BEVEZETÉS A GLOBÁLIS OPTIMALIZÁLÁSBA

Csendes Tibor

Szeged, 2012.

Lektorálta:

(Még senki, de a következőket tervezem felkérni: Csallner András Erik, Fülöp János, Pintér János.)

Előszó

A jelen jegyzet[1] a Szegedi Tudományegyetemen 2002-től tartott Bevezetés a Globális Optimalizálásba című tárgy anyagát tartalmazza. A tárgy heti két óra előadást és egy óra gyakorlatot jelent. Az utóbbit számítógépes teremben, aktív gyakorlással kell tölteni.

A tárgy az Operációkutatás és alkalmazott matematika szakirány része. Más hallgatók speciálkollégiumként vehetik fel. Közvetlenül kapcsolódik a Nemlineáris optimalizálás tárgyhoz (ami szintén az említett szakirány része). Ezek felvételének ideális sorrendje: Nemlineáris optimalizálás, majd Globális optimalizálás.

A jegyzet a lineáris algebra, kalkulus és numerikus matematika tárgyakra támaszkodik. Aktuális változata egy része, a hozzá kapcsolódó feladatok, gyakorlatok és adataik elérhetők a

`http://www.inf.u-szeged.hu/~csendes/go.pdf`

címen.

A tárgy olyan tudást kíván adni, amely elegendő egyszerűbb optimalizálási munkák elvégzéséhez, és amelyet önálló gyakorlással továbbfejlesztve egy-egy szakterület teljes globális optimalizálási feldolgozását is végre lehet hajtani. Mivel az érintett programcsomagok gyakran változnak, az anyag főleg az állandó vagy kevésbé változó ismereteket tartalmazza.

A rendelkezésre álló rövid idő (kb. 14×3 óra) nem elég a globális optimalizálás teljes körű tárgyalására, ezért a legfontosabb definíciókat, összefüggéseket és az elméletet az érintett optimalizálási eljárások tárgyalása előtt csak a feltétlenül szükséges terjedelemben ismertetem. A teljesen önálló optimalizáláshoz ez persze nem elegendő. Ennek ellenére bízom benne, hogy a tárgyalt anyag segít a leggyakoribb hibákat elkerülni, és a viszonylag könnyen kezelhető programok segítségével (támaszkodva a mind több esetben rendelkezésre álló readme fájlokra, súgó, tanácsadó varázslókra) önálló munkával is lehetséges a további szükséges eljárások megismerése. A teljes itt közreadott anyag kicsit több, mint amit egy féléves kurzusban át lehet adni, ez némi rugalmasságot követel az előadótól, illetve a gyakorlatvezetőtől.

További cél segítséget nyújtani a globális optimalizáláshoz olyanok számára is, akik ezt a hagyományos képzés keretében nem tanulták. Így a jegyzet alapján az egyszerűbb feladatok esetén az olvasó elegendő útmutatást kap ahhoz, hogy a feladatát úgy fogalmazza meg, illetve írja át, hogy az a rendelkezésre álló szoftverrel hatékonyan megoldható legyen.

A Globális optimalizálás a korábbi tanulmányokból a Lineáris algebra, Numerikus matematika, Operációkutatás, Kombinatorikus optimalizálás és a Nemlineáris optimalizálás tárgyakra támaszkodik. A Globális optimalizálás tárgyat ideális esetben a Nemlineáris optimalizálás után jó felvenni. Az Operációkutatás és alkalmazott matematika szakirányban mindkettő rendszeresen meg lesz hírdetve.

A jelen jegyzet a korábbi speciálkollégiumok és doktori kurzusok során csiszolódott anyagot is tartalmazza. Itt mondok köszönetet korábbi hallgatóimnak és munkatársaimnak a jegyzet

[1] Minden megjegyzést szívesen látok és előre is köszönök, különösen, ha hibákra hívják fel a figyelmem. Az email címem: `csendes@inf.u-szeged.hu`

létrejöttéhez, illetve a javításához nyújtott segítségükért. Várom a további véleményeket és javaslatokat is.

Szeged, 2012. szeptember

a szerző

Jelölések

Itt a legfontosabb, szinte mindig a megadott formában használatos jelöléseket adjuk meg, de ezektől helyenként — ahol a tárgyalás ezt megköveteli — eltérhetünk.

AD	automatikus differenciálás
α	intervallum sorozatok konvergencia rendje
$c(X)$	a központi alak alappontja az X intervallumban
$f(x)$	a célfüggvény
$f(X)$	a célfüggvény értékkészlete az X intervallumon
$F(X)$	a célfüggvény befoglaló függvénye az X intervallumon
$F'(X)$	az egyváltozós függvény deriváltja befoglaló függvénye
$f(x^*)$	a globális minimum értéke
$\tilde{f}$	intervallumos optimalizálási módszerben a globális minimum aktuális felső becslése
$g_i(x)$	egy feltételi függvény
$H(x)$	a célfüggvény Hesse-mátrixa
$H(X)$	a célfüggvény Hesse-mátrixa befoglaló függvénye
$\mathbb{I}$	a kompakt intervallumok halmaza
$\mathbb{I}^n$	az n-dimenziós kompakt intervallumok halmaza
$m(X)$	az X intervallum középpontja
$\nabla f(x)$	a célfüggvény gradiense
$pf^*(X)$	a RejectIndex algoritmus paraméter intervallumos optimalizálási eljárásban
$\mathbb{R}$	a valós számok halmaza
$\mathbb{R}^n$	az n-dimenziós valós vektorok halmaza
$x, y, ...$	változók
$X, Y, ..., A, B, ...$	intervallumok vagy mátrixok
$\underline{X}, \overline{X}$	az X intervallum alsó és felső korlátja, $X = [\underline{X}, \overline{X}]$

x^*	a globális minimumpont
$w(X)$	az X intervallum szélessége

1. fejezet

Bevezetés

Optimalizálási feladatok a mindennapi élet számos területén előfordulnak, főleg a mérnöki, gazdasági alkalmazások területén, de természetesen a tudományos kutatásban is. Ide tartoznak azok a problémák, amelyekben a kérdésfeltevés a következő sémát követi: mikor lesz minimális egy mennyiség, melyik esetben optimális egy beállítás, milyen paraméterek mellett lesz maximális egy együttható értéke stb. Gyakorlati esetben a hasonló kérdések úgy hangzanak például, hogy: mikor lesz a legnagyobb a profit, ha különben adott termelési feltételeknek elegetteszünk, mely esetben lesz minimális a költsége egy beruházásnak, miközben előírt mennyiséget gyártunk, és a lehetséges megoldásainkat feltételek korlátozzák.

Az optimalizálás a matematika, azon belül az operációkutatás, vagy más szempontból a numerikus matematika része. Érintkezik a számítástudománnyal, és van számítástechnikai vetülete is. Az operációkutatás mint önálló tudományterület a múlt század közepétől létezik, és számos közös részterülete van az alkalmazott matematikával. Az OR Today című szakmai folyóirat egy korábbi felmérése szerint az Amerikai Egyesült Államokban az operációkutatási szakemberek álláskilátása volt a negyedik legjobb a vizsgált nagyon sok szakma közül.

A finomabb kategorizálás szerint a globális optimalizálás a nemlineáris optimalizálás, vagy más néven a matematikai programozás témaköréhez tartozik. Az utóbbi név a lineáris programozás analógiájára azt a területet jelöli, amikor az optimalizálandó függvény, vagy a feltételi halmazt kijelölő függvények valamelyike nem lineáris.

Az egyik, Hans-Paul Schwefel professzortól hallott történet szerint a SIEMENS cég számára az atomerőművek fűtőelemeinek elhelyezését optimalizálták egy (később tárgyalt) ún. evolúciós algoritmussal. A mérnökök által gyakorlati megfontolásokon és szimmetria elven alapuló korábbi megoldáson kb. egy százalékot sikerült javítani a hatékonyság szempontjából. Ennek ellenére nem tudható, hogy mi lenne az optimális elhelyezés, és az sem, hogy a jelen megoldás a célfüggvény értékében mennyire tér el attól. Mégis, az új elhelyezési javaslat akkora megtakarítást jelentett, hogy a kutató intézete német Márkában is 9 jegyű támogatásban részesült.

Egy másik hasonló, nagy volumenű optimalizálási feladatban egy nagy európai multi számára kellett a telephelyek optimális elhelyezését meghatározni. Jellemző módon a feladat modelljének felállítása nem volt triviális, és a piacon kapható kereskedelmi optimalizáló szoftver nem volt alkalmas a feladat közvetlen megoldására. A talált közelítő megoldás kb. 7%-os megtakarítást jelentett, miközben az érintett vállalkozás éves pénzforgalma Euroban is több százmilliós volt.

A 2000-ben Budapesten rendezett EURO nevű operációkutatási konferencián (a konferencia internetes vendégoldala a `http://www.sztaki.hu/conferences/euro17/` címen érhető el) George L. Nemhauser professzor Large-Scale Discrete Optimization in Airline Scheduling című plenáris előadásával arról számolt be, hogy az amerikai légitársaságok optimalizálási feladatai (pl. a személyzet beosztása, ütemezési, hozzárendelési és szállítási feladatok) az évi több milliárd

Dolláros költségek százalékos nagyságrendjét is megtakaríthatják.

Saját esetünkben a KÉSZ Kft. számára kerestünk egy olyan gyors algoritmust, amely képes a napi építési feladatokhoz meghatározni azt, hogy a leszabandó munkadarabokat milyen sorrendben és milyen orientálással vágják ki a raktáron levő különböző profilú acél rudakból úgy, hogy a veszteség minimális legyen. A teljes leszámolás kb. annyi eset megvizsgálását igényelte volna, ahány elemi részecske van az univerzumban (és emiatt nyilván kivitelezhetetlen lett volna). A javasolt heurisztika a részfeladatok nagy részén garantáltan optimális megoldást szolgáltatott, a többin pedig jobb eredményeket tudott adni, mint a korábban használt eljárás. Az ilyen jellegű nyersanyagok felhasználásának éves volumene az érintett vállaltnál milliárdos nagyságrendű.

Ezen példák mindegyikében kimondatlanul is nyilván a legjobb megoldás megtalálásában voltunk érdekeltek, és nem csak egy olyant keresünk, amely egy szűk környezetében a legjobb értéket adja. Emiatt ezek is a globális optimalizálás témakörébe tartoznak. Ennek ellenére a globális optimalizálási feladatok leggyakoribb kezelési módja az ignorálás, tehát a felhasználó sokszor megelégszik egy helyi kereső eljárás által adott közelítő megoldással — ami nyilvánvaló módon mind a célfüggvényértékben, mind a talált optimalizálandó változók értékében tetszőlegesen messze lehet a valódi megoldástól.

Másrészt nyilván vannak olyan feladatok is, amelyekben a globális optimum ismerete nem nélkülözhetetlen — feltéve, hogy annak értékéhez elegendően közel tudunk jutni. Így a legtöbb kis méretű, egyszerű gyakorlati feladatban a megoldás egy ezrelék relatív pontosságú ismerete az alkalmazás szempontjából elegendő lehet. Ilyen esetben tehát az abszolút optimum meghatározására az esetleg kilátástalanul hosszú számítás már nem fizetődik ki.

A jegyzet és a tárgy fő rendező elve a rendelkezésre álló információ típusa szerinti osztályozás lesz: tehát az azonos jellegű, a feladatra vonatkozó ismeretek alapján egy csoportba sorolt módszereket együtt tárgyaljuk majd, mégpedig a legszegényesebb információ forrástól a leginformatívabb felé haladva.

2. fejezet

Alapfogalmak

Ebben a fejezetben azokat az optimalizálási alapfogalmakat és összefüggéseket foglaljuk össze, amelyeket a továbbiakban használni fogunk. Ezek nagyrészt a nemlineáris optimalizálás témakörébe tartoznak.

A nemlineáris optimalizálás alapfeladata a következő:

$$\min_{x \in X \subseteq \mathbb{R}^n} f(x) \tag{1}$$

ahol $f(x) : \mathbb{R}^n \to \mathbb{R}$ valós függvény, és X azon n-dimenziós pontok halmaza, amelyekre $g_i(x) = 0$ és $h_j(x) \leq 0$, ahol $g_i(x), h_j(x) : \mathbb{R}^n \to \mathbb{R}$ valós függvények, $i = 1, 2, \ldots, m_1$, és $j = 1, 2, \ldots, m_2$ valamely nemnegatív egész m_1, m_2 számokra.

A nemlineáris optimalizálási feladatok típusait a feladatban szereplő függvények tulajdonságai (folytonosság, deriválhatóság, konvexitás, ...) határozzák meg. Ezek döntik el azt is, mely megoldó algoritmusok alkalmazhatók.

Az (1) általános nemlineáris optimalizálási feladat fontosabb speciális esetei:

Feltétel nélküli optimalizálási feladat: ha $m_1 = m_2 = 0$, tehát a lehetséges megoldások halmaza a teljes n-dimenziós valós tér.

Korlátozott feladat: ha $m_1 = 0$, és csak $a_i \leq x_i \leq b_i$ alakú feltétel van, ahol $a_i, b_i \in \mathbb{R}$, $i = 1, 2, \ldots, n$.

Négyzetösszeg alakú célfüggvényű feladat: ha $f(x)$ felírható $\sum_{j=1}^{m}(f_j - f_{mod}(j, x))^2$ alakban, ahol m pozitív egész, f_j valós szám ($j = 1, 2, \ldots, m$), és $f_{mod}(j, x)$ az ún. modell-függvény.

Egészértékű optimalizálási feladat: ha a lehetséges megoldások halmaza az egész számokból álló n-dimenziós vektorok halmaza.

A célfüggvény tulajdonságai szerinti osztályok: konvex, konkáv, sima vagy kétszer folytonosan differenciálható, nem-differenciálható, kvadratikus, Lipschitz-folytonos etc.

A megadott optimalizálási feladatosztályok nagyon redundánsak, azaz több osztály minden feladatához lehet találni olyan, másik osztálybeli feladatot, hogy a két feladat szélsőértékei és azok helye megegyeznek. Mutassunk ki ilyen ekvivalenciát! (minimalizálás - maximalizálás, konvex - konkáv, legkisebb négyzetes feladatok - általános nemlineáris feladatok, feltételes - feltétel nélküli feladatok stb.)

Néhány alapvető definíció:

helyi minimumpont az x^* pont, ha van olyan $N(x^*)$ környezete, hogy minden $x \in N(x^*)$ pontra $f(x) \geq f(x^*)$.

helyi minimum az x^* helyi minimumpontban felvett célfüggvény-érték, $f(x^*)$.

szeparált helyi minimumpont x^*, ha helyi minimumpont, és nem torlódási pontja helyi minimumpontoknak.

globális minimumpont egy helyi minimumpont, ha a lehetséges megoldások X halmazán minden x pontra $f(x) \geq f(x^*)$.

globális minimum az x^* globális minimumpontban felvett célfüggvény-érték, $f(x^*)$.

nyeregpont az olyan pont, amelyben a gradiens nulla, de minden környezetében van nála kisebb és nagyobb célfüggvény-értéket felvevő pont is.

egy helyi minimumpont vonzáskörzete azon folytonos görbéknek a lehetséges megoldások halmazába eső pontjainak halmaza, amelyek egy helyi minimumpontban végződnek, és amelyek mentén a célfüggvény értéke monoton csökken.

szeparálható célfüggvény az $f(x)$, ha felírható $f_1(x) + f_2(x)$ alakban, és $f_1(x)$-nek van olyan globális minimumpontja, amely globális minimumpontja $f_2(x)$-nek is (és emiatt $f(x)$-nek is).

Megjegyzések:

1. A helyi minimumpont meghatározásában talán a kisebb jel természetesebb lenne, de ekkor az analízisbeli tétel (folytonos függvény korlátos intervallumon, ill. korlátos tartományon felveszi szélsőértékeit) nem teljesülne pl. a konstans függvényekre.

2. A gyakorlati feladatok szempontjából az olyan helyi minimumpontok a fontosak, amelyeknek vonzáskörzete elég nagy. A túl kicsi vonzáskörzetű minimumpont a gyakorlatban beállíthatatlan, és érzékeny az input adatok pontosságára.

3. A nyeregpont azért fontos, mert a helyi kereső eljárások gyakori megállási feltétele az, hogy a gradiens nulla legyen. Ilyen esetben illik azt is ellenőrizni, hogy a talált pont nem nyeregpont-e.

4. A szeparálható célfüggvények sajnos ritkák a nemlineáris optimalizálási feladatokban, ráadásul ennek a tulajdonságnak a felismerése, és az ellenőrzése is hasonló nehézségű, mint az eredeti feladat.

5. Fontos jellemzője egy nemlineáris optimalizálási feladatnak, hogy milyen a globális optimum vonzáskörzete és a lehetséges megoldások halmaza mértékének aránya. Gyakorlati feladatokon tipikus az 5–30% érték erre az arányra. Egy százalék alatt már kifejezetten nehéznek számít a feladat, de találkoztunk 0.001% alatti értékkel is.

A feladat kitűzésekor viszonylag sok információ áll rendelkezésre, mégis a megoldó algoritmusok ebből csak nagyon kevésre támaszkodhatnak. Jellemző az a helyzet, hogy a megoldás "látható", mégis a program nem tudja megtalálni. Osztályozzuk az optimalizálási módszereket aszerint, hogy milyen információt használnak fel a keresés során:

- Talán a legegyszerűbb eset, amikor az algoritmus csak a célfüggvényértékre támaszkodhat általa megválasztott pontokban. Ezeket az eljárásokat direkt keresőknek hívják. Ekkor a felhasználónak egy olyan szubrutint kell mellékelnie, amely a paramétertérbeli pont ismeretében kiszámítja a célfüggvény értékét. Ehhez nem is kell feltétlenül ismerni a függvény pontos alakját, hiszen az függhet pl. aktuális mérési adatoktól, a véletlentől stb. Gyakori eset, hogy csak egy a célfüggvény meghatározására szolgáló eljárás áll rendelkezésre, de a függvény konkrét alakja explicit módon nem jelenik meg. Ennek a feladatosztálynak a jellemző módszerei a rácsmenti keresés, a Monte Carlo módszer különböző változatai és a véletlen séta algoritmusok. Ezek a primitív, robusztus módszerek rendszerint rossz hatékonyságúak, megbízhatatlanok, de nagyon sok feladatra alkalmazhatók. Csak az egyenesmenti keresés használja ki a célfüggvény esetleg meglévő monotonitását egy adott irányban.

- Széles körben használnak olyan módszereket, amelyek a célfüggvényen túl annak első deriváltjának (gradiensének) értékét is használják. Ezeket elsőrendű, vagy gradiens módszereknek hívják. Ide tartoznak a legmeredekebb lejtő-, és konjugált gradiens módszerek. Angol nevüket (hill climbing methods) onnan kapták, hogy a gradiens a leggyorsabb növekvést mutató irányt jelöli. A gradiens ismeretében persze egyszerűsödik a feladat annyiban, hogy a direkt módszereknek ezt az irányt is keresni kellett. A hatvanas években népszerűek voltak ezek az eljárások, annak ellenére, hogy a gradienst kiszámító szubrutinok megadása sokszor nehéz. A legtöbb numerikus szubrutin-könyvtárban megtalálhatók, a Fletcher és Powell nevével jelzetteket érdemes keresni. Ezen módszerek kritikus része az egyenesmenti keresés, ennek hatékonysága sokat javíthat vagy ronthat az összteljesítményen. Érdemes megvizsgálni a gradiens módszerek működését a bosszantásukra kitalált Rosenbrock (vagy banán) függvényen: $f(x, y) = (1 - y)^2 + 100(y^2 - x)^2$. Az optimálisnak tűnő negatív gradiens irányok mentén az algoritmus oda-vissza hintázik a meredek völgy falain, és igen keveset halad a minimumpont felé. Ezen a jelenségen segít valamit a konjugált gradiens módszer. Ez utóbbi elfogadható hatékonyságú a folytonosan deriválható függvényeken (bizonyos esetekben szuperlineáris konvergencia is elérhető). Bizonyos optimalizáló szubrutinok lemondanak a gradienst számító rutinról azon az áron, hogy a gradiens helyett annak numerikus becslésével számolnak (kényelem vs. kerekítési hibák).

- A harmadik, ma leggyakrabban használt algoritmus-osztály a másodrendű-, vagy Newton-módszerek osztálya. Ezek a célfüggvény és a gradiens mellett még a Hesse-mátrix értékére is támaszkodnak. A nevük onnan származik, hogy lényegében a nemlineáris függvény zérushelyének megkeresésére szolgáló Newton algoritmust alkalmazzák a célfüggvény deriváltjára (gradiensére). Más szóval, egy adott pontban előállítják a célfüggvény kvadratikus polinom modelljét, majd annak minimumába lépnek. Nagyon jó (kvadratikus) konvergencia sebességet lehet bizonyítani sima függvényekre, és gyakorlati feladatokon is messze hatékonyabbak, mint az előzőek. Érzékenyek viszont az indulópont megválasztására, szeretnek a helyi minimumpont közeléből indulni. Azokat az idetartozó módszereket, amelyek nem igénylik a Hesse mátrixot, mert azt numerikusan közelítik, quasi-Newton módszereknek nevezik. A Newton módszerek használata előtt győződjünk meg arról, hogy a célfüggvény tényleg kétszer folytonosan deriválható. Ennek teljesülése nélkül meglepően rossz, értelmetlen megoldásokat kapunk.

Programozási feladat:

$\star$ Írjunk egyszerű szubrutint a gradiens segítségével való optimalizálásra, és teszteljük!

2.1. Egy dimenziós feladatok

Tekintsük a következő definíciókat, és tegyük fel, hogy $f(x)$ egy valós függvény valamely I intervallumon (ez lehet véges vagy végtelen, nyitott vagy zárt, vagy csak egyik oldalról zárt).

DEFINÍCIÓ. Azt mondjuk, hogy az $x^* \in I$ pont *globális minimumpontja* az $f(x)$ függvénynek, ha $f(x^*) \leq f(x)$ minden $x \in I$ pontra.

DEFINÍCIÓ. Azt mondjuk, hogy az $x^* \in I$ pont *szigorú globális minimumpontja* az $f(x)$ függvénynek, ha $f(x^*) < f(x)$ minden $x \in I$, $x \neq x^*$ pontra.

DEFINÍCIÓ. Azt mondjuk, hogy az $x^* \in I$ pont *helyi minimumpontja* az $f(x)$ függvénynek, ha létezik olyan δ pozitív valós szám, hogy $f(x^*) \leq f(x)$ minden olyan $x \in I$ pontra, amelyre $x^* - \delta < x < x^* + \delta$.

DEFINÍCIÓ. Azt mondjuk, hogy az $x^* \in I$ pont *szigorú helyi minimumpontja* az $f(x)$ függvénynek, ha létezik olyan δ pozitív valós szám, hogy $f(x^*) < f(x)$ minden olyan $x \in I$ pontra, amelyre $x^* - \delta < x < x^* + \delta$ és $x^* \neq x$.

DEFINÍCIÓ. Azt mondjuk, hogy az $x^* \in I$ pont *stacionárius pontja* az $f(x)$ függvénynek, ha létezik $f'(x^*)$, és $f'(x^*) = 0$.

Az előző definíciók nyilvánvaló módosításai adják az $f(x)$-re vonatkozó *globális maximumpont*, *szigorú globális maximumpont*, *helyi maximumpont* és a *szigorú helyi maximumpont* definícióit. A továbbiakban a tárgyalást a minimalizálási feladatokra korlátozzuk, a maximalizálási feladatokra vonatkozó eredmények az $f(x)$ célfüggvény $-f(x)$-el való helyettesítésével adódnak.

A következő tételek az egyváltozós függvények minimalizálására vonatkozó alapvető eredményeket foglalják össze.

1. Tétel. *Tegyük fel, hogy $f(x)$ differenciálható függvény egy I intervallumon. Ha x^* egy helyi minimumpontja $f(x)$-nek, akkor vagy x^* egy végpontja I-nek, vagy $f'(x^*) = 0$.*

BIZONYÍTÁS. Tegyük fel, hogy x^* egy helyi minimumpontja $f(x)$-nek, és x^* nem végpontja az I intervallumnak. A feltevésünk szerint $f'(x)$ létezik, így azt kell megmutatnunk, hogy akkor $f'(x^*) = 0$. Mivel $f(x^*) \leq f(x)$ az x^*-hoz elegendően közeli x-ekre, az $f(x) - f(x^*)$ különbség nemnegatív minden ilyen x-re. Ezek után

$$\frac{f(x) - f(x^*)}{x - x^*} \geq 0 \text{ ha } x^* < x, \text{ illetve } \frac{f(x) - f(x^*)}{x - x^*} \leq 0 \text{ ha } x^* > x$$

— feltéve, hogy x elegendően közeli x^*-hoz. Ezekből $f'(x^*) \geq 0$ és $f'(x^*) \leq 0$ adódik

$$f'(x) = \lim_{x \to x^*} \frac{f(x) - f(x^*)}{x - x^*}$$

alapján, azaz $f'(x^*) = 0$ következik. □

Ha egy függvény stacionárius pontjait meghatároztuk, akkor a következő eredmény alapján eldönthetjük, hogy ezek minimumpontok-e.

2. Tétel. *Tegyük fel, hogy az $f(x)$, $f'(x)$ és $f''(x)$ függvények folytonosak egy I intervallumon, és $x^* \in I$ egy stacionárius pontja $f(x)$-nek. Ekkor*

a, ha $f''(x) \geq 0$ minden $x \in I$ pontra, akkor x^* globális optimumpontja $f(x)$-nek az I intervallumon,

b, ha $f''(x) > 0$ minden $x \in I$ pontra úgy, hogy $x \neq x^*$, akkor x^* szigorú globális minimumpontja $f(x)$-nek az I intervallumon,

c, ha $f''(x^*) > 0$, akkor x^* szigorú helyi minimumpontja $f(x)$-nek az I intervallumon.

BIZONYÍTÁS. Ha $x \in I$, és $x \neq x^*$, akkor a középérték-tétel és az $f'(x^*) = 0$ feltevés alapján

$$f(x) - f(x^*) = \frac{f''(z)}{2}(x - x^*)^2, \tag{2}$$

ahol z szigorúan x és x^* közötti pont. Következésképpen, ha $f''(x) \geq 0$ minden $x \in I$ pontra, akkor $f(x) \geq f(x^*)$ minden $x \in I$ pontra, mivel $(x-x^*)^2/2 \geq 0$ minden $x \in I$ pontra. Ez igazolja az a, állítást, és az érvelés megfelelő átalakítása igazolja a b, állítást. Végül ha $f''(x^*) > 0$, akkor az f'' folytonosságából következik, hogy létezik olyan $\delta > 0$, hogy $f''(x) > 0$ minden $x \in I$-re amelyre $x^* - \delta < x < x^* + \delta$. Ekkor (2)-ből az adódik, hogy $f(x) > f(x^*)$ minden olyan $x \in I$ pontra, amelyre $x \neq x^*$ és $x^* - \delta < x < x^* + \delta$, azaz az x^* pont az $f(x)$ függvény szigorú helyi minimumpontja. □

A maximumok feltételeiben az $f''(x^*) \geq 0$, $f''(x) > 0$ és $f''(x^*) > 0$ feltételeket az a, b, és c, állításokban megfelelően $f''(x^*) \leq 0$-ra, $f''(x) < 0$-ra és $f''(x^*) < 0$-ra kell cserélni. Vegyük észre, hogy a 2. Tételben a globális információk alapján a globális minimumpontra-, helyi információk alapján pedig a helyi minimumpontra vonatkozó állításokat lehet nyerni.

PÉLDA. a, Tekintsük az $f(x) = 3x^4 - 4x^3 + 1$ függvényt. Mivel $f'(x) = 12x^3 - 12x^2 = 12x^2(x-1)$, ezért az $f(x)$ stacionárius pontjai az $x = 0$ és az $x = 1$. Hasonlóan $f''(x) = 36x^2 - 24x = 12x(3x-2)$ miatt $f''(0) = 0$ és $f''(1) = 12$. Ebből következően $x = 1$ egy szigorú helyi minimumpontja $f(x)$-nek, de a 2. Tétel nem mond semmit az $x = 0$ stacionárius pontról. Elemezve $f(x)$ értékeit az $x = 0$ pont körül, azt kapjuk, hogy $x^4 < x^3$ a $0 < x < 1$ pontokra, tehát $f(x) < 1$ az origótól közvetlenül jobbra lévő pontokra, míg $f(x) > 1$ attól közvetlenül balra. Ebből az adódik, hogy $x = 0$ se nem helyi minimumpont, se nem helyi maximumpont, hanem egy inflexiós pontja $f(x)$-nek. Vegyük észre, hogy $\lim_{x\to+\infty} f(x) = +\infty$ és $\lim_{x\to-\infty} f(x) = +\infty$, tehát $f(x)$-nek nincs globális maximumpontja az $\mathbb{R}$ halmazon.

b, Az $f(x) = \ln(1 - x^2)$ függvény az $I = (-1, 1)$ nyitott intervallumon definiált. Mivel $f'(x) = -2x/(1-x^2)$, az $f(x)$ függvénynek csak egy stacionárius pontja van az I intevallumon, és ez az $x = 0$ pont. Ez szigorú globális maximumpontja $f(x)$-nek, hiszen

$$f''(x) = \frac{(1-x^2)(-2) - (-2x)(-2x)}{(1-x^2)^2} = \frac{-2(1+x^2)}{(1-x^2)^2}$$

minden $x \in I$ pontra.

2.2. Többváltozós függvények feltétel nélküli optimalizálása

A következőekben kiterjesztjük eddigi eredményeinket a többváltozós függvényekre az analízis és a lineáris algebra segítségével.

3. Tétel. *Tegyük fel, hogy $f(x)$ egy valós értékű függvény, amelynek első parciális deriváltjai léteznek az $\mathbb{R}^n$ egy D részhalmazán. Ha x^* belső pontja D-nek és helyi minimumpontja $f(x)$-nek, akkor x^* stacionárius pontja $f(x)$-nek, azaz $\partial f/\partial x_i$ értéke az x^* pontban 0 minden $i = 1, 2, \ldots, n$-re.*

2.3. Többváltozós függvények feltételes optimalizálása

Ebben a fejezetben a nemlineáris optimalizálási feladatokra vonatkozó általános, alapjában véve a helyi minimumok meghatározására való legfontosabb eredményeket foglaljuk össze röviden. Ezek tehát a globális minimum megkeresésére vonatkozóan közvetlenül nem adnak segítséget.

2.3.1. Egyenlőség típusú feltételek esete

Ha csak egyenlőség-feltételünk van, akkor az (1) feladat alakja a következő lesz:

$$\begin{aligned} &\min f(x) \\ &g_j(x) = 0 \quad j = 1, 2, \ldots, m, \end{aligned} \tag{3}$$

ahol $x \in \mathbb{R}^n$. Az erre a helyzetre kidolgozott Lagrange módszer szerint a (3) feladat helyett vizsgálhatjuk az ekvivalens

$$\min F(x, \lambda) = \min f(x) - \sum_{j=1}^{m} \lambda_j g_j(x) \tag{4}$$

korlátozás nélküli feladatot. Itt a λ_j értékek az ún. Lagrange-szorzók. Ezek az érzékenységi együtthatóknak felelnek meg, amik azt fejezik ki, hogy f optimális értéke mennyire változik meg, ha a g feltételeket perturbáljuk. Ebből a következő feltételek adódnak az elsőrendű optimalitási feltétel alapján:

$$\frac{\partial F}{\partial x_i} = 0, \ i = 1, 2, \ldots, n,$$

$$\frac{\partial F}{\partial \lambda_j} = 0, \ j = 1, 2, \ldots, m.$$

Vegyük észre, hogy a λ_j szerinti parciális deriválásból mindig az eredeti feltételi egyenleteket kapjuk vissza.

PÉLDA. Tekintsük a következő egyszerű feladatot a Lagrange módszer működésének illusztrálására:

$$\min (x_1 - 2)^2 + (x_2 - 2)^2,$$

$$x_1 - 1 = 0.$$

Lényegében egy (2, 2) középpontú parabolát kell az $x_1 = 1$ egyenes mentén minimalizálni. A megoldás nyilvánvalóan az $(1, 2)$ pontban van, és a célfüggvényérték itt 1.

A Lagrange módszer szerint az optimalizálandó Lagrange függvény az

$$F(x, \lambda) = f(x) - \lambda g(x) = (x_1 - 2)^2 + (x_2 - 2)^2 - \lambda(x_1 - 1).$$

Az elsőrendű optimalitási feltételekből a következő nemlineáris egyenletrendszer adódik:

$$\begin{aligned} 2(x_1-2)-\lambda &= 0,\\ 2(x_2-2) &= 0,\\ x_1-1 &= 0. \end{aligned}$$

Ez az egyenletrendszer most könnyen megoldható, és az eredmény: $x_1 = 1$ (az utolsó sorból), $x_2 = 2$ (a megelőzőből), és végül $\lambda = -2$. Az eljárás előnye nyilván még megoldható, de bonyolultabb egyenletrendszer esetén jön ki. Másrészt a nemlineáris egyenletrendszer megoldása általában is egyszerűbbnek tekinthető, mint a közvetlen feltételes szélsőérték-keresés.

2.3.2. Egyenlőtlenség típusú feltételek esete

Tekintsük ismét a korábban már említett (1) korlátozott alapfeladatot most a következő alakban:

$$\begin{aligned} &\min f(x)\\ &g_j(x) \le 0 \quad j = 1,2,\dots,m. \end{aligned} \tag{5}$$

ahol $f(x) : \mathbb{R}^n \to \mathbb{R}$ és $g_j(x) : \mathbb{R}^n \to \mathbb{R}$ valós, folytonosan differenciálható függvények, $j = 1,2,\dots,m$, valamely nemnegatív egész m számra. Az (1) egyenlőség feltételei mindegyike két ellenkező irányú egyenlőtlenség feltétellel nyilván felírható, tehát a (5) és az (1) feladatosztály megegyezik, csak a felírásban különböznek. Az egyenlőtlenség feltételek közül azokat, amelyek egyenlőséggel teljesülnek az optimum pontban, *aktív feltételek*nek nevezzük.

Erre a feladatra a szükséges (és egyes esetekben elégséges) feltételeket Karush[2], majd tőle függetlenül Kuhn és Tucker adott meg[3]. Ilyen optimalizálási feltételre számos algoritmust építettek, illetve a megállási feltételek ezt tükrözik.

Tekintsük a következő egyszerűsített (nem minden szempontból teljes) levezetést. Először is, az s_j segédváltozók (slack variables) segítségével vezessük vissza az egyenlőtlenség feltételeket egyenlőség típusúakra:

$$g_j(x) + s_j = 0, \quad j = 1,2,\dots,m, \tag{6}$$

$$s_j \ge 0, \quad j = 1,2,\dots,m. \tag{7}$$

Az utóbbi egyenlőtlenségek kivételével csak egyenlőségek korlátozzák a lehetséges megoldások halmazát. Használjuk a Lagrange módszert és a feladat Lagrange multiplikátoros alakját:

$$F(x,\lambda) = f(x) - \sum_{j=1}^{m} \lambda_j (g_j(x) + s_j). \tag{8}$$

Ekkor a Lagrange módszer alapján

$$\lambda_j = \frac{\partial f}{\partial g_j},\ j = 1,2,\dots,m.$$

[2] W. Karush: Minima of Functions of Several Variables with Inequalities as Side Conditions, M.S. Thesis, University of Chicago, 1939.

[3] Kuhn, H.W., and A.W. Tucker (1951): Nonlinear Programming. Proc. 2nd Berkeley Symp. on Math. Stat. Prob., University of California Press, 481-492.

Más szóval a λ_j épp az f változásának aránya a g_j-éhez, tehát a j. feltételi függvényéhez képest. Minimalizálási feladat esetén ez az érték nem pozitív, mert egy aktív g_j feltételi függvényre a $g_j(x) \leq \varepsilon$ feltétel nyilván nem szűkebb lehetséges megoldási halmazt ad. Erre az optimum értéke nem nőhet — amiből (8) alapján $\lambda_j \leq 0$ adódik. Például ha $f(x) = x$ és $g(x) = a - x$, akkor $\lambda = -1$.

A Lagrange módszert követve Minden x^* helyi minimum pontra, és az ehhez tartozó $s = s^*$ és $\lambda = \lambda^*$ értékekre

$$\frac{\partial F(x^*)}{\partial x} = 0,$$
$$\frac{\partial F(x^*)}{\partial \lambda} = 0,$$

és a λ_j előjele miatt

$$\frac{\partial F(x^*)}{\partial s_j} = -\lambda_j \geq 0,\ j = 1, 2, \ldots, m.$$

Elvégezve a deriválást a következő feltételeket kapjuk:

$$\nabla f(x^*) - \sum_{j=1}^{m} \lambda_j \nabla g_j(x^*) = 0,$$

$$g(x^*) + s^* = 0,$$

és

$$\lambda^* \leq 0,$$

ahol $s = (s_1, s_2, \ldots, s_m)^T$. Ha most $\lambda_j < 0$, akkor a Lagrange módszer szerint

$$\frac{\partial f}{\partial g_j} < 0,$$

azaz ekkor a j. feltétel aktív, és $s_j = 0$. Azaz $g_j(x^*) = 0$.

Másrészt, ha $g_j(x^*) < 0$, akkor $s_j > 0$, és a j. feltétel nem aktív. Ekkor

$$\frac{\partial f}{\partial g_j} = 0,$$

tehát

$$\lambda_j = 0.$$

Összefoglalva: ha $\lambda_j < 0$, akkor $g_j(x^*) = 0$, és ha $g_j(x^*) < 0$, akkor $\lambda_j = 0$. Tehát

$$\lambda_j g_j(x^*) = 0,\ j = 1, 2, \ldots, m.$$

Ezzel nagy vonalakban levezettük a következő tételt:

4. Tétel. *Ha az $f(x)$ függvénynek az x^* pont helyi minimumpontja a lehetséges megoldások*

$$R = \{x \mid g_j(x) \leq 0, j = 1, 2, \ldots, m\}$$

halmazán, ahol f és $g_j,\ j = 1, 2, \ldots, m$ folytonosan differenciálható függvények, valamint az R halmaz határa teljesíti a feltétel vizsgálatot, akkor szükségképpen

$$\nabla f(x^*) - \sum_{j=1}^{m} \lambda_j^* \nabla g_j(x^*) = 0, \tag{9}$$

$$g_j(x^*) \leq 0, \tag{10}$$

$$\lambda_j^* g_j(x^*) = 0,\ j = 1, 2, \ldots, m, \tag{11}$$

$$\lambda_j^* \leq 0,\ j = 1, 2, \ldots, m \tag{12}$$

teljesül valamely $\lambda^* = (\lambda_1^*, \lambda_2^*, \ldots, \lambda_m^*)^T$ *valós vektorra.*

A tételben említett feltétel vizsgálat (constraint qualification) ahhoz szükséges, hogy kiszűrjünk olyan eseteket, amikor a lehetséges megoldások határa extrém jellege miatti szélsőérték nem felel meg az adott feltételeknek. Így egyedülálló lehetséges megoldási pontokban akkor is szélsőértéke van a célfüggvénynek, ha az a tétel feltételeit nem teljesíti. Gyakorlati feladatokban az ilyen esetek szerencsére ritkák.

PÉLDA. Tekintsünk ismét egy egyszerű, áttekinthető feladatot:

$$\min f(x) = x_1^2 + x_2^2,$$

$$3 - x_2 \leq 0,$$

$$x_2 - x_1 \leq 0.$$

Ez az origó körüli egyszerű parabola minimumának megkeresését jelenti, miközben a feltételek azt írják elő, hogy az elfogadható pontok x_2 koordinátája legalább 3 kell hogy legyen, és a lehetséges megoldások mind az $x_2 = x_1$ egyenes alatt vannak. A korlátozás nélküli feladatnak nyilván az origó lenne a megoldása, és a minimum értéke nulla lenne. A korlátozott feladat lehetséges megoldásainak halmaza ezt azonban nem engedi meg.

Tekintsük a Karush-Kuhn-Tucker feltételeket a feladatunkra:

$$\nabla_1 f(x^*) - \sum_{j=1}^{m} \lambda_j^* \nabla_1 g_j(x^*) = 2x_1^* + \lambda_2^* = 0,$$

$$\nabla_2 f(x^*) - \sum_{j=1}^{m} \lambda_j^* \nabla_2 g_j(x^*) = 2x_2^* + \lambda_1^* - \lambda_2^* = 0,$$

$$g_1(x^*) = 3 - x_2^* \leq 0,$$

$$g_2(x^*) = x_2^* - x_1^* \leq 0,$$

$$\lambda_1^* g_1(x^*) = \lambda_1^*(3 - x_2^*) = 0,$$

$$\lambda_2^* g_2(x^*) = \lambda_2^*(x_2^* - x_1^*) = 0,$$

$$\lambda_j^* \leq 0,\ j = 1, 2.$$

A négy egyenlőség feltétel a négy ismeretlen optimális értékre most épp megoldható (bár nemlineáris) egyenletrendszert ad:

$$2x_1^* + \lambda_2^* = 0,$$

$$2x_2^* + \lambda_1^* - \lambda_2^* = 0,$$

$$\lambda_1^*(3 - x_2^*) = 0,$$

$$\lambda_2^*(x_2^* - x_1^*) = 0.$$

Az első kettőből kifejezhetők a λ_j^* értékek: $\lambda_2^* = -2x_1^*$, és $\lambda_1^* = -2x_1^* - 2x_2^*$. A maradék két egyenlet ezután:

$$(-2x_1^* - 2x_2^*)(3 - x_2^*) = 0,$$

$$-2x_1^*(x_2^* - x_1^*) = 0.$$

Az utóbbi egyenletnek két megoldása van: vagy $x_1 = 0$ (és akkor az előző egyenletből x_2 vagy nulla, vagy 3), vagy $x_2 = x_1$ adódik (amiből a $(0, 0)$ és a $(3, 3)$ megoldások adódnak. Ezeket az eredeti feltételekbe helyettesítve csak az utóbbi marad lehetséges megoldásként.

Ezzel megkaptuk a feladatunk optimális megoldását: $x_1^* = 3$, $x_2^* = 3$, $f(x^*) = 18$.

Bár a példánkban a szükséges feltételekre támaszkodva meg tudtuk határozni a megoldást, ez általános esetben nem lehetséges, hiszen nemlineáris egyenletrendszert kell megoldanunk, ami általában nem könnyebb feladat mint egy globális optimalizálási feladat. A Karush-Kuhn-Tucker feltétel jelentősége inkább a feltételes feladatokat megoldó eljárások megállási feltételei megadásában mutatkozik meg.

2.4. Ellenőrző kérdések és gyakorló feladatok

1. Hogyan lehet maximalizálási feladatokat visszavezetni a megadott minimalizálási feladatra?
2. Milyen $f(x)$, $g_i(x)$, és $h_j(x)$ függvények esetén lesz (1) lineáris programozási feladat?
3. Mutassunk olyan optimalizálási feladatot, amely nem illeszthető be a megadott feladatszerkezetbe!
4. Mutassuk meg, hogy az egyenlőség-feltételek kifejezhetők kisebb-egyenlő feltételekkel!
5. Miért nincs szükség nagyobb-egyenlő feltételekre?
6. Mutassuk meg, hogy minden $\mathbb{R}^n$ -beli halmaz megadható a tárgyalt feltételekkel!
7. Adjunk meg olyan feltételt, amelynek csak egy pont tesz eleget!
8. Mutassunk olyan feltétel-rendszert, amely redundáns (azaz elhagyható belőle legalább egy feltétel anélkül, hogy a lehetséges megoldások halmaza megváltozna)! (Fourier módszer)
9. Mutassunk olyan feltétel-rendszert, amely nem redundáns (azaz nem hagyható el belőle feltétel anélkül, hogy a lehetséges megoldások halmaza meg ne változna)!
10. Hogyan használhatók az egyenlőség-feltételek a feladat dimenziójának csökkentésére? Adjunk példát!
11. Mutassunk olyan nemlineáris optimalizálási feladatot, amely a szélsőértékét olyan pontban veszi fel, ahol az illető célfüggvény deriváltja (gradiense) nem nulla!
12. Mutassunk olyan nemlineáris optimalizálási feladatot, amely minimumát olyan pontban veszi fel, ahol a célfüggvény gradiense nem zérus!
13. Fogalmazzuk meg az 1. és 2. definíciót maximalizálási feladatokra!

14. Határozzuk meg azt a furcsa feladatosztályt, amelynek minden helyi minimuma egyben helyi maximum is!

15. Határozzuk meg az egyenes-illesztési feladat optimumát az 1. Tétel alapján! (Az egyenesillesztési feladat az, amikor adott n darab páronként különböző alappontokban ismert függvényértékekhez keressük azt az egyenest, amely ezektől a pontokhoz a legközelebb van az eltérések négyzetösszegét tekintve.)

16. Milyen esetben van az előző feladatnak végtelen sok megoldása?

17. Milyen hatással van az egyenesillesztési feladat optimális megoldására a távolság definíciójának megváltoztatása?

18. Határozzuk meg hasonlóan az adott ponthalmazhoz legjobban illeszkedő kört! (definiáljuk a pont körtől mért távolságát)

19. Próbáljunk ellipszist illeszteni! (Nyitott probléma)

20. Hány minimuma (maximuma) van egy feltétel nélküli konvex (konkáv) optimalizálási feladatnak? Vö. $f(x) = 1/x$, $x \geq 0$.

21. Mutassunk olyan függvényt, amely folytonosan deriválható, de nem sima!

3. fejezet

Direkt kereső módszerek

A "kézzel" való optimalizálás során megállapítottuk, hogy a több-dimenziós térben is igyekszünk egyenesek mentén tisztán látni a célfüggvény változását, és csak amikor az illető egyenes mentén (úgy tűnik) kimerítettük a lehetőségeket, akkor térünk át egy új irányra. Ez a fajta, józan ész diktálta eljárás a lényege az ún. véletlen séta típusú módszereknek. Ezeket akkor szokás megállítani, ha elegendően nagyszámú újabb irány szerint se találtunk az elértnél jobb célfüggvényértékű pontot.

† Milyen legyen az egyenesmenti keresés stratégiája, hogy pl. hosszú monoton szakaszokat ne a szakasz hosszával arányos idő alatt derítsünk fel?

Ennél érezhetően durvább az a megközelítés, amikor egy előre adott finomságú rács minden rácspontjában kiértékeljük a célfüggvényt, majd a legjobb pontot tekintjük a globális minimumpont közelítésének. Ez az ún. rácsmenti keresés nagy-dimenziós feladatok esetén rendkívül számításigényes, mégis egyfajta biztosítékot ad, hogy nem hagyunk figyelmen kívül lényeges tartományokat a lehetséges megoldások halmazából.

† Hogyan növekszik a rácsmenti keresés műveletigénye a feladat dimenziójának növekedésével, ha a rács finomságát nem változtatjuk?

A rácsmenti keresés gyakori alternatívája a véletlen keresés (vagy Monte Carlo módszer). Ez az újabb mintapontot (pl.) egyenletes eloszlással generálja a lehetséges megoldások halmazán. Az előnye, hogy akármikor meg lehet állítani, és a kapott információk a feladatról bizonyos szempontból homogének. Hajlamosak vagyunk azt hinni, hogy a "dimenzionalitás átka" (curse of dimensionality) nem fogja ezt a módszert, pedig ez nem így van. Ugye? Sajnos nehéz értelmes megállási feltételt adni ehhez az algoritmushoz. Gyakran kombinálják hatékony helyi kereső módszerekkel, a véletlen keresés ad azokhoz tűrhető indulópontot. Ennek a módszerosztálynak multiple starts (MS) a neve.

† Hogyan tudunk egyenletes eloszlással pontot generálni egy nem feltétlenül konvex halmazba?
‡ Hogyan lehet egyenletes eloszlással irányt generálni?

És akkor itt egy igazi direkt kereső rutin a véletlen keresők osztályából, próbáljuk megfejteni, bár FORTRAN-ban van írva!

```
      SUBROUTINE LOCAL (M,N,RELCON,MAXFN,X,F,NFEV,R,MIN,MAX)
      IMPLICIT DOUBLEPRECISION (A - H,O - Z)
      DOUBLEPRECISION R(100,15),X(15),X1(15),MIN(1),MAX(1)
      DATA                ZERO,ONEN3,HALF,ONE,TWO/
     *                    0.0,0.001,0.5,1.0,2.0/
C                                     FIRST EXECUTABLE STATEMENT
C                                     INITIAL STEP LENGTH
      H = ONEN3
      DELTF = ONE
      ITEST = 0
      NFEV = 0
      EPS = RELCON
C                                     EVALUATE 100 RANDOM VECTORS
    5 CALL URDMN (R,1500)
      IRNDM = 0
   15 IRNDM = IRNDM+1
      IF (IRNDM.GT.100) GO TO 5
C                                     SELECT A RANDOM VECTOR HAVING NORM
C                                       LESS OR EQUAL TO 0.5
      A = ZERO
      DO 20 I=1,N
      R(IRNDM,I) = R(IRNDM,I)-HALF
   20 A = A+R(IRNDM,I)*R(IRNDM,I)
      IF (A.LE.ZERO) GO TO 15
      A = SQRT(A)
      IF (A.GT.HALF) GO TO 15
C                                     NEW TRIAL POINT
      DO 25 I=1,N
      R(IRNDM,I) = R(IRNDM,I)/A
   25 X1(I) = X(I)+H*R(IRNDM,I)
      CALL FUN (X1,F1,N,M,MIN,MAX)
      NFEV = NFEV+1
      IF (F1.LT.F) GO TO 35
      IF (NFEV.GT.MAXFN) GO TO 50
C                                     STEP IN THE OPPOSITE DIRECTION
      H = -H
      DO 30 I=1,N
   30 X1(I) = X(I)+H*R(IRNDM,I)
      CALL FUN (X1,F1,N,M,MIN,MAX)
      NFEV = NFEV+1
      IF (F1.LT.F) GO TO 35
      IF (NFEV.GT.MAXFN) GO TO 50
      ITEST = ITEST+1
!      IF (ITEST.LT.2) GO TO 15
      IF (ITEST.LT.4) GO TO 15
C                                     DECREASE STEP LENGTH
```

```
      H = H*HALF
      ITEST = 0
C                                     RELATIVE CONVERGENCE TEST FOR THE
C                                       OBJECTIVE FUNCTION
      IF (DELTF.LT.EPS) GO TO 50
C                                     CONVERGENCE TEST FOR THE STEP LENGTH
      IF (ABS(H)-RELCON) 50,15,15
   35 DO 40 I=1,N
   40 X(I) = X1(I)
      DELTF = (F-F1)/ABS(F1)
      F = F1
C                                     INCREASE STEP LENGTH
      H = H*TWO
      DO 45 I=1,N
   45 X1(I) = X(I)+H*R(IRNDM,I)
      CALL FUN (X1,F1,N,M,MIN,MAX)
      NFEV = NFEV+1
      IF (F1.LT.F) GO TO 35
C                                     CHECK TOLERANCE MAXFN
      IF (NFEV.GT.MAXFN) GO TO 50
C                                     DECREASE STEP LENGTH
      H = ABS(H*HALF)
      GO TO 15
   50 RETURN
      END
```

Programozási feladat:

$\star$ Írjuk át valamelyik ismert programozási nyelvre (C vagy PASCAL), és teszteljük!

Tanulságos jelenség, hogy a globális optimalizálási szakirodalom 90%-a nem matematikai folyóiratban jelenik meg, és ezek jórésze is korábban már tanulmányozott módszerek újrafelfedezését tartalmazza. A programfejlesztés, hibajavítás időigényes, ezért feltétlen ajánlott az előre megírt, tesztelt könyvtári szubrutinok használata. A legtöbb esetben ezekkel tudjuk a leghatékonyabban megoldani feladatainkat.

Az optimalizálási módszereket is tartalmazó szubrutin-könyvtárak közül először az IBM nagyszámítógépeken korábban használatos, és az ESZR gépeken ma is elérhető SSP-t kell említeni (Scientific Subroutine Package). Ennek algoritmusai esetenként már elavultak. Jó nevű, és PC-n is elérhető az IMSL szubrutin-könyvtár, és szabvánnyá vált ennek a forráskódok dokumentálására szolgáló rendszere. Optimalizálási feladatokra speciális rutinokat lehet találni a Harwell Subroutine Library-ban és a NAG könyvtárban is. Ezek a forráskódokat tartalmazó könyvtárak szinte kivétel nélkül FORTRAN-ban íródtak, újabban jelentek csak meg kisebb jelentőségű C, illetve PASCAL nyelvű könyvtárak. Ha feltétlen ezen nyelveken kell egy rutin, akkor is inkább használjunk keresztfordítót az előző könyvtárakkal.

Az egyik leggyakrabban újrafelfedezett algoritmus az evolúciós módszer. Ennek lényege, hogy az abszolút bizalmatlan Monte Carlo típusú módszerekkel szemben megkísérli az aktuá-

lisan végrehajtott függvény-kiértékelések eredményét felhasználni az eljárás további előrehaladásában. Az algoritmus alaptípusa a következő:

1. inicializálás: legyen I a kezdeti intervallum, a lehetséges megoldások halmaza (ez általában több-dimenziós), $k = 0$, $I^0 = I$

2. generáljunk N darab pontot egyenletes eloszlással az I^k intervallumban

3. válasszuk ki ezek közül a legkisebb célfüggvény-értékkel rendelkező γ százalékot

4. határozzuk meg azt a legszűkebb I' intervallumot, amely tartalmazza ezeket a kiválasztott pontokat.

5. nagyítsuk meg I'-t a középpontjából a δ -szeresére ($1 < \delta < 2$), legyen ez az intervallum I^{k+1}, és $k = k + 1$, folytassuk a 2. lépésnél.

Az algoritmus megbízhatóságát nyilván növeli, ha N és γ értéke nagy, ekkor viszont egyre inkább hasonlít a Monte Carlo módszerre. Az eljárás-paraméterek ügyes, az adott feladathoz jól illeszkedő megválasztásával akár egy nagyságrendet is javíthatunk a véletlen keresés hatékonyságán. Az algoritmusban nem szerepel megállási feltétel, de bármely ésszerű feltétel megfelel.

Egy másik érdekes direkt-kereső módszer a szimplex-algoritmus nemlineáris optimalizálási feladatokra. Lényegében csak a neve hasonlít a lineáris optimalizálási feladatokra való szimplex-módszerre. A szimplex voltaképp egy (n+1)-dimenziós egységalakzat az n-dimenziós térben (szakasz az 1-dimenziósban, egyenlő oldalú háromszög a két-dimenziósban, tetraéder a 3-dimenziósban, stb.). Az algoritmus röviden:

1. Inicializálás: vegyünk fel véletlenszerűen egy szimplexet a lehetséges megoldások halmazán, és számítsuk ki a célfüggvény értékét a csúcspontokban.

2. billentsük át a szimplexet úgy, hogy a legrosszabb függvényértékű pontját tükrözzük a többi pontja által meghatározott hipersíkra.

3. ha az új pont se jobb, akkor csökkentsük a szimplex méretét

4. ha a szimplex mérete kisebb egy előre adott $\epsilon > 0$ küszöbértéknél, akkor STOP, különben folytassuk a 2. lépésnél.

Az algoritmus kiegészíthető annak ellenőrzésével, hogy a szimplex benne van-e még a lehetséges megoldások halmazában, de ez igazából a feladattól függ. Itt maga a szimplex jelent információt az algoritmus korábban megszerzett tapasztalatából. Ennek ellenére érdemes az egész eljárást többször végrehajtani, hogy ne csak egy helyi minimumot kapjunk.

Szokás az említett két módszert kombinálni: egyszerre egy szimplex-populációval dolgozni.

Programozási feladat:

$\star$ Írjunk egyszerű szubrutint az evolúciós optimalizálási algoritmusra, és teszteljük!

3.1. Példák

3.2. Ellenőrző kérdések és gyakorló feladatok

4. fejezet

Konjugált gradiens módszer

A *konjugált gradiens módszer* az *optimalizálás* elvein alapul, és szimmetrikus pozitív definit mátrixú lineáris egyenletrendszerek megoldására alkalmas. Pontos aritmetikával ugyan véges sok lépésben megtalálná a megoldást, de a kerekítési hibák miatt mégis iterációs eljárásnak kell tekinteni. Számos variánsa ismert.

Legyen A egy szimmetrikus, pozitív definit mátrix, akkor a

$$q(x) = \frac{1}{2}x^T Ax - x^T b$$

kvadratikus függvénynek egyetlen x^* minimumpontja van, és erre $Ax^* = b$ teljesül. Más szóval az $Ax = b$ lineáris egyenletrendszer megoldása ekvivalens a $q(x)$ kvadratikus függvény minimumpontjának meghatározásával.

A többdimenziós optimalizálási eljárások rendszerint az

$$x_{k+1} = x_k + \alpha s_k$$

alakban keresik az új közelítő megoldást, ahol s_k egy *keresési irány*, és α a *lépésköz*. A kvadratikus függvények optimalizálása során a következő észrevételeket tehetjük:

i, A negatív gradiens (amelyik irányában a célfüggvény csökken) a *reziduális vektor*: $-\nabla q(x) = b - Ax = r$.

ii, Adott keresési irány mentén nem kell adaptív módon meghatározni a lépésközt (mint általános nemlineáris minimalizálás esetén kellene), mert az optimális α közvetlenül megadható. A keresési irány mentén ott lesz a célfüggvény minimális, ahol az új reziduális vektor merőleges s_k-ra:

$$0 = \tfrac{\mathrm{d}}{\mathrm{d}\alpha}q(x_{k+1}) = \nabla q(x_{k+1})^T \tfrac{\mathrm{d}}{\mathrm{d}\alpha}x_{k+1} = (Ax_{k+1} - b)^T \left(\tfrac{\mathrm{d}}{\mathrm{d}\alpha}(x_k + \alpha s_k)\right) = -r_{k+1}^T s_k.$$

Az új reziduális vektort ki lehet fejezni a régivel és a keresési iránnyal:

$$r_{k+1} = b - Ax_{k+1} = b - A(x_k + \alpha s_k) = (b - Ax_k) - \alpha A s_k = r_k - \alpha A s_k.$$

Behelyettesítve r_{k+1}-et, és megoldva az egyenletet α-ra azt kapjuk, hogy

$$\alpha = \frac{r_k^T s_k}{s_k^T A s_k}.$$

Ezzel megkaptuk a szimmetrikus, pozitív definit mátrixú lineáris egyenletrendszerek megoldására szolgáló konjugált gradiens módszert. Egy adott x_0 indulópontra legyen $s_0 = r_0 = b - Ax_0$, és iteráljuk $k = 1, 2, \ldots$ értékekre az alábbi lépéseket, amíg a megállási feltételek nem teljesülnek:

1. $\alpha_k = \frac{r_k^T r_k}{s_k^T A s_k}$ (a lépéshossz meghatározása)
2. $x_{k+1} = x_k + \alpha_k s_k$ (iterált közelítő megoldás)
3. $r_{k+1} = r_k - \alpha_k A s_k$ (az új reziduális vektor)
4. $\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$ (segédváltozó)
5. $s_{k+1} = r_{k+1} + \beta_{k+1} s_k$ (az új keresési irány)

Vegyük észre, hogy az α értékét most kicsit más formában határoztuk meg ($r_k^T s_k$ helyett $r_k^T r_k$ áll). Érvényes viszont, hogy

$$r_k^T s_k = r_k^T (r_k + \beta_k s_{k-1}) = r_k^T r_k + \beta_k r_k^T s_{k-1} = r_k^T r_k,$$

mivel az r_k reziduális vektor merőleges az s_{k-1} keresési irányra.

A korábbi *gradiensmódszer*ek egyszerűen a negatív gradienst követték minden iterációs lépésben, de felismerték, hogy ez a meredek falú enyhén lejtő völgyszerű függvények esetén szükségtelenül sok iterációs lépést követelt a völgy két oldalán való oda-vissza mozgással. A kisebb meredekséggel rendelkező irányban viszont lényegesen gyorsabban lehetett volna haladni. A konjugált gradiens módszer ezzel szemben a lépésenkénti merőleges irányváltoztatással kiküszöböli ezt a hátrányt (innen a neve).

A megállási feltétel szokás szerint az, hogy a felhasználó előírja, hogy az utolsó néhány iterált közelítés eltérése és a lineáris egyenletrendszer két oldala különbsége normája ezekben a pontokban adott kis pozitív értékek alatt maradjanak.

A konjugált gradiens módszer nemlineáris optimalizálásra is alkalmas, ha minden iterációs lépésben az eredeti célfüggvény kvadratikus modelljére alkalmazzuk (az adott pontbeli függvényértékre, a gradiensre és a Hesse mátrixra vagy ezek közelítésére támaszkodva).

A konjugált gradiens módszer egy egyszerű megvalósítása a Matlabban:

```
function x = kg(A,b,x);
s = b-A*x;
r = s;
for k=1:20
     a =(r'*r)/(s'*A*s);
     x = x+a*s;
     rr = r-a*A*s;
     s = rr+s*((rr'*rr)/(r'*r));
     r = rr
end
```

Az áttekinthetőség kedvéért a megállási feltételeket elhagytuk a programból, ezek akkor állították meg az iterációt, ha a keresési irány, vagy a reziduális vektor normája, illetve ha a megoldás utolsó két iteráltjának eltérése normája kisebb volt, mint 0.00001. A kiindulási adatok:

$$A = \begin{pmatrix} 4 & 1 \\ 1 & 2 \end{pmatrix}, \qquad b = \begin{pmatrix} 5 \\ 3 \end{pmatrix}, \qquad x = \begin{pmatrix} 3 \\ 3 \end{pmatrix}.$$

Látható, hogy a megoldás $x^* = [1, 1]^T$. A kapott eredmény két iteráció után:

```
r =
     1.0e-014 *
    -0.1554
    -0.0888
ans =
     1.0000
     1.0000
```

Tehát a lineáris egyenletrendszer bal- és jobb oldalának eltérése már a számábrázolás határán volt, és az eredmény is nagyon közeli az elméleti megoldáshoz. Ez teljes összhangban van a módszer (pontos aritmetika használata esetén érvényes) véges számú lépésben való konvergenciájával, de látszik a kerekítési hibák hatása is.

4.1. Ellenőrző kérdések és gyakorló feladatok

1. Milyen feltételek teljesülése esetén áll meg a lineáris egyenletrendszerek megoldására szolgáló iterációs algoritmus?

2. Hogyan lehet nagyobb pontosságot kérni a Matlab iterációs lineáris egyenletrendszer megoldó eljárásaitól?

3. A következő lineáris egyenletrendszerekre vizsgálja meg, hogy a $(0, 0, \ldots, 0)^T$ indulóvektorral számolva a Jacobi, illetve a Gauss-Seidel iteráció konvergál-e a megoldáshoz?

   i, $4x - y = 15, \quad x + 5y = 9$

   ii, $8x - 3y = 10, \quad -x + 4y = 6$

   iii, $-x + 3y = 1, \quad 6x - 2y = 2$

   iv, $2x + 3y = 1, \quad 7x - 2y = 1$

   v, $5x - y + z = 10, \quad 2x + 8y - z = 11, \quad -x + y + 4z = 3$

   vi, $2x + 8y - z = 11, \quad 5x - y + z = 10, \quad -x + y + 4z = 3$

   vii, $x - 5y - z = -8, \quad 4x + y - z = 13, \quad 2x - y - 6z = -2$

   viii, $4x + y - z = 13, \quad x - 5y - z = -8, \quad 2x - y - 6z = -2$

4. Oldja meg a következő lineáris egyenletrendszereket a konjugált gradiens módszerrel:

   i, $4x - y = 3, \quad -x + 5y = 4$

   ii, $8x - 3y = 10, \quad -3x + 4y = 2$

iii, $-10x + 3y = 7, \quad 3x - 12y = 6$

iv, $4x + 3y = 7, \quad 3x - 2y = 1$

v, $5x - y + z = 5, \quad -x + 8y - z = 6, \quad x - y + 4z = 4$

5. Igazolja, hogy a $||x||_1$ vektornorma teljesíti a vektornormák tulajdonságait (amik a mátrixnormák első három tulajdonságának felelnek meg).

6. Bizonyítsuk be $\rho(B) < 1$ esetén a B mátrixhoz tartozó iteráció globális konvergenciáját a sajátértékek, sajátvektorok tulajdonságai alapján!

7. Indokoljuk, hogy a Gauss-Seidel iteráció Matlab programjában miért kellett a ciklusmagot három esetre szétbontani!

5. fejezet

Lipschitz függvények optimalizálása

Ez a fejezet azzal az esettel foglalkozik, amikor a globális optimalizálási feladat megoldásához a célfüggvény Lipschitz konstansának a használatára is támaszkodhatunk. Avval fogunk itt foglalkozni, hogy ez az információ hogyan hasznosítható úgy, hogy megbízható és hatékony eljárást építhessünk rá.

Mint ismeretes, akkor mondjuk, hogy egy $f : \mathbb{R}^n \to \mathbb{R}$ függvény *Lipschitz folytonos* az L Lipschitz konstanssal egy $D \subseteq \mathbb{R}^n$ tartományon, ha minden $x, y \in D$ pontpárra érvényes az

$$|f(x) - f(y)| \leq L||x - y||$$

összefüggés. A Lipschitz folytonosság tehát lényegében azt fejezi ki, hogy a vizsgált tartományon az adott függvény megváltozása arányos az érintett két pont távolságával, az így adódó meredekségnek van egy véges korlátja.

A Lipschitz folytonos függvények gyakorinak mondhatók, amit az is alátámaszt, hogy minden folytonosan differenciálható függvény nyilvánvalóan Lipschitz folytonos minden korlátos zárt (kompakt) tartományon (hiszen a derivált értéke a folytonosság miatt véges minden pontban, és így korlátos is).

Egyes esetekben a Lipschitz konstans, illetve annak egy felső korlátja viszonylag könnyen meghatározható. Ilyen eset áll fenn pl. a polinomokra: a derivált könnyen képezhető, aminek abszolút korlátját is egyszerű megadni adott tartományra. Tekintsük például az $f(x) = x^2 - x$ polinomot. Ennek deriváltja az $f'(x) = 2x - 1$ függvény. Ennek abszolútértékének nyilván korlátja a $[0, 1]$ intervallumon a 3, hiszen $f'(x)$ tagjainak abszolút értékének maximuma az adott intervallumon 2 és 1.

Abban az esetben, ha a célfüggvény képletét nem ismerjük (mert mondjuk csak egy szubrutin formában áll rendelkezésre), akkor a fenti eljárás nem működik. Számos gyakorlati helyzetben a felhasználónak elég világos képe van az optimalizálandó függvényről, nem elképzelhetetlen, hogy a Lipschitz konstans értékét is meg tudja becsülni. A becslés minőségén sok múlik: ha az egy érvényes felső becslése L-nek, akkor az erre alapuló módszer megbízható eredményt tud adni. Ha viszont a megadott konstans téves, tehát nála nagyobb meredekség is előfordul a keresési tartományon, akkor előfordulhat, hogy hamis eredményt kapunk: a valódi globális minimum érték, és a globális minimumpont eltér a megkapottól.

5.1. Példák

5.2. Ellenőrző kérdések és gyakorló feladatok

6. fejezet

DC programozás

6.1. Példák

6.2. Ellenőrző kérdések és gyakorló feladatok

7. fejezet

A korlátozás és szétválasztás módszere

Olyan optimalizálási feladatok megoldására, amelyeket közvetlenül nem lehet valamely bevett eljárással megoldani, hasznos az eredeti feladat egyszerűbb részfeladatokra való felbontása. Ide tartozik az egészértékű lineáris optimalizálási feladatok köre, és a nemlineáris programozás is.

Az alapötlet az eredeti feladat szisztematikus felosztása olyan kisebb, valamely szempontból kezelhetőbb részfeladatokra, amelyek bizonyos értelemben a teljes leszámolás egy hatékony megvalósítását adják. A megoldott részfeladatok eredményeit természetesen megfelelően összegezni kell. A módszer erejét az adja, hogy minden lépése automatizálható.

Tekintsük azt a feladatot, amelyben

$$\min f(x)$$

az optimalizálási cél, és a lehetséges megoldásokat azonos dimenziójú, egész koordinátájú x vektorok egy véges és nem üres L halmaza adja meg.

Ennek a feladatnak nyilvánvalóan van optimális megoldása, hiszen a véges sok lehetséges vektor között nyilván kijelölhető az, amelyiknél kisebb célfüggvényértéket a többi nem ad. Sok esetben a lehetséges megoldások száma nagyon nagy. Így például az $n \times n$-es hozzárendelési feladat esetén $n!$ darab lehetséges megoldást kellene ellenőrizni.

A *korlátozás és szétválasztás módszere* (angolul branch-and-bound, B&B) két függvényre támaszkodik:

- a ϕ *szétválasztási függvény* az L lehetséges megoldási halmaz egy tetszőleges L' (amire $|L'| > 1$) részhalmazának megadja egy valódi osztályozását.
- a g *korlátozó függvény* pedig az L egy tetszőleges $L' \neq \emptyset$ részhalmazához hozzárendeli az $f(x),\ x \in L'$ célfüggvényértékek egy alsó korlátját. Amennyiben L' egy x lehetséges vektorból áll, akkor $g(x) = f(x)$.

Erre a két függvényre alapozva már fel lehet építeni a korlátozás és szétválasztás módszer egy változatát. A korlátozás és szétválasztás módszere egy *leszámlálási fá*t épít fel a következők szerint:

0. lépés Az előkészítés során határozzuk meg a $g(L)$ értéket, és legyen L a leszámlálási fa gyökere. Legyen $k = 1$. Címkézzük meg a gyökeret a $g(L)$ értékkel.

1. lépés Az aktuális fa levelein határozzuk meg a címkék minimumát, és válasz-szunk ki egy minimális címkéjű L' levelet.

2. lépés Amennyiben L' már csak egy vektorból áll ($L' = \{\overline{x}\}$), akkor vége az eljárásnak, $\overline{x}$ optimális megoldás.

3. lépés Bővítsük az aktuális fát $\phi(L')$ elemeivel, legyenek ezek L' leszármazottjai az épített keresési fában. Az új levelekre határozzuk meg az alsó korlátokat a g függvény segítségével, és rendeljük őket címkeként a megfelelő levelekhez. Növeljük a k iterációszámot eggyel, és térjünk rá a következő iterációs lépésre (1. lépés).

Az eljárás *végessége* abból adódik, hogy a ϕ definíciója alapján minden L' részfeladatnak legfeljebb $|L'|$ leszármazottja van, és hogy az algoritmus futásának minden fázisában az eredeti L lehetséges megoldási halmaz egy osztályozását jelentik az aktuális levelek. A szétválasztási függvény tulajdonságán múlik, hogy minden újabb szétválasztás valódi osztályozást ad. Ebből az adódik, hogy a keresési fa maximális mélysége $|L|$. A fa végességéből már következik az eljárás végessége is.

Az algoritmus *helyessége* azon múlik, hogy minden iterációs fázisban a lehetséges megoldásoknak az aktuális levelek által meghatározott osztályozása részhalmazaira ismert alsó korlátok legkisebbike alsó korlátja lesz az optimális célfüggvényértéknek. A megálláskor tehát $f(\overline{x}) \leq f(x)$ adódik az $\overline{x}$ vektorra. Ez pedig pontosan azt jelenti, hogy $\overline{x}$ optimális megoldás.

Gyakran hasznos a lehetséges megoldások L halmazát befoglalni egy könnyebben kezelhető halmazba, és a felosztást azon végigkövetni. Érdemes az alapmódszer indítása során egy lehetséges megoldásra vonatkozó (és ezért pontos) felső korlátot adni az optimum értékére. Ennek segítségével a számontartott részfeladatok számát csökkenteni lehet.

PÉLDA. Tekintsük a következő egyszerű 0-1 értékű lineáris programozási feladatot:

$$\min -4x_1 - x_2 - x_3 - x_4$$

feltéve hogy a

$$5x_1 + 3x_2 + 2x_3 + x_4 \leq 5$$

teljesül, és $x_i \in \{0,\ 1\}, i = 1, \ldots, 4$.

Ebben az esetben a lehetséges megoldások halmaza:

$$L = \{(0,0,0,0), (0,0,0,1), (0,0,1,0), (0,0,1,1), (0,1,0,0), (0,1,0,1), (0,1,1,0), (1,0,0,0)\}.$$

Az L-en a célfüggvény alsó korlátjának vegyük a célfüggvény együtthatók összegét, amelyekre van egyes valamely vektorban (ennél kisebb érték nem fordulhat elő): $g(L) = -7$.

Tegyük fel, hogy a szétválasztási függvény L-et olyan két halmazra bontja, hogy L_1-be kerüljenek azok a vektorok, amelyekre $x_1 = 0$, L_2-be pedig azok, amelyekre $x_1 = 1$. Ekkor

$$L_1 = \{(0,0,0,0), (0,0,0,1), (0,0,1,0), (0,0,1,1), (0,1,0,0), (0,1,0,1), (0,1,1,0)\},$$

és $g(L_1) = -3$, illetve

$$L_2 = \{(1,0,0,0)\},$$

és $g(L_2) = -4$.

Mivel a következő lépésben az L_2 levelet kellene tovább osztani, és az már csak egy vektort tartalmaz, ezért $\overline{x} = (1,0,0,0)$ az optimális megoldás.

Vegyük észre, hogy a fenti gyors megoldást csak az tette lehetővé, hogy a lehetséges megoldások halmazából egy lépésben sikerült elkülöníteni egy egyelemű részhalmazt, amelyre a célfüggvény értéke nem volt nagyobb, mint a többi, a lehetséges megoldások közé tartozó vektor célfüggvény értéke.

Gyakorlati feladatokban persze lényegesen nagyobb számú iteráció kell a megoldáshoz – másrészt ezzel együtt is hatásos és hatékony eszköz lehet a korlátozás és szétválasztás módszere.

PÉLDA. Tekintsük a $\min f(x) = x^2$ feladatot az $X = [-2, 10]$ intervallumon. A szélsőérték nyilván a 0 pontban van. Az $f(x)$ függvény befoglaló függvényértéke a kiindulási intervallumon $[-2, 10] * [-2, 10] = [-20, 100]$.

A kiindulási intervallumot osszuk fel két egyenlő részre. A kapott intervallumokra adódó korlátok:

$$f([-2, 4]) = [-8, 16], \quad f([4, 10]) = [16, 100].$$

Ebből az adódik, hogy a teljes feladatra vonatkozó alsó korlátunk -20-ról -8-ra javul. Vegyük észre, hogy a második részintervallumon a célfüggvényünk monoton, ezért a befoglaló függvényünk pontos.

A következő iterációs lépésben a legígéretesebb részintervallum a $[-2, 4]$. Osszuk fel most ezt. Az ezután meglévő részintervallumokra a korlátok:

$$f([-2, 1]) = [-2, 4], \quad f([1, 4]) = [1, 16], \quad f([4, 10]) = [16, 100].$$

Mivel egy részintervallumon ($[-2, 1]$) kapott felső korlát kisebb, mint egy másik részintervallumra ($[4, 10]$) érvényes alsó korlát, ezért az utóbbi törölhető a keresési tartományból, hiszen nem tartalmazhat optimális megoldást.

A következő néhány iteráció utáni még figyelembe veendő részintervallumok a hozzájuk tartozó korlátokkal:

$$f([-2,\ -0,5]) = [0,25,\ 4], \quad f([-0,5,\ 1]) = [-0,5,\ 1], \quad f([1,\ 4]) = [1,\ 16],$$

$$f([-2,\ -0,5]) = [0,25,\ 4], \quad f([-0,5,\ 0,25]) = [-0,125,\ 0,25],$$
$$f([0,25,\ 1]) = [0,0625,\ 1],$$

$$f([-0,5,\ -0,125]) = [0,015625,\ 0,25], \quad f([-0,125,\ 0,25]) = [-0,03125,\ 0,0625],$$
$$f([0,25,\ 1]) = [0,0625,\ 1].$$

Az optimális célfüggvényértékre vonatkozó bizonytalanság 5 iterációs lépés alatt 120-ról 0,1 alá csökkent. Az optimum helye bizonytalansága 12-ról 1,5-re alakult.

PÉLDA. Ebben az esetben a feladat egy olyan kellemetlen gyár telepítési helyszín meghatározása volt, amelyre a magyarországi nagyobb városok lakosai számával arányos elutasítás figyelembevételével a lehető legkisebb gondot okozza.

A célfüggvény ennek megfelelően

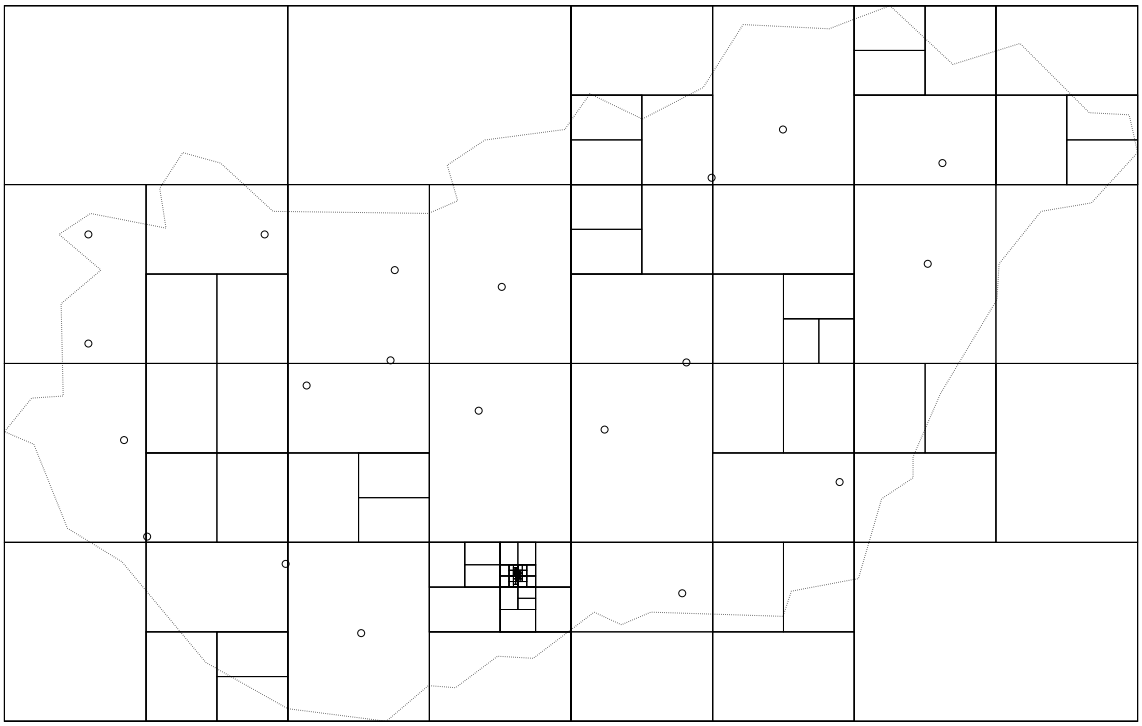
$$f(x) = \sum_i \frac{l_i}{(x - x_i)^2 + (y - y_i)^2},$$

ahol x és y a telepítés helyszínének koordinátái, az i-edik város lakosai száma l_i, koordinátái pedig x_i és y_i. Nyilván $f(x)$ minimalizálása a cél.

Az optimalizálási feladat korlátozását jelentette, hogy

- a gyárnak az országhatárokon belül legalább 50 kilométerre kell lennie,
- a városok 5 kilométeres körzete is kizárt a telepítésből.

A kapott eredményt a következő ábra mutatja – konkrétan a megvizsgált részintervallumok jelölésével:



Érdekes eredmény, hogy ha a határtól való eltérést nem követeltük meg, akkor az optimális pozíció minden esetben a határra adódott, még akkor is, ha figyelembe vettük a határon túli nagyobb városok taszító hatását is.

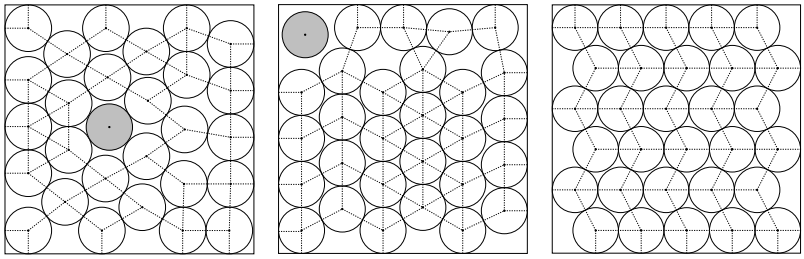
7.1. Körpakolási feladatok

Két ekvivalens megfogalmazás:

- Helyezzünk el adott n darab egybevágó kört átlapolás nélkül, maximális sugárral az egységnégyzetben.
- Helyezzünk el adott n számú pontot az egységnégyzetben úgy, hogy a köztük lévő minimális távolság maximális legyen.

$$\max \min_{1 \le i \ne j \le n} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2},$$

$$\text{ahol } 0 \le x_i, y_i \le 1, \qquad i = 1, 2, \ldots, n.$$



A satírozott körök kis mértékben mozgathatók az optimalitás megtartása mellett (a globális minimumpontok halmaza *pozitív mértékű*). Két kör érintkezését az összekötő vonalak jelzik.

Hardware: PC, Pentium IV 1800 MHz processor, 1 GB RAM. Software: Linux, GNU C/C++, C–XSC Toolbox, PROFIL/BIAS. A sugár értékére kapott korlátok:

$$F_{28}^* = [\underline{0.2305354936426}73, \underline{0.2305354936426}743],\ w \approx 7 \cdot 10^{-15},$$

$$F_{29}^* = [\underline{0.2268829007442}089, \underline{0.2268829007442}240],\ w \approx 2 \cdot 10^{-14},$$

$$F_{30}^* = [\underline{0.2245029645310}881, \underline{0.2245029645310}903],\ w \approx 2 \cdot 10^{-15}.$$

A teljes futási idők: $\approx$ 53, 50, illetve 21 óra. A feladatok megoldásához kb. egy millió részintervallum kellett. A verifikált eljárás az optimális pakolás helyére vonatkozó bizonytalanságot több mint 711, 764, illetve 872 nagyságrenddel csökkentette.

7.2. Ellenőrző kérdések és gyakorló feladatok

8. fejezet

Intervallumos módszerek

8.1. Intervallum-aritmetika és a befoglaló függvények

Legyen I a kompakt valós intervallumok tere. Az intervallum-aritmetika műveletei ezen a halmazon vannak értelmezve. A műveleteket úgy kell definiálni, hogy az $A * B$ eredménye egy olyan C intervallum legyen, amely pontosan azon c valós számok halmaza, amelyekhez léteznek olyan $a \in A$ és $b \in B$ valósok, hogy $c = a * b$. Itt $*$ a négy alapművelet valamelyikét jelöli. Az ilyen aritmetika segítségével követni lehet a kerekítési hibákat, és az adatainkat terhelő bizonytalanság tükröződhet az eredményekben.

Az előző definíció mellett az intervallum-aritmetikát lehet kizárólag a valós aritmetikára támaszkodva is definiálni. Az $[a,\ b]$ és $[c,\ d]$ intervallumokra legyen

$$[a,\ b] + [c,\ d] = [a + c,\ b + d],$$

$$[a,\ b] - [c,\ d] = [a - d,\ b - c],$$

$$[a,\ b][c,\ d] = [\min(ac, ad, bc, bd),\ \max(ac, ad, bc, bd)],$$

$$[a,\ b]/[c,\ d] = [a,\ b][1/d,\ 1/c].$$

Az osztást csak akkor értelmezzük, ha $0 \notin [c,\ d]$. Érdemes megjegyezni, hogy ez utóbbi feltétel jól megfogalmazott gyakorlati feladatokban tapasztalataink szerint szinte kivétel nélkül teljesül. A valós műveleteknek ezt a kiterjesztését intervallumokra természetes vagy naiv intervallum-kiterjesztésnek nevezik. Az utóbbi években vizsgálják az olyan intervallum-aritmetikákat is, amelyek nem csak kompakt intervallumokon definiáltak. Ezeken a nullát tartalmazó intervallummal való osztás is értelmezhető.

Bár az alapműveletek pontosak a fenti értelemben, mégis, a velük kiszámított bonyolultabb függvények durva becslései is lehetnek a megfelelő értékkészletnek. A gyakran emlegetett példa a következő: az $x - x^2$ értékkészlete a $[0,\ 2]$ intervallumon $[-2,\ 0,25]$. Ezzel szemben az intervallum-kiterjesztéssel adódó intervallum $[-4,\ 2]$.

Az intervallum-aritmetika műveleteinek tulajdonságaival foglalkozik az intervallum-algebra. Számos, a valós műveletekre érvényes tulajdonság változatlanul teljesül az intervallum-műveletekre is (pl. a kommutativitás, asszociativitás az összeadásra és a szorzásra), de általában nincs inverz, és érvényes a szubdisztribúciós tulajdonság: $A(B + C) \subseteq AB + AC$.

Az alapműveletekhez hasonlóan könnyen lehet definiálni az elemi függvények intervallum-kiterjesztését is, tehát a számítógépen kiszámítható függvényeket szinte kivétel nélkül meg lehet valósítani természetes intervallum-kiterjesztésben is.

Az intervallum-aritmetika alkalmazása szempontjából alapvető fogalom a befoglaló függvény. Az $F(X) : I^n \to I$ az $f(x)$ n-változós valós függvény befoglaló függvénye, ha $f(x) \in F(X)$

érvényes minden $x \in X$ pontra és $X \in I^n$ intervallumra. Az intervallum-matematika fontos eredménye, hogy az $f(x)$ valós függvényből természetes (vagy naiv) intervallum-kiterjesztéssel adódó $F(X)$ függvény befoglaló függvény.

A befoglaló függvényektől természetes azt elvárni, hogy bővebb argumentum-intervallumra ne adjanak szűkebb eredmény-intervallumot. Ezt a feltételt fogalmazza meg az izotonitás: egy $F(X)$ befoglaló függvény akkor izoton, ha $X \subseteq Y$-ból következik $F(X) \subseteq F(Y)$. Az izotonitás szinte minden intervallum-aritmetika implementációra érvényes.

A befoglaló függvények minőségének fontos mutatója a rend: azt mondjuk, hogy az $F(X)$ befoglaló függvény rendje $\alpha > 0$, ha létezik olyan c valós konstans, hogy $w(F(X)) - w(f(X)) \leq cw(X)^\alpha$ teljesül minden $X \in I^n$-re, ahol $w(X)$ az X intervallum szélessége. A természetes intervallum-kiterjesztéssel adódó befoglaló függvények elsőrendűek, de kidolgozott a magasabbrendű befoglaló függvények elmélete is. Az egynél szélesebb intervallumokra a természetes intervallum-kiterjesztést, a kisebbekre pedig a magasabbrendű befoglaló függvényeket szokták ajánlani.

A számítógépes megvalósítás során minden intervallum-művelet végrehajtása után a kapott intervallumot módosítani szokás. Az intervallum alsó határát lefelé, felső határát felfelé kell kerekíteni a legközelebbi ábrázolható számra. Ezzel az úgynevezett kifelé kerekítési eljárással el lehet érni, hogy a befoglalási tulajdonság a kerekítési hibák ellenére is fennmaradjon. Ezen a módon számítógéppel automatizálható a garantált megbízhatóságú befoglaló függvények előállítása.

Az intervallum-aritmetikához használatos speciális kerekítéseket az IEEE szabvány biztosítja, ezért napjaink szinte minden processzora támogatja. A hetvenes évek közepétől elérhetők olyan programozási nyelvek, amelyek az INTERVAL adattípus használatát támogatják. Ilyen nyelveken még az intervallum-aritmetikát megvalósító szubrutinokat sem kell megírni: a megfelelő befoglaló függvény implementálásához elegendő a függvény kiszámításához használt változók típusát megváltoztatni.

A befoglaló függvényekre támaszkodó numerikus algoritmusok érzékenyek a befoglaló függvény minőségére, pontosságára. A vázolt természetes intervallum-kiterjesztés mellett számos más eljárás is ismert a befoglaló függvények előállítására, például a magasabbrendű deriváltakat is használó ún. középponti alakok, az automatikus deriválásra és monotonitásvizsgálatra épülő stratégiák a befoglaló függvény javítására, illetve az optimális pontosságú befoglaló függvényt generáló eljárás. Ezek a módosítások természetesen növelik az egy befoglaló függvény kiértékeléséhez szükséges számítások mennyiségét.

8.2. A korábbi SpecKoll.-ból

Jegyzet vagy magyar nyelvű irodalom sajnos még nincs. Angol és német (esetleg orosz) nyelvű bevezető könyveket tudok ajánlani:

1. G. Alefeld, J. Herzberger: Einführung in die Intervallrechnung, Bibliographises Institut AG, Mannheim, 1974.

2. G. Alefeld, J. Herzberger: Introduction to Interval Computations, Academic Press, New York, 1983.

3. H. Ratschek, J. Rokne: Computer Methods for the Range of Functions, Ellis Horwood Ltd., Chichester, 1984.

4. S.A. Kalmikov, Yu.I. Sokin, Z.H. Yuldashev: Az intervallum-analízis módszerei (oroszul), Nauka, 1986.

5. H. Ratschek, J. Rokne: New Computer Methods for Global Optimization, Ellis Horwood Ltd., Chichester, 1988.

A jelenlegi numerikus eljárások szinte kivétel nélkül helyi információn alapulnak: pl. a vizsgált függvényt adott pontban kiértékelő szubrutin megadását kívánják meg. Bár a szóbajövő függvények pontos képletét, vagy legalább annak kiszámítási módját ismerni kell, mégis a legtöbb numerikus módszer csak adott pontbeli függvényértékre épül. Számos feladat és módszer esetén igazolható, hogy csak helyi információra támaszkodva az illető feladat véges sok lépésben nem oldható meg, sőt, ilyen módszer se létezhet (vö. Cs. T., Acta Cybernetica, 1988). Az a paradox helyzet áll fenn, hogy valójában az illető függvényről lényegesen többet tudunk, mint amennyit a legtöbb numerikus módszer a megoldáshoz felhasznál. Ezek tehát a fekete doboz elvén működnek.

A doktorandusz hallgatók számára külön követelmény egy 20-30 oldalas esszé írása a tantárgy bővebb témaköréből. A választható témák a következők:

- Affin aritmetika (Ronald van Iwaarden doktori dolgozata alapján)
- Back-boxing, illetve ϵ-infláció (Ronald van Iwaarden doktori dolgozata alapján)
- Kiterjesztett intervallum aritmetikák, Kaucher-féle intervallum aritmetika (elsősorban a Kearfott könyv alapján)
- Intervallumos Newton-iteráció, Prekondícionálás (Kearfott könyve alapján)
- lejtő aritmetika (slope, Dietmar Ratz habilitációs disszertációja alapján)
- változó pontosságú aritmetikák
- Taylor-modellek (Martin Berz munkái alapján)

Feladatok:

- Írjunk egy rövid programot, amely három valós számot összead, majd igazoljuk, hogy van három szám, amelyre a program által adott eredmény a ténylegestől legalább 2002-vel eltér!
- Módosítsuk a programot úgy, hogy az utóbbi három számra pontos legyen!
- Adjunk meg néhány olyan összeadó eljárást, amely a fenti problémára megoldást jelenthet!
- Vizsgáljuk meg az egyes algoritmusok műveletigényét!
- Milyen módszer felel meg a pénzügyi számításokhoz, ahol lényegében csak egész számokkal számolnak, de azért $100.\dot{3} + 100.\dot{3} + 100.\dot{3} = 301$?
- Mit lehet ajánlani olyan alkalmazáshoz, ahol minden szóbajövő szám racionális, és azt szeretnénk, ha $(xy)/y = x$ mindig teljesülne?

Postscript file-ként rendelkezésre álló doktori dolgozatok, illetve kéziratok:

1. S.L.P. Ferguson: Sphere Packings (a Kepler feladat megoldásának részletei)

2. R.J. Van Iwaarden: An improved unconstrained global optimization algorithm, Denver, 1996.

3. F. Messine: Methodes d'Optimisation Globale basees sur l'Analyse d'Intervale pour la Resolution de Problemes avec Contraintes. Toulouse, 1997.

4. A. Wiethoff: Verifizierte globale Optimierung auf Parallelrechnern. Karlsruhe, 1997.

Alapötlet: a valós számokra végzett műveleteket ki lehet terjeszteni intervallumokra is, és ha valamely mennyiségről nem egy konkrét valós számmal való egyenlőségét, hanem egy intervallumba való tartozását ismerjük, akkor az intervallumokra végrehajtott műveletek célirányosnak tűnnek.

Halmazelméleti definíció: $A \circ B := \{a \circ b : a \in A, \quad b \in B\}$; $A, B \in I$, ahol I a valós kompakt intervallumok halmaza (azaz olyan (i, j) pároké, amelyekre $i, j \in I\!R$, és $i \leq j$).

Aritmetikai definíció:
$[a, b] + [c, d] = [a + c, b + d]$
$[a, b] - [c, d] = [a - d, b - c]$
$[a, b] * [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$
$[a, b]/[c, d] = [a, b] * [1/d, 1/c]$, ha $0 \notin [c, d]$.

Megjegyzés: az osztás definiálásánál a $0 \notin [c, d]$ feltétel gyakran előforduló megszorításnak tűnik, de a tapasztalatok szerint nem az.

Állítás: az aritmetikai definíció megfelel a halmazelméletinek, és viszont. Tehát az intervallum-aritmetika ebben az értelemben pontos.

Az intervallum-aritmetika algebrai tulajdonságai:

- a $+$ és a $-$, illetve a $*$ és a $/$ nem inverzei egymásnak, ha intervallumokra alkalmazzuk őket. Például $[0, 1] - [0, 1] = [-1, 1]$, és $[1, 2]/[1, 2] = [1/2, 2]$. Valamint $[0, 0] + [0, 1] - [0, 1] = [-1, 1]$ és az eredmény nem $[0, 0]$.

- érvényes az ún. szubdisztribúciós törvény, azaz $A(B + C) \subseteq AB + AC$. Például $[0, 1]([1, 1] - [1, 1]) = [0, 0] \subset [0, 1][1, 1] - [0, 1][1, 1] = [-1, 1]$. Másrészt viszont az $a \in I\!R$ konstansra $a(B + C) = aB + aC$.

- érvényes az az általános szabály is, hogy a 0-szélességű intervallumokra (amelyekre $w(A) = 0$, ahol $w(A) = b - a$, ha $A = [a, b]$) az intervallum-műveletek megegyeznek a valós számokon szokásos műveletekkel.

- az összeadás és a szorzás kommutatív és asszociatív. Az egyetlen egységelem az $[1, 1]$, az egyetlen zéruselem a $[0, 0]$.

- érvényes az intervallum-műveletek befoglalási izotonitása: $A \subseteq B$, $C \subseteq D$ -ből következik, hogy $A \circ C \subseteq B \circ D$. (Persze csak akkor, ha az illető műveletek definiáltak.)

- definiáljuk az n-dimenziós $A \in I^n$ intervallum szélességét a koordinátánkénti intervallumok szélességének maximumaként: $w(A) := \max(w(A_i)\ i = 1, \ldots, n)$, ha

$$A = (A_1, A_2, \ldots, A_n) \in I^n.$$

  Ekkor teljesülnek a következők:

  1. ha $A \subseteq B$, akkor $w(A) \leq w(B)$
  2. $w(C + D) = w(C) + w(D)$ (az egy dimenziós esetben)
  3. $w(aB) = |a|w(B)$

- Definiáljuk az A intervallum $m(A)$ középpontját a következőek szerint: $m(A) = (a + b)/2$, ha $A \in I$, és $m(A) = (m(A_1), m(A_2), \ldots, m(A_n))$, ha $A \in I^n$. Ekkor $m(A \pm B) = m(A) \pm m(B)$, ha $A, B \in I^n$.

A numerikus algoritmusok gyakran támaszkodnak valamely függvény deriváltjára. A deriváltfüggvények előállítására az alkalmazás gyakoriságának megfelelő sorrendben a következő eljárásokat szokás használni:

1. Az illető függvény "kézzel", papiron való deriválása, majd a megfelelő szubrutin megírása.
2. Numerikus deriválás: amikor a felhasználó vagy a számítógépes program maga ad egy numerikus közelítést (a differencia-hányadost) a derivált aktuális pontbeli értékére. Ha lehet választani, akkor az algoritmusba beépített közelítést válasszuk !
3. A vizsgált függvény szimbolikus deriválása valamely számítógépes algebra rendszerben (DERIVE, REDUCE, MATHEMATICA, MAPLE, ...).
4. Az alább részletezendő automatikus differenciálás.

Ha egy $f(x)$ függvény képlettel megadható, illetve rendelkezésre áll az őt kiszámító szubrutin, akkor a következő eljárással egyszerűen lehet a derivált értékét (nem numerikus becsléssel) meghatározni. A függvény minden változója helyett használjunk olyan adat-szerkezetet, amely két valós számból áll. Az első felel meg a korábbi változó-értéknek, a második pedig egy derivált-értéknek. Minden változóra ez a második valós legyen a szubrutin hívásakor egy. Minden, a függvény kiszámításához használt konstans új adat-szerkezetében a második érték legyen nulla. Ezután csak olyan segéd-rutin készletre van szükség, amely minden műveletre elvégzi a megfelelő operációt az első tagon, és a deriválási szabályoknak megfelelő lépést a második tagon. Például, ha $f(x) = x_1 * x_2$, akkor a szokásos programsor $F = X(1) * X(2)$ lenne. Az automatikus differenciálás megfelelő művelete ezzel szemben

$$F(1) = X(1, 1) * X(2, 1),$$

és

$$F(2) = X(1, 1) * X(2, 2) + X(2, 1) * X(1, 2)$$

lesz. A többi, számítógépen szokásos művelethez is könnyű megadni a megfelelő új aritmetikát. Az automatikus differenciálás jól használható olyan optimalizálási feladatok megoldásához, amikor a célfüggvényt kiszámító szubrutin rendelkezésre áll, de a deriváltak analitikus kiszámítása bonyolult vagy lehetetlen lenne.

Vegyük észre, hogy a derivált értékének kiszámítása során sehol se jelenik meg a derivált-függvény képlete.

Ajánlott irodalom: L.B. Rall: Automatic Differentiation — Techniques and Applications, Springer Verlag, Lecture Notes in Computer Science Vol. 120, Berlin 1981.

8.3. Intervallum-felosztási algoritmus

Az intervallum-felosztási (Moore-Skelboe) algoritmus adott nemlineáris függvény valamely intervallumon vett globális minimumának alsó- és felsőbecslését adja meg. A kezdeti X intervallumban egy olyan X'-t keres meg, hogy $F(X')$ tartalmazza a globális minimum értékét, és az $F(X')$ intervallum szélessége kisebb legyen, mint egy előre adott ε pozitív konstans. Az algoritmus a következő:

1. Legyen $Y := X$ és $y := \min F(X)$. Inicializáljuk az $L = ((Y, y))$ listát.
2. Válasszunk egy olyan k koordinátát, amellyel párhuzamosan az $Y = Y_1 \times \cdots \times Y_n$ -nek maximális hosszúságú éle van.
3. Vágjuk ketté Y-t a k irány mentén: így olyan V_1 és V_2 boxokat kapunk, amelyekre $Y = V_1 \cup V_2$.
4. Számítsuk ki $F(V_1)$-et és $F(V_2)$-t, és legyen $v_i = \min F(V_i)\ \ i = 1,\ 2$-re.
5. Töröljük (Y, y)-t az L listából.
   (a) Monotonitási-vizsgálat: töröljük a (V_i, v_i) párt, ha $0 \notin F'_j(V_i)$ valamely j $(1 \leq j \leq n)$-re és $i = 1,\ 2$-re.
   (b) Kivágási-vizsgálat: töröljük a (V_i, v_i) párt, ha $v_i > \delta$ (ahol δ adott eljárás-paraméter) és $i = 1,\ 2$.
6. Tegyük a (V_1, v_1) és (V_2, v_2) párokból a megmaradtakat a listába. Ha a lista üres, akkor STOP.
7. Jelöljük a lista azon párját, amelynek második eleme a legkisebb, (Y, y)-al.
8. Ha $F(Y)$ szélessége kisebb, mint ε, akkor nyomtassuk ki $F(Y)$ és Y értékét, és STOP.
9. Folytassuk az algoritmust a 2. lépésnél.

Az 5a pontbeli monotonitási teszt akkor töröl valamely intervallumot, ha azon az $f(x)$ függvény szigorúan monoton. Ilyen esetben az adott intervallum nem tartalmazhat a belsejében minimumpontot. Ha az algoritmus azzal áll le, hogy üres lett a lista, akkor meg kell vizsgálni, hogy nem lehetett-e minimumpont az eredeti X intervallum határán (például úgy, hogy az algoritmust újraindítjuk egy $\hat{X} \supset X$ intervallummal. Másik megoldás lehet, ha az 5a lépésben a törlés helyett az aktuális intervallumot helyettesítjük a megfelelő lapjával. Ekkor nincs szükség az $\hat{X}$ intervallummal való ellenőrzésre.

Az 5b pontbeli kivágási teszt olyan részintervallumokat dob el, amelyekre az $f(x)$ függvény lehetséges legkisebb értéke is nagyobb, mint δ. A δ értékét megválaszthatjuk a feladatra vonatkozó előzetes információink alapján, de adaptív módon is: kezdetben legyen $\delta = \max F(X)$,

majd minden vágásnál $\delta = \min(\delta,\ \max F(V_1),\ \max F(V_2))$. Algoritmusunk 5b lépése az utóbbi eljárással biztos nem dob ki olyan részintervallumot, amelyben globális minimumpont van. Teszteredmények az 5a, 5b lépések nélkül, illetve az 5a lépéssel:

	S5	S7	S10	H3†	H6†	GP†	RB	SHCB	RCOS
STU	0.4	0.7	1.4	244.3	249.8	199.5	0.1	142.7	0.1
NFE	90	186	204	11453	11319	10499	56	9024	98
NDE	–	–	–	–	–	–	–	–	–
LLI	48	137	166	5000	5000	5000	28	5000	47
EFF	0.3	0.6	0.5	97.5	49.0	52.8	0.3	77.5	0.8

	S5	S7	S10	H3	H6	GP	RB	SHCB	RCOS
STU	1.2	1.7	2.5	9.5	75.2	746.8	0.1	0.9	0.4
NFE	86	92	94	722	2288	34850	56	384	98
NDE	205	219	227	1158	8141	46355	63	540	149
LLI	3	5	10	361	1238	5000	9	194	29
EFF	1.0	1.0	0.9	16.0	45.1	408.1	0.6	7.9	2.1

8.4. Intervallumos Newton módszer

Az $f(x)$ függvény befoglalását kiszámítjuk. Feltételezzük, hogy $f'(x)$ folytonos függvény az $[a,b]$ intervallumon, és

$$0 \notin \{f'(x), x \in [a,b]\} \ \text{ és } \ f(a)f(b) < 0.$$

Ha az $f(x)$ zérushelyének egy X_n befoglalása ismert, egy jobb X_{n+1} befoglalást a következő iterációs képlettel kaphatunk:

$$X_{n+1} := \left(m(X_n) - \frac{f(m(X_n))}{F'(X_n)}\right) \cap X_n,$$

ahol $m(X)$ az X intervallum egy belső pontja (például a középpontja). Tekintsük az $f(x) = \sqrt{x} + (x+1)\cos(x)$ függvényt a $[2,3]$ intervallumon. A kapott iterációs sorozat az intervallumok $w(X_k)$ szélességével együtt:

k	X_k	$w(X_k)$
1	[2,0, 3,0]	1,0
2	[2,0, 2,3]	0,3
3	[2,05, 2,07]	0,02
4	[2,05903, 2,05906]	0,00003
5	[2,059045253413, 2,059045253417]	0,000000000004

Optimalizálási feladatokra nyilván a célfüggvény deriváltjára kell a képleteinket alkalmazni, hiszen annak a zérushelyeit keressük. Ekkor az iterációs formula a következő lesz:

$$X_{n+1} := \left(m(X_n) - \frac{f'(m(X_n))}{F''(X_n)}\right) \cap X_n.$$

Itt $f'(x)$ a célfüggvény deriváltja, $F''(X)$ pedig a második derivált befoglaló függvénye. Vegyük észre, hogy az iterációs képletünk nem függ közvetlenül magától a célfüggvénytől. Ez rendben is van abból a szempontból, hogy nyilván azonos iterációs sorozatot várunk $f(x)$-re, és annak eltoltjára, $f(x) + c$-re.

Idézet a C-XSC Toolbox könyvből[4] az intervallumos Newton módszernek az adott példára való használatával:

```
#include "interval.h"                /* include interval arithmetic package */
#include "imath.h"                   /* include interval standard functions */

interval F (real& x) {
  return sqrt(x) + (x+1) * cos(x);
}

interval Deriv (interval& x) {
  return (1 / (2 * sqrt(x)) + cos(x) - (x+1) * sin(x));
}

int Criter (interval& x) {           /* computing F(a) * F(b) < 0           */
  interval Fa, Fb;                   /* using point intervals               */
  Fa = Inf(x);                       /* operator <= is the relational       */
  Fb = Sup(x);                       /* opreator 'element of'               */
  return (Sup(Fa*Fb) < 0.0 && !(0 <= Deriv(x)));
}

main() {
  interval y, y_old;
  real mid (interval&);              /* prototype of the midpoint function  */

  cout << "Please enter starting interval:"; cin >> y;
  while (Inf(y) != Sup(y)) {
    if (Criter(y)) {
      do {
        y_old = y;
        cout << "y = " << y << "\n";
        y = (mid(y)-F(mid(y))/Deriv(y)) & y;/* The iteration formula       */
      }                                    /* & is the intersection       */
      while (y != y_old);
    }
    else {
      cout << "Criterion not satisfied! \n";
    }
    cout << "Please enter starting interval: ";
    cin >> y;
  }
}
```

8.5. Példák

1. Az intervallumos Newton módszer működésének illusztrálására tekintsük az $f(x) = x^2 - x$ függvényt. Ez egy egyszerű parabola, amelynek tengelye párhuzamos az y tengellyel, és

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

amelynek két zérushelye a 0 és az 1. A függvény minimumpontja a 0,5, ahol itt a függvényérték -0,25. A célfüggvényünk deriváltja az $f'(x) = 2x - 1$, második deriváltja pedig $f''(x) = 2$.

Tekintsük először az $X_0 = [0, 1]$ induló intervallumot, az iteráció első lépése erre:

$$X_1 = \left(m(X_0) - \frac{f'(m(X_0))}{F''(X_0)} \right) \cap X_0 = \left(0,5 - \frac{0,0}{[2,2]} \right) \cap [0,1] = [0,5, 0,5] \cap [0,1] = [0,5, 0,5].$$

Ez azt jelenti, hogy pontos aritmetikával az intervallumos Newton módszer egy lépésben meg tudja határozni egy kvadratikus függvény minimumát abszolút pontosan. A kifelé kerekítés ezen nyilván ront, de ezzel együtt is nagyon hatékony eszköz ez az optimalizálásban. Nyilván általában nem kvadratikus függvényt kell optimalizálnunk, de mivel a sima függvényeknek egy pont kis környezetében a kvadratikus közelítés tetszőlegesen jó, ezért az intervallumos Newton módszertől hasonlóan jó hatékonyságot várhatunk sima nemlineáris optimalizálásban.

Említésre méltó az is, hogy példánkban nem volt túlbecslés az érintett függvényekben, mert mind a lineáris, mind a konstans függvényhez (a kifelé kerekítés leszámítva az implementációban) pontos befoglaló függvényt kapunk már a természetes intervallum kiterjesztéssel is.

Tekintsük most az $X_0 = [0, 2]$ kezdőintervallumot, erre a következőt kapjuk:

$$X_1 = \left(m([0,2]) - \frac{f'(m([0,2]))}{F''([0,2])} \right) \cap X_0 = \left(1 - \frac{1,0}{[2,2]} \right) \cap [0,2] = [0,5, 0,5] \cap [0,2] = [0,5, 0,5].$$

Ebből az látszik, hogy az előző nagyszerű eredményben nem volt annak szerepe, hogy a kiinduló intervallum középpontja volt a keresett minimumpont. Tekintsünk most egy olyan intervallumot, amely nem tartalmaz minimumpontot, $X_0 = [1, 2]$:

$$X_1 = \left(m([1,2]) - \frac{f'(m([1,2]))}{F''([1,2])} \right) \cap [1,2] = \left(1,5 - \frac{2,0}{[2,2]} \right) \cap [1,2] = [0,5, 0,5] \cap [1,2] = \emptyset.$$

Az intervallumos Newton módszer tehát igazolta, hogy a keresési tartományban nincs minimumpont.

2. Vegyük most az előző példa célfüggvényének a négyzetét: $f(x) = x^4 - 2x^3 + x^2$. Ennek nyilván a 0 és az 1 pontok a minimumpontjai. Az első és a második derivált függvény: $f'(x) = 4x^3 - 6x^2 + 2x$, illetve $f''(x) = 12x^2 - 12x + 2$. Első keresési intervallumként tekintsük az $X_0 = [0, 2]$ intervallumot, ami tehát mindkét minimumpontot (és köztük az egyetlen maximumpontot is) tartalmazza. Erre az intervallumos Newton módszerrel a következő eredményt kapjuk:

$$X_1 = \left(m(X_0) - \frac{f'(m(X_0))}{F''(X_0)} \right) \cap X_0 = \left(1 - \frac{0,0}{[-22,50]} \right) \cap [0,2] = [-\infty, \infty] \cap [0,2] = [0,2].$$

Ez a példa azt mutatja, hogy ha a kiindulási intervallumban több szélsőérték is van, akkor az intervallum nem változik. Figyeljük meg, hogy a metszetképzés kellett ahhoz, hogy a keresési intervallum ne nőjön. A második derivált most egy kvadratikus függvény, amihez a befoglaló függvény általában csak túlbecsléssel adható meg. Esetünkben az értékkészlet, $[-1, 26]$ lényegesen kisebb, mint a kapott befoglalás: $[-22, 50]$. Ennek ellenére az értékkészlettel is a fenti eredményt kaptuk volna, mivel az értékkészlet is tartalmazza a nullát.

A második derivált befoglalására a következő értékeket kapjuk:

$$F''([0,9, 1,1]) = [-1,48, 6,02],$$

illetve

$$F''([0,99, 1, 01]) = [1, 6412, 2, 3612].$$

Sajnos ezen a példa se igazolja azt a közkeletű vélekedést, hogy az intervallumos Newton módszert akkor érdemes használni, ha az argumentum intervallum szélessége egynél kisebb. Az elmondottak miatt csak a második esetben számíthatunk arra, hogy a keresési intervallumunk méretét csökkenteni tudjuk. Ekkor az eredményünk az [1, 1] intervallum. Ennek a magyarázata pedig az, hogy a keresési intervallum középpontjában az első derivált értéke nulla, másrészt a második derivált értékei mindenütt pozitívak, így a szélsőértéket a függvény csak a középpontban veheti fel.

Tekintsünk akkor most egy olyan intervallumot, amelynek felezőpontja nem megoldás: [0,98, 1,01]. Erre az intervallumos Newton módszerrel azt kapjuk, hogy

$$X_1 = \left(m(X_0) - \frac{f'(m(X_0))}{F''(X_0)}\right) \cap X_0 = \left(0,995 - \frac{-0,0098505}{[1,4048,\ 2,4812]}\right) \cap [0,98, 1,01] =$$

$$= (0,995 + [0,003970,\ 0,007012]) \cap [0,98, 1,01] = [0,99897, 1,002012] \cap [0,98, 1,01] =$$

$$= [0,99897, 1,002012].$$

Ezzel a megoldásunkra egy meglehetősen szűk intervallumot kaptunk: a keresési intervallum kb. tizedére (a szélessége 0,03-ról 0,003042-re) csökkent, és ezzel együtt a bizonytalanságunk is a minimum helyét illetően.

PÉLDA. A SIAM (Ipari és Alkalmazott Matematikai) Társaság 2002-ben 10 numerikus feladatot tűzött ki[5]. Feladatonként 10 helyes decimális jeggyel 100 Dollárt lehetett nyerni. A negyedik megadott feladat a következő függvény minimalizálása volt:

$$\exp(\sin(50x)) + \sin(60e^y) + \sin(70\sin(x)) + \sin(\sin(80y)) -$$
$$- \sin(10(x+y)) + \frac{1}{4}(x^2 + y^2).$$

A feladat megoldására egy intervallum aritmetikára alapuló korlátozás és szétválasztás módszert használtunk. A kapott eredmény a $[-10.0, 10.0]$ keresési tartományon a globális minimum értékére a következő alsó- és felső korlátokat adta:

$$[\underline{-3.306868647475}316, \underline{-3.306868647475}196].$$

Az eredményben a kiemelt első 13 jegy matematikai bizonyítóerővel igazoltan helyes. Ehhez 0.26 másodperc CPU-idő, minimális memóriaigény (75 részintervallum tárolására volt szükség), 1975 célfüggvény-, 1158 gradiens- és 92 Hesse-mátrix kiértékelés kellett mindössze.

8.6. Ellenőrző kérdések és gyakorló feladatok

1. Ki lehet-e terjeszteni a négy alapműveletet valós számokról intervallumokra?

2. Igaz-e, hogy két intervallum összege pontosan azokat a pontokat tartalmazza, amelyek előállnak a két argumentum-intervallumbeli pontok összegeként?

[5]Nick Trefethen: A Hundred-Dollar, Hundred-digit Challenge. SIAM News 35(2002)

3. Milyen információra támaszkodik a monotonitási teszt?
4. Mi okozza a befoglaló függvények durva becslését?
5. Mi az a kifelé-kerekítés?
6. Hány féle kerekítést enged meg az IEEE processzor-szabvány?
7. Igaz-e, hogy egy szigorúan monoton függvény deriváltjának befoglaló függvénye nem tartalmazza a nullát?
8. Igaz-e, hogy az intervallumos befoglaló függvény számítása mindig tovább tart, mint az eredeti valós függvényé?
9. Melyik eljárással kapjuk a legjobb befoglaló függvényt az X*X-X függvényre a [-1,1] intervallumra? (X*X nem négyzetreemelés, azonos pontosság esetén a gyorsabb eljárás a jobb)
10. Melyik eljárással kapjuk a legrosszabb befoglaló függvényt az X*X-X függvényre a [-1,1] intervallumra? (X*X nem négyzetreemelés, azonos pontosság esetén a lassabb eljárás a rosszabb)
11. Melyik eljárással kapjuk a legjobb befoglaló függvényt az X*X-X függvényre a [1,10] intervallumra? (X*X nem négyzetreemelés, azonos pontosság esetén a gyorsabb eljárás a jobb)
12. Melyik eljárással kapjuk a legrosszabb befoglaló függvényt az X*X-X függvényre a [1, 10] intervallumra? (X*X nem négyzetreemelés, azonos pontosság esetén a lassabb eljárás a rosszabb)
13. Melyik eljárással kapjuk a legjobb befoglaló függvényt az X*X-X függvényre a [0.4, 0.6] intervallumra? (X*X nem négyzetreemelés, azonos pontosság esetén a gyorsabb eljárás a jobb)
14. Melyik eljárással kapjuk a legrosszabb befoglaló függvényt az X*X-X függvényre a [0.4, 0.6] intervallumra? (X*X nem négyzetreemelés, azonos pontosság esetén a lassabb eljárás a rosszabb)
15. Mutassunk példát arra az esetre, amikor $w(X) + w(Y) \neq w(X + Y)$!
16. Mi az a szubdisztribúciós szabály?
17. Invertálható-e az intervallumos összeadás?
18. Azonos-e az X intervallum négyzete X * X -el?
19. Reprezentálhatók-e mindig a valós műveletek intervallum-műveletekkel?
20. Igaz-e, hogy ha X része Y-nak, akkor minden befoglaló függvényre F(X) is része F(Y)-nak?
21. Milyen programozási nyelvek alkalmasak intervallum aritmetikával való számolásra?
22. Igazoljuk, hogy a vázolt eljárás kerekítés nélküli aritmetika esetén pontosan a derivált értékét adja.

23. Adjunk becslést arra, hogy az automatikus differenciálás és az analitikusan megadott derivált műveletigénye hogyan viszonyul egymáshoz!

24. Hogyan működik a belsőfüggvény deriválása esetünkben?

25. Hogyan lehet eljárásunkat többváltozós függvények parciális deriválására kiterjeszteni? (Mit kell konstansnak, és mit változónak tekinteni?)

26. Határozzuk meg a másodrendű deriváltak előállításához szükséges aritmetikát!

27. Vizsgáljuk meg annak lehetőségét, hogy az automatikus differenciáláshoz hasonlóan felépíthető (?) optimalizálási aritmetikát milyen feladatokra lehet alkalmazni!

28. Az intervallumokra definiált műveletek pontosak, de pontosak-e az ezekkel felépített függvények ? Mutassunk példát!

29. Milyen függvények intervallumon vett értékkészlete számítható pusztán az intervallum végpontjaiban felvett függvényértékekkel?

30. Mit mondhatunk a konvex, és a konkáv függvények befoglaló-függvényeiről?

31. Próbáljuk meg a valós számokra ismert négy alapműveletet általánosítani kompakt valós intervallumokra!

32. Melyik következő állítás igaz az $f(x) = x^6(\sin(1/x)+3)$, ha x nem egyenlő 0-val, és $f(0) = 0$ függvény 0 minimumhelyére?

33. Van-e olyan függvény, amelynek minden pontja nem-szeparált minimumhely és egyben nem-szeparált maximumhely is?

34. Hány globális minimumpontja lehet egy egyváltozós, a [0,1] intervallumra korlátozott nemlineáris optimalizálási feladatnak?

35. Hány szeparált helyi minimumpontja lehet egy egyváltozós, a [0,1] intervallumra korlátozott nemlineáris optimalizálási feladatnak?

36. Hány globális minimuma lehet egy egyváltozós, a [0,1] intervallumra korlátozott nemlineáris optimalizálási feladatnak?

37. Hogyan lehet maximalizálási feladatot minimalizálásira visszavezetni?

38. Milyen optimalizálási módszert érdemes használni, ha az adott optimalizálási feladat megoldásához csak a célfüggvényt kiszámító szubrutin áll rendelkezésre?

39. Milyen optimalizálási módszert érdemes használni, ha az adott optimalizálási feladat megoldásához a célfüggvényt és a gradiensét kiszámító szubrutin áll rendelkezésre?

40. Milyen optimalizálási módszert érdemes használni, ha az adott optimalizálási feladat megoldásához a célfüggvényt, gradiensét és a Hesse mátrixát kiszámító szubrutin is rendelkezésre áll?

41. Milyen optimalizálási módszert érdemes használni, ha az adott optimalizálási feladat megoldásához a célfüggvény befoglaló függvényét kiszámító szubrutin áll rendelkezésre?

42. Milyen programozási nyelvek alkalmasak intervallum aritmetikával való számolásra?

43. Igaz-e, hogy ha X része Y-nak, akkor minden befoglaló függvényre F(X) is része F(Y)-nak?

44. Igaz-e, hogy egy szigorúan monoton függvény deriváltjának befoglaló függvénye nem tartalmazza a nullát?

45. Milyen intervallumot kapunk eredményül, ha az $f(x) = (2x - 1) * (x^2 - x)$ függvény természetes intervallum-kiterjesztését a $[-1, 1]$ intervallumon kiértékeljük?

46. Milyen információra támaszkodik az intervallumos korlátozás és szétválasztás típusú globális optimalizálási algoritmusban a monotonitási teszt?

9. fejezet

Automatikus differenciálás

Ahogy a korábbiakban láttuk, a differenciál-hányadosoknak fontos szerepük van a nemlineáris optimalizálásban, de a numerikus matematika számos területen is szinte elengedhetetlen a használatuk. Ide tartozó problémák vannak a nemlineáris egyenletmegoldásban, az irányításelméletben és az érzékenység-vizsgálatban is. Talán a legismertebb eset a függvények zérushelyének megkeresése, itt a derivált használatával működő Newton-Rawson eljárás konvergencia-sebessége lényegesen jobb, mint a deriváltakat nem használó szelő- vagy húrmódszeré.

Ma már egyes programozási nyelvek (pl. a PASCAL-XSC[6]) is támogatják az automatikus differenciálást megfelelő adattípussal és műveletekkel, és számos szoftver is használja ezt a deriválási módszert[7].

9.1. Deriváltak a számítógépeken

A leggyakrabban használt két módszer a deriváltértékek előállítására azok numerikus közelítése és a "kézzel" való derivált-meghatározás a deriválási szabályok alkalmazásával. A legtöbb numerikus matematikai monográfia és a professzionális numerikus programcsomagok többsége is ezt a két utat javasolja. Mindkét módszernek vannak azonban olyan gyengéi, amelyek számos feladatban lehetetlenné vagy értelmetlenné teszik alkalmazásukat. A ritka kivételek egyike Skeel és Keiper könyve[8], amely a szimbolikus differenciálással szemben is az automatikus deriválást javasolja.

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

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

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

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

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

[7]Pl. a D. Ratz: Automatische Ergebnisverifikation bei globalen Optimierungsproblemen. (Doktori értekezés, Karlsruhei Egyetem, 1992.) című disszertációban leírt optimalizálási eljárás, amely csak a célfüggvény megadását igényli.

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

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

Hátránya viszont, hogy

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

– a gyorsan változó deriváltak becslésére alkalmatlan.

A hüvelykszabály szerint — hacsak lehetséges — érdemes előállítani a deriváltakat számító szubrutinokat. Ezen eljárás előnye, hogy

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

+ a gyorsan változó deriváltértékek is jól meghatározhatók.

A hátránya ezzel szemben, hogy

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

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

Itt kell megjegyezni, hogy a számítógépes algebrarendszerek (mint például a Mathematica, a Maple vagy a Derive) szimbolikus manipulációval elő tudják állítani a képlettel adott függvények deriváltjait. Így ez az előkészítő munka legalább számítógépesíthető, tehát nem feltétlenül kell "kézzel" végrehajtani. Az ilyen szimbolikus deriválás, a vele járó egyszerűsítés és a programozási nyelvre való alakítás időigénye nagyon változó, mindenesetre a számítógépes algebrarendszerek sokat fejlődtek ezen a téren az utóbbi időben[9].

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

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

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

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

+ a levágási hiba miatt nem vesznek el értékes jegyek,

+ a gyorsan változó deriváltak meghatározására is alkalmas, és

[9]Lásd Iri, M.: History of automatic differentiation and rounding error estimation, in: Griewank, A., G. Corliss (Eds.): Automatic Differentiation of Algorithms: Theory, Implementation, and Application. SIAM, Philadelphia, 1991. 3-16.

1.. táblázat. Néhány alapművelet és elemi függvény differenciálása

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

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

Maga az ötlet nem nagyon bonyolult, és jellemző módon többen egymástól függetlenül rátaláltak[10]. Ha valaki kedvet érez hozzá, maga is megpróbálhatja az automatikus differenciálást újra felfedezni: az előző feltételeket teljesítő eljárást kell megadni (eddig nem árultunk el semmi lényegeset a trükkből).

9.2. Az ötlet.

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

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

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

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

[10]Ostrovskij, G.M., Ju. M. Wolin, W.W. Borisov: Über die Berechnung von Ableitungen, Wissenschaftliche Zeitschrift der Technischen Hochschule für Chemie, Leuna-Merseburg 13(1971) 382-384 és Wengert, R.E.: A simple automatic derivative evaluation program, Communications of the ACM 7(1964) 463-464.

9.3. Kiterjesztések.

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

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

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

9.4. Az automatikus differenciálás két változata.

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

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

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

sorrendet követi.

A fordított eljárás az ellentétes irányban alkalmazza a láncszabályt:

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

A fordított algoritmus előnye abban van, hogy ez a végrehajtási sorrend lehetővé teszi többváltozós függvények differenciálása során bizonyos szükségtelen műveletek elhagyását. Ennek az az ára (amit a következő szakasz adatai is alátámasztanak), hogy a fordított algoritmus

tárigénye magasabb, és a sima algoritmus egymenetes végrehajtásával szemben két menetet igényel.

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

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

A sima eljárás eközben esetleg többször is végrehajtja ugyanazt a műveletet, viszont nem igényli a kiszámítási fa létrehozását és tárolását. A fordított algoritmus ezzel szemben először meghatározza az f_i értékeket és a kiszámítási fát, majd ennek segítségével előállítja a $d_i = \partial f / \partial f_i$ értékeket:

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

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

9.5. Művelet- és tárigény.

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

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

9.6. Az automatikus differenciálás veszélyei.

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

1. A zérus gyökök esete. Tekintsük az $f(x) = \sqrt{x_1^4 + x_2^4}$ függvényt. Ez differenciálható, és a gradiense a $(0., 0.)^T$ pontban $(0., 0.)^T$. Az automatikus differenciálás a négyzetgyök művelethez

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

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

azonban nem tud értéket rendelni, ha a gyök argumentuma nulla. A felhasználó számára ilyen esetekben az a leghasznosabb, ha az illető implementáció felhívja a figyelmet erre a hibalehetőségre, pl. az IEEE aritmetikát támogató számítógépekben a NaN (Not a Number) érték hozzárendelésével.

2. A programelágazás esete. Tekintsük az alábbi utasítást:

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

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

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

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

Automatikus differenciálással (is) $\lim_{k\to\infty} f_k'(0) = 1$, habár a valódi $f(x)$ határfüggvényre $f'(0) = 0$. Ebben az esetben is csak azt lehet tanácsolni, hogy a jelenség ismeretében az automatikus differenciálással nyert értékeket ellenőrizni kell. Ehhez viszonylag kényelmesen használható elméleti eredmények is rendelkezésre állnak[11].

[11]Fischer, H.: Special problems in automatic differentiation, in: Griewank, A., G. Corliss (Eds.): Automatic Differentiation of Algorithms: Theory, Implementation, and Application. SIAM, Philadelphia, 1991, 43-50.

9.7. Az automatikus differenciálás implementálása.

A már említett PASCAL-XSC beépített adattípusainak és kiterjesztett alapműveleteinek a használata a legegyszerűbb. A felhasználónak mindössze a megfelelő adattípusokat kell megváltoztatnia. A FORTRAN-90 és C++ nyelvekben ezek az új adattípusok és a kiterjesztett műveletek megvalósítása után ugyanolyan kényelmesen lehet az automatikus differenciálás sima algoritmusát alkalmazni, mint a PASCAL-XSC támogatásával.

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

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

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

⋮

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

⋮

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

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

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

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

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

- Az ADOL-C egy C++ nyelven írt rendszer, amely C vagy C++ nyelvű algoritmusok differenciálására alkalmas sima és fordított eljárással is. A forráskód és a dokumentáció Andreas Griewank címén érhető el (Argonne National Labs, Argonne, IL 60439, USA, illetve elektronikus postával griewank@antares.mcs.anl.gov).
- A MAPLE nevű számítógépes algebrarendszer az 5.1-es változatától kezdve a szimbolikus deriválás mellett képes az automatikus differenciálásra is (a sima algoritmussal). Az "optimize" rutinja csökkentheti a műveletigényt, és az eredményt FORTRAN vagy C nyelven is ki tudja adni.

Meg kell még említeni, hogy az automatikus differenciáláshoz természetes módon kapcsolható a kerekítési hibák becslése és a számított deriváltak alsó- és felsőkorlátjainak meghatározása is. Az utóbbi feladatok (részben az intervallum-aritmetikára támaszkodva) szintén kényelmesen megoldhatók számítógépen. Az automatikus differenciálásnak bő irodalma érhető el, említésre méltó a következő két cikk, illetve könyv:

- Kedem, G.: Automatic differentiation of computer programs, ACM Transactions on Mathematical Software 6(1980) 150-165.
- Rall, L.B.: Automatic Differentiation: Techniques and Applications. Lecture Notes In Computer Science, Vol. 120, Springer-Verlag, Berlin, 1981.

9.8. Példák

9.9. Ellenőrző kérdések és gyakorló feladatok

Programozási feladat:

$\star$ Írjuk meg az automatikus differenciálás szubrutinjait, és teszteljük néhány függvényen!

10. fejezet

A feladat egyszerűsítése nemlineáris transzformációkkal

Ebben a fejezetben egy olyan, szimbolikus számítógépes programokkal végrehajtható algoritmust tárgyalunk, amely alkalmas nemlineáris optimalizálási feladatokban rendundáns optimalizálandó változók megkeresésére, a feladat egyszerűsítésére és az átalakított feladat megoldásából az eredeti feladat megoldásának előállítására.

10.1. Bevezetés

Tekintsük a következő célfüggvényt:

$$F(R_{aw}, I_{aw}, B, \tau) = \left[\frac{1}{m} \sum_{i=1}^{m} |Z_L(\omega_i) - Z'_L(\omega_i)|^2 \right]^{1/2}, \tag{1}$$

ahol $Z_L(\omega_i) \in \mathbb{C}$ a mért komplex impedancia érték, $Z'_L(\omega_i)$ a modellezett impedancia az ω_i frekvencián $i = 1, 2, \ldots, m$ és R_{aw}, I_{aw}, B, τ a modell paraméterek. A modellfüggvény

$$Z'_L(R_{aw}, I_{aw}, B, \tau, \omega) = R_{aw} + \frac{B\pi}{4.6\omega} - \imath \left(I_{aw}\omega + \frac{B \log(\gamma\tau\omega)}{\omega} \right), \tag{2}$$

ahol $\gamma = 10^{1/4}$ és $\imath$ a képzetes egység. Ez a paraméterbecslési feladat eredeti megfogalmazása, a konkrét fizikai paraméterekkel[12]. Erre a modell identifikációs feladatra a következő, az eredetivel ekvivalens modellfüggvényt találtunk:

$$Z'_L(R_{aw}, I_{aw}, B, A, \omega) = R_{aw} + \frac{B\pi}{4.6\omega} - \imath \left(I_{aw}\omega + \frac{A + 0.25B + B \log(\omega)}{\omega} \right). \tag{3}$$

Ez utóbbi lineáris a paraméterekben, a célfüggvény értéke végtelenbe tart ahogy bármely paraméter abszolút értéke nő, és emiatt a feladatnak egyetlen minimumpontja van. Így a szélsőértéket közvetlenül meg lehet határozni iteratív eljárás nélkül is: a gradiens zérushelyét egy véges algoritmussal meg tudjuk adni. Ez általában sokkal egyszerűbb feladat, mint a globális optimalizálási feladat megoldása.

A (3) létezésének az a magyarázata, hogy (1) voltaképp két, egymástól részben független négyzetösszeg minimalizálása: a valós és a képzetes részé. A B paramétert meg lehet kapni

[12]Hantos, Z., Daróczy, B., Csendes, T., Suki, B. and Nagy, S. (1990), Modeling of Low-frequency Pulmonary Impedance in the Dog, J. of Applied Physiology **68**, 849-860.

pusztán a valós rész illesztésével. Ez aztán felhasználható a $B\log(\gamma\tau\omega)$ szorzatban. A (2)-ből a (3)-at adó transzformáció az $A = B\log(\tau)$. Vegyük észre, hogy γ nem meghatározandó paraméter, ezért a logaritmusát (0.25) konstansként használhatjuk.

A hagyományos optimalizálási módszerekkel szemben a bemutatandó eljárás a célfüggvény ismert alakját, képletét használja ki. Az ilyen algoritmusok meglehetősen ritkák. Stoutemyer egy analitikus optimalizálási módszert tárgyalt, egy olyan programot, amely a szimbolikus alakban adott feladat osztályát (mint lineáris, kvadratikus, szeparábilis,...) tudta meghatározni.[13]

Hansen, Jaumard és Lu algoritmusa[14] egyes feltételes optimalizálási feladatok megoldására alkalmas korlátozás és szétválasztás keretben. Ezzel számos feladat pontos megoldását tudták előállítani. Redkovskii és Perekatov hasonló nemlineáris transzformációkat adott meg pozitív definit Hesse-mátrixok eléréséhez a helyi minimalizálásra szolgáló Newton-módszer számára[15].

10.2. Unimodalitás és paraméter transzformációk

DEFINÍCIÓ. Az egyváltozós feltétel nélküli optimalizálási feladat $f(x)$ célfüggvényét *unimodálisnak* hívjuk, ha van olyan x^* pont, hogy minden $x_1 < x_2$ pontpárra ha $x_2 < x^*$, akkor $f(x_1) > f(x_2)$, ha pedig $x^* < x_1$, akkor $f(x_1) < f(x_2)$.

Ha $f(x)$ unimodális és folytonos, akkor $f(x^*)$ az egyetlen helyi minimuma, és így a globális minimuma. Ráadásul az a két állítás, hogy egy egyváltozós folytonos függvény unimodális, és az, hogy egyetlen helyi minimuma van és nincs helyi maximuma — ekvivalensek.

Az unimodalitást általában nem definiálják többváltozós függvényekre. Ennek az az oka, hogy azok az egyváltozós optimalizálási módszerek, amiket az unimodalitási tulajdonságra alapoznak, nem általánosíthatók a többváltozós esetre.

DEFINÍCIÓ. Egy n-dimenziós folytonos függvényt akkor nevezünk unimodálisnak az $X \subseteq \mathbb{R}^n$ nyílt halmazon, ha létezik végtelen folytonos görbéknek egy olyan halmaza, hogy a görbesereg homeomorfikus leképezése az n-dimenziós tér polárkoordináta-rendszerének, és az $f(x)$ függvény szigorúan monoton nő a görbék mentén.

Ez a definíció más szavakkal azt jelenti, hogy egy függvény akkor unimodális, ha egyetlen vonzáskörzete van az X nyílt halmazon. Egy kétszer folytonosan differenciálható (sima) függvényre unimodalitás azt jelenti, hogy csak egy helyi minimumpontja lehet, és nincs helyi maximumpontja X-ben. Ez a definíció jól illeszkedik az optimalizálási szóhasználathoz, mivel a multimodalitást és a több szélsőérték meglétét szinonimának tekintik.

Másrészt egy n-dimenziós unimodális függvény nem feltétlen teljesíti az egydimenziós unimodalitás feltételeit az n-dimenziós tér minden egyenese mentén. Tekintsük például a jól ismert Rosenbrock- vagy banán-függvényt: $f(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$. Ez egyértelműen unimodális, mégis az egydimenziós unimodalitás nem teljesül például az $x_2 = 1$ egyenes mentén.

[13]Stoutemyer, D.R. (1975), Analytical Optimization Using Computer Algebraic Manipulation, ACM Transactions on Mathematical Software **1**, 147-164.

[14]Hansen, P., Jaumard, B., and Lu, S. H. (1991), An Analytical Approach to Global Optimization, Mathematical Programming **52**, 227-254.

[15]Perekatov, A.E. and N.N. Redkovskii (1989), Method for Minimization of Unimodal Non-convex Functions (in Russian), Dokladi AN USSR Ser. A Physical-Mathematics and Technical Sciences **10**, 36-38; Redkovskii, N.N. (1989), Nonlinear Transformations of Coordinates in Unconstrained Optimization Problems (in Russian), Issledovanii Operatsii UAS **34**, 60-65.

A következő tétel a nemlineáris függvények unimodalitása és a homeomorf (kölcsönösen egyértelmű folytonos) változó transzformációk viszonyát tárgyalja. Ennek során először az $y = h(x)$ transzformációkat vizsgáljuk.

1. TÉTEL. Az $f(x)$ folytonos függvény pontosan akkor unimodális az n-dimenziós valós térben, ha létezik a változóknak olyan $y = h(x)$ homeomorf transzformációja, hogy $f(x) = f(h^{-1}(y)) = y^T y + c$, ahol c egy valós konstans, és az origó benne van a $h(x)$ leképezés S értékkészletében.

Tegyük most fel, hogy az $f(x)$ függvényben az x_i változó mindenütt egy $h(x)$ részképlet formájában fordul elő. Vizsgáljuk a továbbiakban azt a transzformációt, amelyik az $f(x)$-ben a $h(x)$ minden előfordulását y_i-re cseréli, és a többi x_j változót átnevezi y_j-re ($i \neq j$).

2. TÉTEL. Ha $h(x)$ sima és szigorúan monoton az x_i változóban, akkor az ennek megfelelő $y = h(x)$ transzformáció egyszerűsíti az optimalizálási feladatot abban az értelemben, hogy $h(x)$ minden előfordulása az $f(x)$ célfüggvényben helyettesíthető egy új változóval. Ekkor $f(x)$ minden helyi minimum (maximum) pontjának a transzformált $g(y)$ függvény helyi minimum (maximum) pontja felel meg.

3. TÉTEL. Amennyiben $h(x)$ sima és szigorúan monoton az x_i változóban, továbbá $h(x)$ értékkészlete egyenlő $\mathbb{R}$-el, akkor a transzformált $g(y)$ függvény minden y^* helyi minimumpontjához (maximumpontjához) van olyan x^*, hogy x^* transzformáltja y^*, és x^* helyi minimumpontja (maximumpontja) az $f(x)$ függvénynek.

1. ÁLLÍTÁS. Ha egy x_i változó a sima $f(x)$ függvény képletében mindenütt egy $h(x)$ képlet alakjában fordul elő, akkor a $\partial f(x)/\partial x_i$ parciális derivált felírható $(\partial h(x)/\partial x_i)p(x)$ alakban, ahol $p(x)$ folytonosan differenciálható.

2. ÁLLÍTÁS. Ha az x_i és x_j változók a sima $f(x)$ függvény képletében mindenütt egy $h(x)$ képlet alakjában fordulnak elő, akkor a $\partial f(x)/\partial x_i$ és a $\partial f(x)/\partial x_j$ parciális deriváltak felírhatók $(\partial h(x)/\partial x_i)p(x)$, illetve $(\partial h(x)/\partial x_j)q(x)$ alakban, ahol $p(x) = q(x)$ folytonosan differenciálható.

Ezek az elméleti eredmények alapján tehát olyan eljárást lehet összeállítani, amely a célfüggvény egyszerűsítését tudja elérni. Ennek főbb lépései:

1. *Parciális deriválás.* Minden változóra határozzuk meg a célfüggvény parciális deriváltjait.
2. *Szorzatra bontás.* Bontsuk szorzat alakra a parciális deriváltakat.
3. *Ellenőrzés.* Ellenőrizzük, hogy a szorzatok egyes tényezői lehetnek-e olyan $h(x)$ függvények, amelyek valamely x_i változó összes előfordulását jellemzik $f(x)$ képletében. Ha van ilyen $h(x)$, akkor alkalmazzuk erre az $y_i = h(x)$ transzformációt. A többi változót változatlanul nevezzük át: $y_j = x_j, j = 1, 2, \ldots, i-1, i+1, \ldots, n$.
4. *Transzformáció.* Az így kapott $g(y)$ függvény optimalizálásával előálló y^* megoldást transzformáljuk vissza az eredeti térbe a $h(x)$ transzformáció inverzével. Adjuk meg az eredeti feladat megoldásaként az így kapott x^* pontot.

Az eljárásunk főbb elemei mind elérhetők szimbolikus algebrarendszerekben (mint például a Derive, Maple, Mathematica, mupad stb.): a szimbolikus deriválás, függvények szorzatra

bontása, a szorzatra bontott függvény elemeinek listaszerű felsorolása, határozatlan integrálás a parciális deriváltakból a transzformáló függvény előállításához, a kiszámítási fa konstruálása, abban a $h(x)$ képlet keresése, ...

10.3. Példák

10.3.1. Rosenbrock függvény

Vegyük először a korban már említett Rosenbrock- vagy banánfüggvényt: $f(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$. A feladatra szokás szerint megadott korlátok $-2 \leq x_1, x_2 \leq 2$. Ezek nem jelentenek valódi korlátozást, mert a korlátozás nélküli, és a jelen feladat megoldásai egybeesnek.

Mivel a célfüggvény négyzetösszeg alakú, a megoldás könnyen leolvasható. A második tag minimumát ott veszi fel, ahol $x_1 = 1$. Ha ezt az értéket behelyettesítjük az első tagba, akkor a zárójelen belül elérhető, hogy nullát kapjunk, így a korábbi lépésünk jogossága igazolódott. Ezután x_2-re is az egy érték adódik optimálisnak. Mivel más értékekkel a 0 célfüggvényértéket nem lehet elérni, ezért a kapott helyi minimumpont egyben globális minimumpont is.

Kövessük most az egyszerűsítő eljárásunk lépéseit.

1. A célfüggvény gradiense $[400(x_1^2 - x_2)x_1 - 2(1 - x_1), -200(x_1^2 - x_2)]^T$.

2. A szorzatra bontás lényegében eredménytelen: a parciális deriváltak csak a megadott képlet konstansszorosait engedik meg. Ennek az a következménye, hogy a célfüggvényben csak olyan képletek keresésének van értelme amelyek a transzformálandó változóban lineárisak.

   Az $f(x)$-ben az x_1 változót tartalmazó nem triviális részképletek: x_1^2, $1-x_1$, $x_1^2-x_2$, $(1-x_1)^2$ és $(x_1^2 - x_2)^2$. Az x_2 pedig a következő képletekben fordul elő: $x_1^2 - x_2$, $(x_1^2 - x_2)^2$ és $100(x_1^2 - x_2)^2$.

3. Ellenőrizzük, hogy az eredeti célfüggvényben az $x_1 + c(x_2)$, $\int 400(x_1^2 - x_2)x_1 - 2(1 - x_1)\, dx_1$ közül melyik képlet az, amely még az x_1 minden előfordulását jellemzi! Itt $c(x_2)$ arra utal, hogy az x_1 szempontjából konstans függvény szerepelhet még a képletben, ami tehát függhet az x_2-től. A legbonyolultabb ilyen illeszkedő képlet maga az x_1, ezért ennek az átalakítására nincs szükség.

   Tekintsük most az x_2 változót. Ennek $x_2 + c(x_1)$ függvényéhez van nem triviális illeszkedő képlet, amely x_2 minden előfordulásában szerepel: ez az $x_1^2 - x_2$. Az ebből adódó transzformáció: $y_1 = x_1$, $y_2 = x_1^2 - x_2$. Vegyük észre, hogy ez a transzformáció sima, és monoton a transzformált változóban, így a korábbi tételeink megfelelő módon alkalmazhatók.

4. Az átalakított célfüggvény ebből $g(y) = (1-y_1)^2+100y_2^2$. Ez már szeparált, tehát felbontható $g(y) = g_1(y_1) + g_2(y_2)$ alakban, kevesebb művelettel számítható, és könnyebben optimalizálható. Az egyetlen minimumpontja $y_1^* = 1$, $y_2^* = 0$. Az eredeti feladat megoldása ebből az inverz transzformációval a $x_1 = y_1$, $x_2 = y_1^2 - y_2$ képletekből adódik: $x_1^* = 1$ és $x_2^* = 1$.

Az átalakítás számos előnye mellett jegyezzük meg azt is, hogy a transzformált célfüggvény már unimodális minden egyenes mentén.

10.3.2. $\cos(e^{x_1} + x_2) + \cos(x_2)$

Vegyük most az $f(x) = \cos(e^{x_1} + x_2) + \cos(x_2)$ függvényt. A szimbolikus deriválás és a szorzatra bontás azt jelzi, hogy az $e^{x_1} + c(x_2)$ alakú transzformáció lehetséges. Az $f(x)$ kiszámítási fájából ehhez az $e^{x_1} + x_2$ képlet illeszkedik. Innen az $y_1 = e^{x_1} + x_2$, $y_2 = x_2$ transzformációt kapjuk. Ez nyilvánvalóan nem invertálható.

Az átalakított optimalizálási feladat

$$\min g(y) = \cos(y_1) + \cos(y_2).$$

Ennek vannak olyan maximumpontjai (például az $y_1 = -2\pi$, $y_2 = 0$), amelyeknek nem felel meg az eredeti feladat egyik szélsőérték pontja sem. Vessük et össze a 2. és a 3. Tétellel! Ennek ellenére $g(y)$ szeparált és egyszerűbben optimalizálható. Ráadásul ez az új feladat világosan tükrözi az eredeti feladat szélsőértékei helyének szerkezetét.

10.3.3. A bevezetőbeli feladat

Vizsgáljuk meg most a bevezetőben részletesen tárgyalt feladatot. Az egyszerűség kedvéért hagyjuk el a négyzetgyököt és az $1/m$ szorzótényezőt az (1)-ben megadott függvényből. Használjuk továbbá a valós és a képzetes rész négyzetösszegét a korábbi abszolútérték négyzetösszeg helyett.

Az egyetlen szorzattá bontható parciális derivált ezután a $\partial F/\partial\tau$, és a használható szorzó tényező B/τ. Ez utóbbi integrálja a $B\log\tau$ képletet adja, ami valóban minden előfordulását jellemzi a τ változónak. Épp az ennek megfelelő $A = B\log\tau$ helyettesítés volt az, amely a bevezetőben tárgyalt linearizálást eredményezte.

Ebben az esetben tehát a nemlineáris változó-transzformáció fő haszna az volt, hogy az általában a globális optimalizálás körébe tartozó nemlineáris paraméterbecslési feladatot átalakította egyszerűbben megoldható, egy helyi minimummal rendelkező egyenesillesztési feladattá.

10.3.4. A légzésmechanika Otis-féle modellje

Vegyük ismét az (1) paraméterbecslési feladatosztályt, és vizsgáljuk meg ezt egy másik modellfüggvénnyel, a légzésmechanika korában széles körben tanulmányozott Otis-féle modelljével. Erről egy elektronikai modell segítségével belátták, hogy egy redundáns paramétert tartalmaz[16]. A modellfüggvény:

$$Z_m(s) = \frac{Ds^3 + Cs^2 + Bs + 1}{Gs^2 + Fs} \tag{4}$$

ahol $s = \imath\omega$, és

$$\begin{aligned}
B &= R_C(C_1 + C_2) + R_1C_1 + R_2C_2, && (5)\\
C &= I_C(C_1 + C_2) + [R_C(R_1 + R_2) + R_1R_2]C_1C_2, && (6)\\
D &= I_C(R_1 + R_2)C_1C_2, && (7)\\
F &= C_1 + C_2, && (8)\\
G &= (R_1 + R_2)C_1C_2. && (9)
\end{aligned}$$

[16] Avanzolini, G., and Barbini, P. (1982), Comment on "Estimating Respiratory Mechanical Parameters in Parallel Compartment Models", IEEE Transactions on Biomedical Engineering **29**, 772-774.

Eszerint a feladat optimalizálandó változói R_C, R_1, R_2, I_C, C_1 és C_2. A korábbiakban ismertetett eljárással megadható ugyan egy a feladatot egyszerűsítő átalakítás, $R_1' = R_1C_1$, de a leíró paraméterek száma ettől még nem változik. Ennek az az oka, hogy a tárgyalt módszer csak arra alkalmas, hogy egy változónak az eredeti függvénybeli képletét egyszerűsítse, miközben a többi változó változatlanul marad a célfüggvényben. A jelen esetben viszont csak több paraméter egyidejű átalakításával lehet kimutatni, hogy egyel több paramétere van a modellfüggvénynek, mint amennyi feltétlen szükséges.

Másrészt viszont maguk az (5) - (9) egyenletek is definiálnak egy paraméter transzformációt. Eszerint a modellfüggvény megadható az eredeti 6 helyett 5 paraméter használatával is. Minden új paraméter lineáris függvénye a régiknek (ha csak egyet hagyunk változni közülük). Eszerint pedig teljesülnek a 3. Tétel feltételei. Eszerint a transzformált feladat helyi minimumpontjait vissza lehet alakítani az eredeti feladat minimumpontjaiba. A dimenziók eltérése miatt ez azt fogja jelenteni, hogy az egyszerűsített feladat egy megoldásának az eredeti térben megoldásoknak egy sokasága felel majd meg. Egyben ismét egy jellemzését is megkaptuk az eredeti feladat megoldásai szerkezetének.

A példa tanulsága az, hogy érthetővé vált, hogy miért tartott olyan soká, amire a tudományos szakterület felismerte az eredeti modell redundáns voltát. Másrészt a példa arra is utal, hogy a bemutatott eljáráson túl további lehetőségek vannak a nemlineáris paraméter-transzformáció előtt.

10.4. Ellenőrző kérdések és gyakorló feladatok

1. Mutassunk olyan feladatot, amely példa arra, hogy egy egyszerűsítő transzformációval akár kettővel is lehet csökkenteni egy optimalizálási feladat dimenzióját!

2. Határozza meg a tárgyalt szimbolikus módszerrel hogy milyen egyszerűsítő helyettesítést lehet alkalmazni az $f(x_1, x_2) = e^{(x_1+x_2)^2}$ függvényen, és értelmezze az eredményt!

3. Határozza meg hogy milyen egyszerűsítő helyettesítést lehet alkalmazni az $f(x_1, x_2) = 10^{(2x_1+3x_2)^2}$ függvényen, és értelmezze az eredményt!

4. Jellemezzen egy egyszerű optimalizálási feladatosztályt (ami tetszőleges dimenziójú feladatokat is tartalmaz), amely nem egyszerűsíthető a tárgyalt módszerrel!

5. Adjon meg egy olyan egyszerű nemlineáris optimalizálási feladatot, amelynek csak globális minimumpontjai vannak, azok mind az $y = x^2$ görbe pontjai. Alkalmazza a bemutatott eljárást ennek a feladatnak a megoldására, és magyarázza meg az eredményt!

6. Ha a transzformációs képletre több lehetőség is van, melyiket érdemes választani, az egyszerűt, vagy a bonyolultat? Érveljen a válasza mellett!

7. Vázolja, hogy az utolsó példában kifejtett eljárást hogyan lehetne formalizálni!

8. Írja meg a bevezetett eljárás egyes lépéseit valamely szimbolikus algebra rendszerben!

11. fejezet

Tesztelés

Ebben a fejezetben a dolgozatban több helyen is említett tesztelés részletkérdéseire, elveire térünk ki.

Először a standard globális optimalizálási tesztfeladatokat ismertetjük. Ezeket a szokásos nemlineáris optimalizálási tesztfeladatoktól az különbözteti meg, hogy a lehetséges megoldások halmazán rendszerint több helyi minimumpont van, és így egy helyi kereső eljárás nem feltétlenül tudja a globális minimum helyét meghatározni.

11.1. Standard globális optimalizálási tesztfeladatok

Az első kilenc feladat alkotja az ún. standard globális optimalizálási feladatok körét. Ezeknek a használata tipikus globális optimalizálási eljárások tesztelésére. Ezek tehát a rövidítéseket használva: S5, S7, S10, H3, H6, GP, RCOS, SHCB és RB. Itt az egyes feladatok leírását adjuk meg. Mindegyik probléma csak a változókra vonatkozó egyszerű korlátokkal rendelkezik.

Shekel feladatok

A célfüggvény 4-változós:

$$f(x) = -\sum_{i=1}^{m} \frac{1}{(x-a^i)^T(x-a^i)+c^i}$$

ahol a^i valós vektor, c^i valós szám ($i = 1, \ldots, m$), a lehetséges megoldások halmaza $0 \leq x_j \leq 10$ ($j = 1, 2, 3, 4$). A feladathoz tartozó adatok:

	i									
	1	2	3	4	5	6	7	8	9	10
a_1^i	4	1	8	6	3	2	5	8	6	7,0
a_2^i	4	1	8	6	7	9	5	1	2	3,6
a_3^i	4	1	8	6	3	2	3	8	6	7,0
a_4^i	4	1	8	6	7	9	3	1	2	3,6
c^i	0,1	0,2	0,2	0,4	0,4	0,6	0,3	0,7	0,5	0,5

Az $m = 5, 7, 10$ eseteket az S5, S7, S10 rövidítésekkel szokás jelölni. Ezek rendre 5, 7, illetve 10 helyi minimumponttal rendelkeznek, amelyek helyét közelítőleg az a^i vektorok adják meg, az

egyes helyi minimumok értéke pedig $-1/c^i$ körül van. A feladatok jellemzője, hogy (különösen nagy m esetén) a globális minimum vonzáskörzete kicsi a lehetséges megoldások halmazához képest, ezért az egyes módszerek gyakran nem találják meg a globális minimumot.

Hartman feladatok

A célfüggvény alakja:

$$f(x) = -\sum_{i=1}^{m} c_i \exp(-\sum_{j=1}^{n} a_{i,j}(x_j - p_{i,j})^2)$$

ahol $p_i, .$ közelítőleg az i. helyi minimumpont, és c_i közelítőleg a megfelelő helyi minimum. A lehetséges megoldások halmaza $0 \leq x_i \leq 1$, ahol $i = 1, 2, \ldots, n$. A feladathoz tartozó adatok $n = 3$ esetén:

i	$a_{i,j}$			c_i	$p_{i,j}$		
1	3,0	10,	30,	1,0	0,3689	0,1170	0,2673
2	0,1	10,	35,	1,2	0,4699	0,4387	0,7470
3	3,0	10,	30,	3,0	0,1091	0,8732	0,5547
4	0,1	10,	35,	3,2	0,03815	0,5743	0,8828

Az $n = 6$ esetre az adatok:

i	$a_{i,j}$						c_i
1	10,	3,0	17,	3,5	1,7	8,	1,0
2	0,05	10,	17,	0,1	8,0	14	1,2
3	3,0	3,5	1,7	10,	17,	8,	3,0
4	17,	8,0	0,05	10,	0,1	14	3,2

i	$p_{i,j}$					
1	0,1312	0,1696	0,5569	0,0124	0,8283	0,5886
2	0,2329	0,4135	0,8307	0,3736	0,1004	0,9991
3	0,2348	0,1451	0,3522	0,2883	0,3047	0,6650
4	0,4047	0,8828	0,8732	0,5743	0,1091	0,0381

A globális minimumpontok körül a célfüggvény rendkívül lapos, és ez nem kedvez az intervallum-aritmetikán alapuló módszereknek. A Hartman feladatok rövid jelölése H3, illetve H6.

Goldstein-Price feladat

A kétváltozós célfüggvény

$$\begin{aligned} f(x) &= [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2] \times \\ &\quad [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)], \end{aligned}$$

és a lehetséges megoldások halmaza $-2 \leq x_i \leq 2$ $(i = 1, 2)$. Ezen négy helyi minimumpont található. A célfüggvény bonyolultsága miatt a természetes intervallum-kiterjesztésen alapuló befoglaló függvény általában több nagyságrenddel szélesebb, mint a megfelelő értékkészlet. Jelölése GP.

Branin feladat

A kétváltozós célfüggvény alakja

$$f(x) = \left(x_2 - \frac{5,1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10,$$

a lehetséges megoldások halmaza $-5 \leq x_1 \leq 10$, $0 \leq x_2 \leq 15$. Ezen három globális minimumpont van. Jelölése RCOS.

Six-hump-camel-back feladat

A célfüggvény ismét kétváltozós:

$$f(x) = 4x_1^2 - 2,1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4,$$

a $-5 \leq x_i \leq 5$ lehetséges megoldási halmazzal. A függvény szimmetrikus az origóra, és három pár helyi minimumpontja van. Jelölése SHCB.

Rosenbrock feladat

A korábban már ismertetett

$$f(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$$

függvényt globális optimalizálási tesztfeladatként a $-1 \leq x_i \leq 5$ korlátokkal szokás használni. Nem nehéz globális optimalizálási feladat, inkább egyes egyszerűbb quasi-Newton helyi kereső algoritmusoknak nehéz. A szinthalmazok alakja miatt szokás banán-függvénynek is nevezni. Kicsit eltérő paraméterezéssel, és más dimenziókkal is lehet vele találkozni. Rövidítése RB.

Megoldások

A jegyzetben tárgyalt algoritmusokhoz kötődő adatokat (a globális minimum értéke és helye) összefoglaltuk a 3. Táblázatban[17]. Az utolsó, Rosenbrock feladat megoldása pontosan a 0 érték, amit a célfüggvény az $(1, 1)^T$ helyen vesz fel. A többi esetben a megadott jegyek az adott módszerrel elérhető decimális jegyek. Más programozási nyelvet, illetve eltérő számábrázolási pontosságot használva az itt megadottól eltérő értékeket lehet kapni. A Six-Hump-Camel-Back feladat kivételével egy-egy globális optimumpont van. Az SHCB esetén három azonos célfüggvény értékű optimális megoldás létezik.

[17] A technikai részleteket bővebben lásd a következő cikkben: Csendes, T.: Nonlinear parameter estimation by global optimization — efficiency and reliability. Acta Cybernetica 8(1988) 361-370

3.. táblázat. A tesztfüggvények globális minimumai és globális minimumpontjai (FORTRAN, egyszeres pontosságú valós számokkal).

Teszt-függvény	$f(x^*)$	x_1^*	x_2^*	x_3^*	x_4^*	x_5^*	x_6^*
S5	-10,153206	3,99995	4,00014	4,00011	4,00016		
S7	-10,402947	4,00061	4,00072	3,99945	3,99958		
S10	-10,536416	4,00075	4,00061	3,99967	3,99948		
H3	-3,8627815	0,1146	0,5557	0,8525			
H6	-3,3223667	0,201536	0,149909	0,476906	0,27524	0,31160	0,65735
GP	2,9996490	0,000068	-1,0001				
RCOS	0,39788723	-3,1416	12,275				
	0,39788723	3,1416	2,2750				
	0,39788723	9,425	2,4750				
SHCB	-1,0316286	0,0899	-0,7126				
	-1,0316286	-0,0899	0,7126				
RB	0,0	1,0	1,0				

11.2. Egyéb globális optimalizálási tesztfeladatok

EX2

Ez a tesztfeladat egy éles gyakorlati problémából származik[18], lényegében egy a komplex számok körében értelmezett nemlineáris regressziós feladat. A célfüggvény legkisebb négyzetek összege (az abszolút érték a complex számok miatt kell):

$$f(x) = \sum_{i=1}^{6} \left| f_i - \left(x_1 + \frac{x_2}{\omega_i^{x_3}} - \imath \left(\omega_i x_4 - \frac{x_5}{\omega_i^{x_3}} \right) \right) \right|^2 ,$$

ahol az f_i-k értéke rendre $5,0 - 5,0\imath$, $3,0 - 2,0\imath$, $2,0 - \imath$, $1,5 - 0,5\imath$, $1,2 - 0,2\imath$ és $1,1 - 0,1\imath$, továbbá $\omega_i = \pi i/20$, ahol $i = 1, 2, ..., 6$. A keresési tartomány $[0,1]^2 \times [1,1,1,3] \times [0,1]^2$.

A feladat szokásos, a célfüggvény kiértékelésén alapuló módszerekkel nem nehéz, de kiemelkedően nagy műveletigényű a megbízható megoldása intervallumos eljárásokkal. Vigyázat, a feladatot is tárgyaló egyik közleményben[19] az utolsó negatív jel helyett tévesen pozitív szerepel.

EX3

Ez egy mesterségesen létrehozott tesztfeladat, amely a sztochasztikus globális optimalizálási algoritmusok megbízhatóságának mérésére alkalmas. A célfüggvény:

$$f(x) = \sum_{i=1}^{4} x_i^6 (\sin(1/x_i) + 2)$$

[18] Z. Hantos, B. Suki, T. Csendes, B. Daróczy, and S. Nagy: Modeling of low-frequency pulmonary impedance in dogs., J. Applied Physiology 68(1990), pp. 849–860

[19] Csendes, T. and D. Ratz: Subdivision direction selection in interval methods for global optimization, SIAM J. Num. Anal. 34(1997) 922-938

ha $\Pi_{i=1}^{4} x_i \neq 0$, és $f(x) = 0$ különben. A keresési tartomány $[-2, 2]^4$.
Igazoljuk a következő állításokat:

- $f(x)$ folytonos, és folytonosan deriválható.
- $f(x)$-nek megszámlálhatóan végtelen sok helyi minimuma és maximuma van (a korlátozás nélküli esetben).
- $x^6 \leq f(x) \leq 3x^6$ érvényes minden x-re.
- $f(x)$ globális minimuma nulla, és ezt az értéket csak a nulla pontban veszi fel.
- A globális minimum vonzáskörzete nullamértékű.
- A globális minimumpont nem szeparált.

Irodalomjegyzék

[1] Bazaraa, M.S., H.D. Sherali, and C.M. Shetty: Nonlinear Programming — Theory and Algorithms, Wiley, 1993.

[2] Bomze, I.M., T. Csendes, R. Horst, and P.M. Pardalos (eds.): Developments in Global Optimization. Kluwer, Dordrecht, 1997.

[3] Csallner András Erik: Intervallum-felosztási eljárások a globális optimalizálásban. PhD disszertáció, Szeged, 1999. Elérhető a `http://www.jgytf.u-szeged.hu/∼csallner/` internetes címen.

[4] Dixon, L.C.W., G.P. Szegő (eds.): Towards Global Optimisation. North-Holland, Amsterdam, 1974.

[5] Dixon, L.C.W., G.P. Szegő (eds.): Towards Global Optimisation 2. North-Holland, Amsterdam, 1978.

[6] Forgó Ferenc: Nemkonvex és diszkrét programozás: korszerű ismeretek gazdasági szakemberek számára, Közgazdasági és Jogi Könyvkiadó, Budapest, 1978.

[7] Gazdag-Tóth Boglárka: Globális Optimalizálás. BME Tankönyvtár, Budapest, 2011. `http://tankonyvtar.ttk.bme.hu/pdf/3.pdf`

[8] Gerencsér László: Nemlineáris programozási feladatok megoldása szekvenciális módszerekkel, SZTAKI Tanulmányok 49/1976, Budapest.

[9] Gill, P. E., W. Murray, M.H. Wright: Practical Optimization. Academic Press, London, 1981.

[10] Hansen, E.: Global Optimization Using Interval Analysis. Marcel Dekker, New York, 1992.

[11] Horst, R., P.M. Pardalos, and N.V. Thoai: Introduction to Global Optimization, Kluwer, Dordrecht, 1995.

[12] Komlósi Sándor: Az optimalizáláselmélet alapjai. Dialóg Campus Kiadó, Budapest, Pécs, 2001.

[13] Kósa András: Optimumszámítási modellek, Műszaki Könyvkiadó, Budapest, 1979.

[14] Krekó Béla: Optimumszámítás. Nemlineáris programozás, Közgazdasági és Jogi Könyvkiadó, Budapest, 1972.

[15] Martos Béla: Nemlineáris Optimalizálás, Akadémiai Könyvkiadó, Budapest, 1974.

[16] Neumaier, Arnold globális optimalizálási portálja számos további linkkel, forráskódokkal, tesztfeladatokkal: `http://www.solon.math.univie.ac.at/globopt`

[17] Pintér, J.D.: Global Optimization in Action, Kluwer, Dordrecht, 1996.

[18] Ratschek, H., J. Rokne: Computer Methods for the Range of Functions. Ellis Horwood, Chichester, 1984.

[19] Ratschek, H., J. Rokne: New Computer Methods for Global Optimization. Ellis Horwood, Chichester, 1988.

[20] Törn, A., A. Žilinskas: Global Optimization. (Lecture Notes in Computer Science No. 350, G. Goos and J. Hartmanis, Eds.) Springer, Berlin, 1989.

Az irodalomjegyzék persze nem teljes, csak néhány fontosabb, illetve a tananyaghoz közvetlenül kapcsolódó könyvet, jegyzetet adunk meg. A szakterület első komolyabb műveinek a bergamói Workhshop on Global Optimization két kiadványa számít [4, 5].

Magyar nyelven sokáig lényegében nem volt irodalma a globális optimalizálásnak. Új jegyzet Gazdag-Tóth Boglárkáé. A kevés hazánkban megjelent nemlineáris optimalizálási könyvet épp ezért szerepeltetjük itt.

Az itt megadott a könyvek, jegyzetek kiegészítésül ajánlhatók jegyzetünkhöz, de a félév anyagához nem feltétlenül szükségesek. Hasznosak lehetnek viszont diákköri dolgozathoz vagy diplomamunkához. A listában szereplő könyvek az SZTE Informatikai Intézete könyvtárában, és a nagyobb könyvtárakban megtalálhatók.

Tárgymutató

Magyar-angol szószedet

Itt a leggyakoribb szakkifejezéseket gyűjtöttem össze azok angol nyelvű változatával. Ez remélhetőleg segít majd az inkább az angol kifejezéseket ismerőknek, és fordítva, megkönnyíti majd az angol szakszöveg olvasását azoknak, akik nem ismerik az angol szakirodalmat.

a változókra vonatkozó korlátokkal rendelkező feladat	bound constrained problem
befoglalási izotonitás	inclusion isotonity
direkt kereső	direct search methods
egészértékű optimalizálási feladat	integer optimization problem
elsőrendű módszer	first order method
feltétel nélküli optimalizálási feladat	unconstrained optimization problem
globális minimum	global minimum (többes szám: global minima)
globális minimumpont	global minimizer
gradiens módszer	gradient method
helyi minimum	local minimum (többes szám: local minima)
helyi minimumpont	local minimizer
intervallum felosztási módszer	interval subdivision method
konkáv függvény	concave function
konvex függvény	convex function
korlátozott feladat	bounded problem
kvadratikus függvény	quadratic function
legmeredekebb lejtő módszere	steepest descent method
Lipschitz-folytonos függvény	Lipschitzian function
maximum	maximum (többes szám: maxima)
másodrendű módszerek	second order methods
minimum	minimum (többes szám: minima)
művelet kiterjesztése	operation overloading

nemdifferenciálható függvény	non-differentiable function
négyzetösszeg típusú függvény	sum-of-squares type objective function
nyeregpont	saddlepoint
optimalizálási feltétel	constraint
regresszió	regresion
regressziós egyenlet	regression equation
sima függvény	smooth function
stacionárius pont	stationary point, critical point
szeparált célfügggvény	separable objective function
szeparált helyi minimumpont	separable local minimizer
szigorú globális minimumpont	strict global minimizer
szigorú helyi minimumpont	strict local minimizer
véletlen keresés	random search
vonzáskörzet	region of attraction

A Függelék

Tematika

A tárgy bevezetést ad a globális optimalizálási módszerek használatába, ezek elméleti alapjait és numerikus megvalósításuk problémáit tárgyaljuk. Az érdeklődők betekintést nyerhetnek néhány optimalizálási szoftver használatába is. A tantárgy a programtervező matematikus képzés operációkutatás és alkalmazott matematika szakirányába tartozik.

A tárgyat speciálkollégiumként programozó, programtervező matematikus és közgazdász programozó hallgatóknak ajánlom (a második évfolyamtól). Az előfeltétele ekkor az első analízis és numerikus kurzusok felvétele.

A tematika kulcsszavakban:

- a globális optimalizálási feladat különböző alakjai, műveletigényének viszonya a lineáris programozáséhoz,
- az egyes globális optimalizálási feladatok egymásba való átalakítása, redukálása egydimenziós feladatra
- a globális optimalizálási módszerek osztályai, a felhasznált információ típusa szerinti csoportosítás
- sztochasztikus és multi-start eljárások globális optimalizálásra, ezek konvergenciája és megállási feltételei
- a Lipschitz-konstans ismeretére támaszkodó módszerek, konvergenciatételek, egy- és többdimenziós eljárások
- intervallum-aritmetika: a 4 alapművelet, a négyzetreemelés, a gyökvonás, a standard függvények kiterjesztése intervallum-argumentumra
- a bit-billentés szerepe számítógépeken
- a naiv-, vagy természetes intervallum-kiterjesztés becslésének minősége, lineáris konvergencia
- a központi alak (centered form), és más befoglaló függvények, négyzetes konvergencia
- az automatikus deriválás és szerepe a befoglaló függvények javításában
- a Moore-Skelboe intervallum-felezési algoritmus, és alkalmazása globális optimalizálásra és érzékenység-vizsgálatra

- konvergencia-sebesség,
- gyorsító vizsgálatok intervallumos korlátozás és szétválasztás típusú algoritmusokban,
- intervallumos Newton algoritmus, patologikus feladatok.
- intervallumos módszer a szinthalmaz jellemzésére
- néhány intervallum-aritmetikát támogató programozási nyelv: PASCAL-SC, PASCAL-XSC, C-XSC, FORTRAN-XSC, ACRITH, ARITHMOS, ...

Az anyagot tartalmazó jegyzet előzetesen elérhető a következő internetes címen:

`http://www.inf.u-szeged.hu/~csendes/go.ps.gz`

Ezen túl angol nyelvű bevezető könyveket tudok ajánlani. Ezek ismeretére nincs feltétlenül szükség az elhangzottak megértéséhez.

+ R. Horst and P.M. Pardalos (eds.): Handbook of Global Optimization, Kluwer, 1995.

+ R. Horst, P.M. Pardalos, N.V. Thoai: Introduction to Global Optimization, Kluwer, 1995.

+ R.B. Kearfott: Rigorous Global Search: Continuous Problems, Kluwer, 1996.

Az angol nyelvű globális optimalizálási portál internetes címe:

`http://www.solon.math.univie.ac.at/globopt`

B Függelék

Tételek

A Globális optimalizálás című tárgy kollokviumának tételei:

1. Nemlineáris optimalizálási alapfogalmak
2. A globális optimalizálási módszerek osztályai
3. Direkt kereső eljárások
4. Evolúciós, genetikus és szimulált megeresztési módszerek
5. DC-programozás
6. Eljárások Lipschitz-folytonos célfüggvényekre
7. Intervallumos korlátozás és szétválasztás algoritmus
8. Intervallumos gyorsító lépések
9. Automatikus differenciálás

C Függelék

Dolgozat feladatok

A következő oldalakon néhány előkészített dolgozat feladatsor található. Ezek orientálhatják a felkészülést, illetve mintául szolgálhatnak más tudás-felméréshez is. A dolgozatok célja inkább annak tisztázása, hogy a hallgatók mit tudtak elsajt́ítani a hallottakból, mintsem azt megtalálni, hogy mit nem tudnak. Másrészt a formális definíció – tétel – bizonyítás számonkérése helyett a hangsúly inkább a konkrét alkalmazáson van.

A számolási feladatok a gyakorlat, az elméleti feladatok az előadás jegyét hivatottak meghatározni. A dolgozat eredménye alapján (az óralátogatás, az órai aktivitás, a leadott kötelező programok minősége és más külön teljesítmények figyelembevételével) a hallgatók megajánlott jegyeket kaphatnak. Ezen szóbeli beszámolóval, vizsgával, vagy teszttel lehet javítani.

Globális Optimalizálás záródolgozat[20] 2002. IV. 29.

Számolási feladatok:

1. Határozza meg az $f(x) = x\sin(2\pi x) + 2$ függvény befoglaló függvényét természetes intervallum kiterjesztéssel az $X = [0, 1]$ intervallumban! 2 pont

2. Számolja ki automatikus differenciálással az $f(x) = (x-2)(x-3)$ függvény deriváltját az 1 pontban! 2 pont

3. Határozza meg az órán tanult szimbolikus módszerrel hogy milyen egyszerűsítő helyettesítést lehet alkalmazni az $f(x_1, x_2) = e^{(x_1+x_2)^2}$ függvényen, és értelmezze az eredményt!2 pont

4. Fogalmazza meg büntetőfüggvény segítségével azt a korlátozás nélküli feladatot, amelynek megoldása megfelel a
$$\min x^2 + y^2,$$
$$(x-2)^2 + (y-3)^2 - 2 \leq 0$$
feltételes optimalizálási feladatának! 2 pont

5. Oldja meg a Lagrange módszer segítségével a következő feltételes optimalizálási feladatot: $\min\ (x_1-1)^2 + (x_2-1)^2,\ \ x_2 - 2 = 0.$ 2 pont

Elméleti feladatok:

1. Adjon meg egy olyan optimalizálási feladatot, amelynek van olyan helyi minimumpontja, amely nem globális minimumpont, de helyi maximumpont! 2 pont

2. Mutasson olyan optimalizálási feladatot, amelyre a keresési tartomány egy harmada az egyik globális minimumpont vonzáskörzete! 2 pont

3. Igaz-e, hogy van olyan determinisztikus optimalizálási eljárás, amely csak a célfüggvény értékét kiszámító szubrutinra támaszkodik mint információforrásra, és véges számú lépésben megadja minden nemlineáris optimalizálási feladat megoldását? Indokolja! 2 pont

4. Melyik tanult globális optimalizáló eljárást használná egy olyan minimalizálási feladat megoldására, amely csak egyszerű korlátokat tartalmaz az optimalizálandó változókra, rendelkezésre áll a célfüggvény és gradiense értékét adott pontban megadó szubrutin, a célfüggvény kétszer folytonosan differenciálható, és az egyetlen globális minimumpont vonzáskörzete mértéke kb. 1%-a a keresési tartományának? 2 pont

5. Ismertesse az egyik tanult globális optimalizálási eljárást, adja meg azt a feladatosztályt, amelynek megoldására ez különösen alkalmas, említse meg az előnyeit és hátrányait! 6 pont

[20] A kettes az elérhető pontok 50%-aval, a jeles 80%-al érhető el. A rendelkezésre álló idő 120 perc.

Globális Optimalizálás záródolgozat

2002. V. 6.

Számolási feladatok:

1. Határozza meg az $f(x) = x\ln(x) + 2$ függvény befoglaló függvényét természetes intervallum kiterjesztéssel az $X = [1, 2]$ intervallumban! 2 pont

2. Számolja ki automatikus differenciálással az $f(x) = (x-1)(x-2)$ függvény deriváltját az 3 pontban! 2 pont

3. Határozza meg az órán tanult szimbolikus módszerrel hogy milyen egyszerűsítő helyettesítést lehet alkalmazni az $f(x_1, x_2) = 10^{(2x_1+3x_2)^2}$ függvényen, és értelmezze az eredményt!

   2 pont

4. Fogalmazza meg büntetőfüggvény segítségével azt a korlátozás nélküli feladatot, amelynek megoldása megfelel a

$$\min 2x^2 + 3y^2,$$
$$(x-1)^2 + (y-1)^2 - 1 \leq 0$$

   feltételes optimalizálási feladatának (legyen tekintettel a nem lehetséges megoldásból induló helyi keresésre)! 2 pont

5. Oldja meg a Lagrange módszer segítségével a következő feltételes optimalizálási feladatot: $\min\ (x_1-2)^2 + (x_2-3)^2,\ \ x_1 + x_2 - 1 = 0.$ 2 pont

Elméleti feladatok:

1. Adjon meg egy olyan optimalizálási feladatot, amelynek van olyan nem szeparált helyi minimumpontja, amely nem globális minimumpont, de mégis helyi maximumpont! 2 pont

2. Mutasson olyan kétdimenziós optimalizálási feladatot, amelyre a keresési tartomány egy negyede az egyik globális minimumpont vonzáskörzete! 2 pont

3. Igaz-e, hogy van olyan sztochasztikus optimalizálási eljárás, amely csak a célfüggvény értékét kiszámító szubrutinra támaszkodik mint információforrásra, és véges számú lépésben megadja minden nemlineáris optimalizálási feladat megoldását? Indokolja! 2 pont

4. Melyik tanult globális optimalizáló eljárást használná egy olyan minimalizálási feladat megoldására, amely csak egyszerű korlátokat tartalmaz az optimalizálandó változókra, rendelkezésre áll a célfüggvény és gradiense befoglaló függvényét kiszámító szubrutin, a célfüggvény kétszer folytonosan differenciálható? 2 pont

5. Ismertesse az egyik tanult globális optimalizálási eljárást, adja meg azt a feladatosztályt, amelynek megoldására ez különösen alkalmas, valamint említse meg az előnyeit és hátrányait! 6 pont

D Függelék

Kötelező programok

A kurzus gyakorlatának elfogadásához egy program önálló megírása is hozzátartozik. Ezeket az alábbi listából lehet kiválasztani. A *-al jelöltek nehezebbeknek számítanak, ezért a gyakorlati jegy meghatározása során sikeres leadásuk esetén többletpont jár.

A programok tetszőleges programozási nyelven készülhetnek, de a C, a FORTRAN és a JAVA preferált, már mint hogy ügyesebb implementálást jelent — de az elfogadást nem befolyásolja. A C az elterjedtsége miatt javasolt, a FORTRAN a leggyakrabban használt az optimalizálásban, és jól is illik a feladatkör igényeihez, a JAVA-program pedig a hordozhatósága (?) miatt jó lehet az algoritmusok bemutatására.

A programozási feladatok leadása úgy történik, hogy a futtatható változat és a forráskód is kell ehhez: ezek ismeretében a felmerülő kérdésekre válaszolni kell tudni.

A választható programozási feladatok:

1. *Kalkulátor.* Írjon egy eljárást programozható kalkulátorra egyszerű globális optimalizálási feladatok megoldására, úgy hogy szerény méretű feladatokra reális időben lehessen közelítő megoldást kapni. Tesztelje az algoritmust polinomokkal.

2. *Rácsmenti keresés.* A feladat egy olyan algoritmust kódolni, amely valamely célfüggvény értékeit határozza meg egy n-dimenziós intervallumban úgy, hogy a rács finomsága a program input paramétere.

3. *Véletlen keresés - Monte Carlo módszer.* A feladat olyan programot írni, amely egy célfüggvény minimumának megkeresésére vállalkozik úgy, hogy az inputban meghatározott többdimenziós intervallumba a felhasználó által megadott számú véletlen pontot generál egyenletes eloszlással, ezeken kiértékeli a célfüggvényt, majd visszaadja ezek közül a legjobbat.

4. *Excel Solver.* Oldjon meg egy kifejező globális optimalizálási feladatot az Excel Solver nevű eszközével (ezt néha külön implementálni kell a bővítménykezelővel). Dokumentálja az indítópont szerepét, a megbízhatóságot.

5. *Maple.* A Maple (vagy a Mathematica, esetleg a Derive, vagy a Mupad) programra építve adjon meg egy olyan optimalizálási algoritmust, amely azon alapul, hogy a korlátozás nélküli optimalizálási feladat célfüggvényének gradiense zérushelyeit határozza meg.

6. *SPSS.* Egy statisztikai programcsomag (pl. a felsőoktatásban ingyenes SPSS) nemlineáris regressziós eljárását használja egy globális optimalizálási feladat megoldására: mutasson példát, amikor több, lényegesen eltérő helyi minimum létezik.

7. *Rétegelt mintáztatás.* Implementálja a rétegelt mintáztatás módszerét (dobjon N db. pontot egyenletes eloszlással egy n-dimenziós intervallumba, tartsa meg a célfüggvény szerinti legjobb 10%-ot, rajzoljon ezek köré intervallumot stb.), és oldjon meg vele egy globális optimalizálási feladatot.

8. *Nemlineáris szimplex.* Implementáljuk a nemlineáris szimplex módszert (az n-dimenziós térben $n+1$ pontból álló szimplex minden csúcsában határozzuk meg a célfüggvény értékét, majd billentsük úgy, hogy a legrosszabbat változtatjuk meg), és teszteljük egy nem triviális feladaton.

9. *Evolúciós módszer.* Valósítsa meg az evolúciós módszer egy változatát (válasszon N darab pontot egyenletes eloszlással egy n-dimenziós intervallumban, tartsa meg ezek legjobb 50%-át, minden ilyen pontból képezzen egy utódot úgy, hogy a szülőből mint várható értékből kiindulva normális eloszlással kicsit eltérő pontot generál, vegye a teljes populáció legjobb 50%-át stb.), és oldjon meg vele egy jellemző globális optimalizálási feladatot.

10. *Egy-dimenziós Lipschitz-optimalizálás.* Implementálja az egy-dimenziós, az L Lipschitz konstans ismeretén alapuló módszert (képezze a végpontokból az L meredekséggel rajzolt alulról korlátozó lineáris törtfüggvényt, és ezt javítsa fokozatosan annak minimum-pontjában való függvénykiértékelés, és a törtvonal megfelelő javítása segítségével), és mutassa be a működését egy áttekinthető feladaton.

11. *Szimulált hőkezelés.* Implementálja a simulated annealing módszer egy egyszerű változatát (dobjon egyenletes eloszlással N véletlen pontot az n-dimenziós keresési intervallumban, majd minden pontból lépjen egy nem rosszabb célfüggvényértékű pontba csökkenő szórású normális eloszlás alapján stb.), és tesztelje egy egyszerű feladaton.

12. *Multistart.* Valósítsa meg a többszörös indítás módszerét (válasszon ki egyenletes eloszlással néhány ígéretes indulópontot, és hajtson végre helyi keresést ezekből startolva...) egy egyszerű formában, és illusztrálja a működését egy alkalmas feladaton.

13. *GLOBAL.* Töltse le a GLOBAL programot a következő internetes címről és oldjon meg vele egy tetszés szerinti globális optimalizálási feladatot.

    `ftp://ftp.jate.u-szeged.hu/pub/math/optimization/index.html`

14. *Intervallumos B&B.* Az alapműveletek intervallumos kiterjesztését írja meg szubrutinként, majd erre alapozva adjon meg egy egyszerű korlátozás és szétválasztás típusú optimalizáló eljárást. Tesztelje egy kiismerhető polinomon.

15. *Körpakolás négyzetben***.* A feladat az egységnégyzetben meghatározni adott n-re azt az n darab pontot, amelyek közötti minimális távolság a lehető legnagyobb. A feladathoz tetszőleges, akár a hálózatról származó algoritmus is használható.

16. *Fekete-pontok meghatározása***.* A feladat egy gömb felületén adott számú pont meghatározása úgy, hogy az azok távolságának reciprok összege minimális legyen. A megoldásra tetszőleges módszer bevethető.

17. *A legrövidebb***.* Adjon meg egy olyan, minimális hosszúságú globális optimalizálási algoritmust, amely a Shekel-10 feladatot 1 másodpercen belül képes megoldani legalább 2 jegy pontossággal.

E Függelék

Az UNIRANDI nevű direkt helyi kereső rutin

```
C   ROUTINE NAME        - LOCAL
C
C-----------------------------------------------------------------------
C
C   COMPUTER             - IBM PC/DOUBLEPRECISION
C
C   LATEST REVISION      - OCTOBER 15, 1986
C                            (ALTERATIONS IN THE COMMENTS AND IN THE
C                            ROUTINE PARAMETERS TO MEET THE REQUIREMENTS
C                            OF THE LATEST (JULY 31, 1986) VERSION OF
C                            THE GLOBAL ROUTINE.)
C
C   PURPOSE              - MINIMUM OF FUNCTION OF N VARIABLES USING
C                            A RANDOM WALK METHOD UNIRANDI WITH A GIVEN
C                            INITIAL STEP LENGTH.
C
C   USAGE                - CALL LOCAL (M,N,RELCON,MAXFN,X,F,NFEV,R,
C                            MIN,MAX)
C
C   ARGUMENTS    M       - NUMBER OF RESIDUALS OR OBSERVATIONS.(INPUT)
C                N       - THE NUMBER OF UNKNOWN PARAMETERS (I.E.,
C                            THE LENGTH OF X) (INPUT)
C                RELCON  - CONVERGENCE CRITERION. (INPUT) THIS IS
C                            SATISFIED IF, ON TWO SUCCESSIVE ITERATIONS,
C                            THE PARAMETER ESTIMATES (I.E., X(I),
C                            I=1,...,N) OR THE FUNCTION VALUE DIFFERS,
C                            COMPONENT BY COMPONENT, BY AT MOST RELCON.
C                MAXFN   - MAXIMUM NUMBER OF FUNCTION EVALUATIONS (I.E.,
C                            CALLS TO SUBROUTINE FUNCT) ALLOWED. (INPUT)
C                            THE ACTUAL NUMBER OF CALLS TO FUNCT MAY
C                            EXCEED MAXFN SLIGHTLY.
C                X       - VECTOR OF LENGTH N CONTAINING PARAMETER
C                            VALUES.
C                          ON INPUT, X SHOULD CONTAIN THE INITIAL
C                            ESTIMATE OF THE LOCATION OF THE MINIMUM.
```

```
C                         ON OUTPUT, X CONTAINS THE FINAL ESTIMATE
C                           OF THE LOCATION OF THE MINIMUM.
C               F       - SCALAR CONTAINING THE VALUE OF THE OBJECTIVE
C                           FUNCTION AT THE FINAL PARAMETER ESTIMATES.
C                         ON INPUT, F SHOULD CONTAIN THE FUNCTION VALUE
C                           AT THE INITIAL PARAMETER ESTIMATES.
C                         ON OUTPUT, F CONTAINS THE FUNCTION VALUE
C                           AT THE FINAL PARAMETER ESTIMATES.
C               NFEV    - THE NUMBER OF FUNCTION EVALUATIONS (OUTPUT)
C               R       - A VECTOR OF LENGTH 3*N USED AS WORKING SPACE.
C               MIN     - A VECTOR OF LENGTH N CONTAINING SCALING
C                           FACTORS SUPPLIED BY THE GLOBAL ROUTINE
C                           (INPUT)
C               MAX     - A VECTOR OF LENGTH N CONTAINING SCALING
C                           FACTORS SUPPLIED BY THE GLOBAL ROUTINE
C                           (INPUT)
C
C   REFERENCE           - TIMO JARVI: A RANDOM SEARCH OPTIMIZER WITH
C                           AN APPLICATION TO A MAX-MIN PROBLEM. A
C                           SPECIAL TRAJECTORY ESTIMATION PROBLEM,
C                           PUBLICATIONS OF THE INSTITUTE FOR APPLIED
C                           MATHEMATICS, NO. 3, UNIVERSITY OF TURKU,
C                           1973.
C
C   PRECISION/HARDWARE  - SINGLE AND DOUBLE/H32
C                       - SINGLE/H36,H48,H60
C
C   REQUIRED ROUTINES   - URDMN,FUN
C
C-----------------------------------------------------------------------
C
      SUBROUTINE LOCAL (M,N,RELCON,MAXFN,X,F,NFEV,R,MIN,MAX)
      IMPLICIT DOUBLEPRECISION (A - H,O - Z)
      DOUBLEPRECISION R(100,15),X(15),X1(15),MIN(1),MAX(1)
      DATA                ZERO,ONEN3,HALF,ONE,TWO/
     *                    0.0,0.001,0.5,1.0,2.0/
C                                 FIRST EXECUTABLE STATEMENT
C                                 INITIAL STEP LENGTH
      H = ONEN3
      DELTF = ONE
      ITEST = 0
      NFEV = 0
      EPS = RELCON
C                                 EVALUATE 100 RANDOM VECTORS
    5 CALL URDMN (R,1500)
      IRNDM = 0
   15 IRNDM = IRNDM+1
```

```
      IF (IRNDM.GT.100) GO TO 5
C                                     SELECT A RANDOM VECTOR HAVING NORM
C                                       LESS OR EQUAL TO 0.5
      A = ZERO
      DO 20 I=1,N
      R(IRNDM,I) = R(IRNDM,I)-HALF
   20 A = A+R(IRNDM,I)*R(IRNDM,I)
      IF (A.LE.ZERO) GO TO 15
      A = SQRT(A)
      IF (A.GT.HALF) GO TO 15
C                                     NEW TRIAL POINT
      DO 25 I=1,N
      R(IRNDM,I) = R(IRNDM,I)/A
   25 X1(I) = X(I)+H*R(IRNDM,I)
      CALL FUN (X1,F1,N,M,MIN,MAX)
      NFEV = NFEV+1
      IF (F1.LT.F) GO TO 35
      IF (NFEV.GT.MAXFN) GO TO 50
C                                     STEP IN THE OPPOSITE DIRECTION
      H = -H
      DO 30 I=1,N
   30 X1(I) = X(I)+H*R(IRNDM,I)
      CALL FUN (X1,F1,N,M,MIN,MAX)
      NFEV = NFEV+1
      IF (F1.LT.F) GO TO 35
      IF (NFEV.GT.MAXFN) GO TO 50
      ITEST = ITEST+1
!      IF (ITEST.LT.2) GO TO 15
      IF (ITEST.LT.4) GO TO 15
C                                     DECREASE STEP LENGTH
      H = H*HALF
      ITEST = 0
C                                     RELATIVE CONVERGENCE TEST FOR THE
C                                       OBJECTIVE FUNCTION
      IF (DELTF.LT.EPS) GO TO 50
C                                     CONVERGENCE TEST FOR THE STEP LENGTH
      IF (ABS(H)-RELCON) 50,15,15
   35 DO 40 I=1,N
   40 X(I) = X1(I)
      DELTF = (F-F1)/ABS(F1)
      F = F1
C                                     INCREASE STEP LENGTH
      H = H*TWO
      DO 45 I=1,N
   45 X1(I) = X(I)+H*R(IRNDM,I)
      CALL FUN (X1,F1,N,M,MIN,MAX)
      NFEV = NFEV+1
```

```
      IF (F1.LT.F) GO TO 35
C                                     CHECK TOLERANCE MAXFN
      IF (NFEV.GT.MAXFN) GO TO 50
C                                     DECREASE STEP LENGTH
      H = ABS(H*HALF)
      GO TO 15
   50 RETURN
      END
```

F Függelék

Szorgalmi feladatok

1. Először írjunk programot N valós szám összeadására, majd igazoljuk, hogy már három szám esetén is az illető program által (véges számábrázolás mellett) megadott eredmény és a valódi összeg eltérése lehet pl. 2003-nál nagyobb. Mit lehet tenni?

2. Adott egy körpálya véges sok benzinkúttal, amelyekben pontosan annyi benzin van, ameny-nyivel körbe lehet autózni a pályát. Feltesszük, hogy a benzintartály mérete elegendően nagy ahhoz, hogy akár a teljes benzinmennyiséget befogadja. Igazoljuk, hogy akkor létezik olyan pont (nyilván egy benzinkút, ahol tankolással kezdünk), ahonnan indulva körbe lehet menni anélkül, hogy kifogyna a benzin.

3. Le lehet-e rajzolni (pl.) egy asztalterítőre kontinuum sok (általában különböző méretű) nyolcast anélkül, hogy azok vonala metszené egymást? Nehezebb a feladat három szakaszból álló szimmetrikus csillagokkal, de ugyanaz az eredmény.

4. Adott egy épülőfélben lévő ház, a padláson 3 izzó, a pincében ezekhez három kapcsoló. A pincéből nem lehet látni, hogy világítanak-e a lámpák. Tetszőleges kapcsolgatás után (azt tudhatjuk, hogy melyik a bekapcsolt állapot) a padláson meg kell mondani, hogy melyik kapcsoló melyik körtéhez tartozik. A pincébe tehát nem lehet már visszamenni. Vigyázat, a feladat megoldható, és minél elméletibb megoldást keresünk, annál reménytelenebb!

5. Tekintsük az $1 = \sqrt{1} = \sqrt{(-1)(-1)} = \sqrt{-1}\sqrt{-1} = -1$ levezetést! Melyik egyenlőség a hibás? Egy gúnyos megjegyzés szerint mindegyik helyes, csak az egyenlőség nem tranzitív. Bármilyen hihetetlen, ez nem is áll messze az igazságtól. Érdekességként a Derive nevű szimbolikus matematikai program helyből a harmadik egyenletet véli hamisnak — ami egybevág a hallgatók leggyakoribb tippjével (ez az egyenlőség abszolút rendben van).

6. Vegyünk egy narancsot, tegyük fel, hogy a héj vastagságának aránya állandó a narancs sugarához képest. Igazoljuk azt a meglepő állítást, hogy a narancs dimenziójának növelésével ($3, 4, \ldots$) a héj nélküli, ehető rész térfogatának aránya nullához tart!

7. Tekintsünk egy kört, amelybe bele van rajzolva 6 maximális sugarú egybevágó, egymást nem átfedő kisebb kör. Igazoljuk, hogy a nagyba ekkor még egy, az előzőekkel megegyező méretű kis kör is belefér! Még megoldatlan feladat négyzetbe 31 maximális sugarú egybevágó kör rajzolása.

8. (Erdős Pál:) Legyen adott $2 < N < \infty$ pont a síkban úgy, hogy nem mind esik egy egyenesre. Igazoljuk, hogy akkor van olyan két pont az N-ből, amelyek által meghatározott egyenesen nincs más pont az eredeti halmazból.

9. Húzzuk be egy tetszőleges háromszög szögharmadoló félegyeneseit. Igazoljuk, hogy ezek metszéspontjai egyenlő oldalú háromszöget alkotnak!

10. Azt kíséreljük meg igazolni, hogy $90^o = 91^o$. Vegyünk egy AB szakaszt, mérjünk fel az A pontban 91^o-ot, a B pontban pedig 90-et (úgy, hogy a félegyenesek az AB szakasz azonos oldalára essenek). Mérjük fel egy ugyanazt a távolságot a félegyenesekre: AD = BC. Legyen az AB felezőpontja X, a CD felezőpontja Y. Húzzuk meg ezekben a szakaszfelező merőlegeseket. Legyen ezek metszéspontja S. (Azt egyszerűen be lehet látni, hogy a felezőmerőlegesek egyetlen pontban metszik egymást.) Ekkor AS = BS, és DS = CS, tehát az ADS háromszög egybevágó a BCS háromszöggel. Ebből adódik, hogy az SAD szög megegyezik az SBC szöggel, amiből kivonva az SAB = SBA szöget kapjuk, hogy $90^o = 91^o$ ...

11. Szorzás orosz módra (vagy orosz parasztszorzás, 'multiplication a la Russe'): tegyük fel, hogy csak kettővel tudunk szorozni és osztani. Pl. a 27 x 13 helyett így 54 x 6 -ot írunk, és megjegyezzük, hogy a 27-el volt valami bajunk, hiszen a 13 nem osztható kettővel. A következő lépés: 54 x 6 helyett 108 x 3, majd 216 x 1, és a 108-al volt bajunk. Innen már csak az eredményt kell leolvasni: 216 + 108 + 27 = 351. Indokoljuk, hogy hogyan jöhetett ki az eredmény, és miért ez a jelenlegi leggyorsabb szorzási eljárás számítógépeken!

12. Egy televíziós vetélkedőben a játék egy fázisában a nyertes választhat három ajtó közül, amelyek mögött egy-egy majom, illetve egy Mercedes autó van. A játékos nyilván az autót szeretné. Mikor a kezét az egyik kilincsre teszi, a játékvezető (minden esetben) felkiált, hogy ne azt válassza. Annak igazolásául, hogy tudja, hogy melyik ajtó mögött mi van, kinyitja az egyik másik ajtót, ami mögött egy majom van (ezt nyilván mindig meg lehet tenni).

    Indokolt-e változtatni az első választáson, és milyen valószínűséggel van autó az egyes ajtók mögött, ha a játék fair, az ajándékok elosztása egyenletes eloszlással történik, és a játék alatt nem változik a helyük?

13. Igazoljuk, hogy minden zárt síkgörbének van olyan négy pontja, amelyek egy négyzetet határoznak meg! (nyitott probléma)

14. Igazoljuk, hogy van olyan, az x és y tengelyekkel párhuzamos oldalú, racionális oldalhosszú négyzet, amelyben van olyan pont, amelynek minden csúcstól vett távolsága egész szám! (nyitott)

15. Tekintsünk egy a_0, b_0 oldalú téglalapot. Osszuk fel ezt véges sok téglalapra, és legyenek a kis téglalapok oldalai rendre a_i és b_i, $i = 1, 2, \ldots, n$. Igazoljuk, hogy érvényes a

$$|\sin(a_0)|\,|\sin(b_0)| \leq \sum_{i=1}^{n} |\sin(a_i)|\,|\sin(b_i)|$$

    egyenlőtlenség! Nyitott a megfordítása: a megadott egyenlőtlenségeknek elegettevő a_i, b_i párokhoz milyen további feltételek teljesülése esetén rendelhető megfelelő téglalappakolás?

G Függelék

Bevezetés az Excel Solver használatába

Az Excel táblázatkezelő nem elsősorban optimalizálásravaló, mégis a Solver nevű kiegészítő eljárása alkalmas lineáris és nemlineáris optimalizálási feladatok megoldására.

Az Excel 2007-es változatának Bal fölső sarkában a színes gombot megnyomva, majd ott az Excel beállításait kérve érhetjük el Bővítmények kezelését. Itt a választékból jelöljük meg a Solvert. Ezután meg kell nyomni az Ugrás gombot, majd az új párbeszédes ablakban ismét kipipálni a Solvert, és az OK gombbal érvényesíteni a döntésünket. Ezt követően a Solver használatra kész, az Adatok fül választása után jobb oldalt fönn találjuk az elemzés rovatban.

7.1. Alkalmazási példa

Tekintsük a $\min x^2,\ x \geq 1$ egyszerű korlátozott nemlineáris optimalizálási feladatot. Ennek megoldásához Írjunk 2-t az A1 cellába, a B1-et pedig töltsük ki úgy, hogy a szerkesztősorba =A1*A1 -et viszünk be. Utóbbi esetben persze az A1 cellát meg is mutathatjuk az egérrel, nem kell begépelni a címét. Ha mindent jól csináltunk, akkor B2 tartalma 4 lesz.

Ezután nyomjuk meg az Adatok fül elemzés rovatában a Solver gombot. Ennek hatására egy új párbeszédes ablak jelenik meg, amelyben a feladatunkat megadhatjuk. A célcella az optimalizálandó függvényt kell hogy tartalmazza, tehát vagy írjuk be a rovatba, hogy B2, vagy a rovat jobb szélén levő gomb megnyomása után mutassuk meg a B2 cellát az egérrel. Ez után ENTER-t nyomva, vagy az új ablak jobb szélén levő gombbal vissza tudunk jutni az alap Solver ablakba.

A Solver több dolgot tud végrehajtani: minimalizálni, maximalizálni, vagy célértéket keresni. A második sorban lévő választási lehetőségekből kérjük a minimalizálást a megfelelő rádiógombbal.

A Solver párbeszédes ablaka harmadik sorában válasszuk ki módosuló cellaként az A1-et. Amennyiben több változós lenne a feladatunk, akkor persze minden érintett cellát meg kellene mutatni.

Végezetül állítsuk be a feladatunk korlátozó feltételét. Ehhez a Solver ablak bal alsó részében lévő ablakbe kell bevinnünk a feltételt. Nyomjuk meg a hozzáadás gombot ettől jobbra. A megjelenő új ablakban bal oldalon mutassuk meg az A1 cellát, ami az x változónkat tartalmazza, a relációjelet változtassuk meg nagyobb egyenlúre, és a jobb oldali rovatba írjunk be 1-et. Az OK gomb megnyomásával fel is vesszük ezt a feltételt a teljesítendők közé, és vissza is térünk a Solver ablakba. Ha több feltételt kell megadnunk, akkor mindegyik után a Felvesz gombot nyomjuk meg.

Ezzel minden lényeges adatot megadtunk, a feladat megoldásra kész. A Solver ablak jobb felső sarkában lévő Megoldás gombot megnyomva A Solver jelenti, hogy megoldást talált, de

kérdezi, hogy a régi táblázatot felülírhatja-e ezzel. A felülírás elfogadása után megkapjuk a helyes eredményt: a feladatunk keresett szélsőértéke 1, és ezt a célfügvény az 1 lehetséges megoldási pontban veszi föl. □

A célérték keresés a jelen kurzus keretén kívül esik, gyakorlati alkalmazásokban hasznos lehet. Amennyiben az optimalizálási feladat megoldása során az elvárhatónál gyengébb minőségű közelítést kaptunk, a Solver ablak Beállítás gombját megnyomva tudunk állítani a megoldó algoritmus paraméterein. Túl magas elvárásaink ne legyenek, de sok gyakorlati helyzetben jól használható a Solver.

7.2. Házi feladatok

1. Határozzuk meg az $f(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$ Rosenbrock feladat minimumát a (-1, 1) pontból indulva! Próbáljunk ki több beállítási lehetőséget a megoldás módszerére vonatkozóan!

2. Határozzuk meg a $f(x) = x^6(\sin(1/x) + 2)$ függvény szélsőértékét az 1 pontból indulva! Kísérletezzünk, hogy milyen módon tudunk a valódi megoldáshoz közel kerülni.

3. Keressünk választ a Fermat sejtésre az $(a^n + b^n - c^n)^2 + \sin^2(a\pi) + \sin^2(b\pi) + \sin^2(c\pi)$ függvény minimalizálásával $n \geq 2$ esetére! Érveljünk a kapott eredmény alapján a globális optimalizálási feladatok nehézsége mellett!

H Függelék

Bevezetés a MATLAB használatába

A MATLAB egy numerikus programkönyvtár, amely elsősorban mátrixműveletek hatékony alkalmazására készült (innen a neve is). Mindazok, akiknek van tapasztalata magasszintű programozási nyelvekkel, és dolgoztak már ciklusokkal, feltételes utasításokkal, szubrutinhívással, és logikai relációkkal, azok ezt a tudást közvetlenül használhatják a MATLAB alkalmazása során.

A programcsomag számos numerikus eljárást tartalmaz, könnyen használható két- és háromdimenziós megjelenítést, és magas szintű programozhatóságot. A MATLAB elsősorban azért alkalmas a közelítő számítások oktatására, mert könnyen lehet a segítségével ilyen programokat írni és módosítani.

A következő rövid bevezetés a MATLAB használatába inkább csak támogatást nyújt, de a programcsomag teljes körű megismeréséhez valamely felhasználói kézikönyvet érdemes elolvasni (pl. [**?**]).

A MATLAB programot a Linux és a Windows operációs rendszerekben a szokásos módon, a megfelelő ikonra való dupla kattintással lehet elindítani. Az ezután kapott párbeszédes ablakban a felhasználó által begépelt utasításokat a `>>` prompt után lehet megadni. A programból való kilépéshez egyszerűen írjuk be a `quit` vagy az `exit` utasítást a prompt után.

A tárgyalandó további nagyobb témák:

- Programozás m-fájlokkal: szkriptek
- Adattípusok (osztályok) a MATLAB-ban
- Hogyan lehet hatékony programokat írni MATLAB-ban?
- Adatállományok olvasása és írása
- Ritka mátrixok kezelése
- A MATLAB súgó rendszere
- Polinomok a MATLAB-ban

Aritmetikai műveletek és függvények

Itt és a továbbiakban is a MATLAB utasításokat `typewriter` betűtípussal írjuk. Az alapvető műveletek írásmódja nem nagyon meglepő:

`+`	összeadás
`-`	kivonás
`*`	szorzás
`/`	osztás
`^`	hatványozás
`i, pi`	a megfelelő konstansok
`NaN`	Not-a-Number, nem ábrázolható szám
`Inf`	végtelen

A programozási nyelvekben megszokott standard függvények itt is elérhetők, és nevük is közel van a szokásoshoz, pl.:

```
abs(#)   cos(#)   exp(#)   log(#)   log10(#)   cosh(#)   sin(#)
tan(#)   sqrt(#)  floor(#) acos(#)  tanh(#)
```

A # persze az adott függvény argumentumát jelöli, és további információt más függvényekről a MATLAB online súgója ad.

PÉLDA. Tekintsük a következő egyszerű képletet, amely a π egy közelítésének pontos decimális jegyei számát adja meg:

$$-\log_{10}\left(\frac{3.141626-\pi}{\pi}\right).$$

Ennek MATLAB kódja, amit tehát a $>>$ jelű prompt után kell begépelni:

```
>> -log10 ((3.141626 - pi) / pi)
```

Az ENTER megnyomása után a következő választ (answer) kapjuk:

```
ans =
      4.9741
```

Ennek eléréséhez tehát nem kellett semmit se írni a sorvégére. Ha visszaírt válasz nélkül kérjük, akkor pontosvesszőt kell írni a sor végére, mint hasonló rendszerekben.

Alapértelmezésben tehát 5 jegyet kapunk. Ha pontosabb megjelenítésre van szükség, akkor a `format long` utasítás kb. 15 decimális jegyet eredményez:

```
>> format long
3*cos(sqrt(4.7))
ans =
      -1.68686892236893
```

A felhasználók saját függvényeiket ún. M-file-okban adhatják meg. Ezek az adatállományok a MATLAB saját formátumát követik, .m kiterjesztésűek, és a MATLAB támogatja a szerkesztésüket. A fentiek szerint definiált új függvényeket ugyanúgy lehet a MATLAB-on belül használni, mint a rendszer saját függvényeit. Ha elsőre nem találja a frissen írt .m állományt, akkor ellenőrizzük a Set Path utasítást a File menüsorban, és ha kell, adjuk az új könyvtárat az eddigiekhez.

PÉLDA. Definiáljuk a $\mathrm{fun}(x) = 1 + x - x^2/4$ függvényt a MATLAB Editor/Debugger ablakában, és mentsük el a fun.m állományba. Ehhez a következő formátumot kell követni:

```
function y=fun(x)
y=1+x-x.^2/4;
```

A „.^" használatát hamarosan megmagyarázzuk. A változók és a függvények nevében használhatunk kis és nagybetűket, a lényeg az, hogy a hivatkozások során következetesen járjunk el. Ezután a MATLAB parancsablakában (Command Window) a következő módon használhatjuk a definiált függvényt:

```
>> cos(fun(3))
ans=
      -0.1782
```

A függvény kiértékelésére használhatjuk a `feval` utasítást is:

```
>> feval('fun',4)
ans=
      1
```

Vegyük észre, hogy ebben az esetben a függvény nevét karaktersorozatként kell megadni.

Műveletek mátrixokkal

A mátrixok kezelése érthető módon az erőssége a MATLAB-nak. Lényegében minden változót mátrixként kezel:

```
>> A=[1 2 3;4 5 6;7 8 9]
A=
      1 2 3
      4 5 6
      7 8 9
```

Ahogy látható, a mátrixok definiálásában a sorokat pontosvesszővel választjuk el, az egyes mátrixelemeket pedig szóközzel. A mátrixokat soronként begépelve is be lehet vinni:

```
>> A=[1 2 3
     4 5 6
     7 8 9]
A=
      1 2 3
      4 5 6
      7 8 9
```

A mátrixokat beépített függvények segítségével is generálhatjuk:

```
>>Z=zeros(3,5);          egy 3-szor 5-ös, csupa nullából álló mátrixot ad,
 >>X=ones(3,5);          egy 3-szor 5-ös, csupa egyesből álló mátrixot kapunk,
 >>Y=0:0.5:2             egy 1-szer 5-ös mátrixot generál:
Y=
   0 0.5000 1.0000 1.5000 2.0000
```

A definiált mátrixokon elemenként függvényeket lehet végrehajtani:

```
>>cos(Y)            egy megfelelő 1 × 5-ös mátrixot ad:
ans=
      1.0000 0.8776 0.5403 0.0707 −0.4161
```

A mátrixok komponenseit ügyesen lehet kezelni a MATLAB-ban, tekintsük például a következő utasításokat:

```
>>A(2,3)              az A mátrix egy elemét választja ki,
ans=
      6
A(1:2,2:3)           az A mátrix egy részmátrixát adja,
ans=
       2 3
      5 6
A([1 3],[1 3])       az A mátrix egy részmátrixa kiválasztásának egy másik módja:
ans=
      1 3
      7 9
>>A(1,1)=sin(3.14); értékadás egy mátrixelemnek.
```

A mátrixokra a következő műveleteket alkalmazhatjuk:

```
+          összeadás,
−          kivonás,
*          szorzás,
^          hatványozás, és
'          konjugált transzponálás.
```

PÉLDA. A mátrixműveletek illusztrálásaként tekintsük a következő egyszerű utasítássort:

```
>>B=[1 2;3 4];
>>C=B'
```

```
C=
      1 3
      2 4
>>3*(B*C)^3
ans=
      13080 29568
      29568 66840
```

Itt tehát C a B transzponáltja, és az utolsó mátrix a $3(B * C)^3$.

A felsoroltakon túl természetesen számos egyéb utasítás érhető el mátrixok manipulálására. Ezekkel kapcsolatban érdemes az online súgót, a felhasználói kézikönyvet, vagy más leírást (pl. [**?**]) tanulmányozni.

A MATLAB erőssége azon függvények széles köre, amelyek mátrixok elemein hajthatók végre. Korábban erre láttunk példát, amikor egy 1×5-ös mátrix elemeinek koszinuszát határoztuk meg. A mátrixokra vonatkozó összeadás, kivonás és skaláris szorzás természetesen mátrix elemenként történik, de a szorzás, osztás és a hatványozás már nem. Ahhoz, hogy ezeket a műveleteket a mátrixelemeken hajthassuk végre, a műveleti jelek elé egy pontot kell írni: .*, ./ és .^. A mátrixokra és azok elemeire vonatkozó műveleteket nem szabad összekeverni:

```
>>A=[1 2;3 4];
>>A^2              ez az AA mátrixszorzatot adja:
ans=
      7 10
      15 22
>>A.^2             ezzel az A mátrix elemei négyzetét kapjuk:
ans=
      1 4
      9 16
>>cos(A./2)        ezzel pedig az A mátrix elemei felének koszinuszát határozzuk meg:
ans=
      0.8776  0.5403
      0.0707 -0.4161
```

Megjelenítés

A MATLAB görbék és felületek két-, vagy háromdimenziós ábráit tudja megjeleníteni. Az itt röviden bemutatott lehetőségeken túliakat az online súgó, szakkönyv ([**?**]), vagy a felhasználói leírás segítségével kereshetjük meg.

A kétdimenziós görbék megjelenítésére a `plot` utasítást lehet használni. A következő példa az $y = \cos(x)$ és az $y = \cos^2(x)$ függvényeket ábrázolja a $[0, \pi]$ intervallumon:

```
>>x=0:0.1:pi;
>>y=cos(x);
>>z=cos(x).^2;
```

```
>>plot(x,y,x,z,'o')
```

Az első sor adja meg a megjelenítési tartományt, 0.1 lépésközzel. A következő kettő definiálja a két függvényt. Vegyük észre, hogy az első három sor mindegyike pontosvesszővel végződik. Ez megakadályozza, hogy az ezekben definiált mátrixok megjelenjenek a párbeszédes ablakban. A negyedik sor tartalmazza a plot utasítást, és jeleníti meg a grafikont. Az első két argumentum eredményezi az $y = \cos(x)$ függvény ábrázolását, az utolsó három pedig a $z = \cos^2(x)$ megjelenítését, éspedig úgy, hogy az egyes (x_k, z_k) pontokat 'o' jelöli.

A plot utasításnak egy hasznos alternatívája az fplot. Általános alakja a következő:

```
fplot('name',[a,b],n).
```

Ennek hatására a program veszi a `name.m` adatállományból a függvényt, és az $[a, b]$ intervallumból vett n darab alappontban meghatározott érték alapján elkészül az ábra. Az n alapértelmezése 25.

```
>>fplot('tanh',[-2,2])
```
a $\tanh(x)$ függvényt jeleníti meg a $[-2, 2]$ intervallumon.

A `plot` és a `plot3` utasítások alkalmasak paraméteres függvények két-, illetve háromdimenziós ábrázolására. Ezek különösen differenciálegyenletek megoldásának megjelenítésére alkalmasak. Például a $c(t) = (2\cos(t), 3\sin(t))$ ellipszist a $0 \leq t \leq 2\pi$ tartományon a következő utasítással lehet ábrázolni:

```
>>t=0:0.2:2*pi;
>>plot(2*cos(t),3*sin(t))
```

A $c(t) = (2\cos(t), t^2, 1/t)$ 3-dimenziós görbe képét a $0.1 \leq t \leq 4\pi$ paraméter-tartományon pedig a következő utasítással lehet ábrázolni:

```
>>t=0.1:0.1:4*pi;
>>plot3(2*cos(t),t.^2,1./t)
```

A háromdimenziós felületek ábrázolásához egy téglatestet kell megadni a felület értelmezési tartományán, amelyet a `meshgrid` paranccsal lehet definiálni. Ezután a `mesh` vagy a `surf` parancsokkal kapjuk az ábrát, mint a következő példában is:

```
>>x=-pi:0.1:pi;
>>y=x;
>>[x,y]=meshgrid(x,y);
>>z=sin(cos(x+y));
>>mesh(z)
```

(Vegyük észre, hogy az utolsó sorban egyik példában sincs pontosvessző a sor végén.)

Programok, ciklusok, vezérlés

A logikai és relációs jelek, műveletek a MATLAB-ban is hasonlók a magasszintű programozási nyelvekben megszokottakhoz:

Relációjelek:

`==`	egyenlő
`~=`	nem egyenlő
`<`	kisebb
`>`	nagyobb
`<=`	kisebb vagy egyenlő
`>=`	nagyobb vagy egyenlő

Logikai műveletek és konstansok:

`~`	negáció
`&`	és
`|`	vagy
`1`	igaz
`0`	hamis

A `for`, `if` és `while` utasítások a MATLAB-ban is úgy működnek, mint a hasonló programozási nyelvekben. Ezeknek az alapvető formája:

```
for (ciklusváltozó = kifejezés)
     utasítások
end

if (logikai kifejezés)
     utasítások
else
     utasítások
end

while (kifejezés)
     utasítások
end
```

A következő példa azt mutatja, hogyan lehet egymásbaágyazott ciklusokkal egy mátrixot generálni. Ha a példaprogramot `nest.m` néven mentjük el, akkor a MATLAB promptjához `nest`-et írva az A mátrixot eredményezi. Vegyük észre, hogy a mátrix bal felső sarkából kiindulva a Pascal háromszöget kapjuk.

```
for i=1:5
    A(i,1)=1;A(1,i)=1;
end
for i=2:5
    for j=2:5
        A(i,j)=A(i,j-1)+A(i-1,j);
    end
end
A
```

A `break` parancs hatására befejeződik egy ciklus:

```
for k=1:100
    x=sqrt(k);
    if ((k>10)&(x-floor(x)==0))
      break
    end
end
k
```

A `disp` utasítás szöveg, vagy egy mátrix kiíratására használható:

```
n=10;
k=0;
while k<=n
    x=k/3;
    disp([x x^2 x^3])
    k=k+1;
end
```

A MATLAB programírás hatékony módja a felhasználó által definiált függvények összeállítása. Ezek input és output paramétereket is használhatnak, és más programokból szubrutinként hívhatók. Ennek bemutatására tekintsük a következő egyszerű programot, amit a MATLAB File / New menüpontban elérhető Editor / Debugger segítségével szerkesszünk meg, és mentsük el `pasc.m` néven.

```
function P=pasc(n,m)
%Input    - n a sorok száma
%         - m a prímszám
%Output   - P a Pascal háromszög

for j=1:n
    P(j,1)=1;P(1,j)=1;
end
for k=2:n
    for j=2:n
        P(k,j)=rem(P(k,j-1),m)+rem(P(k-1,j),m);
```

```
    end
end
```

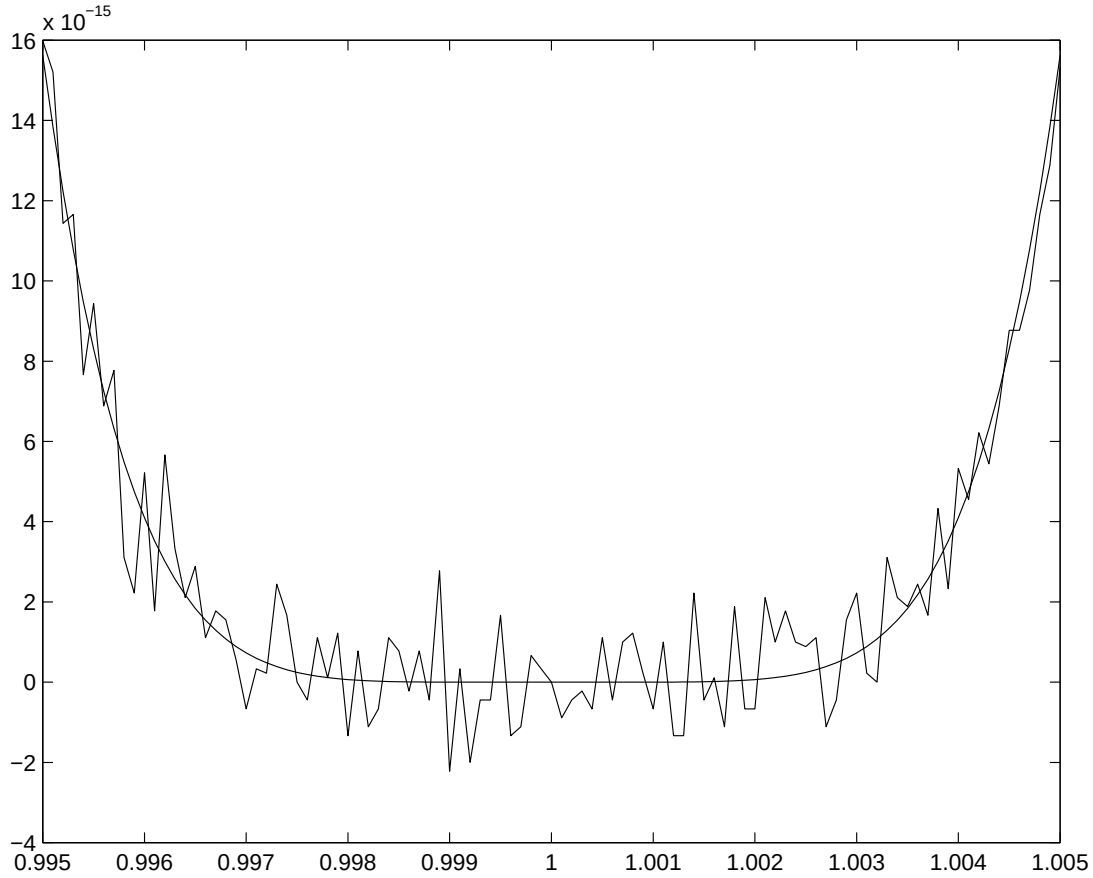
Ezután a MATLAB parancssorába írjuk be azt, hogy `P=pasc(5,3)`, és láthatjuk a mod 3 Pascal háromszög első 5 sorát. A másik érdekes teszt `P=pasc(175,3);` (most fontos a pontosvessző), és aztán gépeljük be azt, hogy `spy(P)`, ami ritka mátrixot generál nagy n esetén.

PÉLDA. Az alábbi rövid Matlab program megjeleníti az $y = (1-x)^6$ függvényt, és ennek Horner-elrendezés szerint átrendezett, de ekvivalens alakját, ahol

$$z = ((((((x-6)*x+15)*x-20)*x+15)*x-6)*x+1).$$

```
>> x = (9950:10050)/10000; % definialja a pontsorozatot
>> y = (1-x).^6;
>> z = ((((((x-6).*x+15).*x-20).*x+15).*x-6).*x+1);
>> plot(x,[y;z]); % egy grafikont jelenit meg
>> print -deps hornerdemo.ps % kiirja egy fajlba
```

A kapott ábra a két nagyon eltérő görbével (a sima az $(1-x^6)$):



Néhány olyan Matlab utasítás, amely az adott számítógép, illetve a szoftver környezet megismerését szolgálja:

az `eps` a gépi pontosság aktuális értékét adja,

a `computer` azonosítja a használt számítógép típusát,

a `realmax` a legnagyobb pozitív gépi szám,

a `realmin` a legkisebb pozitív gépi szám.

Végül, meglepetésként gépeljük be `spy` utasítást, és a sorvégi pontosvessző nélkül nyomjuk meg az Enter gombot.

I Függelék

Az esszé követelményei

A félévi munka egyik fontos eleme a megírandó esszé. Ez a kapható pontszám komoly részét adja, és ez lenne az eredménye a hallgatók önállóan végzendő munkájának. Az esszé egy olyan rövid (15-20 oldalas) jelentés, amely a következő főbb részeket kell hogy tartalmazza:

1. A kitűzött feladat, téma, módszer pontos, részletes megfogalmazása, a szakirodalom megismerése alapján annak alapos leírása, kitérve a nehézségeire, és eltéréseire a szomszédos területektől. Fontos részletezni az alkalmazási területeket. Érdemes itt kis méretű, áttekinthető példákat használni.

2. A megoldására megírt, felhasznált programok részletes megadása, leírása. Ki kell térni az alkalmazott számítógépes megoldások okaira, előnyeire is (mint pl. a választott adattípus, számformátum, algoritmus, stb.).

3. A bemutatott algoritmusok hatékonyságát, sebességét, műveletigényét, pontosságát és egyéb említésre méltó tulajdonságát alkalmas teszteléssel kell demonstrálni. Fontos azt is jellemezni, hogy milyen méretű feladatok megoldását lehet a tárgyalt módszerekkel elérni. Ennek eredményét táblázatos vagy grafikonos formában, kifejező módon kell megadni.

4. Az esszé foglalja össze a szűkebb szakterület mélyebb vagy szélesebb körű megismeréséhez ajánlható irodalmat is, legyen az nyomtatott vagy elektronikus formájú.

Az esszét nyomtatott vagy elektronikus formában kell a gyakorlatvezetőnek benyújtani (a konkrét feltételeket a gyakorlatvezető adja meg). A használt szövegszerkesztő lehetőleg LaTeX vagy Word legyen.

Világos, hogy a heti egy órányi gyakorlat nem elegendő a globális optimalizálás olyan szintű megismertetésére, amelyre támaszkodva az eddigiekben leírt szintű esszé minden további nélkül megírható lenne. A tárgy célja viszont az is, hogy az önálló munkát serkentse, hozzászoktassa a hallgatóságot ahhoz, hogy egy kapott feladathoz a szükséges részletesebb ismereteket maga szerezze meg. Ehhez érdemes támaszkodni a hálózaton keresztül elérhető előzetes jegyzet irodalomjegyzékére, az interneten elérhető adatokra, és az évfolyamtársak segítségére is. Korlátozott mértékben a gyakorlatvezetővel való konzultáció is segíthet.

Az esszé önálló munkát kell hogy tükrözzön, ez azonban nem zárja ki, hogy a hallgató másokkal közösen dolgozzon. Lényeges viszont, hogy a kapott feladat megoldásának minden részletével tisztában legyen, így a leadáskor vagy később kapott, az esszével kapcsolatos kérdésekre kimerítő, teljes választ tudjon adni.

Az esszé értékét növeli, ha az a Matlab, Scilab, Netlib vagy Octave programjait tárgyalja. A hallgató javasolhat is esszé formában feldolgozandó témát, ezt azonban csak a gyakorlatvezető előzetes egyetértése esetén lehet elfogadni. Ez a lehetőség például szakdolgozat, diplomamunka, diákköri munka során már megismert témák bevonása révén előnyös lehet.

Néhány esszétéma:

1. A NEOS optimalizálási szerver
2. A GAMS modellezési nyelv, formalizálási rendszer
3. A CPLEX optimalizálási rendszer
4. A MINOS optimalizálási rendszer
5. Milyen segítséget kaphatunk optimalizálási feladatok megoldásához a Maple szimbolikus algebrarendszertől?
6. Milyen segítséget kaphatunk optimalizálási feladatok megoldásához a Mathematica szimbolikus algebrarendszertől?
7. Milyen segítséget kaphatunk optimalizálási feladatok megoldásához a Derive szimbolikus algebrarendszertől?
8. Milyen segítséget kaphatunk optimalizálási feladatok megoldásához a Mu-pad szimbolikus algebrarendszertől?
9. Nemlineáris függvény minimumának megkeresése (egy változós eset) a Matlab numerikus programrendszerben
10. Nemlineáris függvény minimumának megkeresése (több változós eset) a Matlab numerikus programrendszerben
11. Az automatikus differenciálás szerepe a nemlineáris optimalizálásban
12. Az intervallum-aritmetika szerepe a nemlineáris optimalizálásban
13. A mesterséges neuronhálók szerepe az optimalizálásban
14. Az ún. hangyakolóniák módszere
15. Genetikus algoritmusok az optimalizálásban
16. Szimulált megeresztés (simulated annealing) az optimalizálásban
17. A nemlineáris szimplex módszer
18. Pakolási feladatok
19. Geometriai alakzatok lefedési problémái
20. Mit tud az Excel Solver a globális optimalizálás terén?

J Függelék

Egy minta esszé

Itt egy olyan esszét adunk meg, amely mintaként szolgál a hallgatóknak. Az itt megadott esszé kb. a kettes szintet jelenti. Ehhez képest a jobb jegyet érdemlő esszé alaposabb áttekintését nyújtja a kiírt témának, részletesebb tesztelést és kifejezőbb illusztrációt tartalmaz.

Nemlineáris optimalizálás a Matlabban

1. A nemlineáris optimalizálás alapfeladatát a következő formában szokás megadni:

$$\min f(x)$$

$$g_i(x) \leq 0,\ i = 1, 2, \ldots, m_1,$$

$$h_j(x) = 0,\ j = 1, 2, \ldots, m_2,$$

ahol $f(x) : \mathbb{R}^n \to \mathbb{R}$ egy általában kétszer folytonosan differenciálható függvény, m_1 darab egyenlőtlenség és m_2 darab egyenlőségfeltétel határozza meg a lehetséges megoldások halmazát. Itt a g_i és a h_j függvények hasonlóan általában kétszer folytonosan differenciálható függvények. Az n számot a feladat dimenziójának hívjuk. A feladat a nevét arról kapta, hogy az említett függvények nemlineárisak lehetnek.

Ezzel szemben az operációkutatás című tárgyban megismert lineáris programozási feladat leírásában szereplő minden függvény lineáris. Ez az eltérés lényegi, a lineáris programozásra használatos algoritmusok esetünkben haszontalanok. Amíg a lineáris programozási feladathoz létezik polinomiális műveletigényű megoldó algoritmus, addig a nemlineáris optimalizálási feladat több nagyon egyszerű részproblémája is NP-teljes. Részben emiatt is a legtöbbször megelégedünk közelítő megoldással.

A feladat nehézségét jelzi, hogy általános esetben több lényegesen különböző helyi minimumpont van (aminek van olyan környezete, amelyben nincs nálánál kisebb célfüggvényértékű pont), és ezek teljes körű megismerése, összevetése nehéz. Maga a nemlineáris optimalizálás emiatt az esetek túlnyomó többségében megelégszik egy helyi minimumpont megtalálásával.

Az alkalmazási terület meglehetősen széles. A közkeletű felfogás szerint lényegében minden érdekes gyakorlati probléma nemlineáris. Eszerint a lineáris optimalizálási feladatok a legtöbbször egyszerűsítés után jönnek képbe. Az e felfogással ellentétes vélemény szerint pedig minden, amit a számítógépen megoldunk, az lineáris feladat... A való helyzet valószínűleg a két álláspont között van.

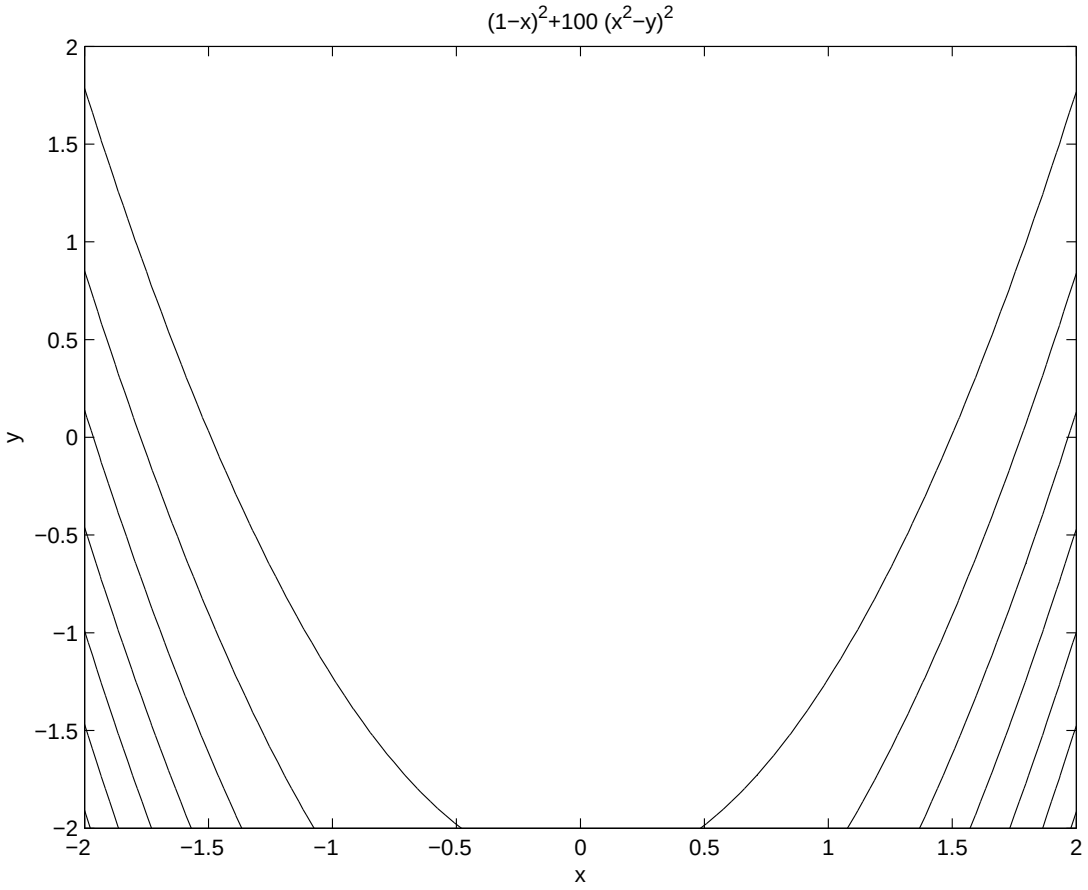
A nemlineáris feladatosztály egyik sokszor idézett példánya a Rosenbrock feladat:

$$\min(1-x_1)^2+100(x_1^2-x_2)^2$$
$$-2\leq x_1,x_2\leq 2.$$

Az érdekességét az adja, hogy egyrészt szemléletesen azonnal látszik, hogy mi a megoldás, mégis a hagyományos nemlineáris optimalizáló algoritmusoknak meggyűlik a bajuk vele.

Ha papiron ceruzával kell megoldani, akkor a következő érvelést lehet használni. A célfüggvény két nemnegatív szám összege, ennek lehető legkisebb értéke így nulla. Világos, hogy az első tag akkor lesz nulla, ha $x_1 = 1$. Ha ezt beírjuk a második tagba, akkor azt kapjuk, hogy x_2-nek is egynek kell lennie ahhoz, hogy a célfüggvény optimumát megkapjuk. Ezután már csak az van hátra, hogy ellenőrizzük, hogy ez a pont benne van-e a lehetséges megoldások halmazában. Mivel a korlátozó feltételek most csak egyszerű alsó- és felső korlátok az argumentumokra, ezt könnyű ellenőrizni.

Ez könnyen ment, nehéz elképzelni, hogy mi miatt jelent ez gondot a számítógépes algoritmusoknak. A magyarázathoz nézzük meg a célfüggvény szintvonalait (azon görbéket, amelyek mentén a célfüggvény értéke állandó):



1. Ábra. A Rosenbrock függvény szintvonalai a $[-2,2]^2$ tartományon. Az sajnos nem nagyon látszik, hogy a kisebb függvényértékű pontok egy ívelt, banán formát mutatnak. A rajz az `ezcontour('(1-x)^2+10*(x^2-y)^2',[-2 2 -2 2]);` Matlab utasítással készült.

Namármost a legtöbb optimalizáló program a célfüggvény különböző modelljén alapulva annak lineáris, de inkább kvadratikus alakját használja. Ezek a modellfüggvények viszont

a célfüggvény relatív egyszerűsége ellenére következetesen eltérnek attól, és emiatt a kereső programok sok lépésből álló iterációra kényszerülnek. A megoldások jellemző megjelenése emiatt egy sok rövid szakaszból álló törtvonal, amely megkísérli a banán alakú völgy alját követni. összefoglalva azt mondhatjuk, hogy a Rosenbrock függvény nehezen, viszonylag nagy műveletigény árán oldható meg.

2. A Matlab maga meglehetősen kevés támogatást nyújt optimalizáláshoz. Van viszont egy kiegészítő csomagja, az Optimization Toolbox. A jelen esszé ennek lehetőségeire nem tér ki. Említésre méltó viszont három Matlab rutin: az `fzero`, amely valójában egyváltozós függvények zérushelyét keresi meg, az `fminbnd`, ami egyváltozós függvények minimalizálására vállalkozik és az `fminsearch`, ami pedig 'többdimenziós függvények minimalizálását végzi. Lássuk ezek alkalmazását.

2.1 Tekintsük először az `fzero` eljárást. Ahhoz, hogy ezt be lehessen vetni, az eredeti feladatot át kell fogalmazni egyenlet megoldássá. Sajnos a feladat maga az nagyon többdimenziós, egy változóval épp a lényegét, a kanyarodó völgyet veszítjük el. Minden esetre tekintsük azt az egyenest, amely átmegy a minimumponton, és amely mentén egyszerűen kifejezhető a függvényünk: $y = x$.

Ha az `y` minden előfordulását `x`-re cseréljük az eredeti függvényben, akkor egy egyváltozós függvényt kapunk, amely épp az említett egyenes mentén adja meg a Rosenbrock függvény értékeit:

$$f_1(x) = (1 - x)^2 + 100(x^2 - x)^2.$$

Ez szemre még hasonló függvény, és a polinom fokszáma is megfelel az eredetiének. Ennek a függvénynek keressük tehát a minimumát. Ahhoz hogy ennek megtalálásához az `fzero` eljárást használni tudjuk, a minimalizálási feladatot át kell írni zérushely keresési feladattá. Ehhez tűzzük ki azt a feladatot, hogy megkeresendő az első derivált zérushelye. Egy egyváltozós függvény deriváltjának zérushelye nem mindig minimum (lehet maximum is), de legalább a nyeregpontokkal nem kell foglalkozni. Az f_1 függvény deriváltja:

$$f_2(x) = -2(1 - x) + 200(x^2 - x)(2x - 1).$$

Ennek a zérushelyének a meghatározásához gépeljük be a következő Matlab utasítást:

```
>> fzero('-2*(1-x)+200*(x^2-x)*(2*x-1)',0)
```

A kapott válasz meglepő:

```
ans =
       0.0102
```

Ha ezt az eredményt visszaírjuk a vizsgált függvénybe, akkor

```
-2*(1-0.0102)+200*(0.0102^2-0.0102)*(2*0.0102-1)
ans =
      -0.0016
```

adódik. Hát ez nem nagyon szép. Ami bíztató, az az, hogy ha a keresett megoldás közelében adjuk meg a második argumentumban az indulópontot, akkor már jobb eredményt kapunk:

```
>> fzero('-2*(1-x)+200*(x^2-x)*(2*x-1)',2)
ans =
       1
```

Vegyük észre, hogy ez maga az egészszám, amit kerestünk, tehát nem is 1.0000 az eredmény, ami a közelítő jellegre utalhatott volna. Ehhez illusztrálásaként tekintsük a következő számítást:

```
>> 1000001/10000000
ans =
       1.0000
```

Térjünk vissza hamisnak tűnő megoldásra. Ha most a visszahelyettesítéshez nem a kiírt értéket használjuk, hanem a pontosat, akkor lényegében igazoljuk, hogy a talált megoldás egy zérushely:

```
>> y = fzero('-2*(1-x)+200*(x^2-x)*(2*x-1)',0)

y =
       0.0102

-2*(1-y)+200*(y^2-y)*(2*y-1)
ans =
      2.2204e-016
```

Ez az érték ugyanis a dupla pontos számábrázolás határán van — még ha a nullát épp ennél sokkal jobban meg is lehet közelíteni.

Ebből az következik, hogy itt bizony van egy másik gyök, tehát a Matlab a feltett kérdésre helyesen válaszolt — vagyis nekünk az egyváltozós változat megoldásait ellenőriznünk kell.

A tapasztalatunk szerint a keresett megoldást akkor kapjuk meg, ha az indulópontot a [0.8, 10] intervallumban választjuk meg. Ez minden esetre óvatosságra int a `fzero` használatával kapcsolatban. Ha sok, véletlenül választott indulóponttal kérjük a zérushelyek meghatározását, akkor összesen három ilyent találunk: 0.0102, 0.4898, és 1.0000. Ha az ezeknek megfelelő [0.0102, 0.0102], [0.4898, 0.4898] és [1.0000, 1.000] pontokat behelyettesítjük az eredeti függvényünkbe, akkor a következő értékeket kapjuk rendre: 0.9899, 6.5051 és 0. Innen egyértelműen megadhatjuk a minimum helyét és értékét, és ez összhangban is van a kézzel elért eredménnyel. Az egyetlen hiányosság az, hogy nem lehetünk biztosak benne, hogy az egyváltozós függvény minden zérushelyét megtaláltuk...

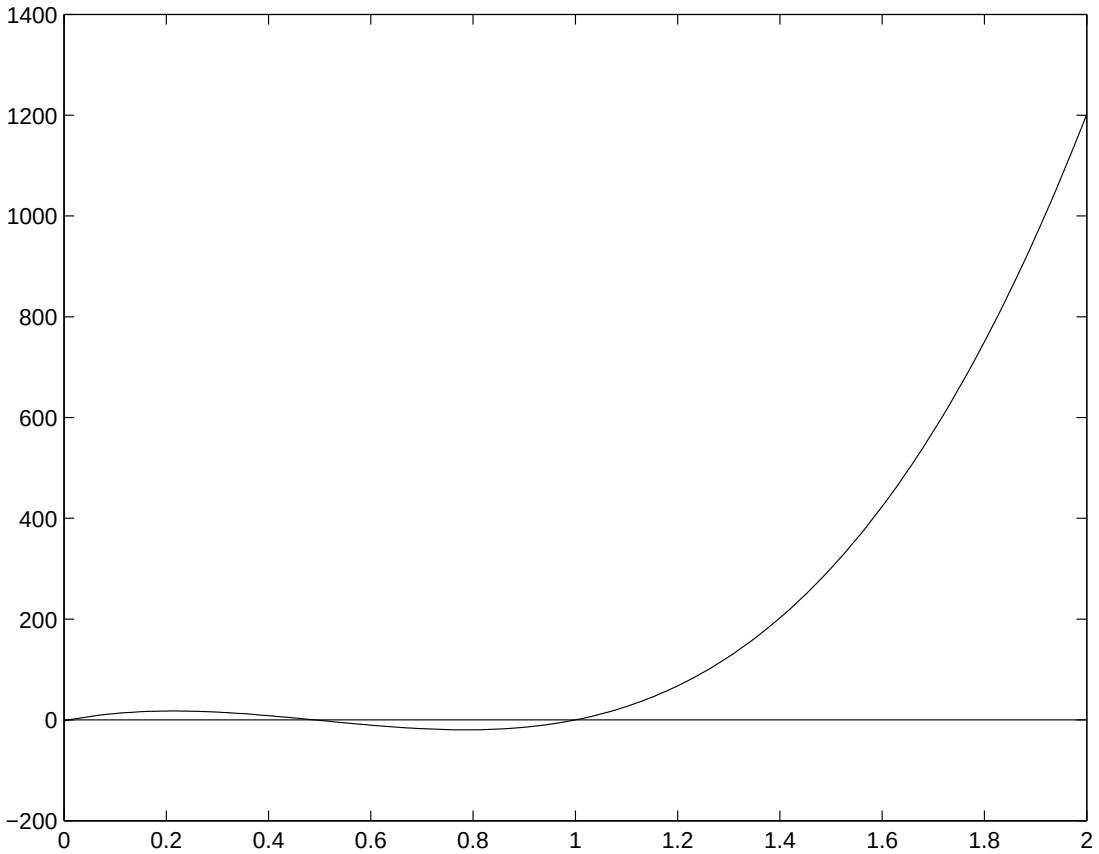
A véletlen indulópontot használó utasítás a [0, 10] intervallumra vonatkozóan:

```
y = fzero('-2*(1-x)+200*(x^2-x)*(2*x-1)',10*rand(1))
y =
   1
```

Az optimalizált függvény alakját a következő rövid program rajzolja ki:

```
>> x = x=0:0.01:2.0;
>> y = -2*(1-x)+200*(x.^2-x).*(2*x-1);
>> z = x-x;
>> plot(x,y,x,z);
```

Ennek eredménye (a `z` függvénnyel az x-tengelyt ravaszkodtam az ábrára...):



2. Ábra. A Rosenbrock függvény $y = x$ egyenes menti értékei deriváltfüggvénye. Kis jóindulattal leolvasható az `fzero` program által megtalált három zérushely.

Az `fzero` utasítás ráadásul csak páratlan multiplicitású zérushelyek meghatározására jó, tehát például az x^2 zérushelyét nem találja. Ez elég rossz hír az optimalizálás szempontjából, mert általában egy nemlineáris függvény deriváltja zérushelyei multiplicitása mindenféle lehet.

2.2. Vegyük most az `fminbnd` utasítást. Ez egydimenziós függvények optimalizálására szolgál. Nézzük először, az x^2 függvénnyel mit tud kezdeni:

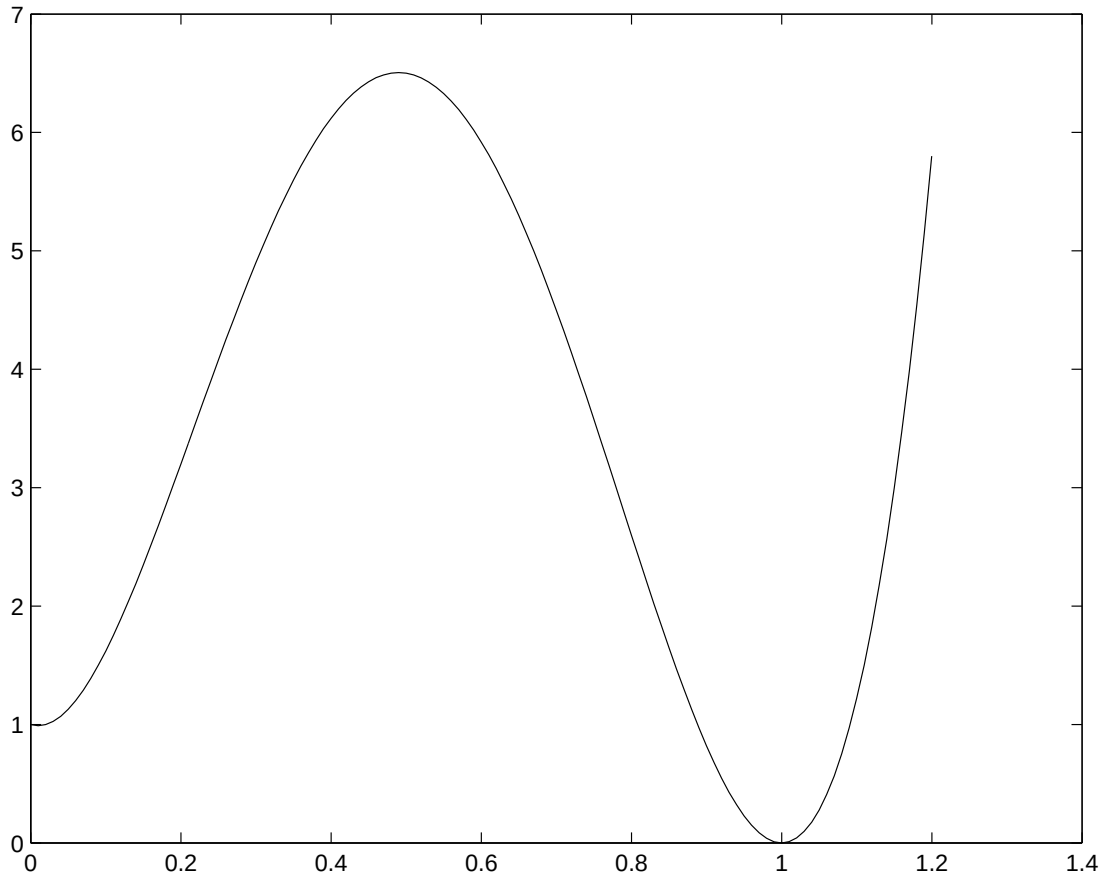
```
fminbnd('x^2',-1,1)
ans =
       -2.7756e-017
```

Ez rendben is van lényegében. A paraméterezés: függvény, alsókorlát, felsőkorlát. Vigyázzunk, az első utasítás után ne tegyünk pontosvesszőt, mert akkor az eredmény nem jelenik meg! Vegyük akkor az előző szakasz egydimenzióssá átalakított Rosenbrock függvényét.

```
>> fminbnd('(1-x)^2+100*(x^2-x)^2',0,2)
ans =
       1.0000
```

Ez is rendben van, bár az $(1-x)^2$ függvény minimumhelyeként az 1-et adta meg (tehát pontosabban tudta megállapítani). Nézzük meg, hogy milyen függvényt is minimalizáltunk:

```
>> x=0.0:0.01:1.2;
>> y=(1-x).^2+100*(x.^2-x).^2;
>> plot(x,y);
```



3. Ábra. A Rosenbrock függvény $y = x$ egyenes mentére korlátozott, egydimenziós változata. Jól látható a megtalált minimum az 1 pontban.

A bal sarokban azért van egy gyanús rész, ha rázoomolunk, akkor

```
>> fminbnd('(1-x)^2+100*(x^2-x)^2',0,0.4)
ans =
       0.0102
```

szépen megkapjuk az előző szakaszban kapott másik minimumjelöltet. Akkor viszont tisztázzuk a 2.1. szakasz 0.4898 szélsőértékét is:

```
>> fminbnd('(1-x)^2+100*(x^2-x)^2',0.4,0.6)
ans =
       0.6000
```

Ez nem segít, mert csak a megadott keresési intervallum szélét kaptuk meg — ez azt jelezheti, hogy az intervallum belsejében nincs megoldás. Valóban,

```
>> fminbnd('(1-x)^2+100*(x^2-x)^2',0.3,0.7)
ans =
       0.7000
```

is erre utal. Nézzük akkor meg a vizsgált függvényünk negáltját:

```
>> fminbnd('-((1-x)^2+100*(x^2-x)^2)',0.3,0.7)
ans =
       0.4898
```

Bingó. Megvan a harmadik pont is: OK, ez nem minimumpont volt, de a maximumpont deriváltja is nulla. Akkor úgy érezhetjük, hogy ez a program az eddigiek szerint lényegében tudja azt, amit el lehet várni tőle, és azt megbízhatóan hozza is.

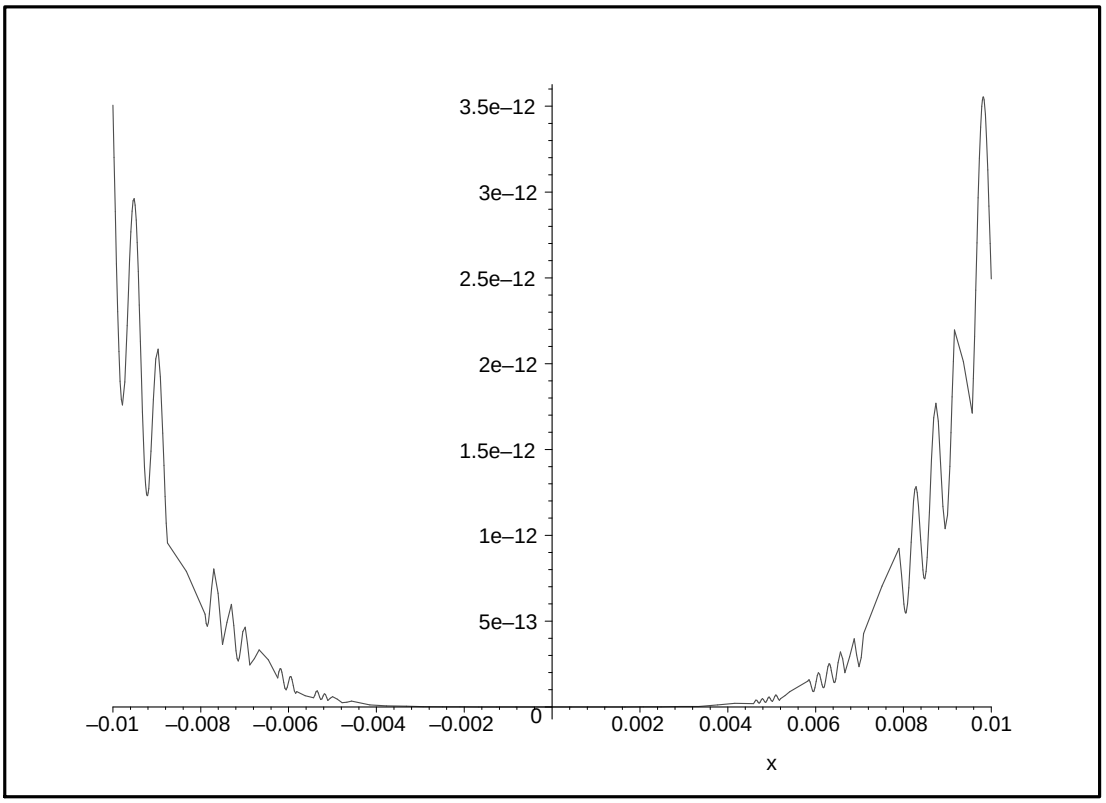
Nézzünk akkor egy nehezebb feladatot. Az $x^6(\sin(1/x)+2)$ elég ellenséges, mert bár kétszer folytonosan differenciálható (ahogy könnyen belátható), mégis a nulla tetszőleges környezetében megszámlálhatóan végtelen sok helyi minimumpontja van. Emiatt a megoldása lényegében reménytelen olyan módszerek számára, amelyek csak a függvény-kiértékeléseken alapulnak. Egyes programok panaszkodnak, hogy a nullában nem tudják kiértékelni a függvényt. Annyiban ez igaz is, hogy maga az $1/x$ függvény persze gondot jelent. Másrészt azonban az x^6-al való beszorzás miatt a függvény maga a nullában is folytonos. Óvatos implementálással a számítógépes megvalósítás is megoldható. A függvény alakja a 4. Ábrán látható, és már ez is elég borzasztó - bár a zűrös szakasz persze a nulla közelében van.

A feladat megoldása az `fminbnd` programmal:

```
>> fminbnd('x^6*(sin(1/x)+2)^2', -1, 1)
ans =
       0.0018
```

Bár ez nem az igazi megoldás, de azért nem nagyon rossz. Végülis a keresési tartomány helyes kb. 4 ezrelékébe sikerült beletalálni. Az már nagyobb baj, hogy ezen nem könnyű javítani. Az azért érdekes, hogy a következő egyszerű eszköz segít:

```
>> fminbnd('x^6*(sin(1/x)+2)^2', -1-rand(1), 1+rand(1))
ans =
       2.5606e-006
```



4. Ábra. Az $x^6(\sin(1/x)+2)$ függvény a [-0.01, 0.01] intervallumban. Jól látható a fő tendencia, és az hogy a függvénynek sok helyi minimumpontja van.

Ez azért csak meglepő, hogy a keresési intervallum határának véletlenszerű tologatása ennyit javít a megoldáson. Ez még akkor is szép, ha a két pont között csak 3 nagyságrendnyi a különbség. Az még nagyobb előny, hogy amíg a rögzített intervallumon nincs értelme újrafuttatni a programot (mert ugyanazt az eredményt fogjuk kapni), addig a véletlennel terhelt oldalú intervallumokkal ez hasznos lehet.

A függvény ismeretében a legjobb az lenne, ha nagyon sok véletlenszerű pontban kiértékelnénk a függvényt, és csak a legígéretesebb közelében indítanánk el az `fminbnd` programot. Tanulságos, hogy a Matlabon belül megkeressük az `fminbnd` kódját, az fminbnd.m állományt, akkor egy mindössze 250 soros programot találunk, és ebből is 50 sor magyarázat, megjegyzés.

2.3. Végül lássuk az `fminsearch` utasítást. Ez hosszra csak kicsit bonyolultabb, mint `fminbnd`, kb. 300 soros, amiből ismét közel 50 sor a magyarázat. A legegyszerűbb alakja:

```
X = fminsearch(FUN,X0);
```

ahol `FUN` a minimalizálandó függvény, `X0` az indulóérték, és `X` a kapott eredmény, egy helyi minimumpont. Vegyünk először megint egy egyszerű, áttekinthető feladatot: mi lehet az $(x+y)^2$ minimuma?

```
>> fminsearch('(x(1)+x(2))^2',[1.2 1.2])
ans =
       0.0249    -0.0249
```

Ez első pillantásra meghökkentőnek tűnhet, de rendben van, hiszen a függvény minden olyan pontja optimális, ahol a két argumentum összege nulla. A csak kicsit eltérő függvénnyel alapjában más megoldást kapunk:

```
>> fminsearch('x(1)^2+x(2)^2',[1.2 1.2])
ans =
     1.0e-004 *
       -0.3946 -0.1129
```

Ha alaposabban megnézzük a minimalizálandó függvényt, akkor ez az eredmény is elfogadható lesz: most a két paraméterünk mindegyikének nullához közelinek kell lennie az optimum közelében. Érdekes a Matlab kiíratási formátuma: nagyon kis számok helyett egyszerűen kiírta, hogy az utolsó sort mivel kell beszorozni... Azért ennél pontosabb megoldást is adhatott volna — valószínűleg az alapbeállítások inkább gyors mint pontos megoldást preferálnak.

Itt a megfelelő pont, amikor az `fminsearch` algoritmusáról is szólnunk kell. A program az ún. Nelder-Mead algoritmuson alapul, más néven a szimplex eljáráson. Ez sajnos könnyen összetéveszthető a lineáris programozás szimplex algoritmusával, bár lényegében nem sok köze van hozzá. Itt az indulópont köré a program egy szimplexet (n-dimenziós térben $n+1$ pontból álló szabályos alakzatot) rajzol. Megvizsgálja a csúcspontokban a célfüggvény értékét, és a legrosszabb célfüggvényértékű pontot tükrözi a többiek által megadott síkra. Így egy újabb szimplexet kapunk, és így tovább. Végül is, durván szólva a célfüggvény által meghatározott domborzaton egy szögletes tárgy gurul le. Más szóval a háttérben működő algoritmus egy elég robusztus, de nem nagyon gyors nemlineáris helyi kereső módszer.

Ennek fényében nagyon szép a következő eredmény:

```
>> fminsearch('x^6*(sin(1/x)+2)',1.2)
ans =
-3.3307e-015
```

Az, hogy szemre hasonló futásidő árán az `fminsearch` sokkal jobb eredményt adott, mint az `fminbnd`, azzal magyarázható, hogy az előbbi bumfordisága most épp előnyös: mivel a futás elején nem tud nagyon finom keresést végezni (még nagyok a szimplexek), ezért az x^6 függvény minimumhelyének eléggé a közelébe tud férkőzni, és nem akad fenn a korai, a globális minimumtól távoli helyi minimumokban. A 12 nagyságrenddel jobb minimum becslés minden esetre említésre méltó.

Ezek után térjünk vissza a korábban tárgyalt Rosenbrock függvényhez:

```
>> fminsearch('(1-x(1))^2+100*(x(1)^2-x(2))^2',[0 0])
ans =
       1.0000    1.0000
```

Bár ezen a feladaton érezhetően tovább gondolkodott (mondjuk 1 másodpercet), de ez már megint szép eredmény, még ha nem is az 1-et, mint egészt mutatja az eredmény (mármint hogy

az 1.000 a lehetséges 1 helyett a hátsó jegyekben megmutatkozó kicsi eltérésre utal).

3. Sebesség, műveletigény, pontosság, gyakorlati alkalmazás.

Ebben a szakaszban jellemezni próbáljuk az optimalizálási eljárásunk sebességét. Ahhoz, hogy ezt meg tudjuk tenni, egy nehéz feladatot kell vizsgálnunk, és a megoldás minőségén is jó ha leolvasható, hogy mennyire jó a talált közelítő megoldás. Ezért vizsgáljuk az előző szakasz $\sin(1/x)$-es függvényét.

A keresési algoritmus alapbeállításai:

```
defaultopt=struct('Display','notify','MaxIter',
       '200*numberOfVariables',...  'MaxFunEvals',
       '200*numberOfVariables','TolX',1e-4,'TolFun',1e-4);
```

- A `Display` csak annyit határoz meg, hogy ha a megfelelő megoldást nem tudja elérni, akkor ennek okát írja meg nekünk.
- A `MaxIter` a megengedett maximális iterációszámot állítja be az optimalizált változók számának 200-szorosára. Ez az esetek többségében elegendő. Ha mégsem, akkor erről értesítést kapunk, és ha van értelme, akkor ezt az algoritmus paramétert nagyobb értékre állíthatjuk.
- A `MaxFunEvals` a megoldáshoz felhasznált függvényhívások számának megengedett felső határát szabja meg. Ennek alapértelmezése is az optimalizált változók számának 200-szorosa.
- Azt, hogy mikor tekintjük a feladatot megoldottnak, a megoldás jóságára vonatkozó két kritérium adja meg. Az első, a `TolX` azt határozza meg, hogy az optimalizálandó változókra vonatkozóan mit tekintünk elfogadhatóan kis eltérésnek. A program megvalósításától függően ezt az algoritmus paramétert különféle módon lehet használni. Egy jellemző módszer, hogy ha az utolsó két (vagy néhány) iterált eltérése ez a korlát alatt marad, akkor ebből a szempontból már elfogadhatónak tekintjük a közelítést.
- Hasonló értelmezésű a `TolFun` algoritmus paraméter. Ez azt üzeni a programnak, hogy akkor kérjük az iteráció leállítását, ha a legutóbbi néhány iteráció során a célfüggvény értéke nem változott többel mint a `TolFun` érték. Az utóbbi két algoritmus paraméterre az alapbeállítás (10^{-4}) mérsékelt igényeket támaszt — ami jó összhangban van a Matlab kiíratási formátumával is.

3.1 Ezek után vizsgáljuk meg, hogy az `fminsearch` milyen hatékony, mennyire gyors az optimalizálásban. Emlékezzünk vissza, hogy az abszolút minimumát a célfüggvényünk a 0 pontban veszi fel, és a célfüggvény érték minimuma is nulla, de az ehhez a minimumhoz tartozó völgy szélessége is nulla — tehát ennek megtalálása reménytelenül nehéz. Másrészt minél közelebbi megoldást kapunk a nullához, annál jobb a közelítés, annál alaposabb a keresés.

A méréseinkhez a következő rövid programot írtuk a Matlab szerkesztőjével (és mentettük el sebesség néven):

```
tic
fminsearch('x^6*(sin(1/x)+2)',1.0,optimset('TolX',1e-2),'TolFun',1e-2)
toc
```

Az előző teszt azt tanulmányozza, hogy milyen hatással van a megoldás helyének pontossága az ennek megkereséséhez szükséges CPU-időre. Az eredményeket az 1. Táblázatban foglaltuk össze.

tolerancia	CPU -idő
10^{-2}	0.18
10^{-4}	0.20
10^{-6}	0.24
10^{-8}	0.33
10^{-10}	0.35
10^{-12}	0.39
10^{-14}	0.44
10^{-16}	0.51
10^{-18}	0.51
10^{-20}	0.53
10^{-30}	0.82

1. Táblázat. A tolerancia értékek hatása az ezek eléréséhez szükséges CPU-időre az $x^6 * (\sin(1/x) + 2)$ optimalizálási feladat esetén a `fminsearch` Matlab programmal.

Megjegyezzük, hogy az első hét esetben a kapott megoldás azonos volt. Az nem is meglepő, hogy ezután a programnak jobban kellett igyekeznie. Megjegyzendő, hogy bár a megoldásnak determinisztikusnak kell lennie, mégis a futási idők változatlan paraméterezés mellett is vadul változtak. Emiatt a táblázatunk 3 független futás átlagát tartalmazza. A futásidő eltérése talán a futtató operációs rendszer egyéb futó processzein múlott. Erre utal az is, hogy a számításigényesebb feladatokon a CPU-idő stabilitása nagyobb volt.

Érdekes, hogy a 10^{-30} mint megállási feltétel paraméter már valóban sok volt a programnak: a táblázatban megadott érték esetén a program panaszkodott, hogy a megengedett maximális függvényhívásszám a feladat megoldásához kevésnek bizonyult. Mégis, a megfelelő paraméter megnövelése sem segített — talán valamilyen belső, rejtett feltétel megakadályozta ennek érvényesülését. Ekkor már a program által adott megoldás `-2.9957e-030` volt, ami nagyon jó — még ha van is ennél is jobb, az adott számítási környezetben ábrázolható megoldás.

Hogy ezt a megoldást jobban megbecsüljük, megmutatjuk, hogy milyen eredményre számíthatnánk egy egyszerű, Monte Carlo módszertől. Tekintsük a következő egyszerű feladatot:

$$\min x^2$$

a $[-1, 1]$ intervallumon. Az ehhez tartozó véletlen kereső program:

```
tic
z = 10
for i=1:100000
     x=2*rand(1)-1;
     y=x^2;
     if (y<z)
       z=y;
       end
end
z
toc
```

A kapott eredmény 3.4373e-011 volt, 3.5700 másodperc alatt. Az `fminsearch` ugyanezen a feladaton -8.8818e-016 eredményt adott 0.1600 másodperc alatt.

3.2. Vizsgáljuk meg az optimalizálás eredményének függését a kiinduló ponttól:

```
tic
fminsearch('x^6*(sin(1/x)+2)',1.0,optimset('TolX',1e-9),'TolFun',1e-9)
toc
```

A kapott eredmények:

indulóérték	eredmény	CPU -idő
1.0e10	-1.4211e-005	2.5440
1.0e05	-1.1369e-009	1.0110
1.0e00	-8.8818e-016	0.0800
1.0e-05	1.0521e-005	0.0300
1.0e-10	1.0500e-010	0.0300

2. Táblázat. Az indulóértékek hatása az ezek eléréséhez szükséges CPU-időre az $x^6 * (\sin(1/x) + 2)$ optimalizálási feladat esetén a `fminsearch` Matlab programmal.

Az első két sorban megadott esetben a Matlab nullával való osztásra panaszkodott, és kevesel-te a megengedett függvényhívások számát. Az utolsó sorokban látható eredmény pedig arra utal, hogy ott tényleges keresés nem is történt. Ez alapján azt mondhatjuk, hogy az `fminsearch` ro-busztus eljárás, nem feltétlen igényli a megoldáshoz közeli indulópontot, bár az előnyös lehet. A számítási idő is ezt tükrözi, még ha a technikai részletekre oda is kell figyelni.

3.3. Végül tekintsünk egy gyakorlati feladatot. A döntéstámogatási feladatkör egyik alap-problémája a páronkénti összehasonlításokon alapuló preferenciamátrixok konzisztenciájának megteremtése. Az alap kérdés az, hogy több szakértő véleményét hogyan lehet egy egységes döntéssé formálni szakmailag meggyőző, elméletileg alátámasztott formában.

Vegyük azt az esetet, hogy adott N darab alternatívánk, pl. vásárolandó autótípusunk. A szakértőink különböző szempontokat figyelembevéve olyan kijelentéseket tesznek, hogy párokat képezve, az egyik alternatíva hányszor tekinthető jobbnak a másiknál. Az eredményeket írjuk egy Q mátrixba:

$$Q = \begin{pmatrix} 1.0000 & 2.0000 & 4.0000 \\ 0.5000 & 1.0000 & 2.0000 \\ 0.2500 & 0.5000 & 1.0000 \end{pmatrix}.$$

Vegyük észre, hogy ebben a példában a megadott preferenciák következetesek, konzisztensek, mert pl. az első alternatívánál a második kétszer kedvezőbb ($Q(1,2) = 2$), a másodiknál a harmadik is kétszer jobb ($Q(2,3) = 2$), és ennek megfelelően a harmadik alternatíva az elsőnél négyszer jobb ($Q(1,3) = 4$). Ez persze általában nem sikerül így, a szakértők által kitöltött preferenciamátrix általában nem konzisztens, tehát a megadott mátrixra nem fog teljesülni az, hogy minden sora konstansszorosa egy másik sorának. A konzisztencia azt is jelenti, hogy a Q mátrix rangja 1, mert egy sorából a többi lineáris kombinációként származtatható.

Az elérni kívánt esetben tehát az egyes alternatívák egymáshoz való viszonyát kimerítően jellemezni tudjuk olyan formában, hogy minden alternatívához hozzárendelünk egy pozitív w_i számot vagy súlyt, és a Q' konzisztens mátrix ebből már generálható:

$$Q' = \begin{pmatrix} 1.0000 & w_2/w_1 & w_3/w_1 \\ w_1/w_2 & 1.0000 & w_3/w_2 \\ w_1/w_3 & w_2/w_3 & 1.000 \end{pmatrix}.$$

Visszatérve a kitűzött feladathoz, azt úgy formalizálhatjuk, hogy egy általában nem konzisztens Q mátrixhoz keresünk olyan pozitív w súlyvektort, hogy

$$||Q - Q'||_F^2 = \sum_{i=1}^{N} \sum_{j=1}^{N} (Q(i,j) - w_j/w_i)^2$$

minimális legyen.

A feladat megoldásához ismét az `fminsearch` programot használtam. A feladatot most egy külön függvény formájában adtam meg:

```
function f = qw(w);
Q = [1 2 4; 0.5 1 2; 0.25 0.5 1];
f = 0;
for i=1:3
f = f+(Q(i,1)-w(1)/w(i))^2+(Q(i,2)-w(2)/w(i))^2+(Q(i,2)-w(3)/w(i))^2;
end
```

A programon mindenképpen lehet még javítani, például a Q mátrixot illene kivinni a hívó programba, illetve a vektorizáláson is lehet még igazítani... Minden esetre a jelen formában is szépen futott a program, és egy másodpercen belül a következő eredményt kaptuk:

```
>> fminsearch('qw',[1 1 1])
ans =
     0.4215 0.8430 1.6859
```

Ez nem egészen az, amit vártunk (az pl. az [1 2 4]' vektor lehetett volna), de lényegében rendben van, mert a harmadik alternatíva négyszer preferáltabb, mint az első stb. Vegyük észre azonban, hogy maga a program teljesen érzéketlen a w vektor abszolút értékére, csak a vektorkomponensek arányait érzékeli. Ha ezt a vektort visszaírjük az optimalizálandó nem

negatív függvénybe, akkor a 7.1458e-008 függvényértéket kapjuk, ami elég jó érték, tekintve hogy a paramétereket csak 4-5 értékes jegyre adtuk meg.

Ezt az is mutatja, hogy ha az előző feladatot a `w = [1 2 4]'` vektorral indítjuk, akkor az eredmény már megnyugtatóbb:

```
>> fminsearch('qw',[1 2 4])
ans =
     1 2 4
```

Vegyük észre, hogy nem egyszerűen közeli értékeket kaptunk, hanem magukat az egészszámokat (nem volt nehéz ezekre jutni). Ez persze azt is jelenti, hogy akkor indokolatlanul sok változó optimalizálását kértük a programtól. hiszen ha csak w arányai számítanak, akkor a w egyik komponensét rögzíthetjük. Említsük meg azt is, hogy bár nem eröltettük, de a program csak pozitív eredményt adott — erre vonatkozó ténylegesen megadott korlát nélkül is. Ez a probléma szerkezetéből adódhat.

Ezek után tekintsük a feladat perturbált, kicsit elrontott változatát. Vegyük azt az egyszerű esetet, amikor a Q mátrix jobb felső sarkában levő négyes rontjuk csak el. Ennek helyére írjunk `4+0.01` értéket, ami egy század mértékben perturbálja a korábbi számot (megpróbáltam ez utóbbit még `rand(1)`-el beszorozni, de a véletlen eltérések áttekinthetetlenné tették az eredményeket). A változó mértékű perturbálásra vonatkozó eredményeket a 3. Táblázat tartalmazza:

preturbáció	az eredmény
1.0e-2	$[1.0074, 2.0170, 4.0385]^T$
1.0e-3	$[1.0023, 2.0048, 4.0100]^T$
1.0e-4	$[1.0007, 2.0015, 4.0030]^T$
1.0e-5	$[1.0007, 2.0015, 4.0029]^T$
1.0e-6	$[1.0002, 2.0004, 4.0009]^T$
1.0e-7	$[1.0001, 2.0003, 4.0006]^T$
1.0e-8	$[1.0001, 2.0002, 4.0004]^T$
1.0e-9	$[1.0000, 2.0001, 4.0002]^T$
1.0e-10	$[1.0000, 2.0000, 4.0001]^T$
1.0e-11	$[1, 2, 4]^T$

3. Táblázat. A döntéstámogatási preferenciamátrix konzisztensé tétele optimalizálási feladata megoldása az alkalmazott perturbáció függvényében. Figyeljük meg a pontos megoldástól való eltérés csökkenését a perturbáció csökkenése függvényében.

Megállapíthatjuk, hogy a kapott eredmények azt tükrözik, hogy a perturbáció csökkenésével a várt, pontos eredmény jól megközelíthető volt, vagyis a vizsgált feladaton az inkonzisztens preferenciamátrix konzisztensé tétele numerikus szempontból megbízható eredményt ad.

Megjegyzendő, hogy a fenti vizsgálatokat csak a Matlab 6.5 oktatói változattal sikerült megcsinálni, a korábbi, 5.0-ás hallgatói változat nem ismerte fel az itt leírt optimalizálási programok neveit (ez a vizsgálat a műveletigény megállapításához kellett volna, mert ott még működik a `flops` rutin). Ennek ellenére a Matlab 6.5-ös hallgatói demováltozat valószínűleg eléri ezeket a

szubrutinokat.

Összefoglalva, a Matlab által az alapcsomagban kínált optimalizálási eljárások használhatóak, több esetben ügyesek voltak, ennek ellenére érdemes tapasztalatot gyűjteni mielőtt komoly, nehezebb feladat megoldásához kezdünk velük.

4. Irodalom

A területtel való alaposabb megismerkedéshez a következő könyveket, dokumentumokat ajánlom:

1. Csendes Tibor: Bevezetés a globális optimalizálásba. Jegyzet előkészületben. Elérhető a `www.inf.u-szeged.hu/~csendes/go.ps.gz` internetes címen.

2. Higham, Desmond J. and Nicholas J. Highham: MATLAB Guide. SIAM, Philadelphia, 2000.

3. Neumaier, Arnold internetes vendégoldala a globális optimalizálásról. Elérhető a következő címen: `www.mat.univie.ac.at/~neum`

4. Stoyan Gisbert (szerk.): MATLAB (4. és 5. verzió). TypoTEX Kiadó, Budapest, 1999.

Tartalomjegyzék