac{f(\check{x}_k)}{F'(X_k)}, \qquad X_{k+1} = N(f; X_k, \check{x}_k) \cap X_k$$

ahol $\check{x}_k$ jelölje az X_k intervallum középpontját. Említésre méltó az a tétel, mi szerint ha $N(f; X, \check{x}) \subseteq X$, akkor az X intervallumban pontosan egy zérushely van.

Nézzünk erre egy egyszerű példát. Legyen $f(x) = \frac{1}{9}x^2 - 1$, és keressük a gyökeit az $X_0 = [-3, 6]$ intervallumon. Ekkor $\check{x}_0 = 1.5$, valamint a derivált legszűkebb befoglalófüggvényével, az $F'(X) = \frac{2}{9}X$-el számolva $F'(X_0) = [-4, 8]$. Könnyen átgondolhatóak a következő intervallumos számítások, melyek részletes definícióit megtalálhatjuk a [1] könyvben:

$$N(f; X_0, \check{x}_0) = 1.5 - \frac{-0.75}{[-0.667, 1.333]} = 1.5 - ([-\infty, -0.0625] \cup [0.375, \infty]) = ([-\infty, 1.125] \cup [1.5625, \infty])$$

$$N(f; X_0, \check{x}_0) \cap X_0 = [-3, 1.125] \cup [1.5625, 6] = X_{1,1} \cup X_{1,2}$$



6. ábra. Egy kétdimenziós intervallum hámozása.

$$N(f; X_{1,2}, \check{x}_{1,2}) = 3.781 - \frac{0.5887}{[0.347, 1.333]} = 3.781 - [0.442, 1.696] = [1.516, 2.285]$$

$$N(f; X_{1,2}, \check{x}_{1,2}) \cap X_{1,2} = [2.086, 3.340] = X_{2,2}$$

A számítások további eredményeit lásd az 5. ábrán. Többváltozós esetben számos változat elképzelhető arra vontkozóan, hogy a kapott lineáris rendszer megoldáshalmazát hogyan korlátozzuk. Az algoritmust itt felhasználhatjuk a megoldás létezésének és *egyértelműségének* bizonyítására. A módszerről bővebben a [8, 14] könyvekben olvashatunk.

2.6. Feltétel propagáció (constraint propagation)

A kódlista sok információt tartalmaz egy adott intervallum-kiterjesztés kiértékelése alatt keletkező közbülső mennyiségek közötti kapcsolatról. Ez a kapcsolat néha használható arra, hogy csökkentsük a túlbecslések mértékét az ilyen intervallumos kiterjesztésekben, vagy hogy segítsen a nemlineáris egyenletrendszerek megoldásában. A GlobSol-ban felhasznált ezzel kapcsolatos módszerekről, az ún. substitution-iteration technikáról a [11] könyv 227–234. oldalain olvashatunk bővebben.

2.7. Hámozás

Ha a problémánk olyan, hogy korlátozó feltételek is adottak, akkor a program egy ún. hámozási technikát (peeling) [9] alkalmaz arra, hogy a feltételek által meghatározott tartományból kiválogassa az összes lehetséges alacsonyabb dimenziós részt. Az így kapott intervallumok azután ugyanúgy feldolgozásra kerülnek, azonban az alacsonyabb dimenziószám miatt várhatóan kevesebb műveletet igényelnek. Ezen kívül ha a globális optimumra van egy jó felső korlátunk, akkor ezzel az eljárással a hámozás során keletkező intervallumok nagy része eldobható.

A két dimenziós esetre a 6. ábrán láthatunk példát. Itt összesen 9 új intervallumot kapunk: egyet, amely $\mathbb{R}^2$-beli, négyet, amelyek $\mathbb{R}$-beliek, és négyet, amelyek diszkrét pontok. Ezeket az ábrán rendre az a, b és c betűk jelölik.

3. Alkalmazások

Ebben a fejezetben bemutatunk néhány olyan problémát, amelyek megoldására a GlobSol kiválóan használható. Az érdeklődő olvasó figyelmébe ajánljuk továbbá a GlobSol honlapját:
`http://interval.louisiana.edu/GlobSol/`
ahol érdekes gyakorlati alkalmazásokról található leírásokat és esettanulmányokat, amelyek címszavakban a következők: rakétamotor fúvókájának tervezése, portfólió kezelés, valutakereskedelem, génkutatás, orvosi képfeldolgozás, kémiai számítások és jelfeldolgozás. Ezen alkalmazások rövid ismertetését az [5] cikkben is megtalálhatjuk.

3.1. Nemlineáris egyenletrendszerek

A Polygon egy korábbi számában olvashattunk az azonos átmérőjű körök egységnégyzetbe való pakolásáról [16]. A probléma megoldásának van egy olyan lépése, amely a kapott megoldások helyességének ellenőrzésére szolgál. Amikor ugyanis adott egy pakolás, a körök egymáshoz viszonyított elhelyezkedéséből egy nemlineáris egyenletrendszert tudunk felírni. Itt azt kell figyelembe venni, hogy két kör érintkezik-e. A kapott nemlineáris egyenletrendszert megoldva megkapjuk, hogy a módszerünk által szolgáltatott koordináta-értékek helyesek-e, illetve mennyire pontosak.

Példaként említhetjük, hogy 10 kör esetében egyszerűsítéseket alkalmazva egy 5 változós nemlineáris egyenletrendszert tudunk felírni [15]:

$$\begin{aligned}
1 - 2x_1 + x_1^2 + x_4^2 - x_5^2 &= 0 \\
x_1^2 - 2x_1x_5 + x_3^2 &= 0 \\
2 - 4x_2 + 4x_2^2 + x_4^2 + 2x_4x_5 - 2x_4 - 2x_5 &= 0 \\
5 - 6x_5 - 8x_2 + x_5^2 + 8x_5x_2 + 4x_2^2 + 4x_4^2 - 4x_4 - 4x_4x_3 - 4x_4x_5 + 2x_3 + x_3^2 + 2x_3x_5 &= 0 \\
4x_2^2 + 1 - 2x_3 - 2x_5 + x_3^2 + 2x_3x_5 - 3x_5^2 &= 0.
\end{aligned}$$

Az egyenletrendszer megoldásából az x_5 értéke az, amely számunkra fontos, ez adja meg ugyanis a középpontok közti minimális távolság maximumát. A GlobSol az így kapott problémát viszonylag rövid idő alatt megoldja és az x_5 értékére eredményül a

$$[0.421279543983902000, 0.421279543983904800]$$

intervallumot adja.

A probléma átírása azonban nehezen automatizálható. Ha semmilyen – kézi vagy gépi – egyszerűsítési módszert sem vetünk be, akkor a nemlineáris egyenletrendszerünk dimenziószáma meglehetősen nagy lesz, amelynek megoldása kihívást jelent a GlobSol számára.

Megjegyezzük továbbá, hogy az eredeti, teljes körpakolási feladat különös módon már kis körszám esetén is nehézségeket okoz, a cikk írásakor elérhető GlobSol verzió már 5 kör esetére sem ad kielégítő eredményt.

3.2. Nem differenciálható problémák

A valós életből vett problémák megoldása során számos esetben találkozhatunk olyan célfüggvénnyel, amelyben abszolútérték vagy két kifejezés maximuma szerepel. Ilyen esetekben gondot okozhat az, hogy a függvényünk nem differenciálható. Így az alkalmazott automatikus deriválás miatt nem használhatjuk a Fortran 90 nyelvben rendelkezésre álló utasításokat, amelyek az esetszétválasztást kezelni tudják. Hogy ilyen függvényeket is tudjunk optimalizálni, a GlobSol-ban rendelkezésre áll a [10] cikkben bevezetett $x_p = \chi(x_s, x_q, x_r)$ függvény, melynek használatával a GlobSol kódlistát generál a megfelelő IF-THEN-ELSE típusú elágazásokhoz. A valós értékű χ függvény definíciója a következő:

$$\chi(x_s, x_q, x_r) = \begin{cases} x_q & \text{ha} \quad x_s < 0, \text{ és} \\ x_r & \text{különben.} \end{cases}$$

A χ függvény használatával megoldható az $x_p = |x_q|$, valamint az $x_p = \max\{x_q, x_r\}$ alakú függvények definiálása és használata.

Tekintsük például a következő feladatot. Adott a síkon egy egységnyi szakasz, az egyszerűség kedvéért legyen ez az x tengelyen. Az optimalizálandó függvényünk értéke ettől a szakasztól vett távolság lesz, azaz vagy a közelebbi végponttól vett távolság, vagy pedig a megfelelő y koordintátaérték abszolútértéke. Feladatunk az így megadott függvény minimalizálása.

A probléma leírásából rögtön látszik, hogy a minimum a megadott egységnyi intervallumon vétetik fel, tehát válaszként a teljes intervallumot kell megadni. A GlobSol-ban ezt a problémát a χ függvény segítségével oldhatjuk meg, hiszen ezzel tudjuk kezelni az esetszétválasztást. Az eredeti probléma célfüggvénye azonban differenciálható volt, így jogos annak a kérdésnek a felvetése, hogy miért kell akkor a χ függvényt használni. Erre azt a választ adhatjuk, hogy ennek segítségével gyorsan és áttekinthetően tudjuk kezelni a problémát, valamint a GlobSol is a helyes választ adja, ha így adjuk meg a fenti függvényt (visszaadja a szakaszt több rövidebb részre felbontva).

3.3. EX3

Tekintsük az $f(x) = x^6(\sin(1/x) + 2)$ függvényt. Ezen függvény globális minimumának megkeresése hagyományos eljárásokkal igazoltan reménytelen [6]. Ha a GlobSol segítségével próbáljuk minimalizálni ezt a függvényt a $[-1, 1]$ intervallumon, akkor azt a hibaüzenetet kapjuk, hogy a 0 szerepel a nevezőben. Ezt a hibaüzenetet az intervallum aritmetika szabályai miatt kapjuk, ugyanis nem megengedett olyan intervallummal osztani, amely tartalmazza a 0-t. Próbálkozhatunk úgy is, hogy a $[-1, -\varepsilon]$ és $[\varepsilon, 1]$ intervallumon keresünk, ahol $\varepsilon > 0$ egy rögzített kicsi lebegőpontos szám. Ebben az estben a GlobSol már be tudja határolni a minimumot, ellenőrzött megoldáshelyként olyan kis intervallumot ad, amelynek bal végpontja az ε szám.

4. Egy egyszerű példa

Ebben a részben egy egyszerű példán keresztül bemutatjuk a GlobSol használatát. Példánk nem a valós életből vett, de tartalmaz olyan jellegzetességeket, amelyen keresztül a program különböző tulajdonságai, beállítási lehetőségei jól szemléltethetők. Megemlítjük továbbá, hogy példánkkal (legalábbis ebben a formában) a hagyományos minimumkereső módszerek nehezen boldogulnak.

Mint azt korábban említettük, a felhasználónak a problémát Fortran 90 nyelven kell programoznia. Ahhoz, hogy egyszerűen leírható problémákat oldjunk meg, nincs szükség mélyebb Fortran 90 ismeretekre, mindazonáltal a [13] könyvet ajánljuk, mint bevezetőt, de az Interneten is rengeteg szabadon letölthető Fortran 90 tankönyvet találhatunk (például a `http://www.liv.ac.uk/HPC/F90page.html` címen).

Próbáljuk tehát megoldani a

$$\min_{(x,y)\in[0,6]\times[0,5]} f(x,y) = x^2 + y^2$$

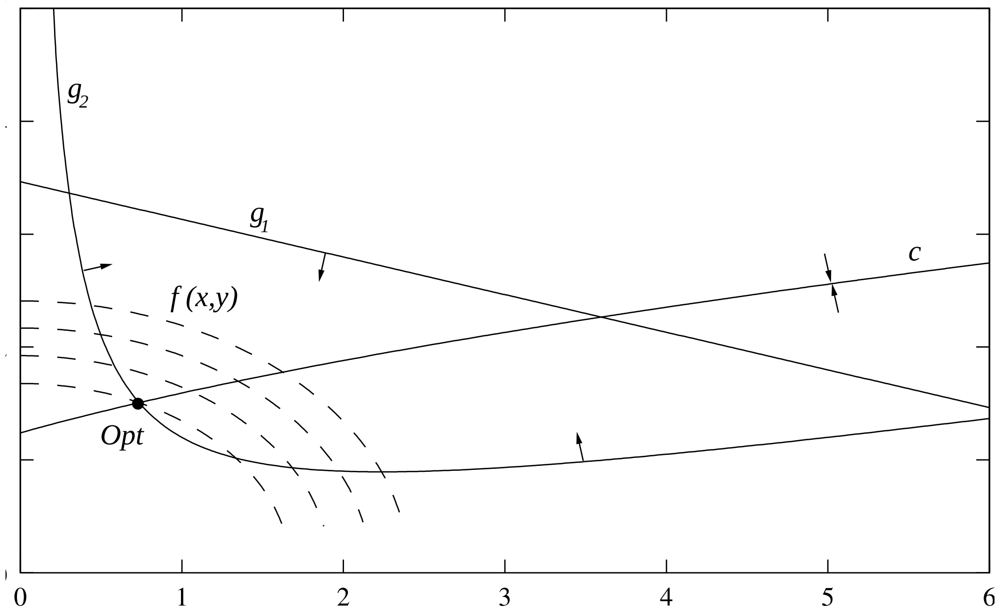
optimalizálási feladatot a

$$\begin{aligned} 3x + y &\leq 6\sqrt{3} \\ 5xy - x^2 &\geq 5 \\ x - y^2 &= 2\sqrt{3} - 5 \end{aligned}$$

korlátozó feltételekkel kiegészítve. Az így megadott feladatot szemlélteti a 7. ábra. Itt $f(x, y)$ jelöli a célfüggvény szintvonalait, amiket szaggatott vonallal tüntettünk fel; g_1, g_2 jelöli a két egyenlőtlenség feltételt, c pedig az egyenlőség feltételt. Az egyes görbéken elhelyezett nyilak jelölik azt, hogy az adott görbék a sík mely részét engedik meg. Az optimális megoldásnak megfelelő pontot pedig *Opt* jelöli.

A GlobSol-ban a fenti problémát a következőképpen oldhatjuk meg. Hozzunk létre egy könyvtárat, amelyben a fájlokat fogjuk tárolni. Ami kell:

- egy olyan fájl, amely a célfüggvény leírását tartalmazza Fortran 90 nyelven. A fájl nevét megválaszthatjuk. Példánkban ennek a fájlnak `x2y2.f90` lesz a neve.
- egy olyan adatfájl, amely tartalmazza a kiindulási intervallumokat, a korlátozó feltételre vonatkozó megkötéseinket, valamint esetleg még kiindulási értékeket a helyi kereséshez. A fájl neve meg kell, hogy egyezzen a célfüggvényt leíró fájl nevével, kiterjesztése azonban `DT?` legyen. Itt a `?` helyére valamilyen számot kell írni.
- egy `Makefile` nevű fájl, ami a fordításhoz szükséges paramétereket tartalmazza. Ezt nem fontos nekünk teljes egészében elkészíteni, átmásolhatjuk az `integration_test_data` könyvtárból (ebben a könyvtárban találhatunk néhány előre elkészített tesztpéldát), amit aztán csak egy picit kell módosítanunk.
- egy `GlobSol.CFG` nevű adatállomány, amelyet szintén nem fontos nekünk elkészíteni: a példaprogramokat tartalmazó könyvtárból átmásolhatjuk ide. Általában ezt a fájlt nem nagyon kell módosítanunk, de ha mégis, akkor nagy segítségünkre lehet az a tény, hogy meglehetősen bőségesen kommentált a már előre elkészített példánya, ami alapján viszonylag könnyen boldogulunk. Néhány egyszerű példa lefuttatása után azonban mindenképp érdemes ezt a fájlt kicsit tüzetesebben is átnézni, hiszen valóban rengeteg paraméter beállítását teszi lehetővé.



7. ábra. A példa kétdimenziós ábrázolása.

Ezek után nézzük meg, hogyan kell a fent leírt problémát kódoló fájlt elkészíteni.

```
PROGRAM x2y2
  USE CODELIST_CREATION

  TYPE(CDLVAR),  DIMENSION(2)  :: X
  TYPE(CDLINEQ), DIMENSION(2)  :: G
  TYPE(CDLEQ),   DIMENSION(1)  :: C
  TYPE(CDLLHS),  DIMENSION(1)  :: F

  OUTPUT_FILE_NAME = 'x2y2.CDL'
  CALL INITIALIZE_CODELIST(X)

  F(1) = X(1)**2 + X(2)**2
  G(1) = 3*X(1) + X(2) - 6*SQRT(3.D0)
  G(2) = -(5*X(1)*X(2) - X(1)**2 - 5)
  C(1) = -X(1) + X(2)**2 - 5 + 2*SQRT(3.D0)

  CALL FINISH_CODELIST
END PROGRAM x2y2
```

Nézzük meg, hogy itt az egyes sorok mit is jelentenek.

- Először megadtuk a programunk nevét, valamint használjuk azt a modult, aminek segítségével a GlobSol a 2.2. fejezetben megismert kódlistát elkészíti.
- Optimalizálandó változónk típusa `CDLVAR`, ami a "kódlista változó"rövidítése, a célfüggvény típusa pedig `CDLLHS`, amely az egész kódban csak egyszer szerepelhet és akkor is csak egy értékadás baloldalán.
- Egyenlőtlenség feltételeket `CDLINEQ` típusú változók, egyenlőség feltételeket pedig `CDLEQ` típusú változók deklarálásával írhatunk. Itt figyelni kell arra, hogy egyenlőtlenség esetén mindig úgy kell felírni a feltételt, hogy a megadott kifejezés 0-nál kisebb vagy egyenlő legyen.

- Meg kell adni annak a fájlnak a nevét, amely majd az automatikusan elkészülő kódlistát tartalmazza. Ennek neve egyezzen meg a `.f90` fájl nevével, kiterjesztése pedig `.CDL` legyen.
- Ezután kell a kódlista inicializáló eljárást meghívni, majd a korlátozó feltételeket és a minimalizálandó célfüggvényt megadni.
- Végül a kódlista készítését lezáró eljárás hívásával és a program végének megfelelő kulcsszavakkal fejezzük be a Fortran 90 kód megírását.

Ha ezzel megvagyunk, akkor készítsük el azt a fájlt, amely a kiindulási adatokat tartalmazza. Ennek struktúrája szigorú szabályok szerint épül fel. A legelső sorban szereplő lebegőpontos szám értéke azt mondja meg, hogy mekkora a helyre vontakozó tolerancia. Itt túl kicsi számot nem adhatunk meg, mert akkor a GlobSol azt helyettesíteni fogja az itt használható legkisebb értékkel. Ha viszont "nagyon nagy" számot adunk meg, akkor a kapott eredmény nem lesz túl pontos. A 2. sortól az $n+1$-edik sorig (példánkban most $n = 2$) bezárólag szereplő intervallumok adják meg a program számára az egyes változók kiindulási intervallumát. Itt pontosan annyi intervallumot kell feltüntetnünk, ahány változó adott az eredeti problémában. Az $n+2$-edik sortól a $2n+1$-edik sorig $2-2$ T(rue), illetve F(alse) logikai értéket írhatunk, amellyel megmondhatjuk, hogy mely alsó, illetve felső korlátok reprezentálnak valódi korlátozó feltételeket és melyek azok, amelyek esetleg átléphető korlátokat adnak a keresési térben. A $2n+2$-edik sortól a $3n+1$-edik sorig szereplő értékek opcionálisak. Itt adhatjuk meg az egyes változók általunk sejtett kiindulási értékét. Az utolsó sor szintén opcionális, az itt szereplő logikai érték azt mutatja, hogy az engedélyezzük-e a közelítő optimum-pont további finomítását egy helyi keresővel.

```
1D-10
0 6
0 5
T T
T T
```

A következőkben megmutatjuk, hogy a `Makefile` fájlban mely sorokat kell kijavítanunk.

```
PROB = x2y2
VERS = 1
DATA =
COPYDATA =
INTOPT_DIR = /opt/GlobSol/
...
```

Itt az egyes sorok értelmezése a következő:

1. A `.f90` és `.DT?` fájlok nevei, most `x2y2`
2. Ide egy számot kell írnunk, amely megegyezik a `.DT?` fájl nevében a `?` helyére írt számmal. Ezzel tehát ugyanazon problémára más-más kiindulási adatokat tartalmazó adatállományt rendelhetünk.
3. Olyan opcionális adatfájlok nevei, amelyeket a `.f90` fáljunk olvas be. Ha egy vagy több adatfájlt olvas be a programunk, akkor azok neveit kell ide beírni, különben hagyjuk üresen.
4. Olyan opcionális adatfájlok nevei, amelyeket a különböző verziókhoz olvasunk be.
5. a GlobSol könyvtárának abszolút elérési útja.

Ezen kívül fontos még, hogy a fordítóprogram paramétereit tartalmazó sorokat is beállítsuk, ha szükséges. Ezt azonban itt nem részletezzük, csak megemlítjük, hogy itt is már előre elkészítettek néhány gyakori Fortran 90 fordítónak megfelelő beállítást.

Ha mindezzel kész vagyunk, akkor a `make` parancs kiadásával a `Makefile` feldolgozásra kerül: a fordító először lefordítja a Fortran kódot, abból legyártja a kódlistát és a derivált kódlistát, valamint elvégzi a kódlista optimalizálását az `optimize_codelist` nevű programmal futtatásával, amely eltávolítja a kódlistából a redundáns sorokat. Ezután a `find_global_min` nevű program kerül végrehajtásra, amelynek eredménye az `x2y2.OT1` nevű fájlba kerül. Lássuk, hogy példánk futtatásának eredménye mit tartalmaz.

```
 Output from FIND_GLOBAL_MIN on  03/01/2000  at  12:03:28.
 Version for the system is: December 7, 1999

 Codelist file name is: x2y2G.CDL
 Box data file name is: x2y2.DT1

 Initial box:
[    0.0000000000000000000D+00,   0.6000000000000000000D+01 ]
[    0.0000000000000000000D+00,   0.5000000000000000000D+01 ]

 BOUND_CONSTRAINT:
   T T   T T
 ---------------------------------------
 CONFIGURATION VALUES:

 EPS_DOMAIN:   0.1000D-09   MAXITR:   20000
 MAX_CPU_SECONDS:    0.3600000000000000000D+04
 DO_INTERVAL_NEWTON: T  QUADRATIC: T  FULL_SPACE: F
 VERY_GOOD_INITIAL_GUESS: F
 USE_SUBSIT: T
 OUTPUT UNIT: 7  PRINT_LENGTH: 2
 PHI_MUST_CONVERGE: T
 EQ_CNS_MUST_CONVERGE: T
 INEQ_CNS_MUST_CONVERGE: T
 PHI_THICKNESS_FACTOR:   0.5000000000000000000D+00
 EQ_CNS_THICKNESS_FACTOR:   0.5000000000000000000D+00
 INEQ_CNS_THICKNESS_FACTOR:   0.5000000000000000000D+00
 PHI_MUST_CONVERGE: T
 EQ_CNS_MUST_CONVERGE: T
 INEQ_CNS_MUST_CONVERGE: T
 PHI_CONVERGENCE_FACTOR:   0.1000000000000000000D-13
 EQ_CNS_CONVERGENCE_FACTOR:   0.1000000000000000000D-13
 INEQ_CNS_CONVERGENCE_FACTOR:   0.1000000000000000000D-13
 CONTINUITY_ACROSS_BRANCHES: F
 SINGULAR_EXPANSION_FACTOR:   0.1000000000000000000D+02
 HEURISTIC PARAMETER ALPHA:   0.5000000000000000000D+00
 Default point optimizer was used.

 THERE WERE NO BOXES IN COMPLETED_LIST.

 LIST OF BOXES CONTAINING VERIFIED FEASIBLE POINTS:

 Box no.: 1
 Box coordinates:
[    0.7352997765381279000D+00,   0.7352997765382527000D+00 ]
[    0.1507049488703128000D+01,   0.1507049488703178000D+01 ]
PHI:
[    0.2811863922777381000D+01,   0.2811863922777716000D+01 ]
 B%LIUI(1,*):
 F F
 B%LIUI(2,*):
 F F
 B%SIDE(*):
 F F
 B%PEEL(*):
 T T
 Level:  3
 Box contains the following approximate root:
```

```
   0.735299776538190300D+00   0.150704948870315300D+01

 OBJECTIVE ENCLOSURE AT APPROXIMATE ROOT:
[    0.281186392277754600D+01,   0.281186392277755100D+01 ]
 Unknown =  T    Contains_root = T
 Changed coordinates:
 T F
 U0:
[    0.641200626388966000D+00,   0.641200626389070800D+00 ]
 U:
[    0.000000000000000000D+00,   0.100000000000000000D+01 ]
[    0.217471131840985400D+00,   0.217471131841031000D+00 ]
 V:
[   -0.375936486351107300D+00,  -0.375936486350993100D+00 ]
 INEQ_CERT_FEASIBLE:
  T F
 NIN_POSS_BINDING: 1
 ------------------------------------------------
 ALGORITHM COMPLETED WITH LESS THAN THE MAXIMUM NUMBER,
       20000  OF BOXES.
 Number of bisections:  3
 No. dense interval residual evaluations -- gradient code list:  63
 Number of orig. system inverse midpoint preconditioner rows:  6
 Number of orig. system C-LP preconditioner rows:  100
 Number of solutions for a component in the expanded system:  511
 Total number of forward_substitutions:  246
 Number of Gauss--Seidel steps on the dense system:  106
 Number of gradient evaluations from a gradient code list:  12
 Total number of dense slope matrix evaluations:  63
 Total number second-order interval evaluations of the
       original function:  23
 Total number dense interval constraint evaluations:  334
 Total number dense interval constraint gradient component
       evaluations:  454
 Total number dense point constraint gradient component
       evaluations:  62
 Total number dense interval reduced gradient evaluations:  64
 Total number of calls to FRITZ_JOHN_RESIDUALS:  22
 Number of times the interval Newton method made a coordinate
       interval smaller:  58
 Total time spent doing linear algebra (preconditioners
       and solution processes):  1.000000163912773E-02
 Number of times the approximate solver was called:  3
 Number Fritz-John matrix evaluations:  20
 Number of times SUBSIT decreased one or more
       coordinate widths:  4
 Total number of boxes processed in loop:  4
 BEST_ESTIMATE:    0.281186392277771600D+01
 Overall CPU time:    0.499999988824129100D-01
 CPU time in PEEL_BOUNDARY:    0.999999977648258200D-02
 CPU time in REDUCED_INTERVAL_NEWTON:    0.100000016391277300D-01
```

A példából kiolvasható az előzetesen beállított kiindulási intervallumok értéke, valamit a korlátozó feltételek logikai értékei. Az utána következő viszonylag hosszú lista részletezi a GlobSol beállított paramétereit. Ezután jön annak az intervallumnak a részletezése, amely tartalmaz ellenőrzött lehetséges megoldást. Láthatjuk, hogy a GlobSol garantálja, hogy az általa megadott intervallumok tartalmazzák a globális minimum helyét és értékét.

A végén pedig kapunk egy részletes statisztikát a program futása közben végrehajtott műveletek és eljárások alkalmazásainak számáról, valamint a szükséges CPU időről: 5 századmásodperc, amit egy 433 MHz-es Pentium Celeron gépen Linux operációs rendszerrel működő Fujitsu Fortran 90 fordítóval kaptunk. Itt láthatjuk, hogy a probléma megoldásához összesen 63 függvény- és 12 gradines kiértékelésre volt szükség.

5. Numerikus tesztek

Ebben a fejezetben bemutatunk néhány standard tesztprobémán elvégzett futtatási eredményt, valamit ezek összevetését két hagyományos módszerrel valamit intervallumos módszerekkel. A GlobSol-t egy IBM RS/6000 gépen NAG Fortran 90 fordítóval futtatuk. A tesztproblémák a következők voltak: Shekel-5, Shekel-7, Shekel-10, Hartman-3, Hartman-6, Goldstein-Price, Branin RCOS, Six-Hump-Camel-Back, Rosenbrock és az EX2 függvény. A feltüntetett módszerek: Törn, Price és De Biase módszerei; A és B is egy klaszterező típusú, hagyományos sztochasztikus globális optimalizálási algoritmus, A egy kvázi-Newton módszert, B egy egyszerűsített véletlen keresőt használ helyi keresővel (lásd [6]); intA egy intervallumos módszert jelöl, amely csak egy nagyon egyszerű gyorsító eljárást használ, intB pedig ugyanaz az intervallumos módszer, de kiegészítve egyéb gyorsító eljárásokkal [7]; GlobSolA, GlobSolB és GlobSolC pedig háromféle pontossági beállítással a GlobSol eredményeit mutatja.

Az 1. táblázatban az egyes módszerek által szolgáltatott megoldások értékes jegyeit tüntettük fel. Az értékes jegyeket a következőképpen számítottuk: az $\tilde{x}$ közelítő szám értékes jegyeinek a száma $-\log_{10}|\frac{x^*-\tilde{x}}{x^*}|$, ha tudjuk az x^* pontos megoldást, és nem 0. Ha 0, akkor nem kell leosztani az x^*-gal. Intervallumos esetben a kapott intervallum szélessége kerül a számlálóba és a középpontja a nevezőbe. Láthatjuk, hogy a GlobSol eredményei lettek a legjobbak.

A 2. táblázat tartalmazza az egyes módszerek függvényhívásainak számát. Ebből a táblázatból kiolvashatjuk, hogy a GlobSol általában kevesebb függvénykiértékelést végzett, mint a hagyományos eljárások. Szembetűnő a Shekel problémák esetén a különbség. Itt kivételt képez a GP probléma, ennek megoldásához az intervallumos módszereknek magas kiértékelésszámra volt szükségük. Észrevehetjük továbbá, hogy az intervallumos módszerekhez képest a GlobSolnak általában kb. kétszer annyi kiértékelésre volt szükség.

A 3. táblázatban az egyes módszerek által használt ún. standard időt adjuk meg, aminek egysége a Shekel-5 függvény (4, 4, 4, 4) pontban való 1000-szeres kiértékeléséhez szükséges idő. Ez az általunk használt gépen 0.06 másodperc volt. Itt szembetűnő, hogy a hagyományos módszerek sokkal jobb eredményt mutatnak az intervallumos módszerekhez képest, azonban az utóbbiak az intervallum-aritmetika műveleteit használják, ami természetesen nagyobb időigényű, mint a valós aritmetika. Igaz továbbá az is, hogy az intervallumos módszerek garantált megoldást adnak az egyes problémákra, így megbízhatóbbak és ez a megbízhatósági szint a hagyományos módszerekkel nem érhető el.

1. táblázat. Értékes jegyek száma a globális minimumnál

Módszer	Teszt függvények									
	S5	S7	S10	H3	H6	GP	RCOS	SHCB	RB	EX2
Törn*	3.0	2.9	3.2	4.3	2.6	3.9	4.5	–	–	–
Price*	6.2	5.3	5.8	6.4	6.0	3.9	6.3	–	–	–
De Biase	2.9	3.4	2.0	4.7	4.7	4.8	–	6.4	–	–
A*	7.0	6.8	6.7	6.8	6.9	4.3	7.2	7.1	10.1	–
B*	4.9	4.0	5.2	4.3	4.3	4.4	4.1	4.5	4.7	–
intA*;**	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
intB**	8.0	8.0	8.0	8.0	8.0	8.0	8.0	8.0	8.0	8.0
GlobSolA	-	-	-	-	11.6	0.3	13.8	12.2	0.6	–
GlobSolB	10.7	10.3	10.3	14.1	14.0	0.8	13.2	14.2	1.4	0.4
GlobSolC	14.3	14.2	14.1	14.1	14.0	12.3	13.2	14.2	13.6	12.0

* Ezek deriváltmentes eljárások.

** Itt a megállási feltételnek megadott intervallum szélességből számolt értékek szerepelnek.

2. táblázat.
Függvénykiértékelések száma. Az utolsó 5 módszer esetén ez intervallumos befoglalófüggvényt jelent.

Módszer	Teszt függvények									
	S5	S7	S10	H3	H6	GP	RCOS	SHCB	RB	EX2
Törn*	3 649	3 606	3 874	2 584	3 447	2 499	1 558	–	–	–
Price*	3 800	4 900	4 400	2 400	7 600	2 500	1 800	–	–	–
De Biase	620	788	1 160	732	807	378	597	717	–	–
A*	990	1 767	2 396	216	1 446	436	330	233	410	–
B*	1 083	1 974	2 689	697	2 610	386	464	267	1 524	–
intA*	184	434	490	95 723	90 357	179 045	415	142 618	135	90 390
intB	144	149	158	303	1 681	15 694	202	734	316	274 229
GlobSolA	208	264	268	371	1 926	36 614	207	598	131	2 664
GlobSolB	211	315	323	607	1 709	36 683	249	630	205	2 705
GlobSolC	235	300	314	634	1473	36 646	249	634	216	2 963

* Ezek deriváltmentes eljárások.

3. táblázat. Az egyes eljárások CPU időigénye standard időegységben kifejezve.

Módszer	Teszt függvények									
	S5	S7	S10	H3	H6	GP	RCOS	SHCB	RB	EX2
Törn*	10.0	12.4	14.4	8.0	15.6	4.1	3.7	–	–	–
Price*	13.9	20.0	19.7	7.5	47.5	2.8	4.4	–	–	–
De Biase	26.1	23.0	33.7	17.6	23.1	16.8	15.2	23.2	–	–
A*	3.0	4.9	7.0	1.2	4.2	1.3	1.4	1.2	1.0	–
B*	3.5	6.0	8.8	1.9	14.2	1.5	1.6	1.3	1.5	–
intA*	200.9	655.5	1 047.8	461 760.8	675 540.7	1 262 822.9	47.8	541 358.8	9.6	524 971.3
intB	129.2	186.6	277.5	162.7	2 933.0	4 244.0	28.7	95.7	28.7	348 803.8
GlobSolA	128.3	222.0	284.8	240.0	6 240.3	4 735.5	38.0	131.0	15.2	8 887.5
GlobSolB	132.7	257.2	332.5	661.2	5 397.8	4 744.7	54.0	124.0	19.3	8 639.0
GlobSolC	236.5	251.5	319.5	564.3	4 944.7	4 751.0	54.8	132.7	20.0	9 717.0

* Ezek deriváltmentes eljárások.

6. GlobSol szerver

Amennyiben nem adottak a feltételeink a GlobSol installálásához (lásd a függelékben megadott útmutatót), lehetőségünk van egy GlobSol szerveren kipróbálni az ötleteinket. Ha igénybe akarjuk venni ezt a szolgáltatást, nincs más dolgunk, csak küldjük el e-mailben az optimalizálandó problémánkat a következő címre:

`h531733@mirror.cc.u-szeged.hu`

Itt egy program értelmezi a levelet, tehát ennek megadott módon tartalmaznia kell a fájlokat. Ha a levelünk `Subject:` mezője a `globsol_info_file` bejegyzést tartalmazza, akkor a szerver visszaküld egy levelet, amely tartalmaz egy példát, amit itt is láthatunk:

```
To: h531733@mirror.cc.u-szeged.hu
Cc:
Attachmnt:
Subject: globsol
```

```
---Message text---

>> problem <<
        HERE COMES THE NAME OF YOUR PROBLEM!
Be careful! In the problem.f90 the OUTPUT_FILE_NAME must be problem.CDL
>> problem.f90 <<
        HERE COMES THE problem.f90 FILE!
>> problem.DT1 <<
        HERE COMES THE problem.DT1 FILE!
>> makefile <<
        HERE COMES THE makefile, OR IF THERE ISN'T,
        WE USE THE GENERIC makefile!
>> GlobSol.CFG <<
        HERE COMES THE GlobSol.CFG, OR IF THERE ISN'T,
        WE USE THE GENERIC GlobSol.CFG!
>> other.file1 <<
        IF YOU HAVE OTHER FILES, WHAT WE HAVE TO USE
        FOR YOUR PROBLEM, PUT HERE!
        Of course, change the "other.file1" to the
        name of your file...
>> other.file2 <<
 ...
>> end <<
```

Ez alapján a szerver elindítja az optimalizálást, és a kapott eredményt visszaküldi a feladónak. Ha valamilyen hibával leáll a fordító, akkor a hibaüzenetet küldi vissza. Ha a program 2 napon belül nem fut le, akkor a szerver leállítja az optimalizálást, és 'it hasn't stopped' hibaüzenetet küld.

A `Subject:` mezőbe `info`-t írva bármely kérésre, kérdésre szívesen válaszolunk.

7. Köszönetnyilvánítás

Köszönettel tartozunk Dr. Csendes Tibornak, a Szegedi Tudományegyetem docensének ötleteiért és a cikk megírásához nyújtott segítségéért. Köszönet illeti Baker Kearfott-ot és George Corliss-t, a GlobSol megismeréséhez nyújtott segítségükért, valamint Diamant Tibort és Csúri Miklóst a Szegedi Tudományegyetem Számítóközpontjának munkatársait, akik a program IBM RS/6000-es környezetben való használatát lehetővé tették számunkra.

Hivatkozások

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New York, 1983.

[2] P. Chin, R.M. Corless, and G.F. Corliss. Globsol case study: Inexact greatest common denominators. Technical Report, Marquette University Department of Mathematics, Statistics, and Computer Science, Milwaukee, Wisc., 1998.
Elektronikus változat: `http://www.mscs.mu.edu/~globsol/public_html/Papers/wn_gcd1.ps`

[3] P. Chin, R.M. Corless, and G.F. Corliss. Optimization strategies for the floating-point GCD. In *Proc. Internat. Symp. on Symbolic and Algebraic Comput. (ISSAC '98) O. Gloor, Ed.*, pages 228–235, New York, 1998. ACM Press.
Elektronikus változat: `http://www.mscs.mu.edu/~georgec/Pubs/chapt.html#1998a`

[4] R.M. Corless, P.M. Gianni, and B.M. Trager. The singular value decomposition for polynomial systems. In *Proc. Internat. Symp. on Symbolic and Algebraic Comput. (ISSAC) A. H. M. Levelt, Ed.*, pages 195–207, New York, 1995. ACM Press.

[5] G.F. Corliss and R.B. Kearfott. Rigorous global search: Industrial applications. In T. Csendes, editor, *Developments in Reliable Computing*. Kluwer, Dordrecht, 1999.

[6] T. Csendes. Nonlinear parameter estimation by global optimization – efficiency and reliability. *Acta Cybernetica*, 8:361–370, 1989.

[7] T. Csendes. New subinterval selection criteria for interval global optimization. *Journal of Global Optimization, közlésre benyújtva.*
Elektronikus változat: `http://www.inf.u-szeged.hu/~csendes/pfconv.ps.gz`

[8] E.R. Hansen. *Global Optimization Using Interval Analysis.* Marcel Dekker, Inc., New York, 1992.

[9] R.B. Kearfott. An interval branch and bound algorithm for bound constrained optimization problems. *Journal of Global Optimization*, 2:259–280, 1992.

[10] R.B. Kearfott. A Fortran 90 environment for research and prototyping of enclosure algorithms for nonlinear equations and global optimization. *ACM Trans. Math. Software*, 21(1):63–78, 1995.

[11] R.B. Kearfott. *Rigorous Global Search: Continuous Problems.* Kluwer, Dordrecht, 1996.

[12] G. Mayer. Epsilon–inflation in verification algorithms. *J. Comput. Appl. Math.*, 60:147–169, 1994.

[13] M. Metcalf and J. Reid. *Fortran 90 Explained.* Oxford University Press, Oxford, 1993.

[14] A. Neumaier. *Interval Methods for System of Equations.* Cambridge Univerity Press, Cambridge, England, 1990.

[15] R. Peikert, D. Würtz, M. Monagan, and C. de Groot. Packing cirlces in a square: A review and new results. In P. Kall, editor, *System Modelling and Optimization, volume 180 of Lecture Notes in Control and Information Sciences*, pages 45–54. Springer-Verlag, Berlin, 1992.

[16] Szabó Péter Gábor. Optimális körelhelyezések a négyzetben. *Polygon, közlésre elfogadva.*
Elektronikus változat: `http://www.inf.u-szeged.hu/~pszabo/polygon.ps.gz`

A. Installálás

Az ebben a cikkben ismertetett globális optimalizáló program Fortran 90 forráskódja szabadon hozzáférhető a

`http://interval.louisiana.edu/GlobSol/download_GlobSol.html`

címen. A programot eredetileg UNIX munkaállomásra tervezték, de használható személyi számítógépeken Linux, illetve MS-DOS alatt is. Az alapvető standard beállítások elsősorban a SunOS-t támogatják, de apró módosításokkal más UNIX rendszeren is működik. A fenti címről letölthető tömörített állomány mérete kb. 1,1 Mbyte, a teljes forráskód pedig 5,5 Mbyte. A fordításhoz szükséges kb. 20 Mbyte szabad hely és tapasztalat szerint működik 16 Mbyte memóriával is. Használatához szükségünk van egy Fortran 90 fordítóprogramra. Tudomásunk szerint semelyik operációs rendszerhez sincs olyan ingyenes Fortran 90 fordító, amellyel a GlobSol sikeresen lefordítható és használható. Másrészt megemlítjük, hogy különböző operációs rendszerekhez ingyenes próbaverziókat találhatunk a Fujitsu Ltd. honlapján: `http://www.fujitsu.co.uk/devtools.htm`, valamint a Numerical Algorithms Group honlapján: `http://www.nag.co.uk/nagware/NP/NP_trial.html`. Ezekkel a fordítókkal sikeresen le tudtuk fordítani a GlobSol-t, illetve tapasztalatokat szerezhettünk még egy AIX 4.3.2 operációs rendszeren működő (2.2(261) verziójú) NAGWare f90 fordítóval kapcsolatban is. (Ezen a rendszeren működik az ismertetett GlobSol szerver is.)

A következő 4 fordítóhoz külön installáló fájlokat találunk, de további fordítók is használhatók, amennyiben megváltoztatjuk a `makefile` megfelelő sorait. Az installáló (kicsomagoló) fájlokat szintén a GlobSol Download oldalán érhetjük el.

- `unpack_sun_sun` — SunOS Sun Fortran fordítóhoz
- `unpack_sun_nag` — SunOS NAG Fortran fordítóhoz
- `unpack_dos_dec.bat` — MS-DOS DEC Fortran fordítóhoz
- `unpack_dos_nag.bat` — MS-DOS NAG/Salford Fortran fordítóhoz

Amennyiben Linux operációs rendszerre telepítjük, az `unpack_sun_nag` fájlt használhatjuk.

Az `unpack_dos_dec.bat` és az `unpack_dos_nag.bat` batch fájlok használják a WinZip parancssori változatát, a `wzunzip` parancsot, de ezt kicserélhetjük `unzip` vagy `pkunzip` parancsra, amennyiben ez szükséges. Ha egyik sem áll rendelkezésre, a WinZip parancssori segédeszközei elérhetőek a WinZip honlapján a `http://www.winzip.com/` címen. Az MS-DOS DEC Fortran fordítóhoz szükség van GNU make-re, amit letölthetünk az `ftp://ftp.gnu.org/pub/gnu/make/` címről, illetve a GlobSol Download oldaláról is. Az `unpack_sun_sun` és az `unpack_sun_nag` shell-script fájlok az `unzip` parancsot használják, de ez általában minden UNIX rendszerű gépen elérhető.

Miután lefuttattuk a megfelelő `unpack` fájlt, meg kell változtatnunk a `makefile`-ban a 116. sort, vagyis írjuk be a GlobSol könyvtárunk elérési útját, illetve ugyanezt írjuk be az `fmake` fájlba is. Ha nem a megadott installáló fájlhoz tartozó fordítót használjuk, akkor a `makefile` 150-200. sorában az `F90`, `F77`, illetve az `LNK` deklarációkat kell még megváltoztatnunk. A Fujitsu fordító esetén ez a következőképpen néz ki:

```
F90 = frt -O0 -X9 -Am -M$(MODULEDIR) -I$(MODULEDIR) -c
F90 = frt -O0 -X9 -Am -M$(MODULEDIR) -I$(MODULEDIR)
F90 = frt -O0 -X9 -c
```

a NAGWare Fortran 90 fordítóhoz pedig a következő bejegyzések szükségesek:

```
F90 = f90 -O -I$(MODULEDIR) -c
LNK = f90 -O -I$(MODULEDIR)
F77 = f90 -O -c
```

Ezek után adjuk ki a `make` parancsot. A fordítás hosszú időt vehet igénybe. Amennyiben problémák merülnek fel a fordítás során, probálkozzunk a következőkkel:

- Ha a `dsecnd.f90` fájl fordításánál akadunk el, akkor az azt jelenti, hogy a mi fordítónk más időfüggvényeket használ. Ha ismerjük a megfelelő függvények neveit, cseréljük ki a fájlban feltételezett függvényeket. Ha nem tudjuk, vagy nem sikerül úgy se lefordítani, akkor egyszerűen a `GlobSol.zip` fájlunkból tömörítsük ki az általános `dsecnd.generic.f90` fájlt, majd nevezzük át `dsecnd.f90`-re. Ez egy "nem csinál semmit" fájl, tehát nem tudjuk majd, hogy esetenként mennyi CPU időt használtunk, de ettől a programunk még tökéletesen fog működni.
- Amennyiben a `sun_stubs.f90` lefordítása jelent gondot, itt is először próbálkozhatunk a fordítónk IEEE lebegőpontos számábrázolást beállító függvényeinek beírásával. Azok ismerete nélkül a szintén az eredeti `GlobSol.zip` fájlba tömörített `sun_stubs.generic.f90` fájlt nevezzük át a GlobSol könyvtárunkban `sun_stubs.f90`-re. Így a fordítással nem lehet probléma, hiszen ez is egy "nem csinál semmit" fájl.
- Megjegyezzük, hogy ha a `unix_make_library` (vagy `make_library`) futása során áll le a fordítónk hibaüzenettel, akkor az általában az előző két fájl hibájából adódik.

Ha sikeresen lefordítottuk a GlobSol-t, akkor próbáljuk ki a korábban említett példára.

Tóth Boglárka és Vinkó Tamás, programtervező matematikus hallgatók, SZTE Informatikai Tanszékcsoport, Szeged, Árpád tér 2. Email: {h531733, h531661}@stud.u-szeged.hu