



Tudományos és szimbolikus számítások, optimalizálás alkalmazásai

című tananyag

Dr. Csendes Tibor — Dr. Vinkó Tamás

Szegedi Tudományegyetem Természettudományi és Informatika Kar

Informatikai Intézet Számítógépes Optimalizálás Tanszék

2018

A tananyag az EFOP-3.5.1-16-2017-00004 pályázat támogatásával készült

Tudományos és szimbolikus számítások, optimalizálás alkalmazásai

Csendes Tibor — Vinkó Tamás

2018. szeptember 30

Tartalomjegyzék

Jelölések	8
Bevezetés az I. részhez	10
1. Aritmetika	13
1.1. Adatábrázolás	13
1.2. Aritmetikai algoritmusok	13
1.3. Néhány érdekes megvalósítás	15
1.4. Legnagyobb közös osztó	16
1.4.1. Algoritmus egész számokra	16
1.4.2. Algoritmus polinomokra	17
1.4.3. Kiterjesztett euklideszi algoritmus	17
1.4.4. Euklideszi algoritmus költsége	18
1.5. További megjegyzések	18
1.6. Feladatok	19
2. Nagypontosságú aritmetika	20
2.1. Kongruenciák	20
2.2. Kínai maradéktétel	21
2.3. Nagyméretű lineáris egyenletrendszerek pontos megoldása	24
2.4. Ismételt négyzetre emelés	25
2.5. Néhány érdekes probléma	26
2.6. Feladatok	27
3. Gyors aritmetika	28
3.1. Nagy számok szorzása	28
3.1.1. Implementáció	29
3.1.2. Karacuba a gyakorlatban	30
3.2. Nagy mátrixok szorzása	30
3.3. Mátrixok invertálása	31
3.4. Számok osztása	32
3.5. Barrett redukció	33
3.6. Feladatok	34
4. Műveletek polinomokkal	36
4.1. Alapfogalmak	36
4.2. Polinomok kiértékelése - Horner módszer	36
4.3. Polinomok kiértékelése - prekondicionálással	36
4.4. Tabellázás	38

4.5.	Polinomok szorzása	39
4.5.1.	Véges Fourier-transzformált	40
4.6.	Feladatok	42
5.	Polinomok gyökeiről	44
5.1.	Sturm tétel	45
5.2.	További korlátok polinomokra	46
5.3.	Laguerre módszer	47
5.4.	Rezultánsok	48
5.5.	Gröbner bázisok	49
5.6.	Feladatok	50
6.	Prímtesztelés	52
6.1.	Alapfogalmak	52
6.2.	Szita módszer	52
6.3.	Fermat kis tétele	53
6.3.1.	Álprímek	53
6.3.2.	Algoritmus a Fermat kis tétele alapján	53
6.3.3.	Carmichael számok	54
6.4.	Miller-Rabin teszt	55
6.5.	Mersenne prímek	56
6.6.	AKS algoritmus	57
6.7.	Nagyméretű prímszámok generálása	57
6.8.	Feladatok	58
7.	Prímfaktorizálás	60
7.1.	Legnagyobb prímtényező	60
7.2.	Prímtényezők száma	60
7.3.	Prímfelbontás	61
7.3.1.	Próbálgatásos osztás	61
7.3.2.	Pollard-féle Monte Carlo módszer	62
7.3.3.	Fermat-faktorizáció	64
7.3.4.	Euler-faktorizáció	64
7.4.	Faktorizáló algoritmusokról	65
7.5.	További megjegyzések	66
7.5.1.	Szemiprímek	66
7.5.2.	Ikerprímek	66
7.6.	Feladatok	66
8.	Titkosítás, véletlenszámok	68
8.1.	Titkosítás	68
8.1.1.	Cézár-rejtjel	68
8.1.2.	One-time pad	68
8.2.	Véletlenszám generálás	70
8.2.1.	Véletlenszámok előállítása külső eszközök felhasználásával	70
8.2.2.	Neumann négyzetközép	71
8.2.3.	Lineáris kongruenciák	71
8.2.4.	Mersenne twister	73
8.2.5.	Statisztikai próbák	73
8.2.6.	További megjegyzések	74

8.3. Feladatok	74
9. Polinomok faktorizálása	76
9.1. Felbontás modulo p	76
9.1.1. Berlekamp algoritmus	79
9.1.2. Példa	80
9.2. Feladatok	82
10. Programkönyvtárak	84
10.1. BLAS	84
10.1.1. Változatok	85
10.2. GNU könyvtárak	85
10.2.1. GSL	85
10.2.2. GMP	85
10.2.3. MPFR	86
10.3. További függvénykönyvtárak	86
10.3.1. MPFQ	86
10.3.2. FFTW	86
10.3.3. NTL	86
10.3.4. FLINT	87
10.3.5. Boost	87
10.3.6. CLN	87
10.4. Scipy és Numpy	88
10.5. Rendszerek	88
10.5.1. PARI/GP	88
10.5.2. SageMath	88
10.6. Online anyagok	89
10.6.1. NIST DLMF	89
10.6.2. Wolfram Functions	89
10.6.3. ESF, DDMF	89
10.6.4. Interval Computations	90
10.6.5. OEIS	90
10.7. Feladatok	90
Előszó a II. részhez	92
11. Alapfogalmak	93
11.1. Intervallum matematika	94
11.1.1. Intervallum-aritmetika és a befoglaló függvények	94
11.1.2. Intervallum-felosztási algoritmus	98
11.1.3. Intervallumos Newton módszer	99
11.1.4. Példák	100
11.2. Automatikus differenciálás	102
11.2.1. Deriváltak a számítógépeken	102
11.2.2. Az ötlet.	104
11.2.3. Kiterjesztések.	104
11.2.4. Az automatikus differenciálás két változata.	105
11.2.5. Művelet- és tárigény.	106
11.2.6. Az automatikus differenciálás veszélyei.	106
11.2.7. Az automatikus differenciálás implementálása.	107

11.3. Ellenőrző kérdések és gyakorló feladatok	108
12. A hátizsák feladat	111
12.1. A hátizsák feladat megoldása teljes leszámolással	111
12.2. A hátizsák feladat megoldása közvetett, implicit leszámolással	112
12.3. Kapcsolódó feladatok	114
12.4. Ellenőrző kérdések és gyakorló feladatok	115
13. Az utazó ügynök feladat	118
13.1. Az utazó ügynök feladat heurisztikus algoritmusai	122
13.1.1. A távolságvektor szerepe a heurisztikákban	123
13.1.2. A heurisztikus algoritmusok hatásossága vizsgálata	124
13.2. Az utazó ügynök feladat megoldása Matlabbal	126
13.3. Ellenőrző kérdések és gyakorló feladatok	127
14. A szabási feladat	129
14.1. Az oszlogenerálás módszere a szabási feladatra	130
14.2. Heurisztikák a szabási feladatra	133
14.3. Ellenőrző kérdések és gyakorló feladatok	135
15. Hálózati problémák	136
15.1. A legrövidebb út probléma	136
15.2. A maximális folyam probléma	137
15.3. A Ford-Fulkerson eljárás a maximális folyam meghatározására	138
15.4. Projektek ütemezése, CPM	141
15.5. Program kiértékelés és felülvizsgálat, PERT	146
15.6. Ellenőrző kérdések és gyakorló feladatok	149
16. Sztochasztikus programozás	151
16.1. Az újságárus probléma	152
16.2. A folytonos keresletű újságárus probléma	155
16.3. Ellenőrző kérdések és gyakorló feladatok	156
17. Gradiens módszer	157
17.1. Konjugált gradiens módszer	159
17.2. Ellenőrző kérdések és gyakorló feladatok	162
18. A korlátozás és szétválasztás módszere	164
18.1. Körpakolási feladatok	167
Tárgymutató	169
Magyar-angol szószedet	169
Függelékek	171
A. Tematika	171
B. Mintafeladatok a dolgozatokhoz	173
B.1. A tudásfelmérő dolgozat feltételei	175

C. Bevezetés a MATLAB használatába	177
Aritmetikai műveletek és függvények	177
Műveletek mátrixokkal	179
Megjelenítés	181
Programok, ciklusok, vezérlés	182
D. Az esszé, illetve kötelező program követelményei, témák	186
E. Egy minta esszé	191
F. Az előadás fóliái	206
1. óra	206
2. óra	216
3. óra	225
4. óra	233
5. óra	241
6. óra	251
7. óra	260
8. óra	268
9. óra	274
10. óra	280
11. óra	288
12. óra	293
G. Kiegészítő szorgalmi feladatok	301
H. A 2002-es SIAM numerikus verseny feladatai	303
I. A standard normális eloszlás táblázata	305
J. Kötelező program témák	306
Irodalomjegyzék	311

A tananyag az EFOP-3.5.1-16-2017-00004 pályázat támogatásával készült.

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
$\mathbb{R}$	a valós számok halmaza
$\mathbb{R}^n$	az n -dimenziós valós vektorok halmaza
$x, y, \dots$	változók
$X, Y, \dots, A, B, \dots$	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

I. rész

TUDOMÁNYOS ÉS SZIMBOLIKUS SZÁMÍTÁSOK

Bevezetés az I. részhez

Kezdjük egy egyszerűen tűnő kérdéssel, mit jelent a következő kifejezés:

42

A kérdés persze nem pontos, talán inkább az lenne a jobban idevágó változat, hogy milyen (programozási nyelvekből jól ismert) típusú a kifejezés. Az esetek többségében a válasz: 'egész szám'. Lehet azonban racionális szám, valós szám, egy egész szám modulo m (ahol $m > 42$), stb.

Miért fontos ezt lerögzítenünk? Egy halmaz elemein végzett művelet függhet attól, hogy mi az alaphalmaz. Például $42/13$ nem definiált az egészek halmazán, viszont igen a valós számokon. Valamint számít a számítógépes reprezentáció is:

- a diszkrét halmazokat általában teljes pontossággal tudjuk ábrázolni,
- míg a folytonosakat csak közelítőleg vagy implicit módon (szimbolikusan).

A hagyományos numerikus kurzussal ellentétben, ahol lebegőpontos számokkal végeztünk közelítő számításokat, itt leginkább kifejezésekre, algebrai objektumokon végrehajtott eljárásokra koncentrálnunk.

Tipikus algebrai számítások Bár a jegyzet az itt felsorolt témakörök csak egy részét fedi le, a következők tartoznak ide: tetszőleges pontosságú aritmetika, nincs kerekítés, szimbolumokkal és változókkal vegyesen számolunk, (matematikai értelemben vett) függvényekkel számolunk, kifejezések átalakítása (lehetőleg automatikusan), és szimbolikus műveletvégzés.

Matematikai objektumok ábrázolása

A következő szinteket különböztetjük meg:

1. szint: az objektumok szintje, ahol ha matematikai értelemben azonosak a kifejezések, akkor nincs megkülönböztetés
 - $2 + 2$ és a $3 \cdot 3 - 5$ ugyanaz az objektum (számokon)
 - $(x + 1)^2(x - 1)$ és $x^3 + x^2 - x - 1$ ugyanaz (polinomokon)
2. szint: forma szintje, ahol eltérő ábrázolást megkülönböztetjük
3. szint: adatstruktúra szintje, ahol pedig a memória-tárolásban eltérő ábrázolásokat tekintjük különbözőeknek
 - például az $x^3 + x^2 - x - 1$ polinom esetén ez történhet az együtthatók tömbjével, vagy láncolt listájával

Alapműveletek végrehajtása

Az aritmetikai algoritmusok implementációja és végrehajtása alapvetően két, egymástól különböző kontextusban történhet: numerikusan vagy szimbolikusan. Az alábbi táblázat néhány érdekes aspektust mutat arra vonatkozóan, hogy ez a két megvalósítás valóban különbözik egymástól:

	numerikusan	szimbolikusan
költség	olcsó, általában pontatlan	drága, mindig pontos
válasz	gyakran ad is valamilyen választ	gyakran nem ad választ
zérusok	a zérusok nem speciálisak	a zérusok gyakran egyszerűsítene
zérus detektálás	elég pici a szám?	a kifejezés ekvivalens a 0-val?

Számítási tárrobbanás

Az algebrai algoritmusoknál fontos a tárigény vizsgálata is. Számítási tárrobbanás *expression swell* Ennek illusztrációjaként számítsuk ki egy 9 dimenziós Hankel mátrix sajátértékeit.

$$H_{i,j} = H_{i-1,j+1}$$

(a numerikus kurzuson megismert Hilbert mátrix ennek egy speciális esete)

A mátrix 9 dimenzióban így néz ki:

$$\begin{pmatrix} -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 \end{pmatrix}$$

A karakterisztikus polinomja, amelynek gyökei a mátrix sajátértékei:

$$\lambda^9 + \lambda^8 - 40\lambda^7 - 24\lambda^6 + 240\lambda^5 + 144\lambda^4,$$

ami átírható szorzat alakban:

$$\lambda^4(\lambda + 6)(\lambda^4 - 5\lambda^3 - 10\lambda^2 + 36\lambda + 24)$$

ami alapján megállapíthatjuk, hogy a 0 négyszeres gyöke, -6 egyszeres gyöke, valamint van még egy negyedfokú tag, ami további 4 sajátértéket eredményez. A negyedfokú polinomok gyökeit meg tudjuk határozni, hiszen van rá megoldóképlet. Az egyik gyök a négy közül:

$$\frac{5/4 + 1/12 \sqrt{3} \sqrt{\frac{8 \left(3142 + 18 i \sqrt{8071} \right)^{2/3} + 155 \sqrt[3]{3142 + 18 i \sqrt{8071}} - 1856}{\sqrt[3]{3142 + 18 i \sqrt{8071}}}}}{\sqrt[3]{\frac{8 \left(3142 + 18 i \sqrt{8071} \right)^{2/3} + 155 \sqrt[3]{3142 + 18 i \sqrt{8071}} + 1856}{\sqrt[3]{3142 + 18 i \sqrt{8071}}}} - 666 \sqrt{3} \sqrt[3]{3142 + 18 i \sqrt{8071}} - 930 \sqrt[3]{3142 + 18 i \sqrt{8071}} \sqrt{\frac{8 \left(3142 + 18 i \sqrt{8071} \right)^{2/3} + 155 \sqrt[3]{3142 + 18 i \sqrt{8071}} + 1856}{\sqrt[3]{3142 + 18 i \sqrt{8071}}}}} \sqrt[3]{3142 + 18 i \sqrt{8071}} \sqrt{\frac{8 \left(3142 + 18 i \sqrt{8071} \right)^{2/3} + 155 \sqrt[3]{3142 + 18 i \sqrt{8071}} + 1856}{\sqrt[3]{3142 + 18 i \sqrt{8071}}}}}$$

ahol bosszantó módon még a legjobb igyekezetünk ellenére sem tudjuk úgy megjeleníteni a képletet, hogy az teljes egészében látszódjon. Nyilvánvalóan ha numerikus megoldást keresünk, akkor nincs ilyen probléma.

Heurisztikák, valószínűségi módszerek

A jegyzetben több olyan algoritmust is bemutatunk majd, amelyek valószínűségi alapon adnak választ. Ennek motivációja a szokásos, a tárigényt és a futási időt szeretnénk csökkenteni. Az ilyen típusú eljárásokat alapvetően két csoportba osztjuk:

1. Monte Carlo módszerek:

- a rossz válasz pozitív valószínűségű

2. Las Vegas módszerek:

- soha nincs rossz válasz, de lehet, hogy nempolinomiális ideig fut

1. fejezet

Aritmetika

1.1. Adatábrázolás

Egész számok Az egy gépi szóban ábrázolható számokat egyszeres pontosságú egészeknek nevezzük. Ezek helyiértékes ábrázolása:

$$n = \sum_{i=0}^{k-1} d_i B^i \quad B \in \mathbb{Z}, B > 1, d_i \in \mathbb{Z}$$

alakú, amelyet praktikusán így tároljuk: $[d_0 d_1 \dots d_{k-1}]$. Ez a tárolás történhet például láncolt listában vagy csak simán egy tömbben.

Az egész számok másik lehetséges ábrázolása a moduláris ábrázolás, ahol egyszeres pontosságú, páronként relatív prímek moduláris képeként ugyancsak láncolt listában vagy tömbben tárolunk. Erről részletesebben a 2. fejezetben lesz szó.

Az adatstruktúra megválasztása befolyásolja az aritmetikai algoritmusok sebességét:

- az összeadás, kivonás és szorzás gyorsan megy a moduláris aritmetikában – ahogyan ezt majd látni fogjuk a 3. fejezetben;
- az oszthatósági kérdések azonban a helyiértékes ábrázolás esetén végezhetőek el hatékonyabban.

Racionális számok A racionális számokat egész számok párjainak tekintjük: számláló és nevező formában tároljuk. Célszerű a legnagyobb közös osztóval (a legnagyobb közös osztóról a fejezet végén lesz szó) egyszerűsített ábrázolást választani. A szám előjele a számláló előjele.

Polinomok A fentiek alapján itt elsősorban racionális együtthatós polinomokról van szó, amelyek azonban lehetnek többváltozósak is. Kétféle ábrázolás használatos:

- faktorizált ábrázolás, pl $(x^2 + 1)(y - 1)$
- vagy kiterjesztett ábrázolás, pl $x^2 y - x^2 + y - 1$

1.2. Aritmetikai algoritmusok

Ebben a szakaszban az aritmetikai műveletek algoritmikus végrehajtását tárgyaljuk. Részletesen csak az egész számokra vonatkozó műveleteket nézzük meg, majd röviden kitérünk a polinomokra vonatkozó algoritmusokra is.

Egész számokra Lényegében mintha kézzel számolnánk, csak a számrendszer alapja lehet más.

- Az összeadás műveletigénye $\mathcal{O}(n)$.
- A szorzás $\mathcal{O}(nm)$ műveletigényű. A 3. fejezetben látjuk majd, ezt be lehet gyorsítani. Megjegyezzük továbbá, hogy speciális esetekre is léteznek megoldások, mint például
 - a két szám közül az egyik sokkal kisebb, mint a másik
 - négyzetre emelés
 - konstanssal történő szorzás (amikor ugyanazzal a konstans taggal több alkalommal is szorzunk)
- Az osztás pedig $\mathcal{O}(n(m+1))$ műveletigényű. Megjegyezzük, hogy az eredmény a két egész hányadosa és a maradék együtt. Érdekes módon sok programozási nyelvben ez a két művelet két külön utasítás.

Érdemes még megjegyezni, hogy egy N egész számra a gyökvonást az $\lfloor \sqrt{N} \rfloor$ alakban értelmezzük, tehát az a legnagyobb olyan egész szám, amely kisebb vagy egyenlő, mint N négyzetgyöke.

Továbbá fontos művelet még a számrendszer átváltó (például tízesből kettesbe).

Polinomokra Polinomok esetében az összeadás és szorzás lényegében ugyanúgy megy, mint egész számok esetén, csak a helyiértékek szerepét itt a hatványtagok veszik át, ráadásul az átvitelre nem kell ügyelni.

Egy $p(x)$ polinom fokszámát a $\deg(p)$ jelöli. Két (racionális együtthatós) $p(x)$ és $q(x)$ polinomra, ahol $\deg(p) \geq \deg(q)$ definiáljuk a $\text{quot}(p, q)$ hányadost és $\text{rem}(p, q)$ maradékot úgy, hogy

$$p(x) = \text{quot}(p, q) \cdot q + \text{rem}(p, q),$$

ahol $0 \leq \deg(\text{rem}(p, q)) < \deg(q)$.

Kapcsolat a polinomok és az egész számok szorzása között Tegyük fel, hogy össze akarunk szorozni a $p(x)$ és $q(x)$ nem-negatív együtthatós polinomokat, amelyek n -nél kisebb fokszámúak, és az együtthatóik felső korlátja K . Vegyük most $X = \beta^k > nK^2$ a β alapra és szorozzuk össze a $a = p(X)$ és $b = q(X)$ egészeket, amelyeket a p és q polinomok $x = X$ helyén vett kiértékeléssel kapunk. Ha most $r(x) = p(x)q(x) = \sum c_i x^i$, akkor világos, hogy $r(X) = \sum c_i X^i$. Mivel $c_i < nK^2 < X$, ezért a c_i együtthatók leolvashatóak az $r(X)$ -ből. Példaként tekintsük a

$$(6x^5 + 6x^4 + 4x^3 + 9x^2 + x + 3)(7x^4 + x^3 + 2x^2 + x + 7)$$

szorzást, ahol tehát a fokszám $n = 6$ -nál kisebb és az együtthatók felső korlátja $K = 9$. Vehetjük most $X = 10^3 > nK^2$, és végrehajtjuk az egész szorzást

$$6\ 006\ 004\ 009\ 001\ 003 \times 7\ 001\ 002\ 001\ 007 = 42\ 048\ 046\ 085\ 072\ 086\ 042\ 070\ 010\ 021,$$

ahonnan leolvashatjuk, hogy

$$42x^9 + 48x^8 + 46x^7 + 85x^6 + 72x^5 + 86x^4 + 42x^3 + 70x^2 + 10x + 21$$

a végeredmény. Megfordítva, tegyük fel, hogy össze akarunk szorozni két egészet $a = \sum_{0 \leq i < n} a_i \beta_i$ és $b = \sum_{0 \leq j < n} b_j \beta_j$. Végezzük el a polinom szorzást a $p(x) = \sum_{0 \leq i < n} a_i x_i$ és $q(x) = \sum_{0 \leq j < n} b_j x_j$ polinomokkal, amelynek az eredménye legyen $r(x)$. Ezután számoljuk ki az $r(\beta)$ helyettesítési értéket, hogy megkapjuk ab értékét. Megjegyezzük, hogy az $r(x)$ együtthatói nagyobbak is lehetnek, mint β , sőt akár $n\beta^2$ értékig is mehet. Például $a = 123$, $b = 456$, és $\beta = 10$, kapjuk, hogy $p(x) = x^2 + 2x + 3$, $q(x) = 4x^2 + 5x + 6$, és a szorzatuk $r(x) = 4x^4 + 13x^3 + 28x^2 + 27x + 18$, és $r(10) = 56088$. Ezek a példák tehát jól rávilágítanak a polinomokkal és az egészekkel végzett műveletek analógiájára, valamint mutatják annak korlátait is.

1.3. Néhány érdekes megvalósítás

Ebben a szakaszban bemutatunk néhány, az alpműveletekre vonatkozó érdekes implementációt. A példák tekinthetők programozási gyakorlatnak is, nagyon tanulságosak.

Két előjel nélküli egész szám szorzása. A feladat természetesen úgy érdekes, hogy nincs megengedve az összeadás használata. Ahelyett: bitműveletek a megengedettek, és természetesen lehet ciklus utasításokat használni.

```
multiply_no_operator(x, y){
    sum = 0;
    while (x){
        if (x & 1) sum = add_no_operator(sum, y);
        x >>= 1; y <<= 1;
    }
    return sum;
}

add_no_operator(a, b){
    sum = 0;
    carryin = 0;
    k = 1;
    temp_a = a; temp_b = b;
    while (temp_a || temp_b){
        ak = a & k;
        bk = b & k;
        carryout = (ak & bk) | (ak & carryin) | (bk & carryin);
        sum |= (ak ^ bk ^ carryin);
        carryin = carryout << 1;
        k <<= 1;
        temp_a >>= 1;
        temp_b >>= 1;
    }
    return sum | carryin;
}
```

Két egész szám osztása. Számítsuk ki az x/y értékét úgy, hogy csak összeadás, kivonás és eltolás műveleteket használhatunk.

Az ötlet, hogy használjuk az alábbi rekurziót:

$$\frac{x}{y} = \begin{cases} 0, & \text{ha } x < y, \\ 1 + \frac{x-y}{y} & \text{különben.} \end{cases}$$

Ez nagyon hasznos képlet, viszont rekurzív hívásként implementálva, amennyiben a két szám különbsége nagyon nagy, akkor túl sok lépést végez az algoritmus, és könnyen betelhet a verem. Ezért érdekesebb a

$$\frac{x}{y} = \begin{cases} 0, & \text{ha } x < y, \\ 2^k + \frac{x-2^k y}{y} & \text{különben} \end{cases}$$

képletet használni, ahol k az a legnagyobb érték, amelyre $2^k y \leq x$. Ilyenkor a $2^k y$ értéke számolható y szám k darab balra léptetésével.

Szorzás és kivonás csak összeadással. A feladatunk, hogy írjunk eljárást a szorzás és a kivonás műveletekre úgy, hogy csak az összeadás operátort használhatjuk.

A kivonás viszonylag egyszerű, ha kihasználjuk, hogy $a - b = a + (-1)b$, és alkalmazzuk a következő eljárást, amely egy a számot negál:

```
neg = 0
d = a < 0 ? 1 : -1
while (a != 0) {
    neg += d
    a += d
}
return neg
```

Vegyük észre, hogy itt arról van szó, hogy kihasználjuk a túlcsondulást, amikor egy pozitív számból negatív lesz.

A szorzáshoz pedig ismételt összeadást használhatunk, amely persze lassú, de működik.

1.4. Legnagyobb közös osztó

A négy alapl művelet mellett talán az egyik legfontosabb művelet a legnagyobb közös osztó. A jegyzetben elsősorban egész számokra használjuk majd, de a fogalom természetesen értelmezhető (egyváltozós) polinomokra is: ebben az esetben a legmagasabb fokszámú közös osztót keressük.

Jelölése: $d = (a, b)$, vagy $d = \text{lko}(a, b)$, valamint használatos még a $d = \text{gcd}(a, b)$ jelölés is (greatest common divisor).

Ha $d = 1$, akkor azt mondjuk, hogy a és b relatív prímek.

Kapcsolódó fogalom: legkisebb közös többszörös. Ennek jelölése és kiszámítása: $[a, b] = ab / (a, b)$.

1.4.1. Algoritmus egész számokra

Lássuk a több, mint 2000 éves algoritmust. Legyen az input két egész szám: $a_0 > a_1$. Számítsuk ki a következő maradékos osztás sorozatot:

$$\begin{aligned} a_0 &= q_1 a_1 + a_2 & (a_2 < a_1) \\ a_1 &= q_2 a_2 + a_3 & (a_3 < a_2) \\ &\vdots \\ a_{k-1} &= q_k a_k + a_{k+1} & (a_{k+1} < a_k). \end{aligned}$$

Amennyiben $a_{k+1} = 0$ teljesül, akkor álljunk meg.

Fontos megjegyzés, hogy a $(q_1, \dots, q_k, a_k)$ sorozatot az a_0, a_1 *hányadosvektorának* nevezzük.

Tömörebb pszeudokód változat (itt is fölteszük, hogy $a > b$):

```
while (b != 0) {
    r = rem(a, b)
    a = b
    b = r
}
return a
```

A fenti kódban használt $\text{rem}(a, b)$ eljárás az a -nak a b -vel vett osztásának maradékát adja vissza.

1.4.2. Algoritmus polinomokra

Az euklideszi algoritmus természetesen működik polinomok esetében is. Ugyanúgy, mint az egész számokra, itt is teljesül, hogy

$$\text{lko}(p, q) = \text{lko}(q, \text{rem}(p, q))$$

A pszeudokód egyébként pontosan megegyezik az egész számokra megadott kóddal a kiegészítéssel, hogy a feltétel ebben az esetben a fokszámra vonatkozik: $\deg(a) > \deg(b)$.

Példa. Határozzuk meg a

$$p(x) = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x + 5$$

és

$$q(x) = 3x^6 + 5x^4 - 4x^2 - 9x + 21$$

polinomok legnagyobb közös osztóját. A következőkben csak az r maradékpolinomokat adjuk meg, amelyek tehát

$$\begin{aligned} r_1 &= -\frac{5}{9}x^4 + \frac{1}{9}x^2 - \frac{1}{3} \\ r_2 &= -\frac{117}{25}x^2 - 9x + \frac{441}{25} \\ r_3 &= \frac{233150}{6591}x - \frac{102500}{2197} \\ r_4 &= \frac{1288744821}{543589225} \end{aligned}$$

A két megadott polinom tehát relatív prím.

1.4.3. Kiterjesztett euklideszi algoritmus

A fentiekben bevezetett hányadosvektor felhasználható arra, hogy a kiterjesztett euklideszi algoritmust definiáljuk, amelynek végrehajtásával az

$$\text{lko}(a_0, a_1) = v_0 \cdot a_0 + v_1 \cdot a_1$$

alakot tudjuk előállítani. Vegyük észre, hogy itt a legnagyobb közös osztót, mint a két szám lineáris kombinációját írjuk fel. Megjegyezzük továbbá, hogy a v_0 és v_1 értékeket a moduláris algoritmusoknál (a 2. fejezetben) majd még felhasználjuk. Az algoritmus megadásához definiáljuk a következő mátrixot:

$$M_i = \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix}.$$

Ekkor felírhatjuk a következő összefüggést:

$$\begin{pmatrix} a_i \\ a_{i+1} \end{pmatrix} = M_i \cdot \begin{pmatrix} a_{i-1} \\ a_i \end{pmatrix}.$$

Innen kapjuk, hogy

$$\begin{pmatrix} a_i \\ a_{i+1} \end{pmatrix} = M_i \cdot M_{i-1} \cdot \dots \cdot M_1 \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}.$$

Az $M_i \cdot M_{i-1} \cdot \dots \cdot M_1$ mátrix szorzat a $q_1, \dots, q_i$ sorozat meghatározásával együtt számolható. Legyen most

$$M_i \cdot M_{i-1} \cdot \dots \cdot M_1 = \begin{pmatrix} u_{11}^{(i)} & u_{12}^{(i)} \\ u_{21}^{(i)} & u_{22}^{(i)} \end{pmatrix}.$$

Ekkor

$$a_k = u_{11}^{(k)} a_0 + u_{12}^{(k)} a_1,$$

tehát lényegében a v_0 és v_1 értékek egyfajta melléktermékek, amelyekből tehát előáll az

$$\text{luko}(a_0, a_1) = v_0 \cdot a_0 + v_1 \cdot a_1$$

alak, amit kerestünk.

1.4.4. Euklideszi algoritmus költsége

Azt, hogy két egész relatív prím, polinom időben el tudjuk dönteni. Legrosszabb esetben n darab osztás kell (lásd lentebb). Hagyományosan egy osztás $\mathcal{O}(n^2)$ -be kerül, ezért a teljes költség $\mathcal{O}(n^3)$ lesz. Ugyanakkor lehet gyorsabban is osztani (ahogy azt tanuljuk is majd később), akkor $\mathcal{O}(n^2 \log n)$ a költség.

Végezetül megjegyezzük, hogy amennyiben ismerjük a prímfelbontásokat, akkor könnyű a legnagyobb közös osztó kiszámítása.

1.5. További megjegyzések

Legrosszabb eset. Lamé tétele 1845-ből: legyenek $n \geq 1$, és $u > v > 0$ olyan pozitív egészek, hogy az u -ra és v -re elvégzett euklideszi algoritmus pontosan n osztási lépést tartalmaz, továbbá u ezen belül a lehető legkisebb. Ekkor $u = F_{n+2}$, $v = F_{n+1}$, amely nem más, mint két egymást követő Fibonacci szám. A Fibonacci számok rekurzív definíciója: $F_0 = 0$, $F_1 = 1$ és $F_n = F_{n-1} + F_{n-2}$.

Több szám legnagyobb közös osztója. Ha az $u_1, \dots, u_n$ egész számokra akarjuk a legnagyobb közös osztót kiszámítani, akkor a következő rekurziót kell alkalmaznunk:

$$\text{luko}(u_1, \dots, u_n) = \text{luko}(u_1, \text{luko}(u_2, \dots, u_n))$$

Relatív prímek gyakorisága. Dirichlet tétele ad egy benyomást a relatív prímek előfordulásának gyakoriságáról. Ha u és v véletlen egészek, akkor annak a valószínűsége, hogy $\text{luko}(u, v) = 1$ az

$$\frac{6}{\pi^2} \approx 0.60793.$$

Ez az eredmény azért releváns, mert amikor több szám legnagyobb közös osztóját számítjuk, akkor ezen eredmény szerint gyakran gyorsan végezhetünk (azzal az eredménnyel, hogy 1 a keresett érték).

Bináris luko. Az algoritmusnak létezik egy változata, amelyben nincs szükség osztás műveletre, hanem csak kivonásra, páros/páratlan ellenőrzésre, és shift (páros szám bináris előállításának jobbra tolása, azaz felezése) műveletre.

Az algoritmus pszeudokódja az alábbi.

Bemenet: a , b pozitív egészek

Kimenet: g és d egészek úgy, hogy g páratlan és $\text{luko}(a, b) = g * 2^d$

$d = 0$

while (a és b egyaránt páros) {

$a = a/2$

$b = b/2$

$d++$

```

}
while (a <> b) {
    if (a páros) a = a/2
    else if (b páros) b = b/2
    else if (a > b) a = (a - b)/2
    else b = (b - a)/2
}
g = a
return g, d

```

Lánc törték. Az euklideszi algoritmus kapcsolatban van a lánc törtékkel is. Ez valójában az algoritmus kiterjesztése valós számokra. Tekintsük az $\text{Inko}(a, b)$ meghatározásakor kapott együtthatókat. Meglepő módon az alábbi összefüggést kapjuk:

$$\frac{a}{b} = q_0 + \frac{1}{q_1 + \frac{1}{q_2 + \frac{1}{\dots q_{n-2} + \frac{1}{q_{n-1}}}}} = [q_0, q_1, \dots, q_{n-1}]$$

Vegyük példaként az következőt:

$$\frac{180}{146} = [1, 4, 3, 2, 2].$$

Ha most kitöröljük az utolsó elemet, akkor kapjuk:

$$[1, 4, 3, 2] = \frac{37}{30} \Rightarrow -30 \cdot 180 + 37 \cdot 146 = 2 = \text{Inko}(180, 146).$$

Tudjuk, hogy minden valós számnak egyértelmű lánc tört felírása van. A lánc tört alakot előállító algoritmus akkor és csak akkor áll meg véges lépésben, ha az inputja egy racionális szám.

1.6. Feladatok

1. Írjunk egy eljárást, amely $\mathcal{O}(n)$ művelettel kiszámolja az x^y értékét, ahol y egy dupla pontosságú lebegő szám, x pedig egész.
2. Teszteljük le, hogy egy adott programozási nyelven (környezetben) az a^2 (négyzetre emelés) vagy pedig az $a \cdot a$ (önmagával történő szorzás) a gyorsabban elvégzett művelet. Terjesszük ki vizsgálatainkat magasabb hatványozásra is.
3. Hogyan módosítanánk a ? részben bemutatott algoritmust úgy, hogy $\mathcal{O}(n^2)$ helyett csak $\mathcal{O}(n \log n)$ legyen a műveletigény?
4. Írjunk osztás eljárást, amely csak összeadás műveletet használ.
5. Mutassuk meg, hogy $\text{Inko}(x, y, z)$ az a legkisebb egész, amely kifejezhető $ax + by + cz$ alakban, ahol $a, b, c \in \mathbb{Z}$.
6. Implementáljuk a bináris legnagyobb közös osztó eljárás, és végezzünk részletes tesztelést a futási idejére, összevetve a hagyományos változattal.
7. Vajon lehetséges bárhogyan is letesztelni programozással azt, hogy egy adott számítógépen az összeadás művelet ugyanannyi ideig tart, mint a szorzás?

2. fejezet

Nagypontosságú aritmetika

Nagyméretű vagy nagypontosságú számolásnál a hagyományos (egyszeres pontosságú) aritmetika nem használható. Itt elsősorban a számolások közbeni részeredmények pontossága a kritikus fontosságú. Ebben a fejezetben tárgyalt moduláris aritmetika egy lehetséges megoldást ad erre a feladatra.

Az alábbiakban fölteszük, hogy a, b, m egész számok bár a tárgyalt fogalmak és algoritmusok (racionális együtthatós) polinomok esetén is működnek.

2.1. Kongruenciák

Diszkrét matematika tantárgyból már tanultuk, itt most csak gyorsan átvesszük a legfontosabb fogalmakat. A

$$b \equiv c \pmod{m}$$

jelentése¹: a b és c számok m -mel osztva ugyanazt a maradékot adják. Másképpen is fogalmazhatunk:

- a különbségük osztható m -mel,
- van olyan k egész szám, hogy $b - c = km$.

Az egymással kongruens számok m darab osztályba: az m szerinti maradékosztályokba sorolhatók.

Aritmetikai műveletek kongruenciákkal. Tetszőleges m -re, ha

$$a \equiv b \pmod{m} \text{ és } c \equiv d \pmod{m}$$

akkor

$$a + c \equiv b + d \pmod{m} \text{ és } ac \equiv bd \pmod{m}.$$

Láthatjuk tehát, hogy az összeadás (a kivonás) és a szorzás műveletek könnyen elvégezhetők modulo m .

Az előzőek alapján nézzük most, hogyan boldogulunk az egyenletekkel. Például:

$$\begin{aligned} 5x + 3 &\equiv 6x - 8 \pmod{11} \\ 5x - 6x &\equiv -3 - 8 \pmod{11} \\ (6 - 5)x &\equiv 3 + 8 \pmod{11} \\ x &\equiv 11 \equiv 0 \pmod{11} \end{aligned}$$

Ez a speciális feladat könnyen megoldható volt, de ebből már sejtjük, hogy mi hiányzik: vajon hogyan tudunk osztani modulo m ? Ebben segít a következő tétel.

¹az egyenlőséget úgy olvassuk, hogy b kongruens c -vel modulo m

2.1.1. Tétel. Ha b és m relatív prímek, akkor $0, 1, \dots, m-1$ maradékok között van pontosan egy olyan c , amelyre

$$bc \equiv 1 \pmod{m}.$$

Ezt c számot a b reciprokának tekintjük modulo m .

A tétel ugyan definiálja a reciprokot — de vajon hogyan tudjuk meghatározni? Erre alkalmas az 1.4.3. szakaszban tárgyalt kiterjesztett euklideszi algoritmus, amelynek eredménye:

$$\text{lko}(b, m) = ub + vm.$$

Amennyiben b és m relatív prímek, tehát $\text{lko}(b, m) = ub + vm = 1$, akkor innen

$$1 \equiv ub \pmod{m}$$

adódik, tehát az u szám lesz b reciproka modulo m .

Lássuk most, hogyan tudjuk ezt alkalmazni az alábbi példára:

$$3x \equiv 5 \pmod{11}$$

A kiterjesztett euklideszi algoritmust elvégezve kapjuk, hogy

$$\text{lko}(3, 11) = 4 \cdot 3 + (-1) \cdot 11 = 1,$$

tehát a 4 lesz a 3 inverze modulo 11, ezért beszorozhatunk 4-gyel és kapjuk:

$$x \equiv 4 \cdot 3x \equiv 4 \cdot 5 \equiv 9 \pmod{11}$$

tehát a megoldás 9 (ami az $5/3$ megfelelője modulo 11).

Alkalmazás: nagy számok moduláris szorzása. Ha m prímszám, akkor minden b számhoz relatív prím, amely nem osztható m -mel. Tehát minden olyan b számnak, amely nem kongruens 0-val mod m van reciproka mod m . Alkalmazzuk ezt a megfigyelést! Ha például az a feladtunk, hogy számítsuk ki a

$$88 \cdot 76 - 65 \cdot 92 \pmod{3}$$

értékét, akkor ehhez először számítsuk ki:

$$88 \equiv 1, \quad 76 \equiv 1, \quad 65 \equiv 2, \quad 92 \equiv 2 \pmod{3},$$

majd pedig innen:

$$88 \cdot 76 - 65 \cdot 92 \equiv 1 \cdot 1 - 2 \cdot 2 \equiv -3 \equiv 0 \pmod{3}.$$

Megjegyezzük, hogy esetleg hasznos lehet több modulusra is előre kiszámolni a fentieket, így adott esetben rendkívül gyorsan elvégezhetjük a moduláris szorzást.

2.2. Kínai maradéktétel

A fentiekben tárgyaltak egyik legfontosabb hasznosítását a következő híres tétel adja meg.

2.2.1. Tétel (Kínai maradéktétel). Tegyük fel, hogy $m_1, \dots, m_k$ páronként relatív prímek. Ekkor

A) az $m_1, \dots, m_k$ szerinti maradékok meghatározzák az $M = m_1 \cdot \dots \cdot m_k$ szerinti maradékokat is, azaz ha

$$\begin{aligned} b &\equiv c \pmod{m_1} \\ b &\equiv c \pmod{m_2} \\ &\vdots \\ b &\equiv c \pmod{m_k} \end{aligned}$$

teljesülnek, akkor $b \equiv c \pmod{M}$; valamint

B) minden $b_1, \dots, b_k$ számsorozathoz van pontosan egy olyan M -nél kisebb szám, amelyre

$$\begin{aligned} b &\equiv b_1 \pmod{m_1} \\ b &\equiv b_2 \pmod{m_2} \\ &\vdots \\ b &\equiv b_k \pmod{m_k} \end{aligned}$$

és ezt a b számot ki is lehet számítani.

A tétel lehetővé teszi, hogy M -nél kisebb számokat az $m_1, \dots, m_k$ szerinti maradékaikkal egyértelműen reprezentáljuk.

Az ábrázolásnak van egy rendkívüli előnye, ugyanis a maradékokon végezhetünk egyszerű (tehát egyszeres pontosságú, és emiatt hardver szinten támogatott sebességű) műveleteket.

Példa. Legyen $m = 3$ és $n = 5$. Ekkor $M = 3 \cdot 5 = 15$. A kapott hozzárendelések:

$$\begin{aligned} 0 &\rightarrow [0, 0] \\ 1 &\rightarrow [1, 1] \\ &\vdots \\ 8 &\rightarrow [2, 3] \\ 9 &\rightarrow [0, 4] \\ &\vdots \\ 14 &\rightarrow [2, 4], \end{aligned}$$

azaz a $0, 1, \dots, M - 1$ számokat reprezentáljuk egy számpárral:

$$x \rightarrow [x \bmod m, x \bmod n].$$

Természetesen ez nem csak két értékkel érvényes. Ha például a b számot a $b_1, \dots, b_{100}$ sorozat, a c számot pedig a $c_1, \dots, c_{100}$ sorozat reprezentál, akkor

- a $b + c$ számot a $b_1 + c_1, \dots, b_{100} + c_{100}$ sorozat,
- a bc számot a $b_1 c_1, \dots, b_{100} c_{100}$ sorozat reprezentálja.

Ilyenkor egyébként a $b_i c_i$ számot el lehet osztani maradékosan m_i -vel és a maradékot írni a helyére. Legyünk azonban óvatosak: a bc szorzatot csak addig határozzák meg $m_1, \dots, m_k$ szerinti maradékai, amíg maga is kisebb, mint M .

Gyakorlati megvalósítás. Ahhoz, hogy a b szám m_1 szerinti legkisebb maradékát megtudjuk, el kell osztani b -t m_1 -gyel. Nehezebb feladat a b számot a $b_1, \dots, b_k$ maradékokból visszakapni. Ehhez keressük b -t a következő formában:

$$b \equiv c_1 + c_2 m_1 + c_3 m_1 m_2 + \dots + c_k m_1 m_2 \dots m_{k-1} \pmod{M},$$

ahol $0 \leq c_1 < m_1, \dots, 0 \leq c_k < m_k$. Ekkor a $c_1, \dots, c_k$ -ra a következő egyenleteknek kell teljesülni:

$$\begin{aligned} b_1 &\equiv c_1 \pmod{m_1} \\ b_2 &\equiv c_1 + c_2 m_1 \pmod{m_2} \\ &\dots \\ b_k &\equiv c_1 + c_2 m_1 + \dots + c_k m_1 \dots m_{k-1} \pmod{m_k}. \end{aligned}$$

Ezek az egyenletek könnyen megoldhatók: Mivel az m_i és m_j számok relatív prímek, az euklidészi algoritmussal megtalálható az az u_{ij} szám, amely az m_i reciproka modulo m_j :

$$0 < u_{ij} < m_j, \quad u_{ij} m_i \equiv 1 \pmod{m_j}.$$

Ezeket az u_{ij} számokat *előre ki lehet számolni* és eltárolni. Most már meg tudjuk oldani a fenti kongruenciákat:

$$\begin{aligned} c_1 &\equiv b_1 \pmod{m_1} \\ c_2 &\equiv u_{12}(b_2 - c_1) \pmod{m_2} \\ c_3 &\equiv u_{13}u_{23}(b_3 - c_1 + c_2 m_1) \pmod{m_3} \\ &\dots \end{aligned}$$

Példa. Legyen ismét $m = 3$ és $n = 5$. Tudjuk, hogy $\text{lnc}(3, 5) = 1 = 2 \cdot 3 + (-1) \cdot 5$. Ha most a $[2, 3]$ pár eredeti számát akarjuk visszakapni, akkor

$$\begin{aligned} x &= 2 \cdot (-1) \cdot 5 + 3 \cdot 2 \cdot 3 \pmod{15} \\ x &= -10 + 18 \pmod{15} \\ x &= 8 \pmod{15} \end{aligned}$$

vagyis a keresett szám az $x = 8$.

Összefoglalva: ha egész számokon kell összeadást, kivonást és szorzást elvégezni, akkor megbecsüljük, hogy mekkorára nőhet az eredmény, majd választunk olyan $m_1, \dots, m_k$ számokat, amelyek

- páronként relatív prímek,
- szorzatuk $M = m_1 \dots m_k$ legalább kétszer akkora, mint az eredmény nagysága,
- egyenként nem hosszabbak egy (gépi) szónál.

Minden b bemenő adathoz kiszámoljuk a $b_1, \dots, b_k$ számokat, amelyek rendre a b legkisebb maradékai az $m_1, \dots, m_k$ maradékokra nézve. Ugyanazokat a műveleteket, amelyeket az eredeti számolás követel, most modulo m_1 hajtjuk végre. Tehát minden művelet végrehajtása után külön kiszámítjuk az eredmény legkisebb maradékát modulo m_1 és azzal számolunk tovább. Hasonlóképpen járunk el a modulo $m_2, \dots$, modulo m_k maradékokkal is.

Ismerjük tehát a b' eredmény $b'_1, \dots, b'_k$ maradékait. Kiszámoljuk a b' -höz tartozó $c'_1, \dots, c'_k$ számokat. Ha szükséges, akkor előállítjuk a b' számot.

2.3. Nagyméretű lineáris egyenletrendszerek pontos megoldása

Tekintsük az

$$\mathbf{Ax} = \mathbf{b}$$

lineáris egyenletrendszert, ahol $\mathbf{A}$ egy $n \times n$ -es mátrix, a $\mathbf{b}$ pedig egy n -dimenziós vektor.

A közelítő (lebegőpontos számításokkal elvégzett) megoldást már megtanultuk a Közelítő és szimbolikus számítások kurzuson. Ott láttuk is, hogy amennyiben az $\mathbf{A}$ mátrix rosszul kondicionált, akkor a közelítő megoldás reménytelenül rossz is lehet a kerekítési hibák miatt. Bár általában a bemenő adatok pontatlanságán már nem tudunk változtatni, célunk most, hogy kerekítési hibák nélkül számoljunk.

Ehhez tegyük fel, hogy $\mathbf{A}$ és $\mathbf{b}$ egészeket tartalmaznak (ha racionális számokkal adott, akkor felszorozunk, ha pedig lebegőpontos számokkal, akkor azokat racionális számoknak tekintjük és felszorozunk).

Felhasználjuk a jól ismert Cramer-szabályt, amihez definiáljuk a következő mennyiségeket:

$$u_1 = \begin{vmatrix} b_1 & a_{12} & \dots & a_{1n} \\ \dots & & & \\ b_n & a_{n2} & \dots & a_{nn} \end{vmatrix}, \quad u_2 = \begin{vmatrix} a_{11} & b_1 & \dots & a_{1n} \\ \dots & & & \\ a_{n1} & b_n & \dots & a_{nn} \end{vmatrix}, \quad \dots, \quad u_n = \begin{vmatrix} a_{11} & \dots & a_{1,n-1} & b_1 \\ \dots & & & \\ a_{n1} & \dots & a_{n,n-1} & b_n \end{vmatrix},$$

továbbá legyen K az $\mathbf{A}$ mátrix determinánsa: $K = \det(\mathbf{A})$. Ekkor a Cramer szabály:

$$\mathbf{x}_i = \frac{u_i}{K},$$

amiből az $\mathbf{Ax} = \mathbf{b}$ átalakításával kapjuk, hogy

$$\mathbf{Au} = K\mathbf{b},$$

és ha $K \neq 0$, akkor a megoldás egyértelmű.

Most becsülnünk kell a K értékét, amihez felhasználjuk a Hadamard tételét, amely szerint

$$\det(\mathbf{A}) = K \leq (a_{11}^2 + \dots + a_{n1}^2)^{1/2} \cdot \dots \cdot (a_{1n}^2 + \dots + a_{nn}^2)^{1/2}.$$

Ha például minden i, j -re $a_{ij} < M$, akkor $K \leq (\sqrt{n}M)^n$.

A tétel következményeként kapjuk, hogy amennyiben $a_{ij} \leq M$ és $b_j \leq M$, akkor

$$u_i \leq (\sqrt{n}M)^n.$$

Mivel az $\mathbf{Au} = K\mathbf{b}$ egyenlet megoldásai az $u_1/K, \dots, u_n/K$ számok, ezért ezeket a $K, u_1, \dots, u_n$ számokat kell meghatároznunk.

Ehhez hajtsuk végre a következő lépéseket:

1. Szükségünk van egy prímszám-táblázatra, amely sok, M -nél kisebb, de ezen belül minél nagyobb prímszámokat tartalmaz. Válasszuk ki ezek közül p_1 -et. Számoljuk ki a $K \bmod p_1$ értékét. Ha ez a szám 0, akkor p_1 úgynevezett szerencsétlen prím, dobjuk el, és válasszunk helyette másikat, és számoljuk ki a maradékos osztást². Legyen tehát

$$K^{(1)} := K \bmod p_1$$

(a maradékok közül a legkisebb).

²Megjegyezzük, hogy annak a valószínűsége, hogy p_1 szerencsétlen meglehetősen kicsi, ugyanis $1/p_1$ a valószínűsége, hogy p_1 a K osztója és tudjuk, hogy p_1 jó nagy

2. Most $p_2, \dots, p_s$ -re ismételjük meg a fentieket. Ekkor kapjuk a $K^{(1)}, K^{(2)}, \dots, K^{(s)}$ nemzérus sorozatot, valamint teljesül, hogy $p_1 p_2 \dots p_s > (\sqrt{n}M)^n$.

Ha például $n = 100$, akkor $M = 10^7$, a prímek közel vannak M -hez, akkor $s \approx 120$.

3. Oldjuk meg az

$$\mathbf{A}\mathbf{u} \equiv K^{(i)}\mathbf{b} \pmod{p_i} \quad (i = 1, \dots, s)$$

egyenletrendszereket. Minden ilyen rendszert úgy oldunk meg, mint a hagyományos lineáris egyenletrendszereket, csak minden lépés után érdemes redukálni a legkisebb maradékra, valamint az osztás helyett a modulo reciprokával való szorzás műveletét alkalmazzuk (amelynek értékét a kiterjesztett euklidészi algoritmussal kapunk meg). Ezek eredménye:

$$\begin{aligned} \mathbf{u}^{(1)} &= (u_1^{(1)}, \dots, u_n^{(1)}) \\ &\vdots \\ \mathbf{u}^{(s)} &= (u_1^{(s)}, \dots, u_n^{(s)}). \end{aligned}$$

4. Végül használjuk a kínai maradéktételt: számítsuk ki azokat a legkisebb abszolút értékű $L, v_1, \dots, v_n$ számokat, amelyekre

$$\begin{aligned} L &\equiv K^{(1)}, v_1 \equiv u_1^{(1)}, \dots, v_n \equiv u_n^{(1)} \pmod{p_1} \\ &\vdots \\ L &\equiv K^{(s)}, v_1 \equiv u_1^{(s)}, \dots, v_n \equiv u_n^{(s)} \pmod{p_s} \end{aligned}$$

Belátható, hogy $L = K, v_1 = u_1, \dots, v_n = u_n$, tehát meghatároztuk a keresett számokat.

Vegyük észre, hogy a fenti módszer rendkívül jól párhuzamosítható: a kongruencia-rendszereket párhuzamosan is megoldhatjuk, ha van erre alkalmas hardverünk.

2.4. Ismételt négyzetre emelés

Egy másik hasznos alkalmazás a moduláris hatványozás. Az algoritmus a következő: ki akarjuk számolni az a^n értékét, ahol n pozitív egész szám.

1. Határozzuk meg az n bináris reprezentációját:

$$n = 2^k + n_{k-1}2^{k-1} + \dots + n_12 + n_0,$$

ahol $n_i \in \{0, 1\}$. Továbbá legyen $b_k := a$.

2. Minden $i = k-1, \dots, 0$ -ra hajtsuk végre a következő lépéseket: ha $n_i = 1$, akkor $b_i := b_{i+1}^2 a$, különben $b_i := b_{i+1}^2$.

3. A végeredmény: b_0 .

Az algoritmus például az a^{13} -on értékét az $((a^2 \cdot a)^2)^2 \cdot a$ alakban számolja ki.

Számoljuk csak végig...

Moduláris aritmetikában, például, a $8^{13} \pmod{17}$ -et a következőképpen számoljuk:

$$\begin{aligned} 8^{13} &\equiv (((8^2 \cdot 8)^2) \cdot 8 \equiv (((13 \cdot 8)^2)^2) \cdot 8 \\ &\equiv (2^2)^2 \cdot 8 = 4^2 \cdot 8 \equiv 16 \cdot 8 \equiv 9 \pmod{17}. \end{aligned}$$

Ez természetesen sokkal gyorsabb, mint direktbe kiszámolni a 8^{13} értékét és maradékosan elosztani 17-tel.

2.5. Néhány érdekes probléma

Nagy pontosságú 'csalás'? Vizsgáljuk a következő állításokat:

$$\sum_{n=1}^{\infty} \frac{\lfloor n \tanh(\pi) \rfloor}{10^n} = \frac{1}{81}.$$

Ez a kiértékelés rossz, bár 268 jegyig pontos. Továbbá

$$\sum_{n=1}^{\infty} \frac{\lfloor n \tanh(\pi/2) \rfloor}{10^n} = \frac{1}{81}.$$

szintén rossz, bár az első 12 jegyig pontos. Mindkét végtelen sor értéke egyébként transzcendens szám.

Miért érdekes egyáltalán ez a kérdés? A $\tanh(\pi)$ és $\tanh(\pi/2)$ értéke közel van 1-hez, ezért az lehet az érzésünk, hogy $\lfloor n \tanh(\pi) \rfloor = n - 1$ nagyon sok n -re. Továbbá

$$\sum_{n=1}^{\infty} \frac{n-1}{10^n} = \frac{1}{81}.$$

ezért volt a fenti sejtésünk. Nézzük most a lánc tört alakot:

$$\tanh(\pi) = [0, 1, \underline{267}, 4, 14, 1, 2, 1, 2, 2, 1, 2, 3, 8, 3, 1, \dots]$$

és

$$\tanh(\pi/2) = [0, 1, \underline{11}, 14, 4, 1, 1, 1, 3, 1, 295, 4, 4, 1, 5, 17, 7, \dots]$$

Nem lehet véletlen, hogy mindkét lánc törtben a harmadik szám az, amely egyel kisebb csak a fentebb mutatott pontosságnál! A magyarázat túlmutat a jegyzet keretein, a részleteket a [?] könyvben megtalálhatjuk.

Elég pontos? A

$$\sum_{n=1}^{\infty} \frac{\lfloor ne^{\pi\sqrt{163/9}} \rfloor}{2^n} = 1280640$$

kiértékelés pontos az első kb *félmilliárd* jegyre.

A π értéke nem 22/7. A π értékére a 22/7 racionális közelítést még Arkhimédész adta. Amennyiben a MATLAB Symbolic Toolbox-át használjuk, akkor

```
>> syms x
>> f=(x^4*(1-x)^4)/(1+x^2);
>> int(f,[0,1])
ans =
    22/7 - pi
```

Amit itt történik, az tulajdonképpen azon alapszik, hogy

$$\int_0^t \frac{x^4(1-x)^4}{1+x^2} dx = \frac{1}{7}t^7 - \frac{2}{3}t^6 + t^5 - \frac{4}{3}t^3 + 4t - 4 \arctan(t).$$

Innen vegyük észre, hogy

$$\int_0^1 x^4(1-x)^4 dx = \frac{1}{630},$$

valamint

$$\frac{1}{2} \int_0^1 x^4(1-x)^4 dx < \int_0^t \frac{x^4(1-x)^4}{1+x^2} dx < \int_0^1 x^4(1-x)^4 dx.$$

Innen azt kapjuk, hogy

$$223/71 < 22/7 - 1/630 < \pi < 22/7 - 1/1260 < 22/7$$

amely már elvezet a

$$3\frac{10}{71} < \pi < 3\frac{10}{70}$$

becsléshez.

2.6. Feladatok

1. Végezzünk numerikus kísérleteket arra vonatkozóan, hogy kiderítsük: vajon mekkora méretű számokkal végzett műveletek esetében éri meg használni a kínai maradéktételt?
2. Adott két egyenes, döntsük el, hogy metszik-e egymást (a feladat megoldásához használjuk nagy-pontosságú aritmetikát).
3. Maradék fa (remainder tree). Az $n \bmod p_1, n \bmod p_1, n \bmod p_3, n \bmod p_4$ kiszámításához használhatjuk a maradék fa elvét. Számítsuk ki $n \bmod p_1 p_2 p_3 p_4$ értékét, majd ezt redukáljuk modulo $p_1 p_2$ -vel, hogy megkapjuk $n \bmod p_1 p_2$ értékét, majd ezt redukáljuk modulo p_1 -gyel, hogy megkapjuk $n \bmod p_1$ -et, és így tovább. A feladat, hogy implementáljuk ezt az eljárást.
4. Írjunk faktoriális számolót, teszteljük a határait a használt programozási nyelv beépített típusaival. Hogyan változik a végrehajtási idő és a számítási kapacitás ha nagy pontosságú aritmetikát használunk?
5. Kedvenc programozási nyelvünkön számítsuk ki az $5^{4^{3^2}}$ értékét. Győződjünk meg róla, hogy az első és az utolsó 20-20 számjegy:

62060698786608744707...92256259918212890625.

3. fejezet

Gyors aritmetika

3.1. Nagy számok szorzása

Legyenek adottak u és v egész számok (10-es számrendszerben):

$$\begin{aligned}u &= u_{m-1}u_{m-2} \dots u_1u_0 = u_0 + 10u_1 + \dots + 10^{m-1}u_{m-1} \\v &= v_{n-1}v_{n-2} \dots v_1v_0 = v_0 + 10v_1 + \dots + 10^{n-1}v_{n-1}.\end{aligned}$$

Ha összeszorozzuk őket, akkor a következő alakú eredményt kapjuk:

$$\begin{aligned}uv = w &= w_{m+n-1}w_{m+n-2} \dots w_1w_0 \\&= w_0 + 10w_1 + \dots + 10^{m+n-1}w_{m+n-1}.\end{aligned}$$

Amennyiben ezt a ?? szakaszban tárgyalt, naiv szorzási algoritmussal számoljuk ki, akkor mn darab elemi műveletet végzünk el. Karacuba 1962-ben publikált észrevétele alapján azonban lehet gyorsabban is!

Az általánosság megszorítása nélkül tegyük fel, hogy u és v $2n$ jegyű egész számok, tehát

$$\begin{aligned}u &= u_{2n-1}u_{2n-2} \dots u_1u_0 \\v &= v_{2n-1}v_{2n-2} \dots v_1v_0.\end{aligned}$$

Írjuk most fel őket a következő alakban:

$$u = 10^n U_1 + U_0 \quad v = 10^n V_1 + V_0$$

ahol

$$\begin{aligned}U_1 &= u_{2n-1} \dots u_n \quad (\text{bal fele}) \\U_0 &= u_{n-1} \dots u_0 \quad (\text{jobb fele})\end{aligned}$$

valamint V_0 és V_1 hasonlóan. Amennyiben ezt a felírást használjuk, akkor a szorzatra a következő alakot kapjuk:

$$uv = 10^{2n}U_1V_1 + 10^n(U_0V_1 + U_1V_0) + U_0V_0,$$

ahol

$$U_0V_1 + U_1V_0 = (U_1 - U_0)(V_0 - V_1) + U_0V_0 + U_1V_1.$$

Innen pedig átrendezéssel:

$$uv = (10^{2n} + 10^n)U_1V_1 + 10^n(U_1 - U_0)(V_0 - V_1) + (10^n + 1)U_0V_0.$$

A fentiek szerint tehát 4 helyett 3 darab n -jegyű szorzás is elég. Megmutatható továbbá: az összeadások száma nem több, mint $9n$.

A formálisabb műveletigény vizsgálatához legyen $T(n)$ két n jegyű szám *szorzásához* szükséges elemi műveletek száma. Teljesül, hogy

$$T(2n) \leq 3T(n) + 9n. \quad (3.1)$$

Teljesül továbbá, hogy $T(1) = 1$. Nézzük meg néhány értékre:

$$\begin{aligned} T(4n) &\leq 3^2 T(n) + 3 \cdot 9n + 2 \cdot 9n = 3^2 T(n) + 9n(3^2 - 2^2) \\ T(8n) &\leq 3^2 T(n) + 9n(3^2 - 2^2) + 9 \cdot 2^2 n = 3^3 T(n) + 9n(3^3 - 2^3) \end{aligned}$$

Az általános képlet

$$T(2^k n) \leq 3^k T(n) + 9n(3^k - 2^k), \quad (3.2)$$

amelynek bizonyításához teljes indukciót használunk. Tegyük fel, hogy (3.2) teljesül minden k -nál kisebb számra. Alkalmazzuk (3.1) képletet:

$$T(2 \cdot 2^{k-1} n) \leq 3T(2^{k-1} n) + 9 \cdot 2^{k-1} n.$$

Most alkalmazva az indukciós feltevést kapjuk, hogy

$$T(2^{k-1} n) \leq 3^{k-1} T(n) + 9n(3^{k-1} - 2^{k-1}).$$

Helyettesítsünk be az előbbi egyenlőtlenségbe:

$$\begin{aligned} T(2 \cdot 2^{k-1} n) &\leq 3 \cdot 3^{k-1} T(n) + 9n \cdot 3 \cdot (3^{k-1} - 2^{k-1}) + 9 \cdot 2^{k-1} n \\ &= 3^k T(n) + 9n(3^k - 2^k), \end{aligned}$$

amely tehát bizonyítja a (3.2) helyességét.

Ha most $n = 10$ és $k = 2$, akkor $T(40) \leq 9T(10) + 450$. Tudjuk, hogy $T(10) \leq 100$, ezért $T(40) \leq 1350$; hagyományos szorzással ez $40 \cdot 40 = 1600$ lenne.

Minél nagyobb n , annál nagyobb a nyereség. Az általános képlet két n jegyű szám összeszorzásának műveletigényére:

$$27n^{\log_2 3} \approx 27n^{1.85} = \mathcal{O}(n^{1.85}).$$

3.1.1. Implementáció

A következőkben megmutatjuk a Karacuba algoritmus egy lehetséges implementációját MATLAB-ban. A kódban a `cut` paraméter meghatározza, hogy mekkora számjegyű számok esetében használjuk a hagyományos szorzást.

```
function uv = Kara(u,v,cut)
    m=floor(log10(u))+1; # u számjegyeinek száma
    n=floor(log10(v))+1; # v számjegyeinek száma
    N=max(n,m);
    if (N<cut)
        return u*v;
    end
    k=floor(N/2);
    pow10k = power(10,k);
    U0=rem(u,pow10k); U1=floor(u./pow10k);
    V0=rem(v,pow10k); V1=floor(v./pow10k);
```

```

T0=Kara (U1, V1, cut) ;
T1=Kara (U1-U0, V0-V1, cut) ;
T2=Kara (U0, V0, cut) ;
uv = (power(10, 2*k) + pow10k)*T0 + pow10k*T1 + (pow10k+1)*T2;
end;

```

3.1.2. Karacuba a gyakorlatban

Amennyiben n jegyű számokat kell összeszoroznunk, akkor az általános szabály, hogy alkalmazzuk a hagyományos $\mathcal{O}(n^2)$ algoritmust $n < 16$ esetben, a Karacuba módszert $16 \leq n < 4096$ között és a később tárgyalandó FFT módszert (lásd 4.5.1. szakasz¹) nagyobb n -ekre.

3.2. Nagy mátrixok szorzása

Legyenek A és B $n \times n$ -es mátrixok. Az $AB = C$ szorzatmátrix elemeinek kiszámítása a

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

képlettel történik. Mivel a k, i és j indexeket 1-től n -ig kell futtatni, ezért adódik, hogy a műveletigény n^3 szorzás és $n^2(n-1)$ összeadás.

Az alábbiakban bemutatjuk Strassen módszerét, amely kihasználja, hogy bizonyos számolt részeredmények többször is felhasználhatóak a C mátrix elemeinek kiszámolásához.

A Strassen-képleteket 2×2 -es A és B mátrixokra mutatjuk meg. A mátrix elemeire a szokásos jelöléseket használva definiáljuk:

$$\begin{aligned}
I &= (a_{11} + a_{22})(b_{11} + b_{22}) \\
II &= (a_{21} + a_{22})b_{11} \\
III &= a_{11}(b_{12} + b_{22}) \\
IV &= a_{22}(b_{21} + b_{11}) \\
V &= (a_{11} + a_{12})b_{22} \\
VI &= (a_{21} - a_{11})(b_{11} + b_{12}) \\
VII &= (a_{12} + a_{22})(b_{21} + b_{22})
\end{aligned}$$

Ellenőrizhető, hogy

$$\begin{aligned}
c_{11} &= I + IV - V - VII, & c_{12} &= III + V, \\
c_{21} &= II + IV, & c_{22} &= I + III - II + VI.
\end{aligned}$$

Azt kaptuk tehát, hogy az eredeti 8 szorzás és 4 összeadás helyett 7 szorzás és 18 összeadás elvégzése szükséges.

Mivel a fenti képletek nem használják fel a kommutativitást, ezért azok 4×4 -es mátrixokra is használhatók, ahol az a_{ij} és b_{ij} elemek 2×2 -es blokkokat jelölnek. Ezt rekurzívan folytatva használhatjuk őket 2^k rendű mátrixok összeszorozására.

Nézzük meg most a műveletigényt formálisan is. Legyen $n \times n$ -es mátrixok szorzásának műveletigényében $S(n)$ az összeadások és kivonások száma, és $M(n)$ a szorzások száma. Strassen képleteket

¹bár a 4.5.1 szakaszban a gyors Fourier transzformációt polinomok konvolúciójának kiszámítására tárgyaljuk, a módszer természetesen (nagy méretű) számok szorzására is alkalmazható

alkalmazva két négyzetes 2^k rendű mátrix összeszorzásához $2^{k-1} \times 2^{k-1}$ -es blokkok 7 szorzatát kell kiszámolnunk. Ez $7M(k-1)$ szorzást jelent, így

$$M(k) = 7M(k-1).$$

Mivel $M(0) = 1$, azt kapjuk, hogy

$$M(k) = 7^k = 2^{k \log_2 7} = n^{\log_2 7}$$

ahol $n = 2^k$ a mátrixok mérete volt.

Az összeadást és kivonást illetően $2^{k-1} \times 2^{k-1}$ -es blokkok 18 összeadását és kivonását, valamint 7 szorzását kell elvégeznünk. Az utóbbi $7S(k-1)$, az előbbi pedig $18(2^{k-1})^2 = 9 \cdot 2^{2k-1}$ összeadást jelent. Így

$$S(k) \leq 9 \cdot 2^{2k-1} + 7S(k-1).$$

Vezessük be a $B(k) = 7^{-k}S(k)$ jelölést, amivel a rekurzió átírható

$$B(k) = \frac{9}{2} \left(\frac{4}{7}\right)^k + B(k-1)$$

alakba. k szerint összeadva, és kihasználva, hogy $B(0) = 0$ kapjuk, hogy

$$B(k) = \frac{9}{2} \sum_{j=1}^k \left(\frac{4}{7}\right)^j < \frac{9}{2} \cdot \frac{4}{3} = 6.$$

Ebből tehát

$$S(k) \leq 6 \cdot 7^k = 6 \cdot 2^{k \log_2 7} = 6n^{\log_2 7} = 6n^{2,807}$$

Megállapíthatjuk tehát, hogy Strassen módszere $\mathcal{O}(n^{2,81})$ műveletigényű. Érdekességgént megjegyezzük, hogy Strassen publikációjának címe: *Gaussian elimination is not optimal*.

Amennyiben a szorzandó mátrixok rendje nem 2 hatvány, akkor megfelelő mennyiségű 1 hozzá vételel a főátlóba 2-hatvány rendűvé tehetőek.

Az a sejtés, hogy minden $\varepsilon > 0$ -ra létezik olyan algoritmus amely két $n \times n$ -es mátrixok szorzatát $cn^{2+\varepsilon}$ művelettel számol ki, ahol a c konstans az ε függvénye. A jelenlegi csúc: $c \cdot n^{2,3728639}$ (a gyakorlatban használhatatlanul nagy c értékkel).

3.3. Mátrixok invertálása

Bár mátrixok invertálása a gyakorlatban egy olyan művelet, amelyet lehetőség szerint kerülni kell, Frobenius módszere az invertálásra némi tanulsággal szolgál.

Tegyük fel, hogy A egy olyan mátrix, amely blokk-formában fölírható

$$\begin{pmatrix} P & Q \\ R & S \end{pmatrix}$$

ahol P négyzetes és nonszinguláris, továbbá $U = S - R(P^{-1}Q)$ szintén nonszinguláris. Akkor

$$A^{-1} = \begin{pmatrix} P^{-1} + (P^{-1}Q)(U^{-1}RP^{-1}) & -(P^{-1}Q)U^{-1} \\ -(U^{-1}RP^{-1}) & U^{-1} \end{pmatrix}$$

Az állítás helyessége egyszerű ellenőrzéssel bizonyítható. Vegyük észre, hogy a számítás két mátrixinvertálást (P és U) és hat darab mátrixszorzást (rendre $P^{-1}Q$, $R(P^{-1}Q)$, RP^{-1} , $U(RP^{-1})$, $(P^{-1}Q)(U^{-1}RP^{-1})$ és $(P^{-1}Q)U^{-1}$) igényel.

3.4. Számok osztása

Olyan algoritmust keresünk, amely két n jegyű szám hányadosának n jegyig pontos meghatározását $\mathcal{O}(n^2)$ elemi művelettel megadja. Meglepő módon mindezt a Newton-módszer biztosítja².

Amennyiben adott az $f : \mathcal{R} \rightarrow \mathcal{R}$ differenciálható függvény, akkor a Newton-iteráció képlete:

$$x_{m+1} = x_m - \frac{f(x_m)}{f'(x_m)},$$

feltéve, hogy $f'(x_m) \neq 0$.

Most az $1/u$ kiszámításához az $1/x - u = 0$ egyenletet kell megoldanunk. Alkalmazzuk erre a Newton képletet, ahol tehát az $f(x) = 1/x - u$ függvényt használjuk:

$$x_{m+1} = x_m - \frac{1/x_m - u}{-1/x_m^2} = 2x_m - ux_m^2 = x_m(2 - ux_m) \quad (3.3)$$

Megjegyezzük, hogy szokás még az

$$x_{m+1} = x_m + x_m(1 - ux_m)$$

alakot is használni. Ez matematikailag ekvivalens az előző képlettel, ugyanakkor a számítógépes implementációban előnyösebb lehet. Ahhoz, hogy az eredményt $2n$ bit pontosságra megkapjuk az (3.3) képlet használatával, ahhoz ki kell számolni a szorzatot x_m és $(2 - ux_m)$ között ahol x_i pontossága adott (n bit). Ezzel szemben a szorzatot az x_i és $(1 - ux_i)$ között csak n bitre pontosan kell, hogy kiszámítsuk, hiszen az első n bit (a bináris pont után) az $(1 - ux_i)$ -ben mind nullák.

Ha az $1/u$ számot n értékes jegy pontossággal akarjuk tudni, akkor elég x_k -t kiszámítani, ahol

$$k = \lceil \log n \rceil + 1.$$

Amennyiben a hiba adott úgy, hogy $\epsilon_i = 1 - ux_i$, akkor

$$\begin{aligned} \epsilon_{i+1} &= 1 - ux_{i+1} \\ &= 1 - u(x_i(2 - ux_i)) \\ &= 1 - 2ux_i + u^2x_i^2 \\ &= (1 - ux_i)^2 \\ &= \epsilon_i^2. \end{aligned}$$

Ez itt nem más, mint a jól ismert kvadratis konvergencia.

A kezdeti érték megválasztása egy érdekes probléma lehet. Érdemes az u értékét átskálázni úgy, hogy $0,5 \leq D \leq 1$ teljesüljön, természetesen úgy, hogy a számlálót is ugyanekkor értékkel eltoljuk. Ezután a következő lineáris közelítő formát használhatjuk:

$$x_0 = T_1 + T_2u \approx \frac{1}{u}$$

a kezdeti érték választásához. Ahhoz, hogy a hiba abszolút értékének maximumát minimalizáljuk a $[0,5, 1]$ intervallumon, használjuk a

$$x_0 = \frac{48}{17} - \frac{32}{17}u$$

képletet.

²ezt a módszert a Közelítő és szimbolikus számítások kurzusokon nemlineáris egyenletek gyökeinek megkeresésére tanuljuk

Pszudokód A következő pszudokód a fentiekben tárgyalt Newton-módszer egy lehetséges megvalósítását írja le.

```
Vegyük u lebegőpontos alakját (kettes számrendszerben):
      M * pow(2, e) (ahol tehát 1 <= M < 2)
u' = u / pow(2, e+1) // skálázzunk 0.5 és 1 közé
N' = N / pow(2, e+1) // a számlálót is
x = 48/17 - 32/17 * u'
repeat ceil(log2((P+1)/log2(17)))-szer
    x = x + x * (1 - u' * x)
end
return u' * X
```

3.5. Barrett redukció

A Barrett redukció a

$$c = a \bmod n.$$

naiv kiszámítását (amely persze az iménti, 3.4. szakaszban tárgyalt gyors osztás eljárást használná) optimalizálja. Természetesen csak bizonyos feltételek mellett érdemes használni, mégpedig akkor, ha n konstans és $a < n^2$ teljesül. Ekkor az osztást érdemes szorzásra cserélni.

Az általános elv a következő. Legyen $s = 1/n$ az n inverze lebegőpontos aritmetikával számolva. Akkor

$$a \bmod n = a - \lfloor as \rfloor n$$

ahol szokás szerint $\lfloor x \rfloor$ az alsó-egészrész jelöli. Az eredmény pontos mindaddig, amíg az s értékét megfelelő pontossággal ki tudjuk számítani.

Barrett azt az esetet vizsgálta, amikor a számok elférnek egy gépi szóban. Tekintsük a következő `reduce` nevű függvényt:

```
function reduce(a)
    q = a / n // egész osztás (alsó-egészrész az eredmény)
    return a - q * n
```

Barrett ötlete az volt, hogy az $1/n$ értékét közelítsük $m/2^k$ értékkel, hiszen a 2^k értékkel történő osztás elvégezhető jobbra léptetéssel. Az m értékének kiszámításához adott 2^k esetén használjuk az

$$\frac{1}{n} = \frac{2^k/n}{n(2^k/n)} = \frac{2^k/n}{2^k} = \frac{m}{2^k}$$

összefüggést. Ahhoz, hogy m egész legyen a $2^k/n$ értékét kell valahogyan kerekítenünk. A legközelebbi egészre kerekítés a legjobb közelítést adhatja, azonban kaphatjuk hogy $m/2^k$ nagyobb lesz, mint $1/n$, ami alulcsordulást eredményezhet. Ezért általában az $m = \lfloor 2^k/n \rfloor$ számítási módot használjuk. Kapjuk tehát a módosított változatot a `reduce` függvényre:

```
function reduce(a)
    q = (a * m) >> k // a ">> k" jelentése: k-szoros bitléptetés
    return a - q * n
```

Mivel azonban $m/2^k \leq 1/n$ ezért a q értéke nagyon kicsi is lehet, emiatt aztán az a értékére csak a $[0, 2n)$ intervallumot tudjuk biztosítani a $[0, n)$ helyett. Egy megfelelő kivonás művelet azonban ezt is korrigálja, amivel megkapjuk a végleges változatot:

```
function reduce(a)
    q = (a * m) >> k
    a -= q * n
    if n <= a {
        a -= n
    }
    return a
```

3.6. Feladatok

1. Vessük össze a 3.1 szakaszban ismertetett Karacuba módszert a MATLAB beépített szorzás műveletével végrehajtási idő tekintetében. Ehhez próbáljuk meg a kódot minél jobban optimalizálni.
2. A számtani-mértani közép definíciója a következő: Legyen x és y két egész szám. Először számítsuk ki a számtani közepüket, amely legyen a_1 , majd számoljuk ki a mértani közepüket, ezt jelölje g_1 :

$$a_1 = \frac{1}{2}(x + y)$$

$$g_1 = \sqrt{xy}$$

A kapott két számnak újra kiszámoljuk a számtani és a mértani közepét, és ezt iteráljuk minden a_n és g_n párra:

$$a_{n+1} = \frac{1}{2}(a_n + g_n)$$

$$g_{n+1} = \sqrt{a_n g_n}$$

Ekkor az a_n és a g_n sorozatok ugyanahhoz a számhoz tartanak, ami x és y számtani-mértani közepe.

A számtani-mértani közép meghatározása felhasználható, többek között a Gamma függvény kiszámításához. Derítsük ki, hogy hogyan, és implementáljuk az algoritmusokat.

3. A Gauss-Legendre algoritmus a π számjegyeinek gyors kiszámítására használható. Meglepő módon mindössze 25 lépésből 45 millió jegyre pontosan meghatározza a π értékét. A lépések a következők:

(a) Kezdeti értékek:

$$a_0 = 1 \quad b_0 = \frac{1}{\sqrt{2}} \quad t_0 = \frac{1}{4} \quad p_0 = 1.$$

(b) Ismételjük az alábbiakat mindaddig, amíg az a_n és b_n különbsége a kívánt pontosságon belül van:

$$a_{n+1} = \frac{a_n + b_n}{2},$$

$$b_{n+1} = \sqrt{a_n b_n},$$

$$t_{n+1} = t_n - p_n(a_n - a_{n+1})^2,$$

$$p_{n+1} = 2p_n.$$

(c) A közelítés értéke:

$$\pi \approx \frac{(a_{n+1} + b_{n+1})^2}{4t_{n+1}}.$$

Implementáljuk az eljárást nagy pontosságú aritmetika fölhasználásával.

4. Hogyan módosítanánk a Karacuba eljárást komplex számok szorzására? Naivan 4 darab szorzás és 2 összeadásra van szükség, de ez módosítható 3 szorzásra és 5 összeadásra.
5. Strassen algoritmus polinomok gyors szorzására is használható, amelyet az alábbiakban vázolunk. A naiv eljárás $\mathcal{O}(mn)$ műveletigényű, ahol m és n a két polinom tagjainak a száma. Tegyük fel, hogy $n = m = 2k + 1$. A $P(x)$ és $Q(x)$ összeszorzásához tekintsük a következő alakot:

$$\begin{aligned} P(x) &= P_1(x) + x^{2^k} \cdot P_2(x) \\ Q(x) &= Q_1(x) + x^{2^k} \cdot Q_2(x), \end{aligned}$$

ahonnan

$$P(x) \cdot Q(x) = P_1(x)Q_1(x) + x^{2^k} \cdot (P_1(x)Q_2(x) + P_2(x)Q_1(x)) + x^{2^{k+1}} \cdot P_2(x)Q_2(x).$$

Definiáljuk tehát a következő 3 szorzatot:

$$\begin{aligned} A(x) &= P_1(x)Q_1(x) \\ B(x) &= P_2(x)Q_2(x) \\ C(x) &= (P_1(x) + P_2(x)) \cdot (Q_1(x) + Q_2(x)) \end{aligned}$$

Akkor

$$P(x)Q(x) = A(x) + x^{2^k} \cdot (C(x) - A(x) - B(x)) + x^{2^{k+1}} \cdot B(x).$$

Mutassuk meg, hogy a futási idő $\mathcal{O}(3^k) = \mathcal{O}(n \log_2 3) = \mathcal{O}(n^{1,585})$.

Implementáljuk az eljárást, és végezzünk numerikus teszteket a végrehajtási időre (összehasonlítva a naiv módszerrel).

6. Olvasnivaló:

<https://randomascii.wordpress.com/2014/10/09/intel-underestimates-error-bound/>

4. fejezet

Műveletek polinomokkal

4.1. Alapfogalmak

Ebben a fejezetben elsősorban egyváltozós polinomokra vonatkozó műveletek elvégzését tárgyaljuk. Az $x_1, \dots, x_k$ változók polinomjának nevezünk minden olyan kifejezést, amely ezekből a változókból és valós számokból a szorzás, összeadás és kivonás segítségével építhető fel. Példa egy háromváltozós polinomra: $P(x_1, x_2, x_3) = 1/3 + x_1^3 - 5x_2x_3$

Mint azt korábban tárgyaltuk, a $P(x)$ egyváltozós $(n-1)$ -ed fokú polinomot az együtthatóival szokás megadni.

Látjuk majd, hogy az algoritmusok műveletigényének meghatározásánál különbséget teszünk a lineáris és a nemlineáris műveletek között.

lineáris művelet: összeadás és olyan szorzás, ahol az egyik tényező ismert szám. Egy polinom helyettesítési értékének kiszámítása lineáris műveletek sorozatával elvégezhető.

nemlineáris művelet: olyan szorzás, amelynél mindkét tényező paramétert tartalmazó kifejezés vagy olyan osztás, amelyben a nevező paramétert tartalmaz.

4.2. Polinomok kiértékelése - Horner módszer

Tekintsük a

$$P(x) = a_0 + a_1x + \dots + a_nx^n$$

egyváltozós polinomot. Ha a műveleteket úgy végezzük el, ahogyan azok ki vannak jelölve, akkor $n-2$ darab összeadást és $2n-1$ szorzást kell elvégeznünk.

Átalakítással azonban a szorzások száma n -re csökkenthető. Ezt a

$$P(x) = a_0 + x(a_1 + x(a_2 + \dots + x(a_{n-1} + xa_n) \dots))$$

alakkal tudjuk elérni. Megjegyezzük, hogy se kevesebb szorzás, se kevesebb összeadás nem elegendő.

4.3. Polinomok kiértékelése - prekondicionálással

Tekintsük most azt az esetet, amikor a $P(x)$ polinomot kell kiértékelnünk **különböző** x helyeken. Ilyenkor bizonyos dolgokat megéri előre kiszámolni, mert azokat többször is felhasználjuk a számolás során. Természetesen a prekondicionáláshoz csak az $a_0, \dots, a_n$ együtthatókat használhatjuk fel.

Érdekes módon itt arról van szó, hogy egy

$$Q(x) = b_0 + b_1x + \dots + b_{m-1}x^{m-1}$$

polinomot felfoghatjuk úgy, mint két vektor,

$$\mathbf{b} = (b_0, b_1, \dots, b_{m-1}) \text{ és } \mathbf{x} = (1, x, \dots, x^{m-1})$$

skalárszorzatát: $\mathbf{b}\mathbf{x}^T$. Tegyük fel, hogy m páros. Ekkor $Q(x)$ tehát felírható a következő alakban:

$$\begin{aligned} Q(x) &= (b_0 + b_{m-1}x^{m-1}) + (b_1x + b_{m-2}x^{m-2}) + (b_2x^2 + b_{m-3}x^{m-3}) + \dots \\ &= \left((b_0 + x^{m-1})(1 + b_{m-1}) + (b_1 + x^{m-2})(x + b_{m-2}) + \dots - \frac{m}{2}x^{m-1} \right) - \\ &\quad - (b_0b_{m-1} + b_1b_{m-2} + \dots + b_{m/2-1}b_{m/2}). \end{aligned}$$

Nézzük meg ennek az alaknak a műveletigényét:

- (i) az $x^2, x^3, \dots, x^{m-1}$ előállításához $m - 2$ darab szorzás,
- (ii) a $b_0b_{m-1} + \dots + b_{m/2-1}b_{m/2}$ előállításához $m/2$ darab szorzás,
- (iii) míg a nagyobb zárójelben lévő kifejezés kiszámításához $m/2$ szorzás szükséges.

A (ii) alatti műveletek *előre* elvégezhetők, azok nem függnak az x értékétől, hiszen csak a $Q(x)$ polinom együtthatói szerepelnek benne. Az (i) alatti műveletekben a b_i együtthatók nem fordulnak elő¹.

Számolt műveletek száma: $m + m/2$. Ha most úgy gondolkodunk, hogy k darab prekondicionált n -ed fokú polinomot kellene *ugyanezen* a helyen kiértékelni, akkor ez a szám $km/2 + m$ lenne². Ezt a tulajdonságot fogjuk felhasználni a

$$P(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$$

polinom kiértékeléséhez, ahol $n \leq km$. Osszuk a tagokat m -es csoportokba:

$$P(x) = Q_0(x) + Q_1(x)x^m + Q_2(x)x^{2m} + \dots + Q_{k-1}(x)^{(k-1)m},$$

ahol

$$\begin{aligned} Q_0(x) &= a_0 + \dots + a_{m-1}x^{m-1}, \\ Q_1(x) &= a_m + \dots + a_{2m-1}x^{m-1}, \\ &\dots \\ Q_{k-1}(x) &= a_{(k-1)m} + \dots + a_{km-1}x^{m-1}, \end{aligned}$$

Mint láttuk, a $Q_i(x)$ polinomok kiszámítása összesen legfeljebb $km/2 + m$ műveletbe kerül, hiszen $i = 0, \dots, k-1$, tehát pontosan k darab van belőlük. Ha x^m már megvan, akkor $P(x)$ kiszámítása a $Q_0(x), \dots, Q_{k-1}(x)$ együtthatóiból további $k-1$ művelettel elvégezhető. Az egész számolás így legfeljebb $km/2 + m + k$ műveletet követel, ahol $km \geq n$.

Ha például $k = m = \lceil \sqrt{n} \rceil$, akkor a műveletigény $n/2 + 2\lceil \sqrt{n} \rceil$.

¹Ennek akkor van jelentősége, ha több polinomot is ugyanazon az x helyen kell kiértékelni. De most azzal az esettel foglalkozunk, amikor *különböző* helyeken történik a kiértékelés.

²Ennek magyarázata: ilyenkor a fenti (i) alatti műveletekből (kihasználva azt, hogy mindegyik polinomot ugyanazon helyen kell kiértékelni) csak $m - 2$ darab szorzás kell, míg marad a (iii) alatti műveletek sora, amiből pedig $km/2$ kell, hiszen a polinomok együtthatói különbözőek.

A polinom prekondicionálása tehát abból áll, hogy előre ki kell számolni és eltárolni k darab összeget, amelyek

$$s_j = a_{j \cdot m} a_{(j+1) \cdot m-1} + a_{j \cdot m+1} a_{(j+1) \cdot m-2} + \dots$$

alakúak és $j = 0, \dots, k-1$. A számítógépek programozási könyvtáraiban az elemi függvényeket rendszerint közelítő polinommal számolják. A polinomok gyakran ilyen prekondicionált formában vannak jelen.

4.4. Tabellázás

Azzal a feladattal folytatjuk, amikor a $P(x) = a_0 + a_1x + \dots + a_nx^n$ polinomhoz szeretnénk elkészíteni a

$$P(u_1), \dots, P(u_k)$$

táblázatot, ahol $u_1, \dots, u_k$ adott számok.

Amennyiben a 4.2. szakaszban tárgyalt Horner elrendezést használjuk, akkor ez kn darab nemlineáris művelet. Az iménti, 4.3. szakaszban ismertetett prekondicionálással ez $\approx kn/2$.

Lehet-e azzal nyerni, ha az egész táblázatot *egyszerre* számítjuk ki? Feltéve, hogy $k \geq n$ az alábbi eljárással $k \log^2 n$ művelettel ez megoldható, ahol a nemlineáris műveletek száma $k \log n$.

A $P(u_1), \dots, P(u_n)$ értékeket akarjuk kiszámolni. Ehhez kiszámoljuk az

$$I(x, u_1, \dots, u_n) = (x - u_1)(x - u_2) \dots (x - u_n)$$

polinom együtthatóit is, amelyek gyökei az $u_1, \dots, u_n$ számok.

$$I(x, u_1, \dots, u_n) = x^n - \sigma_1 x^{n-1} + \sigma_2 x^{n-2} - \dots$$

alakban írható fel, ahol

$$\begin{aligned} \sigma_1 &= u_1 + \dots + u_n, \\ \sigma_2 &= u_1 u_2 + u_1 u_3 + \dots + u_{n-1} u_n, \\ &\dots \\ \sigma_n &= u_1 u_2 \dots u_n \end{aligned}$$

Ezeket az $u_1, \dots, u_n$ változók elemi szimmetrikus polinomjainak nevezzük.

Egy rekurzív eljárást adunk meg. Tegyük fel, hogy az algoritmus már adott, ha $P(x)$ foka kisebb, mint m . Most definiáljuk az eljárást minden olyan $P(x)$ polinomra, amelynek foka kisebb vagy egyenlő, mint $2m$. Feltehetjük, hogy $P(x)$ foka $n = 2m$ (ha nem, akkor nulla együtthatójú tagokkal kiegészítjük):

$$P(x) = a_0 + a_1x + \dots + a_{2m}x^{2m}.$$

Ki akarjuk számítani $P(x)$ értékét az $u_1, \dots, u_{2m}$ helyeken. Osszuk ezeket két részre: $u_1, \dots, u_m$ és $u_{m+1}, \dots, u_{2m}$. Képezzük az

$$I(x, u_1, \dots, u_m), \quad I(x, u_{m+1}, \dots, u_{2m})$$

polinomokat – ezeket meg tudjuk határozni, hiszen feltettük, hogy az eljárás m -re már ismert. Osszuk el a $P(x)$ polinomot az $I(x, u_1, \dots, u_m)$ polinommal:

$$P(x) = I(x, u_1, \dots, u_m)Q_0(x) + R_0(x),$$

valamint az $I(x, u_{m+1}, \dots, u_{2m})$ polinommal:

$$P(x) = I(x, u_{m+1}, \dots, u_{2m})Q_1(x) + R_1(x).$$

Tudjuk, hogy az $R_0(x)$ és $R_1(x)$ maradékpolinomok fokszáma kisebb, mint m . Mivel az I polinom eltűnik az $u_1, \dots, u_m$ és $u_{m+1}, \dots, u_{2m}$ helyeken, ezért

$$\begin{aligned} P(u_1) &= R_0(u_1), \dots, P(u_m) = R_0(u_m) \\ P(u_{m+1}) &= R_1(u_{m+1}), \dots, P(u_{2m}) = R_1(u_{2m}) \end{aligned}$$

ahelyett, hogy a $2m$ -ed fokú $P(x)$ polinomot értékelnénk ki az $u_1, \dots, u_{2m}$ helyeken, az m -nél kisebb fokú $R_0(x)$ és $R_1(x)$ polinomokat értékeljük ki az $u_1, \dots, u_m$ és $u_{m+1}, \dots, u_{2m}$ helyeken.

Az m -nél kisebb fokú polinomokra az eljárás definiálva van.

Hátravan még az $I(x, u_1, \dots, u_{2m})$ kiszámítása, de ez könnyű, mert

$$I(x, u_1, \dots, u_{2m}) = I(x, u_1, \dots, u_m)I(x, u_{m+1}, \dots, u_{2m}).$$

Példa. Számítsuk ki a $P(x) = x^2 + 3x + 4$ polinomot az $u_1 = 5$ és $u_2 = 6$ helyeken. A fenti eljárást használva

$$I(x, 5, 6) = (x - 5)(x - 6) = x^2 - 11x + 30.$$

Továbbá

$$I_1(x, 5) = x - 5, \quad \text{és} \quad I_2(x, 6) = x - 6$$

Ezeket fölhasználva kapjuk, hogy

$$P(x) = (x - 5)(x + 8) + 44 \quad \text{és} \quad P(x) = (x - 6)(x + 9) + 58,$$

ezért $P(5) = 44$ és $P(6) = 58$.

4.5. Polinomok szorzása

A

$$P(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1} \quad \text{és} \quad Q(x) = b_0 + b_1x + \dots + b_{n-1}x^{n-1}$$

polinomok szorzata az az

$$R(x) = P(x)Q(x) = c_0 + c_1x + \dots + c_{2n-2}x^{2n-2}$$

polinom, amelynek c_i együtthatóit az alábbi képletek határozzák meg:

$$c_i = a_0b_i + a_1b_{i-1} + \dots + a_ib_0.$$

A $(c_0, \dots, c_{2n-2})$ sorozatot az $(a_0, \dots, a_{n-1})$ és $(b_0, \dots, b_{n-1})$ sorozatok *konvolúciójának* nevezzük. Erre egyébként tekinthetünk úgy is, mint két n dimenziós vektor olyan szorzata, amelynek eredménye egy $2n$ dimenziós vektor, amely tehát különbözik a hagyományos belső- vagy külső szorzattól (ahol rendre számot, illetve mátrixot kapunk eredményül).

Ha a $(c_0, \dots, c_{2n-2})$ sorozatot a fentebb felírt módon akarjuk meghatározni, akkor n^2 darab szorzást kell elvégezni az a_i és b_j együtthatókkal. Azonban gyorsabban is lehet: Toom eljárással, amelyet a következőkben ismertetünk.

Kiindulásnak vegyünk $2n - 1$ darab számot, például a

$$0, 1, \dots, 2n - 2$$

számokat. Most ahelyett, hogy az $R(x) = P(x)Q(x)$ polinom együtthatóit számolnánk ki, először számoljuk az $R(x)$ helyettesítési értékeit a $k = 0, 1, \dots, 2n - 2$ helyeken:

$$\begin{aligned} R(k) &= P(k)Q(k) \\ &= c_0 + c_1k + c_2k^2 + \dots \\ &= (a_0 + a_1k + a_2k^2 + \dots)(b_0 + b_1k + b_2k^2 + \dots). \end{aligned}$$

Ehhez tehát ki kell számolni a $P(x)$ és $Q(x)$ helyettesítési értékeit a $0, 1, \dots, 2n - 2$ helyeken, majd összeszorozni őket. Ezután pedig meg kell határoznunk az $R(x)$ együtthatóit az $R(0), R(1), \dots, R(2n - 2)$ értékekből. Vegyük észre, hogy ez az interpolációs feladat valójában a $c_0, \dots, c_{2n-2}$ ismeretlenek kiszámítását jelenti a

$$\begin{aligned} c_0 &= R(0) \\ c_0 + c_1 + c_2 + \dots + c_{2n-2} &= R(1) \\ c_0 + 2c_1 + 2^2c_2 + \dots + 2^{2n-2}c_{2n-2} &= R(2) \\ &\dots \\ c_0 + (2n-2)c_1 + \dots + (2n-2)^{2n-2}c_{2n-2} &= R(2n-2) \end{aligned}$$

lineáris egyenletrendszerből.

Ebben a pillanatban még az a benyomásunk, hogy semmit nem nyertünk, sőt egy lineáris egyenletrendszert kell megoldanunk, amelynek műveletigénye $O(n^3)$. Amennyiben viszont megint különbséget teszünk a lineáris és a nemlineáris műveletek között, akkor gyorsabbak is lehetünk.

Bár a

$$P(0)Q(0), P(1)Q(1), \dots, P(2n-2)Q(2n-2)$$

értékek kiszámítása $2n-1$ darab nemlineáris művelet, a fenti lineáris egyenletrendszer (amely ugyanarra az eredményre vezet) megoldható csak lineáris művelettel! Ehhez vegyük észre, hogy paraméterek csak az $R(0), R(1), \dots, R(2n-2)$ -ben vannak, ezeket pedig a megoldás során nem kell egymással se szorozni, se osztani. A nemlineáris műveletek száma tehát $2n - 1$. A $(c_0, \dots, c_{2n-2})$ együtthatókat definiáló *eredeti* képletben n^2 darab nemlineáris művelet fordult elő. Ebből a szempontból tehát a Toom-féle eljárás egyszerűbb.

Tovább is léphetünk: a lineáris műveletekből sem kellene sokat elvégezni. Ez könnyen teljesíthető, csak a helyettesítési értékeket kell megfelelően megadni! A fentiekben azt az esetet tárgyaltuk, amikor az $1, 2, \dots, 2n - 2$ értékekkel számoltunk. Azonban nem vagyunk ezekhez a számokhoz kötve. Ha ezeket komplex egységgyököknek választjuk, akkor a konvolúció megvalósítható $O(n \log n)$ művelettel. Ez lesz a véges Fourier-transzformált módszer.

4.5.1. Véges Fourier-transzformált

Legyen $P(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$ polinom. Legyen $r \geq n$ és

$$\omega = \omega_r = e^{\frac{2\pi i}{r}}$$

az első komplex r -edik egységgyök, és $\omega^2 = -1$. Az $\omega^0 = 1, \omega, \omega^2, \dots, \omega^{r-1}$ számok mind különbözők és $\omega^r = 1$.

Képezzük most a $P(1), P(\omega), \dots, P(\omega^{r-1})$ értékeket:

$$\hat{a}_k = P(\omega^k) = a_0 + a_1\omega^k + a_2\omega^{2k} + \dots + a_{n-1}\omega^{(n-1)k}$$

az $(\hat{a}_0, \dots, \hat{a}_{r-1})$ komplex számsorozatot az $(a_0, \dots, a_{n-1})$ számsorozat *véges Fourier-transzformáltjának* nevezzük.

A transzformációnak két, számunkra most fontos tulajdonsága van:

- a visszatranszformálás hasonló képlettel történik:

$$a_j = \frac{1}{r}(\hat{a}_0 + \hat{a}_1\omega^{-j} + \hat{a}_2\omega^{-2j} + \dots + \hat{a}_{r-1}\omega^{-(r-1)j}),$$

- valamint ha $r \geq 2n-2$, és a $(c_0, \dots, c_{2n-2})$ sorozat az $(a_0, \dots, a_{n-1})$ és $(b_0, \dots, b_{n-1})$ sorozatok konvolúciója, akkor

$$\hat{c}_0 = \hat{a}_0\hat{b}_0, \quad \hat{c}_1 = \hat{a}_1\hat{b}_1, \dots, \hat{c}_{r-1} = \hat{a}_{r-1}\hat{b}_{r-1}.$$

A konvolúció tehát a következő műveletsorozat: transzformáció, r darab szorzás, visszatranszformáció.

Legyen $r \geq n$ és

$$F_k^{(r)}(a_0, \dots, a_{n-1}) = \hat{a}_k \quad (k = 0, \dots, r-1).$$

A $2r$ elemű Fourier-transzformáció nem más mint 2 darab r elemű Fourier-transzformáció. Ez a tény egy *divide and conquer* típusú algoritmusra vezet el bennünket.

Legyen $a_0, \dots, a_{2r-1}$ egy $2r$ elemű sorozat. Ennek a transzformálásához az $\omega_{2r} = e^{\frac{2\pi i}{2r}}$ egységgyököket használjuk. Ezzel a jelöléssel:

$$\begin{aligned} \hat{a}_k &= a_0 + a_1\omega_{2r}^k + a_2\omega_{2r}^{2k} + \dots + a_{2r-1}\omega_{2r}^{(2r-1)k} \\ &= a_0 + a_2\omega_{2r}^{2k} + a_4\omega_{2r}^{4k} + \dots + a_1\omega_{2r}^k + a_3\omega_{2r}^{3k} + \dots \\ &= a_0 + a_2(\omega_{2r}^2)^k + a_4(\omega_{2r}^2)^{2k} + \dots + \omega_{2r}^k[a_1 + a_3(\omega_{2r}^2)^k + a_5(\omega_{2r}^2)^{2k} + \dots], \end{aligned}$$

ahol tehát megfelelően csoportosítottuk az együtthatókat. Ha $\omega_{2r} = e^{\frac{2\pi i}{2r}}$, akkor $\omega_{2r}^2 = \omega_r$, ezért

$$F_k^{(2r)}(a_0, \dots, a_{2r-1}) = F_k^{(r)}(a_0, a_2, \dots, a_{2r-2}) + \omega_{2r}^k F_k^{(r)}(a_1, a_3, \dots, a_{2r-1})$$

Megmutatható:

$$K(2r) \leq 2K(r) + 6r$$

továbbá $K(1) = 0$, ezért a fenti képletet m -szer egymás után alkalmazva

$$K(2^m) \leq 3m \cdot 2^m$$

Ha r a 2 -nek egész kitevőjű hatványa, akkor

$$K(r) \leq 3r \log_2 r.$$

Szokás szerint: ha r nem 2 hatvány, akkor a sorozatot nullákkal egészítjük ki.

Gyors Fourier transzformáció: Konkrét megvalósítás

Legyen

$$\mathbf{a} = [a_0, a_1, \dots, a_{n-1}]^T \quad \mathbf{b} = [b_0, b_1, \dots, b_{n-1}]^T$$

két n -dimenziós vektor, amelynek komponensei komplex számok. az $\mathbf{a} * \mathbf{b} = \mathbf{c}$ konvolúciót akarjuk kiszámolni

Legyenek

$$F^{-1}(\mathbf{a}) := F^{-1}[a_0, a_1, \dots, a_{n-1}, 0, \dots, 0] := (f_0, f_1, \dots, f_{2n-1})^T$$

és

$$F^{-1}(\mathbf{b}) := F^{-1}[b_0, b_1, \dots, b_{n-1}, 0, \dots, 0] := (g_0, g_1, \dots, g_{2n-1})^T$$

a $2n$ komponensre kiegészített $\mathbf{a}$ és $\mathbf{b}$ vektorok véges inverz Fourier transzformáltjaik. Legyen továbbá

$$F^{-1}(\mathbf{a}) \circ F^{-1}(\mathbf{b}) := [f_0g_0, f_1g_1, \dots, f_{2n-1}g_{2n-1}]^T$$

a komponensenkénti szorzásokkal kapott vektor. Ekkor

$$F^{-1}(\mathbf{a} * \mathbf{b}) = F^{-1}(\mathbf{a}) \circ F^{-1}(\mathbf{b}).$$

A fenti, utolsó képlettel ekvivalens alak:

$$\mathbf{a} * \mathbf{b} = F[F^{-1}(\mathbf{a}) \circ F^{-1}(\mathbf{b})],$$

amely lehetővé teszi, hogy két n -dimenziós vektor konvolúcióját az FFT segítségével is kiszámoljuk, amennyiben n kettőnek egész kitevőjű hatványa.

4.6. Feladatok

1. A fejezetben nem volt szó polinomok osztásáról. A szakirodalomból keressünk egy erre alkalmas (gyors) algoritmust és implementáljuk, teszteljük.
2. Teszteljük le a MATLAB Symbolic Toolbox rendszerében elérhető polinom szorzás végrehajtási idejét összevetve a (lebegőpontos) `conv` függvénnyel, ami polinomok/vektorok konvolúcióját számítja ki.
3. A Horner módszer használható az osztott differencia

$$\frac{p(y) - p(x)}{y - x}$$

kiszámítására. Legyen adott a

$$p(x) = \sum_{i=0}^n a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \dots + a_n x^n$$

polinom a következő lépéseket kell végrehajtanunk:

$$\begin{array}{ll} b_n = a_n, & d_n = b_n, \\ b_{n-1} = a_{n-1} + b_n x, & d_{n-1} = b_{n-1} + d_n y, \\ \vdots & \vdots \\ b_1 = a_1 + b_2 x, & d_1 = b_1 + d_2 y, \\ b_0 = a_0 + b_1 x. & \end{array}$$

Ennek végén kapjuk, hogy

$$\begin{aligned} p(x) &= b_0, \\ \frac{p(y) - p(x)}{y - x} &= d_1, \\ p(y) &= b_0 + (y - x)d_1. \end{aligned}$$

Teszteljük a módszer hatékonyságát a naív eljárással összevetve, különösen olyan példákra, amelyekre x és y értéke közel azonos.

4. Mint azt tárgyaltuk, a Horner-módszer optimális abban az értelemben, hogy egy polinom helyettesítési értékének kiszámításához a legkevesebb szorzás és összeadás műveletet használja. Azonban ezt úgy éri el, hogy az egymást követő összeadás és szorzás műveletek függenek a korábban végrehajtott műveletek eredményeitől. Így a módszer nehezen, vagy egyáltalán nem párhuzamosítható.

Az Estrin-módszer párhuzamos gépekre alkalmas eljárás, amelynek ötlete a következő példán mutatható be. Jelölje

$$P_n(x) = C_0 + C_1x + C_2x^2 + C_3x^3 + \dots + C_nx^n$$

egy n -változós polinomot. Az Estrin alakot írva a következő résszámitásokat kapjuk:

$$\begin{aligned} P_3(x) &= (C_0 + C_1x) + (C_2 + C_3x)x^2 \\ P_4(x) &= (C_0 + C_1x) + (C_2 + C_3x)x^2 + C_4x^4 \\ P_5(x) &= (C_0 + C_1x) + (C_2 + C_3x)x^2 + (C_4 + C_5x)x^4 \\ P_6(x) &= (C_0 + C_1x) + (C_2 + C_3x)x^2 + ((C_4 + C_5x) + C_6x^2)x^4 \\ P_7(x) &= (C_0 + C_1x) + (C_2 + C_3x)x^2 + ((C_4 + C_5x) + (C_6 + C_7x)x^2)x^4 \\ P_8(x) &= (C_0 + C_1x) + (C_2 + C_3x)x^2 + ((C_4 + C_5x) + (C_6 + C_7x)x^2)x^4 + C_8x^8 \\ P_9(x) &= (C_0 + C_1x) + (C_2 + C_3x)x^2 + ((C_4 + C_5x) + (C_6 + C_7x)x^2)x^4 + (C_8 + C_9x)x^8 \\ &\dots \end{aligned}$$

amelyből láthatjuk, hogy az x^{2^n} értékét egyszer kell csak számolnunk, és azt megtarthatjuk addig, amíg szükség van rá.

Feladatunk, hogy implementáljuk a módszert, és vessük össze a végrehajtási idejét a Horner módszerrel szekvenciális nyelven és olyan környezetben is, ahol párhuzamos végrehajtás is megengedett.

5. A Ruffini-algoritmus egy $P(x)$ polinom $x - r$ taggal történő osztásának hatékony kiszámolására alkalmas. Keressük meg a leírását és készítsük el az implementációját.

5. fejezet

Polinomok gyökeiről

Azt mondjuk, hogy c a $p(x)$ polinom gyöke, ha $p(c) = 0$. Amennyiben $p(x)$ valós együtthatós polinom, amelynek gyöke az α komplex szám, akkor gyöke ennek $\bar{\alpha}$ konjugált is.

Egy fontos és egyszerű észrevétel, hogy az $p(x)$ polinom $(x-c)$ első fokú polinommal való osztásának maradéka egyenlő az $p(x)$ polinom $p(c)$ értékével. Legyen ugyanis

$$p(x) = (x - c)q(x) + r.$$

Ha most vesszük mindkét oldal értékét $x = c$ -nél, akkor $p(c) = r$.

Ennek a ténynek egy következménye, hogy a c szám akkor és csak akkor gyöke az $p(x)$ polinomnak, ha $p(x)$ osztható $(x - c)$ -vel. Ha $p(x)$ osztható valamely $ax + b$ első fokú polinommal, akkor osztható az $x - \frac{-b}{a}$ -val is, vagyis egy $x - c$ alakú polinommal.

Egy polinom gyökeinek a meghatározása tehát ekvivalens lineáris osztóinak felkutatásával. A Horner-elrendezés segítségével meghatározhatjuk a

$$p(x) = (x - c)q(x) + r$$

alakot.

Példa. Legyen $p(x) = x^4 - 8x^3 + x^2 + 4x - 9$, és ezt osszuk el $x + 1$ -gyel:

	1		-8		1		4		-9	
-1		1		-1 · 1 + (-8) = -9		-1 · (-9) + 1 = 10		-1 · 10 + 4 = -6		-1 · (-6) + (-9) = -3

Ezért a hányados: $x^3 - 9x^2 + 10x - 6$, a maradék pedig $p(-1) = -3$.

Többszörös gyök. Amennyiben található olyan k szám, amelyre $p(x)$ maradék nélkül osztható az $(x - c)^k$ -val, de nem osztható $(x - c)^{k+1}$ -gyel, akkor a k szám a $p(x)$ polinom c gyökének *multiplicitása*.

Az algebra alaptétele a következőt állítja: minden legalább első fokú komplex együtthatós polinomnak van legalább egy (általában komplex) gyöke. Az alaptétel szerint egy polinom tehát elsőfokú polinomok szorzatára lehet bontani – ezzel majd később foglalkozunk.

Minden n -edfokú komplex együtthatós polinomnak n gyöke van, ha minden gyökét annyiszor számítjuk, amennyi a multiplicitása. Ez utóbbi tényre építve állíthatjuk, hogy bármely, n -nél nem magasabb fokú polinomot egyértelműen meghatároznak azon értékei melyeket az ismeretlenek bármely, n -nél több különböző értéke mellett vesz fel. Ha ismerjük egy n -ed fokú polinom értékét $n + 1$ különböző pontban, akkor bármely más pontban is kiszámíthatjuk¹.

¹Ez a tény adódik a Lagrange-féle interpolációs képletből.

Megoldóképletek. A másodfokú polinomegyenlet megoldásának módszerét már a régi görögök is ismerték. A harmad- és negyedfokú polinomok gyökeire csak a XVI. században találtak képleteket. 300 évig folytak sikertelen kísérletek a magasabb fokú polinomok gyökeinek meghatározására szolgáló képlet megkeresésére.

Majd 1832-ben Abel és Ruffini bebizonyította, hogy ilyen képletek nem léteznek $n \geq 5$ esetén *tetszőleges* n -ed fokú polinomra.

Fontos azonban látni, hogy az Abel-Ruffini tétel nem zárja ki azt a lehetőséget, hogy talán minden *konkrét* komplex együtthatós polinom gyökei mégis kifejezhetők lennének valahogyan az együtthatók-ból gyökvonások valamilyen kombinációi segítségével²

1830-as évek eleje: Galois³ megmutatta, hogy milyen feltételek mellett oldható meg radikálokkal valamely adott polinomegyenlet. Ezen eredmény alapján kiderült, hogy minden $n \geq 5$ -re megadható olyan n -ed fokú polinom, amely nem oldható meg radikálokkal.

5.1. Sturm tétel

Valós együtthatós $f(x)$ polinom valós gyökeinek számának meghatározására mutatunk egy eljárást, amely az euklideszi algoritmustól csupán a fellépő maradékok előjelében különbözik.

Legyen

$$\begin{aligned} f_0(x) &:= f(x) \\ f_1(x) &:= f'(x) \\ &\dots \\ f_{j-1}(x) &:= q_{j-1}(x)f_j(x) - f_{j+1}(x) \\ &\dots \\ f_{m-1}(x) &:= q_{m-1}(x)f_m(x) \end{aligned}$$

ahol az $f_{j+1}(x)$ maradékpolinom fokszáma kisebb, mint az $f_j(x)$ osztó polinom fokszáma. $f_m(x)$ az összes $f_j(x)$ polinom legnagyobb közös osztója.

A fent definiált polinomsorozat alapvető tulajdonságai:

5.1.1. Segéd-tétel. Ha az $f(x)$ polinomnak a véges $[a, b]$ intervallumba eső gyökei mind egyszeresek, akkor minden $x \in [a, b]$ -re $f_m(x) \neq 0$ (sőt, valójában $f_m(x)$ állandó előjelű az $[a, b]$ -ben), ha valamely $1 \leq j \leq m$ -re és $x \in [a, b]$ -re $f_j(x) = 0$, akkor

$$f_{j-1}(x)f_{j+1}(x) < 0,$$

ha valamely $x \in [a, b]$ -re $f_0(x) = 0$, akkor

$$f'_0(x)f_1(x) > 0.$$

Legyen $f_0(x), f_1(x), \dots, f_m(x)$ valós együtthatós polinomoknak olyan sorozata, amelyben a fokszámok monoton csökkennek. Ezt *Sturm sorozatnak* nevezzük valamely $[a, b]$ intervallumon, ha a fenti lemmában szereplő 1-3 tulajdonságok teljesülnek és ezen felül még az is igaz, hogy

$$f(a)f(b) \neq 0.$$

Jelöljük $S(x)$ -szel az $f_0(x), f_1(x), \dots, f_m(x)$ Sturm sorozatban előforduló *előjelváltások* számát. Jelváltás akkor van, ha pozitív szám után negatív következik, vagy fordítva.

²tehát, hogy minden konkrét polinom gyöke megadható radikálokkal (n -edik gyökökkel: $\sqrt[n]{\cdot}$)

³„Don't cry, Alfred! I need all my courage to die at twenty.”

5.1.2. Tétel (Sturm). Legyen $f(x)$ valós együtthatós polinom. Ha

$$f(x) := f_0(x), f_1(x), \dots, f_m(x)$$

Sturm sorozat valamely $[a, b]$ intervallumon, akkor $f(x)$ -nek (a, b) -ben pontosan $S(a) - S(b)$ számú gyöke van.

Példa. Legyen $f(x) = x^3 + 3x^2 - 1$, határozzuk meg a valós gyökeinek a számát. A polinom Sturm rendszere:

$$\begin{aligned} f(x) &= x^3 + 3x^2 - 1, \\ f_1(x) &= 3x^2 + 6x, \\ f_2(x) &= 2x + 1, \\ f_3(x) &= 1. \end{aligned}$$

Most állapítsuk meg ennek a rendszernek az előjelváltásainak számát a $(-\infty, \infty)$ intervallumon:

	$f(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$	váltások száma
$-\infty$	—	+	—	+	3
∞	+	+	+	+	0

Kapjuk tehát, hogy az $f(x)$ polinomnak 3 valós gyöke van.

Valós gyökök szétválasztása. A Sturm tétel alkalmazható valós gyökök szétválasztására is a következőképpen:

1. Számítsuk ki $S(x)$ értékét különböző x -ekre.
2. Ha találunk olyan x_1, x_2 párt, amelyre $S(x_1) \neq S(x_2)$, akkor biztosak lehetünk abban, hogy x_1 és x_2 közrefog egy gyököt.
3. Felezzük meg az intervallumot, amelyben a gyök található: $S((x_1 + x_2)/2)$.
4. Menjünk vissza az 1. lépésre.

5.2. További korlátok polinomokra

Legyen adott

$$f(x) = \sum_{i=0}^n a_i x^i = a_n \prod_{i=1}^n (x - \alpha_i)$$

n -ed fokú polinom. Vezessük be a következő jelöléseket:

- $H(f)$ (vagy $\|f\|_\infty$), amelyet magasságnak nevezünk és $\max_{i=0, \dots, n} |a_i|$
- $\|f\|$ a kettes norma $\sqrt{\sum_{i=0}^n |a_i|^2}$
- $L(f)$ (vagy $\|f\|_1$), amelyet hosszúságnak nevezünk és $\sum_{i=0}^n |a_i|$

Ekkor adódik, hogy

$$H(f) \leq \|f\| \leq L(f) \leq (n+1)H(f).$$

Igaz továbbá, hogy

$$|a_n| \prod_{\alpha_i > 1} |\alpha_i| < \|f\|.$$

Ezek következményeképpen adódik továbbá, hogy

$$|a_{n-i}| \leq \binom{n}{i} |a_n| \prod_{\alpha_i > 1} |\alpha_i| \leq \binom{n}{i} \|f\|.$$

Gyökökre vonatkozó korlátok A fentiekben bemutatott Sturm módszerét *valós* gyökök számának meghatározására. Néha azonban szükségünk lehet a valós és a komplex gyökök megkülönböztetésére. Tekintsük ehhez a következő példát.

Legyen W_{20} az a polinom, amelynek gyökei a $-1, -2, \dots, -20$ számok. Azaz $W_{20} = (x+1)(x+2)\dots(x+20) = x^{20} + 210x^{19} + \dots + 20!$. Tekintsük most a

$$W_{20}(x) + 2^{-23}x^{19}$$

polinomot. Arra számítunk, hogy ennek is húsz darab valós gyöke van, az eredeti W_{20} -hoz közel. A helyzet azonban az, hogy ennek csak 10 valós gyöke van, közelítőleg a $-1, -2, \dots, -7, -8.007, -8.917, -20.847$ értékekkel, valamint 5 pár komplex konjugált gyöke, amelyek közelítő értéke rendre a $-10.095 \pm 0.6435i, -11.794 \pm 1.652i, -13.992 \pm 2.519i, -16.731 \pm 2.813i, -19.502 \pm 1.940i$ értékek.

Legyen most $f = \sum_{i=0}^n a_i x^i$, valamint az $f(x)$ gyökei az $\alpha_1, \dots, \alpha_n$ számok. Vezessük be az alábbi jelölést:

$$\bullet \text{ rb}(f) = \max_{1 \leq i \leq n} |\alpha_i|,$$

A következő korlátok teljesülnek:

$$\bullet \text{ rb}(f) \leq 1 + \max(|a_i|)/|a_n|.$$

$$\bullet \text{ rb}(f) \leq 2 \max \left\{ \frac{|a_{n-1}|}{a_n}, \sqrt{\frac{|a_{n-2}|}{a_n}}, \dots, \sqrt[n-1]{\frac{|a_1|}{a_n}}, \sqrt[n]{\frac{|a_0|}{a_n}} \right\}$$

Amennyiben ezeket a korlátokat a $W_{20}(x)$ polinomra alkalmazzuk, akkor rendre a 1.38×10^{19} és 420 értékeket kapjuk, amely mutatja a második korlát erősségét, legalábbis az elsővel szemben.

5.3. Laguerre módszer

Ebben a szakaszban bemutatunk egy módszert, amely Edmond Laguerre nevéhez fűződik. A Newton-módszerrel ellentétben ez polinomok gyökeinek megkeresésére van specializálva.

A módszer tárgyalása előtt ismertetünk egy gyökkorlátos eredményt is, amely szintén Laguerre-től származik.

Valós gyökök további korlátai A módszere olyan $p(x) = \sum_{k=0}^n a_k x^k$ polinomokra alkalmazható, amelyeknek csak valós gyökei vannak. Ezek a gyökök a

$$-\frac{a_{n-1}}{na_n} \pm \frac{n-1}{na_n} \sqrt{a_{n-1}^2 - \frac{2n}{n-1} a_n a_{n-2}}$$

intervallumban vannak.

Például az $x^4 + 5x^3 + 5x^2 - 5x - 6$ polinomnak négy valós gyöke van: $-3, -2, -1$ és 1 . A képlet a

$$-\frac{5}{4} \pm \frac{3}{4} \sqrt{\frac{35}{3}};$$

intervallumot adja, amely lebegőpontosan a $[-3.8117, 1.3117]$ intervallum – ebben az esetben tehát egész jó befoglalást kaptunk a gyökökre.

Gyökök megkeresése A módszer egyik tulajdonsága, hogy a kezdeti értéktől *függetlenül* majdnem mindig konvergál *valamelyik* gyökhöz, legyen az akár komplex szám. Az eljárás a következő:

1. Válasszunk egy x_0 kezdeti értéket, és legyen $a = \infty$.
2. Adott lépésszámmig vagy $a < \epsilon$ teljesüléséig hajtsuk végre a következőket:
 - Ha $p(x_k)$ nagyon kicsi, akkor az algoritmus véget ér, x_k egy közelítő érték.
 - Legyen $G = \frac{p'(x_k)}{p(x_k)}$ és $H = G^2 - \frac{p''(x_k)}{p(x_k)}$.
 - Legyen $a = \frac{n}{G \pm \sqrt{(n-1)(nH-G^2)}}$, ahol az előjelet úgy választjuk, hogy a nevező a lehető legnagyobb abszolút értékű legyen (a lebegőpontos számolásoknál ez fontos).
 - Végül legyen $x_{k+1} = x_k - a$.

5.4. Rezultánsok

A rezultánsok fogalmát a következő kérdés megválaszolására vezették be: mi a szükséges és elégséges feltétele annak, hogy az $f(x)$ és $g(x)$ polinomoknak van közös gyöke? Legyen

$$\begin{aligned} f(x) &= f_0 + f_1x + \dots + f_mx^m \\ g(x) &= g_0 + g_1x + \dots + g_nx^n, \end{aligned}$$

$\alpha_1, \dots, \alpha_m$ az f polinom gyökei, $\beta_1, \dots, \beta_n$ pedig a g polinom gyökei. Ekkor

$$\text{res}(f, g) = f_m^n g_n^m \prod_{j=1}^n \prod_{i=1}^m (\alpha_i - \beta_j)$$

sorozatot az f és g polinomok *rezultánsának* nevezzük.

Ez a definíció matematikai értelemben teljesen rendben van, azonban praktikusán használhatatlan, mivel a gyökök ismeretét tételezi föl. A *Sylvester-mátrixot* előállítjuk az együtthatókból:

$$\begin{pmatrix} f_m & f_{m-1} & \dots & f_1 & f_0 & 0 & 0 & \dots & 0 \\ 0 & f_m & f_{m-1} & \dots & f_1 & f_0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & f_m & f_{m-1} & \dots & f_1 & f_0 & 0 \\ 0 & \dots & 0 & 0 & f_m & f_{m-1} & \dots & f_1 & f_0 \\ g_n & g_{n-1} & \dots & g_1 & g_0 & 0 & 0 & \dots & 0 \\ 0 & g_n & g_{n-1} & \dots & g_1 & g_0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & g_n & g_{n-1} & \dots & g_1 & g_0 & 0 \\ 0 & \dots & 0 & 0 & g_n & g_{n-1} & \dots & g_1 & g_0 \end{pmatrix}$$

A Sylvester-mátrix determinánsát kell meghatározni, és megvizsgálni, hogy mikor 0 az értéke. Ez megtehető $\mathcal{O}((n+m)^3)$ lépésben például a jól ismert Gauss-eliminációval.

Egy másik lehetséges megoldást a következő algoritmus ad, ahol `LeadCoeff` a megadott polinom főegyütthatóját adja, a `deg` a polinom fokszámát, a `mod` művelet pedig a két polinom osztásával keletkező maradékot határozza meg.

```
Input: F, G polinomok
F[1] = F;
F[2] = G;
i = 2;
R[1] = 1;
while deg(F[i]) > 0 {
    F[i+1] = F[i-1] mod F[i]
    di = deg(F[i]);
    di1 = deg(F[i-1]);
    R[i] = power(-1, di-di1) * power(LeadCoeff(F[i]), di1-di) * R[i-1]
    i++;
}
if (F[i] <> 0) return R[i] * power(LeadCoeff(F[i]), deg(F[i-1]))
else return 0;
```

Szimbolikus eset. A rezultánsok legfontosabb előnyei, hogy a bemeneti polinomok szimbolikus együtthatókat is tartalmazhatnak. Legyen például

$$\begin{aligned} f(x) &:= 2x^3 - \xi x^2 + x + 3 \\ g(x) &:= x^2 - 5x + 6. \end{aligned}$$

Ekkor

$$\begin{aligned} \text{res}(f, g) &= \begin{vmatrix} 2 & -\xi & 1 & 3 & 0 \\ 0 & 2 & -\xi & 1 & 3 \\ 1 & -5 & 6 & 0 & 0 \\ 0 & 1 & -5 & 6 & 0 \\ 0 & 0 & 1 & -5 & 6 \end{vmatrix} \\ &= 36\xi^2 - 429\xi + 1260 = 0. \end{aligned}$$

A másodfokú egyenlet gyökei: $\xi = 20/3$ és $\xi = 21/4$.

5.5. Gröbner bázisok

Tekintsük a következő, m darab polinomból álló egyenletrendszert:

$$\begin{aligned} f_1(x_1, \dots, x_n) &= 0 \\ &\vdots \\ f_m(x_1, \dots, x_n) &= 0 \end{aligned}$$

A Gröbner bázisok meghatározásával tulajdonképpen meghatározzuk a rendszer megoldását is, mert ezeknek ugyanazok a gyökei.

Példa. A következő 3 változós rendszer megoldását keressük:

$$\begin{aligned}f_1 &= xz - xy^2 - 4x^2 - \frac{1}{4} \\f_2 &= y^2z + 2x + \frac{2}{1} \\f_3 &= x^2z + y^2 + \frac{1}{2}x\end{aligned}$$

A rendszer Gröbner bázisa (a $x < y < z$ rendezéssel):

$$\begin{aligned}g_1 &= z + 64/65x^2 - 432/65x^3 + 168/65x^2 - 354/65x + 8/5 \\g_2 &= y^2 - 8/13x^4 + 54/13x^3 - 8/13x^2 + 17/26x \\g_3 &= x^5 - 27/4x^4 + 2x^3 - 21/16x^2 + x + 5/32\end{aligned}$$

Érdekes módon ezt az alakot 'trianguláris' alaknak nevezzük, a Gauss-elmináció eredményéhez hasonlóan. Az analógia ott van, hogy a Gauss elimináció valójában a Gröbner bázisok módszerének egy speciális esete, amikor minden egyenlet lineáris.

A fenti alakból x meghatározható a 3. egyenletből, majd az behelyettesítjük a 2. egyenletbe megkapjuk y értékét, stb. A közelítő megoldás egyébként

$$(-0.128475, 0.321145, -2.356718)$$

Megjegyezzük, hogy egyváltozós polinomegyenlet megoldása lényegesen egyszerűbb, mint egy polinom rendszer megoldásának megkeresése. Itt most arról van szó, hogy úgy oldjuk meg a rendszert, hogy azt átalakítjuk, majd végül egyváltozós polinomok sorozatát kell megoldanunk.

A Gröbner bázisok kiszámítására szolgáló algoritmus, sőt, még a kapcsolódó alapfogalmak is túlmutatnak a jegyzet keretein, így azokat itt nem ismertetjük.

5.6. Feladatok

1. Implementáljuk a 5.2 szakaszban ismertetett korlátokat, és teszteljük az erősségüket véletlen együtthatós polinomokra.
2. A Horner-módszer a Newton-módszerrel kombinálva alkalmas polinomok össze való gyökének megkeresésére. Legyen adott a $p_n(x)$ n -ed fokú polinom, amelynek gyökeire igaz, hogy $z_n < z_{n-1} < \dots < z_1$. Legyen x_0 kezdeti érték úgy, hogy $x_0 > z_1$. Hajtsuk végre a következő iterációs lépéseket:
 - (a) Használjuk a Newton-módszert a z_1 megkeresésére.
 - (b) Használjuk a Horner-módszer a $(x - z_1)$ taggal történő leosztásra, amellyel megkapjuk a p_{n-1} -et. Térjünk vissza az 1. lépésre, ahol használjuk a p_{n-1} polinomot és a z_1 -et, mint kezdeti értéket.

Implementáljuk és teszteljük ezt az eljárást.

3. Végezzünk kísérleteket véletlen együtthatós polinomok gyökeinek eloszlásáról. Használjunk ehhez többfajta véletlenszám generátort különböző paraméterekkel.
4. Az $x^2 - S = 0$ egyenlet gyöke az $\sqrt{S}$. A négyzetgyök reciproka gyakran előfordul különféle numerikus módszerekben, ezért érdekes lehet a kiszámítására gyors eljárást kidolgozni. Természetesen csak közelítő eljárásról lehet szó. A módszer egyik legendás implementációja a Quake III nevű játék C forráskódjában található, amelyet *majdnem szó szerint* idézünk itt:

```

float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y;          // evil floating point bit level hacking
    i = 0x5f3759df - ( i >> 1 ); // what the f&*k?
    y = * ( float * ) &i;
    y = y*(threehalfs-(x2 * y * y)); // 1st iteration
    //y = y*(threehalfs-(x2 * y * y)); // 2nd iteration, this can be removed
    return y;
}

```

A kódban természetesen a 0x5F3759DF konstans kelti fel elsősorban az olvasó figyelmét. Feladatunk, hogy derítsük ki, vajon mi szerepe van pont ennek az értéknek egy egyébként általános módszerben?

5. Bizonyítsuk be, hogy ha $p(x) \in \mathbb{Z}_p$ n -ed fokú polinom, akkor *legfeljebb* n gyöke van.

6. fejezet

Prímtesztelés

Ebben a fejezetben olyan módszerekről lesz szó, amelyek választ adnak arra a döntési kérdésre, hogy egy egész szám prímszám-e. Néhány valószínűségi algoritmust fogunk megismerni, valamint megemlíjtjük a 'PRIMES is in P' cikkről is. Láttuk a korábbi fejezetekben, hogy a prímeket használjuk a moduláris algoritmusoknál is.

6.1. Alapfogalmak

Egy N egész szám prímszám, ha $ab = N$ esetén $a = 1$ vagy $b = 1$.

Az *aritmetika alaptétele* szerint minden pozitív egész szám felírható prímszámok szorzataként.

Továbbá már Euklidész is tudta, hogy végtelen sok prímszám van. Bizonyításának egy picit módosított változata a következő. Tegyük fel, hogy véges sok prímszám van: $p_1 < p_2 < \dots < p_r$. Legyen $N = p_1 \cdot p_2 \cdot \dots \cdot p_r$. Az $N - 1$ egész szám, mivel szintén prímszámok szorzatára bontható, ezért szükségképpen van közös prímosztója az N -nel, ami legyen p_i . Ekkor p_i osztja a különbségüket, 1-et, ami lehetetlen.

6.2. Szita módszer

Eratoszthenész alapján egy lehetséges módszer N prímtesztelése a következő: kezdjük el osztogatni $3, \dots, \sqrt{N}$ -nel és nézzük meg, hogy van-e maradék.

Az algoritmus pszeudokódja a következő [8]:

Bemenet: egy n pozitív egész szám.

Kimenet: 1, ha az n szám prím, 0, ha összetett.

```
if (n == 2) return 1,
if (n mod 2 == 0) return 0,
t = 3; s = 4;
while (s <= n) {
    if (n mod t == 0) return 0;
    else {
        t = t + 2;
        s = s + 2 * t - 1;
        print(t, s);
    }
}
return 1;
```

6.3. Fermat kis tétele

6.3.1. Tétel (Fermat, 1640). *Ha p prím és $a \in \mathbb{N}$ ehhez relatív prím, akkor*

$$a^{p-1} \equiv 1 \pmod{p}.$$

Az a tehát nem osztható p -vel. A tétellel tehát azt tudjuk megmondani, hogy egy egész szám **nem** prímszám. A fenti fölírásban ez p .

A tétel egy másik változata a következő: bármely p prímszámmra teljesül bármely $a \in \mathbb{N}$ egész szám esetén, hogy

$$a^p \equiv a \pmod{p}.$$

Érdekességképpen megjegyezzük, hogy Fermat egyébként nem adott bizonyítást a tételre, csak egy levélben írta azt le. Leibniz bizonyította, de annak pontos dátum nem ismert.

6.3.1. Álprímek

Ha N összetett és

$$a^{N-1} \equiv 1 \pmod{N}$$

valamely a -ra, akkor azt mondjuk, hogy N egy a -alapú **álprím**.

Sarrus 1820-ban az alábbi eredményre jutott: $N = 341$ az $a = 2$ alapra álprím (hiszen $341 = 11 \cdot 31$).

6.3.2. Algoritmus a Fermat kis tétele alapján

A Fermat kis tétele alapján a következő, valószínűségi algoritmus adódik:

Bemenet: N egész

Kimenet: 'összetett' vagy 'valószínűleg prím'

- 1) Legyen $2 \leq a \leq N - 2$ véletlen szám (egyenletes eloszlással választva).
- 2) Számítsuk ki

$$b = a^{N-1} \bmod N,$$

ehhez használjuk az ismételt hatványozás algoritmust a 2.4 szakaszból. Itt a $\bmod$ művelet az 'osztás maradéka'.

- 3) Ha $b \neq 1$, akkor a válasz 'összetett', különben 'valószínűleg prím'.

Tulajdonságok. Az algoritmusra a következő állítások teljesülnek:

- Ha a és N nem relatív prímek, akkor b és N sem azok, ezért az algoritmus ekkor korrekt választ ad.
 - Az ' a és N relatív prímek?' kérdés eldöntésére az euklidészi algoritmust használjuk, végrehajtva azt az 1) és 2) lépés között.
- Ha $\text{Inko}(a, N) = 1$, akkor a 'valószínűleg prím' vagy helyes vagy helytelen válasz.

Példa. Teszteljük le 5 próbálkozással, hogy a 97 prímszám-e. Tegyük fel, hogy a következő eredményeket kaptuk:

a	b	iteráció	jelentés
41	1	1	$41^{96} \equiv 1 \pmod{97}$
32	1	2	$32^{96} \equiv 1 \pmod{97}$
94	1	3	$94^{96} \equiv 1 \pmod{97}$
17	1	4	$17^{96} \equiv 1 \pmod{97}$
73	1	5	$73^{96} \equiv 1 \pmod{97}$

Az algoritmus az összes véletlenszerűen generált a érték esetében igaznak találja az $a \equiv 1 \pmod{n}$ kifejezést, ami azt jelzi, hogy a 97 prímszám, tehát 1 visszatérítési értékkel leáll.

Példa. Annak személetesítésére, hogy az $a^{p-1} \equiv 1 \pmod{p}$ kiszámításához, ahol p egy nagy méretű prímszám nem kell kiszámolnunk az a^{p-1} értékét, hanem annak csak p -vel osztási maradékát, elevenítsük fel az ismételt négyzetre emelés módszerét.

Tegyük fel, hogy $2^{154} \pmod{155}$ értékét akarjuk meghatározni. A 154 bináris alakja: 10011010. A hatványok tehát bináris alakban rendre: 1, 10, 100, 1001, 10011, 100110, 1001101, és 10011010 vagy 10-es számrendszerben: 1, 2, 4, 9, 19, 38, 77, és 154. A következő számításokat kell elvégeznünk:

$$\begin{aligned}
2^1 &\equiv 2 \cdot 1^2 \equiv 2 \pmod{155} \\
2^2 &\equiv 2^2 \equiv 4 \pmod{155} \\
2^4 &\equiv 4^2 \equiv 16 \pmod{155} \\
2^9 &\equiv 2 \cdot 16^2 \equiv 47 \pmod{155} \\
2^{19} &\equiv 2 \cdot 47^2 \equiv 78 \pmod{155} \\
2^{38} &\equiv 78^2 \equiv 39 \pmod{155} \\
2^{77} &\equiv 2 \cdot 39^2 \equiv 97 \pmod{155} \\
2^{154} &\equiv 97^2 \equiv 109 \pmod{155}.
\end{aligned}$$

Először is az eredményből leolvashatjuk, hogy 155 nem prím szám, hiszen ha az lenne, akkor az 1 végeredményt kaptuk volna. Amennyiben a p számnak n bináris számjegye van, az $a^{p-1} \pmod{p}$ kiszámítása $2n$ szorzással valamint n darab maradék művelettel kiszámítható. Ezek műveletigénye $\mathcal{O}(n^2)$, ezért a teljes műveletigény $\mathcal{O}(n^3)$ lesz.

6.3.3. Carmichael számok

Ha az

$$a^{N-1} \equiv 1 \pmod{N}$$

kongruencia igaz minden N -hez relatív prím a -ra, akkor N **univerzális álprím**. Ezek a Carmichael számok. Végtelen sok van belőle, de sokkal ritkábbak, mint a prímszámok. A legkisebb ilyen: 561. Ezek tehát olyan inputok, amelyekre a Fermat teszt rossz választ ad, ezt minden esetben figyelembe kell vennünk az algoritmus használatakor.

Szerencsére a Carmichael számok karakterizációja megadható, amelyet a következő tételben foglaltunk össze.

6.3.2. Tétel. N akkor és csak akkor Carmichael, ha négyzetmentes és $p-1$ osztja $N-1$ -et, ahol p az N prímfaktora, továbbá: N páratlan és legalább 3 prímfaktora van.

Az 561 utáni hat Carmichael szám a következő (az $a \mid b$ jelentése: a osztója b -nek):

$$\begin{array}{lll} 1105 = 5 \cdot 13 \cdot 17 & (4 \mid 1104; & 12 \mid 1104; & 16 \mid 1104) \\ 1729 = 7 \cdot 13 \cdot 19 & (6 \mid 1728; & 12 \mid 1728; & 18 \mid 1728) \\ 2465 = 5 \cdot 17 \cdot 29 & (4 \mid 2464; & 16 \mid 2464; & 28 \mid 2464) \\ 2821 = 7 \cdot 13 \cdot 31 & (6 \mid 2820; & 12 \mid 2820; & 30 \mid 2820) \\ 6601 = 7 \cdot 23 \cdot 41 & (6 \mid 6600; & 22 \mid 6600; & 40 \mid 6600) \\ 8911 = 7 \cdot 19 \cdot 67 & (6 \mid 8910; & 18 \mid 8910; & 66 \mid 8910) \end{array}$$

Nagyobb számokra egyre kevesebb van belőlük: 20.318.200 darab van 1 és 10^{21} között. Ez 50 milliárd számból 1.

További információt a Carmichael számokról a <https://oeis.org/A002997> linken találhatunk az OEIS (Online Encyclopedia of Integer Sequences) oldalán.

6.4. Miller-Rabin teszt

Azokat az egész számokat, amelyek kielégítenek olyan prímtesztet, amelynek minden prímszám eleget tesz, de a legtöbb összetett szám nem, valószínű prímnek nevezzük. A valószínű prímek egy része összetett szám, ezek a 6.3.1. szakaszban tárgyalt álprímek. Az angol nyelvű szakirodalomban a PRP jelölést használják (*probable prime*).

Az ebben a szakaszban ismertetett algoritmus kihasználja, hogy ha p prímszám, akkor minden olyan a szám esetén, amelyre $\text{luko}(a, p) = 1$ és $p - 1 = 2^s r$, ahol r páratlan szám, fennáll a következő összefüggések egyike:

- $a^r \equiv 1 \pmod{p}$, vagy
- létezik olyan j , $0 \leq j \leq s - 1$ úgy, hogy $a^{2^j r} \equiv p - 1 \pmod{p}$.

Legyen $N - 1 = 2^s t$, ahol t páratlan és $s \geq 1$, hiszen a páros számokról könnyű eldönteni, hogy prímszámok-e. Válasszunk egy a véletlen számot, majd számítsuk ki a következőket:

1. $u_0 \equiv a^t \pmod{N}$
2. $u_{i+1} \equiv u_i^2 \pmod{N}$, ahol $i = 1, 2, \dots, s$.
3. Válasz: N valószínűleg prím, ha $u_0 = 1$, vagy valamely i -re ($0 \leq i < s$) az $u_i = -1$

Megjegyezzük, hogy $u_s \equiv a^{2^s t} \equiv a^{N-1}$ ezért ennek értéke $1 \pmod{N}$ ha N egy prímszám. Ez a feltétel az alapja az elfogadásnak, mert ha $u_i \equiv -1$, akkor $u_j \equiv 1$ minden $i + 1 \leq j \leq s$ indexre. Általában a tesztet k -szor egymástól függetlenül elvégezzük, hogy biztosabb választ kapjunk.

Tegyük fel, hogy N prímszám. Akkor mindig azt a választ adjuk, hogy 'valószínűleg prím'. Mivel tudjuk, hogy $u_s \equiv 1 \pmod{N}$, ezért vagy az van, hogy $u_0 \equiv 1 \pmod{N}$ vagy pedig van olyan utolsó u_i , amelyre $u_i \equiv 1 \pmod{N}$ NEM teljesül és $u_{i+1} \equiv u_i^2 \equiv 1 \pmod{N}$. De ha N prím, akkor az $x^2 \equiv 1 \pmod{N}$ egyenlet megoldása az $x \equiv \pm 1$, ezért szükségképpen $u_i \equiv -1 \pmod{N}$, ami azt jelenti, hogy N -et valószínűleg prím-nek deklaráltuk.

Amennyiben N összetett szám, akkor az a véletlenszámok halmazának felével ezt az eredményt kapjuk. Ehhez tegyük fel, hogy $N = p_1 p_2 \dots p_r$, és vegyünk egy $a \pmod{N}$ értéket véletlenszerűen. A kínai maradéktétel szerint ez ugyanaz, mintha az $a_1 \pmod{p_1}, a_2 \pmod{p_2}, \dots, a_r \pmod{p_r}$ értékeket

választottuk volna véletlenszerűen. Ahhoz, hogy az $u_i \bmod N$ értéke ne legyen egyenlő -1 -gyel, annak kell teljesülnie, hogy

$$\begin{aligned} u_i &\equiv -1 \pmod{p_1} \\ u_i &\equiv -1 \pmod{p_2} \\ &\vdots \\ u_i &\equiv -1 \pmod{p_r} \end{aligned}$$

Ha N egy Carmichael szám, akkor az algoritmus legalább $1/2$ valószínűséggel meg is adja N osztóját.

Ha a hiba valószínűségét ε alá akarjuk csökkenteni, akkor $\log_2 \varepsilon^{-1}$ -szer kell végrehajtani. Ha mindig azt a választ kapjuk, hogy 'valószínűleg prím', akkor N prím.

Példa. Teszteljük le, hogy 97 valószínű prímszám-e. Ehhez a következő lépéseket kell elvégezni:

1. Keressük meg d és s értékeket, amelyre $96 = d \cdot 2^s$ alakú, ahol d páratlan szám.
 - Kezdjük $s = 0$ értékkel, ekkor $d = 96$.
 - Növeljük s értékét. Ekkor $d = 3$ és $s = 5$ ezért $96 = 3 \cdot 2^5$
2. Válasszuk a értéket ($1 < a < 97 - 1$). Most az $a = 2$ választással dolgozunk.
3. Számítsuk ki $a^d \pmod{n}$ értékét, azaz $2^3 \pmod{97}$ értékét. Mivel ez nem kongruens $1 \pmod{97}$ folytatjuk a tesztet a következő feltétel ellenőrzésével.
4. Számítsuk ki $2^{3 \cdot 2^r} \pmod{97}$ értékét ($0 \leq r < s$). Amennyiben ez kongruens $96 \pmod{97}$, akkor 97 valószínű prím. Egyébként 97 biztosan összetett.
 - $r = 0$: $2^3 \equiv 8 \pmod{97}$
 - $r = 1$: $2^6 \equiv 64 \pmod{97}$
 - $r = 2$: $2^{12} \equiv 22 \pmod{97}$
 - $r = 3$: $2^{24} \equiv 96 \pmod{97}$

Kaptuk, hogy 97 valószínű prím.

6.5. Mersenne prímek

6.5.1. Tétel. Ha $2^p - 1$ prím, akkor p prím.

A bizonyítást könnyű meggondolni, ugyanis $2^{uv} - 1$ osztható $2^u - 1$ -gyel. Ez abból adódik, hogy ha p összetett, akkor $2^p - 1 = 2^{uv} - 1 = (2^u)^v - 1^v = (2^u - 1)(\dots)$.

Mersenne 1644-ben a következőt állította: $2^p - 1$ prímszám, ha

$$p = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257$$

és minden más $p < 257$ -re összetett.

Ez az állítás sok matematikus érdeklődését keltette föl, így például Euler 1772-ben megmutatta, hogy $2^{31} - 1$ prím, ezzel megcáfolta Mersenne állítását. Lucas száz évvel később bebizonyította, hogy $2^{127} - 1$ prím, de $2^{67} - 1$ nem az. Powers 1911-ben bővítette a listát: $2^{89} - 1$ prím, $2^{107} - 1$ prím, amellyel tehát újabb Mersenne prímeket talált. Ezután Kraichik 1922-ben megmutatta, hogy $2^{257} - 1$ összetett, amellyel tehát Mersenne utolsó számáról mutatta meg az eredeti sejtés hamisságát.

Bár a számítógépek használatával sokkal hatékonyabbak lehetünk, azóta sem találtak *sokkal* több Mersenne prímet.

- 2016.január: 49. Mersenne prím: $2^{74,207,281} - 1$,
- 2017. december 26: 50. Mersenne prím: $2^{77,232,917} - 1$, amely a jegyzet írásának időpontjában egyben a legnagyobb ismert prímszám, nagyjából 23 millió számjeggyel.

Megjegyezzük, hogy 1997 óta az összes Mersenne prímet a GIMPS projekt (Great Internet Mersenne Prime Search) nevű, teljesen elosztott számítógépes rendszerrel keresik.

6.6. AKS algoritmus

Az AKS algoritmus a szerzőkről, három indiai matematikusról (Manindra Agrawal, Neeraj Kayal és Nitin Saxena) lett elnevezve. A 2002-es publikációjuk címe elárulja a lényegét, 'PRIME is in P'. Ez az első olyan eljárás, amely determinisztikus, futási ideje polinomiális és nem alapszik semmilyen hipotézisre. A következő, egyébként régóta ismert azonosságra épül.

6.6.1. Tétel. Az n egész szám akkor és csak akkor prím, ha $\text{Inko}(a, n) = 1$ és

$$(x + a)^n \equiv (x^n + a) \pmod{n}$$

ahol a maradékos osztást a polinom együtthatóin kell elvégezni.

Ez lényegében a Fermat kis tételének általánosítása ($x = 0$).

Példa. Igazoljuk, hogy az 5 prímszám. Ekkor

$$(x + 1)^5 = x^5 + 5x^4 + 10x^3 + 10x^2 + 5x + 1 \equiv (x^5 + 1) \pmod{5},$$

a polinom minden együtthatójának 5-tel való osztási maradéka 0, kivéve az első és utolsó együtthatókat.

Az algoritmus részletes tárgyalása túlmutat a jegyzet keretein, azonban a kapcsolódó Wikipédia oldalon http://en.wikipedia.org/wiki/AKS_primality_test találhatunk egy szép illusztrációt is az algoritmus futására.

6.7. Nagyméretű prímszámok generálása

Valószínűleg prímszámok generálása Az algoritmus bemenete egy k pozitív egész szám és t egy biztonsági paraméter, kimenete pedig egy k hosszúságú, 10-es alapú számrendszerbeli valószínűleg prímszám. Az algoritmus végrehajtásához felhasználjuk a Miller-Rabin teszt eljárást.

```
p = rand(10^(k-1), 10^k)();
if (p mod 2 = 0) p = p + 1;
while (true){
    if (miller_rabin(p, t) = 1) return p;
    p = p + 2;
}
```

Erős prímszámok generálása Egy p prímszámot *erős prímnek* nevezünk, ha léteznek az r, s, q egész számok a következő tulajdonságokkal:

- $p - 1$ -nek van egy nagy prímtényezője, jelöljük ezt r -rel,
- $p + 1$ -nek van egy nagy prímtényezője, jelöljük ezt s -sel,

- és $r - 1$ -nek van egy nagy prímtényezője, jelöljük ezt q -val.

A következő eljárás használja az imént ismertetett prímszám generátort, amelyre `random_prim` függvényként hivatkozunk a kódban. A bemenet itt is egy k pozitív egész szám és t biztonsági paraméter, a kimenet pedig egy k hosszúságú, 10-es számrendszerbeli erős prímszám.

```
k1 = floor(k/2); // alsó egészrész
q = random_prim(k1, t);
i = 1;
while (true){
    r = 2 * i * q + 1;
    if (miller_rabin(r, t) == 1) break;
    i++;
}
s = random_prim(k1, t);
mh = mod(power(s,r-2), r); // moduláris hatványozással számítva!
p0 = 2 * s * mh - 1;
j = 1;
while (true){
    p = p0 + 2 * j * r * s;
    if (miller_rabin(p, t)==1) return p;
    j++;
}
```

6.8. Feladatok

1. A *levágható prím* olyan prímszám, amelynek a számjegyeit levágva rendre prímszámokat kapunk. Például a 9137 prímszám egy balról levágható prím, hiszen 137, 37 és 7 mind prímszámok. Ugyanakkor a 7393 prímszám egy jobbról levágható prím, mert 739, 73, 7 is mind prímszámok. Összesen 4260 balról levágható- és 83 jobbról levágható prímszámot ismerünk. Feladatunk, hogy írjunk egy eljárást levágható prímelek keresésére.
2. Egy prímszámot *mírp*-nek nevezzük, ha visszafelé olvasva egy másik prímszámot ad. Írjunk eljárást, amely mírpeket keres.
3. Jelölje p_n az n -edik prímszámot. Ekkor $p_n\#$ *primoriálist* az első n darab prímszám szorzataként definiáljuk:

$$p_n\# = \prod_{i=1}^n p_i,$$

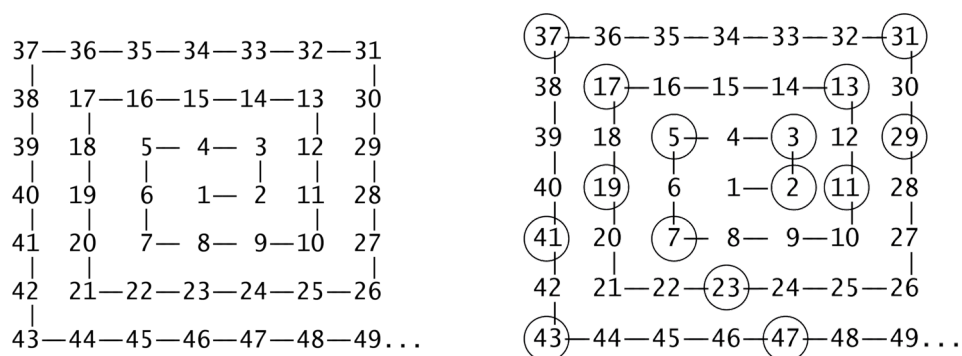
ahol tehát p_i az i -edik prímszám.

A *primoriálisprím* egy olyan prímszám, amely $p_n\# \pm 1$ alakban felírható. Az eddig ismert legnagyobb primoriálisprím a $1098133\# - 1$ ($n = 85586$), amelynek 476.311 számjegye van. Írjunk eljárást, amely primoriálisprímeket keres.

4. A *prímhézag* két egymás utáni prímszám különbsége, melynek jelölése: $g_n = p_{n+1} - p_n$. Bizonyítsuk be, hogy

$$p_{n+1} = 2 + \sum_{i=1}^n g_i.$$

5. Az Ulam spirál a prímszámok furcsa mintázatát mutató megjelenítés. Az alábbi ábra bal oldalán az egész számok spirális elrendezését látjuk, míg a jobb oldalon pedig ebben a sorozatban a prímszámokat tartalmazza¹:



Készítsünk számítógépes implementációt az Ulam spirál nagyfelbontású kirajzolására.

¹forrás: <https://hu.wikipedia.org/wiki/Ulam-spirál>

7. fejezet

Prímfaktorizálás

Ebben a fejezetben az alábbi fontos tételből indulunk ki.

7.0.1. Tétel (Számelmélet alaptétele). *Minden pozitív egész n egyértelműen írható*

$$n = p_1 \cdot p_2 \cdot \dots \cdot p_t, \quad p_1 \leq p_2 \leq \dots \leq p_t,$$

alakban, ahol mindegyik p_k prímszám.

Ez alapján a következő feladatokat vizsgáljuk:

- (i) meghatározni p_t -t, a legnagyobb prímtényezőt,
- (ii) meghatározni t értékét, azaz a prímtényezők számát,
- (iii) megtalálni a fenti fölbontást.

7.1. Legnagyobb prímtényező

Foglalkozzunk először röviden az első kérdéssel: mekkora szokott p_t lenni?

Karl Dickman, 1930: mi annak a valószínűsége, hogy egy 1 és x közötti véletlenszám legnagyobb prímosztója $\leq x^\alpha$? Azt találta, hogy $x \rightarrow \infty$ esetén ez a valószínűség az $F(\alpha)$ határeloszláshoz tart, ahol

$$F(\alpha) = \int_0^\alpha F\left(\frac{1}{1-t}\right) \frac{dt}{t}, \quad (0 \leq \alpha \leq 1); \quad F(\alpha) = 1 \text{ ha } \alpha > 1.$$

Ha $1/2 \leq \alpha \leq 1$, akkor ez a formula az

$$F(\alpha) = 1 - \int_\alpha^1 F\left(\frac{1}{1-t}\right) \frac{dt}{t} = 1 - \int_\alpha^1 \frac{dt}{t} = 1 + \ln \alpha$$

alakra egyszerűsödik. Így például $\alpha = 1/2$ -re kapjuk, hogy $1 - F(1/2) = \ln 2$, vagyis durván 69 százalék annak a valószínűsége, hogy egy x -nél kisebb véletlen egész szám osztható egy $\sqrt{x}$ -nél nagyobb prímmel!

7.2. Prímtényezők száma

Intenzíven vizsgálták t értékét is, az összes prímosztók számát.

Nyilván $1 \leq t \leq \log_2 n$, ezeket a korlátokat azonban csak ritkán éri el. Megmutatható, hogy ha n -et véletlenül választjuk 1 és x között, akkor bármely rögzített c -re, annak a valószínűsége, hogy $t \leq \ln \ln x + c\sqrt{\ln \ln x}$, a

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^c e^{-u^2/2} du$$

értékhez tart, ha $x \rightarrow \infty$. Más szavakkal: t lényegében normális eloszlású.

7.3. Prímfelbontás

Jelen ismereteink szerint ezek nehezebb feladatok, mint a prímteszt. Tehát az egy megoldatlan probléma, hogy vajon létezik-e polinomiális idejű algoritmus a prímfelbontásra

7.3.1. Próbálgatásos osztás

Osszuk és bontuk tényezőkre: vagyis osszuk el n -et a

$$p = 2, 3, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35, \dots$$

számokkal (próbaosztókkal) mindaddig, amíg eljutunk az

$$n \bmod p = 0$$

esetig. Ekkor p az n legkisebb prímosztója, és az eljárást folytatjuk az $n = n/p$ -vel.

A fenti sorozatban az első három tag után felváltva mindig 2-vel illetve 4-gyel növelünk.

Ha azt találjuk, hogy

$$n \bmod p \neq 0 \text{ és } \lfloor n/p \rfloor \leq p \tag{7.1}$$

akkor n prím szám. Ennek belátásához tegyük fel, hogy n összetett, mégis teljesül (7.1). Akkor

$$n \geq p_1^2,$$

ahol p_1 az n első prímtényezője. Ugyanakkor $p_1 > p$ miatt¹

$$\begin{aligned} p_1^2 &\geq (p+1)^2 > p(p+1) > p^2 + (n \bmod p) \\ &\geq \lfloor n/p \rfloor p + (n \bmod p) = n. \end{aligned}$$

Példa. Legyen $n = 25852$. Azonnal adódik, hogy $n = 2 \cdot 12926$, így $p_1 = 2$. Ezután $12926 = 2 \cdot 6463$, tehát $p_2 = 2$. Most $n = 6463$ nem osztható 2, 3, 5, ..., 19 egyikével sem, viszont $n = 23 \cdot 281$, tehát $p_3 = 23$. Végül $281 = 12 \cdot 23 + 5$ és $12 \leq 23$, vagyis $p_4 = 281$.

Az $n = 25852$ prímtényezőinek meghatározásához összesen 12 osztási műveletre volt szükségünk. Ha a 25849 számot próbáltuk volna fölbontani (ami egyébként prím szám), akkor legalább 38 osztást kellett volna elvégeznünk. Ez talán jól illusztrálja, hogy az algoritmus futási ideje kb. $\max(p_{t-1}, \sqrt{p_t})$ -vel arányos. Ha n kicsi, akkor érdekesebb táblázatból kinézni. Például, ha n kisebb, mint egymillió, akkor csak az ezer alatti 168 darab prímot kell használnunk (ugyanis elég a próbaosztást $\sqrt{n}$ -ig elvégezni).

Beiktathatunk egy prímtesztet is (lásd előző fejezet), ez általában gyorsít. Ilyenkor a futási idő arányos p_{t-1} -gyel, ami tehát n második legnagyobb prímtényezője.

A Próbálgatásos osztás algoritmus általában talál néhány apró prímtényezőt, majd hosszúra nyúló kutatásba kezd a fennmaradó nagyok után. Ha $n = (2^{521} - 1) \cdot (2^{607} - 1)$ és faktorizálandó a $10n$, akkor a $2 \cdot 5 \cdot n$ felbontást hamar megtalálja, de utána nagyon sokáig eredménytelenül keresgél.

¹ott még nem tart az algoritmus a p értékekkel, hogy elérje a p_1 értékét, hiszen az (7.1) feltétel szerint a p NEM osztja n -et, viszont tudjuk, hogy p_1 osztja n -et

Prímek száma adott korlátig. Legyen $\pi(x)$ az x -ig terjedő prímek száma tehát $\pi(2) = 1, \pi(10) = 4$, stb.

Charles de la Vallée Poussin 1899-ben a következőt állította:

$$\pi(x) = \int_2^x \frac{dt}{\ln t} + \mathcal{O}(xe^{-A\sqrt{\log x}})$$

alkalmas $A > 0$ -val.

Bernhard Riemann sejtése 1859-ből:

$$\pi(x) = L(x) - \frac{1}{2}L(x^{1/2}) - \frac{1}{3}L(x^{3/2}) + \dots,$$

ahol $L(x) = \int_2^x \frac{1}{\ln t} dt$. Ezt a sejtést Littlewood 1914-ben megcáfolta: megmutatta, hogy alkalmas C pozitív konstanssal

$$\pi(x) > L(x) + C\sqrt{x} \cdot \log \log \log x / \log x$$

teljesül végtelen sok x -re.

Ezek az eredmények mind azt jelzik, hogy a prímszámok rejtélyes valamik, eloszlásuk megismeréséhez nagyon mély matematikai elméletekre van szükség.

7.3.2. Pollard-féle Monte Carlo módszer

Nézzünk akkor most egy módszert, amely valószínűségi alapon keresi egy összetett szám prímtényező felbontását.

Az algoritmus az alábbi észrevételen alapszik. Ahhoz, hogy két szám, x és y , kongruens modulo p legyen 0,5 valószínűséggel,

$$1.177 \cdot \sqrt{p}$$

számot kell véletlenszerűen kiválasztani².

Legyen $x \equiv y \pmod{p}$. Ha p az n egész prímfaktora, akkor

$$p \leq \text{lko}(x - y, n) \leq n$$

mivel p osztója az $(x - y)$ -nak³ és n -nek egyaránt. Egy valószínűségi módszert mutatunk.

Feltételezve, hogy a p prímszám osztója n -nek, ezért az $x_0, x_1, x_2, \dots$ sorozatban, ahol $x_i = f(x_{i-1})$ és $f: \mathbb{Z}_p \rightarrow \mathbb{Z}_p$, ismétlődéseket fogunk találni, azaz bizonyos k értékre

$$x_{2k} \equiv x_k \pmod{p}$$

Természetesen p -t nem ismerjük, ezért a modulo n szerinti maradékos osztást vizsgáljuk. Tehát most $f: \mathbb{Z}_n \rightarrow \mathbb{Z}_n$ alakú, hiszen csak n -et ismerjük, csak ezt a függvényt tudjuk legyártani. Ilyenkor is kapunk ismétlődést az f által generált sorozatban, és bár ezek nem ugyanazok a számok, mint amit a mod p -vel számolt függvénnyel kapnánk, de attól még hívjuk őket x_k -nak és x_{2k} -nak. Azt biztosan tudjuk, hogy $x_k - x_{2k}$ különbséget a p osztja. Tehát bár a p értékét továbbra sem tudjuk, de föltettük, hogy $p|n$.

Amennyiben

$$1 < d = \text{lko}(x_k - x_{2k}, n) < n$$

teljesül, akkor d egy nem triviális osztója n -nek.

²Ez ugyanaz, mint a születésnap probléma: mekkora házibulit kell rendeznünk ahhoz, hogy legalább 0,5 valószínűséggel legyen két azonos születésnapú vendégünk? Meglehető válasz: 23 ember elég.

³hiszen a feltevésből következik, hogy $x - y \equiv 0 \pmod{p}$

Azt szeretnénk elérni, hogy az $x_0, x_1, \dots$ sorozat mutasson valami véletlenszerűséget. Ez az $f(x)$ polinom választásán múlik. Az $f(x) = ax + b$ bizonyítottan nem jó. A következő legegyszerűbb eset: $f(x) = x^2 + 1$ Azt nem tudjuk, hogy ez a függvény elég véletlen-e, ugyanakkor csak $(p+1)/2$ különböző értéket vesz föl mod p , ami jó tulajdonság

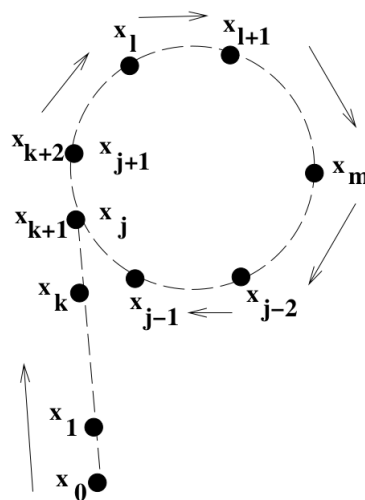
A módszer pszeudokódja az alábbi:

```

1: a=2; b=2;
2: while (true) {
3:   a = (a * a + 1) mod n; // f(x) = x^2 + 1 (mod n)
4:   b = (b * b + 1) mod n;
5:   b = (b * b + 1) mod n; // x = f(f(x))
6:   d = lnko(a-b, n);
7:   if (1 < d && d < n) return d;
8:   if (d == n) return n; // kudarc
9: }
```

A 4. és 5. sorok együtt biztosítják, hogy a b sorozat kétszer gyorsabban megy, mint az a . Ha $\text{lnko}(a-b, n) = 1$, akkor folytatódik a ciklus.

A tapasztalat azt mutatja, hogy nagyobb n -ekre elég jól működik, kicsi n -re nem érdemes használni. Hívják ρ módszernek is, a ciklikusság miatt. Ennek magyarázatához lásd a 7.1. ábrát, ahol a $x_i \bmod p$ sorozat lépéseit mutatjuk be.



7.1. ábra. Pollard faktorizációs eljárást ρ -módszernek is nevezzük.

Ha a módszer kudarcot vall, akkor kipróbálhatjuk az $f(x) = x^2 + c$ függvényt is, ilyenkor a $c \neq 0, 1, -2$.

Ezzel az algoritmussal faktorizálták a 8. Fermat számot. Ezek az $F_n = 2^{2^n} + 1$, alakú egészek, amely $n = 8$ esetben összetett számot ad. Az $n = 8$ -ra azért működött hatékonyan, mert az F_8 két prímszám szorzata: $p_1 p_2$, és $p_1 = 12389263661552897$ sokkal kisebb, mint p_2 .

Rendkívül érdekes tény, hogy csak $n = 1, 2, 3, 4$ -re tudjuk, hogy F_n prímszám.

Példa. Faktorizáljuk az $n = 1387 = 19 \cdot 73$ számot. Legyen $x_0 = 0$ és használjuk az $y_i \equiv x_{2i} - x_i \bmod 1387$ függvényt.

i	x_i	x_{2i}	y_i	$\text{lnko}(y_i, 1387)$
1	2	1	1	1
2	2	26	24	1
3	5	620	615	1
4	26	582	556	1
5	677	829	152	19

Az ötödik lépés végén a 19 prímtényezőt meghatároztuk.

7.3.3. Fermat-faktorizáció

A Fermat-faktorizáció a következő megfigyelésen alapszik. Tetszőleges páratlan szám felírható két négyzetszám különbségeként, azaz $n = a^2 - b^2$. Az algoritmus pszeudokódja az alábbi.

```
Bemenet: n páratlan szám
a1 = ceil(sqrt(n)); // felső egészrész
b1 = a1 * a1 - n;
while (negyzetszam(b1)) {
    a1++;
    b1 = a1 * a1 - n;
}
return a1 + sqrt(b1);
```

Láthatjuk, hogy az algoritmus szisztematikusan keres olyan a és b értékeket, amelyre $n = a^2 - b^2$.

A Fermat-faktorizáció jól működik, ha n két azonos nagyságrendű osztóval rendelkezik.

Négyzetszámok meghatározására az alábbi karakterizációt használhatjuk: az utolsó két jegyük

$$00, s1, s4, 25, t6, s9$$

alakú, ahol s páros és t páratlan.

Példa. Legyen $n = 517$. Az algoritmus az alábbi lépéseket hajtja végre.

a1	b1	megjegyzés
23	12	$\text{ceil}(\sqrt{517}) = 23$
24	59	
25	108	
26	159	
27	212	
28	267	$324 = 18 \cdot 18$, négyzetszám
29	324	

Az algoritmus itt a $29+18 = 47$ visszatérési értékkel leáll, azaz $517 = 29^2 - 18^2 = (29-18)(29+18) = 11 \cdot 47$.

7.3.4. Euler-faktorizáció

Az Euler módszer alapja, hogy megkeresse az adott számot két négyzetszám összegeként kifejezve, mégpedig azt két különbözőképpen megadva. Például az 1000009 felírható $1000^2 + 3^2$ vagy $972^2 + 235^2$ alakban, ahonnan Euler módszere az $1000009 = 293 \cdot 3413$ felbontást adja.

A módszer a Fermat-faktorizációnál általában gyorsabb, ha a faktorok nincsenek közel egymáshoz. Ugyanakkor fontos megjegyeznünk, hogy amennyiben valamelyik prímfaktor $4k + 3$ alakú, akkor a módszer nem működik.

Legyen tehát $n = a^2 + b^2 = c^2 + d^2$ alak ismert, ebből n -et két négyzet-összeg szorzataként adjuk meg. Először

$$a^2 - c^2 = d^2 - b^2$$

ahonnan kapjuk, hogy

$$(a - c)(a + c) = (d - b)(d + b) \quad (7.2)$$

Legyen most $k = \text{lko}(a - c, d - b)$ és $h = \text{lko}(a + c, d + b)$ azaz léteznek olyan l, m, l', m' konstansok, amelyre

$$(a - c) = kl, \quad (d - b) = km, \quad \text{és } \text{lko}(l, m) = 1,$$

valamint

$$(a + c) = hm', \quad (d + b) = hl', \quad \text{és } \text{lko}(l', m') = 1.$$

Ezeket behelyettesítve az (7.2) képletbe kapjuk, hogy

$$klhm' = kmhl'$$

A közös tagokat kiejtve adódik, hogy

$$lm' = l'm$$

Használjuk ki, hogy (l, m) és (l', m') relatív prímekből álló párok, tehát

$$l = l' \quad \text{és} \quad m = m'$$

Innen

$$(a - c) = kl, \quad (d - b) = km, \quad (a + c) = hm, \quad (d + b) = hl.$$

Láthatjuk, hogy $\text{lko}(a + c, d - b)$ és $l = \text{lko}(a - c, d + b)$. A Bahmagupta–Fibonacci azonosságot⁴ alkalmazva kapjuk, hogy

$$\begin{aligned} (k^2 + h^2)(l^2 + m^2) &= (kl + hm)^2 + (km - hl)^2 \\ &= ((a - c) + (a + c))^2 + ((d - b) - (d + b))^2 \\ &= (2a)^2 + (2b)^2 \\ &= 4n. \end{aligned}$$

Mivel minden faktor két négyzetszám összege, ezért az egyikük páros számpárokat tartalmaz: vagy (k, h) vagy pedig (l, m) . Tegyük fel, hogy a (k, h) páros. A faktorizáció eredménye ekkor

$$n = \left(\left(\frac{k}{2} \right)^2 + \left(\frac{h}{2} \right)^2 \right) (l^2 + m^2).$$

7.4. Faktorizáló algoritmusokról

A prímfaktorizáló eljárásokat alapvetően két kategóriába szokás sorolni.

- A speciális célú kategóriába azok az algoritmusok tartoznak, amelyeknek a futási ideje a faktorizálandó egész valamelyik prímtényezőjének tulajdonságától (mérete, speciális alakja, stb) függ. A tényleges futási idő algoritmusonként általában különböző. Ebbe a kategóriába tartoznak azok is tehát, amelyek a legkisebb prímfaktor méretétől függő futási idővel rendelkeznek:

⁴ $(a^2 + b^2)(c^2 + d^2) = (ac - bd)^2 + (ad + bc)^2 = (ac + bd)^2 + (ad - bc)^2$

- a 7.3.1. szakaszban tárgyalt próbálgatásos módszer,
- a 7.3.2. szakaszban tárgyalt Pollard-féle ρ -módszer,
- a 7.3.3. szakaszban tárgyalt Fermat-módszer,
- valamint a 7.3.4. szakaszban tárgyalt Euler-faktorizáció.

Megjegyezzük, hogy természetesen további eljárások is léteznek.

- Az általános célú kategóriába azok az algoritmusok tartoznak, amelyek futási ideje a faktorizálandó egész méretétől függ. Ilyen algoritmus például a Dixon eljárás, amely a következő ötleten alapszik. Mint azt láttuk, a Fermat-faktorizációban olyan a és b egészeket keresünk, amelyekre $n = a^2 - b^2$. Dixon eljárásában elég olyan számpárt találni, amelyekre $a^2 \equiv b^2 \pmod{n}$. Például az $n = 84923$ esetében a keresést 292-nél kezdve (ez az első olyan szám, amely $\sqrt{n}$ -nél nagyobb) találjuk, hogy $505^2 \bmod 84923 = 256$, amely pontosan a 16 négyzete. Ezért $(505 - 16)(505 + 16) = 0 \bmod 84923$. Ha kiszámítjuk az n és $505 - 16$ legnagyobb közös osztóját, akkor kapjuk, hogy az 163, amely az n egyik prímfaktora. Az algoritmusban használunk egy alkalmas B korlátot, amelynek választása alapvetően tetszőleges, ugyanakkor túl kicsi korlát esetén csökken annak a valószínűsége, hogy nemtriviális prímfaktort találjuk, míg túl nagy korlát esetén nagyon sokáig fut az eljárás, mert túl sok ellenőrzést kell elvégezni. Például 292 darab olyan négyzetszám van, amely kisebb, mint 84923, ebből 662 darab 84923-nél kisebb olyan van, amelynek prímfaktorai a 2, 3, 5 vagy 7 prímszámok, és 4767 darab szám van, amelynek prímfaktorai 30-nál kisebbek. Az általános fogalom itt a B -sima szám, amelyre teljesül, hogy egyik prímfaktora sem nagyobb, mint B .

Megemlíttük még, hogy a kvantumszámítógépekre is készült egy faktorizáló eljárás, amely Shor-algoritmus néven ismert.

7.5. További megjegyzések

7.5.1. Szemiprímek

Adott számjegyű számok közül nem mindegyiknek ugyanolyan nehéz a prímfaktorizálása. Ezek közül különösen fontos halmazt alkotnak a szemiprímek, amelyek az $n = pq$ alakú egészek, ahol p és q prím. Más néven ezek az RSA számok, a jelenleg ismert eljárásoknak ezek a legnehezebben felbontható számok.

Az 'RSA factoring challenge' egy 1991-2007 között zajló, pénzdíjas verseny volt, amelyben megadott összetett számok szemiprím felbontását kellett megkeresni. Bár a verseny már nem aktív, a keresés nem állt le. A legnagyobb kihívás az RSA-2048, amelynek 617 decimális számjegye van (2.048 bit), egyelőre sikertelen a felbontás megtalálása. A legnagyobb sikeresen faktorizált RSA szám 232 decimális számjegyű (768 bit), ezt 2009-ben találták meg.

7.5.2. Ikerprímek

Az ikerprímek azok a prím számpárok, amelyeknél a prímhézag pontosan 2. Bár nagyméretű prímek esetén a prímhézag általában egyre nagyobb, a mai napig nyitott kérdés, hogy vajon végtelen sok ikerprímszám van-e.

7.6. Feladatok

1. Legyenek m és n relatív prímek. Faktorizálás nélkül mutassuk meg, hogy $m|N$ és $n|N$ fennállásából következik, hogy $mn|N$. Faktorizálás felhasználásával könnyebb a bizonyítás?

2. Egy természetes számot k -majdnem prímszámnak nevezünk, ha pontosan k darab prímfaktora van. Írjunk eljárást, amely ilyen számokat keres adott k értékre.

3. Legyen $\phi(n)$ azon egészek száma, amelyek relatív prímek n -hez (Euler-függvény). Mutassuk meg, hogy pozitív n egészre

$$n = \sum_{d|n, d>0} \phi(d)$$

teljesül.

4. A Lucas-Lehmer teszt a 6.5. szakaszban tárgyalt Mersenne prímek keresésére szolgál. A teszt a következő. Legyen p páratlan prím. A $2^p - 1$ Mersenne szám akkor és csak akkor prímszám, ha $2^p - 1$ osztja $S(p - 1)$ -et, ahol

$$S(n + 1) = \begin{cases} (S(n)^2 - 2) & \text{ha } n > 0, \text{ és} \\ 4 & \text{ha } n = 0. \end{cases}$$

Írjunk programot, amely a beépített típusok felhasználásával kiszámítja a Mersenne prímeket. Legfeljebb a 47. ilyen prímig menjünk el.

5. Írjunk hatékony Mersenne szám faktorizációs eljárást.
6. Készítsük el a 7.3.4. szakaszban tárgyalt Euler-faktorizáció implementációját.

8. fejezet

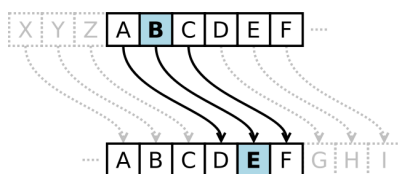
Titkosítás, véletlenszámok

8.1. Titkosítás

A feladat, amivel ebben a részben foglalkozunk a következőképpen fogalmazható meg. B egy üzenetet akar küldeni A -nak úgy, hogy C , aki hallgatózik, ne értse az üzenetet. A rendelkezik egy kulccsal, ami a rejtjelezést megfejtí.

8.1.1. Cézár-rejtjel

Az ötlet rendkívül egyszerű: toljuk el az ábécé karaktereit k lépéssel, természetesen cirkulárisan. Az $k = 4$ eltolást a 8.1. ábra illusztrálja.



8.1. ábra. Cézár-rejtjelezés $k = 4$ eltolással. Forrás: Wikipédia.

8.1.2. One-time pad

Arra a kérdésre ad *igen* választ, hogy létezik-e tökéletes titkosítási eljárás, azaz olyan eljárás, melyben a titkosított szöveg ismerete alapján semmiféle információt nem lehet megtudni a nyílt szöveg tartalmáról.

A módszer a következő: legyen az üzenet n hosszúságú egy k ábécé felett. A kulcs is n hosszúságú, amelyet betűnként hozzáadunk $\bmod k$.

A módszer nagyon biztonságos, de a kulcs túl hosszú, amennyiben az üzenet is hosszú.

Példa. Tegyük fel, hogy 26 elemű az ábécénk és a kulcsunk az XMCKL karaktersorozat. Elküldendő szó: HELLO. Az üzenet titkosításának lépéseit a 8.1. táblázat mutatja, míg a visszafejtését pedig a 8.2. táblázat.

A kulcsok előállításához használhatunk (pszeudo-)véletlenszám generátort (lásd a 8.2. szakaszt ebben a fejezetben). Amennyiben A és B ugyanazt a kiindulási értéket (*seed*-et) használja, akkor ugyanazt a sorozatot kapják. A módszer jellemzője, hogy mindkét félnél tehát ugyanaz a kulcs van. Az ilyen protokollokat *szimmetrikusnak* nevezzük.

Ettől különbözik a következő szakaszban tárgyalt Diffie-Hellman protokoll (1976), amely nyilvános kulcsú titkosítást tesz lehetővé.

8.1. táblázat. A HELLO üzenet titkosítása 26 elemű ábécével és az XMCKL kulcs használatával

	H	E	L	L	O	üzenet
	7 (H)	4 (E)	11 (L)	11 (L)	14 (O)	üzenet
+	23 (X)	12 (M)	2 (C)	10 (K)	11 (L)	kulcs
=	30	16	13	21	25	üzenet + kulcs
=	4 (E)	16 (Q)	13 (N)	21 (V)	25 (Z)	üzenet + kulcs (mod 26)
	E	Q	N	V	Z	titkosított szó

8.2. táblázat. A HELLO üzenet visszafejtése 26 elemű ábécével és az XMCKL kulcs használatával

	E	Q	N	V	Z	titkosított szó
	4 (E)	16 (Q)	13 (N)	21 (V)	25 (Z)	titokszó
–	23 (X)	12 (M)	2 (C)	10 (K)	11 (L)	kulcs
=	-19	4	11	11	14	titokszó – kulcs
=	7 (H)	4 (E)	11 (L)	11 (L)	14 (O)	titokszó – kulcs (mod 26)
	H	E	L	L	O	üzenet

Diffie-Hellman protokoll

A protokoll két külön kulcsot használ, egyet a titkosításhoz és egyet a megfejtéshez. Ez tehát *aszimmetrikus* protokoll. A módszer illusztrálásához tekintsük a következő példát.

Példa. Alice és Bob megegyeznek, hogy a $p = 23$ prímszámot és a $g = 5$ bázist használják. Alice választ egy titkos egész számot: $a = 6$, majd elküldi Bob-nak az $A = g^a \mod p$ értéket:

$$A = 5^6 \mod 23, \quad A = 15.625 \mod 23 \Rightarrow A = 8.$$

Bob is választ egy titkos egész számot: $b = 15$, majd elküldi Alice-nek a $B = g^b \mod p$ értéket:

$$B = 5^{15} \mod 23, \quad B = 30.517.578.125 \mod 23 \Rightarrow B = 19.$$

Alice ezután kiszámolja az $s = B^a \mod p$ értéket:

$$s = 19^6 \mod 23, \quad s = 47.045.881 \mod 23 \Rightarrow s = 2$$

Bob kiszámolja az $s = A^b \mod p$ értéket:

$$s = 8^{15} \mod 23, \quad s = 35.184.372.088.832 \mod 23 \Rightarrow s = 2$$

Alice és Bob közös titka tehát $s = 2$.

Nyilvánvaló, hogy ha valaki tudja a két személyes egészet (6-ot és 15-öt), akkor kiszámíthatja s értékét:

$$s = 5^{6 \cdot 15} \mod 23$$

$$s = 5^{15 \cdot 6} \mod 23$$

$$s = 5^{90} \mod 23$$

$$s = 807.793.566.946.316.088.741.610.050.849.573.099.185.363.389.551.639.556.884.765.625 \mod 23$$

$$s = 2$$

Vegyük észre, hogy ebben a számolásban a moduláris hatványozást használjuk, amelynek hatékony kiszámítását a 2.4. szakaszban tárgyaltuk.

A protokoll alkalmazásánál természetesen sokkal nagyobb a , b és p értékeket kell használni. A példában a $g^{ab} \bmod 23$ műveletnek csak 23 különböző eredménye lehet, ez egyszerűen végigpróbálgható.

Azonban ha p legalább 300 jegyű prím, a és b pedig 100 jegy hosszú, akkor a megtalálásához az összes rendelkezésre álló számítógép sem lenne elegendő, pedig ismerhetjük: $g, p, g^b \bmod p$ és $g^a \bmod p$ értékeit. Ezt *diszkrét logaritmus* problémának nevezzük, amit tehát véges ciklikus testek felett értelmezünk.

Megjegyezzük, hogy egyébként g -nek nem kell nagynak lennie, sokszor elég a 2,3 vagy 5.

Diszkrét logaritmus. Legyen $\mathbb{Z}_m$ az egészek modulo m véges halmaza. A $\mathbb{Z}_m^*$ az *invertálható* elemek halmaza, azaz olyan $a \in \mathbb{Z}_m$, amelyre létezik $b \in \mathbb{Z}_m$ úgy, hogy

$$ab \equiv 1 \pmod{m}.$$

Ezek egyébként azok az elemek, amelyekre $\text{lko}(a, m) = 1$. Az az eset az érdekes most, amikor m egy p prímszám, ekkor $\mathbb{Z}_p^* = \{1, 2, \dots, p-1\}$.

Egy g számot *generátornak* nevezünk, ha a $g, g^2, g^3, \dots, g^{p-1} \bmod p$ tartalmazza az összes $1, 2, \dots, p-1$ számot. Minden p prímhez létezik generátor. Nagyon sok g generátor létezik, de ezek megtalálására nincs ismert determinisztikus polinomiális idejű eljárás. Sőt, még annak a letesztelése, hogy egy adott szám generátor-e sem megoldott determinisztikus polinomiális időben.

Például a $p = 13$ a $\mathbb{Z}_p^*$ generátora a $g = 2$, ugyanakkor a $g = 4$ nem az.

Ezen definíciók felhasználásával a *diszkrét logaritmus* meghatározása a következő. Legyenek adottak az y, g és p számok, ahol p prím és g a $\mathbb{Z}_p^*$ generátora, keressük meg azt az x -et, amelyre

$$g^x \equiv y \pmod{p}$$

Ez lesz az y szám g alapú diszkrét logaritmus modulo p .

8.2. Véletlenszám generálás

Tekintsük a következő, Java nyelven írt kódot:

```
public static void main(String ... args) {
    System.out.println(randomString(-6225973)+' '+
        randomString(1598025));
}

public static String randomString(int seed) {
    Random rand = new Random(seed);
    StringBuilder sb = new StringBuilder();
    for (int i=0; i<5; i++)
        sb.append((char) ('a' + rand.nextInt(26)));
    return sb.toString();
}
```

Vajon mi lesz a kimenete?

A példa jól illusztrálja azt a tényt, hogy bár látszólag egy olyan kódot mutattunk, amelynek kimenetétől azt várjuk, hogy mutasson valami véletlenszerűséget, az mégis teljesen determinisztikus.

8.2.1. Véletlenszámok előállítása külső eszközök felhasználásával

Tegyük fel, hogy szükségünk van egy véletlen számjegyre és ezt nem számítógépen akarjuk előállítani. Az alábbi módszerek közül melyek alkalmasak erre? A következő lista a [1] könyvből származik:

1. Kinyitunk találomra egy telefonkönyvet és az első telefonszám egyes helyiértékű számjegyét használjuk.
2. Ugyanaz, mint az előbb, de most az *oldalszám* egyes helyiértékű jegyét tekintjük.
3. Feldobunk egy szabályos ikozaédert, amelynek a 20 oldalára rendre a $0, 0, 1, 1, \dots, 9, 9$ számok vannak írva és a legfelső lapon lévő számot használjuk.
4. Egy Geiger-Müller-számlálót egy percig radioaktív sugárzásnak teszünk ki, és a számláló egyes helyiértékű számjegyét használjuk. Feltesszük, hogy a számláló tízes számrendszerben számlál és eredeti tartalma nulla.
5. Rápillantunk a karóránkra. Ha a másodpercmutató $6n$ és $6(n+1)$ másodperc között áll, akkor az n számjegyet választjuk.
6. Megkérjük egy barátunkat, hogy mondjon egy véletlen számjegyet.
7. Megkérjük egy ellenségünket, hogy mondjon egy számjegyet véletlenszerűen és ez a számot használjuk.
8. Tegyük fel, hogy tíz ló indul egy versenyen, és semmit sem tudunk a képességeikről. A lovakhoz valamilyen módon hozzárendeljük a $0 - 9$ számjegyeket és a győztes ló számát használjuk.

8.2.2. Neumann négyzetközép

Neumann János 1946-ban a következő, *négyzetközép-módszernek* nevezett eljárást javasolta. Induljunk ki egy tetszőleges, lehetőleg sok számjegyet tartalmazó számból. Az utolsó számot a sorozatban emeljük négyzetre és ennek a középső jegyei alkossák a következő számot.

Ha tehát például 10 jegyű számokra van szükségünk, és az utolsó szám 5 772 156 649, akkor ennek négyzete

$$33317792380594909201,$$

és így a következő szám 7 923 805 949 lesz.

Sajnos a módszer nem igazán jó: általában hamar periodikussá válik. A periodikusság nem meglepő, hiszen csak véges sok 10 jegyű szám van, és amint egy szám megismétlődik, a sorozat periodikus lesz. Az igazi kérdés az, hogy ez milyen hamar következik be. Sajnos a tapasztalat szerint a módszerben általában pár száz lépés után a periodikusság bekövetkezik.

Vegyük észre, hogy a módszer úgy javasol véletlenszerű sorozat gyártását, amelynek minden tagját egyértelműen meghatározza az előtte levő. Ez önmagában nem feltétlenül probléma, viszont az biztos, hogy csak azzal az elvárással élhetünk, hogy '*nem véletlen, de annak látszik*' típusú sorozatot gyártunk. Az alábbi két követelménynek kell eleget tenni az $x_0, x_1, \dots, x_i$ sorozatnak:

- (i) legyen (majdnem) egyenletes eloszlású, és
- (ii) ne lehessen x_i -t kitalálni az $x_0, \dots, x_{i-1}$ ismeretével.

8.2.3. Lineáris kongruenciák

Egyenletes eloszlású, pszeudovéletlen számok generálása lineáris kongruenciával az alábbi iteráción alapszik:

$$x_{i+1} \equiv ax_i + c \pmod{m}$$

A definícióból nyilvánvaló, hogy a sorozat ciklikus lesz. Tapasztalat szerint a következőket tudjuk:

1. a $c = 0$ esetben a generálási eljárás egy picit gyorsabb, viszont lecsökken a periódus hossza;

2. az $a = 1$ eset elvethető, olyankor nem kapunk véletlenszerűen viselkedő sorozatot, az $a = 0$ még rosszabb választás;
3. valamint a leghosszabb (m lépésű) periódus eléréséhez szükséges és elegendő, hogy
 - c és m legyenek relatív prímek
 - $b = a - 1$ osztható az m minden prímfaktorával
 - $b = a - 1$ a 4 többszöröse, ha m is a 4 többszöröse
 - b nem 0.
4. A fentiek alapján az m gyakran 2^{32} vagy 2^{64} , mert akkor csak levágni kell a modulus számításához.

Példa. Nézzünk most egy látszólag jónak (elégge szofisztikáltnak) tűnő választást. Kérdés, hogy mi annak a lineáris kongruenciasorozatnak a periódushossza, amelyben a következő paraméter választással éltünk:

$$\begin{aligned}x_0 &= 5772156648, & a &= 3141592621 \\c &= 2718281829, & m &= 10000000000\end{aligned}$$

A válasz, talán nagy meglepetésre: 3.

RANDU. A RANDU eljárást még a 60-as években használták, amely a következő kongruencia sorozatot használja:

$$x_{i+1} = 65539x_i \pmod{2^{31}}$$

A kiindulási érték x_0 páratlan szám kell, hogy legyen. A paraméterek választását a 32-bites szó használatát alkalmazó számítógépek indokolták, amelyeken a $\pmod{2^{31}}$ művelet hardveres szinten gyorsan elvégezhető. Azonban $65539 = 2^{16} + 3$ és

$$x_{i+2} = (2^{16} + 3)x_{i+1} = (2^{16} + 3)^2 x_i \pmod{2^{31}}$$

ha most kibontjuk a négyzetes tagot:

$$x_{i+2} = (2^{32} + 6 \cdot 2^{16} + 9)x_i = (6 \cdot (2^{16} + 3) - 9)x_i \pmod{2^{31}}$$

mivel $2^{32} \pmod{2^{31}} = 0$.

Ebből már látszik, hogy

$$x_{i+2} = 6x_{i+1} - 9x_i$$

Ez lényegében megkérdőjelezi az összes 60-as években számított (tudományos) eredményt, ahol ezt a módszert használták.

Lineáris kongruenciák a gyakorlatban A gyakorlatban úgy állítunk elő véletlenszámokat, hogy az

$$x_{n+1} = (ax_n + c) \pmod{m}$$

sorozatból indulunk ki, ahol az x -ek *egészek*; ezután az $u_n = x_n/m$ *törtet*et használjuk. Ekkor az u_n -ekre vonatkozó rekurzió:

$$u_{n+1} = (au_n + c/m) \pmod{1}.$$

Vegyük figyelembe azonban, hogy az au_n szorzat olyan nagy, hogy a c/m akár nem is változtat rajta. Továbbá a $\pmod{1}$ is levághat értékes jegyeket, ezért lebegőpontos számításnál ekkor *dupla* pontosság kell.

8.2.4. Mersenne twister

A módszert Matsumoto és Nishimura javasolt 1997-ben. Szintén lineáris kongruencia módszer, viszont a periódusa $2^{19937} - 1$, amely egy Mersenne prím. A standard változat az MT19937 nevet viseli és 32-bites szóhosszat használ. A 64-bites változat az MT19937-64, ami egy másik sorozatot gyárt.

Az algoritmus sokkal gyorsabb, mint más generátorok; mégis: nem alkalmas titkosításra, mert eleendően sok tag megfigyelése után kitalálható a sorozat. További hátránya még, hogy az inicializálásra nagyon érzékeny, az elején esetleg nem is felel meg véletlen teszteknek.

Ezzel együtt szinte az összes, széles körben használt programozási nyelvben ez a standard eljárás a `rand()` függvényre.

8.2.5. Statisztikai próbák

A statisztikai próbák arra adnak lehetőséget, hogy matematikailag megalapozott módon eldöntsük egy sorozatról, hogy véletlenszerű-e.

Csak említés szintjén: a χ^2 -próba diszkrét (nominális (vagy kategoriális)) változók vizsgálatára, a Kolmogorov-Szmirnov-próba folytonos eset (amikor tehát a véletlen valós számok végtelen sok értéket vehetnek fel).

Diehard tesztek A teszt-sorozat egy alapvetésként szolgál véletlenszerűség vizsgálatára. Röviden ismertetjük a felhasznált módszereket.

Születésnap beosztás Válasszunk véletlen pontokat egy nagy intervallumból. A pontok közötti távolságok aszimptotikusan exponenciális eloszlást kell, hogy kövessenek. A teszt neve a korábban említett születésnap paradoxonból jön.

Átlapoló permutációk: Vizsgáljunk 5 egymás utáni véletlenszám sorozatát. A 120 különböző sorba rendezés ugyanakkora valószínűséggel kellene, hogy előforduljon.

Mátrixok rangjai Válasszunk néhány bitet néhány számból, amelyek véletlenszámokból származnak. Ezekből képezzünk $\{0, 1\}$ feletti mátrixokat. Számoljuk össze a rangjaikat.

Majom teszt: Tekintsük bitek egy sorozatát 'szavaknak'. Számoljuk össze az átlapoló szavak számát. Azon szavak száma, amelyek nem fordulnak elő egy adott eloszlást kell, hogy kövessenek. Az elnevezés a végtelen majom problémából ered.

Számoljuk össze az 1-eseket: Számoljuk össze az 1-es bitek számát, majd ezt alakítsuk 'betűkké'. Számoljuk össze az öt betűs 'szavak' előfordulásának számát.

Parkolóház teszt: Helyezzünk el véletlenszerűen egységköröket egy 100×100 -as négyzetbe. Egy kör akkor parkolt sikeresen, ha nincs átfedése a már eddig sikeresen parkolt körök egyikével sem. 12.000 próbálkozás után a sikeresen parkolt körök száma egy bizonyos normális eloszlást kell, hogy kövessen.

Minimális távolság teszt: Helyezzünk el 8000 pontot véletlenszerűen egy 10000×10000 -es négyzetbe, majd számítsuk ki a párok közötti legrövidebb távolságot. Ennek a távolságnak a négyzete exponenciális eloszlást kell, hogy kövessen egy adott átlaggal.

Véletlen gömbök teszt: Válasszunk véletlenszerűen 4000 pontot egy 1000 oldalhosszúságú kockából. Helyezzünk el gömböket a kiválasztott pontokra, amelyeknek a sugara a többi ponttól mért távolság minimuma legyen. A legkisebb gömb-térfogat exponenciális eloszlást kell, hogy kövessen.

The squeeze test: Szorozzuk meg a 2^{31} -et a $(0, 1)$ intervallumból véletlenül választott lebegőpontos számokkal addig, amíg nem érjük el az 1-et. Ismételjük meg ezt 100000 alkalommal. Az 1 eléréséhez szükséges próbálkozások száma egy adott eloszlást kell, hogy kövessen.

Átlapoló összegek teszt: Hozzunk létre a $(0, 1)$ intervallumból választott véletlen lebegőpontos számokból egy hosszú sorozatot. Adjuk össze 100 egymás után következő elemet. Az összegnek normális eloszlásúnak kell lennie.

Run teszt: Hozzunk létre a $(0, 1)$ intervallumból választott lebegőpontos számokból egy hosszú sorozatot. Számoljuk össze a növekvő és csökkenő run¹-ok számát. Ezek egy adott eloszlást kell, hogy kövessenek.

Craps test Játszunk le 200000 craps² fordulót, miközben összeszámoljuk a nyeréseket és a dobások számát játékonként. Minden ilyen összeszámlálás egy adott eloszlást kell, hogy kövessen.

8.2.6. További megjegyzések

A <http://www.random.org/> honlapon véletlenszámok sorozatát érhetjük el. A szolgáltatás egy része ingyenes. Az oldal üzemeltetői szerint a megadott véletlenszám sorozatok atmoszférikus zaj eredetűek, így jobbak, mint a fentebb tárgyalt pseudo véletlenszám generátorok.

Ugyanitt érdemes átolvasni a <http://www.random.org/analysis/> oldalt is, ahol a statisztikai elemzésekről találunk további érdekességeket, hivatkozásokat.

Az érdeklődő olvasó figyelmébe ajánljuk továbbá a [1] könyv 3.5 szakaszát, amely a *'Mit jelent az, hogy véletlen sorozat?'* címet viseli.

Végül megemlítjük, hogy a fentiekben lényegében egyenletes eloszlású véletlenszámok előállításáról volt szó, azonban másfajta eloszlásra is szükség lehet. Továbbá sok esetben véletlen permutációt (*shuffle*) vagy véletlen kombinációt kell előállítanunk - az ehhez használt módszerek a már megismert eljárások kiterjesztését kívánják.

8.3. Feladatok

1. Amennyiben a használt operációs rendszerünk lehetővé teszi nem csak szoftveres alapú véletlenszám generálást, akkor írjuk olyan programot, amely ezt a hardveres alapú generálást demonstrálja.
Példaként a UNIX rendszerekben elérhető `/dev/random` nevű fájlt lehet itt említeni, amely a rendszerben működő eszközmeghajtóktól gyűjt zajt, és ezt használja föl pseudo-véletlenszám generálására.
2. Feltéve, hogy adott egy eljárás, amely egyenletes eloszlással véletlenszámot generál, írjunk egy olyan eljárást, amely adott átlaggal és szórással normális eloszlású véletlenszámokat generál.
3. Cinkelés érme helyrehozatala. Amennyiben adott egy érménk, amely P_0 valószínűséggel 0 (fej) és P_1 valószínűséggel 1 (írás) értéket ad, ahol $P_0 \neq P_1$, akkor annak a valószínűsége, hogy az 1 értéket 0 követ $P_1 \cdot P_0$, míg annak, hogy a 0 értéket 1 követ $P_0 \cdot P_1$. Ez a két szorzat valószínűség már megegyezik. Ezért a kiegyensúlyozást úgy végezhetjük el, hogy csak akkor adunk vissza 0 vagy 1 értéket, ha két egymás utáni dobás eredménye különböző.
Ennek egy általánosítása a következő feladat. Írjunk egy eljárást, amely 0 vagy 1 értékkel tér

¹Egy sorozat *run* értéke az a maximális részsorozat, amely egymás után azonos értékeket tartalmaz. Például a 22 hosszú sorozat: '+++++---++-+++++---' összesen hat darab run-t tartalmaz, ezek közül 3 olyan van, amely '+' jelet, a többi pedig '-' jelet.

²a craps játékot két kockával játszták, bővebben lásd <https://hu.wikipedia.org/wiki/Craps>

vissza, de úgy, hogy az 1 érték, átlagosan egyszer fordul elő N hívásból, ahol $N \in [3, 6]$. Ezután írunk egy másik eljárást, amely az előzőt használja a véletlenszám generálásra, viszont azt helyrehozza.

4. Generáljunk 100 darab véletlen (x, y) számpárt egyenletes eloszlással úgy, hogy azokra teljesüljön a

$$10 \leq \sqrt{x^2 + y^2} \leq 15$$

egyenlőtlenség sorozat. Rajzoljuk ki a kapott (x, y) számpárokat egy Descartes-féle koordináta rendszerben.

5. Adott (n, e, d) RSA kulcsokkal írjunk RSA titkosító eljárást.

9. fejezet

Polinomok faktorizálása

A polinomok gyökeinek egzisztenciájára vonatkozó tétel alapján bebizonyítható, hogy a komplex és valós számtestek fölötti polinomok irreducibilis faktorizációja létezik és egyértelmű.

Olyan polinomokat keresünk, amelyek a polinomgyűrűben ugyanolyan szerepet játszanak, mint az egész számok gyűrűjében a prímszámok.

A polinomokat általában valamilyen test felett definiáljuk (ahol az osztás elvégezhető). Leggyakrabban racionális számok felett, valós vagy komplex számok felett, vagy mint a fejezetben bemutatott példában is: egészek modulo p prím felett. Külön jelentőségű az egészek mod 2 feletti polinomoknak: ezek számos analógiát mutatnak a kettes számrendszerben kifejezett egészekkel. Azonban polinomoknál az együttthatók nincsennek kapcsolatban, nincs 'átvitel'.

A test (ami fölött a polinomokat tekintjük) legrögzítése fontos lehet, mert például az $x^2 - 2$ polinom az egészek és a racionális számok felett irreducibilis, ugyanakkor felbontható a valós és komplex számok felett:

$$x^2 - 2 = (x - \sqrt{2})(x + \sqrt{2}).$$

Másik példa: az $x^2 - 1$ a racionális és valós számok felett irreducibilis, ugyanakkor

$$x^2 - 1 = (x - i)(x + i)$$

A következőkben rátérünk a talán legfontosabb eset tárgyalására, amely a véges testek feletti polinomok faktorizációja. Az ide tartozó eljárások a fordulnak elő leggyakrabban az egyes számítógépes algebrai rendszerekben.

Megjegyezzük, hogy az n -ed fokú polinom mod 2 felbontása egyszerűbb, mint egy n bites 2-es számrendszerben felírt számé.

9.1. Felbontás modulo p

Berlekamp 1967

Legyen p prímszám; a következőkben a polinomokkal mindenféle számítást modulo p végzünk. Tegyük fel, hogy adott egy $u(x)$ polinom, amelynek együttthatói a $\{0, 1, \dots, p-1\}$ halmazból valók, és feltehetjük, hogy $u(x)$ normált. Célunk az, hogy $u(x)$ -et

$$u(x) = p_1(x)^{e_1} \cdot \dots \cdot p_r(x)^{e_r}$$

alakban fejezzük ki, ahol $p_1(x), \dots, p_r(x)$ egymástól különböző, normált irreducibilis polinomok.

Előkészítő lépés: eldöntjük, hogy $e_i > 1$ valamely i indexre. Ha

$$u(x) = u_n x^n + \dots + u_0 = v(x)^2 w(x)$$

alakú, akkor a szokásos módon, de modulo p , kifejezett deriváltjára

$$u'(x) = n \cdot u_n x^{n-1} + \dots + u_1 = 2v(x)v'(x)w(x) + v(x)^2 w'(x),$$

ami tehát a $v(x)$ többszöröse. Ezért határozzuk meg a

$$d(x) = \text{luko}(u(x), u'(x))$$

polinomot. Ha

- $d(x) = 1$: akkor $u(x)$ négyzetmentes.
- $d(x) \neq 1$ és $d(x) \neq u(x)$: akkor d az u valódi tényezője
- $d(x) \neq 1$ és $d(x) = u(x)$: akkor $u'(x) = 0$, és ebből következik, hogy x^k -nak az u_k együtthatója csak akkor nem 0, ha a k a p többszöröse.
Ebből következik, hogy

$$u(x) = v(x^p) = (v(x))^p \quad (9.1)$$

alakú, tehát ilyenkor valójában a $v(x)$ irreducibilis tényezőit kell meghatározni.

Az (9.1) bizonyítása:

Ha $v_1(x)$ és $v_2(x)$ modulo p vett polinomok, akkor

$$\begin{aligned} (v_1(x) + v_2(x))^p &= v_1(x)^p + \binom{p}{1} v_1(x)^{p-1} v_2(x) + \dots \\ &\quad \dots + \binom{p}{p-1} v_1(x) v_2(x)^{p-1} + v_2(x)^p \\ &= v_1(x)^p + v_2(x)^p, \end{aligned}$$

hiszen a binomiális együtthatók mind többszörösei a p -nek. Továbbá, a Fermat tétel szerint bármilyen a egész számra: $a^p \equiv a \pmod{p}$.

Ezért, ha $v(x) = v_m x^m + \dots + v_0$, akkor

$$\begin{aligned} v(x)^p &= (v_m x^m)^p + (v_{m-1} x^{m-1})^p + \dots + (v_0)^p \\ &= v_m x^{mp} + v_{m-1} x^{(m-1)p} + \dots + v_0 \\ &= v(x^p). \end{aligned}$$

Feltesszük tehát, az előzőek alapján, hogy $u(x)$ négyzetmentes, keressük az

$$u(x) = p_1(x) \cdot \dots \cdot p_r(x)$$

alakot, ahol $p_i(x)$ irreducibilis.

Ez volt az előkészítő rész, mehetünk tehát az algoritmus fő lépéseire.

A kínai maradéktétel érvényes polinomokra is. Ha $(s_1, s_2, \dots, s_r) \pmod{p}$ vett egészek bármilyen rendezett r -ese, akkor létezik pontosan egy olyan $v(x)$ polinom, hogy

$$\begin{aligned} v(x) &\equiv s_1 \pmod{p_1(x)} \\ &\quad \dots \\ v(x) &\equiv s_r \pmod{p_r(x)} \end{aligned} \quad (9.2)$$

$$\deg(v) < \deg(p_1) + \dots + \deg(p_r) = \deg(u)$$

Jelölés:

$$\begin{aligned} g(x) &\equiv h(x) \pmod{f(x)} \text{ ugyanaz, mint} \\ g(x) &\equiv h(x) \pmod{f(x) \text{ és } p}, \end{aligned}$$

mivel most modulo p aritmetikával dolgozunk.

A (9.2)-beli $v(x)$ polinom módot ad arra, hogy $u(x)$ tényezőit megkapjuk. A (9.2)-ből következik: ha $r \geq 2$, és $s_1 \neq s_2$, akkor

$$\begin{aligned} \text{luko}(u(x), v(x) - s_1) &\text{ osztható } p_1(x)\text{-szel,} \\ &\text{de nem osztható } p_2(x)\text{-szel} \end{aligned}$$

Ez a megfigyelés azt mutatja, hogy a (9.2) megfelelő $v(x)$ megoldásaiból információt szerezhetünk $u(x)$ tényezőire nézve.

Ezért vizsgáljuk (9.2)-t: $v(x)$ kielégíti a

$$v(x)^p \equiv s_j^p = s_j \equiv v(x) \pmod{p_j(x)} \quad 1 \leq j \leq r$$

feltételt (ne felejtjük: modulo p dolgozunk), ebből következik

$$v(x)^p \equiv v(x) \pmod{u(x)}, \quad \deg(v) < \deg(u) \quad (9.3)$$

Továbbá, bebizonyítható, hogy

$$x^p - x \equiv (x - 0)(x - 1) \dots (x - (p - 1)) \pmod{p}$$

amiből következik, hogy

$$v(x)^p - v(x) = (v(x) - 0)(v(x) - 1) \dots (v(x) - (p - 1)) \quad (9.4)$$

bármilyen $v(x)$ polinomra érvényes, amikor modulo p dolgozunk.

Ha $v(x)$ kielégíti (9.3)-t, akkor $u(x)$ osztja $(v(x)^p - v(x))$ -et, ezért $u(x)$ minden irreducibilis tényezője kell, hogy ossza a (9.4) jobb oldalnak p darab relatív prím tényezőjéből valamelyiket. Azaz (9.3) minden megoldásának (9.2) alakúnak kell lennie valamely $s_1, \dots, s_r$ számokkal, és (9.3) megoldásainak száma p^r , ahol r az $u(x)$ prímtényezőinek száma.

Ezért aztán (9.3) kongruencia $v(x)$ megoldásai $u(x)$ felbontásához adnak kulcsot.

Első pillantásra úgy tűnik, hogy (9.3) összes megoldásának megtalálása nehezebb $u(x)$ felbontásánál, de valójában ez nincs így, hiszen a (9.3) megoldáshalmaza az összeadásra nézve zárt.

Legyen $\deg(u) = n$. Állítsuk elő a

$$Q = \begin{pmatrix} q_{0,0} & q_{0,1} & \dots & q_{0,n-1} \\ \vdots & \vdots & & \vdots \\ q_{n-1,0} & q_{n-1,1} & \dots & q_{n-1,n-1} \end{pmatrix}$$

$n \times n$ -es mátrixot, ahol

$$x^{pk} \equiv q_{k,n-1}x^{n-1} + \dots + q_{k,1}x + q_{k,0} \pmod{u(x)}, \quad (9.5)$$

ahol $k = 0, \dots, n - 1$.

Ekkor $v(x) = v_{n-1}x^{n-1} + \dots + v_1x + v_0$ pontosan akkor megoldása (9.3)-nak, ha

$$(v_0, v_1, \dots, v_{n-1}) \cdot Q = (v_0, v_1, \dots, v_{n-1})$$

(ne felejtjük, hogy modulo p dolgozunk)

A $(v_0, v_1, \dots, v_{n-1}) \cdot Q = (v_0, v_1, \dots, v_{n-1})$ egyenlőség akkor és csak akkor igaz, ha

$$v(x) = \sum_j v_j x^j = \sum_j \sum_k v_k q_{k,j} x^j \equiv \sum_k v_k x^{pk} = v(x^p) \equiv v(x)^p,$$

ahol a kongruenciákat $(\text{mod } u(x))$ kell érteni.

Az első egyenlőség csak a $v(x)$ definíciója

A második egyenlőség a $vQ = v$ egyenlet részletesebb kiírása

A harmadik egyenlőség az már egy kongruencia, az (9.5) miatt igaz

Utána a negyedik egyenlőség az polinom definíció, semmi különös,

Végül az utolsó egyenlőség megint egy kongruencia, az pedig a korábban látott és bizonyított (9.1) miatt igaz.

A fentiek képezik a Berlekamp algoritmus alapötletét. Knuth II. 430-434 De a következőkben leírunk egy 'rövidített' változatot

9.1.1. Berlekamp algoritmus

Input: $u(x)$ polinom és p prímszám

Output: $u(x)$ faktorizációja modulo p

1. lépés. Vizsgáljuk meg, hogy $\text{lko}(u(x), u'(x))$ értékét. Ha ez 1, akkor mehetünk tovább, hiszen $u(x)$ négyzetmentes. Ha nem 1, akkor csináljuk végig azokat a lépéseket, amiket a főlíásor elején láttunk.

2. lépés. Készítsük el a Q mátrixot.

3. lépés. Keressük meg a $(Q - I)^T$ mátrix nullterét. A nulltér dimenziója lesz az $u(x)$ faktorainak száma modulo p . A fenti jelöléseinket használva ez r .

Ennek eredménye olyan r elemű vektorsorozat lesz, amelyre $v^{(j)}(Q - I) = 0$ teljesül $(1 \leq j \leq r)$.

4. lépés. Ha $r > 1$, akkor számítsuk ki az

$$\text{lko}(u(x), v^{(2)}(x) - s) \text{ mod } p$$

értékét, ahol $v^{(2)}(x)$ a $v^{(2)}$ vektor által reprezentált polinom és $0 \leq s < p$.

5. lépés. Ha $v^{(2)}$ fölhasználásával nem sikerült r részre bontani $u(x)$ -et, akkor a további faktorok meghatározását az

$$\text{lko}(v^{(k)}(x) - s, w(x)) \text{ mod } p$$

kiszámításával folytatjuk, ahol $0 \leq s < p$, $w(x)$ pedig az összes eddig megkapott tényező, $k = 3, 4, \dots$, mindaddig, amíg az r darab tényezőt meg nem kapjuk.

/pub/tuszi/gyakorlat/poli-fakt/polinom-faktor-pelda-Knuth.mw

Műveletigény: ha p kicsi, akkor $O(n^3 + prn^2)$. megmutatható, hogy r átlagos száma $\ln(n)$

9.1.2. Példa

Az alábbiakban részletesen végigszámolunk egy példát a fenti algoritmus demonstrálására. Ehhez a Maple rendszert használjuk, és nem az algoritmus implementációját mutatjuk be, hanem a számolás menetét. A példában az

$$u(x) = x^8 + x^6 + 10x^4 + 10x^3 + 8x^2 + 2x + 8$$

polinom faktorizálását végezzük el $p = 13$ választással.

```
> with(PolynomialTools): with(LinearAlgebra):  
> u := x^8+x^6+10*x^4+10*x^3+8*x^2+2*x+8:  
> p := 13:
```

Deriváljuk $u(x)$ -et és megnézzük, hogy a legnagyobb közös osztója $u(x)$ -szel micsoda.

```
> du:=diff(u,x);  
du := 8x^7 + 6x^5 + 40x^3 + 30x^2 + 16x + 2  
> gcd(u, du);
```

1

Mivel 1 lett a legnagyobb közös osztó, ezért $u(x)$ négyzetmentes, és így mehetünk tovább. Előállítjuk a Q mátrixot.

```
> Q:=Matrix(8):  
> Q[1]:=Transpose(Vector([1,0,0,0,0,0,0,0]));  
Q1 := [ 1 0 0 0 0 0 0 0 ]  
> Q[2]:=Transpose(CoefficientVector(modpol(x^13, u, x, p), x));  
Q2 := [ 2 1 7 11 10 12 5 11 ]  
> Q[3]:=Transpose(CoefficientVector(modpol(x^26, u, x, p), x));  
Q3 := [ 3 6 4 3 0 4 7 2 ]  
> Q[4]:=Transpose(CoefficientVector(modpol(x^39, u, x, p), x));  
Q4 := [ 4 3 6 5 1 6 2 3 ]  
> 39+p;
```

52

```
> Q[5]:=Transpose(CoefficientVector(modpol(x^52, u, x, p), x));  
Q5 := [ 2 11 8 8 3 1 3 11 ]  
> 52+p;
```

65

```
> Q[6]:=Transpose(CoefficientVector(modpol(x^65, u, x, p), x));  
Q6 := [ 6 11 8 6 2 7 10 9 ]  
> 65+p;
```

78

```
> Q[7]:=Transpose(CoefficientVector(modpol(x^78, u, x, p), x));  
Q7 := [ 5 11 7 10 0 11 7 12 ]  
> 78+p;
```

91

```
> Q[8]:=Transpose(CoefficientVector(modpol(x^91, u, x, p), x));  
Q8 := [ 3 3 12 5 0 11 9 12 ]
```

> Q;

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 7 & 11 & 10 & 12 & 5 & 11 \\ 3 & 6 & 4 & 3 & 0 & 4 & 7 & 2 \\ 4 & 3 & 6 & 5 & 1 & 6 & 2 & 3 \\ 2 & 11 & 8 & 8 & 3 & 1 & 3 & 11 \\ 6 & 11 & 8 & 6 & 2 & 7 & 10 & 9 \\ 5 & 11 & 7 & 10 & 0 & 11 & 7 & 12 \\ 3 & 3 & 12 & 5 & 0 & 11 & 9 & 12 \end{bmatrix}$$

Megvan a Q mátrix. Most meg kell oldanunk a $v^T(Q - I) = 0$ homogén lineáris egyenletrendszert, ami azt jelenti, hogy meg kell keresnünk a $(Q - I)^T$ mátrix nullterét.

> QIT:=Transpose(Q-IdentityMatrix(8));

$$QIT := \begin{bmatrix} 0 & 2 & 3 & 4 & 2 & 6 & 5 & 3 \\ 0 & 0 & 6 & 3 & 11 & 11 & 11 & 3 \\ 0 & 7 & 3 & 6 & 8 & 8 & 7 & 12 \\ 0 & 11 & 3 & 4 & 8 & 6 & 10 & 5 \\ 0 & 10 & 0 & 1 & 2 & 2 & 0 & 0 \\ 0 & 12 & 4 & 6 & 1 & 6 & 11 & 11 \\ 0 & 5 & 7 & 2 & 3 & 10 & 6 & 9 \\ 0 & 11 & 2 & 3 & 11 & 9 & 12 & 11 \end{bmatrix}$$

> ns:=Nullspace(QIT) mod p;

$$ns := \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 5 \\ 5 \\ 0 \\ 9 \\ 5 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 9 \\ 11 \\ 9 \\ 10 \\ 12 \\ 0 \\ 1 \end{bmatrix} \right\}$$

Ezeket a vektorokat, mint polinomokat kell most megvizsgálnunk.

> v2:=FromCoefficientVector(ns[2],x);

$$v2 := x^6 + 5x^5 + 9x^4 + 5x^2 + 5x$$

```

> f:=0:
> for s from 0 to p-1 do
>   Gcd(u, v2-s) mod p:
>   if (%<>1) then
>     f:=f+1:
>     faktor[f]:=%%:
>     fi:
>   end do:
>   print("faktor"); f;
>   for ff from 1 to f do
>     faktor[ff];
>   end do;

```

„faktor”

2

$$x^5 + 5x^4 + 9x^3 + 5x + 5$$

$$x^3 + 8x^2 + 4x + 12$$

Ezek alapján már 2 faktorunk megvan, de kell egy harmadik is (ezt onnan tudjuk, hogy az előbb a nulltér 3 dimenziós volt).

```

> v3:=FromCoefficientVector(ns[3], x);
> v3 := x^7 + 12x^5 + 10x^4 + 9x^3 + 11x^2 + 9x
> for s from 0 to p-1 do
>   Gcd(faktor[1], v3-s) mod p;
> end do;

```

1

1

1

1

1

1

$$x^4 + 2x^3 + 3x^2 + 4x + 6$$

1

$$x + 3$$

1

1

1

1

Végül nézzük meg, mit ad a Maple beépített faktorizálója (elsősorban az kimenet formátuma miatt érdekes, nyilván az eredmény ugyanaz, mint a fentiek).

```

> Factors(u) mod p;
[1, [[x + 3, 1], [x^3 + 8x^2 + 4x + 12, 1], [x^4 + 2x^3 + 3x^2 + 4x + 6, 1]]]

```

9.2. Feladatok

1. A Rolle-féle gyöktétel szerint egész együtthatós polinom minden $x = p/q$ racionális megoldására (ahol p és q relatív prím egészek, $q \neq 0$) teljesül, hogy:

- p olyan egész, amely osztója a_0 -nak (konstans tag), és
- q olyan egész, amely osztója a_n -nek (főegyüttható).

A tétel felhasználható arra, hogy eldöntsük, egy polinomnak vannak-e racionális gyökei, és pozitív válasz esetén meg is keressük azokat.

Egy példán bemutatva: az $x^3 - 7x + 6$ polinom esetén a gyökjelöltek azok a racionális számok, amelyeknek a számlálója 6 osztója, nevezője pedig 1 osztója. Adódik, hogy a jelöltek a $\pm 1, \pm 2, \pm 3$ és ± 6 . Ezek 1, 2 és -3 gyökök (az összes, mint tudjuk). A visszaellenőrzést a Horner-módszerrel elvégezhetjük.

Írjunk egy eljárást, amely adott polinomra elvégzi az itt vázolt műveleteket.

2. Mutassuk meg, hogy $x^2 + x + 1$ irreducibilis mod 5 és mod 29.
3. Mutassuk meg, hogy $x^3 - a$ irreducibilis mod 7 hacsak $a = 0$ vagy $a = \pm 1$.
4. A PARI/GP szofver csomag tartalmazza a fejezetben ismertett Berlekamp algoritmus implementációját. Ez elérhető a <http://pari.math.u-bordeaux.fr/> címen. A feladatunk, hogy fedezzük fel az implementáció részleteit.
5. Kronecker módszer. Mivel az egész együtthatós polinomok faktorainak is egész együtthatós polinomoknak kell lenniük, és egy egész együtthatós polinom helyettesítési értéke egész pontokban, egész (mivel egész számok véges szorzata, véges összege egész, vagyis az az egész számok zártak szorzásra és összeadásra), így polinom értékeinek prímfaktorizációját felhasználva, véges számú elemre redukálhatjuk a lehetséges (polinom)faktorokat.
Például, vegyük az $f(x) = x^5 + x^4 + x^2 + x + 2$ polinomot. Ha ez a polinom felbontható az egész számok felett, akkor legalább az egyik faktorának első vagy másodfokúnak kell lennie (lásd az algebra alaptételének egy megfelelő változatát). Három értékre van szükségünk ahol, hogy egy legfeljebb másodfokú polinomot egyértelműen meg tudjunk határozni. Legyenek ezek a pontok az $f(0) = 2, f(1) = 6$ és az $f(-1) = 2$. Ha a 3 közül bármelyik érték 0 akkor máris találunk egy gyököt és így a faktorizációs tétel alapján egy faktort is. Ha viszont egyik sem 0 akkor mindegyiknek csak véges mennyiségű osztója van. Például 2 egész számok segítségével csak a következőképpen írható:

$$1 \cdot 2, 21, (-1) \cdot (-2), \text{ vagy } (-2) \cdot (-1).$$

Vagyis ha másodfokú faktor létezik akkor annak 1, 2, -1 vagy -2 értékűnek kell lennie az $x = 0$ pontban, és az $x = -1$ pontban. Mivel 6-ot 8-féleképpen lehet felbontani így összesen $4 \cdot 4 \cdot 8 = 128$ kombinációs lehetőség van, amelynek fele csak előjelben különbözik így ezeket elhagyva csak 64 másodfokú polinom marad, amit le kell tesztelni. Csak ezek a lehetséges faktori $f(x)$ -nek. Ezek tesztelése szerint

$$p(x) = x^2 + x + 1$$

amit úgy kaptunk, hogy $p(0) = 1, p(1) = 3$ és $p(-1) = 1$ osztja $f(x)$ -et a megfelelő pontokban. Az $f(x)$ polinomot $p(x)$ -szel osztva megkapjuk a másik faktort: $q(x) = x^3 - x + 2$, így $f = pq$. Mivel kiderül, hogy mind p és q irreducibilis így f irreducibilis faktorizációja:

$$f(x) = p(x)q(x) = (x^2 + x + 1)(x^3 - x + 2).$$

Feladatunk, hogy a példa alapján készítsük el a Kronecker módszer implementációját.

10. fejezet

Programkönyvtárak

10.1. BLAS

A BLAS (Basic Linear Algebra Subroutines) egy specifikáció, amely meghatározza, hogyan kell megvalósítani alacsony szintű eljárásokat általános lineáris algebrai műveletekre, mint például vektorok összeadása, skalárral történő szorzás, skaláris szorzat, lineáris kombináció és mátrixok szorzása. Ezek lényegében de facto standard műveletek lineáris algebrai könyvtárak részére. Bár a specifikáció általános, a BLAS implementációk általában sebességre optimalizáltak egy adott architektúrára.

A referencia implementáció 1979-ben készült még FORTRAN nyelvre, megtalálható a <http://netlib.org/blas/blast-forum> címen.

A BLAS funkcionalitás szempontjából 3 eljárás halmazra bontható, amelyeket szinteknek nevezünk. Ezek egyrészt kronológiai sorrendben jönnek egymás után, ugyanakkor a számítási költségük is különbözik egymástól. A modern BLAS implementációk mindhárom szintet tartalmazzák.

BLAS-1 Az alapjai 1973-1977 között készültek. Összesen 15 (főleg) vektor műveletet tartalmaz, amelyek közül néhány például az „AXPY” ($y = \alpha x + y$), skaláris szorzat, $x = \alpha x$ alakú művelet, stb. 4 verzióban áll rendelkezésre (Single, Double, Complex, Integer), 46 eljárás. Miért BLAS 1? Mert $\mathcal{O}(n^1)$ műveletet végzünk $\mathcal{O}(n^1)$ adaton.

BLAS-2 A BLAS-2 kifejlesztése 1984-1986 között történt. Jellemzően mátrix-vektor műveleteket tartalmaz, úgy mint a „GEMV” ($y = \alpha \cdot A \cdot x + \beta \cdot x$) és „GER” ($A = A + \alpha \cdot x \cdot y^T, x = T - 1 \cdot x$) típusú műveletek. 4 verzió létezik (S/D/C/Z), 66 eljárás. Miért BLAS-2? Mert $\mathcal{O}(n^2)$ műveletet végzünk $\mathcal{O}(n^2)$ adaton.

BLAS-3 A következő szint a BLAS-3, amelynek kifejlesztése 1987-1988 között történt. 9 (főleg) mátrix-mátrix műveletet tartalmaz, például „GEMM” típusúakat ($C = \alpha \cdot A \cdot B + \beta \cdot C, C = \alpha \cdot A \cdot AT + \beta \cdot C, B = T - 1 \cdot B$). 4 verzió létezik (S/D/C/Z), 30 eljárás. Miért BLAS-3? Mert $\mathcal{O}(n^3)$ műveletet végzünk $\mathcal{O}(n^2)$ adaton.

LAPACK A 'Linear Algebra Package' egy olyan eljárásgyűjtemény, amely numerikus lineáris algebrai algoritmusok implementációit tartalmazza. Többek között lineáris egyenletrendszerek megoldását, legkisebb négyzetek módszerét, sajátérték feladatokat megoldását, szinguláris érték felbontást, valamint mátrix faktorizációs eljárásokat tartalmaz.

Elérhetőség Minden BLAS1/2/3 kód elérhető a <http://www.netlib.org/blas> címen. Része a standard matematikai könyvtáraknak.

10.1.1. Változatok

A BLAS tehát egy referencia, amelynek számos további implementációja készült, ezekből válogatunk néhányat.

ATLAS (Automatic Tuned Linear Algebra System) egy open source BLAS változat, automatikus tuning alapú az installáláskor. Ezen tulajdonsága miatt gyakran ezzel a megvalósítás képezi az teljesítmény összehasonlítások alapját.

Letölthető a <http://math-atlas.sourceforge.net/> címről.

GotoBLAS Kazushige Goto által kézzel optimalizált assembly kód. Nagyjából 2008-ban leállt a továbbfejlesztése. Elérhető a <http://www.tacc.utexas.edu/tacc-projects/gotoblas2> címen.

OpenBLAS szintén open source, a GotoBLAS egy máig is aktívan fejlesztett és karbantartott fork-ja. <http://www.openblas.net/>

Intel MTL az Intel cég Math Kernel Library nevű csomagja, amely nem csak BLAS implementációt, hanem többek között LAPACK könyvtárat is tartalmaz. Természetesen Intel processzorokra optimalizált kód.

Ingyenesen elérhető a <https://software.intel.com/mkl> címen.

10.2. GNU könyvtárak

10.2.1. GSL

A C nyelven írt, jelenleg 2.5 verzióánál tartó Scientific Library elsődleges célja, hogy lecserélje a hagyományosan még FORTRAN nyelven implementált numerikus eljárásokat tartalmazó könyvtárakat.

A könyvtár a következő témaköröket fedi le:

Complex Numbers	Roots of Polynomials	Special Functions
Vectors and Matrices	Permutations	Combinations
Sorting	BLAS Support	Linear Algebra
CBLAS Library	Fast Fourier Transforms	Eigensystems
Random Numbers	Quadrature	Random Distributions
Quasi-Random Sequences	Histograms	Statistics
Monte Carlo Integration	N-Tuples	Differential Equations
Simulated Annealing	Numerical Differentiation	Interpolation
Series Acceleration	Chebyshev Approximations	Root-Finding
Discrete Hankel Transforms	Least-Squares Fitting	Minimization
IEEE Floating-Point	Physical Constants	Basis Splines
Wavelets	Sparse BLAS Support	Sparse Linear Algebra

A GSL a legtöbb modern, széles körben használt programozási nyelvre és környezetre elérhető.

A csomag az <https://www.gnu.org/software/gsl/> oldalról tölthető le.

10.2.2. GMP

A 'GNU Multiple Precision Arithmetic Library' programkönyvtár tetszőleges pontosságú aritmetikát kínál egészekre, racionális számokra és lebegőpontos számokra egyaránt. A pontosságnak csak a használt számítógép memóriája szab határt. A fejlesztők elsődleges motivációja, hogy a GMP a leggyorsabb tetszőleges pontosságú aritmetikai könyvtár legyen. Aktívan használják kriptográfiai alkalmazásokban és általános számítógépes algebrai rendszerekben.

Elérhető a <http://gmplib.org/> oldalon.

10.2.3. MPFR

A C nyelven írt 'Multiple Precision Floating-Point Reliably' programkönyvtár a GMP rendszeren alapszik és tetszőleges pontosságú lebegőpontos számítást tesz lehetővé korrekt kerekítéssel¹ és kivételkezeléssel.

Elérhető a <http://www.mpfr.org/> oldalon.

További MPFR alapú csomagok

MPC Az MPC egy olyan C nyelven írt könyvtár, amely komplex számokra kínál tetszőleges pontosságú aritmetikát, korrekt kerekítéssel.

Elérhető a <http://www.multiprecision.org/> címen.

MPFI Az MPFI csomagban tetszőleges pontosságú lebegőpontos intervallum aritmetikai eljárásokat találunk. Használatával intervallumos befoglalásokat kapunk a numerikus számításainkra.

Elérhető a <http://mpfi.gforge.inria.fr/> oldalon.

10.3. További függvénykönyvtárak

10.3.1. MPFQ

MPFQ egy véges testek fölött végzett számításokra alkalmas könyvtár. A többi hasonló csomaghoz képest az MPFQ úgy optimalizál sebességre, hogy a felhasználónak előre rögzítenie kell bizonyos tulajdonságokat, mint például használt mod érték, majd ezt a kód fordításakor kihasználja.

Elérhető a <http://mpfq.gforge.inria.fr/> címen.

10.3.2. FFTW

A 'Fastest Fourier Transform in the West' könyvtár diszkrét Fourier transzformáció kiszámítására alkalmas, ingyenes. Ahogyan azt az X szakaszban tárgyaltuk, az eljárás műveletigénye $\mathcal{O}(n \log(n))$. Számos megvalósítást tartalmaz, amely lehetővé teszi, hogy a bemenetek előzetes vizsgálata után kiválassza az adott helyzetre leginkább alkalmas verziót a tényleges számítás elvégzésére. A kettő hatvány hosszúságú bemenetekre optimális, továbbá nagyon gyors még olyan vektorokra, amelyek hossza kis méretű prímtényező szorzatából áll. 1999-ben elnyerte a 'Wilkinson Prize for Numerical Software' díjat.

Elérhető a <http://www.fftw.org/> oldalon.

10.3.3. NTL

Victor Shoup által C++ nyelven írt 'Number Theory Library' programcsomag számelméleti eljárások gyűjteménye. Támogatja a tetszőleges pontosságú egész és lebegőpontos aritmetikát, véges testek fölötti számításokat, vektorokkal és mátrixokkal végzett műveleteket, valamint lineáris algebrát is. Polinomok faktorizálására rendkívül gyors implementációt tartalmaz.

Ingyenesen elérhető a <http://shoup.net/ntl/> oldalon.

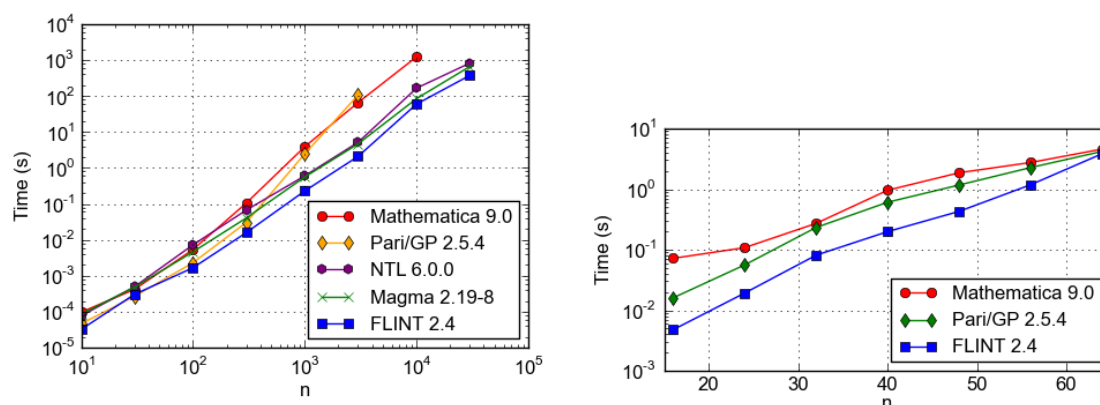
¹ azt mondjuk, hogy egy kerekítés korrekt, ha a szám és a kerekített érték között nincs az adott számítógépes környezetben elérhető ábrázolható szám (gépi szám)

10.3.4. FLINT

A 'Fast Library for Number Theory' egy C könyvtár, szintén számelméleti eljárások gyűjteménye. Ingyenesen elérhető a <http://flintlib.org/> oldalon.

Megjegyezzük, hogy Az NTL csomag szerzője készített egy (folyamatosan frissülő) összehasonlító tesztet a FLINT-tel, amely nagyon vegyes képet mutat: művelettől függően hol az egyik, hol a másik csomag a hatékonyabb, lásd a <http://shoup.net/ntl/benchmarks.pdf> dokumentumban.

Ugyanakkor a FLINT oldalon is találhatunk összehasonlító teszteket. Érdeességképpen két olyan példát ide is beemelünk, amelyek a jegyzet szempontjából relevánsak: polinom faktorizálás modulo 17 (lásd 10.1. ábra bal oldali grafikonja) és egy szóban elférő egészek faktorizálása, ahol az input 10,000 egymás utáni n bites egész $(2^{n-1} + 1, \dots, (2^{n-1} + 10000$ alakban - lásd 10.1. ábra jobb oldali grafikonján.



10.1. ábra. A FLINT könyvtár futási idejének összehasonlítása néhány hasonló csomaggal. A bal oldali ábrán polinomok faktorizálása modulo 17 a feladat, míg a jobb oldali ábrán $2^{n-1} + k$ alakú egészek faktorizálása.

10.3.5. Boost

A Boost egy C++ eljárásgyűjtemény, több, mint 80 különféle könyvtárral, amelyeknek egy része tudományos számításra alkalmas algoritmusok implementációját tartalmazza. Ezek közül kiemeljük a következőket:

Boost.Multiprecision: nagy pontosságú egész, racionális, lebegőpontos és komplex számokkal végzett aritmetikai műveleteket tartalmaz.

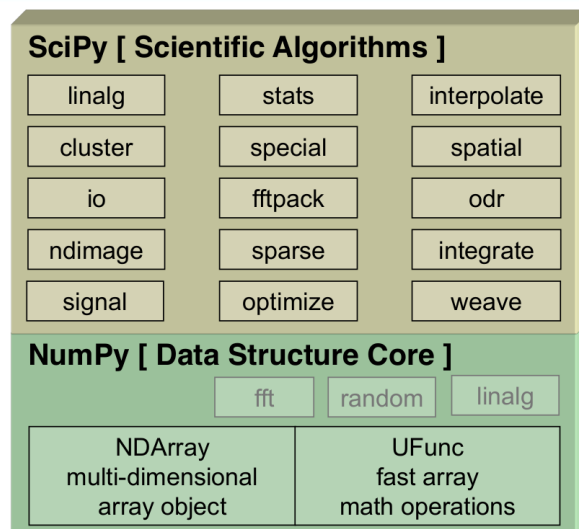
Boost.Random véletlenszám generátorokat és eloszlásfüggvényeket tartalmaz.

A könyvtár ingyenesen elérhető a <http://www.boost.org/> címen.

10.3.6. CLN

A 'Class Library for Numbers' ingyenes programkönyvtár, amely tetszőleges pontosságú aritmetikát tesz lehetővé. C++ nyelven íródott és használata lehetővé teszi az operátor felültöltést, valamint objektum orientált programozást.

Elérhető a <http://www.ginac.de/CLN/> címen.



10.2. ábra. A Numpy és a Scipy csomagok egymásra épülése és az elérhető adattípusok és matematikai függvények

10.4. Scipy és Numpy

A Numpy a Python programozási nyelvhez írt könyvtár, amely nagyméretű és magasdimenziós tömbök és mátrixok deklarálását és ezekkel végzett hatékony matematikai műveletek elvégzését támogatja. Használatával a MATLAB-hoz hasonló környezetet kapunk. Erősen támaszkodik BLAS könyvtárra (hasonlóan a MATLAB-hoz).

A Scipy tudományos és szimbolikus számítások elvégzésére alkalmas eljárásokat tartalmaz, amelyek a Numpy csomagra épülnek. Az 10.2. ábrán a két könyvtár egymásra épülése és az egyes komponensei láthatóak.

A könyvtárak a <https://www.numpy.org/> és <https://www.scipy.org/> oldalakon érhetőek el.

10.5. Rendszerek

A három óriás, MATLAB, Maple és Mathematica mellett további (ingyenes) számítógépes algebrai algoritmusokat tartalmazó és azok fejlesztését támogató rendszer létezik. Ezek közül szemezgetünk néhányat.

10.5.1. PARI/GP

A PARI/GP egy olyan számítógépes algebrai rendszer, amely elsősorban számelméleti műveletek hatékony végrehajtását támogatja. De ezen felül kezeli a mátrixokat, polinomokat, hatványsorokat, algebrai számokat, stb. A PARI részt C nyelven írták, a GP pedig egy ehhez tartozó szkript nyelv, amely interaktív végrehajtást tesz lehetővé. A GP nyelven keresztül hozzáférhetünk az összes PARI eljáráshoz.

Elérhető a <http://pari.math.u-bordeaux.fr/> címen.

10.5.2. SageMath

A SageMath egy teljesen ingyenes open source matematikai rendszer. Python-alapú interfészen keresztül érhető el, és magába foglal több másik open source matematikai programcsomagot. A mottó:

„Creating a viable free open-source alternative to Magma, Maple, Mathematica and Matlab”. Többek között az MPIR, MPFR, MPC, MPFI, PARI és NTL csomagokat egyaránt használja.

Elérhető a <http://www.sagemath.org/> oldalon.

Példa. Illusztrációképpen bemutatjuk, hogyan lehet a kiterjesztett euklidészi algoritmust, az Euler féle ϕ függvényt, valamint a kínai maradéktételt elérni, illetve használni SageMath-ban:

```
sage: d,u,v = xgcd(12,15)
sage: d == u*12 + v*15
True
sage: n = 2005
sage: inverse_mod(3,n)
1337
sage: 3 * 1337
4011
sage: prime_divisors(n)
[5, 401]
sage: phi = n*prod([1 - 1/p for p in prime_divisors(n)]); phi
1600
sage: euler_phi(n)
1600
sage: x = crt(2, 1, 3, 5); x
11
sage: x % 3 # x mod 3 = 2
2
sage: x % 5 # x mod 5 = 1
1
```

10.6. Online anyagok

10.6.1. NIST DLMF

A NIST 'Digital Library of Mathematical Functions' projektje a híres Abramowitz és Stegun szerzőpáros *Handbook of Mathematical Functions* című könyvének újraírását tűzte ki célul.

A honlap elérhető a <http://dlmf.nist.gov/> címen, 36 fejezetből áll. Érdekes módon egyébként nem csak matematikai függvények definícióit tartalmazza, hanem algoritmusok (általában nagyon) rövid leírását is, hivatkozással. A kapcsolódó könyv 968 oldalas.

10.6.2. Wolfram Functions

A Wolfram Functions honlap szintén matematikai függvények leírását tartalmazza: definíciókat, speciális értékeket, általános tulajdonságokat, sorként történő ábrázolásukat, integrálokat, differenciál egyenleteket, stb.

Elérhető a <http://functions.wolfram.com/> címen.

10.6.3. ESF, DDMF

Az 'Encyclopedia of Special Functions' és a 'Dynamic Dictionary of Mathematical Functions' szintén matematikai függvények gyűjteménye, a megjelenített képletek és ábrázolások azonban közvetlenül egy számítógépes algebrai rendszerből jönnek. Így a felhasználónak lehetősége van bizonyos paraméterek beállítására is és az eredmények interaktív lekérdezésére.

A két gyűjtemény elérhető a <http://algo.inria.fr/online.html> oldalról.

10.6.4. Interval Computations

A jegyzetben nem volt szó intervallum matematikáról, de azért itt megemlítjük a kapcsolódó gyűjtőoldalt, ahol bevezető anyagok, szoftverek, könyvek, kurzusok, konferenciák és alkalmazások vannak összegyűjtve.

Az portál az <http://www.cs.utep.edu/interval-comp/> címen érhető el.

10.6.5. OEIS

Az 'Online Encyclopaedia of Integer Sequences' pedig egy rendkívül gazdag és kimerítő adatbázisként szolgál a legkülönbözőbb egész- és racionális sorozatokkal kapcsolatban. Az OEIS azonosító lényegében standard hivatkozás. Nemcsak cím szerint kereshető, de amennyiben a keresőjébe az illető sorozat első néhány tagját beírjuk, akkor is képes megtalálni az esetlegesen illeszkedő sorozatokat. Minden sorozatot külön oldalon tárgyal, általában néhány tagot megadva, valamint a képletét (ha van), elnevezéseit, és az ismert tulajdonságait felsorolva szakirodalomra való hivatkozásokkal együtt.

Elérhető a <https://oeis.org/> címen.

10.7. Feladatok

1. Telepítsünk különböző BLAS könyvtárakat és teszteljük le különböző méretű mátrixokra vonatkozó műveletek (leginkább szorzás) végrehajtási ideje közötti különbségeket.
2. Implementáljunk gráf algoritmusokat mátrixok (és így BLAS könyvtárak) felhasználásával)
3. Keressük fel a 'More digits friendly' nevű verseny honlapját (<http://rnc7.loria.fr/competition.html>), ahol megtalálható az a 16 feladat, amelyet tetszőleges pontosságú aritmetikát használó programcsomagok tesztelésére javasoltak a szervezők. Válasszunk ki 2 ilyen csomagot és végezzünk el minél többet a tesztek közül.
4. Végezzünk futási idő összehasonlításokat a SageMath és a MATLAB, Maple és Mathematica csomagok között. Például: nagyméretű mátrixok szorzása, mátrixok SVD fölbontása, nagyméretű bináris mátrixok szorzása, prímtesztelés, egészek faktorizálása, nagyméretű hatványok kiszámítása és ilyen tagok összeszorozása.
5. Próbáljuk meg megkeresni a kedvenc numerikus programozási környezetünkben használt BLAS könyvtárat, és lecserélni egy másikkal.

II. rész

AZ OPTIMALIZÁLÁS ALKALMAZÁSAI

Előszó a II. részhez

A jegyzet II. része a Szegedi Tudományegyetemen 2004-től tartott Az optimalizálás alkalmazásai című tárgy anyagát tartalmazza. A tárgy heti két óra előadást és egy óra gyakorlatot jelent.

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

<http://www.inf.u-szeged.hu/~csendes/optalk.pdf>

címen.

A tárgy olyan tudást kíván adni, amely elegendő egyszerűbb optimalizálási, operációkutatási munkák elvégzéséhez, és amelyet önálló gyakorlással továbbfejlesztve egy-egy ilyen feladat teljes megoldását meg lehet találni. Mivel az érintett eljárások, programsomagok gyakran változnak, az anyag főleg az állandó vagy kevésbé változó ismereteket tartalmazza.

A rendelkezésre álló rövid idő (kb. 14×3 óra) nem elég a teljes körű tárgyalására, ezért a legfontosabb definíciókat, összefüggéseket és az elméletet az érintett eljárások tárgyalása előtt csak a feltétlenül szükséges terjedelemben ismertetjük. 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 modellek és eljárások megismerése. A teljes itt közreadott anyag több, mint amit egy féléves kurzusban át lehet adni, ez némi rugalmasságot követel az előadótól, illetve a gyakorlatvezetőtől.

További cél segítséget nyújtani az optimalizálási, operációkutatási vizsgálatokhoz 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 kreditrendszerre való áttéréssel megváltozott a tantárgyak teljesítésének feltétele is. A félreértések, tévedések elkerülése céljából hasznos előre áttekinteni az Informatikai Intézet vendégoldalán hogy mi mindent kell ahhoz teljesíteni, hogy a vizsgára bocsáthatósági szintet el lehessen érni.

Itt mondok köszönetet korábbi hallgatóimnak és munkatársaimnak a jegyzet létrejöttéhez, illetve a javításához nyújtott segítségükért. Várom a további véleményeket és javaslatokat is.

11. fejezet

Alapfogalmak

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 Európában 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 vállalatnál 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.

11.1. Intervallum matematika

11.1.1. Intervallum-aritmetika és a befoglaló függvények

Legyen $\mathbb{I}$ a kompakt valós intervallumok tere. Az intervallum-aritmetika műveletei ezen a halmazon vannak értelmezve. A műveleteket úgy kell definiálni, hogy az $A \circ B$ 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 $c = a \circ b$. Itt $\circ$ a négy alpművelet 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. Ezekben a nullát tartalmazó intervallummal való osztás is értelmezhető.

Bár az alpműveletek 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 $F(X) : \mathbb{I}^n \rightarrow \mathbb{I}$ az $f(x)$ n -változós valós függvény befoglaló függvénye, ha $f(x) \in F(X)$ érvényes minden $x \in X$ pontra és $X \in \mathbb{I}^n$ 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 $X \subseteq Y$ -ből 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 $w(F(X)) - w(\bar{f}(X)) \leq cw(X)^\alpha$ teljesül minden $X \in \mathbb{I}^n$ -re, ahol $w(X)$ az X intervallum szélessége, és $\bar{X}$ az X intervallum felső korlátja. 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 előállítás.

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 monotonitás-vizsgálatra 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.

Az intervallum matematikát részletesen tárgyaló 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, Bibliographisches 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 információra 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.

Esszé írásra a kapcsolódó 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, Prekondicionálás (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: $A \circ B := \{a \circ b : a \in A, b \in B\}$; $A, B \in \mathbb{I}$, ahol $\mathbb{I}$ a valós kompakt intervallumok halmaza (azaz olyan (i, j) pároké, amelyekre $i, j \in \mathbb{R}$, és $i \leq j$).

Aritmetikai definíció:

$$\begin{aligned} [a, b] + [c, d] &= [a + c, b + d] \\ [a, b] - [c, d] &= [a - d, b - c] \\ [a, b] * [c, d] &= [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] \\ [a, b]/[c, d] &= [a, b] * [1/d, 1/c], \text{ ha } 0 \notin [c, d]. \end{aligned}$$

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:

- az $+$ és a $-$, illetve az $*$ 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 $a \in \mathbb{R}$ 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 $A \in \mathbb{I}^n$ intervallum szélességét a koordinátánkénti intervallumok szélességének maximumaként: $w(A) := \max(w(A_i) \mid i = 1, \dots, n)$, ha $A = (A_1, A_2, \dots, A_n) \in \mathbb{I}^n$. 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 következők szerint: $m(A) = (a + b)/2$, ha $A \in \mathbb{I}$, és $m(A) = (m(A_1), m(A_2), \dots, m(A_n))$, ha $A \in \mathbb{I}^n$. Ekkor $m(A \pm B) = m(A) \pm m(B)$, ha $A, B \in \mathbb{I}^n$.

11.1.2. 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 $Y = Y_1 \times \dots \times Y_n$ -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, általában a globális minimum legjobb ismert felső korlátja) 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

Itt S5 - RCOS standard globális optimalizálási tesztfüggvények, a hatékonyságot jelző mutatók pedig: STU – standard időegység, NFE – A függvényhívások száma, NDE – a deriválthívások száma, LLI – a maximális listahossz, és EFF az intervallumos módszer relatív hatékonysága, a hagyományos, sztochasztikus algoritmusokhoz képest.

11.1.3. 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]	0,0000000000004

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 könyvből¹ az intervallumos Newton módszernek az adott példára való használatával:

¹Hammer, R. M. Hocks, U. Kulisch, D. Ratz: C++ Toolbox for Verified Computing. Springer, Berlin, 1995

```

#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);               /* operator '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 */
            } while (y != y_old); /* & is the intersection */
        }
        else {
            cout << "Criterion not satisfied! \n";
        }
        cout << "Please enter starting interval: ";
        cin >> y;
    }
}

```

11.1.4. 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 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 kvadrátikus függvény minimumát abszolút pontosan. A kifelé kerekítés ezen nyilván ront, de ezzel együtt is nagyon hatékony eszköz ez az optimalizálásban. Nyilván általában nem kvadrátikus 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 kerekítést 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 ez 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

$$\begin{aligned} X_1 &= \left(m(X_0) - \frac{f'(m(X_0))}{F''(X_0)} \right) \cap X_0 = \left(0,995 - \frac{-0,0098505}{[1,4048, 2,4812]} \right) \cap [0,98, 1,01] = \\ &= (0,995 + [0,003970, 0,007012]) \cap [0,98, 1,01] = [0,99897, 1,002012] \cap [0,98, 1,01] = \end{aligned}$$

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

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

PÉLDA. A SIAM (Ipári és Alkalmazott Matematikai) Társaság 2002-ben 10 numerikus feladatot tűzött ki². 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:

$$[-3.306868647475316, -3.306868647475196].$$

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.

11.2. 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 érzékenységvizsgálatban 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-XSC³) is támogatják az automatikus differenciálást megfelelő adattípussal és műveletekkel, és számos szoftver is használja ezt a deriválási módszert⁴.

11.2.1. 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 könyve⁵, 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.

²Nick Trefethen: A Hundred-Dollar, Hundred-digit Challenge. SIAM News 35(2002)

³Klatte, R., U. Kulisch, M. Neaga, D. Ratz, Ch. Ullrich: PASCAL-XSC, Springer-Verlag, Berlin, 1991.

⁴Pl. a D. Ratz: Automatische Ergebnisverifikation bei globalen Optimierungsproblemen. (Doktori értekezés, Karlsruhei Egyetem, 1992.) című disszertációban leírt optimalizálási eljárás, amely csak a célfüggvény megadását igényli.

⁵Skeel, R.D., J.B. Keiper: Elementary Numerical Computing with MATHEMATICA. McGraw-Hill Inc., New York, 1993.

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
- + 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 számbábrázolás, 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 időben⁶.

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,

⁶Lásd 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.

11.1. táblázat. Néhány alapl művelet 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 levágási hiba miatt nem vesznek el értékes jegyek,
- + a gyorsan változó deriváltak meghatározására is alkalmas, és
- + a deriváltak kiszámításának műveletigénye általában kisebb, mint a numerikus deriválásé, illetve az analitikus deriváltakat kiszámító szubrutinoké.

Maga az ötlet nem nagyon bonyolult, és jellemző módon többen egymástól függetlenül rátaláltak⁷. 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).

11.2.2. 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 11.1 Táblázat egyes alapl műveletek 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 négyzetre-emelést szorzással értelmezve az $(1, 1) * (1, 1) = (1, 2)$ párt kapjuk, amelyből kiolvasható, hogy $f(2) = 1$, és $f'(2) = 2$.

11.2.3. Kiterjesztések.

A képlettel megadott függvények differenciálásával szemben szokás kiemelni az "algoritmusok differenciálását". Ezen az automatikus differenciálás egyszerű kiterjesztését értik feltételes utasításokat is

⁷Ostrovskij, 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.

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.

11.2.4. 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

$$\begin{aligned} du &= h'(x)dx, \\ dv &= k'(x)dx, \\ dw &= [g_u(u, v)h'(x) + g_v(u, v)k'(x)]dx, \\ dy &= f'(w)[g_u(u, v)h'(x) + g_v(u, v)k'(x)]dx \end{aligned}$$

sorrendet követi.

A fordított eljárás az ellentétes irányban alkalmazza a láncszabályt:

$$\begin{aligned} dy &= f'(w)dw, \\ dy &= f'(w)[g_u(u, v)du + g_v(u, v)dv], \\ dy &= f'(w)[g_u(u, v)du + g_v(u, v)k'(x)dx], \\ dy &= f'(w)[g_u(u, v)dh'(x) + g_v(u, v)k'(x)]dx. \end{aligned}$$

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:

$$\begin{aligned} f_1 &= x_1 = 2 & d_1 &= (1, 0), \\ f_2 &= x_2 = 1 & d_2 &= (0, 1), \\ f_3 &= 1 & d_3 &= (0, 0), \\ f_4 &= f_3 - f_2 = 0 & d_4 &= d_3 - d_2 = (0, -1), \\ f_5 &= f_4^2 = 0 & d_5 &= 2f_4d_4 = (0, 0), \\ f_6 &= f_1f_5 = 2 & d_6 &= f_1d_5 + d_1f_5 = (0, 0). \end{aligned}$$

11.2. 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, $L(\cdot)$: az argumentumok meghatározásának műveletigénye a $\{+, -, *, /, \sqrt{\cdot}, \log, \exp, \sin, \cos\}$ alapl műveletek felett, és $S(\cdot)$: az argumentumok meghatározásának tárigénye.

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

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:

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

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

11.2.5. Művelet- és tárigény.

A 11.2 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).

11.2.6. Az automatikus differenciálás veszélyei.

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

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

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

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

Automatikus differenciálással (is) $\lim_{k \rightarrow \infty} f'_k(0) = 1$, 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 állnak⁸.

11.2.7. Az automatikus differenciálás implementálása.

A már említett PASCAL-XSC beépített adattípusainak és kiterjesztett alapl műveleteinek 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;

:
```

⁸Fischer, 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.

```

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.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 Tennessee, 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.

11.3. Ellenőrző kérdések és gyakorló feladatok

1. Ki lehet-e terjeszteni a négy alpműveletek valós számokról intervallumokra?
2. Igaz-e, hogy két intervallum összege pontosan azokat a pontokat tartalmazza, amelyek előállnak a két argumentum-intervallumbeli pontok összegeként?

3. Milyen információra támaszkodik a monotonitási teszt?
4. Mi okozza a befoglaló függvények durva becslését?
5. Mi 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 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 alaplóműveletet általánosítani kompakt valós intervallumokra!
32. Igaz-e, hogy ha X része Y -nak, akkor minden befoglaló függvényre $F(X)$ is része $F(Y)$ -nak?
33. Igaz-e, hogy egy szigorúan monoton függvény deriváltjának befoglaló függvénye nem tartalmazza a nullát?
34. Milyen intervallumot kapunk eredményül, ha az $f(x) = (2x - 1) * (x^2 - x)$ függvény természetes intervallum-kiterjesztését a $[-1, 1]$ intervallumon kiértékeljük?
35. Milyen információra támaszkodik az intervallumos korlátozás és szétválasztás típusú globális optimalizálási algoritmusban a monotonitási teszt?
36. Definiálja a természetes intervallum kiterjesztést, és mutasson rá példát!
37. Ismertesse az intervallum aritmetika néhány, a valós aritmetikáétól eltérő algebrai tulajdonságát!
38. Írja meg az automatikus differenciálás szubrutinjait, és tesztelje néhány függvényen!

12. fejezet

A hátizsák feladat

Adottak szállítandó tárgyak súllyal (vagy térfogattal) és értékkel (vagy fontossággal). A feladat az, hogy meghatározzuk a hátizsákba beteendő holmiknak azt a részhalmazát, amelyek az előző értelemben a leghasznosabbak, és együtt beférnek a korlátozott kapacitású hátizsákba.

Használjuk a következő jelölést:

- m a tárgyak száma,
- a_i az i . tárgy súlya, $i = 1, 2, \dots, m$,
- c_i az i . tárgy értéke, $i = 1, 2, \dots, m$,
- b a rakomány megengedett maximális összsúlya.

Legyen x_i értéke 1, ha az i -edik tárgy bekerült a hátizsákba, és 0, ha nem ($i = 1, 2, \dots, m$).

A megoldandó feladat ezekkel felírva:

$$\max \sum_{i=1}^m c_i x_i,$$

feltéve, hogy

$$\sum_{i=1}^m a_i x_i \leq b, \text{ és}$$

$$x_i \in \{0, 1\}, \quad i = 1, 2, \dots, m.$$

A hátizsák feladat tehát egy egészértékű, bináris lineáris programozási feladat. Egy egyszerű kiterjesztése adódik akkor, ha az elhelyezendő tárgyak között vannak azonosak. Ekkor az optimalizálandó változók értékei nemnegatív egészek lehetnek.

A feladat jellege miatt a kiindulási feladatra legtöbbször fel lehet tenni azt, hogy a súlyok és a súlyhatár nemnegatívak. Ennek ellenére mind az a_i , mind a c_i értékek előjele tetszőleges.

12.1. A hátizsák feladat megoldása teljes leszámolással

A hátizsák feladatot megoldhatjuk a durva erő módszerével (brute force, enumeration). Ennek lényege, hogy felsoroljuk az összes változó-kombinációt, meghatározzuk a lehetséges megoldásokra a célfüggvény értékét, és kiválasztjuk ez alapján az optimálisat.

Az összes változó-kombinációt például lexikografikus sorrendben határozhatjuk meg. Négy bináris változó esetén az így kapott sorozat:

$$(0, 0, 0, 0), (0, 0, 0, 1), \dots, (1, 1, 1, 1).$$

Az eljárás hátránya, hogy nagyszámú változó esetén kezelhetetlenül sok esetre kell ellenőrizni a feltételt, és ez a szám a változók számával exponenciálisan nő.

PÉLDA. Tekintsük a következő feladatot. Legyen a tárgyak súlya rendre 2, 3 és 4, a hasznossága pedig 5, 3, 1, míg a megengedett összsúly 5. A megoldást tartalmazó táblázat (a lehetséges megoldásokat félkövér betű, az optimálist csillag jelzi):

javasolt megoldás	a súlya	az értéke
(0,0,0)	0	0
(0,0,1)	4	1
(0,1,0)	3	3
(0,1,1)	7	4
(1,0,0)	2	5
(1,0,1)	6	6
(1,1,0)*	5*	8*
(1,1,1)	9	9

Vegyük észre, hogy a megoldást a mohó módszerrel is megkapnánk: addig vennénk a legértékesebb tárgyakat csökkenő hasznossági sorrendben, amíg azt a súlyhatár megengedi.

A feladatnak megfeleltethető a következő, többé-kevésbé reális probléma: adott egy komp, ennek teherbírása 5 tonna. Három jármű vár az átvitelre: egy 2 tonnás személyautó, egy 3 tonnás kisteherautó, és egy 4 tonnás szekér. A viteldíjak rendre 500, 300, illetve 100 forint. Üzleti okokból kíváncsiak vagyunk az előírásoknak megfelelő, legnagyobb bevételt jelentő fuvarra.

12.2. A hátizsák feladat megoldása közvetett, implicit leszámolással

A megoldási módszer lényege, hogy a teljes leszámolást ésszerűen gyorsítjuk a feltétel és az értékek figyelembe vételével. Többek között azokat a kiértékeléseket tudjuk megtakarítani, amelyek egyértelműen nem lehetséges megoldásokhoz tartoznak. Például, ha kiderült, hogy a

$$(0, 1, 0, \dots, 1, 0, 0)$$

megsérti a súlykorlátot, akkor nincs értelme a több egyest tartalmazó

$$(0, 1, 0, \dots, 1, 0, 1)$$

ellenőrzésének – ha a megfelelő súlyok pozitívok.

1. Első lépésként alakítsuk át a feladatunkat egy könnyebben áttekinthető, kanonikus alakúra: legyen minden célfüggvény együttható nem pozitív, és a súlyok növekvő sorrendbe rendezettek: $a_1 \leq a_2 \leq \dots \leq a_m$.

Az első feltételt az $x'_i = 1 - x_i$ helyettesítéssel lehet elérni olyan változókra, amelyekre az nem teljesült. Ezeket a változókat természetesen meg kell jegyezni, és az eljárás végén a kapott optimális értékeket megfelelően vissza kell alakítani. A második tulajdonság teljesüléséhez a változókat kell csak alkalmasan átrendezni (és a megoldás után vissza).

2. Az x indulóvektornak válasszuk a nulla vektort. A keresés során ettől haladunk a keresési fa levelei felé, amelyek utolsó komponense egyes.

3. Ha $\sum_{i=1}^m a_i x_i \leq b$ teljesül, akkor adjuk x -et a lehetséges megoldások L halmazához. Hagyjuk ki az ellenőrizendő csúcsok közül azokat, amelyekre a jelen x utolsó nullái egyikének c_i együtthatója negatív.

Ha ez az egyenlőtlenség nem igaz, akkor ugorjuk át mindazokat a vektorokat a keresésben, amelyek előállnak úgy, hogy x utolsó nulla elemei egyike helyett egyes szerepel, és a megfelelő súly pozitív.

4. Generáljunk egy újabb vektort! Ha van még ilyen, akkor folytassuk a 3. Lépéssel. Az új vektor generálása során ha lehet, akkor olyant választunk, amely az előző x vektor utolsó nulla jegyeit módosítva kapható. Ha ez már nem lehetséges, akkor visszatérünk egy korábban félbehagyott ághoz a keresési fában.

5. A legnagyobb célfüggvény értékű talált lehetséges megoldás az optimális megoldás az x^* pontban. Az 1. Lépésben végrehajtott átalakításoknak megfelelően az eredményt visszatranszformáljuk.

PÉLDA. A korábban vizsgált példát most nem érdemes használni, mert könnyen látható, hogy abban nincs lehetőség a keresés gyorsítására, mivel a nem lehetséges megoldások mind a keresési fa levelein vannak.

Tekintsük ezért a következő, kicsit módosított feladatot. Legyen a tárgyak súlya rendre 4, 3 és 3, a hasznossága pedig -5, 3, -1, míg a megengedett összsúly 2.

Kövessük a közvetett leszámolási algoritmus lépéseit.

1. Írjuk fel az eredeti feladatot:

$$\max \sum_{i=1}^3 c_i y_i = \max -5y_1 + 3y_2 - y_3,$$

a feltétel pedig

$$\sum_{i=1}^3 a_i y_i \leq b, \quad \text{azaz} \quad 4y_1 + 3y_2 + 3y_3 \leq 2.$$

A kanonikus alakot úgy kapjuk, hogy az y_2 változót $1 - x_1$ -el helyettesítjük (és c_2 új értéke -3 lesz). Ezzel párhuzamosan a súlyok növekvő sorrendje eléréséhez cseréljük fel a változókat:

$$x_1 = 1 - y_2, \quad x_2 = y_3, \quad x_3 = y_1.$$

Az új feladat ezután:

$$\begin{aligned} \max & -3x_1 - x_2 - 5x_3 + 3, \\ & -3x_1 + 3x_2 + 4x_3 \leq -1. \end{aligned}$$

2. Az indulóvektor $x^T = (0, 0, 0)$.

3. A $(0,0,0)$ vektorra a fenti feltétel nyilvánvalóan nem teljesül, ezért a további keresésből kizárjuk a $(0,1,0)$, $(0,1,1)$ és $(0,0,1)$ eseteket is, mivel az utolsó két súly pozitív, tehát ezekre az esetekre sem teljesülhet a feltételünk.

$$\begin{aligned} \max & -3x_1 - x_2 - 5x_3 + 3, \\ & -3x_1 + 3x_2 + 4x_3 \leq -1. \end{aligned}$$

4. A következő keresési vektor $x^T = (1, 0, 0)$.

3. Az $(1, 0, 0)$ vektor súlya $-3 \leq -1$, tehát $L = \{(1, 0, 0)\}$. Az ehhez a vektorhoz tartozó cél-függvényérték 0. Mivel az $(1, 0, 0)$ vektor hátsó nullái mindegyikéhez negatív célfüggvény-együttható tartozik, ezért más vektort már nem is kell megvizsgálni: a további lehetséges megoldások célfüggvény-értéke kisebb lenne a megtaláltnál.

5. Az átalakított feladat optimális megoldása tehát $(1, 0, 0)$. Az 1. Lépés átalakítása alapján az eredeti feladat optimális megoldása

$$y_1 = x_3 = 0,$$

$$y_2 = 1 - x_1 = 0,$$

$$y_3 = x_2 = 0.$$

Az ehhez tartozó célfüggvényérték pedig természetesen 0.

Az eredmény szépen értelmezhető az eredeti

$$\max -5y_1 + 3y_2 - y_3,$$

$$4y_1 + 3y_2 + 3y_3 \leq 2$$

feladatra. Eszerint a célfüggvény optimauma a teljes, feltétellel nem korlátozott tartományon a $(0, 1, 0)$ pontban lenne. Ez azonban nem lehetséges megoldás, mert a második komponens miatt az előírt feltétel nem teljesül. Az egyenlőtlenséget az $y_2 = 0$ választással teljesíthetjük, ez pedig épp a kapott megoldást adja. Mivel a célfüggvény együtthatók és a súlyok egyike sem nulla, ezért más megoldás nincs is.

12.3. Kapcsolódó feladatok

A hajórakodási feladat

Korlátozott térfogatú szállítóeszköz esetén a hátizsák feladat feltételéhez a térfogatra vonatkozót is meg kell adni. Kövessük a korábbi jelölést:

- m a tárgyak száma,
- a_{1i} az i . tárgy súlya, $i = 1, 2, \dots, m$,
- a_{2i} az i . tárgy térfogata, $i = 1, 2, \dots, m$,
- d_i az i . tárgyból rendelkezésre álló mennyiség,
- c_i az i . tárgy értéke, $i = 1, 2, \dots, m$,
- b_1 a rakomány megengedett maximális összsúlya,
- b_2 a rakomány megengedett maximális össztérfogata.

Legyen x_i értéke ismét 1, ha az i -edik tárgy bekerült a rakományba, és 0, ha nem ($i = 1, 2, \dots, m$). Ha több i -edik tárgy is van a rakományban, akkor x_i értéke legyen a megfelelő egész szám.

A megoldandó feladat ezekkel felírva:

$$\max \sum_{i=1}^m c_i x_i,$$

feltéve, hogy

$$\sum_{i=1}^m a_{ji} x_i \leq b_j, \quad j = 1, 2,$$

$$x_i \leq d_i \text{ és}$$

$$x_i \text{ egész, } i = 1, 2, \dots, m.$$

Állapítsuk meg, hogy a hajórakodási feladat is megoldható a korábban ismertetett implicit leszámolással, de módosítást kell rajta végrehajtani. Bár elvileg a két feltétel egymástól függetlenül is érvényesíthető az ellenőrizendő vektorok számának csökkentésére, de a súlyok és a térfogatok értékei egyszerre nem feltétlenül rendezhetők növekvő sorrendbe. Ennek ellenére a kapott eljárás általában hatékonyabb lesz, mint a teljes leszámolás.

Vegyük észre, hogy hasonló módon további feltételek is érvényesíthetők, pl. a rakomány összhosszára stb.

A fix költség feladat

A termelő, szolgáltató vállalkozások a költségeik csökkentésére törekszenek. Tekintsük azt a feladatot, amely a fix és termeléssel arányos költségek viszonyát vizsgálja.

Használjuk a következő jelölést:

m	a termékek száma
x_i	az i . termék gyártandó mennyisége
d_i	az i . termék gyártási korlátja
a_{ij}	a j . termék egységnyi mennyiségének gyártásához az i . erőforrásból felhasznált mennyiség
b_i	az i . erőforrásból rendelkezésre álló mennyiség
c_i	az i . termék fix költsége (vagy nulla)
$k_i(x_i)$	az i . termék mennyiségtől függő termelési költsége

A matematikai modell ezek alapján:

$$\min \sum_{i=1}^m f_i(x_i) = \min \sum_{i=1}^m c_i + \sum_{i=1}^m k_i(x_i)$$

feltéve, hogy

$$0 \leq x_i \leq d_i, \text{ és}$$

$$Ax \leq b.$$

A feladat nemlineáris $k_i(x)$ esetén szeparábilis, lineárisan korlátozott nemlineáris optimalizálási feladat. Amennyiben a gyártandó termékek mennyisége egész, akkor ráadásul egészértékű (vagy vegyes egészértékű) nemlineáris optimalizálási problémával állunk szemben.

Gyakori eset, hogy a nem fix beruházási függvényrész $x_i^{0.6}$, vagy hasonló alakú, és így feladatunk a konkáv optimalizálás területére tartozik.

12.4. Ellenőrző kérdések és gyakorló feladatok

1. Mutasson olyan értelmes gyakorlati feladatot, amely olyan hátizsák feladatra vezet, amelyben a célfüggvény együtthatók mind negatívak!

2. Hogyan fogalmazná át a teljes leszámolás módszerét a hátizsák feladat azon esetére, amikor több azonos tárgyat kell elhelyezni?
3. Mutasson példát, amelyre az implicit leszámolási eljárás az első lehetséges megoldással meg is találja az optimális megoldást!
4. Mutasson példát, amelyre az implicit leszámolási eljárás is minden szóbajövő vektort meg kell hogy vizsgáljon ahhoz, hogy megtalálja az optimális megoldást!
5. Keressen gyakorlati példát arra az esetre, amikor a hátizsák feladat célfüggvény együtthatói között vannak pozitív és negatív előjelűek is!
6. Mi adja a lényegi eltérést a hátizsák- és a hajórakodási feladat között?
7. Ha egy 2 Ghz-es PC 10 órajel alatt tudja egy vektorról eldönteni, hogy az lehetséges megoldása-e a hátizsák feladatnak, akkor egy nap alatt milyen méretű feladat megoldására lehet biztosan számítani (tehát a legrosszabb esetben)?
8. Keressen reális alkalmazási feladatot, amely olyan hátizsák feladatra vezet, amelyben negatív és pozitív súlyok is kellenek!
9. Indokolja, hogy a fix költség feladat miért nem vezethető vissza közvetlenül a hátizsák feladatra!
10. Adjon meg egy olyan hátizsák feladat osztályt, amely minden elemére nem negatív az optimális célfüggvény érték!
11. Mit lehet mondani annak a hátizsák feladatnak a megoldásairól, amelyben minden súly és minden célfüggvény együttható is megegyezik?
12. Jellemezze a hajórakodási feladatoknak azt a részhalmazát, amely megfeleltethető a hátizsák feladatnak!
13. Tekintsük a $\max x_1 + 2x_2, \quad 2x_1 + x_2 \leq 2$ hátizsák feladatot. Milyen mértékben lehet megváltoztatni a súlykorlátot ahhoz, hogy a megoldás ne változzon? Mi a helyzet a feladat többi paraméterével?
14. Tegyük fel, hogy egy hátizsák feladatban a legnagyobb célfüggvényértékhez tartozó tárgyak összsúlya a megadott korlát alatti. Mit mondhatunk ekkor az optimális megoldásról?
15. Ha a hátizsák feladatban megadott legfontosabb tárgyak összsúlya épp a súlyhatárt adja, akkor mi az optimális megoldás?
16. Mutasson olyan hátizsák feladatot, amelynek optimális megoldásában a legkisebb értékű tárgy is benne van!
17. Igaz az, hogy bármely bináris vektorhoz konstruálható olyan hátizsák feladat, amelynek ez egyetlen optimális megoldása? És olyan, amelynek csak ez nem optimális megoldása?
18. Mi a hátránya a Monte Carlo módszernek (egyenletes eloszlással generálunk vektorokat, és a talált legjobb célfüggvényértékű vektort megjegyezzük) a hátizsák feladat megoldása során?
19. Oldja meg fejben a $\max x_1 + 2x_2, \quad 2x_1 + x_2 \leq 2$ hátizsák feladatot! Melyik módszert használta?
20. Érveljen a teljes leszámolás módszere mellett: mikor előnyösebb az, mint az implicit leszámolás?
21. Mennyi a műveletigénye az implicit leszámolás első, előkészítő lépésének?

22. Mi a megoldása annak a hátizsák feladatosztálynak, amelyben minden súly, érték és súlyhatár megegyezik?
23. Miért nincs értelme a hátizsák feladatnak $m = 1$ esetén?
24. Mutassa meg, hogy a fix költség feladat speciális eseteként előáll a hátizsák feladat!
25. A levezetett számolási példára növelje a súlykorlátot, és tárgyalja annak hatását a lehetséges megoldások halmazára!

13. fejezet

Az utazó ügynök feladat

Az operációkutatás egy nevezetes, központi feladata az utazó ügynök problémája. Legyenek adottak meglátogatandó városok, ismerjük a köztük lévő távolságokat. A feladat egy olyan minimális hosszúságú útvonal megtalálása, amely minden várost érint, mindegyiken csak egyszer halad át, és a körút végén visszatér a kiindulási városba.

A matematikai modell megfogalmazásában legyen a korábbiaknak megfelelően a meghatározandó változók halmaza $x_{ij} \in \{0, 1\}$, $i, j = 1, \dots, n$, ahol n a városok száma, x_{ij} pedig azt adja meg, hogy az i . és a j . város között áthalad-e az aktuális körút. A feladatot a következők szerint fogalmazhatjuk meg:

$$\min \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij},$$

feltéve, hogy

$$\sum_{t=1}^n x_{it} = 1 \quad (i = 1, \dots, n),$$

$$\sum_{t=1}^n x_{tj} = 1 \quad (j = 1, \dots, n),$$

$$\sum_{i \in Q} \sum_{j \in \{1, \dots, n\} \setminus Q} x_{ij} \geq 1 \quad Q \subset \{1, \dots, n\}, \quad Q \neq \emptyset,$$

$$x_{ij} \in \{0, 1\}, \quad i, j = 1, \dots, n.$$

A célfüggvény a megtett útszakaszok költségét összegzi. Az első feltétel azt követeli meg, hogy az ügynök minden városból kimegy, a második pedig azt, hogy mindegyikbe bejut, mindkét esetben pontosan egyszer. E két feltétel teljesülése esetén még előfordulhat, hogy a kapott útvonal különálló körutakból áll, ami a feladat eredeti megfogalmazásának nem felel meg.

Ezt a problémát rendezzi a következő feltétel. Ha lenne olyan zárt körút, amely nem tartalmazza az összes várost, akkor az ehhez tartozó városok alkotta Q halmazra ez a feltétel nem teljesülne, hiszen ekkor a baloldali összeg nullának adódna.

Állapítsuk meg, hogy a megfogalmazott operációkutatási feladat egy lineáris egyenlőség és egyenlőtlenség korlátokkal ellátott nulla-egy lineáris programozási feladat.

Az utazó ügynök feladat számos alkalmazásban fordul elő kisebb-nagyobb módosítással:

Az egyik legkézenfekvőbb a tömegközlekedés *járatütemezési problémája*. Adjuk meg azt az útvonalat, amelyet egy busznak meg kell tennie ahhoz, hogy bizonyos járatok útvonalán a megfelelő szolgáltatást nyújtsa, ehhez a lehető legrövidebb útvonalat keressük, és a járatok teljesítése után térjen vissza az indulási állomásra.

Hasonló eset a fémmegmunkálásban a szerszámgépek olyan vezérlése, hogy a befogott munkadarab lehető legkisebb mozgatása révén minden részművelet helyét érintse a fúró, szegecselő stb. fej, majd térjen vissza a kiindulási helyzetébe.

Tekintsük azt a problémát, amikor a feladat egy gyár által termelt termékek sorrendjének meghatározása – tekintettel arra, hogy az egyik termék gyártásáról egy másiknak a termelésére való átállásnak időben, vagy direkt költségben eltérő ára van. Természetesen minden terméket le kell gyártani, és a teljes termelési ciklus után ugyanabba a helyzetbe tér vissza a gyártási sorrend.

Bonyolultabb a helyzet, ha repülőgépek és azok személyzete útitervét kell optimálisan meghatározni úgy, hogy megadott városokat érintsenek, a hatékonyság a lehető legjobb legyen, de a szervizelési, pihenési stb. előírásokat betartsák.

Közvetve idetartozik az írógépek, számítógépes billentyűzetek tervezése is. Ekkor adott nyelvi környezetben megméri, hogy két betű egymás utáni előfordulása milyen gyakori. Ennek megfelelően a cél olyan billentyűzetet megadni, amelyre a tipikus szövegek gépelése során a lehető legkevesebbet kell mozgatni a kezünket.

Utazó ügynök feladatot old meg a mentős diszpécser is, amikor meghatározza, hogy a mentő az aznapi betegeket milyen sorrendben vegye fel a lakásukban, szállítsa a dialízis kezelésre, majd vissza otthonukba. Ebben a feladatban a kórháznak kiemelt helye van, nyilván nem lehet azt utolsóként érinteni...

Az utazó ügynök feladata interpretálható úgy is, hogy minimális hosszúságú, minden csúcsot érintő irányított körutat kell meghatározni egy adott gráfban.

Az utazó ügynök feladat korábban ismertetett modellje 2^n darab feletti feltételt tartalmazott, ez valódi feladatok esetén elviselhetetlenül nagy szám. A. Tucker 1960-ban kevesebb feltétellel fogalmazta újra a feladatot:

$$\min \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij},$$

feltéve, hogy

$$\sum_{t=1}^n x_{it} = 1 \quad (i = 1, \dots, n),$$

$$\sum_{t=1}^n x_{tj} = 1 \quad (j = 1, \dots, n),$$

$$u_i - u_j + (n-1)x_{ij} \leq n-2 \quad 2 \leq i \neq j \leq n,$$

$$x_{ii} = 0, \quad x_{ij} \in \{0, 1\}, \quad i, j = 1, \dots, n,$$

$$u_i \geq 0, \quad u_i \text{ egész}, \quad i = 2, \dots, n.$$

A részkör mentességet a 3., új feltétel hivatott biztosítani. Az új feladatnak n^2 nagyságrendű feltétele van. A konstrukció lényege, hogy az u_i számokkal alkalmasan sorszámozott körút elemekre a 3. feltétel csak akkor teljesülhet, ha az teljes körút (vö. a következő bizonyítás vége).

ÁLLÍTÁS. Az utazó ügynök feladat két modellje ekvivalens.

BIZONYÍTÁS. A két feladat célfüggvénye megegyezik, tehát a lehetséges megoldások halmazának megegyezését kell igazolni.

Tekintsük először azt az esetet, hogy az eredeti feladatnak az X mátrix egy lehetséges megoldása:

$$x_{1,i_2} = x_{i_2,i_3} = \dots = x_{i_n,1} = 1, \text{ és}$$

$$x_{ij} = 0 \text{ különben.}$$

Ehhez megkonstruáljuk azt az (X, u) párt, amely lehetséges megoldása lesz a második feladatnak. Mivel X lehetséges megoldása az első feladatnak, ezért ehhez tartozik egy körút, amely az $(1, i_2), (i_2, i_3), \dots, (i_n, 1)$ éleket tartalmazza. Definíáljuk most u értékét a következők szerint:

$$u_{i_t} = t, \quad t = 2, \dots, n.$$

Csak a harmadik feltételrendszert kell igazolni, a többi teljesülése nyilvánvaló. Tekintsünk egy tetszőleges ide való indexpárt: $2 \leq i \neq j \leq n$. Az u definíciójából adódik, hogy $u_i - u_j \leq n - 2$. Másrészt $x_{ij} = 1$ pontosan akkor teljesül, ha i és j a körútban közvetlenül egymás utáni indexek: ha $i = i_r$, akkor $j = i_{r+1}$. Innen erre az esetre

$$u_i - u_j + (n - 1)x_{ij} = r - (r + 1) + (n - 1) = n - 2,$$

tehát a harmadik csoport feltétel is teljesül, így X lehetséges megoldása az első feladatnak.

Tekintsük most azt az esetet, amikor a második feladatnak az (X, u) pár egy lehetséges megoldása, de van egy diszjunkt részkörút, tehát $x_{i_1,i_2} = x_{i_2,i_3} = \dots = x_{i_k,i_1} = 1$, és $1 < k < n$. Az általánosság megszorítása nélkül feltehetjük, hogy $1 \notin \{i_1, i_2, \dots, i_k\}$. Mivel (X, u) egy lehetséges megoldása a második feladatnak, ezért

$$\begin{aligned} u_{i_1} - u_{i_2} + (n - 1)x_{i_1,i_2} &\leq n - 2, \\ u_{i_2} - u_{i_3} + (n - 1)x_{i_2,i_3} &\leq n - 2, \\ &\vdots \\ u_{i_k} - u_{i_1} + (n - 1)x_{i_k,i_1} &\leq n - 2 \end{aligned}$$

teljesül. Az egyenlőtlenségeket összeadva azt kapjuk, hogy $k(n - 1) \leq k(n - 2)$, ami $1 < k < n$ esetén ellentmondás. $\square$

Mivel az utazó ügynök feladat feltételrendszere csak az n számtól függ, ezért a feladatot egyértelműen meghatározza az n szám, és a C költségmátrix.

DEFINÍCIÓ. Azt mondjuk, hogy a C mátrix ekvivalens a D mátrixszal (jelölése: $C \sim D$), ha vannak olyan $\alpha_1, \alpha_2, \dots, \alpha_n$ és $\beta_1, \beta_2, \dots, \beta_n$ számok, hogy igaz $c_{ij} = d_{ij} + \alpha_i + \beta_j$ minden indexpárra.

SEGÉDTÉTEL. Ha $C \sim D$, akkor a C mátrix által meghatározott $TSP(C)$ utazó ügynök feladat optimális megoldása megegyezik a $TSP(D)$ feladatával.

BIZONYÍTÁS. Mivel mindkét feladatnak létezik optimális megoldása, ezért a segédtétel állítása korrekt. A két feladat lehetséges megoldásai halmaza megegyezik, legyen ez L , a két célfüggvény pedig z_C és z_D .

Azt fogjuk megmutatni, hogy létezik olyan γ konstans, hogy $z_C(x) = z_D(x) + \gamma$ teljesül minden x lehetséges megoldásra. Mivel $C \sim D$, ezért vannak olyan $\alpha_1, \alpha_2, \dots, \alpha_n$ és $\beta_1, \beta_2, \dots, \beta_n$ konstansok, hogy $c_{ij} = d_{ij} + \alpha_i + \beta_j$ minden $1 \leq i, j \leq n$ indexpárra.

Ekkor

$$z_C(x) = \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} = \sum_{i=1}^n \sum_{j=1}^n (d_{ij} + \alpha_i + \beta_j) x_{ij} =$$

$$\sum_{i=1}^n \sum_{j=1}^n d_{ij} x_{ij} + \sum_{i=1}^n \alpha_i \sum_{j=1}^n x_{ij} + \sum_{j=1}^n \beta_j \sum_{i=1}^n x_{ij}.$$

Mivel x lehetséges megoldás, ezért mindkét utóbbi második szumma egy:

$$\sum_{j=1}^n x_{ij} = \sum_{i=1}^n x_{ij} = 1.$$

Ezért aztán $\gamma = \sum_{i=1}^n \alpha_i + \sum_{j=1}^n \beta_j$ megfelelő választás: $z_C(X) = Z_D(X) + \gamma$. Ez épp a segédétel állítását igazolja. $\square$

KÖVETKEZMÉNY. Az optimális megoldás meghatározását illetően elegendő olyan utazó ügynök feladatokat vizsgálni, amelyekre $C \geq 0$ teljesül.

Vegyük észre, hogy a harmadik feltételcsoporttól eltekintve az utazó ügynök feladat a hozzárendelési feladatot adja. Emiatt az utazó ügynök feladat L lehetséges megoldásai halmaza része a megfelelő hozzárendelési feladat S lehetséges megoldásai halmazának: $L \subset S$. Mivel

$$\min\{z(X) : X \in S\} \leq \min\{z(X) : X \in L\},$$

ezért

- i, ha X optimális megoldása a $H(C)$ hozzárendelési feladatnak és X teljes körút, akkor X egyben optimális megoldása a $TSP(C)$ utazó ügynök feladatnak is, és
- ii, Ha X optimális megoldása a $H(C)$ hozzárendelési feladatnak, akkor $z(X)$ egy alsó korlátja a $TSP(C)$ utazó ügynök feladat optimumértékének.

Az 1 - 2 - 3 - 4 - 5 teljes körúthoz tartozó egy hozzárendelési feladat költségmátrixa (M az adott számítógépes környezetben ábrázolható legnagyobb szám):

$$\begin{bmatrix} M & 1 & 5 & 5 & 5 \\ 5 & M & 1 & 5 & 5 \\ 5 & 5 & M & 1 & 5 \\ 5 & 5 & 5 & M & 1 \\ 1 & 5 & 5 & 5 & M \end{bmatrix}$$

A következő költségmátrixhoz tartozó hozzárendelési feladat optimális megoldása (1 - 2 - 3, 4 - 5) nem ad lehetséges megoldást a kapcsolódó utazó ügynök feladatra:

$$\begin{bmatrix} M & 1 & 5 & 5 & 5 \\ 5 & M & 1 & 5 & 5 \\ 1 & 5 & M & 5 & 5 \\ 5 & 5 & 5 & M & 1 \\ 5 & 5 & 5 & 1 & M \end{bmatrix}$$

Az utazó ügynök feladat tulajdonságai miatt számos esetben ésszerű azt feltételezni, hogy a költségmátrix *szimmetrikus*. E szabály alól azonban vannak természetes kivételek. Még városok közti távolság esetén is lehet eltérés az oda és a visszaút között, de még gyakoribb a termékek gyártási sorrendje megválasztása esetén, hogy az A termék gyártásáról a B-re pl. gyorsabban lehet áttérni, mint fordítva.

Az utazó ügynök feladat nagyszámú feltételére, és a leszámolási eljárás reménytelenül nagy műveletigényére tekintettel a feladatot szokás egyrészt *korlátozás és szétválasztás módszerrel* megoldani, másrészt gyors, *heurisztikus közelítő eljárások* alkalmazni.

A korlátozás és szétválasztás módszerét gyorsítani lehet akkor, ha jó korlátokat tudunk megadni az optimum értékére. Ennek eszköze lehet a korábban ismertetett áttérés a megfelelő hozzárendelési feladatra.

Egon Balas és Paolo Toth számítógépes kísérleteket végzett, amelynek során 400 problémát generáltak véletlenszerűen. Egyenletes eloszlással határozták meg a feladat méretét az $50 \leq n \leq 250$ határok között, és a célfüggvény együtthatókat is az 1 és 100, illetve más esetben az 1 és 1000 közötti egészek közül.

Az így előálló feladatokat megoldották mint TSP feladatot, és mint hozzárendelési feladatot is. Azt kapták, hogy a hozzárendelési feladatok optimumértékei átlaga 99.2 %-a volt az utazó ügynök feladat optimumai átlagának. Az eredményekből kitűnt, hogy n növekedésével a relatív eltérés csökkent.

Ezzel együtt az utazó ügynök feladat az ún. NP nehéz feladatok közé tartozik, tehát nem ismeretesek azt az n feladatméret polinom függvényével korlátozott idő alatt megoldó eljárások. Emiatt előtérbe kerültek a közelítő módszerek, amelyek csaknem optimális megoldást adnak viszonylag rövid idő alatt.

13.1. Az utazó ügynök feladat heurisztikus algoritmusai

A heurisztikus algoritmusok lényege, hogy ezek az eljárások gyorsan, kevés műveletigény árán javítják a közelítő megoldásokat – de arra nem mindig van remény, hogy ezek minden esetre konvergálnának az optimális megoldáshoz.

Legközelebbi város beillesztése: az eddigi részkörutat bővítjük egy olyan újabb várossal, hogy a részkörút városain kívüli városok között megkeressük azt a k -t, amelynek távolsága a legkisebb a részkörút valamely városához. Ezután ezzel bővítjük a részkörutat: Ha c_{ik} volt a kiválasztásban talált minimális távolság, akkor az új részkörútban az i . város után a k . következik, majd ezután az i várost eddig követő i' .

PÉLDA. Tekintsük a legutóbb vizsgált feladatot, amelynek C költségmátrixa

M	1	5	5	5
5	M	1	5	5
1	5	M	5	5
5	5	5	M	1
5	5	5	1	M

Kiinduláshoz vegyük az $\{1, 1\}$ körutat. A többi város ettől mért távolsága rendre 1, 5, 5 és 5. Ez alapján a 2. várossal bővül a részkörút: 1 - 2 - 1.

A következő lépésben tekintsük a 3., 4. és 5. városok távolságait az 1. és 2. városoktól. Ezek a C mátrix főátlója feletti elemek, az első és második sorban (c_{12} kivételével). A minimumot a $c_{23} = 1$ távolság adja, ezzel a kibővített részkörút:
1 - 2 - 3 - 1.

A harmadik lépésben a minimalizálandó távolságok c_{14} , c_{24} , c_{34} , c_{15} , c_{25} és c_{35} . Ezek mindegyike 5, válasszuk a 4. várost a bővítéshez. Ezzel az új részkörút: 1 - 2 - 3 - 4 - 1.

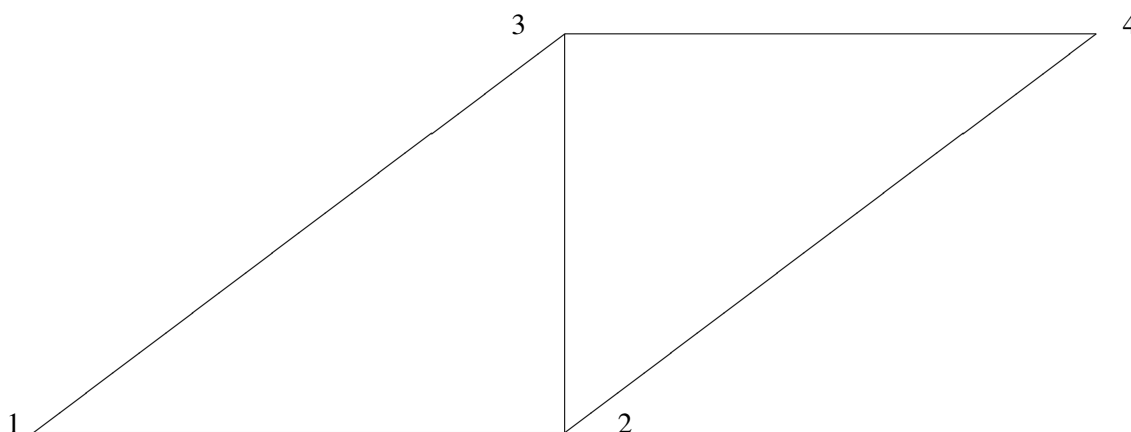
Az utolsó lépésben már csak egy városból választhatunk, és ezt a 4. városhoz kell kötni: 1 - 2 - 3 - 4 - 5 - 1 lesz a heurisztika által talált teljes körút. Ez nyilván lehetséges megoldás, a hozzá tartozó össztávolság 13. Ez egyben optimális megoldás is – bár ez nem jellemző a heurisztikák esetén. $\square$

A legközelebbi város hozzáadása: Az előzőnél egyszerűbb heurisztika: az aktuális útvonalat mohó módon bővíti a meglevő útvonal végpontjához legközelebbi várossal. Ha minden város szerepel már az útvonalban, akkor az utolsót összeköti az elsővel, és így képez teljes körutat.

PÉLDA. Mutassunk most egy olyan példát, amely szuboptimális megoldást eredményez. 3 város esetén ez nem lehetséges, ahogy könnyen belátható. Tekintsük akkor a következő távolságmátrixot:

M	4	5	$\sqrt{73}$
4	M	3	5
5	3	M	4
$\sqrt{73}$	5	4	M

Ez két, a rövidebb oldalával összefordított Pitagorasz-háromszöget tartalmaz. Az így kapott paralelogramma természetes módon ad egy körüljárást, amely 18 hosszú. Ez a körút az 1 - 2 - 4 - 3 - 1.



Kövessük végig a legközelebbi város hozzáadása heurisztikát az 1. városból indulva. Az elsőhöz a második város a legközelebbi (lásd a távolságmátrix első sorát). A 2. városból (az első kivéve a keresésből) a 3. város van a legközelebb. A harmadikból már csak a 4.-be mehetünk. Ezután zárjuk a körutat, ami így 1 - 2 - 3 - 4 - 1. Az ehhez tartozó teljes megtett út $4 + 3 + 4 + \sqrt{73} \approx 19.544$. $\square$

13.1.1. A távolságvektor szerepe a heurisztikákban

A legközelebbi város beillesztése heurisztika $\mathcal{O}(n^2)$ műveletigényű, ha az ún. *távolságvektorokat* használjuk: az iteráció minden lépésében ez a vektor tartalmazza az eddigi részkörútbeli, és az azon kívüli városok távolságát.

Kiindulásként ez a vektor az 1. várostól való távolságokat tartalmazza. Ez később, a k . városnak a részkörútba való bevonása után úgy módosul, hogy a k . városnak a részkörúton kívüli városoktól mért távolságai és az adott városokra vonatkozó, a távolságvektorban meglevő értékek minimumát kell képezni.

A távolságvektor használatával minden iterációs lépésben n -nél kisebb számú összehasonlítást végzünk, míg enélkül a távolságmátrix n^2 -tel arányos számú elemét kellene megvizsgálni, és ez nyilvánvalóan lényegesen rontaná a heurisztika műveletigényét.

PÉLDA. Tekintsük ismét a korábban vizsgált utazó ügynök feladatot, amelynek költségmátrixa

M	1	5	5	5
5	M	1	5	5
1	5	M	5	5
5	5	5	M	1
5	5	5	1	M

A legközelebbi város beillesztése heurisztika első lépésében az 1 - 1 részkörúthoz a távolságvektor az (1, 5, 5, 5) elemeket tartalmazza (ez a mátrix első sora, a főátlóbeli elem kivételével).

A következő lépésben a részkörutat a 2. várossal bővítjük. A távolságvektorban a 2. városra vonatkozó elemet törölhetjük. A következő vektorelem $\min(5, 1) = 1$ lesz. A többi komponens nem változik: az új távolságvektor (-, 1, 5, 5).

Az eljárást tovább folytatva az utolsó két képzett távolságvektor rendre (-, -, 5, 5), illetve (-, -, -, 1) lesz.

□

13.1.2. A heurisztikus algoritmusok hatásossága vizsgálata

A heurisztikus algoritmusok eredményessége minősítésére három vizsgálati módszerosztály használatos:

- a legrosszabb esetek vizsgálata (worst case analysis),
- a valószínűségi analízis (probabilistic analysis), és
- az empirikus analízis (empirical analysis).

1. Az első megközelítés azt vizsgálja, hogy a legrosszabb lehetséges esetben milyen eltérést kaphatunk a heurisztika által adott közelítő megoldás és a valódi optimum között. Ezt fejezi ki lényegében a *heurisztika aszimptotikus hányadosa*. Legyen A egy a tekintett $\mathcal{C}$ problémaosztály feladatai megoldását megközelítő heurisztika, $A(P)$ a P feladaton a heurisztika által adott közelítés lehetséges megoldásának célfüggvény értéke, és $OPT(P)$ a P feladat optimuma. Ekkor amennyiben

$$M_A = \limsup \left\{ \frac{A(P)}{OPT(P)} \mid P \in \mathcal{C} \right\}$$

létezik, akkor az a heurisztika aszimptotikus hányadosa. Az aszimptotikus hányados értelmezéséhez tekintsük a következő egyenlőtlenséget:

$$A(P) \leq (M_A + \varepsilon) OPT(P).$$

Ez az összefüggés véges sok eset kivételével érvényes minden $\mathcal{C}$ -beli feladatra. Mivel minden feladatnak van egy bizonyos (véges) mérete, ezért elegendően nagy feladatméret esetén a fenti feltétel már mindig érvényes, tehát ekkor már minden ekkora feladatra tudjuk, hogy a kapott közelítő megoldás legfeljebb hányszor rosszabb eredményt ad.

Nagyon jó, 1 és 2 közé eső aszimptotikus hányadosok érvényesek egyes *ládapakolási, szabási feladatokra*. Másrészt 1976-ban S. Sahni és T. Gonzalez igazolták, hogy ha létezik aszimptotikus hányados valamely polinomkorlátos TSP heurisztikára, akkor $P = NP$.

Ez utóbbi megállapítás alapján arra lehetne következtetni, hogy akkor nem érdemes polinomkorlátos TSP heurisztikákat keresni. Mégis (vö. a lineáris programozás esetét a $3n$ átlagos, és 2^n legrosszabb

esetre vonatkozó iterációs számmal) *speciális* utazó ügynök feladatok esetén számíthatunk jó aszimptotikus hányadosra, és a gyakorlatban előforduló feladatok esetén is lehetnek egyes heurisztikák gyorsak.

2. A valószínűségi analízis (probabilistic analysis) módszere alkalmazása során azonos méretű feladatok paramétereit (együtthatóit) független valószínűségi változóknak tekintjük. Ezek eloszlására alkalmas feltételek teljesülését feltételezzük (gyakran azt, hogy az eloszlások egyenletesek).

Ebben az esetben $A(P)$, $OPT(P)$, és ezek hányadosa is valószínűségi változó. Az utóbbi várható értékéből az átlagos eltérésre lehet következtetni. Ezzel a módszerrel szemben az a gyakori kifogás, hogy az eloszlásokra vonatkozó feltételek nem feltétlenül helyesek, márpedig ezek a kapott eredményt döntően befolyásolhatják.

3. Az empirikus vizsgálat konkrét feladatmegoldásból kapott eredményekből képez mutatókat. Az eljárás lényege, hogy nagy számú feladatot generálunk úgy, hogy a feladat együtthatóit valamely adott (általában egyenletes) eloszlással egy rögzített számtartományból választjuk. Ezekre a problémákra mind a heurisztika által adott közelítést, mind a pontos megoldást meghatározzuk.

Ezután kiszámoljuk a hányadosokat és azok átlagát képezzük. Ha elegendően sok feladatot vizsgáltunk meg, akkor az eredmény, az empirikus hányados jellemezni fogja a vizsgált feladatosztályon a heurisztika közelítéseinek jószágát. Másrészt a számtartománynak a gyakorlati problémák halmazához való viszonya, és a feltételezett eloszlás általában kérdéses.

Összefoglalva az eddigieket, a heurisztikus algoritmusok közelítő megoldásai minőségének jellemzésére érdemes mindhárom tárgyalt módszert használni, hogy így kiegyensúlyozottabb képet kaphassunk.

Az elsőként ismertetett legközelebbi város beillesztése (nearest addition) heurisztika polinomkorlátos, a műveletigénye $\mathcal{O}(n^2)$. Nincs rá aszimptotikus hányados.

Ehhez a heurisztikához hasonló, de annál jobb megoldást ad a *legközelebbi város beszúrása* (nearest insertion) nevű eljárás. Ez is egy részkörúttal indul, majd az aktuális részkörutat olyan várossal bővíti, amelynek távolsága a részkörút egyik városához minimális.

Az eltérés abban van, hogy az új várost oda fogjuk beszúrni a részkörútba, ahol a beillesztés a legkisebb össztávolság növekedést okozza. Ez a heurisztika is $\mathcal{O}(n^2)$ műveletigényű. Másrészt olyan utazó ügynök feladatokra, amelyek költségmátrixa szimmetrikus, és érvényes rá a háromszög egyenlőtlenség, a legközelebbi város beszúrása heurisztika aszimptotikus hányadosa 2. Más szóval véges sok esettől eltekintve legfeljebb kétszer nagyobb lesz a heurisztikával kapott célfüggvény, mint az optimális.

Érdekes, hogy nem mohó algoritmus, de van értelme a *legtávolabbi város beszúrásának* (farthest insertion). Az eljárás beszúrása az előző heurisztika módszerével történik, itt is a lehető legkisebb költség-növekedést jelentő helyre történik a hozzáadás. Az empirikus vizsgálatok szerint ez a jobb, mint az előbbi három.

A *legolcsóbb beszúrás* (cheapest insertion) módszer az eddigiek bizonyos értelmű kiteljesítése: azt a várost választjuk a meglevő részkörút bővítésére, amelyik alkalmas helyre való beszúrása a lehető legkisebb költség-növekedést jelenti. Az eljárás műveletigénye $\mathcal{O}(n^3)$, és a korábban már említett szimmetrikus költségmátrixú, a háromszög-egyenlőtlenséget teljesítő költségmátrixú TSP feladatokra az eljárás aszimptotikus hányadosa 2.

Mivel az említett heurisztikák mind viszonylag gyorsak, és ezek eredménye függ az induló város megválasztásától, ezért szokásos a heurisztikus algoritmusokat többször is végrehajtani, eltérő városokból indulva. Ha minden városra lefuttattunk egy heurisztikát, akkor az összevont eljárást szokás *all cities* változatnak nevezni.

Az említetteken kívül használatosak heurisztikák a hozzárendelési feladat megoldásából kapott két vagy több részkörút összekapcsolására is.

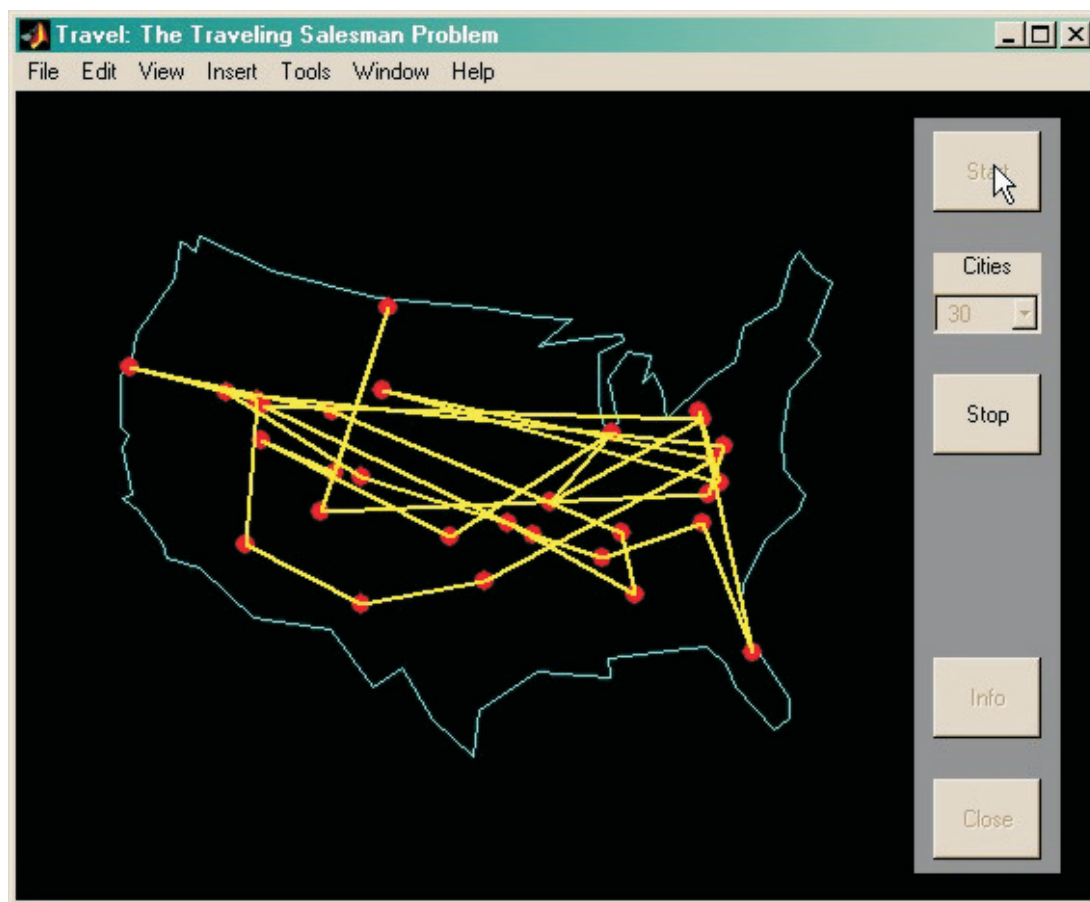
13.2. Az utazó ügynök feladat megoldása Matlabbal

Az ábrán az utazó ügynök feladat egy esetének kezdő helyzetét lehet látni. Az Amerikai Egyesült Államok véletlenül generált 30 városa van kiemelve, és a köztük keresett körút kezdeti változata. Figyeljük meg, hogy a jelen helyzet még nagyon távol van az optimális megoldástól, több helyen még az is kérdéses, hogy egyáltalán körút-e a megadott.

A Matlab Help menüsorában a Demos utasítás kiadása után kapott párbeszédés ablakban kérjük a Matlab bemutató programokat, azután a More demos fület válasszuk, majd a kapott listából a Travelling Salesman programot.

A jelen ábra a Corel Draw Capture programjával készült, és a kapott postscript ábra 30-szor nagyobb, mint a következő, amit a Matlab saját export utasítása adott.

A második ábra a feladat közel stabilizálódott közelítő megoldását mutatja. Az ehhez szükséges számítási idő pár másodperc volt.



Az alábbi programrészlet a Matlab utazó ügynök feladatra írt bemutató programjából való. Az eljárás célja nem a gyors megoldás, hanem a feladat nehézségének, és a megoldás menetének illusztrálása volt. A teljes Matlab program kb. 300 soros.

Figyeljük meg a viszonylag könnyen olvasható algoritmust, amely az eddigi közelítő megoldáson azt kísérli meg, hogy egy véletlenül generált körút szakaszon a bejárás sorrendjét az ellenkezőjére változtatja. Amennyiben a beavatkozás sikeres volt, akkor a javított útvonal lesz a továbbiakban a jelölt a megoldásra.

```
% Try a point for point swap
% =====
swpt1=floor(npts*rand)+1;
```

```

swpt2=floor(npts*rand)+1;

swptlo=min(swpt1,swpt2);
swpthi=max(swpt1,swpt2);

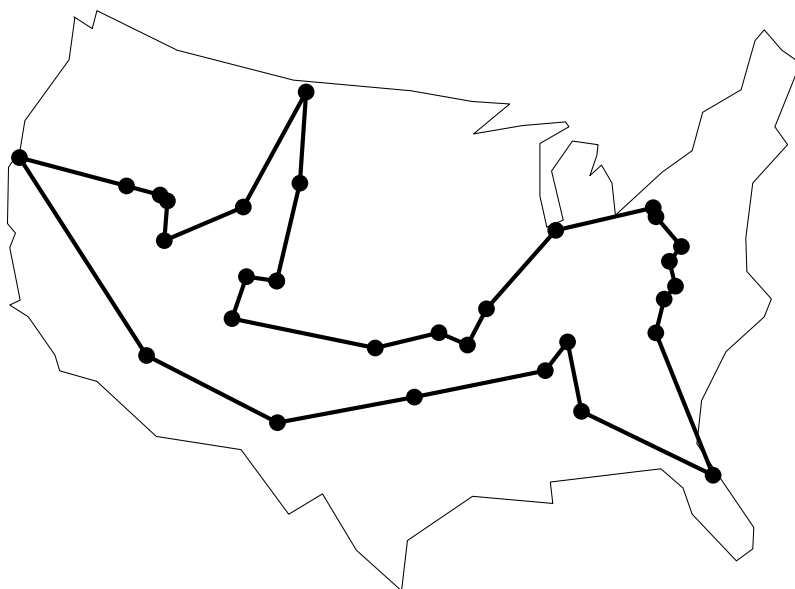
order=1:npts;
order(swptlo:swpthi)=order(swpthi:-1:swptlo);
pnew = p(order);

lennew=LocalPathLength(pnew,distmatrix);
if lennew<len,
    p=pnew;
    len=lennew;
    drawFlag=1;
end;
% =====

```

Érdekes és hatékony a tömbök címzési módja, pl.

```
order(swpthi:-1:swptlo).
```



Start

Cities
30

Stop

Info

Close

13.3. Ellenőrző kérdések és gyakorló feladatok

1. Mutasson olyan feladatot a hétköznapi problémái közül, amely átalakítással megfeleltethető az utazó ügynök feladatnak, vagy tartalmazza azt!

2. Adjon meg olyan egyszerű hozzárendelési feladatot, amelynek optimális megoldása egyben az azonos költségmátrixú utazóügynök feladat megoldása is.
3. Adjon meg olyan egyszerű hozzárendelési feladatot, amelynek optimális megoldása nem megoldása az azonos költségmátrixú utazóügynök feladatnak.

14. fejezet

A szabási feladat

Adott méretű rudak, egyéb munkadarabok leszállása során gyakran felmerül az a feladat, hogy adott hosszúságú nyersanyagból hogyan vágjuk le az előírt összetételű végtermék halmazát úgy, hogy a lehető legkevesebb veszteség maradjon. Ezt a problémát egydimenziós szabási, vagy ládapakolási feladatnak (bin packing) nevezik.

A gyakorlati feladatok közül ide tartozik az, amikor egy építőipari vállalat adott profilú, rögzített hosszúságú fémrudakból a könnyűszerkezetes építkezéshez nagyszámú rövidebb rudat, oszlopot akar levágni, de figyelembe kell venni, hogy a lehető legkevesebb teljes rudat kezdjünk meg, vagy az a cél, hogy az eldobandó nyersanyag mennyisége legyen minimális.

Ilyen feladatra vezet az is, ha egy síküveggyárban adott szélességű, hosszú üvegtáblákból kell megadott összetételű (az eredeti tábla szélességét megtartó) üveglapokat kivágni.

A hátizsák feladathoz hasonló megfogalmazást is lehet adni: adottak különböző súlyú tárgyak, ezek mindegyikét el kell helyezni minimális számú hátizsákban úgy, hogy azok közös súlykorlátját ne lépjük túl.

A számítástechnikából azt a problémát idézhetjük, amikor rögzített méretű tárhelyekre (pl. partíciókra) kell különböző méretű adatállományokat úgy elhelyezni, hogy ehhez a minimális számú tárhelyet használjuk fel.

A logisztikában az a feladat, hogy adott teherbírású járművekből mennyit kell minimálisan alkalmazni ahhoz, hogy adott, különböző súlyú tárgyakból álló rakomány elszállítható legyen,...

Többdimenziós szabási feladatot kapunk, ha az elhelyezendő tárgyak több kiterjedését is korlátozzuk, pl. a hosszát és a szélességét. Felmerülhet, hogy ebben az esetben mindenféle vágást megengedünk-e, vagy csak az anyag szélétől széléig menő, ún. guillotine-vágásokat.

Tekintsük először az egydimenziós szabási feladat L.V. Kantorovicstól származó modelljét. Legyenek adva korlátlan mennyiségben K hosszúságú félkész termékek, ahol K pozitív egész. Az egyes végtermékek hossza legyen $k_1, k_2, \dots, k_n$. Ezek egyike sem nagyobb K -nál. A végtermékekből rendre $r_1, r_2, \dots, r_n$ darabot kell levágni.

A modellben az a_j vektort lehetséges szabásnak nevezzük, ha

$$a_{ij} \geq 0, \quad (i = 1, \dots, n),$$

$$\sum_{t=1}^n k_t a_{t,j} \leq K.$$

Ezek után jelölje $A = (a_1, \dots, a_p)$ az összes lehetséges szabásokból előállított mátrixot. Legyen r az $r_1, \dots, r_n$ komponensekből álló oszlopvektor, és c az a sorvektor, amelynek minden eleme egyenlő, egy félkész termék költsége.

Az egydimenziós szabási feladat optimumszámítási modellje ezekkel a paraméterekkel:

$$\min z(x) = cx,$$

feltéve, hogy

$$Ax = r, \\ x \geq 0, \quad x \text{ egész}, \quad (r \geq 0).$$

A modell használatának nehézségét az mutatja, hogy már a lehetséges szabások A mátrixának az összeállítása is komoly problémát jelent.

A célfüggvény konstans együtthatóit az magyarázza, hogy így a célfüggvény a lehetséges szabásokból a minimális darabszámúhoz tartozó megoldást fogja kiválasztani.

Vegyük észre, hogy a megadott modell egy egészértékű lineáris programozási feladat, amely a felírt formában a standard feladatnak felel meg.

PÉLDA. Tekintsünk egy nagyon egyszerű szabási feladatot: a félkész termékek hossza legyen 1, az egyes végtermékeké pedig 0.25 és 0.5. Írjuk elő, hogy összesen 4 darabot kell levágni az első termékből és kettőt a másodikból.

Ahogy könnyen ellenőrizhető, a lehetséges szabások ezek alapján:

$$(0, 0)^T, (0, 1)^T, (0, 2)^T, (1, 0)^T, (1, 1)^T, (2, 0)^T, (2, 1)^T, (3, 0)^T \text{ és } (4, 0)^T.$$

Ezek segítségével felírhatjuk az optimalizálási feladatot:

$$\min z(x) = cx = \sum_{i=1}^9 x_i,$$

feltéve, hogy

$$Ax = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 2 & 2 & 3 & 4 \\ 0 & 1 & 2 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} x = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$$

és

$$x \geq 0, \quad x \text{ egész}, \quad (r \geq 0).$$

A feladat spekulatív megoldása során vegyük észre, hogy x_1 értéke nulla kell hogy legyen az optimális megoldásban, hiszen ez a teljesítendő termékszámhoz nem járul hozzá, de növeli a költséget.

A másik végletet x_9 képviseli, mert egy ilyen felosztású félkész termék felhasználása adja a legtöbb teljesített készterméket. Ilyen szabásokból viszont nem áll össze a teljes szabási előírás.

Másrészt megállapíthatjuk, hogy a jobboldali r vektor arányait pontosan adja az x_7 változóhoz tartozó szabás. Ebből két darab kell ahhoz, hogy teljesüljön az egyenlőség feltételünk. Ehhez a 2 célfüggvényérték tartozik.

Látható, hogy ennél jobbat nem lehet elérni, mert nincs olyan lehetséges szabás, amelyből egy adná a jobboldali r vektort. Van viszont még egy optimális megoldás, az $x_3 = 1, x_9 = 1$ (és minden további $x_i = 0$): ez is kettes célfüggvény értéket ad – és ezzel ki is merítettük az optimális megoldások halmazát.

□

14.1. Az oszlopgenerálás módszere a szabási feladatra

A szabási feladat ismertett modellje nehezen használható, elsősorban a lehetséges szabások nagy száma miatt. P.C. Gilmore és R.E. Gomory egyszerűsítették a modellt. Első lépésben elhagyták az egészértékűségi feltételt. Az ezt támogató érvelés szerint az ipari alkalmazásokban a nem egész optimális megoldás egész értékre való valamilyen átalakítása elfogadható. Az oszlopgenerálás módszere

ezután a módosított szimplex algoritmust használja, így nem szükséges a teljes A mátrix ismerete, elegendő annak az oszlopnak az előállítás, amelyben generáló elemet keresünk.

Ahhoz, hogy a módosított szimplex algoritmust használni lehessen, az eddigi

$$\min z(x) = cx,$$

feltéve, hogy

$$Ax = r,$$

$$x \geq 0, \quad (r \geq 0).$$

standard alakú feladatunkat lehetséges kanonikus alakra kell hozni. A bázisváltások legyenek azokhoz a lehetséges szabásokhoz tartozók, amelyek épp az egységmátrixot adják: $a'_1 = (1, 0, \dots, 0)^T, \dots, a'_n = (0, \dots, 0, 1)^T$. Tekintsük ezeket az A mátrix első n oszlopának. Ezután már csak annyit kell tenni, hogy minden sornak a $-c$ -szeresét hozzáadjuk a célfüggvényhez. Ez a művelet épp a bázisváltásokhoz tartozó célfüggvény-együtthatókat fogja nullázni.

Legyen d egy olyan n dimenziós vektor, amelynek minden komponense c . Ezzel a feladatunk a következő lehetséges kanonikus alakban írható:

$$\min z(x) = dr + (c - dA)x,$$

feltéve, hogy

$$Ax = r,$$

$$x \geq 0, \quad (r \geq 0).$$

A következő feladat a legkisebb célfüggvény-együttható meghatározása. Legyen az A mátrix egy tetszőleges oszlopvektora $(v_1, \dots, v_n)^T$. Ez alapján a $c - \sum_{t=1}^n d_t v_t$ összeg minimumát keressük, ahol a d vektor n -dimenziós, és minden komponense c . Ezt a szélsőértéket ott kapjuk, ahol a $\sum_{t=1}^n d_t v_t$ összeg maximális. Mivel v egy lehetséges szabás, ezért $\sum_{t=1}^n k_t v_t \leq K$.

A legkisebb célfüggvény-együttható meghatározásához eszerint a következő speciális egészértékű lineáris programozási feladatot kell megoldani:

$$\max w(v) = \sum_{t=1}^n d_t v_t,$$

feltéve, hogy

$$\sum_{t=1}^n k_t v_t \leq K,$$

$$v_t \geq 0, \text{ egész, } t = 1, \dots, n.$$

Ez a feladat a korábban megismert hátizsák feladat a K súlykorláttal, k_t súlyokkal és d_t értékekkel. Ennek megoldásával tudjuk meghatározni a szabási feladat generáló eleme oszlopát.

A szabási feladat legkisebb célfüggvény-együtthatója és a generálóelem ismeretében végre tudjuk hajtani a módosított szimplex algoritmus egy lépését. Ezután az eddigi lépéseket ismételjük. Az egész eljárás akkor fejeződik be, ha az aktuális hátizsák feladat optimuma nem nagyobb, mint c . Ekkor az eredeti feladat minden célfüggvény együtthatója nemnegatív.

PÉLDA. Tekintsük ismét az előző nagyon egyszerű szabási feladatot: a félkész termékek hossza 1, az egyes végtermékeké pedig 0.25 és 0.5. Összesen 4 darabot kell levágni az első termékből és kettőt a másodikból. Az optimalizálási feladat átrendezve úgy, hogy az egységmátrix legyen a baloldalon ($x_1 - x_4$ csere):

$$\min z(x) = cx = \sum_{i=1}^7 x_i,$$

feltéve, hogy

$$Ax = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 2 & 2 & 3 & 4 \\ 0 & 1 & 2 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} x = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$$

$$x \geq 0, \quad (r \geq 0).$$

Az ehhez tartozó táblázat $c = 1$ értékkel

x_4	x_2	x_3	x_1	x_5	x_6	x_7	x_8	x_9	
1	0	0	0	1	2	2	3	4	4
0	1	2	0	1	0	1	0	0	2
1	1	1	1	1	1	1	1	1	0

A szimplex táblázat az új, $c - \sum_{t=1}^n d_t v_t$ célfüggvénnyel:

	x_3	x_1	x_5	x_6	x_7	x_8	x_9	
x_4	0	0	1	2	2	3	4	4
x_2	2	0	1	0	1	0	0	2
	-1	1	-1	-1	-2	-2	-3	-6

A leolvasható bázismegoldás azt jelenti, hogy az $(1, 0)$ szabásból kellene 4 darab, és a $(0, 1)$ szabásból 2 darab. Ez összesen 6 félkész terméket jelent, ami nyilván még javítható. Oldjuk meg először a feladatot a szimplex algoritmussal.

Itt a második oszlop alapján nem lehetne javítani a célfüggvény értékét (de nem is találunk benne generáló elemet). A legkisebb negatív célfüggvény együttható az utolsó oszlopot jelöli ki, és ebben az első mátrixelem (a négyes) lesz a generálóelem. A transzformált, következő szimplex táblázat:

	x_3	x_1	x_5	x_6	x_7	x_8	x_4	
x_9	0	0	1/4	1/2	1/2	3/4	1/4	1
x_2	2	0	1	0	1	0	0	2
	-1	1	-1/4	1/2	-1/2	1/4	3/4	-3

Az előző szimplex táblázat:

	x_3	x_1	x_5	x_6	x_7	x_8	x_4	
x_9	0	0	1/4	1/2	1/2	3/4	1/4	1
x_2	2	0	1	0	1	0	0	2
	-1	1	-1/4	1/2	-1/2	1/4	3/4	-3

A legkisebb célfüggvény-együttható az x_3 változónak a bázisba lépését jelenti.

	x_2	x_1	x_5	x_6	x_7	x_8	x_4	
x_9	0	0	1/4	1/2	1/2	3/4	1/4	1
x_3	1/2	0	1/2	0	1/2	0	0	1
	1/2	1	1/4	1/2	0	1/4	3/4	-2

Erről a szimplex táblázatról már le lehet olvasni a végeredményt: a 3. és a 9. szabásból kell egyet-egyvet venni. Ez lehetséges megoldás lesz, és az ezzel adódó optimális célfüggvény érték 2. Az x_7 -es szabásból kettő is optimális, és ez összhangban is van az eredményünkkel.

Tekintsük most a feladatunkat a Gilmore-Gomory-féle oszlopgenerálásnak megfelelően. A módosított szimplex algoritmusról tanultak miatt az ugyanezen bázismegoldásokon keresztül jut azonos eredményhez. Az eltérés a módosított szimplex algoritmus kivitelezésében, és főleg a legkisebb célfüggvény-együttható előállításában van. Ennek illusztrálásához lépünk vissza a

x_4	x_2	x_3	x_1	x_5	x_6	x_7	x_8	x_9	
1	0	0	0	1	2	2	3	4	4
0	1	2	0	1	0	1	0	0	2
1	1	1	1	1	1	1	1	1	0

feladathoz. Írjuk fel erre az új bázisváltozó meghatározásához a megfelelő hátizsák feladatot:

$$\max w(v) = \sum_{t=1}^n d_t v_t = \sum_{t=1}^n c v_t = \sum_{t=1}^n v_t,$$

feltéve, hogy

$$\sum_{t=1}^n k_t v_t \leq K, \quad v_t \geq 0, \quad \text{egész, } t = 1, \dots, n.$$

Ez a feladat lényegében azt a lehetséges szabást keresi, amely a végtermékekből a legtöbbet állítja elő.

□

14.2. Heurisztikák a szabási feladatra

Az ismertett egzakt módszerek mind nehezen végrehajthatók nagyobb méretű gyakorlati feladatokra. Az alábbi heurisztikák ezzel szemben gyorsan adnak jó közelítő megoldást.

A *first fit módszer* a végtermékeket felsorolási sorban tekinti, az aktuális munkadarabot az első olyan félkész termékre helyezi el, amelyen az elfér. Ebben az értelemben ez egy mohó algoritmus.

A *best fit algoritmus* olyan rudat ad meg, amelyről az aktuális munkadarabot levágva a legkevesebb maradék képződik.

A *worst fit eljárás* pedig értelemszerűen olyan megkezdett rudat választ az aktuális munkadarab levágásához, amelyiken a vágás után a leghosszabb még felhasználható szakasz marad.

Előnyös a leszabandó végtermékeket előzőleg nagyság szerint rendezni. Ez lényegesen javítja a heurisztika eredményét.

Az említetteken felül további egyszerű heurisztikák léteznek. Érdemes megemlíteni, hogy az eddig tárgyalt szabási problémát *offline szabási feladatnak* is nevezhetjük, hiszen az összes adat ismeretében kellett megoldanunk, és az összes félkész terméket az eljárás teljes tartalma alatt felhasználhattuk a megoldás javításához.

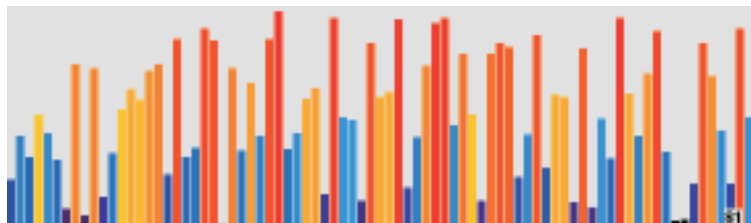
Ezzel szemben online-nak nevezzük a szabási feladatot, ha a leszabandó termékek adatai beérkeztekor véglegesen döntenünk kell elhelyezésükről (a többi adat ismerete nélkül), vagy ha egy időben csak adott rögzített számú félkész termékből lehet szabni. Utóbbi esetben ha különben nem tudnánk tovább haladni, akkor valamelyik félkész termék eddigi szabását véglegesnek kell nyilvánítani, és egy újat vonunk be a szabás meghatározásába.

A szabási feladat heurisztikáinak tömör, látványos illusztrációja található a

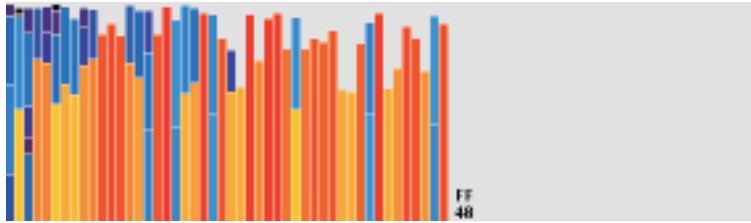
<http://www.cs.arizona.edu/icon/oddsends/bpack/bpack.htm>

címen.

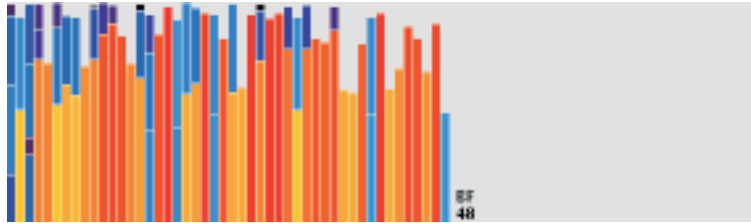
A heurisztikus algoritmusok működését illusztrálja az alábbi öt ábra egy véletlenül generált feladatra:



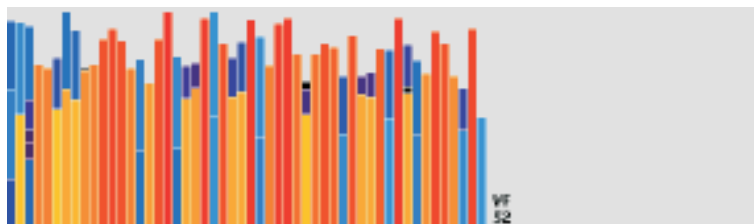
A first fit eredménye:



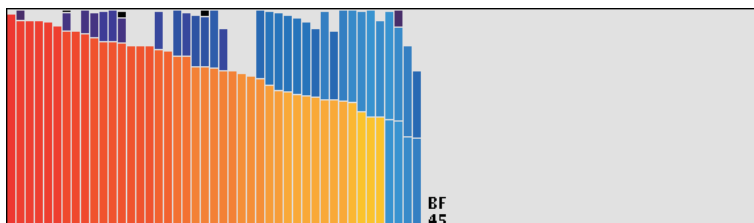
A best fit eredménye:



A worst fit eredménye:



És végül a rendezés utáni best fit adta pakolás:



14.3. Ellenőrző kérdések és gyakorló feladatok

1. Mutasson olyan egydimenziós szabási feladatot, amelyre minden tanult szabási elhelyezési heurisztika optimális!
2. Mutasson olyan egydimenziós szabási feladatot, amelyre egyik ismertett szabási heurisztika sem optimális!
3. Mit lehet mondani a heurisztikák optimalitásáról olyan feladatokra vonatkozóan, amelyekben minden tárgynak azonos a mérete?

15. fejezet

Hálózati problémák

A hálózati problémákban egy *irányított gráffal* megadott lehetőségek közül az ezekhez köthető optimális folyamatokat akarjuk meghatározni. A hálózati feladatokkal kapcsolatban mindig feltesszük, hogy a hálózat minden élének van hossza. Egy *gráfot*, vagy *hálózatot* két halmazzal adhatunk meg. Az első a *csúcsokat*, a másik az ezekből álló párokat, az *irányított éleket* határozza meg. Egy adott hálózatban megjelölhetünk *kezdőpontot* és *végpontot* is.

15.1. A legrövidebb út probléma

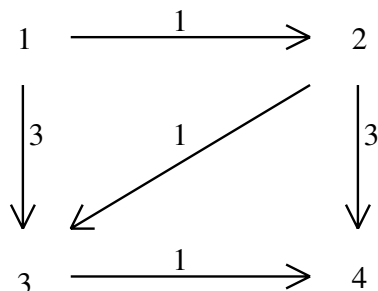
Láncnak nevezzük éleknek egy olyan sorozatát, amelyben az egymást követő bármely két élnek egy közös csúcsa van.

Az *út* egy olyan lánc, amelyben az utolsó él kivételével mindegyik él végpontja a sorozatban következő él kezdőpontja. Az (1,2), (2,3) és (4,3) élek láncot adnak, de az nem út. Út és lánc viszont az (1,2), (2,3) és (3,4) élek sorozata.

A legrövidebb út problémája egy hálózatban egy adott csúcsból kiindulva a többi csúcsba vezető legrövidebb út meghatározását jelenti. Ennek megoldására alkalmas *Dijkstra algoritmus*a amennyiben minden él hossza nemnegatív:

- Lássuk el az első csúcsot az állandó 0 címkével.
- Minden olyan i csúcsot lássunk el ideiglenesen az $(1, i)$ él hosszával mint címkével, amelyhez vezet él az 1 csúcsból. Minden más csúcs (az első kivételével) kapja ideiglenesen a ∞ címkét. A legkisebb ideiglenes címkéhez tartozó egyik csúcs címkéjét állandónak minősítjük.
- Tegyük fel, hogy az i volt az utolsó, a $(k + 1)$. csúcs, amely állandó címkét kapott. Akkor i a k -adik legközelebbi csúcs az elsőhöz. Az ideiglenes címkével rendelkező j csúcsok címkéit módosítsuk az $(i$ címkéje + az (i, j) távolsága) értékre, ha ez kisebb, mint j eddigi ideiglenes címkéje. Ezután ismét adjunk végleges címkét egy olyan csúcsnak, amelynek címkéje a maradék ideiglenes címkék legkisebbike.
- Folytassuk az eljárást amíg minden csúcs állandó címkét nem kap.
- Ha minden csúcsnak végleges címkéje van, akkor az 1. csúcsból egy j csúcsba vezető legrövidebb utat úgy kapjuk, hogy a j csúcsból visszafelé haladva olyan csúcsokon keresztül jutunk el az 1. csúcsba, amelyektől a rákövetkezőbe vezető él hossza épp a két címke különbsége.

PÉLDA. Tekintsük azt az egyszerű legrövidebb út feladatot, amelyben négy városunk van: 1, 2, 3 és 4, és a köztük levő távolságot a következő ábra adja:



Ez kb. egy olyan elhelyezésnek felel meg, amikor a városok egy álló téglalap csúcsaiban vannak, de az 1 - 4 átló nem járható.

A Dijkstra algoritmus első lépése az 1. városból indulva a következő címkézést adja:

$$[0^*, \infty, \infty, \infty],$$

ahol a $*$ azt jelzi, hogy az illető címke végleges.

A következő lépésben meghatározzuk az első várostól mért távolságok alapján annak szomszédainak ideiglenes címkéjét:

$$[0^*, 1, 3, \infty].$$

Az új, ideiglenes címkék közül véglegesítjük a legkisebbet:

$$[0^*, 1^*, 3, \infty].$$

Képezzük most a végleges címkéjű városoktól mért távolságok alapján az új, javított címkéket:

$$[0^*, 1^*, 2, 4].$$

Itt a harmadik címke kettes értéke úgy adódott, hogy a kettes város végleges címkéje + a második és a harmadik város távolsága kisebb mint a korábbi ideiglenes címke értéke, a három. Kössük le ismét az ideiglenes címkék közül a legkisebbet:

$$[0^*, 1^*, 2^*, 4].$$

Az utolsó ideiglenes címkét ismét lehet javítani, és az ezután már végleges is lesz:

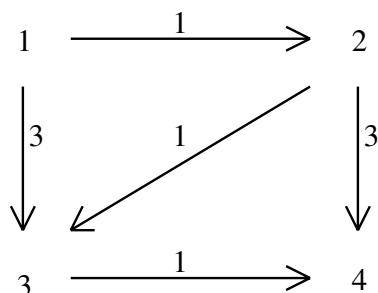
$$[0^*, 1^*, 2^*, 3^*].$$

Ebből az első és negyedik város közti legrövidebb út 3 hosszú: 1 - 2 - 3 - 4.

15.2. A maximális folyam probléma

Egyes döntési helyzetekben olyan hálózatot kell vizsgálni, amelyben az éleknek kapacitások feleltethetők meg. A kérdés az, hogy az egyik kitüntetett csúcsból, a forrásból egy másikba, a nyelőbe mi a maximális eljuttatható mennyiség a hálózat és a kapacitások figyelembevételével. Ezt a feladatot nevezzük a *maximális folyam problémának*.

Tekintsük a korábbi hálózatot úgy, hogy az élekre írt számok a kapacitásokat jelentik, az egyes csúcs a *forrás* és a négyes a *nyelő*:



A későbbi eljárás kedvéért vezessünk be egy *mesterséges élet* a nyelőtől a forrásig. A feladat lineáris programozási feladatként való megfogalmazásához jelölje az x_{ij} változó az (i, j) élen áthaladó anyagmennyiséget.

Egy lehetséges folyamatot kapunk például a következő változó értékekkel:

$$x_{12} = 1, \quad x_{13} = 1, \quad x_{23} = 0, \quad x_{24} = 1, \quad \text{és} \quad x_{34} = 1.$$

Az ehhez tartozó teljes átfolyó mennyiség 2. A lehetséges folyamatoknak eleget kell tenniük a következő két feltételnek:

1. minden éltre az élen átmenő folyam nemnegatív és nem nagyobb, mint a megadott élkapacitás, és
2. minden csúcsra igaz az, hogy a bejövő folyam mennyisége megegyezik a kimenő folyamával.

Az utóbbi feltételt folyammegőrzési feltételnek nevezzük. A probléma lineáris programozási feladatként való megfogalmazásában a fenti feltételek mellett az x_0 célfüggvényt kell maximalizálni, ahol x_0 a nyelőből a forrásba vezető mesterséges élen átmenő folyam mértéke.

Az említett példára vonatkozó lineáris programozási feladat ez alapján:

$$\max x_0 = x_{24} + x_{34},$$

feltéve, hogy

$$\begin{aligned} x_{12} &\leq 1, \\ x_{23} &\leq 1, \\ x_{13} &\leq 3, \\ x_{24} &\leq 3, \\ x_{34} &\leq 1, \\ x_0 &= x_{24} + x_{34}, \\ x_{13} + x_{23} &= x_{34}, \\ x_{12} &= x_{23} + x_{24}, \\ x_0 &= x_{12} + x_{13}. \end{aligned}$$

Ez egy hatváltozós lineáris programozási feladat 4 egyenlőség és 5 egyenlőtlenség feltétellel. Az ábráról könnyen leolvasható, hogy a korábban említett

$$x_{12} = 1, \quad x_{13} = 1, \quad x_{23} = 0, \quad x_{24} = 1, \quad \text{és} \quad x_{34} = 1$$

lehetséges megoldás egyben optimális is. Ez azon múlik, hogy a 4. csúcsba 2-nél nagyobb folyam nem folyhat be (figyelembe véve a 2. csúcsba bemenő folyamot).

Állapítsuk meg, hogy lehetséges megoldást könnyű megadni a maximális folyam problémához: ha minden élen nulla mennyiséget szállítunk, az megfelel a feltételeknek.

Jelöljük I -vel azon élek halmazát, amelyeken kisebb a jelenleg átmenő folyam az él kapacitásánál. Jelölje R azon élek halmazát, amelyeken a jelenlegi folyam pozitív, ez csökkenthető. Legyenek $i(x, y)$, illetve $r(x, y)$ a megfelelő változtatási korlátok.

15.3. A Ford-Fulkerson eljárás a maximális folyam meghatározására

Tekintsük először a címkézési eljárást:

1. lépés. Címkézzük meg a forrást.

2. lépés. Címkezzük meg a csúcsokat és az éleket a forrásba vezető mesterséges él kivételével a következő szabályok szerint.

Ha az x csúcs már kapott címkét, de az y még nem, és $(x, y) \in I$, akkor címkezzük meg az y csúcsot és az (x, y) élt. Ekkor az (x, y) élt *előremenő élnek* hívjuk.

Ha az x csúcs már kapott címkét, de az y csúcs még nem, és $(y, x) \in R$, akkor címkezzük meg az y csúcsot és az (y, x) élt. Az utóbbit *hátramenő élnek* nevezzük.

3. lépés. Folytassuk a címkézési eljárást, amíg a nyelő címkét nem kap, vagy már nem lehet további csúcsokat címkével ellátni.

Ha a címkézéssel elérjük a nyelőt, akkor lesz a forrás és a nyelő között egy címkézett élekből álló lánc. Jelölje ezt C . A C -beli éleken átmenő folyam alkalmas módosításával egyrészt megőrizhetjük a folyam lehetségségét, másrészt növelhetjük a folyamat.

A Ford-Fulkerson módszer a fentiek alapján:

1. lépés. Keressünk egy lehetséges folyamat (a minden élen 0 értékű folyam mindig lehetséges).

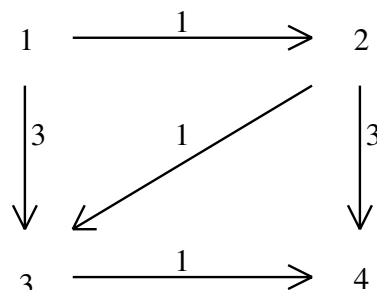
2. lépés. A címkézési eljárással kíséreljük meg elérni a nyelőt. Ha nem lehet a nyelőt így megcímkézni, akkor az aktuális lehetséges folyam maximális. Ha elértük a nyelőt, akkor folytassuk az eljárást a 3. lépésnél.

3. lépés. Határozzunk meg egy magasabb értékű lehetséges folyamat úgy, hogy a megcímkézett láncon az előremutató élek értékeit növeljük, a hátramutatókét pedig csökkentjük a

$$k = \min \left\{ \min_{(x,y) \in C \cap R} r(x, y), \min_{(x,y) \in C \cap I} i(x, y) \right\}$$

értékkel. Folytassuk az algoritmust a 2. lépéssel.

PÉLDA. Tekintsük ismét a



maximális folyam feladatát.

Az első lépés egy lehetséges folyam meghatározása. Legyen ez az, amelyik minden élhez a nulla folyamat rendeli hozzá.

Ezután kezdünk egy címkézési eljárást. Először a forrás, az 1. csúcs kap címkét, majd ez alapján egy olyan él, ami ebből kivezet. Most ez esetben címkét kap a 2. csúcs, és az $(1, 2)$ él. Ezt folytatva a nyelőig tartó címkézett láncot (most utat) kapunk:

$$1 - 2 - 3 - 4.$$

Az előző lánc csak előremenő éleket tartalmaz. A lehetséges folyambővítési lehetőségek rendre:

$$i(1, 2) = 1, \quad i(2, 3) = 1 \quad \text{és} \quad i(3, 4) = 1.$$

Ez alapján az eddigi folyam a címkézett láncunk mentén 1-el növelhető. Ez alapján az ehhez tartozó lehetséges folyam értéke 1.

A lehetséges folyam javítása során azokat az éleket kell vizsgálni a címkézés során, amelyek kapacitását még nem merítettük ki. Ezek alapján már csak egy címkézhető láncot lehet képezni, amely eléri a nyelőt, ez az

$$1 - 3 - 2 - 4.$$

Ezen belül a $3 - 2$ él hátramutató, ennek értékét legfeljebb 1-el lehet csökkenteni. Az eddigi lehetséges folyam ismét 1-el növelhető. Ezután a lehetséges folyam:

$$x_{12} = 1, \quad x_{13} = 1, \quad x_{24} = 1, \quad \text{és} \quad x_{34} = 1.$$

Ennek a lehetséges folyamnak az értéke 2.

Ezután látható, hogy további címkézett láncot már nem lehet képezni, tehát az előző lehetséges folyam egyben maximális is. $\square$

Legyen V' egy hálózat csúcsainak tetszőleges olyan halmaza, amely tartalmazza a nyelőt, de nem tartalmazza a forrást. Ekkor a hálózat olyan (i, j) éleinek halmazát, amelyek i kezdőcsúcsa nem V' -beli, a j végpont viszont V' -beli, a hálózat egy vágásának nevezzük. A vágás tehát élek egy olyan halmaza, amelyeket ha elhagyunk a hálózathoz, akkor a forrásból a nyelő a továbbiakban már nem lesz elérhető.

A vágás kapacitása a vágást alkotó élekből a forrástól a nyelő felé vezető kapacitásainak összege. A következő két segédétel adja meg a vágások és a maximális folyamok közti összefüggést.

SEGÉDTÉTEL. A forrásból a nyelőbe vezető folyamok erősségét bármelyik vágás kapacitása felülről korlátozza.

BIZONYÍTÁS. Tekintsük egy hálózat valamely V' vágását. A hálózat többi csúcsának halmaza legyen V . Válasszunk egy tetszőleges folyamot, legyen ennek értéke f , az (i, j) élen áthaladó mennyiséget pedig x_{ij} . Összegezzük a V -beli összes csúcsra a folyammegőrzési feltételeket. Ez azt adja, hogy mivel kiesnek az olyan élekre vonatkozó tagok, amelyek mindkét végpontja V -ben van, ezért

$$\sum_{i \in V, j \in V'} x_{ij} - \sum_{i \in V', j \in V} x_{ij} = f.$$

Az ebben az egyenletben szereplő első összeg egyenlő a V' -nek megfelelő vágás kapacitásával. Mivel minden x_{ij} érték nemnegatív, így a második összeg is nemnegatív. Ebből adódik a segédétel állítása: a folyamok értéke legfeljebb annyi lehet mint a vágásoké. $\square$

A segédteletből az is következik, hogy bármely vágás értéke felső korlátja a forrásból a nyelőbe áramló maximális folyam értékének. Tehát ha találunk egy olyan lehetséges folyamot, amelynek értéke egyenlő egy vágás kapacitásával, akkor találtunk egy maximális folyamot. Ez egy fajta gyenge dualitást fejez ki.

Tekintsük most azt az esetet, amikor egy adott lehetséges folyamra vonatkozóan a címkézési eljárás nem tudja elérni a nyelőt. Legyen ekkor a címkézetlen csúcsok halmaza által meghatározott a vizsgált vágás.

SEGÉDTÉTEL. Ha a címkézési eljárás nem tudja elérni a nyelőt, akkor a kimaradó, nem címkézett csúcsok által meghatározott vágás kapacitása megegyezik az aktuális folyam erősségével.

BIZONYÍTÁS. Legyen a korábbi jelöléseknek megfelelően V az aktuális folyam mellett megcímkézett csúcsok halmaza, és V' pedig a címkézetleneké. Tekintsünk egy (i, j) élet a vágásból, úgy, hogy $i \in V$

és $j \in V'$. Ekkor az x_{ij} értéknek egyenlőnek kell lennie az (i, j) él kapacitásával, hiszen különben tovább tudtunk volna haladni egy megfelelő előremenő éllel, és akkor j nem a V' -be tartozna.

Tekintsünk ezután egy olyan (i, j) élt, amelyre $i \in V'$ és $j \in V$. Ekkor viszont $x_{ij} = 0$ kell hogy teljesüljön, hiszen különben a címkézési eljárásban egy hátramenő éllel el tudtuk volna érni az i csúcsot, és így az nem tartozhatna a V' halmazba.

Az aktuális lehetséges folyamra tehát az teljesül az előző segédételben igazolt

$$\sum_{i \in V, j \in V'} x_{ij} - \sum_{i \in V', j \in V} x_{ij} = f$$

egyenlet alapján, hogy a jelen vágás kapacitása egyenlő a $\sum_{i \in V, j \in V'} x_{ij}$ összeggel, és az adott folyam erősségével. $\square$

Az eddigi megállapítások szerint tehát amikor a nyelőt nem lehet megcímkézni a forrásból indulva, akkor az aktuális lehetséges folyam egy maximális folyam. Ez összhangban van a Ford-Fulkerson módszerrel.

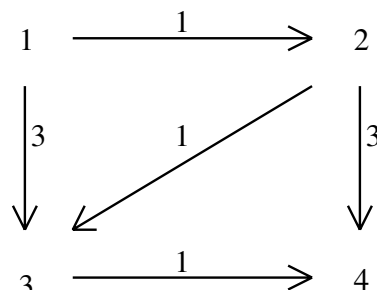
15.4. Projektek ütemezése, CPM

Az összetett munkafolyamatok rögzített befejezési időponttal való teljesítése gondos tervezőmunkát igényel. Ennek része az összefüggő események sorrendjének, időzítésének vizsgálata hálózati modellek segítségével.

Erre a célra két eljárást szokás használni. Ha az egyes munkafolyamatok végrehajtási ideje biztosan tudható, akkor a *kritikus út módszer* (Critical Path Method, CPM), míg ha a tevékenységek időtartama bizonytalan, akkor a *program kiértékelési és felülvizsgálati technika* (Program Evaluation and Review Technique, PERT) használatos. Mindkét eljárást az ötvenes években fejlesztették ki. Számos nagy és kritikus projekt tervezésekor használták ezeket a módszereket, pl. nagy szoftver rendszerek határidős kidolgozásánál, úrkutatási projektekben, vagy épp rakétaindítások visszaszámlálási eljárásának kidolgozásában.

Mindkét eljáráshoz szükség van a projektet alkotó tevékenységek listájára. A projektet akkor tekintjük befejezettnek, ha minden részfeladata befejeződött. Minden tevékenységnek lehetnek *előzményei*, olyan munkafolyamatok, amelyeknek előbb be kell fejeződni ahhoz, hogy az adott tevékenység elkezdődhessen. A munkafolyamat lépéseinek ilyen összefüggését egy projekt-hálózattal adjuk meg.

A tevékenységeket a hálózat gráfjának irányított élei definiálják, a csúcsok pedig a tevékenységek csoportjainak befejezését jelzik. A csúcsokat emiatt *eseménynek* is nevezzük. Az ilyen projekt-hálózatot AOA (Activity On Arc) hálózatnak nevezzük. Ennek illusztrálásához tekintsük ismét a korábbi ábránkat:



Ezen most az 1. csúcs jelzi a projekt kezdetét, a 4. a *befejezés csúcs*. A 2. csúcs az egy hosszú első tevékenység végét, és a 3., illetve 4. csúcsokba vezető munkalépések kezdetét jelzi. Ha egy csúcsba több él is befut (mint pl. a 3. csúcsba), akkor ez azt jelenti, hogy mindkét korábbi tevékenységnek be kell

fejeződnie ahhoz, hogy az esemény utáni megkezdődhessen. Az előzmények nélküli tevékenységeket az első csúcsból kiinduló élekkel adjuk meg.

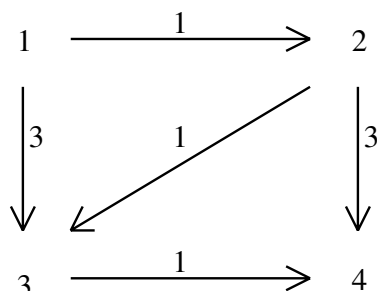
A projekt-hálózatot alakítsuk ki úgy, hogy egy tevékenység végét mutató csúcs sorszáma mindig nagyobb legyen, mint a kezdetét jelzőé. Ennek a szabálynak persze több reprezentáció is megfelel. További feltétel, hogy két adott csúcs között csak egy él mehet. Feltesszük még, hogy egy tevékenységet csak egy él reprezentálhat.

Az utóbbi két feltétel kielégítéséhez szükség lehet az ún. *fiktív tevékenységekre*. Például abban az esetben, amikor az A és B munkafolyamatok azonos feltétellel hajthatók végre, és mindkettő közvetlen előzménye a C tevékenységnek. Ilyen esetben a B lépést egy új csúcshoz köthetjük, ahonnan egy nulla időtartamú fiktív tevékenység új D éle adja a C munkafolyamat megkezdésének feltételét.

Tekintsük most a következő projekt feltételrendszert:

Tevékenység	Előzmények	Időtartam (nap)
A = anyagbeszerzés	–	1
B = alkalmazottak kiképzése	–	3
C = segédanyag termelése	A	1
D = válogatás és csomagolás	A	3
E = szakmunka	B, C	1

Ennek a projektnek épp a korábbi irányított éleket tartalmazó gráfunk felel meg az egyes csúccsal mint a projekt kezdetével, és a négyes csúccsal mint befejezés csúccsal:



Vegyük észre, hogy bár csak a B és C tevékenységeket tüntettük fel, mint az E munkafázis előfeltételét, de a hálózat összefüggései miatt az A folyamatnak is be kell fejeződnie az E megkezdése előtt.

A CPM két kulcsfogalma az esemény *korai időzítése* (Early event Time, ET), és a *kései időzítés* (Late event Time, LT). Az i csúcs korai időzítése, $ET(i)$ az a legkorábbi időpont, amikor a csúcshoz tartozó esemény bekövetkezhet. Ennek megfelelően az i csúcs kései időzítése, $LT(i)$ az a legkésőbbi időpont, amikor a csúcshoz tartozó esemény bekövetkezhet anélkül, hogy a projekt előírt befejezési időpontját késleltetné. Meg lehet mutatni, hogy $ET(i)$ a kezdőponttól az i csúcsba vezető leghosszabb út hossza.

A projekt korai időzítésének kiszámolását az egyes csúccsal kezdjük, $ET(1) = 0$ adódik erre. Ebből kiindulva meghatározzuk a további $ET(i)$ értékeket az élek adta összefüggéseknek megfelelően. Ha egy csúcsba több él is befut, akkor a csúcs korai időzítése az előző összes részfolyamat teljesülését feltételezi.

Ez alapján az $ET(i)$ korai időzítés meghatározását a következő lépések adják:

1. lépés Keressük meg az i csúcsba befutó élek kezdő csúcspontjait. Ezek az események az i esemény *közvetlen előzményei*.
2. lépés Az i esemény minden közvetlen előzményének ET értékéhez adjuk hozzá az i -be vezető megfelelő élhez tartozó tevékenység időtartamát.

3. lépés $ET(i)$ egyenlő az előző lépésben számított összegek maximumával.

PÉLDA. Határozzuk meg a fenti projekt-hálózat korai időzítési értékeit! Definíció szerint $ET(1) = 0$. Az $ET(2)$ érték közvetlenül az $ET(1) + 1$ összegből adódik, mert a 2. csúcsba csak egy él fut be. A következő csúcsra már két tevékenység teljesülését kell vizsgálni: $ET(3) = \max\{ET(1) + 3, ET(2) + 1\} = \max\{0 + 3, 1 + 1\} = \max\{3, 2\} = 3$. Az utolsó csúcsra pedig $ET(4) = \max\{ET(2) + 3, ET(3) + 1\} = \max\{1 + 3, 3 + 1\} = \max\{4, 4\} = 4$.

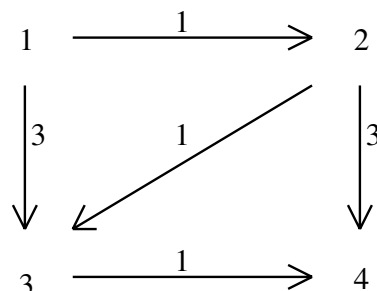
Ez alapján a teljes projekt befejezésének legkorábbi időpontja 4. $\square$

A kései időzítés meghatározását visszafelé, az utolsó, a befejezési csúcsból kiindulva kezdjük. Ebben az eljárásban ha egy i csúcsot több esemény követ, az utóbbiakra korábban számított LT értékekből le kell vonni az i csúcsból ezekhez vezető él hosszát. Az így kapott időpontok közül a legkorábbira teljesülnie kell az i eseménynek, hiszen csak ebben az esetben tartható a kitűzött határidő a teljes projekt teljesítésére.

Általában a befejezési csúcs kései időzítésének olyan időpontot szokás választani, amely alatt az mindenképpen elérhető az egyes csúcsból. Amennyiben az $LT(j)$ kései időzítési értékek mind ismertek a $j > i$ indexekre, az $LT(i)$ a következő eljárással számítható:

1. lépés Keressük meg azokat a csúcsokat, amelyekbe megy él az i csúcsból. Ezek az események az i esemény *közvetlen követői, utódai*.
2. lépés Az i esemény minden közvetlen utódának LT értékéből vonjuk le az i -ből az utódba vezető élhez tartozó tevékenység időtartamát.
3. lépés $LT(i)$ egyenlő az előző lépésben számított értékek minimumával.

PÉLDA. Határozzuk meg a kései időzítés értékeket a jól ismert irányított gráfunkra:



Legyen az $LT(4)$ érték 4, hiszen ekkorra a projekt befejezhető. Innen $LT(3) = 4 - 1 = 3$, hiszen a 3. csúcsnak csak a 4. az utódja. Ezután $LT(2) = \min\{LT(4) - 3, LT(3) - 1\} = \min\{4 - 3, 3 - 1\} = \min\{1, 2\} = 1$. Hasonlóan $LT(1) = \min\{LT(3) - 3, LT(2) - 1\} = \min\{3 - 3, 1 - 1\} = \min\{0, 0\} = 0$. Eszerint legkésőbb 4 időegységgel kell a projektet kezdeni a tervezett teljes készülségi esemény előtt. $\square$

Amennyiben egy i csúcsra $ET(i) = LT(i)$, akkor az illető csúcsban való minden késedelem a projekt határidejének lekésését jelenti. Esetünkben a $(2, 4)$ él 2-re változtatása azt eredményezné, hogy a 2. esemény 1 egységgel később kezdődne a projekt befejezésének késedelme nélkül: ekkor $ET(2) = 1$, és $LT(2) = 2$.

A projekt tervezésekor fontos ismeret az, hogy az egyes tevékenységek mekkora késedelme nem veszélyezteti még a teljes projekt határidőre való befejezését. Egy tevékenység, illetve az ennek megfelelő (i, j) él *tűrőhatára* az a $TH(i, j)$ szám, amennyivel a tevékenység kezdete a legkorábbi lehetséges időponttól eltolódhat anélkül, hogy a projekt befejezése elkésne – a többi tevékenység pontos

végrehajtását feltételezve.

Legyen t_{ij} az (i, j) tevékenység hossza. Az (i, j) tevékenység késése mellett akkor tartható a projekt eredeti határideje, ha az i . esemény korai időzítése + az (i, j) tevékenység hossza + a k tűréshatár még mindig a j . esemény kései időzítésénél nem későbbi időpontot ad. Eszerint

$$ET(i) + t_{ij} + k \leq LT(j),$$

amiből a tűréshatárra

$$TH(i, j) = LT(j) - ET(i) - t_{ij}$$

adódik.

A példánkra adódó tűréshatárok:

$$TH(1, 2) = LT(2) - ET(1) - 1 = 1 - 0 - 1 = 0,$$

$$TH(1, 3) = LT(3) - ET(1) - 3 = 3 - 0 - 3 = 0,$$

$$TH(2, 3) = LT(3) - ET(2) - 1 = 3 - 1 - 1 = 1,$$

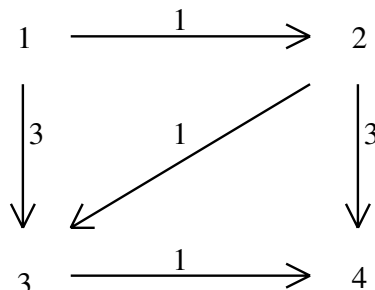
$$TH(2, 4) = LT(4) - ET(2) - 3 = 4 - 1 - 3 = 0,$$

$$TH(3, 4) = LT(4) - ET(3) - 1 = 4 - 3 - 1 = 0.$$

Ennek megfelelően csak a $(2, 3)$ tevékenység halasztható (egy egységgel) anélkül hogy ez a teljes projekt csúszását okozná.

A nulla tűréshatárú tevékenységek bármilyen elhúzódnása késlelteti a projekt befejezését. Ezért az ilyen éleket *kritikus tevékenység*nek nevezzük. A csak kritikus tevékenységekből álló, a kezdés csúcsból a befejezés csúcsba vezető utat *kritikus útnak* hívjuk.

A vizsgált példánkon az $(1, 2)$ és $(2, 4)$ tevékenységek például egy kritikus utat adnak meg, mert ezekre a tűréshatár nulla volt.



Természetesen egy projekt gráfjában több kritikus út is lehet. A tevékenységek flexibilitásának a tűréshatár mellett további mérőszáma a *mozgáshatár*: Egy tevékenység, illetve az ezt reprezentáló (i, j) él mozgáshatára az az időtartam, amennyivel a tevékenység kezdete (vagy hossza) elhúzódhat anélkül, hogy ezzel bármely későbbi tevékenység kezdési időpontja a korábbi kezdési időpontjánál későbbre tolódna. Ennek jelölése $MH(i, j)$.

A mozgáshatár meghatározását a tűréshatárhoz hasonlóan tehetjük meg: tegyük fel, hogy az i . esemény bekövetkezése, illetve az (i, j) tevékenység hossza k időegységnyit tolódik. Ebben az esetben a j esemény akkor nem csúszik az (i, j) tevékenység miatt, ha

$$ET(i) + t_{ij} + k \leq ET(j).$$

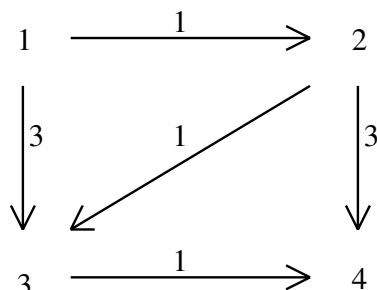
Ebből a mozgáshatár definíciója:

$$MH(i, j) = ET(j) - ET(i) - t_{ij}.$$

Mivel a példánkra az $ET(i)$ értékek megegyeznek a megfelelő $LT(i)$ értékekkel, ezért a mozgáshatárok is megegyeznek a korábbi tűréshatárokkal.

A kritikus út meghatározásának egyik lehetséges módja az ismertetett módszer a tűréshatár értékek kiszámításával, majd ezek ismeretében a nulla tűréshatárú élekből álló út megkeresése a kezdés csúcsból a befejezés csúcsig.

Emellett a kritikus út meghatározására lineáris programozási feladatot is fel lehet írni, amely tükrözi az eredeti probléma sajátosságait. Tekintsük példafeladatunkat ismét:



Jelölje x_i az i . esemény bekövetkeztének időpontját. Minden (i, j) tevékenységre érvényes, hogy a j esemény előtt be kell következnie az i -nek, és az (i, j) tevékenységnek is be kell fejeződnie. A kritikus út meghatározása a $z = x_F - x_1$ függvény minimalizálását jelenti, ahol F a befejezés csúcs. A példánkra adódó lineáris programozási feladat innen:

$$\min z = x_4 - x_1,$$

feltéve, hogy

$$x_2 \geq x_1 + 1,$$

$$x_3 \geq x_1 + 3,$$

$$x_3 \geq x_2 + 1,$$

$$x_4 \geq x_2 + 3,$$

$$x_4 \geq x_3 + 1.$$

A változókra érdemes nemnegativitási feltételt alkalmazni, sőt, $x_1 = 0$ természetes választás lehet. Az utóbbi feltételek nélkül az Excel a -2, -1, 1, 2 értékeket adta, és a kritikus út hosszára 4-et kaptunk. Ha x_1 értékét rögzítettük nullára, akkor az optimális megoldás megfelelően a 0, 1, 3, 4 értékekre módosult.

A kritikus út meghatározására felírt lineáris programozási feladatnak általában sok optimális megoldása van (főleg ha több tűréshatár pozitív).

Reális és gyakori feladat az, hogy egy projekt-hálózat ismert kritikus út megoldását módosítani kell úgy, hogy a teljes projekt időtartamát kell csökkenteni minimális költséggel – feltételezve, hogy minden tevékenység hossza csökkenthető egy ismert költség fejében.

Módosítsuk a példánkat annyiban, hogy minden tevékenység hossza csökkenthető féllal a következő költségek megfizetése esetén: 1, 2, 3, 4, 5. Legyenek az új változók, amelyek az egyes tevékenységek lerövidítésének mértékét adják, rendre A , B , C , D és E . A cél azt meghatározni, hogy hogyan kell ütemezni a projektet ahhoz, hogy az eredeti 4 hosszú végrehajtás 3.5-re módosuljon, és a tevékenységek gyorsításának összköltsége minimális legyen. Az ennek megfelelő lineáris programozási feladat:

$$\min z = 2A + 4B + 6C + 8D + 10E,$$

feltéve, hogy

$$A \leq 0.5, \quad B \leq 0.5, \quad C \leq 0.5, \quad D \leq 0.5, \quad E \leq 0.5,$$

$$x_2 \geq x_1 + 1 - A,$$

$$x_3 \geq x_1 + 3 - B,$$

$$x_3 \geq x_2 + 1 - C,$$

$$x_4 \geq x_2 + 3 - D,$$

$$x_4 \geq x_3 + 1 - E,$$

$$x_4 - x_1 \leq 3.5.$$

Itt $x_1 = 0$, és A, B, C, D, E nemnegatív. Az optimális megoldás

$$x_1 = 0, \quad x_2 = 0.5, \quad x_3 = 2.5, \quad x_4 = 3.5, \quad A = 0.5, \quad B = 0.5, \quad C = D = E = 0.$$

Ez azt jelenti, hogy az $(1, 2)$ és az $(1, 3)$ tevékenységeket kell postamunkában végezni.

15.5. Program kiértékelés és felülvizsgálat, PERT

A CPM, a kritikus út módszere azt feltételezi, hogy a tevékenységek időtartama ismert. A projekt lerövidítési vizsgálat kivételével a munkafolyamatok hossza rögzített. Ezzel szemben a PERT megközelítés a tevékenységek időtartamában a bizonytalanság figyelembevételét is lehetővé teszi.

A PERT egy munkafolyamat időtartamáról három adatot vesz számításba:

- a = a tevékenység legrövidebb lehetséges időtartama,
- b = a tevékenység leghosszabb lehetséges időtartama, és
- m = a tevékenység időtartama legvalószínűbb értéke.

Jelölje a T_{ij} valószínűségi változó az (i, j) munkafolyamat időtartamát. A PERT megközelítés felteszi, hogy ezek a valószínűségi változók béta eloszlásúak. Amennyiben a T_{ij} valószínűségi változó béta eloszlást követ, akkor várható értéke és szórásnégyzete (varianciája):

$$E(T_{ij}) = \frac{a + 4m + b}{6},$$

$$\text{var } T_{ij} = \frac{(b - a)^2}{36}.$$

A PERT feltételezi azt is, hogy a munkafolyamatok időtartamai egymástól független valószínűségi változók. Ebben az esetben a projekt-hálózat egy útjának időtartama mint valószínűségi változó

$$\sum_{(i,j) \in \text{út}} E(T_{ij}),$$

az út teljes idejének varianciája pedig

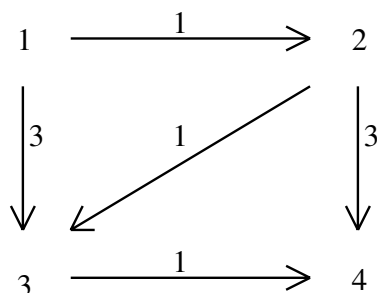
$$\sum_{(i,j) \in \text{út}} \text{var } T_{ij}.$$

Jelölje most a CP valószínűségi változó a CPM által meghatározott kritikus út tevékenységeinek teljes időtartamát. A PERT feltételezi, hogy a kritikus útban elegendően sok tevékenység szerepel ahhoz, hogy alkalmazni lehessen a centrális határeloszlás tételt, és így megállapíthassuk, hogy a

$$CP = \sum_{(i,j) \in \text{kritikus út}} T_{ij}$$

valószínűségi változó közelítőleg normális eloszlású.

PÉLDA. Tekintsük ismét a korábbi projekt-hálózatunkat :



Ehhez most meg kell adni az egyes munkafolyamatok időtartamának eloszlás-paramétereit:

Tevékenység	a	b	m
(1,2)	0.5	1.5	1.0
(1,3)	2.0	4.0	3.0
(2,3)	0.5	1.5	1.0
(2,4)	2.0	4.0	3.0
(3,4)	0.5	1.5	1.0

Vegyük észre, hogy a legvalószínűbb végrehajtási időknek minden élre a korábbi rögzített értékeket választottuk. Ezekkel a számokkal meghatározhatjuk az egyes tevékenységek várható időtartamát és varianciáját:

$$E(T_{12}) = \frac{0.5 + 4 * 1 + 1.5}{6} = 1, \quad \text{var } T_{12} = \frac{(1.5 - 0.5)^2}{36} = \frac{1}{36} = 0.028,$$

$$E(T_{13}) = \frac{2 + 4 * 3 + 4}{6} = 3, \quad \text{var } T_{13} = \frac{(4 - 2)^2}{36} = \frac{4}{36} = 0.111,$$

$$E(T_{23}) = \frac{0.5 + 4 * 1 + 1.5}{6} = 1, \quad \text{var } T_{23} = \frac{(1.5 - 0.5)^2}{36} = \frac{1}{36} = 0.028,$$

$$E(T_{24}) = \frac{2 + 4 * 3 + 4}{6} = 3, \quad \text{var } T_{24} = \frac{(4 - 2)^2}{36} = \frac{4}{36} = 0.111,$$

$$E(T_{34}) = \frac{0.5 + 4 * 1 + 1.5}{6} = 1, \quad \text{var } T_{34} = \frac{(1.5 - 0.5)^2}{36} = \frac{1}{36} = 0.028.$$

Vegyük észre, hogy paraméterezésünk mellett a kapott várható értékek megegyeznek a korábbi rögzített időtartamokkal. A fiktív élekre a várható érték és a variancia is nulla lenne.

Mivel a feltételezés szerint az egyes projekt szakaszok hosszai független valószínűségi változók, ezért a kritikus út teljes ideje várható értéke és annak szórásnégyzete a kritikus út szakaszaira kapott értékek összege lesz. Tekintsük az (1,2) és (2,4) szakaszokból álló kritikus utat:

$$\sum_{(i,j) \in \text{kritikus út}} E(T_{ij}) = E(T_{1,2}) + E(T_{2,4}) = 1 + 3 = 4,$$

a teljes idő varianciája pedig

$$\sum_{(i,j) \in \text{kritikus út}} \text{var } T_{ij} = \text{var } T_{12} + \text{var } T_{24} = 0.028 + 0.111 = 0.139.$$

Ugyanezeket az értékeket kapjuk a másik, (1,3), (3,4) kritikus útra is. A varianciából a megfelelő szórás $\sqrt{0.139} = 0.373$. Eszerint a kritikus utakon a várható értéktől való eltérés várható értéke ez a 0.373.

Feltételezve, hogy a teljes kritikus út megtételéhez szükséges idő normális eloszlású (ez példánkra aligha teljesülhet), meghatározhatjuk annak a valószínűségét, hogy a teljes projekt befejeződik adott idő, mondjuk 5 nap alatt.

Ez a valószínűség $P(CP \leq 5)$. Standardizáljuk a normális eloszlású valószínűségi változónkat:

$$P(CP \leq 5) = P\left(\frac{CP - 4}{0.373} \leq \frac{5 - 4}{0.373}\right) = P(Z \leq 2.681) = 0.996.$$

Itt az $F(2.681) = 0.996$ értéket a standard normális eloszlás táblázatából olvastuk ki. Itt $F(x)$ a standard normális eloszlás eloszlásfüggvénye. A PERT szerint tehát a megadott feltevések mellett annak a valószínűsége, hogy a teljes projekt 5 nap alatt befejeződik, 99.6%.

A standard normális eloszlás megfelelő értékét a táblázat használata helyett megtudhatjuk:

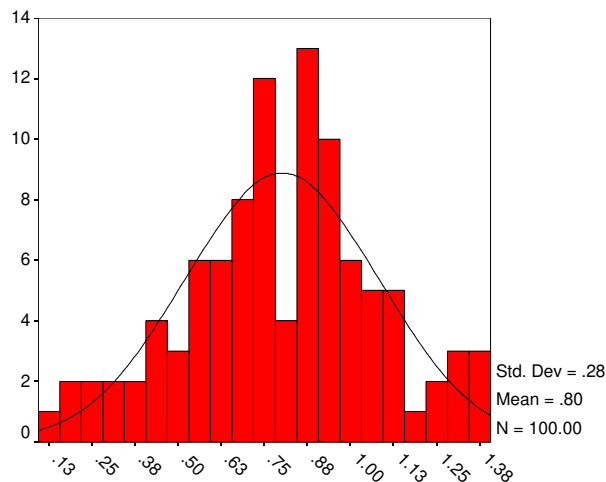
- az Excel `NORM.ELOSZL` függvényét használva,
- az SPSS nevű statisztikai program `CDF.NORMAL(2.681, 0, 1)` utasításával,
- a Matlab `0.5*erfc(-2.681/1.414)` parancsával (ehelyett persze a Matlab statisztikai csomagja `normcdf` utasítása a jobb megoldás — már ha az elérhető),
- a Maple pedig a `stats[statevalf,cdf,normald](2.681);` utasítással adja a kért valószínűségi értéket.

A PERT alkalmazása során tett feltevések nehezen teljesíthetők.

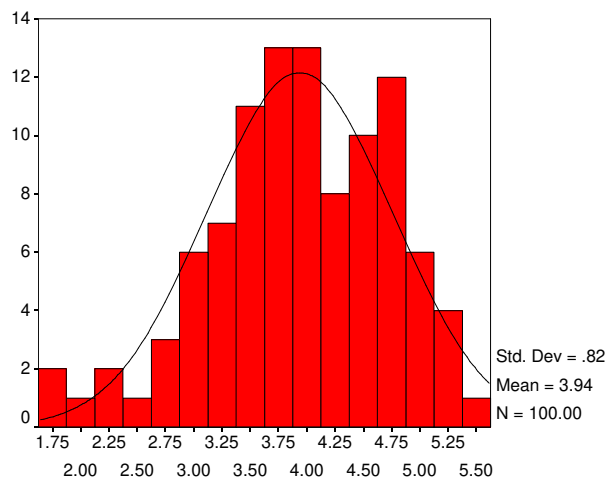
- Így nem könnyű igazolni, hogy az egyes tevékenységek időtartamai egymástól függetlenek.
- Sokszor a munkafolyamatok ideje nem béta eloszlást követ.
- A centrális határeloszlás tétel feltételei teljesüléséhez lehet, hogy nincs elegendő tevékenység a vizsgált útban.
- Gyakran előfordul, hogy a várható időtartamokra alkalmazott CPM eljárás eredménye nem marad kritikus út a véletlen események hatására.

Az utolsó probléma megoldására alkalmazhatunk Monte Carlo szimulációt annak meghatározására, hogy az egyes tevékenységek milyen valószínűséggel kritikusak, illetve hogy mennyi lehet a teljes projekt teljesülése idejének várható értéke és szórása.

A béta eloszlású valószínűségi változók összege:



KETTO



TIZ

Itt az első ábra mutatja két béta eloszlással generált (RV.BETA(2,3)) véletlen szám összegének eloszlását az SPSS nevű program hisztogram rajzoló utasítása eredményeként. Be van rajzolva az illeszkedő normális eloszlás sűrűségfüggvénye is. A következő ábra tíz béta eloszlású véletlen szám összegének eloszlását mutatja. Mindkét megjelenített eloszlás eltér a normálistól.

15.6. Ellenőrző kérdések és gyakorló feladatok

1. Mutasson olyan legrövidebb út feladatot, amelyben minden ideiglenes címke változatlanul véglegessé válik!
2. Tesztelje, hány béta eloszlású valószínűségi változót kell ahhoz összeadnia, hogy az összeg szemre megkülönböztethetetlen legyen egy normális eloszlású változótól!

3. Keressen olyan gyakorlati feladatot, amelyben az egyes tevékenységek végrehajtási idejei valóban független valószínűségi változók!

16. fejezet

Sztochasztikus programozás

Ahogy az újságáros probléma példáján is látszani fog, egy *sztochasztikus programozási probléma* szokás szerint egy alapul szolgáló *determinisztikus feladatra* épül. Miután kiderült, hogy valamely változó valójában egy valószínűségi változóval adható meg, egy újabb, *sztochasztikus optimalizálási modellt* fogalmazunk meg.

A *modell* és *feladat* szavakat szinonimaként használhatjuk. Szigorúbb értelemben a modell adja meg a feladatra vonatkozó feltételezéseket, és határozza meg azokat a matematikai objektumokat, amik a rendszerünk paramétereinek megfelelnek. Ez alapján aztán felírhatjuk a konkrét megoldandó feladatot, majd az eredményt alkalmazhatjuk leírási vagy működtetési céljainkra.

Abban az esetben, amikor a modellre vonatkozó döntést a véletlen esemény előtt kell meghozni, *statikus modellről* beszélünk. Ide tartozik az újságáros probléma is. Ezzel szemben *dinamikus modellnek* nevezünk olyan modelleket, amelyek olyan rendszerekre vonatkoznak, amelyek állapotaikat időben változtatják. Amennyiben a rendszer viselkedését nem befolyásolják véletlen folyamatok, akkor az *optimális vezérlést* a rendszer indulása előtt meg lehet határozni.

A *sztochasztikus dinamikus rendszer* esetén az optimális működtetéshez szükség van a valószínűségi változók aktuális értékeinek megfigyelésére, és a rendszer működésére való hatásuk megállapítására.

Az alapul vett determinisztikus feladatokra a következő két példa szolgál kiindulási pontként:

Meghatározandó olyan x , hogy

$$g_1(x, \xi) \geq 0, \quad g_2(x, \xi) \geq 0, \quad \dots, \quad g_r(x, \xi) \geq 0,$$

$$x \in D,$$

ahol D egy adott, nem véletlen halmaz, amit gyakran véges sok x -re vonatkozó egyenlőtlenség határoz meg. A ξ szimbólum olyan vektort jelöl, amelynek komponensei majd valószínűségi változók lesznek.

A második determinisztikus feladat egy szélsőérték keresés:

$$\min h(x, \xi)$$

feltéve, hogy

$$g_1(x, \xi) \geq 0, \quad g_2(x, \xi) \geq 0, \quad \dots, \quad g_r(x, \xi) \geq 0,$$

$$x \in D,$$

ahol D ismét egy adott, nem véletlen halmaz, amit gyakran véges sok x -re vonatkozó egyenlőtlenség határoz meg. A ξ szimbólum itt is olyan vektort jelöl, amelynek komponensei majd valószínűségi változók lesznek.

Ezeknek a feladatoknak fontos speciális esetei az alábbi problémák:

Meghatározandó olyan x , hogy

$$Tx \geq \xi, \quad (\text{vagy} \quad Tx = \xi),$$

$$Ax = b \quad \text{és} \quad x \geq 0,$$

illetve az, hogy

$$\min h(x, \xi)$$

feltéve, hogy

$$Ax = b \quad \text{és} \quad x \geq 0,$$

ahol esetleg nem csak ξ , de a T mátrixok egyes elemei is véletlen valószínűségi változók.

A legelső determinisztikus problémának felel meg például a következő *valószínűség maximalizálási feladat*:

$$\max P(g_1(x, \xi) \geq 0, g_2(x, \xi) \geq 0, \dots, g_r(x, \xi) \geq 0),$$

úgy, hogy $x \in D$.

16.1. Az újságárus probléma

Számos közgazdasági probléma fogalmazható meg, vagy vezethető vissza arra a feladatra, amikor azt szeretnénk meghatározni, hogy mekkora raktárkészletet tartsunk fenn, illetve rendeljünk meg, ha mind a raktáron tartásnak, mind a túl kicsi készletnek ismert költsége van. A döntéshozó célja nyilván a lehetséges legnagyobb nyereség elérése, illetve a költségei minimalizálása.

Az *újságárus problémának* azokat a feladatokat nevezzük amelyek elegettesznek a következő feltételeknek:

- Arról kell dönteni, hogy mennyi árut rendeljünk. Legyen ennek mennyisége q .
- A d egységnyi kereslet a beszerzett árura egy ismert, $p(d)$ valószínűséggel fordulhat elő. Itt d nemnegatív szám, és a D valószínűségi változó reprezentálja a keresletet.
- A d és q értékekre vonatkozóan egy $c(d, q)$ költség merül fel.

Egy újságárus valóban a fenti problémával szembesül. Ha az adott napi forgalmat alulbecsli, akkor egyes vevőket nem tud kiszolgálni, így lehetséges nyereségtől esik el. Másrészt ha túl sokat rendel, akkor annak a költségeit fizetnie kell, és az el nem adott példányok alapján nyilván nem jut nyereséghez. Ha a kereslet diszkrét valószínűségi változó, akkor a *diszkrét keresletű újságárus problémáról* van szó.

A költségfüggvény alakja arra az esetre, amikor a készlet nem kisebb, mint az igény ($d \leq q$):

$$c(d, q) = c_o q + a,$$

ahol c_o az *egységnyi túlkészletezés* pozitív költsége. Itt tehát c_o az a költség, amivel számolni kell akkor, ha a már így is túlzott készletet még egy egységgel növeljük. a a q -tól nem függő tagokat jelzi. A később bemutatott eljárás miatt ennek részletezése nem szükséges.

Abban az esetben, amikor a megrendelt áru mennyisége kisebb, mint az igény ($d \geq q + 1$), akkor a költségfüggvény:

$$c(d, q) = -c_u q + b,$$

ahol c_u a pozitív *egységnyi alulkészletezési költség*. Ez az az összeg, amivel a költségeinket csökkenteni tudjuk, ha egy egységgel több a készletünk. Itt b a q -tól nem függő tagokat jelzi.

Az újságárus probléma elemzése során tekintsük most a költségfüggvény változását. Ha $E(q)$ a költségfüggvény várható értéke abban az esetben, ha q egységnyt rendeltünk az áruból, akkor a feladat olyan q^* optimális rendelés meghatározása, amely minimalizálja E értékét.

Amennyiben az $E(q)$ konvex függvény, akkor elegendő az $E(q + 1) - E(q)$ értékeket meghatározni. Konvex függvény esetén ugyanis a feladatunk annak a legkisebb q -nak a megkeresésére egyszerűsödik, amelyre $E(q + 1) - E(q)$ pozitív. Azt az eljárást, amely ez alapján az $E(q + 1) - E(q)$ ismételt kiértékelésével határozza meg a keresett optimális megrendelést, *határelemzésnek* nevezik. Alkalmazásának feltétele, hogy a célfüggvény konvex legyen, és hogy az említett különbség könnyen számítható legyen.

Az $E(q + 1) - E(q)$ különbség meghatározásához két esetet kell megvizsgálni:

1. Amikor $d \leq q$. Ekkor egy újabb egység megrendelése további túlkészletezést okoz. Ez c_o -al növeli a költséget. Ennek az esetnek a bekövetkezési valószínűsége $P(D \leq q)$, ahol D a kereslet valószínűségi változója.

2. A másik eset az, amikor $d \geq q + 1$. Ekkor egy további egység megrendelése csökkenti a hiányt, és így csökkenti a költséget is c_u -val. A második eset bekövetkeztének valószínűsége $P(D \geq q + 1) = 1 - P(D \leq q)$.

A fentiek alapján tehát az összes eset $100 \cdot P(D \leq q)$ százalékában a $q + 1$ egység megrendelése c_o -val kerül többbe, mint q egység rendelése, és az esetek $100 \cdot (1 - P(D \leq q))$ százalékában a $q + 1$ egységnyi készlet költsége c_u -val kevesebbe kerül, mint q egységnyié. Ezek alapján átlagosan a $q + 1$ áruegység megrendelése

$$E(q + 1) - E(q) = c_o P(D \leq q) - c_u (1 - P(D \leq q)) = (c_o + c_u) P(D \leq q) - c_u$$

költséggel kerül többbe, mint a q egység rendelése. Mivel a $P(D \leq q)$ valószínűség q növekedésével nő, ezért ha $c_o + c_u$ nemnegatív (ez az esetek többségében reális feltételezés), akkor a fenti különbség monoton nőni fog, így teljesül a határelemzés feltétele.

Ha $E(q + 1) - E(q) \geq 0$, akkor $(c_o + c_u) P(D \leq q) - c_u \geq 0$ -ból $P(D \leq q) \geq c_u / (c_o + c_u)$ adódik. Ha $F(q) = P(D \leq q)$ a kereslet eloszlásfüggvénye, akkor azt a minimális q értéket keressük, amire még teljesül

$$F(q) \geq \frac{c_u}{c_o + c_u}.$$

PÉLDA. Tekintsük azt a feladatot, amikor egy jegyzet kiadásakor a nyomtatott példányszámról kell dönteni. A hallgatóság létszáma ismeretében a várható igényeket többé-kevésbé jól lehet becsülni. A kérdés nyilván az, hogy hány példányban készüljön a jegyzet ahhoz, hogy pl. a három éven belüli teljes költség minimális legyen.

A konkrét számadatok legyenek a következők: az előállítási költség 900 Ft, az eladási ár 1500 Ft. A harmadik év végén a teljes maradék készletől megszabadulunk féláron: eladjuk őket egy nagykereskedőnek (750 Ft). A három éven belül eladható példányok valószínűsége:

eladott jegyzet	valószínűség
50	0.30
100	0.20
150	0.20
200	0.10
250	0.10
300	0.10

A jelölésünk kövesse a bevezetőt: legyen q a megrendelt jegyzetek száma, d pedig a három éven belül ténylegesen eladottak száma. Amennyiben az eladott példányok száma kevesebb, mint a megrendelteké ($d < q$), akkor a teljes költség a következők szerint alakul:

tevékenység	költség
q darab jegyzet vásárlása	$900 q$
d darab jegyzet eladása	$-1500 d$
$q - d$ jegyzet eladása féláron	$-750 (q - d)$
teljes költség	$150 q - 750 d$

A további vizsgálatunk szempontjából ebből az a fontos, hogy a q egységnyi növelése 150 Ft-nyi költség-növekedést jelent (hiszen 900-ért állítják elő a jegyzetet, és a fölös példányoktól csak 750-ért tudunk megszabadulni). Vegyük észre, hogy a jegyzet előállításának a példányszámtól nem függő költségei az optimális példányszámot nem befolyásolják, hiszen ezeket minden q -ra azonos módon kell megfizetni.

Amennyiben az igényelt példányok száma legalább annyi, mint a megrendelteké ($d \geq q$), akkor a teljes költség a következők szerint alakul (a negatív költséget nyereségként értelmezzük):

tevékenység	költség
q darab jegyzet vásárlása	$900 q$
q darab jegyzet eladása	$-1500 q$
teljes költség	$-600 q$

Ebben az esetben a q egységnyi növelése a költségeket 600 forinttal csökkenti (a nyereséget 600 Ft-al növeli). Az eddigi költségelemzés alapján a túlkészletezési költség $c_o = 150$, az alulkészletezési költség pedig $c_u = 600$.

Alkalmazzuk most az optimális megrendelésre vonatkozó levezetett feltételt:

$$F(q) \geq \frac{c_u}{c_o + c_u} = \frac{600}{150 + 600} = \frac{600}{750} = 0.8.$$

A megadott valószínűségek alapján példánkban 200 darab jegyzet megrendelése az optimális, mert erre adódik először a kapott $0.3+0.2+0.2+0.1 = 0.8$ eloszlásfüggvény érték.

A kapott eredmény illusztrálásaként határozzuk meg az egyes költség eltéréseket

$$E(q+1) - E(q) = (c_o + c_u)P(D \leq q) - c_u$$

alapján az egymásra következő eltérések rendre:

$$E(51) - E(50) = (150 + 600) \cdot 0.3 - 600 = 225 - 600 = -375,$$

$$E(101) - E(100) = 750 \cdot 0.5 - 600 = 375 - 600 = -225,$$

$$E(151) - E(150) = 750 \cdot 0.7 - 600 = 525 - 600 = -75,$$

$$E(201) - E(200) = 750 \cdot 0.8 - 600 = 600 - 600 = 0,$$

$$E(251) - E(250) = 750 \cdot 0.9 - 600 = 675 - 600 = 75,$$

$$E(301) - E(300) = 750 \cdot 1.0 - 600 = 750 - 600 = 150.$$

16.2. A folytonos keresletű újságáros probléma

Bizonyos értelemben az előző példában is feltételeztük a kereslet finomabb felbontását, mint az épp definiáltat. Mégis, az az eset külön tárgyalandó, amikor a D keresletet egy folytonos valószínűségi változóval reprezentáljuk. Az előző modellre használt határelemzési eljárást ennek megfelelően módosítani kell.

Eszerint a döntéshozó várható költségét az a q^* megrendelés minimalizálja várható értékben, ahol q^* az a legkisebb megrendelési érték, amelyre teljesül

$$P(D \leq q) = \frac{c_u}{c_o + c_u}.$$

A feltételben az egyenlőséget az tette lehetővé, hogy a kereslet most folytonos valószínűségi változó. Egyszerűen belátható, hogy a fenti feltétel ekvivalens az

$$P(D \geq q) = \frac{c_o}{c_o + c_u}$$

egyenlettel.

PÉLDA. A légitársaságok nemrég még bevett gyakorlata volt, hogy a legnagyobb lehetséges nyereség elérése érdekében több jegyet adtak el, mint ahány utas az adott gépre felfér – arra számítva, hogy nem minden utas fog ténylegesen utazni, egyesek lemondják az utat különböző okok miatt. Vizsgáljuk meg az újságáros probléma modelljével, hogy egy adott esetben hogyan határozható meg az optimális *túlfoglalás*.

A Fokker F70 gép 79 utast tud szállítani. Tegyük fel, hogy egy járatra a jegy ára 40 eFt, a túlfoglalás miatt a gépről lemaradó utas kárpótlás és egy másik járat drágább ára miatt 20 eFt többletköltséget jelent (és visszatérítik a teljes jegyárat). A tapasztalatok szerint a jeggyel rendelkező, de meg nem jelent utasok száma közel normális eloszlást követ 10 várható értékkel és 3 szórással.

Legyen most q a légitársaság által az adott járatra eladott jegyek száma, d pedig a meg nem jelent utasok száma (ez eltér a korábban szokásos jelentéstől). Ekkor $q - d$ lesz a ténylegesen utazásra jelentkezők száma. Ha $q - d \leq 79$, akkor mindenki utazhat, és ekkor a légitársaság költsége $-40(q - d)$ (ezer Forintban). Ha $(q - d) > 79$, akkor 79 utas lesz a gépen (ennek költsége $-79 \cdot 40$), és $q - d - 79$ utas kap kárpótlást fejenként 20 eFt értékben. Ekkor tehát a légitársaság teljes költsége $20(q - d - 79) - 40 \cdot 79 = 20q - 20d - 60 \cdot 79 = 20q - 20d - 4740$.

Amennyiben $q - 79$ a döntési változónk, akkor egy folytonos keresletű újságáros problémát kell megoldani. A fenti adatok alapján $c_u = 40$, és $c_o = 20$. Az optimális döntés feltétele:

$$P(D \leq q - 79) = \frac{c_u}{c_o + c_u} = \frac{40}{20 + 40} = \frac{2}{3}.$$

Standardizáljuk a D valószínűségi változónkat a 10 várható érték és a 3 szórással felhasználásával:

$$P\left(\frac{D - 10}{3} \leq \frac{q - 79 - 10}{3}\right) = 0.6.$$

Bevezetve a $Z = (D - 10)/3$ standard normális eloszlású valószínűségi változót, optimalitási feltételnek azt kapjuk, hogy

$$P\left(Z \leq \frac{q - 79 - 10}{3}\right) = 0.6.$$

A standard normális eloszlás táblázatából megtudhatjuk, hogy $P(Z \leq 0.43) = 0.6664$ (és a következő értékre, 0.44-re már 0.67 valószínűség adódik). Innen a közelítő optimális megoldás

$$0.43 = \frac{q - 79 - 10}{3},$$

azaz

$$q = (0.43 \cdot 3) + 89 = 90.29.$$

Eszerint a légitársaságnak az adott feltételek mellett a legelőnyösebb 90 vagy 91 jegyet eladnia a 79 tényleges férőhelyre. Természetesen ha az igény ennél kisebb, akkor annyi jegyet érdemes kiadni. $\square$

16.3. Ellenőrző kérdések és gyakorló feladatok

1. Keressen olyan gyakorlati problémát, amelyben folytonos a döntési változó az újságáros feladat keretében!
2. Írjon programot valószínűségi változó standardizálására!
3. Milyen tőzsdei kereskedési helyzet felel meg az újságáros feladat föltételeinek?

17. fejezet

Gradiens módszer

A korlátozás nélküli nemlineáris optimalizálási feladatok gyakorlati problémákban gyakran lépnek fel. Általános alakjuk

$$\min f(x),$$

ahol a célfüggvény kétszer folytonosan differenciálható ($f \in C^2$), $f : \mathbb{R}^n \rightarrow \mathbb{R}$. A megoldás egyik módszere az, hogy az eredeti nemlineáris függvényt a megoldáshoz tartó pontokban kvadratikus függvényekkel közelítjük, és a következő iterált a közelítésből kapott megoldás lesz.

A feladat megoldásához helyi kereső eljárásokat szokás használni, amelyek az indulópontához tartozó helyi minimumpont megkeresésére vállalkoznak, általában monoton nem növekvő célfüggvényérték mellett.

Amennyiben a második derivált, a $H_{ij}(x) = \partial^2 f(x) / \partial x_i \partial x_j$ képlettel adott Hesse mátrix ismeretes, akkor a leghatékonyabb a Newton módszer. Ez az érintőmódszer alapján működik, ami egydimenziós nemlineáris egyenletet old meg az $x_{k+1} = x_k - f(x_k) / f'(x_k)$ iterációs képlettel. A többdimenziós optimalizálási feladatra ennek a következő formula felel meg:

$$x_{k+1} = x_k - H^{-1}(x_k) \nabla f(x_k),$$

ahol $\nabla f(x_k)$ az $f(x)$ függvény gradiense az x_k pontban.

A korszerű számítógépes megvalósításokban a megadottnál kisebb lépést szokás tenni. Az ilyen Newton módszerre bizonyos feltételek teljesülése esetén kvadratikus konvergencia érvényes, azaz egy megfelelő x^* helyi minimumpontra

$$\|x^* - x_{k+1}\| \leq C \|x^* - x_k\|^2$$

érvényes egy alkalmas pozitív C konstansra.

Gyakran a Hesse mátrix nem állítható elő egyszerűen, vagy pedig numerikus differenciálással jól közelíthető. Az erre az esetre módosított Newton módszernek *quasi-Newton eljárás* a neve. Erre kvadratikus konvergencia már nem érvényes, csak a *superlineáris konvergencia*, azaz teljesül

$$\lim_{k \rightarrow \infty} \frac{\|x^* - x_{k+1}\|}{\|x^* - x_k\|} = 0.$$

Amennyiben csak a gradiensértékre lehet támaszkodni, akkor olyan eljárást is fel lehet építeni, amely az adott iterációs pontból a negatív gradiens irányában lép tovább:

$$x_{k+1} = x_k - \lambda \nabla f(x_k),$$

ahol λ a lépésköz. Az olyan módszereket, amelyek keresési iránya a negatív gradienssel pozitív belső szorzatot ad, *gradiens módszereknek* nevezzük.

A gradiens módszernek is vannak olyan változatai, amelyek nem igénylik a célfüggvény deriváltjának ismeretét, ennek megfelelő közelítését maga az eljárás állítja elő. A gradiens módszercsalád általában csak lineáris konvergenciát mutat, de a konjugált gradiens módszerrel bizonyos feladatosztályon el lehet érni a szuperlineáris konvergenciát.

PÉLDA. Tekintsük az $f(x) = (x_1 - 1)^2 + (x_1 + x_2 - 2)^2$ függvényt. Ennek a célfüggvénynek a minimuma az $x_1 = 1$, $x_2 = 1$ pontban van, értéke 0. A gradiens

$$\nabla f(x) = (2(x_1 - 1) + 2(x_1 + x_2 - 2), 2(x_1 + x_2 - 2))^T,$$

a Hesse mátrix és inverze pedig

$$H(x) = \begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}, \quad \text{illetve} \quad H(x)^{-1} = \begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 1.0 \end{bmatrix}.$$

Ennek alapján a Newton módszer adta iteráció az $x_0 = (3, 3)^T$ pontból indulva:

$$x_1 = x_0 - H^{-1}(x_0)\nabla f(x_0) = \begin{pmatrix} 3 \\ 3 \end{pmatrix} - \begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 1.0 \end{bmatrix} \begin{pmatrix} 2 \cdot 2 + 2 \cdot 4 \\ 2 \cdot 4 \end{pmatrix}.$$

Innen

$$x_1 = \begin{pmatrix} 3 \\ 3 \end{pmatrix} - \begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 1.0 \end{bmatrix} \begin{pmatrix} 12 \\ 8 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix} - \begin{pmatrix} 6 - 4 \\ -6 + 8 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix} - \begin{pmatrix} 2 \\ 2 \end{pmatrix}.$$

Tehát ez alkalommal egy lépésben megkaptuk az $(1, 1)^T$ optimális megoldást. Ez a jelenség a lényegében kvadratikusan konvergenciára fordulhat elő.

PÉLDA. Tekintsük az előző feladatot, és oldjuk meg a gradiens módszerrel! Minimalizálandó tehát az $f(x) = (x_1 - 1)^2 + (x_1 + x_2 - 2)^2$ függvény. Ennek minimuma az $x_1 = 1$, $x_2 = 1$ pontban van, értéke 0. A gradiens

$$\nabla f(x) = (2(x_1 - 1) + 2(x_1 + x_2 - 2), 2(x_1 + x_2 - 2))^T.$$

Vegyünk egy viszonylag kis lépésközt, $\lambda = 0.1$ -et, és ismét az $x_0 = (3, 3)^T$ indulópontot. Az iterációs sorozat első lépései ezzel az

$$x_{k+1} = x_k - \lambda \nabla f(x_k)$$

képlet alapján:

$$x_1 = x_0 - \lambda \nabla f(x_0) = \begin{pmatrix} 3 \\ 3 \end{pmatrix} - 0.1 \begin{pmatrix} 12 \\ 8 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix} - \begin{pmatrix} 1.2 \\ 0.8 \end{pmatrix} = \begin{pmatrix} 1.8 \\ 2.2 \end{pmatrix}.$$

A következő iterált pontok a Matlab

```
» x = x - 0.1 * [ 2 * (x(1) - 1) + 2 * (x(1) + x(2) - 2) ; 2 * (x(1) + x(2) - 2) ]
```

utasításával kiszámítva:

$$x_2 = \begin{pmatrix} 1.24 \\ 1.80 \end{pmatrix}, \quad x_3 = \begin{pmatrix} 0.984 \\ 1.592 \end{pmatrix}, \quad x_4 = \begin{pmatrix} 0.8720 \\ 1.4768 \end{pmatrix}.$$

A következő néhány kiválasztott közelítő vektor:

$$x_{10} = \begin{pmatrix} 0.8359 \\ 1.2722 \end{pmatrix}, \quad x_{20} = \begin{pmatrix} 0.9245 \\ 1.1221 \end{pmatrix}, \quad x_{50} = \begin{pmatrix} 0.9930 \\ 1.0113 \end{pmatrix}, \quad x_{100} = \begin{pmatrix} 0.9999 \\ 1.0002 \end{pmatrix}.$$

Az iterált vektorok lineáris konvergenciára utalnak.

17.1. Konjugált gradiens módszer

A konjugált gradiens módszer optimalizálásra és szimmetrikus pozitív definit mátrixú lineáris egyenletrendszerek megoldására is 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:

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

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

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

Balról beszorozva s_k^T -vel, és megoldva ezt 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 $k = 1, 2, \dots$ 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 *gradiens* módszerek 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 megoldás felé. A konjugált gradiens módszer ezzel szemben a lépésenkénti megfelelő 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:

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

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 számbábrázolás 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.

A lineáris egyenletrendszerek megoldására szolgáló iterációs Matlab eljárásokat összegeztük az alábbi táblázatban:

függvény	mátrix típus	módszer
bicg	általános	bikonjugált gradiens módszer
bicgstab	általános	stabilizált bikonjugált gradiens módszer
cgs	általános	négyzetes konjugált gradiens módszer
gmres	általános	általánosított minimum-reziduál módszer
minres	Hermite-szimmetrikus	minimum-reziduál módszer
lsqr	általános	konjugált gradiens normális egyenletekre
pcg	Herm. poz. def.	prekondicionált konjugált gradiens
qmr	általános	kvázi-minimál reziduál módszer
symmlq	Hermite-szimmetrikus	szimmetrikus LQ módszer

Ezek a függvények (a `gmres` kivételével) azonos hívási formátumot használnak. A legegyszerűbb hívási mód az

$$x = \text{solver}(A, b),$$

ahol `solver` a táblázatban szereplő egyik eljárás neve. Ha a megállási feltételben a toleranciát módosítani szeretnénk, akkor a hívási forma

$$x = \text{solver}(A, b, \text{tol}),$$

ahol a `tol` érték az a szám, amellyel a $\text{norm}(b - A \cdot x) \leq \text{tol} \cdot \text{norm}(b)$ feltétel teljesülését követeljük meg. A tolerancia alapbeállítása $1e-6$.

Egy adott $n \times n$ -es A mátrix nemnulla elemei százalékos arányát a következő Matlab utasítás adja:

```
» nnz(A) / n^2
```

A gyakrabban használatos, érdekes mátrixok, vektorok közvetlenül is elérhetők a Matlabban:

```
» b = ones(n, 1);
```

az egyesekből álló n hosszú oszlopvektort adja.

```
» A = gallery('wathen', 12, 12); n = length(A)
N =
    481
» nnz(A) / n^2
ans =
    0.0301
```

a 481×481 -es Whaten mátrixot generálja, amelynek rögzített ritkasági szerkezete van véletlen elemekkel. A nemnulla elemek aránya kb. 3%. A prekondicionált konjugált gradiens módszer a következő eredményt adja a fentiekben definiált lineáris egyenletrendszerre.

```
» x = pcg(A, b);
pcg stopped at iteration 20 without converging to the desired
```

```
tolerance 1e-006 because the maximum number of iterations was
reached.
```

```
The iterate returned has relative residual 0.063
```

Ez azt jelenti, hogy az előírt megállási feltétel nem teljesült még a reziduálra, több iteráció végrehajtását kell ehhez engedélyezni.

```
x = pcg(A,b,1e-6,100);
pcg converged at iteration 86 to a solution with relative
residual 8.8e-007
```

Nagyon tanulságos a 12 helyett nagyobb paraméterrel futtatni a fenti utasításokat. 40 esetén az n már közel 5000, a nem nulla mátrixelemek aránya 3 ezrelék. Erre a feladatra a `pcg` eljárás kb. 6 másodpercig futott, míg az $x = b \setminus A$ hétszer tovább. Érdekes, hogy ha a futtatást megismételtük, akkor már közel azonos időre volt szükség. Ennek az lehet a magyarázata, hogy az ismételt futtatás esetén már a memóriában volt a több száz megabyte-nyi adat.

Az iteratív eljárások a hatékony működéshez általában prekondicionálást igényelnek, az eredeti

$$Ax = b$$

egyenlet helyett az M_1 és M_2 mátrixokkal, illetve az $M = M_1 M_2$ mátrixszal a következő egyenleteket fogják használni:

$$M_1^{-1} A M_2^{-1} \cdot M_2 x = M_1^{-1} b,$$

vagy pedig

$$M^{-1} A x = M^{-1} b.$$

Az átalakítás célja az, hogy az eredménymátrix bizonyos értelemben közel legyen az egység-mátrixhoz. A jó prekondicionáló mátrixok meghatározása nehéz feladat, és általában az adott alkalmazás ismeretét kívánja meg, amiből a lineáris egyenletrendszer származik.

Az általunk vizsgált A mátrixnak egy jó prekondicionálója az $M = \text{diag}(\text{diag}(A))$ mátrix, az A mátrix főátlója elemeiből álló diagonális mátrix. Ezzel mint ötödik argumentummal felhívva a `pcg` eljárást, lényegesebben gyorsabban kapunk a megállási feltételnek megfelelő megoldást:

```
» [x,flag,relres,iter] = pcg(A,b,1e-6,100,diag(diag(A)));
» flag, relres, iter
flag =
    0
relres =
    9.0568e-007
iter =
    28
```

Vegyük észre, hogy amikor egynél több eredmény-argumentumot kérünk, akkor nem jönnek üzenetek. A `flag` nulla értéke azt mutatja, hogy a megoldás teljesíti az előírt megállási feltételt `iter` darab iterációs lépés után.

17.2. Ellenőrző kérdések és gyakorló feladatok

1. Milyen optimalizálási feladatra ad megoldást a gradiens módszer egy lépésben?

2. Milyen optimalizálási feladatra ad megoldást a Newton módszer egy lépésben?
3. Igaz-e, hogy minden kvadratikusan konvergens optimalizálási algoritmus gyorsabb minden lineárisan konvergens eljárásnál?

18. 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 fát* épít fel a következők szerint:

- 0. lépés** Az előkészítés során határozzuk meg a $g(L)$ értéket, és legyen L a leszámmlálási 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 válasszunk ki egy minimális címkéjű L' levelet.
- 2. lépés** Amennyiben L' már csak egy vektorból áll ($L' = \{\bar{x}\}$), akkor vége az eljárásnak, $\bar{x}$ optimális megoldás.

3. lépés Bővítjük 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 címkéként a megfelelő levelekhez. Növeljük a k iterációs számot 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 $f(\bar{x}) \leq f(x)$ adódik az $\bar{x}$ vektorra. Ez pedig pontosan azt jelenti, hogy $\bar{x}$ optimális megoldás.

Gyakran hasznos a lehetséges megoldások L halmazát befoglalni egy könnyebben kezelhető halmazba, és a felosztást azon végigkövetni. Érdemes az alpmódszer 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 $x_i \in \{0, 1\}, i = 1, \dots, 4$.

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 $\bar{x} = (1, 0, 0, 0)$ 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 legígéretebb 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-ről 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

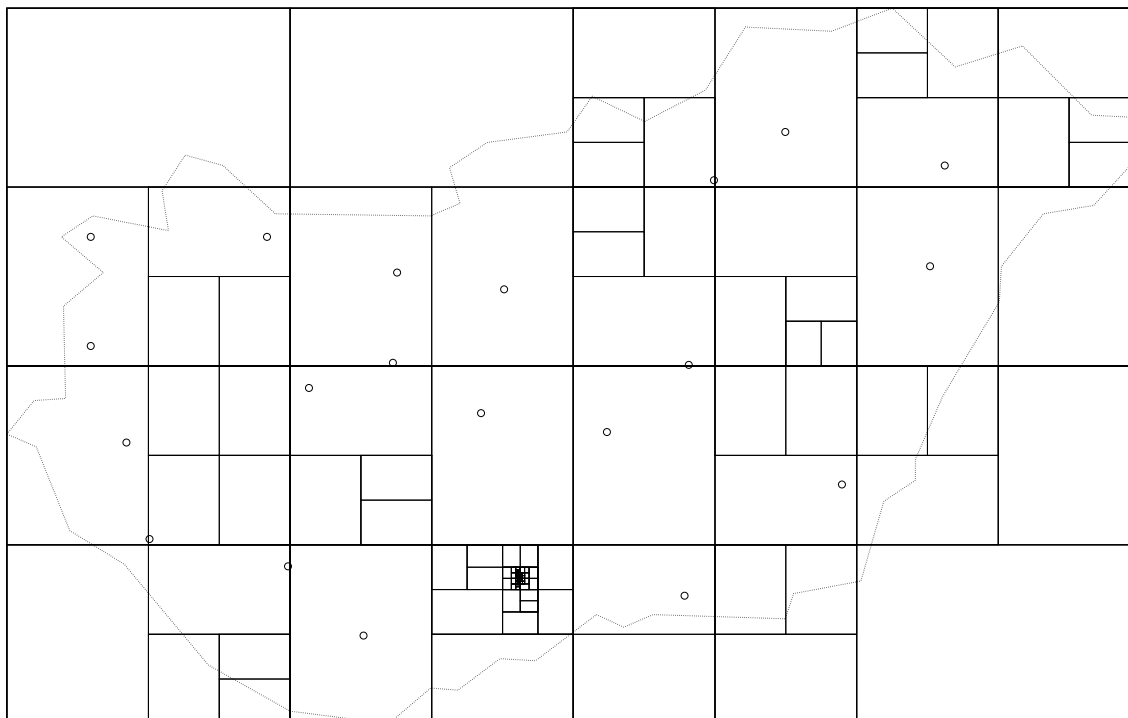
$$f(x) = \sum_i \frac{l_i}{(x - x_i)^2 + (y - y_i)^2},$$

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.

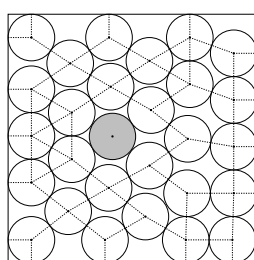
18.1. Körpakolási feladatok

Két ekvivalens megfogalmazás:

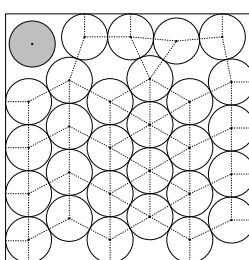
- Helyezzünk el adott n darab egybevágó kört átlapolás nélkül, maximális sugárral az egységnyezetben.
- Helyezzünk el adott n számú pontot az egységnyezetben úgy, hogy a köztük lévő minimális távolság maximális legyen.

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

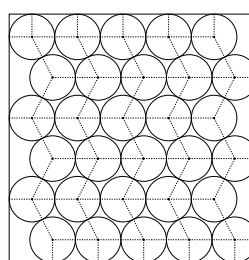
$$\text{ahol } 0 \leq x_i, y_i \leq 1, \quad i = 1, 2, \dots, n.$$



$n = 28$



$n = 29$



$n = 30$

A satírozott körök kis mértékben mozgathatók az optimalitás megtartása mellett (a globális minimum-pontok 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:

$$F_{28}^* = [\underline{0.2305354936426673}, \underline{0.2305354936426743}], \quad w \approx 7 \cdot 10^{-15},$$

$$F_{29}^* = [\underline{0.2268829007442089}, \underline{0.2268829007442240}], \quad w \approx 2 \cdot 10^{-14},$$

$$F_{30}^* = [\underline{0.2245029645310881}, \underline{0.2245029645310903}], \quad w \approx 2 \cdot 10^{-15}.$$

A teljes futási idők: ≈ 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.

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 isotonicity
direkt kereső	direct search method
egészértékű optimalizálási feladat elsőrendű módszer	integer optimization problem first order method
feltétel nélküli optimalizálási feladat	unconstrained optimization problem
globális minimum globális minimumpont gradiens módszer	global minimum (többes szám: global minima) global minimizer point gradient method
helyi minimum helyi minimumpont	local minimum (többes szám: local minima) local minimizer point
intervallum felosztási módszer	interval subdivision method
konkáv függvény konvex függvény korlátozott feladat kvadratikusan függvény	concave function convex function bounded problem quadratic function
legmeredekebb lejtő módszere Lipschitz-folytonos függvény	steepest descent method Lipschitzian function
maximum másodrendű módszerek minimum művelet kiterjesztése	maximum (többes szám: maxima) second order methods minimum (többes szám: minima) operation overloading
nemdifferentiálható függvény négyzetösszeg típusú függvény	non-differentiable function sum-of-squares type objective function

nyeregpon	saddle point
optimalizálási feltétel	constraint
regresszió	regression
regressziós egyenlet	regression equation
sima függvény	smooth function
stacionárius pon	stationary point, critical point
szeparált célfüggvény	separable objective function
szeparált helyi minimumpon	separable local minimizer
szigorú globális minimumpon	strict global minimizer
szigorú helyi minimumpon	strict local minimizer
véletlen keresés	random search
vonzáskörzet	region of attraction

A. függelék

Tematika

Optimalizálás alkalmazásai

Az Optimalizálás Alkalmazásai című tárgy felvételének feltétele: az Operációkutatás I. tárgy teljesítése. A tárgy bevezetést ad az optimalizálási modellezésbe, módszerekbe és alkalmazásukba.

Előadás: Kötelező, 2 óra/5 kredit. Teljesítési módja: Kollokvium.

Gyakorlat: Kötelező, 1 óra/0 kredit. Teljesítési módja: Aláírás.

Tematika

- A hátizsák feladat
- Utazó ügynök feladat
- A szabási feladat
- A maximális folyam probléma
- Sztochasztikus programozás
- Újságárus probléma
- A kritikus út és a PERT módszer
- Gradiens módszer
- A korlátozás és szétválasztás módszere

Ajánlott irodalom

1. Bajalinov Erik és Imreh Balázs: Operációkutatás, Polygon, Szeged, 2001.
2. Az anyagot tartalmazó jegyzet előzetesen elérhető lesz a következő internetes címen: <http://www.inf.u-szeged>.
3. R.B. Kearfott: Rigorous Global Search: Continuous Problems, Kluwer, 1996.

Összehasonlításul, a jelen tárgy elődje, az Operációkutatás II. tárgy tematikája röviden:

- Dualitás
- Egészértékű programozás
- Hozzárendelési feladat megoldása magyar módszerrel
- Szállítási feladat megoldása magyar módszerrel
- Hiperbolikus programozási feladat
- Konvex programozási feladat
- Gradiens módszer

Ugyanezen tárgy tételjegyzéke:

Egészértékű programozási feladat, LP relaxáció, kerekítés; Gomory-féle metsző síkok módszere; Dual all integer eljárás; Hozzárendelési feladat (ekvivalens költségmátrixokra vonatkozó segédttétel, a magyar módszer, a mátrixsorozat tulajdonságai), ezek alapján az optimális megoldás konstrukciója; Hozzárendelési feladat, a magyar módszer helyességének igazolása; tiltásos hozzárendelési feladat; Szállítási feladat (modell, megoldhatóság szükséges és elegendő feltétele, ekvivalens költségmátrixokra vonatkozó segédttétel); Szállítási feladat (magyar módszer, az előálló mátrixsorozat tulajdonságai, ezek alapján az optimális megoldás meghatározása); Szállítási feladat (magyar módszer, helyessége bizonyítása); Nyitott, tiltásos és korlátos szállítási feladatok; Hiperbolikus programozási feladat; Konvex programozási feladat; Általános gradiens módszer; Frank-Wolfe eljárás.

Tudományos és szimbolikus számítások

A kurzus célja, hogy olyan számítógépes eljárásokat ismerjünk meg, amelyek a hagyományos numerikus eljárások bizonyos értelemben vett ellentettjei: tehát a megoldásokat nem közelítőleg keressük.

Az alábbi tematika alapján a számítógépes algebra klasszikus módszereit tekintjük át:

- Matematikai objektumok ábrázolása - nagyméretű egészek, polinomok, hatványsorok
- Alapműveletek - számolás többszörös pontosságú számokkal és polinomokkal
- Racionális aritmetika - Euklideszi algoritmus, legnagyobb közös osztó, általánosított Euklideszi algoritmus
- Moduláris aritmetika - kínai maradéktétel, interpoláció
- Gyors szorzás - Karatsuba algoritmus, diszkrét Fourier transzformáció és gyors Fourier transzformáció, nagy pontosságú reciprok Newton iterációval
- Prímtesztelés, prímfelbontás
- Polinomaritmetika, polinomok gyökei, (rezultáns, Gröbner-bázis)
- Polinomok irreducibilis faktorizációja
- Véletlenszámok generálása
- Programkönyvtárak tudományos és szimbolikus számításokhoz

A gyakorlaton áttekintjük a tudományos számítások elvégzésére alkalmas programozási könyvtárakat.

B. függelék

Mintafeladatok a dolgozatokhoz

Feladatok röpdolgozatokhoz

Példa dolgozatfeladatok az évközi kis felmérő dolgozatokhoz:

1. Definiálja a hátizsák feladatot!
2. Adjon meg egy alkalmazási példát, amelyhez olyan hátizsák feladat tartozik, amelyben a mind a súlyok, mind a célfüggvény együtthatók pozitívak!
3. Definiálja a hátizsák feladatot!
4. Oldja meg a $\max 2x_1 + x_2, x_1 + 2x_2 \leq 2$ hátizsák feladatot!
5. Mi a lényegi eltérés a hátizsák- és a hajórakodási feladat között?
6. A fix költség feladatot melyik optimalizálási feladatosztályba sorolja?
7. Mutasson egy gyakorlati problémát, amely utazó ügynök feladatra vezet!
8. Definiálja a fix költség feladatot!
9. Oldja meg a $\max x_1 + 2x_2, 2x_1 + x_2 \leq 2$ hátizsák feladatot!
10. Mi a lényege az implicit leszámolási algoritmusnak?
11. Melyik optimalizálási feladatosztályba sorolja a hajórakodási feladatot?
12. Mi a megoldása az olyan utazó ügynök feladatnak, amelyben minden városok közti távolság egyenlő?
13. Mennyi a legolcsóbb beszúrás aszimptotikus hányadosa szimmetrikus, és a háromszög egyenlőtlenséget kielégítő távolság mátrixra?
 - 2
 - 3
 - 30 000
 - 300 000

Feladatok a tudásfelmérő dolgozathoz

Példaként álljon itt néhány feladat a félév végi tudásfelmérő dolgozatra való felkészüléshez, iránymutatásul. A felmérés célja az, hogy a hallgatók bemutassák, hogy az előadáson megismert fogalmak összefüggéseivel tisztában vannak, értik az ismertetett algoritmusokat, és az elhangzott információk lényegét elsajátították. A dolgozatban tíz feladat lesz, mindegyikkel 4 pontot lehet elérni. A kredit megszerzésének szükséges feltétele legalább 20 pont elérése.

1. Indokolja, hogy miért hasznos a hátizsák feladat implicit leszámolási eljárásához az, hogy a célfüggvény-együtthatók nem pozitívok, és hogy a súlyok növekvő sorrendben vannak az első lépés után!
2. Ha egy 2 Ghz-es PC 10 órajel alatt tudja egy vektorról eldönteni, hogy az lehetséges megoldása-e a hátizsák feladatnak, akkor egy nap alatt milyen méretű feladat megoldására lehet biztosan számítani (tehát a legrosszabb esetben)?
3. Írja le a hátizsák feladatot részletesen!
4. Mi a lényege a hátizsák feladat implicit leszámolással való megoldásának?
5. Milyen esetben előnyös a hátizsák feladat megoldására az implicit leszámolás módszere?
6. Mi a lényegi eltérés a hátizsák- és a hajórakodási feladat között?
7. Ismertesse részletesen az utazó ügynök feladatot!
8. Soroljon fel legalább 4 olyan gyakorlati feladatot, amelyek az utazó ügynök feladatra vezethetők vissza!
9. Adja meg az utazó ügynök feladat rövidebb alakját! Miért előnyös ez?
10. Mondja ki az utazó ügynök feladat két alakjának ekvivalenciájára vonatkozó tételt! Vázolja a bizonyítást!
11. Mit lehet mondani ekvivalens mátrixokhoz tartozó utazó ügynök feladatokról? Hogyan lehet bizonyítani?
12. Mi a viszonya az utazó ügynök feladatnak a hozzárendelési feladathoz?
13. Mi az oszlopgenerálás módszerének szerepe? Miért előnyös?
14. Ismertessen néhány heurisztikát az utazó ügynök feladatra!
15. Mi a szerepe a távolságvektornak az utazó ügynök feladat heurisztikáiban?
16. Milyen módszerek használatosak heurisztikus algoritmusok hatásosságának jellemzésére?
17. Tárgyalja részletesen az aszimptotikus hányados definícióját és jelentését!
18. Mit tud az utazó ügynök feladat heurisztikáinak aszimptotikus hányados értékeiről?
19. Fogalmazza meg a szabási feladat modelljét!
20. Adjon meg olyan gyakorlati feladatokat, amelyek a szabási feladatra vezetnek!
21. Definiálja a szabási feladat 4 heurisztikáját!
22. Definiálja a legrövidebb út feladatot részletesen!

23. Adja meg Dijkstra algoritmusát!
24. Adja meg a maximális folyam feladat alapfogalmait!
25. Adja meg a Ford-Fulkerson algoritmust és a címkézési eljárást!
26. Mondja ki a maximális folyam probléma címkézési eljárására vonatkozó állítást, és igazolja röviden!
27. Mondja ki és igazolja a folyamok erősségét korlátozó vágás kapacitásokra vonatkozó tételt!
28. Ismertesse a kritikus út módszerét a kapcsolódó fogalmakkal együtt!
29. Mutasson olyan példát, amelyben a CPM módszer korai és kései időzítése minden csúcsra megegyezik!
30. Magyarázza el a lényegi különbséget a tűréshatár és a mozgáshatár között!
31. Hogyan lehet a CPM módszert a projekt lerövidítése optimális változatának meghatározására használni?
32. Ismertesse a PERT módszert!
33. Mit tud az újságárus problémáról?!
34. Mi az eltérés a diszkrét- és a folytonos keresletű újságárus probléma megoldási módszere között?
35. Adjon meg egy olyan konkrét, részletes gyakorlati feladatot, amely az újságárus problémára vezethető vissza!
36. Mit tud a sztochasztikus programozási modellekről?
37. Ismertesse a nemlineáris optimalizálási feladatot és a megoldására szolgáló Newton módszert!
38. Ismertesse a konjugált gradiens módszert!
39. Adjon példát a nemlineáris optimalizálás témaköréből olyan algoritmusra, amely véges számú lépésben konvergál, amelyre lineáris-, szuperlineáris-, illetve olyanra is, amely kvadrátikus konvergencia érvényes!
40. Adjon példát az automatikus differenciálásra és az intervallum aritmetikára!
41. Mi mindenre jó a Newton módszer? Hogyan jellemezhető a konvergenciája?
42. Definiálja a korlátozás és szétválasztás módszerét! Mikor érdemes alkalmazni?

B.1. A tudásfelmérő dolgozat feltételei

- Először is, írja fel a nevét a dolgozatra, és írja alá a megfelelő helyen!
- Készítsék ki a diákigazolványukat, vagy más fényképes igazolványt az asztalra.
- Minden feladat 4 pontot ér, összesen tehát 40 pontot lehet elérni.
- A megoldásra 60 perc áll rendelkezésre, kezdje a neve megadásával, és az aláírással.
- A feladatokat önállóan kell megoldani.

- A mobiltelefonokat kérem kikapcsolni.
- A megadott hely elegendő a helyes válasz megadásához, ne kérjenek újabb papírlapot.
- A dolgozat megírásához semmilyen segédeszközt (kalkulátor, jegyzet, könyv, a szomszéd tanácsa, ...) sem szabad használni.
- A fenti szabályokat megszegőtől a dolgozatát elveszük, és a tudásfelmérés sikertelennek minősül.
- Ha nem szeretné, hogy a dolgozat eredményének közzétételekor a neve megjelenjen, akkor adja meg a hallgatói azonosítóját.
- Ha valami nem világos, kérem, kézfelemeléssel jelezze. A jelenlevő oktatók válaszolnak.

Jó munkát!

C. 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. [27]).

A MATLAB programot a Linux és a Windows operációs rendszerekben a szokásos módon, a megfelelő ikonra való dupla kattintással lehet elindítani. Az ezután kapott párbeszédés ablakban a felhasználó által begépett 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 sor végére. 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 $\text{fun}(x) = 1 + x - x^2/4$ függvényt a MATLAB Editor/Debugger ablakában, és megnézzük 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 MAT-

LAB 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  $1 \times 5$ -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. [27]) 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 ([27]), 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 párbeszédés 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 hasonlóak 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 egymásba ágyazott 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 kiírására 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 paramétereit is használhatnak, és más programokból szubrutinként hívhatók. Ennek bemutatására tekintsük a következő egyszerű programot, amit a MATLAB File / New menüpontban elérhető Editor / Debugger segítségével szerkesztünk meg, és mentjük 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
end
```

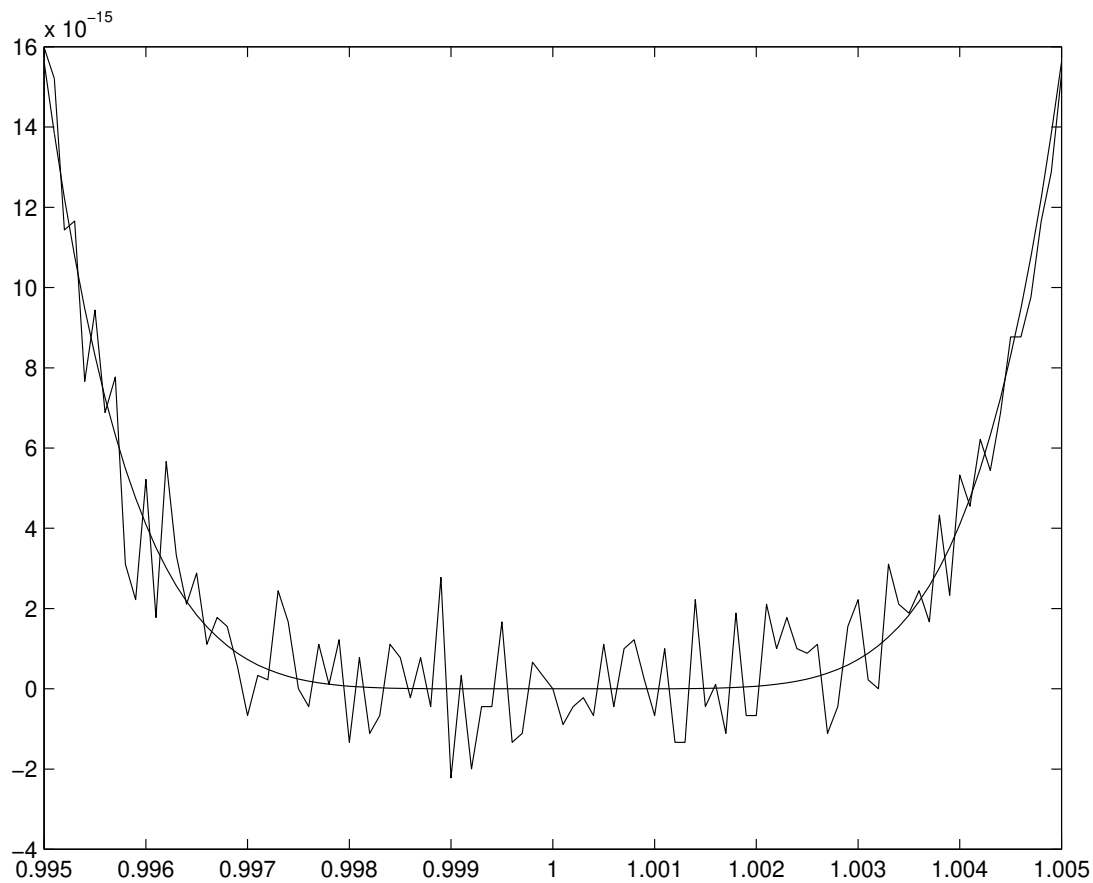
Ezután a MATLAB parancssorába írjuk be azt, hogy `P=pasc(5,3)`, és láthatjuk a mod 3 Pascal háromszög első 5 sorát. A másik érdekes teszt `P=pasc(175,3)`; (most fontos a pontosvessző), és aztán gépeltük be azt, hogy `spy(P)`, ami ritka mátrixot generál nagy n esetén.

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.

D. függelék

Az esszé, illetve kötelező program követelményei, témák

A félévi munka egyik legfontosabb eleme a megírandó esszé vagy kötelező program. Azt, hogy az adott félévben esszét vagy kötelező programot kell-e írni, a félév elején a gyakorlatvezető szabja meg. Ez a kapható összpontszám komoly részét adja, és ez lesz az eredménye a hallgatók önállóan végzett munkájának.

Az esszé

Az esszé egy a kötelező program megírásáról és teszteléséről beszámoló olyan rövid (15-20 oldalas) jelentés, amelynek a következő főbb részeket kell tartalmaznia:

1. A kitűzött feladat pontos, részletes megfogalmazása, a szakirodalom megismerése alapján a feladat alapos leírása, kitérve annak nehézségeire, és eltéréseire a szomszédos területektől. Fontos tárgyalni az alkalmazási területeket. Érdemes itt kis méretű, áttekinthető példákat használni.
2. A megoldására megírt, felhasznált (ha lehet, Matlab, Scilab, Netlib, esetleg Octave) programok részletes megadása, leírása. Ki kell térni az alkalmazott számítógépes megoldások okaira, előnyeire is (mint pl. a választott adattípus, számformátum, algoritmus változat, stb.). A programok tetszőleges programozási nyelven készülhetnek, de az előbb említettek mellett a JAVA is preferált, mert a hordozhatósága (?) miatt jó lehet az algoritmusok bemutatására.
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 $\text{\LaTeX}$ vagy Word legyen. A program futtatható változata és a forráskód is kell ehhez.

Világos, hogy a heti egy órányi gyakorlat nem elegendő az új tantárgy 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 hallgatót ahhoz,

hogy egy kapott feladat megoldásához a szükséges részletesebb ismereteket maga szerezzé 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. Az esszével kapcsolatos kérdésekre a hallgatónak kimerítő, teljes választ kell tudnia adni. Az azonos témájú esszéket összehasonlítják a gyakorlatvezetők. Az esszé értékelésének alapja épp a végzett önálló munka lesz.

Az esszé értékét növeli, ha az a Matlab, Scilab, Netlib vagy Octave programjait tárgyalja. A Matlab elérhető az intézeti kabinet gépein, de a gyakorlatvezetők egy hónapig használható bemutató változatot is tudnak adni otthoni implementálásra. 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.

A kötelező program

A kötelező program olyan házifeladat, amelyet Java nyelven kell írni, és a célja az, hogy a hallgatók bemutassák jártasságukat optimalizálási eljárások implementálásában, és az érintett algoritmusok működésének áttekinthető, könnyen érthető formában való megjelenítésében.

Az esszével szemben a kötelező program dokumentálása más szempontokra helyezi a hangsúlyt:

1. meg kell adni az algoritmus folyamatábráját vagy pszeudokódját,
2. a program teljes forrásszövegét, de legyen elérhető a program a hálózaton, és telepíthető is más gépekre,
3. meg kell adni a felhasználói útmutatót, amely alapján a program teljesértékűen használható, és végül
4. részletesen leírni egy jellemző futtatási példát, amely az algoritmus használhatóságát mutatja.

Tehát nem kell: az adott feladatkört ismertetni, részletesen megadni a kapcsolódó szakirodalmat, nem szükséges a megírt program hatékonyságát és hatásosságát teszteléssel igazolni. Mintaként érdemes megnézni a

<http://www.inf.u-szeged.hu/~csendes>

vendégoldalon a körpakolási illusztráló Java programot. Fontos az önálló munka. Az értékelés alapja az lesz, hogy milyen látványos, ügyes illusztrációt ad a program az adott algoritmushoz.

Esszé- és kötelező program témák

Itt vegyesen adtunk meg témákat esszéíráshoz, illetve kötelező programhoz. ezeken felül is lehet témát választani a gyakorlatvezetővel való egyeztetés alapján. A *-al jelölt feladatok nehezebbeknek számítanak, ezért sikeres megoldásukért többpont jár. Néhány esszétéma (ettől a gyakorlatvezető nyilván eltérhet, és előzetes egyeztetés alapján a hallgató maga választotta témát is kidolgozhat):

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 globális optimalizálási feladatokat.
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 nem triviális feladatokon.
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 ehhez a módszerhez igazodó globális optimalizálási feladatokat.
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 áttekinthető feladatokon.
11. *Több-dimenziós Lipschitz-optimalizálás.* Implementálja a több-dimenziós, az L Lipschitz konstans ismeretén alapuló módszert (vegye a lehetséges megoldások halmazának egy befoglaló intervallumát, és ezen hajtson végre egy korlátozás és szétválasztás eljárást az intervallum végpontokban kiértékelt célfüggvény és a Lipschitz konstans felhasználásával meghatározott alsó- és felső korlátokra támaszkodva), és mutassa be a működését áttekinthető feladatokon.
12. *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 néhány feladaton.

13. *Multistart*. Valósítsa meg a többszörös indítás módszerét (válasszon ki egyetlen eloszlással néhány ígéretes indulópontot, és hajtson végre helyi keresést ezekből startolva...) egy egyszerű formában, és illusztrálja a működését egy alkalmas feladaton.
14. *GLOBAL*. Töltse le a GLOBAL programot a következő internetes címről és oldjon meg vele globális optimalizálási feladatokat.
<ftp://ftp.jate.u-szeged.hu/pub/math/optimization/index.html>
15. *Intervallumos B&B**. Az alapl műveletek intervallumos kiterjesztését írja meg szubrutinként (vagy terjessze ki azokat a megfelelő műveletek túltöltésével), majd erre alapozva adjon meg egy egyszerű korlátozás és szétválasztás típusú optimalizáló eljárást. Tesztelje egy kiismerhető polinomon.
16. *Körpakolás négyzetben**. A feladat az egység-négyzetben meghatározni adott n -re azt az n darab pontot, amelyek közötti minimális távolság a lehető legnagyobb. A feladathoz tetszőleges alkalmas algoritmus használható.
17. *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 bevezethető.
18. *Kissing number**. Határozza meg (közelítő módszerrel), hogy egy magas dimenziós térben az origó körüli egység-sugarú gömbhöz hány azonos sugarú gömb illeszthető átfedés nélkül.
19. *A legrövidebb**. Adjon meg egy olyan, minimális hosszúságú globális optimalizálási algoritmust, amely a Shekel-10 feladatot 1 másodpercen belül képes megoldani legalább 5 jegy pontossággal.
20. Az utazó ügynök feladat Magyarország legnagyobb n településére*.
21. Hova helyezzünk el egy kellemetlen üzemet Magyarországon úgy, hogy a környező települések lakosai számával egyenesen, a távolsággal fordítottan arányos összútalat minimális legyen?*
22. Legkisebb négyzetek módszerének használata
23. Egyváltozós feltétel nélküli szélsőérték feladatok
24. Egyváltozós feltételes szélsőérték-feladatok
25. Többváltozós feltétel nélküli szélsőérték feladatok
26. Többváltozós feltételes szélsőérték feladatok
27. A NEOS optimalizálási szerver*
28. A GAMS modellezési nyelv, formalizálási rendszer*
29. A CPLEX optimalizálási rendszer*
30. A MINOS optimalizálási rendszer*
31. Milyen segítséget kaphatunk optimalizálási feladatok megoldásához a Maple szimbolikus algebrarendszertől?
32. Milyen segítséget kaphatunk optimalizálási feladatok megoldásához a Mathematica szimbolikus algebrarendszertől?

33. Milyen segítséget kaphatunk optimalizálási feladatok megoldásához a Derive szimbolikus algebrarendszertől?
34. Milyen segítséget kaphatunk optimalizálási feladatok megoldásához a Mu-pad szimbolikus algebrarendszertől?
35. Nemlineáris függvény minimumának megkeresése (több változós eset) a Matlab numerikus programrendszerben
36. Az automatikus differenciálás szerepe a nemlineáris optimalizálásban *
37. Az intervallum-aritmetika szerepe a nemlineáris optimalizálásban *
38. A mesterséges neuronhálók az optimalizálásban
39. Az ún. hangyakolóniák módszere minimalizálásban
40. Genetikus algoritmusok az optimalizálásban
41. Szimulált megeresztés (simulated annealing) az optimalizálásban
42. A nemlineáris szimplex módszer (Nelder-Mead)
43. Pakolási feladatok *
44. Geometriai alakzatok lefedési problémái *
45. Mit tud az Excel Solver az optimalizálás terén?
46. A Matlab INTLAB nevű kiegészítő csomagja az intervallumos műveletek támogatására
47. A perceptron használata az optimalizálásban
48. DC optimalizálás *
49. Az órarend összeállítása mint optimalizálási feladat *
50. A hátizsák feladat
51. A hajórakodási feladat
52. A fix költség probléma
53. Az utazó ügynök feladat
54. A kínai postás feladat
55. A legrövidebb út probléma
56. A maximális folyam feladat
57. Projektek ütemezése
58. Program kiértékelés és felülvizsgálat (PERT)
59. Az újságárus probléma
60. A folytonos keresletű újságárus probléma
61. Ütemezés

E. függelék

Egy minta esszé

Itt egy olyan esszét adunk meg, amely mintaként szolgál a hallgatóknak. Az itt megadott esszé kb. a kettős 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:

$$\min f(x)$$

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

$$h_j(x) = 0, \quad j = 1, 2, \dots, m_2,$$

ahol $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ 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 megelégszünk 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:

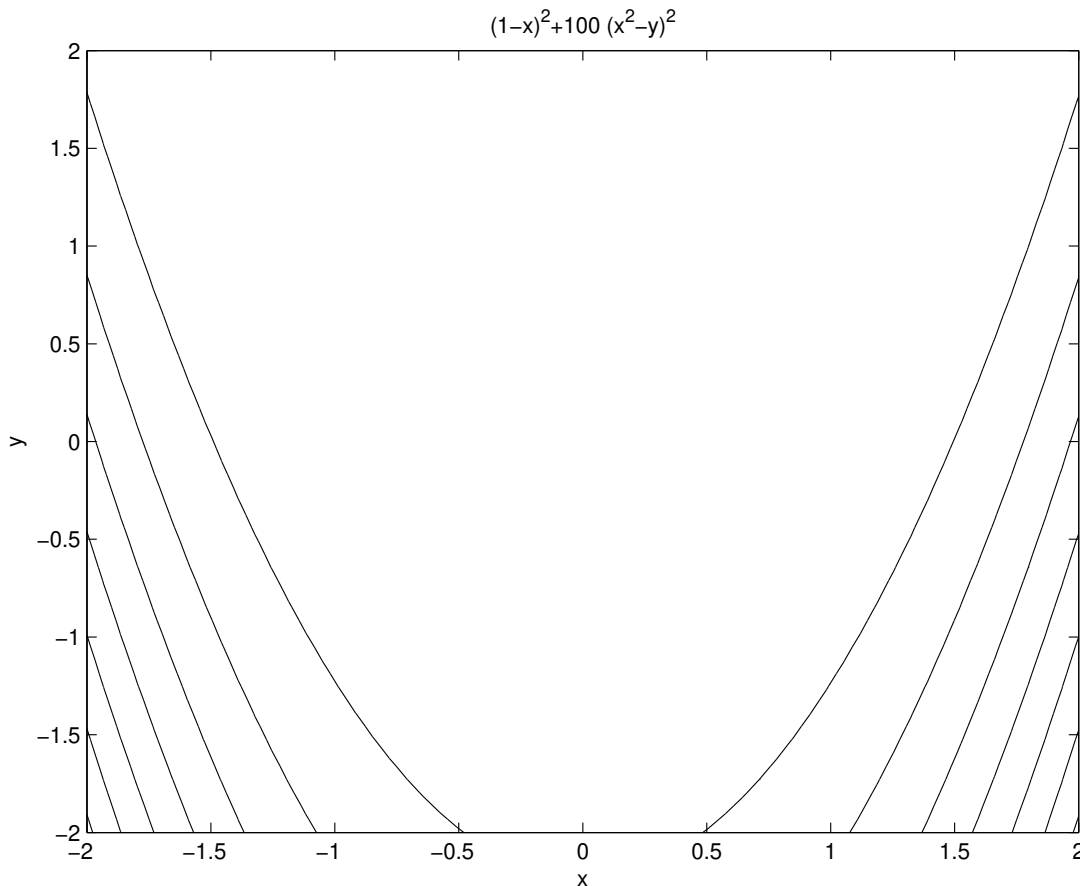
$$\min (1 - x_1)^2 + 100(x_1^2 - x_2)^2$$

$$-2 \leq x_1, x_2 \leq 2.$$

Az érdekességét az adja, hogy egyrészt szemléletesen azonnal látszik, hogy mi a megoldás, mégis a hagyományos nemlineáris optimalizáló algoritmusoknak meggyűlik a bajuk vele.

Ha papíron 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+100*(x^2-y)^2', [-2 2 -2 2])`; Matlab utasítással készült.

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 vi-

szont 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 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, ami nem nagyon szép. Bíztható azonban, hogy ha a keresett megoldás közelében adjuk meg a második argumentumban az indulóponot, akkor már jobb eredményt kapunk:

```
» fzero(' -2*(1-x)+200*(x^2-x)*(2*x-1)', 2)  
ans =  
1
```

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 ilyen 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:

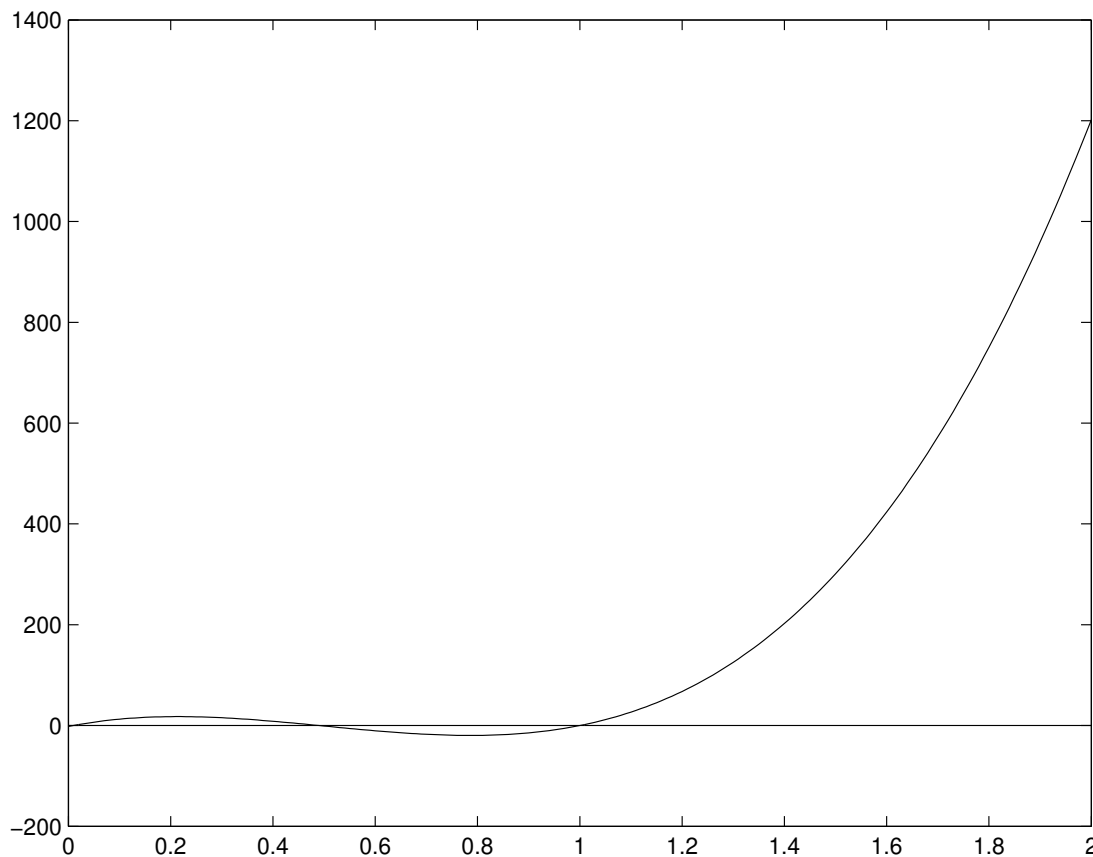
```
» x = x=0:0.01:2.0;
```

```
» 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
```

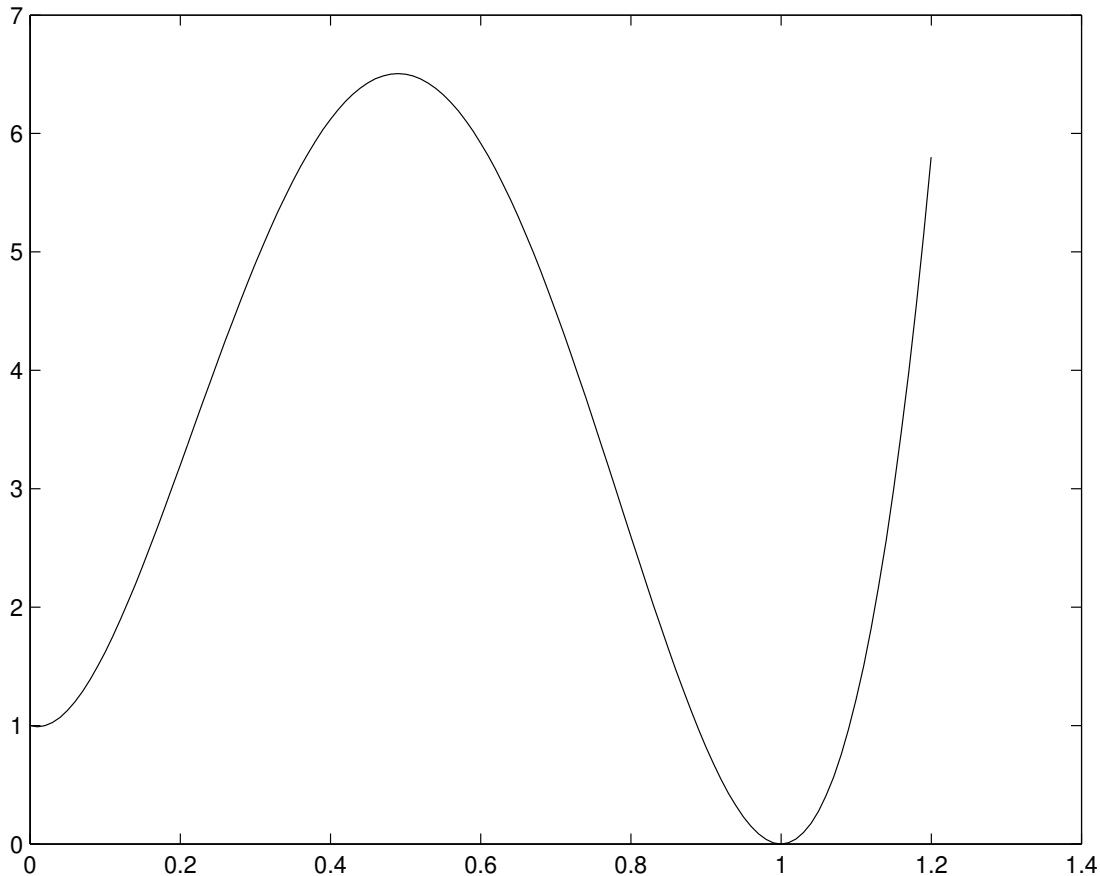
Ez rendben is van lényegében. A paraméterezés: függvény, alsókorlát, felsőkorlát. Vigyázzunk, az első utasítás után ne tegyünk pontosvesszőt, mert akkor az eredmény nem jelenik meg! Vegyük akkor az előző szakasz egydimenzióssá átalakított Rosenbrock függvényét.

```
>> 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 =
```

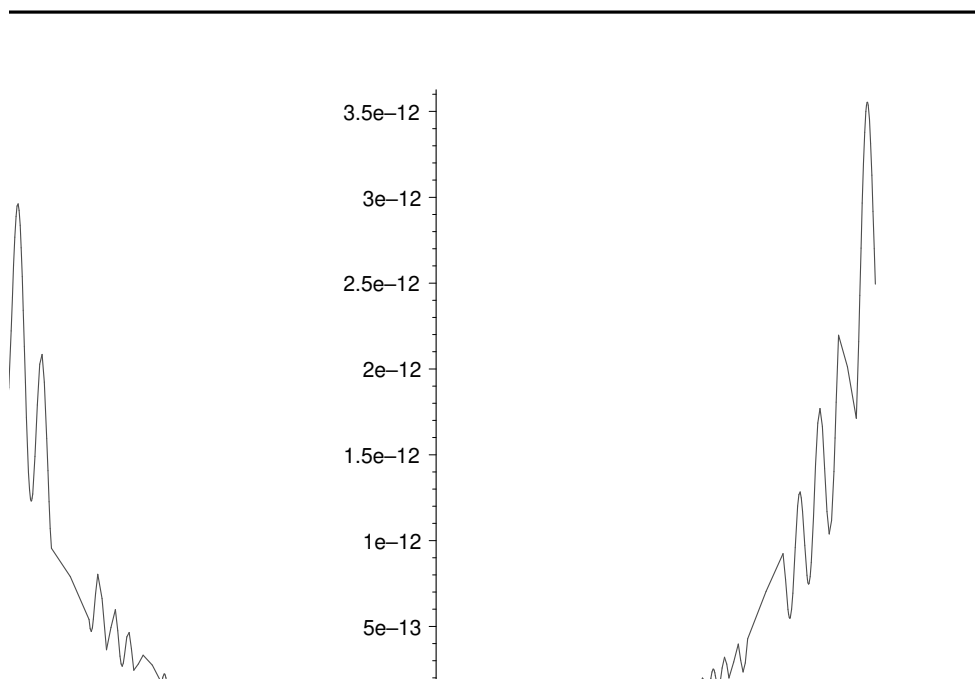
```
0.0018
```

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 hossza 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 meglehetősen 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 közelnek 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 eljárás. 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 iterációs számot 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 pontosságára 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 nagyon 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 keveselte 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` robusztus 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 alapproblémája 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 másodikonál 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 konstansszoros a másikkal. 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:

$$Q' = \begin{pmatrix} 1.0000 & w_2/w_1 & w_3/w_1 \\ w_1/w_2 & 1.0000 & w_3/w_2 \\ w_1/w_3 & w_2/w_3 & 1.0000 \end{pmatrix}.$$

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,3)-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 visszaírjuk 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 =
```

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 erőltettük, 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 négyest 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:

perturbáció	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 alapsomagban kínált optimalizálási eljárások használhatók, 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. Higham: 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). TypoT_EX Kiadó, Budapest, 1999.

F. függelék

Az előadás fóliái

AZ OPTIMALIZÁLÁS ALKALMAZÁSAI.

Csendes Tibor

www.inf.u-szeged.hu/~csendes, csendes@inf.u-szeged.hu

Ez az új tantárgy az elődje, az Operációkutatás II és a Kombinatorikus optimalizálás nyomán az optimalizálási modellek használatába és számítógépes megvalósításukba ad bevezetést. A kredit követelményei:

- Az előadásra járás ugyan nem kötelező, de ajánlott, és a katalógus alapján azok, akik csak kevés alkalommal hiányoznak, plusz pontot kapnak
- Ugyancsak plusz pontot lehet kapni az előadáson az elhangzott anyagra vonatkozó egyszerűbb kérdések megválaszolásáért.
- A gyakorlat keretében az első félévben egy programot kell önállóan megírni és az erre vonatkozó esszét leadni határidőre (április 15., a vázlaté március 15.). A MATLAB rendszer elérhető a hallgatói kabinet gépein.
- A gyakorlaton írt több kisdolgozat összevont pontszáma el kell hogy érje a maximális pontszám 50%-át, csakúgy, mint a félévvégi tudásfelmérő dolgozaté és az esszéé.
- Mindenkinek kell szóbeli vizsgát tennie, aminek része egy 40 pontos dolgozat megírása is.
- Az Intézet döntése szerint az is kell hogy hallgassa az Optimalizálás Alkalmazásai című tárgyat, aki az Operációkutatás II. tárgyat sikeresen befejezte.

MOTIVÁCIÓ

A jelenlévők egy jó része a végzés után optimalizálási feladatokat fog megoldani rutinszerűen, de a modellezésben való jártasság is hasznos több munkakörben. Egy korábbi amerikai felmérés szerint az operációkutatási végzettségű munkába állókat a legkeresettebbek listáján előkelő helyen szerepeltették (lásd OR Today).

H.-P. Schwefel példája: A Siemens megbízásából atomerőművek számára kellett a fűtőelemek helyét meghatározni úgy, hogy javuljon a hatékonyság. Az evolúciós módszerrel talált megoldás több mint 1%-al volt jobb, mint a korábban ismert. Bár nem tudja, hogy a talált közelítés akár csak helyileg optimális-e, és azt sem, hogy milyen messze van az optimumtól, a megbízó elégedett volt ($> 10^8$ DEM).

*SIAM 100 \$, 100 Digits Challenge 2002, #4*¹

A feladat a következő függvény minimumának meghatározása volt:

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

A kapott eredmény a $[-10.0, 10.0]$ keresési tartományon a globális minimum értékére a következő alsó- és felső korlátokat adta:

$$[-3.306868647475316, -3.306868647475196].$$

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.

¹Nick Trefethen: A Hundred-Dollar, Hundred-digit Challenge. SIAM News 35(2002)

AZ ESSZÉ

Az esszé egy olyan rövid (15-20 oldalas) jelentés, amelynek a következő főbb részeket kell tartalmaznia:

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

2. A megoldására megírt, felhasznált, lehetőleg Matlab (vagy Scilab, Netlib, esetleg Octave) programok részletes megadása, leírása. Ki kell térni az alkalmazott számítógépes megoldások okaira, előnyeire is (mint pl. a választott adattípus, számformátum, algoritmus változat, stb.).

3. A bemutatott algoritmusok hatékonyságát, sebességét, műveletigényét, pontosságát és egyéb említésre méltó tulajdonságát alkalmas teszteléssel kell bemutatni. 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 (nyomtatott vagy elektronikus formájú).

- Nyomtatott vagy elektronikus formában kell a gyakorlatvezetőnek beadni,
- a használt szövegszerkesztő lehetőleg $\text{\LaTeX}$ alapú, vagy Word legyen,
- érdemes támaszkodni a hálózaton keresztül elérhető előzetes jegyzet irodalomjegyzékére, az interneten elérhető adatokra, és az évfolyamtársakra,
- korlátozott mértékben a gyakorlatvezetővel való konzultáció is segíthet,
- *önálló munkát* kell tükröznie, és a hallgatónak a kapott feladat megoldásának minden részletével tisztában kell lennie,
- az esszé értékét növeli, ha az a Scilab, Netlib vagy Octave programjait tárgyalja,
- a hallgató javasolhat is esszé formában feldolgozandó témát.

A KÖTELEZŐ PROGRAM

A kötelező program olyan házfeladat, amelyet Java nyelven kell írni, és a célja az, hogy a hallgatók bemutassák jártasságukat optimalizálási eljárások implementálásában, és az érintett algoritmusok működésének áttekinthető, könnyen érthető formában való megjelenítésében.

Az esszével szemben a kötelező program dokumentálása más szempontokra helyezi a hangsúlyt:

1. meg kell adni az algoritmus folyamatábráját vagy pszeudokódját,
2. a program teljes forrásszövegét, de legyen elérhető a program a hálózaton, és telepíthető is más gépekre,
3. meg kell adni a felhasználói útmutatót, amely alapján a program teljesértékűen használható, és végül
4. részletesen leírni egy jellemző futtatási példát, amely az algoritmus használhatóságát mutatja.

Tehát nem kell: az adott feladatkört ismertetni, részletesen megadni a kapcsolódó szakirodalmat, nem szükséges a megírt program hatékonyságát és hatásosságát teszteléssel igazolni.

Mintaként érdemes megnézni a

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

vendégoldalon a körpakolási illusztráló Java programot.

Fontos az önálló munka. Az értékelés alapja az lesz, hogy milyen látványos, ügyes illusztrációt ad a program az adott algoritmushoz.

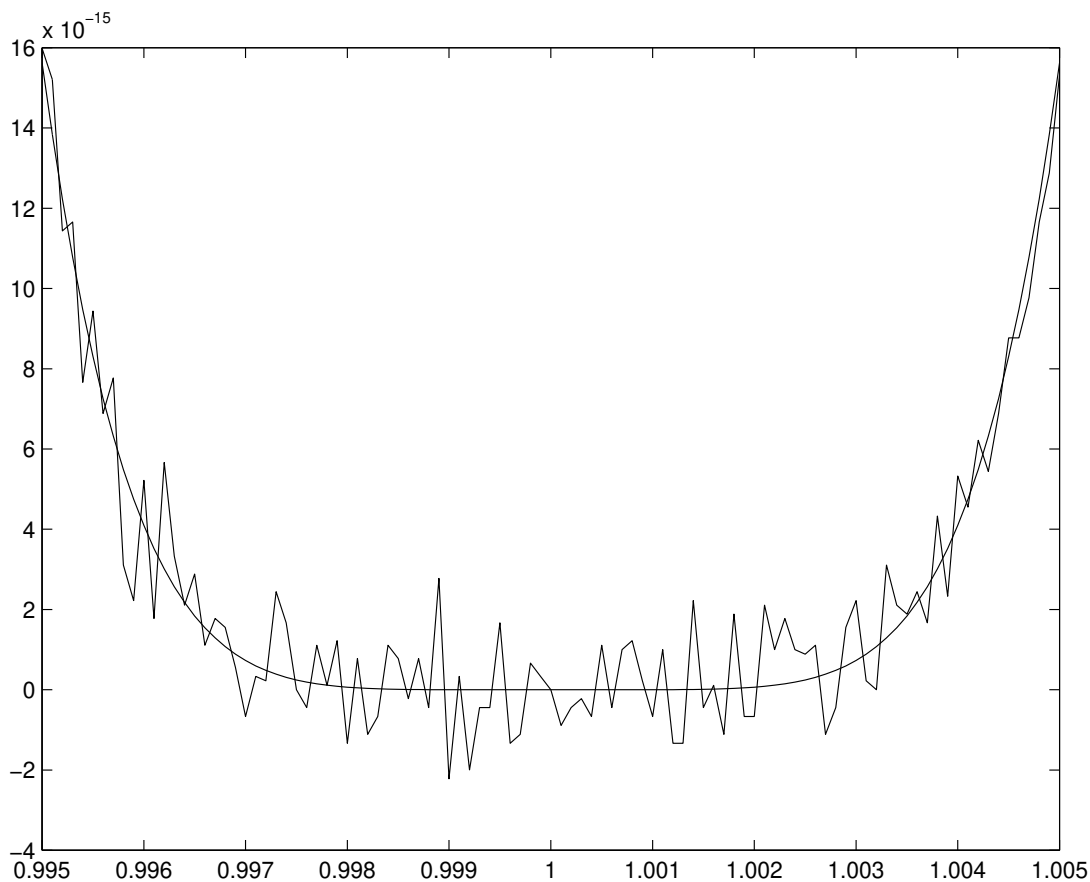
MATLAB MINTA

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 $(1 - x^6)$):

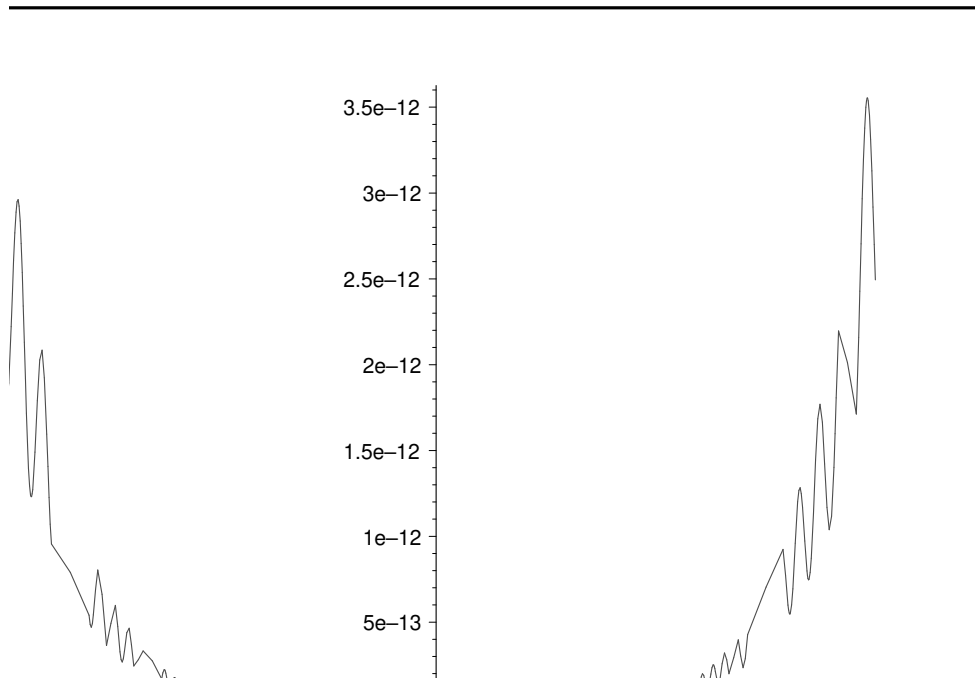


MAPLE MINTA

```
> f(x):=x^6*(sin(1/x)+3);
```

$$f(x) := x^6 * (\sin(1/x) + 3)$$

```
> plot(f(x), x=-0.01..0.01);
```



```
> g(x):=diff(f(x), x);
```

$$g(x) := 6 * x^5 * (\sin(1/x) + 3) - x^4 * \cos(1/x)$$

```
> solve(g(x), x);
```

0,0,0,0

AJÁNLOTT IRODALOM

Az előadás anyagát a készülő jegyzet tartalmazza majd. Ezt kiegészíthetik az ezután fontossági sorrendben megadott könyvek:

1. Bajalinov Erik és Imreh Balázs: Operációkutatás. Polygon Jegyzettár, Szeged, 2001.
2. Imreh Balázs: Kombinatorikus optimalizálás. Novadat Kiadó, Győr.
3. Hillier, F.S., G.J. Lieberman: Bevezetés az operációkutatásba. LSI Oktatóközpont, Budapest, 1994.
4. Winston, W.L.: Operációkutatás I-II. Módszerek és alkalmazások. Aula Kiadó, Budapest, 2003.
5. Tóth Irén (szerk.): Operációkutatás I., Nemzeti Tankönyvkiadó, Budapest, 1999.
6. Csernyák László (szerk.): Operációkutatás II., Nemzeti Tankönyvkiadó, Budapest, 1999.
7. Raffai Mária (szerk.): Döntéselőkészítés – Operációkutatási Módszerek. Novadat Kiadó, Győr, 2000.
8. Kósa András: Optimumszámítási modellek. Műszaki Könyvkiadó, Budapest, 1979.
9. Gáspár László és Temesi József: Matematikai programozási gyakorlatok. Nemzeti Tankönyvkiadó, Budapest, 1999.
10. Csallner András Erik: Intervallum-felosztási eljárások a globális optimalizálásban. PhD disszertáció, Szeged, 1999. Elérhető a <http://www.jgytf.u-szeged.hu/~csallner> internetes címen.
11. Komlósi Sándor: Az optimalizáláselmélet alapjai. Dialóg Campus Kiadó, Budapest, Pécs, 2001.
12. GNU OCTAVE vendégoldal: www.che.wisc.edu/octave
13. MATLAB online kézikönyvek: www.mathworks.com/access/helpdesk/help/fulldocset.shtml
14. NETLIB vendégoldal: www.netlib.org
15. SCILAB vendégoldal: www-rocq.inria.fr/scilab/scilab.html

A HÁTIZSÁK FELADAT

Adottak szállítandó tárgyak súllyal (vagy térfogattal) és értékkel (vagy fontossággal). A feladat az, hogy meghatározzuk a hátizsákba beteendő holmiknak azt a részhalmazát, amelyek az előző értelemben a leghasznosabbak, és együtt beférnek a korlátozott kapacitású hátizsákba.

Használjuk a következő jelölést:

- m a tárgyak száma,
- a_i az i . tárgy súlya, $i = 1, 2, \dots, m$,
- c_i az i . tárgy értéke, $i = 1, 2, \dots, m$,
- b a rakomány megengedett maximális összsúlya.

Legyen x_i értéke 1, ha az i -edik tárgy bekerült a hátizsákba, és 0, ha nem ($i = 1, 2, \dots, m$).

A megoldandó feladat ezekkel felírva:

$$\max \sum_{i=1}^m c_i x_i,$$

feltéve, hogy

$$\sum_{i=1}^m a_i x_i \leq b, \text{ és}$$

$$x_i \in \{0, 1\}, \quad i = 1, 2, \dots, m.$$

A hátizsák feladat tehát egy egészértékű, bináris lineáris programozási feladat. Egy egyszerű kiterjesztése adódik akkor, ha az elhelyezendő tárgyak között vannak azonosak. Ekkor az optimalizálandó változók értékei nemnegatív egészek lehetnek.

A feladat jellege miatt a kiindulási feladatra legtöbbször fel lehet tenni azt, hogy a súlyok és a súlyhatár nemnegatívak. Ennek ellenére mind az a_i , mind a c_i értékek előjele tetszőleges.

A HÁTIZSÁK FELADAT MEGOLDÁSA TELJES LESZÁMOLÁSSAL

A hátizsák feladatot megoldhatjuk a durva erő módszerével (brute force, enumeration). Ennek lényege, hogy felsoroljuk az összes változó-kombinációt, meghatározzuk a lehetséges megoldásokra a célfüggvény értékét, és kiválasztjuk ez alapján az optimálisat.

Az összes változó-kombinációt például lexikografikus sorrendben határozhatjuk meg. Négy bináris változó esetén az így kapott sorozat:

$$(0, 0, 0, 0), (0, 0, 0, 1), \dots, (1, 1, 1, 1).$$

Az eljárás hátránya, hogy nagyszámú változó esetén kezelhetetlenül sok esetre kell ellenőrizni a feltételt, és ez a szám a változók számával exponenciálisan nő.

PÉLDA. Tekintsük a következő feladatot. Legyen a tárgyak súlya rendre 2, 3 és 4, a hasznossága pedig 5, 3, 1, míg a megengedett összsúly 5. A megoldást tartalmazó táblázat (a lehetséges megoldásokat félkövér betű, az optimálisat csillag jelzi):

javasolt megoldás	a súlya	az értéke
(0,0,0)	0	0
(0,0,1)	4	1
(0,1,0)	3	3
(0,1,1)	7	4
(1,0,0)	2	5
(1,0,1)	6	6
(1,1,0)*	5*	8*
(1,1,1)	9	9

Vegyük észre, hogy a megoldást a mohó módszerrel is megkapnánk: addig vennénk a legértékesebb tárgyakat csökkenő hasznossági sorrendben, amíg azt a súlyhatár megengedi.

A feladatnak megfeleltethető a következő, többé-kevésbé reális probléma: adott egy komp, ennek teherbírása 5 tonna. Három jármű vár az átvitelre: egy 2 tonnás személyautó, egy 3 tonnás kisteherautó, és egy 4 tonnás szekér. A viteldíjak rendre 500, 300, illetve 100 forint. Üzleti okokból kíváncsiak vagyunk az előírásoknak megfelelő, legnagyobb bevételt jelentő fuvarra.

A HÁTIZSÁK FELADAT MEGOLDÁSA KÖZVETETT, IMPLICIT LESZÁMOLÁSSAL

A megoldási módszer lényege, hogy a teljes leszámolást ésszerűen gyorsítjuk a feltétel és az értékek figyelembe vételével. Többek között azokat a kiértékeléseket tudjuk megtakarítani, amelyek egyértelműen nem lehetséges megoldásokhoz tartoznak. Például, ha kiderült, hogy a $(0, 1, 0, \dots, 1, 0, 0)$ megsérti a súlykorlátot, akkor nincs értelme a több egyest tartalmazó $(0, 1, 0, \dots, 1, 0, 1)$ ellenőrzésének – ha a megfelelő súlyok pozitívok.

1. Első lépésként alakítsuk át a feladatunkat egy könnyebben áttekinthető, kanonikus alakúra: legyen minden célfüggvény együttható nem pozitív, és a súlyok növekvő sorrendbe rendezettek: $a_1 \leq a_2 \leq \dots \leq a_m$.

Az első feltételt az $x'_i = 1 - x_i$ helyettesítéssel lehet elérni olyan változókra, amelyekre az nem teljesült. Ezeket a változókat természetesen meg kell jegyezni, és az eljárás végén a kapott optimális értékeket megfelelően vissza kell alakítani. A második tulajdonság teljesüléséhez a változókat kell csak alkalmasan átrendezni (és a megoldás után vissza).

2. Az x indulóvektornak válasszuk a nulla vektort. A keresés során ettől haladunk a keresési fa levelei felé, amelyek utolsó komponense egyes.

3. Ha $\sum_{i=1}^m a_i x_i \leq b$ teljesül, akkor adjuk x -et a lehetséges megoldások L halmazához. Hagyjuk ki az ellenőrizendő csúcsok közül azokat, amelyekre a jelen x utolsó nullái egyikének c_i együtthatója negatív.

Ha ez az egyenlőtlenség nem igaz, akkor ugorjuk át mindazokat a vektorokat a keresésben, amelyek előállnak úgy, hogy x utolsó nulla elemei egyike helyett egyes szerepel, és a megfelelő súly pozitív.

4. Generáljunk egy újabb vektort! Ha van még ilyen, akkor folytassuk a 3. Lépéssel. Az új vektor generálása során ha lehet, akkor olyant választunk, amely az előző x vektor utolsó nulla jegyeit módosítva kapható. Ha ez már nem lehetséges, akkor visszatérünk egy korábban félbehagyott ághoz a keresési fában.

5. A legnagyobb célfüggvény értékű talált lehetséges megoldás az optimális megoldás az x^* pontban. Az 1. Lépésben végrehajtott átalakításoknak megfelelően az eredményt visszatranszformáljuk.

A HÁTIZSÁK FELADAT MEGOLDÁSA KÖZVETETT, IMPLICIT LESZÁMOLÁSSAL – PÉLDA

A korábban vizsgált példát most nem érdemes használni, mert könnyen látható, hogy abban nincs lehetőség a keresés gyorsítására, mivel a nem lehetséges megoldások mind a keresési fa levelein vannak.

PÉLDA. Tekintsük ezért a következő, kicsit módosított feladatot. Legyen a tárgyak súlya rendre 4, 3 és 3, a hasznossága pedig -5, 3, -1, míg a megengedett összsúly 2.

Kövessük a közvetett leszámolási algoritmus lépéseit.

1. Írjuk fel az eredeti feladatot:

$$\max \sum_{i=1}^3 c_i y_i = \max -5y_1 + 3y_2 - y_3,$$

a feltétel pedig

$$\sum_{i=1}^3 a_i y_i \leq b, \text{ azaz } 4y_1 + 3y_2 + 3y_3 \leq 2.$$

A kanonikus alakot úgy kapjuk, hogy az y_2 változót $1 - x_1$ -el helyettesítjük (és c_2 új értéke -3 lesz). Ezzel párhuzamosan a súlyok növekvő sorrendje eléréséhez cseréljük fel a változókat:

$$x_1 = 1 - y_2, \quad x_2 = y_3, \quad x_3 = y_1.$$

Az új feladat ezután:

$$\begin{aligned} \max & -3x_1 - x_2 - 5x_3 + 3, \\ & -3x_1 + 3x_2 + 4x_3 \leq -1. \end{aligned}$$

2. Az indulóvektor $x^T = (0, 0, 0)$.

3. A $(0,0,0)$ vektorra a fenti feltétel nyilvánvalóan nem teljesül, ezért a további keresésből kizárjuk a $(0,1,0)$, $(0,1,1)$ és $(0,0,1)$ eseteket is, mivel az utolsó két súly pozitív, tehát ezekre az esetekre sem teljesülhet a feltételünk.

A HÁTIZSÁK FELADAT MEGOLDÁSA KÖZVETETT, IMPLICIT LESZÁMOLÁSSAL –
PÉLDA II.

$$\begin{aligned} \max & -3x_1 - x_2 - 5x_3 + 3, \\ & -3x_1 + 3x_2 + 4x_3 \leq -1. \end{aligned}$$

4. A következő keresési vektor $x^T = (1, 0, 0)$.

3. Az $(1, 0, 0)$ vektor súlya $-3 \leq -1$, tehát $L = \{(1, 0, 0)\}$. Az ehhez a vektorhoz tartozó célfüggvényérték 0. Mivel az $(1, 0, 0)$ vektor hátsó nullái mindegyikéhez negatív célfüggvény-együttható tartozik, ezért más vektort már nem is kell megvizsgálni: a további lehetséges megoldások célfüggvényértéke kisebb lenne a megtaláltnál.

5. Az átalakított feladat optimális megoldása tehát $(1, 0, 0)$. Az 1. Lépés átalakítása alapján az eredeti feladat optimális megoldása

$$\begin{aligned} y_1 &= x_3 = 0, \\ y_2 &= 1 - x_1 = 0, \\ y_3 &= x_2 = 0. \end{aligned}$$

Az ehhez tartozó célfüggvényérték pedig természetesen 0.

Az eredmény szépen értelmezhető az eredeti

$$\begin{aligned} \max & -5y_1 + 3y_2 - y_3, \\ & 4y_1 + 3y_2 + 3y_3 \leq 2 \end{aligned}$$

feladatra. Eszerint a célfüggvény optimuma a teljes, feltétellel nem korlátozott tartományon a $(0, 1, 0)$ pontban lenne. Ez azonban nem lehetséges megoldás, mert a második komponens miatt az előírt feltétel nem teljesül. Az egyenlőtlenséget az $y_2 = 0$ választással teljesíthetjük, ez pedig épp a kapott megoldást adja. Mivel a célfüggvény együtthatók és a súlyok egyike sem nulla, ezért más megoldás nincs is.

A HAJÓRAKODÁSI FELADAT

Korlátozott térfogatú szállítóeszköz esetén a hátizsák feladat feltételéhez a térfogatra vonatkozót is meg kell adni. Kövessük a korábbi jelölést:

- m a tárgyak száma,
- a_{1i} az i . tárgy súlya, $i = 1, 2, \dots, m$,
- a_{2i} az i . tárgy térfogata, $i = 1, 2, \dots, m$,
- d_i az i . tárgyból rendelkezésre álló mennyiség,
- c_i az i . tárgy értéke, $i = 1, 2, \dots, m$,
- b_1 a rakomány megengedett maximális összsúlya,
- b_2 a rakomány megengedett maximális össztérfogata.

Legyen x_i értéke ismét 1, ha az i -edik tárgy bekerült a rakományba, és 0, ha nem ($i = 1, 2, \dots, m$). Ha több i -edik tárgy is van a rakományban, akkor x_i értéke legyen a megfelelő egész szám.

A megoldandó feladat ezekkel felírva:

$$\max \sum_{i=1}^m c_i x_i,$$

feltéve, hogy

$$\sum_{i=1}^m a_{ji} x_i \leq b_j, \quad j = 1, 2,$$

$$x_i \leq d_i, \text{ és}$$

$$x_i \text{ egész, } i = 1, 2, \dots, m.$$

Állapítsuk meg, hogy a hajórakodási feladat is megoldható a korábban ismerttetett implicit leszámolással, de módosítást kell rajta végrehajtani. Bár elvileg a két feltétel egymástól függetlenül is érvényesíthető az ellenőrizendő vektorok számának csökkentésére, de a súlyok és a térfogatok értékei egyszerre nem feltétlenül rendezhetők növekvő sorrendbe. Ennek ellenére a kapott eljárás általában hatékonyabb lesz, mint a teljes leszámolás.

Vegyük észre, hogy hasonló módon további feltételek is érvényesíthetők, pl. a rakomány összhosszára stb.

A FIX KÖLTSÉG FELADAT

A termelő, szolgáltató vállalkozások a költségeik csökkentésére törekszenek. Tekintsük azt a feladatot, amely a fix és termeléssel arányos költségek viszonyát vizsgálja.

Használjuk a következő jelölést:

m	a termékek száma,
x_i	az i . termék gyártandó mennyisége,
d_i	az i . termék gyártási korlátja,
a_{ij}	a j . termék egységnyi mennyiségének gyártásához az i . erőforrásból felhasznált mennyiség,
b_i	az i . erőforrásból rendelkezésre álló mennyiség,
c_i	az i . termék fix költsége (vagy nulla),
$k_i(x_i)$	az i . termék mennyiségtől függő termelési költsége.

A matematikai modell ezek alapján:

$$\min \sum_{i=1}^m f_i(x_i) = \min \sum_{i=1}^m c_i + \sum_{i=1}^m k_i(x_i)$$

feltéve, hogy

$$0 \leq x_i \leq d_i, \text{ és}$$

$$Ax \leq b.$$

A feladat nemlineáris $k_i(x)$ esetén szeparábilis, lineárisan korlátozott nemlineáris optimalizálási feladat. Amennyiben a gyártandó termékek mennyisége egész, akkor ráadásul egészértékű (vagy vegyes egészértékű) nemlineáris optimalizálási problémával állunk szemben.

Gyakori eset, hogy a nem fix beruházási függvényrész $x_i^{0.6}$, vagy hasonló alakú, és így feladatunk a konkáv optimalizálás területére tartozik.

ELLENŐRZŐ KÉRDÉSEK ÉS GYAKORLÓ FELADATOK AZ EDDIGI FELADATOKHOZ

1. Mutasson olyan értelmes gyakorlati feladatot, amely olyan hátizsák feladatra vezet, amelyben a célfüggvény együttthatók mind negatívak!
2. Hogyan fogalmazná át a teljes leszámolás módszerét a hátizsák feladat azon esetére, amikor több azonos tárgyat kell elhelyezni?
3. Mutasson példát, amelyre az implicit leszámolási eljárás az első lehetséges megoldással meg is találja az optimális megoldást!
4. Mutasson példát, amelyre az implicit leszámolási eljárás is minden szóbjövő vektort meg kell hogy vizsgáljon ahhoz, hogy megtalálja az optimális megoldást!
5. Keressen gyakorlati példát arra az esetre, amikor a hátizsák feladat célfüggvény együttthatói között vannak pozitív és negatív előjelűek is!
6. Mi adja a lényegi eltérést a hátizsák- és a hajórakodási feladat között?
7. Ha egy 2 Ghz-es PC 10 órajel alatt tudja egy vektorról eldönteni, hogy az lehetséges megoldása-e a hátizsák feladatnak, akkor egy nap alatt milyen méretű feladat megoldására lehet biztosan számítani (tehát a legrosszabb esetben)?
8. Keressen reális alkalmazási feladatot, amely olyan hátizsák feladatra vezet, amelyben negatív és pozitív súlyok is kellenek!
9. Indokolja, hogy a fix költség feladat miért nem vezethető vissza közvetlenül a hátizsák feladatra!
10. Adjon meg egy olyan hátizsák feladat osztályt, amely minden elemére nem negatív az optimális célfüggvény érték!
11. Mit lehet mondani annak a hátizsák feladatnak a megoldásairól, amelyben minden súly és minden célfüggvény együtttható is megegyezik?
12. Jellemezze a hajórakodási feladatoknak azt a részhalmazát, amely megfeleltethető a hátizsák feladatnak!

ELLENŐRZŐ KÉRDÉSEK ÉS GYAKORLÓ FELADATOK AZ EDDIGI FELADATOKHOZ II

1. Tekintsük a $\max x_1 + 2x_2, 2x_1 + x_2 \leq 2$ hátizsák feladatot. Milyen mértékben lehet megváltoztatni a súlykorlátot ahhoz, hogy a megoldás ne változzon? Mi a helyzet a feladat többi paraméterével?
2. Tegyük fel, hogy egy hátizsák feladatban a legnagyobb célfüggvényértékhez tartozó tárgyak összszúlya a megadott korlát alatti. Mit mondhatunk ekkor az optimális megoldásról?
3. Ha a hátizsák feladatban megadott legfontosabb tárgyak összszúlya épp a súlyhatárt adja, akkor mi az optimális megoldás?
4. Mutasson olyan hátizsák feladatot, amelynek optimális megoldásában a legkisebb értékű tárgy is benne van!
5. Igaz az, hogy bármely bináris vektorhoz konstruálható olyan hátizsák feladat, amelynek ez egyetlen optimális megoldása? És olyan, amelynek csak ez nem optimális megoldása?
6. Mi a hátránya a Monte Carlo módszernek (egyenletes eloszlással generálunk vektorokat, és a talált legjobb célfüggvényértékű vektort megjegyezzük) a hátizsák feladat megoldása során?
7. Oldja meg fejben a $\max x_1 + 2x_2, 2x_1 + x_2 \leq 2$ hátizsák feladatot! Melyik módszert használta?
8. Érveljen a teljes leszámolás módszere mellett: mikor előnyösebb az, mint az implicit leszámolás?
9. Mennyi a műveletigénye az implicit leszámolás első, előkészítő lépésének?
10. Mi a megoldása annak a hátizsák feladatosztálynak, amelyben minden súly, érték és súlyhatár megegyezik?
11. Miért nincs értelme a hátizsák feladatnak $m = 1$ esetén?
12. Mutassa meg, hogy a fix költség feladat speciális eseteként előáll a hátizsák feladat!

AZ UTAZÓ ÜGYNÖK FELADAT

Az operációkutatás egy nevezetes, központi feladata az utazó ügynök problémája. Legyenek adottak meglátogatandó városok, ismerjük a köztük lévő távolságokat. A feladat egy olyan minimális hosszúságú útvonal megtalálása, amely minden várost érint, mindegyiken csak egyszer halad át, és a körút végén visszatér a kiindulási városba.

A matematikai modell megfogalmazásában legyen a korábbiaknak megfelelően a meghatározandó változók halmaza $x_{ij} \in \{0, 1\}$, $i, j = 1, \dots, n$, ahol n a városok száma, x_{ij} pedig azt adja meg, hogy az i . és a j . város között áthalad-e az aktuális körút. A feladatot a következők szerint fogalmazhatjuk meg:

$$\min \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij},$$

feltéve, hogy

$$\sum_{t=1}^n x_{it} = 1 \quad (i = 1, \dots, n),$$

$$\sum_{t=1}^n x_{tj} = 1 \quad (j = 1, \dots, n),$$

$$\sum_{i \in Q} \sum_{j \in \{1, \dots, n\} \setminus Q} x_{ij} \geq 1 \quad Q \subset \{1, \dots, n\}, \quad Q \neq \emptyset,$$

$$x_{ij} \in \{0, 1\}, \quad i, j = 1, \dots, n.$$

A célfüggvény a megtett útszakaszok költségét összegzi. Az első feltétel azt követeli meg, hogy az ügynök minden városból kimegy, a második pedig azt, hogy mindegyikbe bejut, mindkét esetben pontosan egyszer. E két feltétel teljesülése esetén még előfordulhat, hogy a kapott útvonal különálló körutakból áll, ami a feladat eredeti megfogalmazásának nem felel meg.

Ezt a problémát rendezzi a következő feltétel. Ha lenne olyan zárt körút, amely nem tartalmazza az összes várost, akkor az ehhez tartozó városok alkotta Q halmazra ez a feltétel nem teljesülne, hiszen ekkor a baloldali összeg nullának adódna.

Állapítsuk meg, hogy a megfogalmazott operációkutatási feladat egy lineáris egyenlőség és egyenlőtlenség korlátokkal ellátott nulla-egy lineáris programozási feladat.

AZ UTAZÓ ÜGYNÖK FELADAT A GYAKORLATBAN

Az utazó ügynök feladat számos alkalmazásban fordul elő kisebb-nagyobb módosítással.

Az egyik legkézenfekvőbb a tömegközlekedés *járatütemezési problémája*. Adjuk meg azt az útvonalat, amelyet egy busznak meg kell tennie ahhoz, hogy bizonyos járatok útvonalán a megfelelő szolgáltatást nyújtsa, ehhez a lehető legrövidebb útvonalat keressük, és a járatok teljesítése után térjen vissza az indulási állomására.

Hasonló eset a fémmegmunkálásban a *szerszámgépek olyan vezérlése*, hogy a befogott munkadarab lehető legkisebb mozgatása révén minden részművelet helyét érintse a fűrő, szegecselő stb. fej, majd térjen vissza a kiindulási helyzetébe.

Tekintsük azt a problémát, amikor a feladat egy gyár által termelt *termékek sorrendjének meghatározása* – tekintettel arra, hogy az egyik termék gyártásáról egy másiknak a termelésére való átállásnak időben, vagy direkt költségben eltérő ára van. Természetesen minden terméket le kell gyártani, és a teljes termelési ciklus után ugyanabba a helyzetbe tér vissza a gyártási sorrend.

Bonyolultabb a helyzet, ha *repülőgépek és azok személyzete útitervét* kell optimálisan meghatározni úgy, hogy megadott városokat érintsenek, a hatékonyság a lehető legjobb legyen, de a szervizelési, pihenési stb. előírásokat betartsák.

Közvetve idetartozik az *írógépek, számítógépes billentyűzetek tervezése* is. Ekkor adott nyelvi környezetben megméri, hogy két betű egymás utáni előfordulása milyen gyakori. Ennek megfelelően a cél olyan billentyűzetet megadni, amelyre a tipikus szövegek gépelése során a lehető legkevesebbet kell mozgatni a kezünket.

Utazó ügynök feladatot old meg a *betegszállító diszpécser* is, amikor meghatározza azt, hogy a mentő az aznapi betegeket milyen sorrendben vegye fel a lakásukban, szállítsa a dialízis kezelésre, majd vissza otthonukba. Ebben a feladatban a kórháznak kiemelt helye van, nyilván nem lehet azt utolsóként érinteni...

Az utazó ügynök feladata interpretálható úgy is, hogy minimális hosszúságú, minden csúcsot érintő irányított körutat kell meghatározni egy adott gráfban.

AZ UTAZÓ ÜGYNÖK FELADAT RÖVIDEBB ALAKJA

Az utazó ügynök feladat korábban ismertetett modellje 2^n darab feletti feltételt tartalmazott, ez valódi feladatok esetén elviselhetetlenül nagy szám. A. Tucker 1960-ban kevesebb feltétellel fogalmazta újra a feladatot:

$$\min \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij},$$

feltéve, hogy

$$\sum_{t=1}^n x_{it} = 1 \quad (i = 1, \dots, n),$$

$$\sum_{t=1}^n x_{tj} = 1 \quad (j = 1, \dots, n),$$

$$u_i - u_j + (n-1)x_{ij} \leq n-2 \quad 2 \leq i \neq j \leq n,$$

$$x_{ii} = 0, \quad x_{ij} \in \{0, 1\}, \quad i, j = 1, \dots, n,$$

$$u_i \geq 0, \quad u_i \text{ egész}, \quad i = 2, \dots, n.$$

A rész kör mentességet a 3., új feltétel hivatott biztosítani. Az új feladatnak n^2 nagyságrendű feltétele van. A konstrukció lényege, hogy az u_i számokkal alkalmasan sorszámozott körút elemekre a 3. feltétel csak akkor teljesülhet, ha az teljes körút (vö. a bizonyítás vége a következő oldalon).

ÁLLÍTÁS. Az utazó ügynök feladat két modellje ekvivalens.

BIZONYÍTÁS. A két feladat célfüggvénye megegyezik, tehát a lehetséges megoldások halmazának megegyezését kell igazolni.

AZ UTAZÓ ÜGYNÖK FELADAT ISMERTETETT KÉT ALAKJA EKVIVALENCIÁJA

Tekintsük először azt az esetet, hogy az eredeti feladatnak az X mátrix egy lehetséges megoldása:

$$x_{1,i_2} = x_{i_2,i_3} = \cdots = x_{i_n,1} = 1, \text{ és} \\ x_{ij} = 0 \text{ különben.}$$

Ehhez megkonstruáljuk azt az (X, u) párt, amely lehetséges megoldása lesz a második feladatnak. Mivel X lehetséges megoldása az első feladatnak, ezért ehhez tartozik egy körút, amely az $(1, i_2), (i_2, i_3), \dots, (i_n, 1)$ éleket tartalmazza. Definiáljuk most u értékét a következők szerint:

$$u_{i_t} = t, \quad t = 2, \dots, n.$$

Csak a harmadik feltételrendszert kell igazolni, a többi teljesülése nyilvánvaló. Tekintsünk egy tetszőleges ide való indexpárt: $2 \leq i \neq j \leq n$. Az u definíciójából adódik, hogy $u_i - u_j \leq n - 2$. Másrészt $x_{ij} = 1$ pontosan akkor teljesül, ha i és j a körútban közvetlenül egymás utáni indexek: ha $i = i_r$, akkor $j = i_{r+1}$. Innen erre az esetre

$$u_i - u_j + (n - 1)x_{ij} = r - (r + 1) + (n - 1) = n - 2,$$

tehát a harmadik csoport feltétel is teljesül, így X lehetséges megoldása az első feladatnak.

Tekintsük most azt az esetet, amikor a második feladatnak az (X, u) pár egy lehetséges megoldása, de van egy diszjunkt részkörút, tehát $x_{i_1,i_2} = x_{i_2,i_3} = \cdots = x_{i_k,i_1} = 1$, és $1 < k < n$. Az általánosság megszorítása nélkül feltehetjük, hogy $1 \notin \{i_1, i_2, \dots, i_k\}$. Mivel (X, u) egy lehetséges megoldása a második feladatnak, ezért

$$\begin{aligned} u_{i_1} - u_{i_2} + (n - 1)x_{i_1,i_2} &\leq n - 2, \\ u_{i_2} - u_{i_3} + (n - 1)x_{i_2,i_3} &\leq n - 2, \\ &\vdots \\ u_{i_k} - u_{i_1} + (n - 1)x_{i_k,i_1} &\leq n - 2 \end{aligned}$$

teljesül. Az egyenlőtlenségeket összeadva azt kapjuk, hogy $k(n - 1) \leq k(n - 2)$, ami $1 < k < n$ esetén ellentmondás. $\square$

AZ UTAZÓ ÜGYNÖK FELADATOK EKVIVALENCIÁJA

Mivel az utazó ügynök feladat feltételrendszere csak az n számtól függ, ezért a feladatot egyértelműen meghatározza az n szám, és a C költségmátrix.

DEFINÍCIÓ. Azt mondjuk, hogy a C mátrix *ekvivalens* a D mátrixszal (jelölése: $C \sim D$), ha vannak olyan $\alpha_1, \alpha_2, \dots, \alpha_n$ és $\beta_1, \beta_2, \dots, \beta_n$ számok, hogy igaz $c_{ij} = d_{ij} + \alpha_i + \beta_j$ minden indexpárra.

SEGÉDTÉTEL. Ha $C \sim D$, akkor a C mátrix által meghatározott $TSP(C)$ utazó ügynök feladat optimális megoldása megegyezik a $TSP(D)$ feladatával.

BIZONYÍTÁS. Mivel mindkét feladatnak létezik optimális megoldása, ezért a segédtelet állítása korrekt. A két feladat lehetséges megoldásai halmaza megegyezik, legyen ez L , a két célfüggvény pedig z_C és z_D .

Azt fogjuk megmutatni, hogy létezik olyan γ konstans, hogy $z_C(x) = z_D(x) + \gamma$ teljesül minden x lehetséges megoldásra. Mivel $C \sim D$, ezért vannak olyan $\alpha_1, \alpha_2, \dots, \alpha_n$ és $\beta_1, \beta_2, \dots, \beta_n$ konstansok, hogy $c_{ij} = d_{ij} + \alpha_i + \beta_j$ minden $1 \leq i, j \leq n$ indexpárra.

Ekkor

$$\begin{aligned} z_C(x) &= \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} = \sum_{i=1}^n \sum_{j=1}^n (d_{ij} + \alpha_i + \beta_j) x_{ij} = \\ &= \sum_{i=1}^n \sum_{j=1}^n d_{ij} x_{ij} + \sum_{i=1}^n \alpha_i \sum_{j=1}^n x_{ij} + \sum_{j=1}^n \beta_j \sum_{i=1}^n x_{ij}. \end{aligned}$$

Mivel x lehetséges megoldás, ezért mindkét utóbbi második szumma egy: $\sum_{j=1}^n x_{ij} = \sum_{i=1}^n x_{ij} = 1$. Ezért aztán $\gamma = \sum_{i=1}^n \alpha_i + \sum_{j=1}^n \beta_j$ megfelelő választás: $z_C(X) = Z_D(X) + \gamma$. Ez épp a segédtelet állítását igazolja. $\square$

KÖVETKEZMÉNY. Az optimális megoldás meghatározását illetően elegendő olyan utazó ügynök feladatokat vizsgálni, amelyekre $C \geq 0$ teljesül.

AZ UTAZÓ ÜGYNÖK FELADAT ÉS A HOZZÁRENDELÉSI FELADAT

Vegyük észre, hogy a harmadik feltételcsoporttól eltekintve az utazó ügynök feladat a hozzárendelési feladatot adja. Emiatt az utazó ügynök feladat L lehetséges megoldásai halmaza része a megfelelő hozzárendelési feladat S lehetséges megoldásai halmazának: $L \subset S$. Mivel

$$\min \{z(X) : X \in S\} \leq \min \{z(X) : X \in L\},$$

ezért

- (i) ha X optimális megoldása a $H(C)$ hozzárendelési feladatnak és X teljes körút, akkor X egyben optimális megoldása a $TSP(C)$ utazó ügynök feladatnak is, és
- (ii) Ha X optimális megoldása a $H(C)$ hozzárendelési feladatnak, akkor $z(X)$ egy alsó korlátja a $TSP(C)$ utazó ügynök feladat optimumértékének.

Az 1 - 2 - 3 - 4 - 5 teljes körúthoz tartozó egy hozzárendelési feladat költségmátrixa (M az adott számítógépes környezetben ábrázolható legnagyobb szám):

$$\begin{bmatrix} M & 1 & 5 & 5 & 5 \\ 5 & M & 1 & 5 & 5 \\ 5 & 5 & M & 1 & 5 \\ 5 & 5 & 5 & M & 1 \\ 1 & 5 & 5 & 5 & M \end{bmatrix}$$

A következő költségmátrixhoz tartozó hozzárendelési feladat optimális megoldása (1 - 2 - 3, 4 - 5) nem ad lehetséges megoldást a kapcsolódó utazó ügynök feladatra:

$$\begin{bmatrix} M & 1 & 5 & 5 & 5 \\ 5 & M & 1 & 5 & 5 \\ 1 & 5 & M & 5 & 5 \\ 5 & 5 & 5 & M & 1 \\ 5 & 5 & 5 & 1 & M \end{bmatrix}$$

A megfelelő optimális célfüggvényérték mindkét előző hozzárendelési feladatra 5. Az utóbbi költségmátrixhoz tartozó utazó ügynök feladat egy optimális megoldása az 1 - 2 - 3 - 4 - 5 útvonal, amihez az optimális költség 13.

AZ UTAZÓ ÜGYNÖK FELADAT ÉS A HOZZÁRENDELÉSI FELADAT II.

Az utazó ügynök feladat tulajdonságai miatt számos esetben ésszerű azt feltételezni, hogy a költségmátrix *szimmetrikus*. E szabály alól azonban vannak természetes kivételek. Még városok közti távolság esetén is lehet eltérés az oda és a visszaút között, de még gyakoribb a termékek gyártási sorrendje megválasztása esetén, hogy az A termék gyártásáról a B-re pl. gyorsabban lehet áttérni, mint fordítva.

Az utazó ügynök feladat nagyszámú feltételére, és a leszámolási eljárás reménytelenül nagy műveletigényére tekintettel a feladatot szokás egyrészt *korlátozás és szétválasztás módszerrel* megoldani, másrészt gyors, *heurisztikus közelítő eljárásokat* alkalmazni.

A korlátozás és szétválasztás módszerét gyorsítani lehet akkor, ha jó korlátokat tudunk megadni az optimum értékére. Ennek eszköze lehet a korábban ismertetett áttérés a megfelelő hozzárendelési feladatra.

Egon Balas és Paolo Toth számítógépes kísérleteket végzett, amelynek során 400 problémát generáltak véletlenszerűen. Egyenletes eloszlással határozták meg a feladat méretét az $50 \leq n \leq 250$ határok között, és a célfüggvény együtthatókat is az 1 és 100, illetve más esetben az 1 és 1000 közötti egészek közül.

Az így előálló feladatokat megoldották mint TSP feladatot, és mint hozzárendelési feladatot is. Azt kapták, hogy a hozzárendelési feladatok optimumértékei átlaga 99.2 %-a volt az utazó ügynök feladat optimumai átlagának. Az eredményekből kitűnt, hogy n növekedésével a relatív eltérés csökkent.

Ezzel együtt az utazó ügynök feladat az ún. NP nehéz feladatok közé tartozik, tehát nem ismeretesek azt az n feladatméret polinom függvényével korlátozott idő alatt megoldó eljárások. Emiatt előtérbe kerültek a közelítő módszerek, amelyek csaknem optimális megoldást adnak viszonylag rövid idő alatt.

A heurisztikus algoritmusok lényege, hogy ezek az eljárások gyorsan, kevés műveletigény árán javítják a közelítő megoldásokat – de arra nem mindig van remény, hogy ezek minden esetre konvergálnának az optimális megoldáshoz.

HEURISZTIKÁK AZ UTAZÓ ÜGYNÖK FELADATRA

Legközelebbi város beillesztése: az eddigi részkörutat bővítsük egy olyan újabb várossal, hogy a részkörút városain kívüli városok között megkeressük azt a k -t, amelynek távolsága a legkisebb a részkörút valamely városához. Ezután ezzel bővítjük a részkörutat: Ha c_{ik} volt a kiválasztásban talált minimális távolság, akkor az új részkörútban az i . város után a k . következik, majd ezután az i várost eddig követő i' .

PÉLDA. Tekintsük a legutóbb vizsgált feladatot, amelynek C költségmátrixa

M	1	5	5	5
5	M	1	5	5
1	5	M	5	5
5	5	5	M	1
5	5	5	1	M

Kiinduláshoz vegyük az 1 - 1 körutat. A többi város ettől mért távolsága rendre 1, 5, 5 és 5. Ez alapján a 2. várossal bővül a részkörút: 1 - 2 - 1.

A következő lépésben tekintsük a 3., 4. és 5. városok távolságait az 1. és 2. városoktól. Ezek a C mátrix főátlója feletti elemek, az első és második sorban (c_{12} kivételével). A minimumot a $c_{23} = 1$ távolság adja, ezzel a kibővített részkörút:
1 - 2 - 3 - 1.

A harmadik lépésben a minimalizálandó távolságok c_{14} , c_{24} , c_{34} , c_{15} , c_{25} és c_{35} . Ezek mindegyike 5, válasszuk a 4. várost a bővítéshez. Ezzel az új részkörút: 1 - 2 - 3 - 4 - 1.

Az utolsó lépésben már csak egy városból választhatunk, és ezt a 4. városhoz kell kötni: 1 - 2 - 3 - 4 - 5 - 1 lesz a heurisztika által talált teljes körút. Ez nyilván lehetséges megoldás, a hozzá tartozó össztávolság 13. Ez egyben optimális megoldás is – bár ez nem jellemző a heurisztikák esetén. $\square$

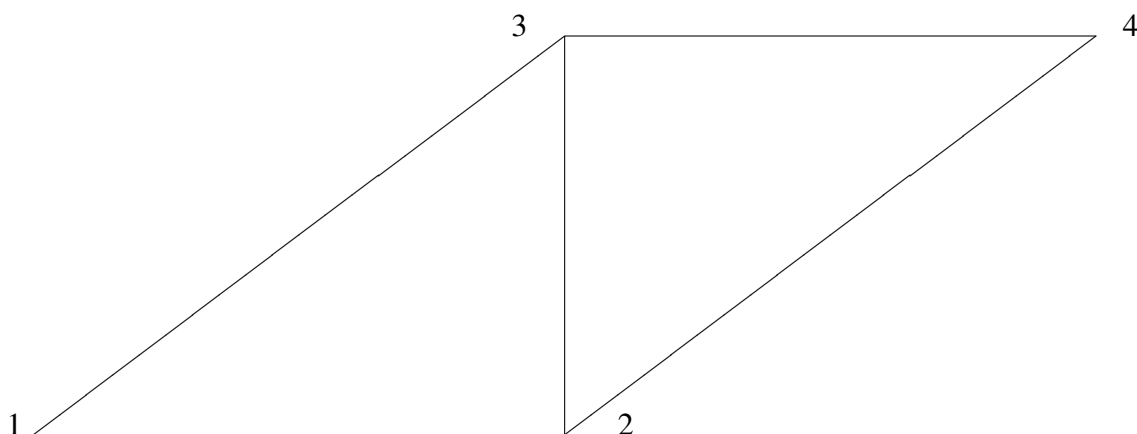
HEURISZTIKÁK AZ UTAZÓ ÜGYNÖK FELADATRA II.

A legközelebbi város hozzáadása: Az előzőnél egyszerűbb heurisztika: az aktuális útvonalat mohó módon bővíti a meglevő útvonal végpontjához legközelebbi várossal. Ha minden város szerepel már az útvonalban, akkor az utolsót összeköti az elsővel, és így képez teljes körutat.

PÉLDA. Mutassunk most egy olyan példát, amely szuboptimális megoldást eredményez. 3 város esetén ez nem lehetséges, ahogy könnyen belátható. Tekintsük akkor a következő távolságmátrixot:

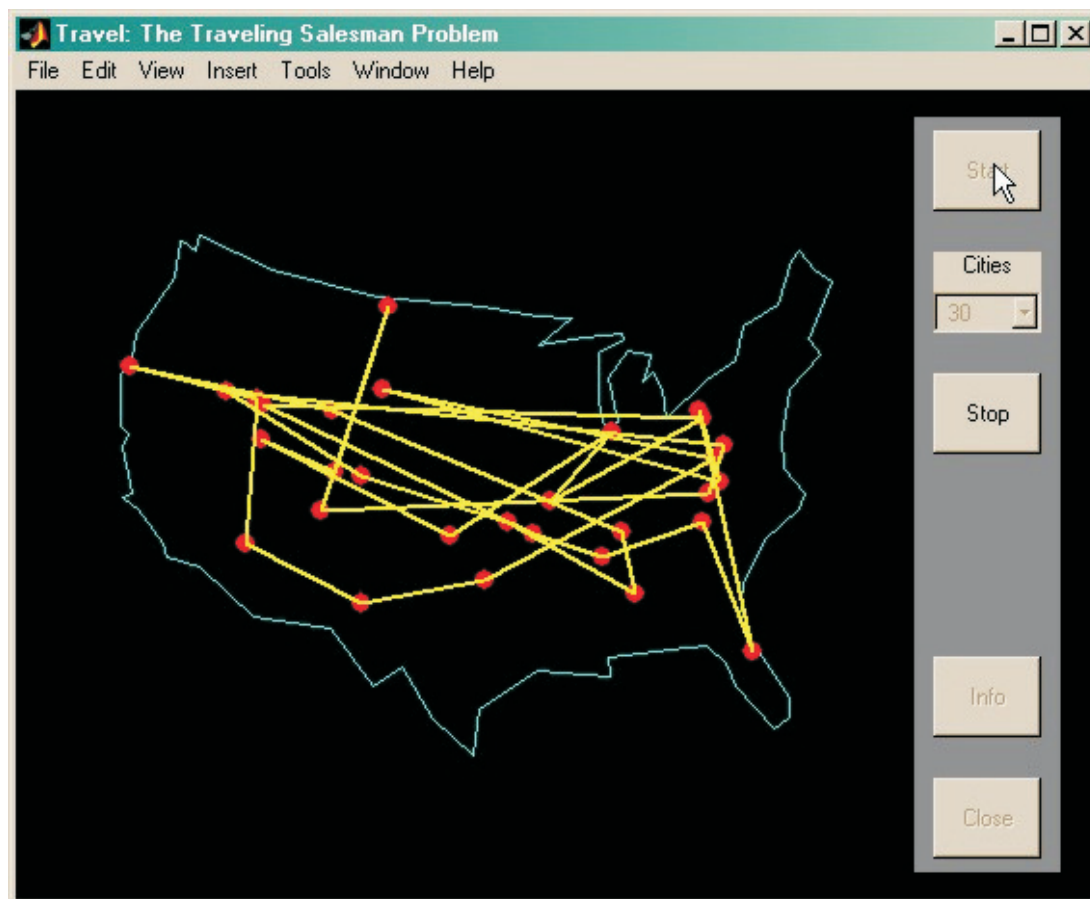
M	4	5	$\sqrt{73}$
4	M	3	5
5	3	M	4
$\sqrt{73}$	5	4	M

Ez két, a rövidebb oldalával összefordított Pitagorasz-háromszöget tartalmaz. Az így kapott paralelogramma természetes módon ad egy körüljárást, amely 18 hosszú. Ez a körút az 1 - 2 - 4 - 3 - 1.



Kövessük végig a legközelebbi város hozzáadása heurisztikát az 1. városból indulva. Az elsőhöz a második város a legközelebbi (lásd a távolságmátrix első sorát). A 2. városból (az első kivéve a keresésből) a 3. város van a legközelebb. A harmadikból már csak a 4.-be mehetünk. Ezután zárjuk a körutat, ami így 1 - 2 - 3 - 4 - 1. Az ehhez tartozó teljes megtett út $4 + 3 + 4 + \sqrt{73} \approx 19.544$. $\square$

AZ UTAZÓ ÜGYNÖK FELADAT MEGOLDÁSA MATLABBAL



Az ábrán az utazó ügynök feladat egy esetének kezdő helyzetét lehet látni. Az Amerikai Egyesült Államok véletlenül generált 30 városa van kiemelve, és a köztük keresett körút kezdeti változata. Figyeljük meg, hogy a jelen helyzet még nagyon távol van az optimális megoldástól, több helyen még az is kérdéses, hogy egyáltalán körút-e a megadott.

A Matlab Help menüsorában a Demos utasítás kiadása után kapott párbeszédés ablakban kérjük a Matlab bemutató programokat, azután a More demos fület válasszuk, majd a kapott listából a Travelling Salesman programot.

A jelen ábra a Corel Draw Capture programjával készült, és a kapott postscript ábra 30-szor nagyobb, mint a következő, amit a Matlab saját export utasítása adott.

AZ UTAZÓ ÜGYNÖK FELADAT MEGOLDÁSA MATLABBAL II.

Az alábbi programrészlet a Matlab utazó ügynök feladatra írt bemutató programjából való. Az eljárás célja nem a gyors megoldás, hanem a feladat nehézségének, és a megoldás menetének illusztrálása volt. A teljes Matlab program kb. 300 soros.

Figyeljük meg a viszonylag könnyen olvasható algoritmust, amely az eddigi közelítő megoldáson azt kísérli meg, hogy egy véletlenül generált körút szakaszon a bejárás sorrendjét az ellenkezőjére változtatja. Amennyiben a beavatkozás sikeres volt, akkor a javított útvonal lesz a továbbiakban a jelölt a megoldásra.

```
% Try a point for point swap
% =====
swpt1=floor(npts*rand)+1;
swpt2=floor(npts*rand)+1;

swptlo=min(swpt1, swpt2);
swpthi=max(swpt1, swpt2);

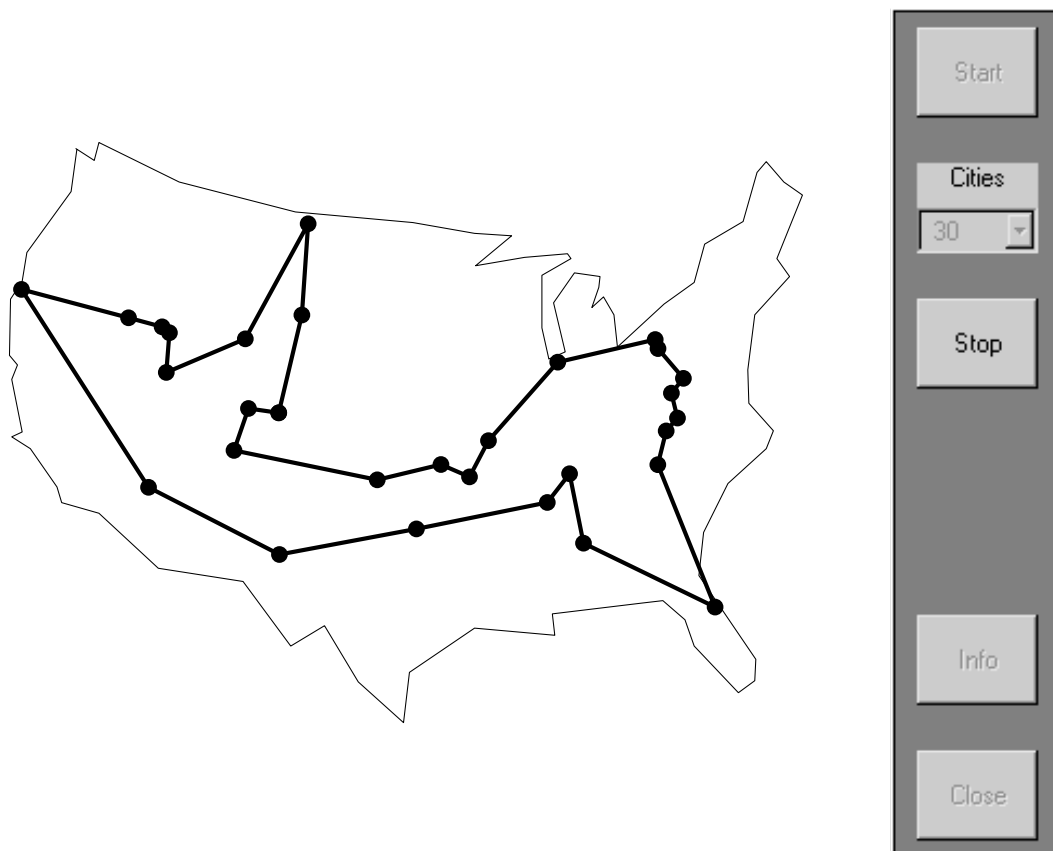
order=1:npts;
order(swptlo:swpthi)=order(swpthi:-1:swptlo);
pnew = p(order);

lennew=LocalPathLength(pnew, distmatrix);
if lennew<len,
    p=pnew;
    len=lennew;
    drawFlag=1;
end;
% =====
```

Érdekes és hatékony a tömbök címzési módja, pl.

```
order(swpthi:-1:swptlo).
```

AZ UTAZÓ ÜGYNÖK FELADAT MEGOLDÁSA MATLABBAL III.



Ez az ábra a feladat közel stabilizálódott közelítő megoldását mutatja. Az ehhez szükséges számítási idő pár másodperc volt.

A TÁVOLSÁGVEKTOR SZEREPE A HEURISZTIKÁKBAN

A legközelebbi város beillesztése heurisztika $\mathcal{O}(n^2)$ műveletigényű, ha az ún. *távolságvektorokat* használjuk: az iteráció minden lépésében ez a vektor tartalmazza az eddigi részkörútbeli, és az azon kívüli városok távolságát.

Kiindulásként ez a vektor az 1. várostól való távolságokat tartalmazza. Ez később, a k . városnak a részkörútba való bevonása után úgy módosul, hogy a k . városnak a részkörúton kívüli városoktól mért távolságai és az adott városokra vonatkozó, a távolságvektorban meglévő értékek minimumát kell képezni.

A távolságvektor használatával minden iterációs lépésben n -nél kisebb számú összehasonlítást végzünk, míg enélkül a távolságmatrix n^2 -tel arányos számú elemét kellene megvizsgálni, és ez nyilvánvalóan lényegesen rontaná a heurisztika műveletigényét.

PÉLDA. Tekintsük ismét a korábban vizsgált utazó ügynök feladatot, amelynek költségmatrixa

M	1	5	5	5
5	M	1	5	5
1	5	M	5	5
5	5	5	M	1
5	5	5	1	M

A legközelebbi város beillesztése heurisztika első lépésében az 1 - 1 részkörúthoz a távolságvektor az (1, 5, 5, 5) elemeket tartalmazza (ez a mátrix első sora, a főátlóbeli elem kivételével).

A következő lépésben a részkörutat a 2. várossal bővítjük. A távolságvektorban a 2. városra vonatkozó elemet törölhetjük. A következő vektorelem $\min(5, 1) = 1$ lesz. A többi komponens nem változik: az új távolságvektor (-, 1, 5, 5).

Az eljárást tovább folytatva az utolsó két képzett távolságvektor rendre (-, -, 5, 5) és (-, -, -, 1) lesz. □

A HEURISZTIKUS ALGORITMUSOK HATÁSOSSÁGA VIZSGÁLATA

A heurisztikus algoritmusok eredményessége minősítésére három vizsgálati módszerosztály használatos:

- a legrosszabb esetek vizsgálata (worst case analysis),
- a valószínűségi analízis (probabilistic analysis), és
- az empirikus analízis (empirical analysis).

1. Az első megközelítés azt vizsgálja, hogy a legrosszabb lehetséges esetben milyen eltérést kaphatunk a heurisztika által adott közelítő megoldás és a valódi optimum között. Ezt fejezi ki lényegében a *heurisztika aszimptotikus hányadosa*. Legyen A egy a tekintett $\mathcal{C}$ problémaosztály feladatai megoldását megközelítő heurisztika, $A(P)$ a P feladaton a heurisztika által adott közelítés lehetséges megoldásának célfüggvény értéke, és $OPT(P)$ a P feladat optimuma. Ekkor amennyiben

$$M_A = \limsup \left\{ \frac{A(P)}{OPT(P)} \mid P \in \mathcal{C} \right\}$$

létezik, akkor az a heurisztika aszimptotikus hányadosa. Az aszimptotikus hányados értelmezéséhez tekintsük a következő egyenlőtlenséget:

$$A(P) \leq (M_A + \epsilon) OPT(P).$$

Ez az összefüggés véges sok eset kivételével érvényes minden $\mathcal{C}$ -beli feladatra. Mivel minden feladatnak van egy bizonyos (véges) mérete, ezért elegendően nagy feladatméret esetén a fenti feltétel már mindig érvényes, tehát ekkor már minden ekkora feladatra tudjuk, hogy a kapott közelítő megoldás legfeljebb hányszor rosszabb eredményt ad.

Nagyon jó, 1 és 2 közé eső aszimptotikus hányadosok érvényesek egyes *ládapakolási, szabási feladatokra*. Másrészt 1976-ban S. Sahni és T. Gonzalez igazolták, hogy ha létezik aszimptotikus hányados valamely polinomkorlátos TSP heurisztikára, akkor $P = NP$.

Ez utóbbi megállapítás alapján arra lehetne következtetni, hogy akkor nem érdemes polinomkorlátos TSP heurisztikákat keresni. Mégis (vö. a lineáris programozás esetét a $3n$ átlagos, és 2^n legrosszabb esetre vonatkozó iterációszámmal) *speciális* utazó ügynök feladatok esetén számíthatunk jó aszimptotikus hányadosra, és a gyakorlatban előforduló feladatok esetén is lehetnek egyes heurisztikák gyorsak.

2. A valószínűségi analízis (probabilistic analysis) módszere alkalmazása során azonos méretű feladatok paramétereit (együtthatóit) független valószínűségi változóknak tekintjük. Ezek eloszlására alkalmas feltételek teljesülését feltételezzük (gyakran azt, hogy az eloszlások egyenletesek).

Ebben az esetben $A(P)$, $OPT(P)$, és ezek hányadosa is valószínűségi változó. Az utóbbi várható értékéből az átlagos eltérésre lehet következtetni. Ezzel a módszerrel szemben az a gyakori kifogás, hogy az eloszlásokra vonatkozó feltételek nem feltétlenül helyesek, márpedig ezek a kapott eredményt döntően befolyásolhatják.

3. Az empirikus vizsgálat konkrét feladatmegoldásból kapott eredményekből képez mutatókat. Az eljárás lényege, hogy nagy számú feladatot generálunk úgy, hogy a feladat együtthatóit valamely adott (általában egyenletes) eloszlással egy rögzített számtartományból választjuk. Ezekre a problémákra mind a heurisztika által adott közelítést, mind a pontos megoldást meghatározzuk.

Ezután kiszámoljuk a hányadosokat és azok átlagát képezzük. Ha elegendően sok feladatot vizsgáltunk meg, akkor az eredmény, az empirikus hányados jellemezni fogja a vizsgált feladatosztályon a heurisztika közelítéseinek jóságát. Másrészt a számtartománynak a gyakorlati problémák halmazához való viszonya, és a feltételezett eloszlás általában kérdéses.

Összefoglalva az eddigieket, a heurisztikus algoritmusok közelítő megoldásai minőségének jellemzésére érdemes mindhárom tárgyalt módszert használni, hogy így kiegyensúlyozottabb képet kaphassunk.

TOVÁBBI HEURISZTIKÁK AZ UTAZÓ ÜGYNÖK FELADATRA

Az elsőként ismertetett legközelebbi város beillesztése (nearest addition) heurisztika polinomkorlátos a távolságvektor használatával, a műveletigénye $\mathcal{O}(n^2)$. Nincs rá aszimptotikus hányados.

Ehhez a heurisztikához hasonló, de annál jobb megoldást ad a *legközelebbi város beszúrása* (nearest insertion) nevű eljárás. Ez is egy részkörúttal indul, majd az aktuális részkörutat olyan várossal bővíti, amelynek távolsága a részkörút egyik városához minimális.

Az eltérés abban van, hogy az új várost oda fogjuk beszúrni a részkörútba, ahol a beillesztés a legkisebb össztávolság növekedést okozza. Ez a heurisztika is $\mathcal{O}(n^2)$ műveletigényű. Másrészt olyan utazó ügynök feladatokra, amelyek költségmátrixa szimmetrikus, és érvényes rá a háromszög egyenlőtlenség, a legközelebbi város beszúrása heurisztika aszimptotikus hányadosa 2. Más szóval véges sok esettől eltekintve legfeljebb kétszer nagyobb lesz a heurisztikával kapott célfüggvény, mint az optimális.

Érdekes, hogy nem mohó algoritmus, de van értelme a *legtávolabbi város beszúrásának* (farthest insertion). Az eljárás beszúrása az előző heurisztika módszerével történik, itt is a lehető legkisebb költség-növekedést jelentő helyre történik a hozzáadás. Az empirikus vizsgálatok szerint ez a jobb, mint az előbbi három.

A *legolcsóbb beszúrás* (cheapest insertion) módszer az eddigiek bizonyos értelmű kiteljesítése: azt a várost választjuk a meglevő részkörút bővítésére, amelyik alkalmas helyre való beszúrása a lehető legkisebb költség-növekedést jelenti. Az eljárás műveletigénye $\mathcal{O}(n^3)$, és a korábban már említett szimmetrikus költségmátrixú, a háromszög-egyenlőtlenséget teljesítő költségmátrixú TSP feladatokra az eljárás aszimptotikus hányadosa 2.

Mivel az említett heurisztikák mind viszonylag gyorsak, és ezek eredménye függ az induló város megválasztásától, ezért szokásos a heurisztikus algoritmusokat többször is végrehajtani, eltérő városokból indulva. Ha minden városra lefuttattunk egy heurisztikát, akkor az összevont eljárást szokás *all cities* változatnak nevezni. Az említetteken kívül használatosak heurisztikák a hozzárendelési feladat megoldásából kapott két vagy több részkörút összekapcsolására is.

A SZABÁSI FELADAT

Adott méretű rudak, egyéb munkadarabok leszabása során gyakran felmerül az a feladat, hogy adott hosszúságú nyersanyagból hogyan vágjuk le az előírt összetételű végtermék halmazát úgy, hogy a lehető legkevesebb veszteség maradjon. Ezt a problémát egydimenziós szabási, vagy ládapakolási feladatnak (bin packing) nevezik.

A gyakorlati feladatok közül ide tartozik az, amikor egy építőipari vállalat adott profilú, rögzített hosszúságú fémrudakból a könnyűszerkezetes építkezéshez nagyszámú rövidebb rudat, oszlopot akar levágni, de figyelembe kell venni, hogy a lehető legkevesebb teljes rudat kezdjük meg, vagy az a cél, hogy az eldobandó nyersanyag mennyisége legyen minimális.

Ilyen feladatra vezet az is, ha egy síküveggyárban adott szélességű, hosszú üvegtáblákból kell megadott összetételű (az eredeti tábla szélességét megtartó) üveglapokat kivágni.

A hátizsák feladathoz hasonló megfogalmazást is lehet adni: adottak különböző súlyú tárgyak, ezek mindegyikét el kell helyezni minimális számú hátizsákban úgy, hogy azok közös súlykorlátját ne lépjük túl.

A számítástechnikából azt a problémát idézhetjük, amikor rögzített méretű tárhelyekre (pl. partíciókra) kell különböző méretű adatállományokat úgy elhelyezni, hogy ehhez a minimális számú tárhelyet használjuk fel.

A logisztikában az a feladat, hogy adott teherbírású járművekből mennyit kell minimálisan alkalmazni ahhoz, hogy adott, különböző súlyú tárgyakból álló rakomány elszállítható legyen,...

Többdimenziós szabási feladatot kapunk, ha az elhelyezendő tárgyak több kiterjedését is korlátozzuk, pl. a hosszát és a szélességét. Felmerülhet, hogy ebben az esetben mindenféle vágást megengedünk-e, vagy csak az anyag szélétől széléig menő, ún. guillotine-vágásokat.

A SZABÁSI FELADAT MODELLJE

Tekintsük először az egydimenziós szabási feladat L.V. Kantorovicstól származó modelljét. Legyenek adva korlátlan mennyiségben K hosszúságú félkész termékek, ahol K pozitív egész. Az egyes végtermékek hossza legyen $k_1, k_2, \dots, k_n$. Ezek egyike sem nagyobb K -nál. A végtermékekből rendre $r_1, r_2, \dots, r_n$ darabot kell levágni.

A modellben az a_j vektort lehetséges szabásnak nevezzük, ha

$$a_{ij} \geq 0, \quad (i = 1, \dots, n),$$

$$\sum_{t=1}^n k_t a_{t,j} \leq K.$$

Ezek után jelölje $A = (a_1, \dots, a_p)$ az összes lehetséges szabásokból előállított mátrixot. Legyen r az $r_1, \dots, r_n$ komponensekből álló oszlopvektor, és c az a sorvektor, amelynek minden eleme egyenlő, egy félkész termék pozitív költsége.

Az egydimenziós szabási feladat optimumszámítási modellje ezekkel a paraméterekkel:

$$\min z(x) = cx,$$

feltéve, hogy

$$Ax = r,$$

$$x \geq 0, \quad x \text{ egész}, \quad (r \geq 0).$$

A modell használatának nehézségét az mutatja, hogy már a lehetséges szabások A mátrixának az összeállítása is komoly problémát jelent.

A célfüggvény konstans együtthatóit az magyarázza, hogy így a célfüggvény a lehetséges szabásokból a minimális darabszámúhoz tartozó megoldást fogja kiválasztani.

Vegyük észre, hogy a megadott modell egy egészértékű lineáris programozási feladat, amely a felírt formában a standard feladatnak felel meg.

PÉLDA A SZABÁSI FELADATRA

PÉLDA. Tekintsünk egy nagyon egyszerű szabási feladatot: a félkész termékek hossza legyen 1, az egyes végtermékeké pedig 0.25 és 0.5. Írjuk elő, hogy összesen 4 darabot kell levágni az első termékből és kettőt a másodikból.

Ahogy könnyen ellenőrizhető, a lehetséges szabások ezek alapján:

$$(0, 0)^T, (0, 1)^T, (0, 2)^T, (1, 0)^T, (1, 1)^T, (2, 0)^T, (2, 1)^T, (3, 0)^T \text{ és } (4, 0)^T.$$

Ezek segítségével felírhatjuk az optimalizálási feladatot:

$$\min z(x) = cx = \sum_{i=1}^9 x_i,$$

feltéve, hogy

$$Ax = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 2 & 2 & 3 & 4 \\ 0 & 1 & 2 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} x = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$$

és

$$x \geq 0, \quad x \text{ egész}, \quad (r \geq 0).$$

A feladat spekulatív megoldása során vegyük észre, hogy x_1 értéke nulla kell hogy legyen az optimális megoldásban, hiszen ez a teljesítendő termékszámhoz nem járul hozzá, de növeli a költséget.

A másik végletet x_9 képviseli, mert egy ilyen felosztású félkész termék felhasználása adja a legtöbb teljesített készterméket. Ilyen szabásokból viszont nem áll össze a teljes szabási előírás.

Másrészt megállapíthatjuk, hogy a jobboldali r vektor arányait pontosan adja az x_7 változóhoz tartozó szabás. Ebből két darab kell ahhoz, hogy teljesüljön az egyenlőség feltételünk. Ehhez a 2 célfüggvényérték tartozik.

Látható, hogy ennél jobbat nem lehet elérni, mert nincs olyan lehetséges szabás, amelyből egy adná a jobboldali r vektort. Van viszont még egy optimális megoldás, az $x_3 = 1, x_9 = 1$ (és minden további $x_i = 0$): ez is kettes célfüggvény értéket ad – és ezzel ki is merítettük az optimális megoldások halmazát. $\square$

AZ OSZLOPGENERÁLÁS MÓDSZERE A SZABÁSI FELADATRA

A szabási feladat ismertett modellje nehezen használható, elsősorban a lehetséges szabások nagy száma miatt. P.C. Gilmore és R.E. Gomory egyszerűsítették a modellt.

Első lépésben elhagyták az egészértékűségi feltételt. Az ezt támogató érvelés szerint az ipari alkalmazásokban a nem egész optimális megoldás egész értékre való valamilyen átalakítása elfogadható.

Az oszlopgenerálás módszere ezután a módosított szimplex algoritmust használja, így nem szükséges a teljes A mátrix ismerete, elegendő annak az oszlopnak az előállítása, amelyben generáló elemet keresünk.

Ahhoz, hogy a módosított szimplex algoritmust használni lehessen, az eddigi

$$\min z(x) = cx,$$

feltéve, hogy

$$Ax = r,$$

$$x \geq 0, \quad (r \geq 0).$$

standard alakú feladatunkat lehetséges kanonikus alakra kell hozni. A bázisváltozók legyenek azokhoz a lehetséges szabásokhoz tartozók, amelyek épp az egységmátrixot adják: $a'_1 = (1, 0, \dots, 0)^T, \dots, a'_n = (0, \dots, 0, 1)^T$. Tekintsük ezeket az A mátrix első n oszlopának. Ezután már csak annyit kell tenni, hogy minden sornak a $-c$ -szeresét hozzáadjuk a célfüggvényhez. Ez a művelet épp a bázisváltozókhoz tartozó célfüggvény-együtthatókat fogja nullázni.

Legyen d egy olyan n dimenziós vektor, amelynek minden komponense c . Ezzel a feladatunk a következő lehetséges kanonikus alakban írható:

$$\min z(x) = dr + (c - dA)x,$$

feltéve, hogy

$$Ax = r,$$

$$x \geq 0, \quad (r \geq 0).$$

AZ OSZLOPGENERÁLÁS MÓDSZERE A SZABÁSI FELADATRA II.

A következő feladat a legkisebb célfüggvény-együttható meghatározása. Legyen az A mátrix egy tetszőleges oszlopvektora $(v_1, \dots, v_n)^T$. Ez alapján a $c - \sum_{t=1}^n d_t v_t$ összeg minimumát keressük, ahol a d vektor n -dimenziós, és minden komponense c . Ezt a szélsőértéket ott kapjuk, ahol a $\sum_{t=1}^n d_t v_t$ összeg maximális. Mivel v egy lehetséges szabás, ezért $\sum_{t=1}^n k_t v_t \leq K$.

A legkisebb célfüggvény-együttható meghatározásához eszerint a következő speciális egészértékű lineáris programozási feladatot kell megoldani:

$$\max w(v) = \sum_{t=1}^n d_t v_t,$$

feltéve, hogy

$$\sum_{t=1}^n k_t v_t \leq K,$$

$$v_t \geq 0, \text{ egész, } t = 1, \dots, n.$$

Ez a feladat a korábban megismert hátizsák feladat a K súlykorláttal, k_t súlyokkal és d_t értékekkel. Ennek megoldásával tudjuk meghatározni a szabási feladat generáló eleme oszlopát.

A szabási feladat legkisebb célfüggvény-együtthatója és a generálóelem ismeretében végre tudjuk hajtani a módosított szimplex algoritmus egy lépését. Ezután az eddigi lépéseket ismétljük. Az egész eljárás akkor fejeződik be, ha az aktuális hátizsák feladat optimuma nem nagyobb, mint c . Ekkor az eredeti feladat minden célfüggvény együtthatója nemnegatív.

PÉLDA AZ OSZLOPGENERÁLÁS MÓDSZERRE

PÉLDA. Tekintsük ismét az előző nagyon egyszerű szabási feladatot: a félkész termékek hossza 1, az egyes végtermékeké pedig 0.25 és 0.5. Összesen 4 darabot kell levágni az első termékből és kettőt a másodikból. Az optimalizálási feladat átrendezve úgy, hogy az egységmátrix legyen a baloldalon ($x_1 - x_4$ csere):

$$\min z(x) = cx = \sum_{i=1}^7 x_i,$$

feltéve, hogy

$$Ax = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 2 & 2 & 3 & 4 \\ 0 & 1 & 2 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} x = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$$

$$x \geq 0, \quad (r \geq 0).$$

Az ehhez tartozó táblázat $c = 1$ értékkel

x_4	x_2	x_3	x_1	x_5	x_6	x_7	x_8	x_9	
1	0	0	0	1	2	2	3	4	4
0	1	2	0	1	0	1	0	0	2
1	1	1	1	1	1	1	1	1	0

A szimplex táblázat az új, $c - \sum_{t=1}^n d_t v_t$ új célfüggvénnyel:

	x_3	x_1	x_5	x_6	x_7	x_8	x_9	
x_4	0	0	1	2	2	3	4	4
x_2	2	0	1	0	1	0	0	2
	-1	1	-1	-1	-2	-2	-3	-6

A leolvasható bázismegoldás azt jelenti, hogy az $(1, 0)$ szabásból kellene 4 darab, és a $(0, 1)$ szabásból 2 darab. Ez összesen 6 félkész terméket jelent, ami nyilván még javítható. Oldjuk meg először a feladatot a szimplex algoritmussal.

Itt a második oszlop alapján nem lehetne javítani a célfüggvény értékét (de nem is találunk benne generáló elemet). A legkisebb negatív célfüggvény együttható az utolsó oszlopot jelöli ki, és ebben az első mátrixelem (a négyes) lesz a generálóelem. A transzformált, következő szimplex táblázat:

	x_3	x_1	x_5	x_6	x_7	x_8	x_4	
x_9	0	0	1/4	1/2	1/2	3/4	1/4	1
x_2	2	0	1	0	1	0	0	2
	-1	1	-1/4	1/2	-1/2	1/4	3/4	-3

PÉLDA AZ OSZLOPGENERÁLÁS MÓDSZERRE II.

Az előző szimplex táblázat:

	x_3	x_1	x_5	x_6	x_7	x_8	x_4	
x_9	0	0	1/4	1/2	1/2	3/4	1/4	1
x_2	2	0	1	0	1	0	0	2
	-1	1	-1/4	1/2	-1/2	1/4	3/4	-3

A legkisebb célfüggvény-együttható az x_3 változónak a bázisba lépését jelenti.

	x_2	x_1	x_5	x_6	x_7	x_8	x_4	
x_9	0	0	1/4	1/2	1/2	3/4	1/4	1
x_3	1/2	0	1/2	0	1/2	0	0	1
	1/2	1	1/4	1/2	0	1/4	3/4	-2

Erről a szimplex táblázatról már le lehet olvasni a végeredményt: a 3. és a 9. szabásból kell egyet-egyet venni. Ez lehetséges megoldás lesz, és az ezzel adódó optimális célfüggvény érték 2. Az x_7 -es szabásból kettő is optimális, és ez összhangban is van az eredményünkkel.

Tekintsük most a feladatunkat a Gilmore-Gomory-féle oszlopgenerálásnak megfelelően. A módosított szimplex algoritmusról tanultak miatt az ugyanezen bázismegoldásokon keresztül jut azonos eredményhez. Az eltérés a módosított szimplex algoritmus kivitelezésében, és főleg a legkisebb célfüggvény-együttható előállításában van. Ennek illusztrálásához lépünk vissza a

x_4	x_2	x_3	x_1	x_5	x_6	x_7	x_8	x_9	
1	0	0	0	1	2	2	3	4	4
0	1	2	0	1	0	1	0	0	2
1	1	1	1	1	1	1	1	1	0

feladathoz. Írjuk fel erre az új bázisváltozó meghatározásához a megfelelő hátizsák feladatot:

$$\max w(v) = \sum_{t=1}^n d_t v_t = \sum_{t=1}^n c v_t = \sum_{t=1}^n v_t,$$

feltéve, hogy

$$\sum_{t=1}^n k_t v_t \leq K, \quad v_t \geq 0, \quad \text{egész, } t = 1, \dots, n.$$

Ez a feladat lényegében azt a lehetséges szabást keresi, amely a végtermékekből a legtöbbet állítja elő. □

HEURISZTIKÁK A SZABÁSI FELADATRA

Az ismertetett egzakt módszerek mind nehezen végrehajthatók nagyobb méretű gyakorlati feladatokra. Az alábbi heurisztikák ezzel szemben gyorsan adnak jó közelítő megoldást.

A *first fit módszer* a végtermékeket felsorolási sorban tekinti, az aktuális munkadarabot az első olyan félkész termékre helyezi el, amelyen az elfér. Ebben az értelemben ez egy mohó algoritmus.

A *best fit algoritmus* olyan rudat ad meg, amelyről az aktuális munkadarabot levágva a legkevesebb maradék képződik.

A *worst fit eljárás* pedig értelemszerűen olyan megkezdett rudat választ az aktuális munkadarab levágásához, amelyiken a vágás után a leghosszabb még felhasználható szakasz marad.

Előnyös a leszabandó végtermékeket előzőleg nagyság szerint rendezni. Ez lényegesen javítja a heurisztika eredményét.

Az említetteken felül további egyszerű heurisztikák léteznek. Érdemes megemlíteni, hogy az eddig tárgyalt szabási problémát *offline szabási feladat*nak is nevezhetjük, hiszen az összes adat ismeretében kellett megoldanunk, és az összes félkész terméket az eljárás teljes tartalma alatt felhasználhattuk a megoldás javításához.

Ezzel szemben online-nak nevezzük a szabási feladatot, ha a leszabandó termékek adatai beérkeztekor véglegesen döntenünk kell elhelyezésükről (a többi adat ismerete nélkül), vagy ha egy időben csak adott rögzített számú félkész termékből lehet szabni. Utóbbi esetben ha különben nem tudnánk tovább haladni, akkor valamelyik félkész termék eddigi szabását véglegesnek kell nyilvánítani, és egy újat vonunk be a szabás meghatározásába.

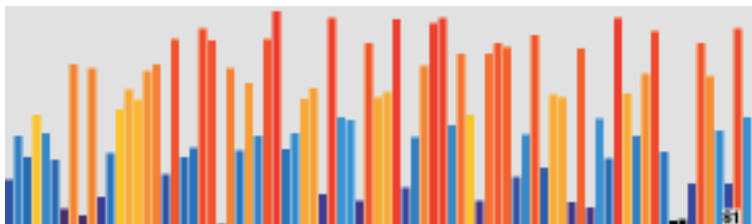
A szabási feladat heurisztikáinak tömör, látványos illusztrációja található a

<http://www.cs.arizona.edu/icon/oddsends/bpack/bpack.htm>

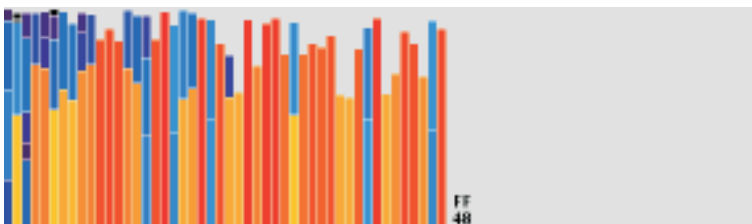
címen

HEURISZTIKÁK A SZABÁSI FELADATRA II.

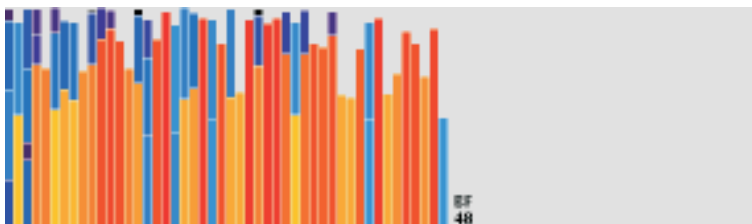
A heurisztikus algoritmusok működését illusztrálja az alábbi öt ábra egy véletlenül generált feladatra:



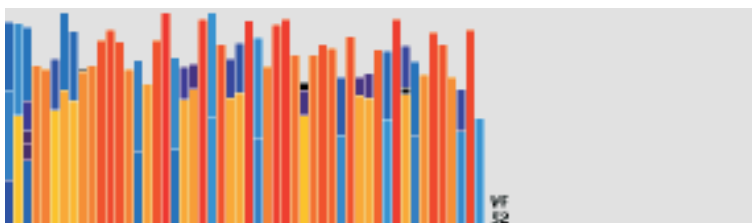
A first fit eredménye:



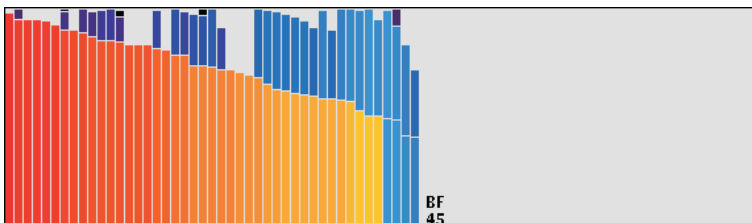
A best fit eredménye:



A worst fit eredménye:



És végül a rendezés utáni best fit adta pakolás:



A LEGRÖVIDEBB ÚT PROBLÉMA

A *legrövidebb út feladata* hálózati problémák közé tartozik. Ezekben egy *irányított gráffal* megadott lehetőségek közül az ezekhez köthető optimális folyamatokat akarjuk meghatározni. A hálózati feladatokkal kapcsolatban mindig feltesszük, hogy a hálózat minden élének van hossza. Egy *gráfot*, vagy *hálózatot* két halmazzal adhatunk meg. Az első a *csúcsokat*, a másik az ezekből álló párokat, az *irányított éleket* határozza meg. Egy adott hálózatban megjelölhetünk *kezdőpontot* és *végpontot* is.

Láncnak nevezzük éleknek egy olyan sorozatát, amelyben az egymást követő bármely két élnek egy közös csúcsa van.

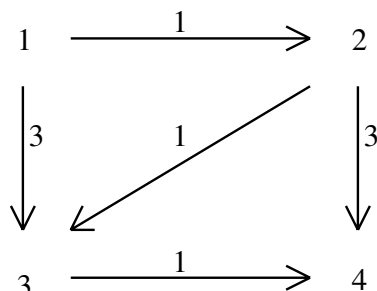
Az *út* egy olyan lánc, amelyben az utolsó él kivételével mindegyik él végpontja a sorozatban következő él kezdőpontja. Az (1,2), (2,3) és (4,3) élek láncot adnak, de az nem út. Út és lánc viszont az (1,2), (2,3) és (3,4) élek sorozata.

A legrövidebb út problémája egy hálózatban egy adott csúcsból kiindulva a többi csúcsba vezető legrövidebb út meghatározását jelenti. Ennek megoldására alkalmas *Dijkstra algoritmusa* amennyiben minden él hossza nemnegatív:

- Lássuk el az első csúcsot az állandó 0 címkével.
- Minden olyan i csúcsot lássunk el ideiglenesen az $(1, i)$ él hosszával mint címkével, amelyhez vezet él az 1 csúcsból. Minden más csúcs (az első kivételével) kapja ideiglenesen a ∞ címkét. A legkisebb ideiglenes címkéhez tartozó egyik csúcs címkéjét állandónak minősítjük.
- Tegyük fel, hogy az i volt az utolsó, a $(k+1)$. csúcs, amely állandó címkét kapott. Akkor i a k -adik legközelebbi csúcs az elsőhöz. Az ideiglenes címkével rendelkező j csúcsok címkéit módosítsuk az $(i$ címkéje + az (i, j) távolsága) értékre, ha ez kisebb, mint j eddigi ideiglenes címkéje. Ezután ismét adjunk végleges címkét egy olyan csúcsnak, amelynek címkéje a maradék ideiglenes címkék legkisebbike.
- Folytassuk az eljárást amíg minden csúcs állandó címkét nem kap.
- Ha minden csúcsnak végleges címkéje van, akkor az 1. csúcsból egy j csúcsba vezető legrövidebb utat úgy kapjuk, hogy a j csúcsból visszafelé haladva olyan csúcsokon keresztül jutunk el az 1. csúcsba, amelyektől a rákövetkezőbe vezető él hossza épp a két címke különbsége.

PÉLDA DIJKSTRA ALGORITMUSÁRA

Tekintsük azt az egyszerű legrövidebb út feladatot, amelyben négy városunk van: 1, 2, 3 és 4, és a köztük levő távolságot a következő ábra adja:



Ez kb. egy olyan elhelyezésnek felel meg, amikor a városok egy álló téglalap csúcsaiban vannak, de az 1 - 4 átló nem járható.

A Dijkstra algoritmus első lépése az 1. városból indulva a következő címkézést adja:

$$[0^*, \infty, \infty, \infty],$$

ahol a $*$ azt jelzi, hogy az illető címke végleges.

A következő lépésben meghatározzuk az első várostól mért távolságok alapján annak szomszédainak ideiglenes címkéjét:

$$[0^*, 1, 3, \infty].$$

Az új, ideiglenes címkék közül véglegesítjük a legkisebbet:

$$[0^*, 1^*, 3, \infty].$$

Képezzük most a végleges címkéjű városoktól mért távolságok alapján az új, javított címkéket:

$$[0^*, 1^*, 2, 4].$$

Itt a harmadik címke kettes értéke úgy adódott, hogy a kettes város végleges címkéje + a második és a harmadik város távolsága kisebb mint a korábbi ideiglenes címke értéke, a három. Kössük le ismét az ideiglenes címkék közül a legkisebbet:

$$[0^*, 1^*, 2^*, 4].$$

Az utolsó ideiglenes címkét ismét lehet javítani, és az ezután már végleges is lesz:

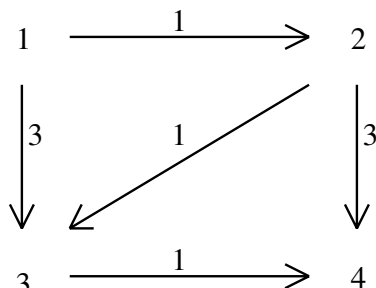
$$[0^*, 1^*, 2^*, 3^*].$$

Ebből az első és negyedik város közti legrövidebb út 3 hosszú: 1 - 2 - 3 - 4.

A MAXIMÁLIS FOLYAM PROBLÉMA

Egyes döntési helyzetekben olyan hálózatot kell vizsgálni, amelyben az éleknek kapacitások feleltethetők meg. A kérdés az, hogy az egyik kitüntetett csúcsból, a forrásból egy másikba, a nyelőbe mi a maximális eljuttatható mennyiség a hálózat és a kapacitások figyelembevételével. Ezt a feladatot nevezik a *maximális folyam problémának*.

Tekintsük a korábbi hálózatot úgy, hogy az élekre írt számok a kapacitásokat jelentik, az egyes csúcs a *forrás* és a *nyelő*:



A későbbi eljárás kedvéért vezessünk be egy *mesterséges élet* a nyelőtől a forrásig. A feladat lineáris programozási feladatként való megfogalmazásához jelölje az x_{ij} változó az (i, j) élen áthaladó anyagmennyiséget.

Egy lehetséges folyamot kapunk például a következő változó értékekkel:

$$x_{12} = 1, \quad x_{13} = 1, \quad x_{23} = 0, \quad x_{24} = 1, \quad \text{és} \quad x_{34} = 1.$$

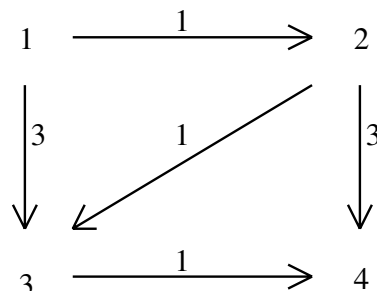
Az ehhez tartozó teljes átfolyó mennyiség 2. A lehetséges folyamoknak eleget kell tenniük a következő két feltételnek:

1. minden éltre az élen átmenő folyam nemnegatív és nem nagyobb, mint a megadott *élkapacitás*, és
2. minden csúcsra igaz az, hogy a *bejövő folyam* mennyisége megegyezik a *kimenő folyam*ével.

Az utóbbi feltételt *folyammegőrzési feltételnek* nevezzük. A probléma lineáris programozási feladatként való megfogalmazásában a fenti feltételek mellett az x_0 célfüggvényt kell maximalizálni, ahol x_0 a nyelőből a forrásba vezető mesterséges élen átmenő folyam mértéke.

A MAXIMÁLIS FOLYAM PROBLÉMA II.

Az említett



példára vonatkozó lineáris programozási feladat ez alapján:

$$\max x_0 = x_{24} + x_{34},$$

feltéve, hogy

$$\begin{aligned} x_{12} &\leq 1, \\ x_{23} &\leq 1, \\ x_{13} &\leq 3, \\ x_{24} &\leq 3, \\ x_{34} &\leq 1, \\ x_0 &= x_{24} + x_{34}, \\ x_{13} + x_{23} &= x_{34}, \\ x_{12} &= x_{23} + x_{24}, \\ x_0 &= x_{12} + x_{13}. \end{aligned}$$

Ez egy hatváltozós lineáris programozási feladat 4 egyenlőség és 5 egyenlőtlenség feltétellel. Az ábráról könnyen leolvasható, hogy a korábban említett

$$x_{12} = 1, \quad x_{13} = 1, \quad x_{23} = 0, \quad x_{24} = 1, \quad \text{és} \quad x_{34} = 1$$

lehetséges megoldás egyben optimális is. Ez azon múlik, hogy a 4. csúcsba 2-nél nagyobb folyam nem folyhat be (figyelembe véve a 2. csúcsba bemenő folyamot).

Állapítsuk meg, hogy lehetséges megoldást könnyű megadni a maximális folyam problémához: ha minden élen nulla mennyiséget szállítunk, az megfelel a feltételeknek.

Jelöljük I -vel azon élek halmazát, amelyeken kisebb a jelenleg átmenő folyam az él kapacitásánál. Jelölje R azon élek halmazát, amelyeken a jelenlegi folyam pozitív, ez csökkenthető. Legyenek $i(x, y)$, illetve $r(x, y)$ a megfelelő változtatási korlátok.

FORD-FULKERSON ELJÁRÁS A MAXIMÁLIS FOLYAM MEGHATÁROZÁSÁRA

Tekintsük először a címkézési eljárást:

- 1. lépés.** Címkézzük meg a forrást.
- 2. lépés.** Címkézzük meg a csúcsokat és az éleket a forrásba vezető mesterséges él kivételével a következő szabályok szerint.
Ha az x csúcs már kapott címkét, de az y még nem, és $(x, y) \in I$, akkor címkézzük meg az y csúcsot és az (x, y) élt. Ekkor az (x, y) élt *előremenő* élnek hívjuk.
Ha az x csúcs már kapott címkét, de az y csúcs még nem, és $(y, x) \in R$, akkor címkézzük meg az y csúcsot és az (y, x) élt. Az utóbbit *hátramenő* élnek nevezzük.
- 3. lépés.** Folytassuk a címkézési eljárást, amíg a nyelő címkét nem kap, vagy már nem lehet további csúcsokat címkével ellátni.

Ha a címkézéssel elérjük a nyelőt, akkor lesz a forrás és a nyelő között egy címkézett élekből álló lánc. Jelölje ezt C . A C -beli éleken átmenő folyam alkalmas módosításával egyrészt megőrizhetjük a folyam lehetőségességét, másrészt növelhetjük a folyamat.

A Ford-Fulkerson módszer a fentiek alapján:

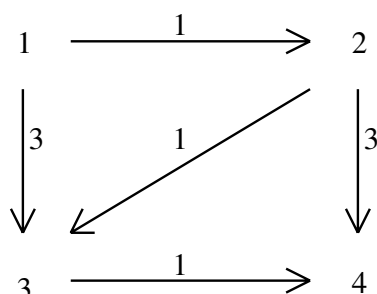
- 1. lépés.** Keressünk egy lehetséges folyamat (a minden élen 0 értékű folyam mindig lehetséges).
- 2. lépés.** A címkézési eljárással kíséreljük meg elérni a nyelőt. Ha nem lehet a nyelőt így megcímkézni, akkor az aktuális lehetséges folyam maximális. Ha elértük a nyelőt, akkor folytassuk az eljárást a 3. lépésnél.
- 3. lépés.** Határozzunk meg egy magasabb értékű lehetséges folyamat úgy, hogy a megcímkézett láncon az előremutató élek értékeit növeljük, a hátramutatókét pedig csökkentsük a

$$k = \min \left\{ \min_{(x,y) \in C \cap R} r(x, y), \min_{(x,y) \in C \cap I} i(x, y) \right\}$$

értékkel. Folytassuk az algoritmust a 2. lépéssel.

PÉLDA A FORD-FULKERSON ELJÁRÁS VÉGREHAJTÁSÁRA

Tekintsük ismét a



maximális folyam feladatot.

Az első lépés egy lehetséges folyam meghatározása. Legyen ez az, amelyik minden élhez a nulla folyamot rendeli hozzá.

Ezután kezdjünk egy címkézési eljárást. Először a forrás, az 1. csúcs kap címkét, majd ez alapján egy olyan él, ami ebből kivezet. Most ez esetben címkét kap a 2. csúcs, és az $(1, 2)$ él. Ezt folytatva a nyelőig tartó címkézett láncot (most utat) kapunk:

$$1 - 2 - 3 - 4.$$

Az előző lánc csak előremenő éleket tartalmaz. A lehetséges folyambővítési lehetőségek rendre:

$$i(1, 2) = 1, \quad i(2, 3) = 1 \quad \text{és} \quad i(3, 4) = 1.$$

Ez alapján az eddigi folyam a címkézett láncunk mentén 1-el növelhető. Ez alapján az ehhez tartozó lehetséges folyam értéke 1.

A lehetséges folyam javítása során azokat az éleket kell vizsgálni a címkézés során, amelyek kapacitását még nem merítettük ki. Ezek alapján már csak egy címkézhető láncot lehet képezni, amely eléri a nyelőt, ez az

$$1 - 3 - 2 - 4.$$

Ezen belül a $3 - 2$ él hátramutató, ennek értékét legfeljebb 1-el lehet csökkenteni. Az eddigi lehetséges folyam ismét 1-el növelhető. Ezután a lehetséges folyam:

$$x_{12} = 1, \quad x_{13} = 1, \quad x_{24} = 1, \quad \text{és} \quad x_{34} = 1.$$

Ennek a lehetséges folyamnak az értéke 2. Ezután látható, hogy további címkézett láncot már nem lehet képezni, tehát az előző lehetséges folyam egyben maximális is.

A MAXIMÁLIS FOLYAM PROBLÉMA ELMÉLETE

Legyen V' egy hálózat csúcsainak tetszőleges olyan halmaza, amely tartalmazza a nyelőt, de nem tartalmazza a forrást. Ekkor a hálózat olyan (i,j) éleinek halmazát, amelyek i kezdő-csúcsa nem V' -beli, a j végpont viszont V' -beli, a hálózat egy vágásának nevezzük. A vágás tehát élek egy olyan halmaza, amelyeket ha elhagyunk a hálózatból, akkor a forrásból a nyelő a továbbiakban már nem lesz elérhető.

A vágás kapacitása a vágást alkotó élekből a forrástól a nyelő felé vezetők kapacitásainak összege. A következő két segédétel adja meg a vágások és a maximális folyamok közti összefüggést.

SEGÉDTÉTEL. A forrásból a nyelőbe vezető folyamok erősségét bármelyik vágás kapacitása felülről korlátozza.

BIZONYÍTÁS. Tekintsük egy hálózat valamely V' vágását. A hálózat többi csúcsának halmaza legyen V . Válasszunk egy tetszőleges folyamot, legyen ennek értéke f , az (i,j) élen áthaladó mennyiséget pedig x_{ij} . Összegezzük a V -beli összes csúcsra a folyammegőrzési feltételeket. Ez azt adja, hogy mivel kiesnek az olyan élekre vonatkozó tagok, amelyek mindkét végpontja V -ben van, ezért

$$\sum_{i \in V, j \in V'} x_{ij} - \sum_{i \in V', j \in V} x_{ij} = f.$$

Az ebben az egyenletben szereplő első összeg egyenlő a V' -nek megfelelő vágás kapacitásával. Mivel minden x_{ij} érték nemnegatív, így a második összeg is nemnegatív. Ebből adódik a segédétel állítása: a folyamok értéke legfeljebb annyi lehet mint a vágásoké. $\square$

A segédteletből az is következik, hogy bármely vágás értéke felső korlátja a forrásból a nyelőbe áramló maximális folyam értékének. Tehát ha találunk egy olyan lehetséges folyamot, amelynek értéke egyenlő egy vágás kapacitásával, akkor találtunk egy maximális folyamot. Ez egy fajta gyenge dualitást fejez ki.

A MAXIMÁLIS FOLYAM PROBLÉMA ELMÉLETE II.

Tekintsük most azt az esetet, amikor egy adott lehetséges folyamra vonatkozóan a címkézési eljárás nem tudja elérni a nyelőt. Legyen ekkor a címkézetlen csúcsok halmaza által meghatározott a vizsgált vágás.

SEGÉDTÉTEL. Ha a címkézési eljárás nem tudja elérni a nyelőt, akkor a kimaradó, nem címkézett csúcsok által meghatározott vágás kapacitása megegyezik az aktuális folyam erősségével.

BIZONYÍTÁS. Legyen a korábbi jelöléseknek megfelelően V az aktuális folyam mellett megcímkézett csúcsok halmaza, és V' pedig a címkézetleneké. Tekintsünk egy (i, j) élet a vágásból, úgy, hogy $i \in V$ és $j \in V'$. Ekkor az x_{ij} értéknek egyenlőnek kell lennie az (i, j) él kapacitásával, hiszen különben tovább tudtunk volna haladni egy megfelelő előremenő éllel, és akkor j nem a V' -be tartozna.

Tekintsünk ezután egy olyan (i, j) élt, amelyre $i \in V'$ és $j \in V$. Ekkor viszont $x_{ij} = 0$ kell hogy teljesüljön, hiszen különben a címkézési eljárásban egy hátramenő éllel el tudtuk volna érni az i csúcsot, és így az nem tartozhatna a V' halmazba.

Az aktuális lehetséges folyamra tehát az teljesül az előző segédtételben igazolt

$$\sum_{i \in V, j \in V'} x_{ij} - \sum_{i \in V', j \in V} x_{ij} = f$$

egyenlet alapján, hogy a jelen vágás kapacitása egyenlő a $\sum_{i \in V, j \in V'} x_{ij}$ összeggel, és az adott folyam erősségével. $\square$

Az eddigi megállapítások szerint tehát amikor a nyelőt nem lehet megcímkézni a forrásból indulva, akkor az aktuális lehetséges folyam egy maximális folyam. Ez összhangban van a Ford-Fulkerson módszerrel.

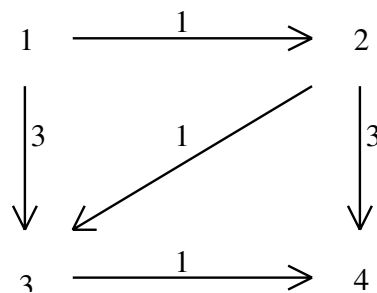
PROJEKTEK ÜTEMEZÉSE, CPM, PERT

Az összetett munkafolyamatok rögzített befejezési időponttal való teljesítése gondos tervezőmunkát igényel. Ennek része az összefüggő események sorrendjének, időzítésének vizsgálata hálózati modellek segítségével.

Erre a célra két eljárást szokás használni. Ha az egyes munkafolyamatok végrehajtási ideje biztosan tudható, akkor a *kritikus út módszer* (Critical Path Method, CPM), míg ha a tevékenységek időtartama bizonytalan, akkor a *program kiértékelési és felülvizsgálati technika* (Program Evaluation and Review Technique, PERT) használatos. Mindkét eljárást az ötvenes években fejlesztették ki. Számos nagy és kritikus projekt tervezésekor használták ezeket a módszereket, pl. nagy szoftver rendszerek határidős kidolgozásánál, úrkutatási projektekben, vagy épp rakétaindítások visszaszámlálási eljárásának kidolgozásában.

Mindkét eljáráshoz szükség van a projektet alkotó tevékenységek listájára. A projektet akkor tekintjük befejezettnek, ha minden részfeladata befejeződött. Minden tevékenységnek lehetnek *előzményei*, olyan munkafolyamatok, amelyeknek előbb be kell fejeződni ahhoz, hogy az adott tevékenység elkezdődhessen. A munkafolyamat lépéseinek ilyen összefüggését egy projekt-hálózattal adjuk meg.

A tevékenységeket a hálózat gráfjának irányított élei definiálják, a csúcsok pedig a tevékenységek csoportjainak befejezését jelzik. A csúcsokat emiatt *eseménynek* is nevezzük. Az ilyen projekt-hálózatot AOA (Activity On Arc) hálózatnak nevezzük. Ennek illusztrálásához tekintsük ismét a korábbi ábránkat:



Ezen most az 1. csúcs jelzi a projekt kezdetét, a 4. a *befejezés csúcs*. A 2. csúcs az egy hosszú első tevékenység végét, és a 3., illetve 4. csúcsokba vezető munkalépések kezdetét jelzi. Ha egy csúcsba több él is befut (mint pl. a 3. csúcsba), akkor ez azt jelenti, hogy mindkét korábbi tevékenységnek be kell fejeződni ahhoz, hogy az esemény utáni megkezdődhessen. Az előzmények nélküli tevékenységeket az első csúcsból kiinduló élekkel adjuk meg.

PROJEKTEK ÜTEMEZÉSE, CPM, PERT II.

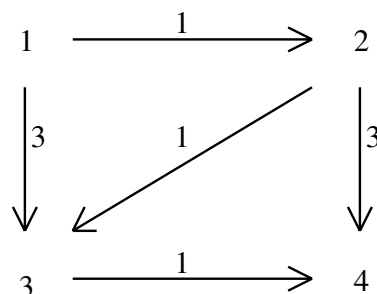
A projekt-hálózatot alakítsuk ki úgy, hogy egy tevékenység végét mutató csúcs sorszáma mindig nagyobb legyen, mint a kezdetét jelzőé. Ennek a szabálynak persze több reprezentáció is megfelel. További feltétel, hogy két adott csúcs között csak egy él mehet. Feltesszük még, hogy egy tevékenységet csak egy él reprezentálhat.

Az utóbbi két feltétel kielégítéséhez szükség lehet az ún. *fiktív tevékenységekre*. Például abban az esetben, amikor az *A* és *B* munkafolyamatok azonos feltétellel hajthatók végre, és mindkettő közvetlen előzménye a *C* tevékenységnek. Ilyen esetben a *B* lépést egy új csúcshoz köthetjük, ahonnan egy nulla időtartamú fiktív tevékenység új *D* éle adja a *C* munkafolyamat megkezdésének feltételét.

Tekintsük most a következő projekt feltételrendszert:

Tevékenység	Előzmények	Időtartam (nap)
A = anyagbeszerzés	–	1
B = alkalmazottak kiképzése	–	3
C = segédanyag termelése	A	1
D = válogatás és csomagolás	A	3
E = szakmunka	B, C	1

Ennek a projektnek épp a korábbi irányított éleket tartalmazó gráfunk felel meg az egyes csúccsal mint a projekt kezdetével, és a négyes csúccsal mint befejezés csúccsal:



Vegyük észre, hogy bár csak a *B* és *C* tevékenységeket tüntettük fel, mint az *E* munkafázis előfeltételét, de a hálózat összefüggései miatt az *A* folyamatnak is be kell fejeződnie az *E* megkezdése előtt.

CPM, A KORAI IDŐZÍTÉS MEGHATÁROZÁSA

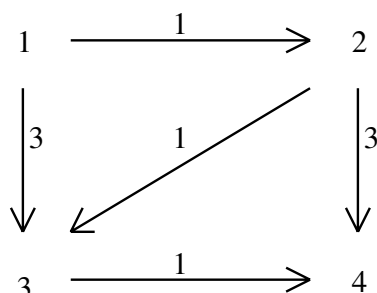
A CPM két kulcsfogalma az esemény *korai időzítése* (Early event Time, ET), és a *kései időzítés* (Late event Time, LT). Az i csúcs korai időzítése, $ET(i)$ az a legkorábbi időpont, amikor a csúcshoz tartozó esemény bekövetkezhet. Ennek megfelelően az i csúcs kései időzítése, $LT(i)$ az a legkésőbbi időpont, amikor a csúcshoz tartozó esemény bekövetkezhet anélkül, hogy a projekt előírt befejezési időpontját késleltetné. Meg lehet mutatni, hogy $ET(i)$ a kezdőponttól az i csúcsba vezető leghosszabb út hossza.

A projekt korai időzítésének kiszámolását az egyes csúccsal kezdjük, $ET(1) = 0$ adódik erre. Ebből kiindulva meghatározzuk a további $ET(i)$ értékeket az élek adta összefüggéseknek megfelelően. Ha egy csúcsba több él is befut, akkor a csúcs korai időzítése az előző összes részfolyamat teljesülését feltételezi.

Ez alapján az $ET(i)$ korai időzítés meghatározását a következő lépések adják:

1. lépés Keressük meg az i csúcsba befutó élek kezdő csúcspontjait. Ezek az események az i esemény *közvetlen előzményei*.
2. lépés Az i esemény minden közvetlen előzményének ET értékéhez adjuk hozzá az i -be vezető megfelelő élhez tartozó tevékenység időtartamát.
3. lépés $ET(i)$ egyenlő az előző lépésben számított összegek maximumával.

PÉLDA. Határozzuk meg a következő



projekt-hálózat korai időzítési értékeit! Definíció szerint $ET(1) = 0$.

Az $ET(2)$ érték közvetlenül az $ET(1) + 1$ összegből adódik, mert a 2. csúcsba csak egy él fut be.

$$ET(3) = \max\{ET(1) + 3, ET(2) + 1\} = \max\{0 + 3, 1 + 1\} = \max\{3, 2\} = 3.$$

$$ET(4) = \max\{ET(2) + 3, ET(3) + 1\} = \max\{1 + 3, 3 + 1\} = \max\{4, 4\} = 4.$$

Ez alapján a teljes projekt befejezésének legkorábbi időpontja 4. □

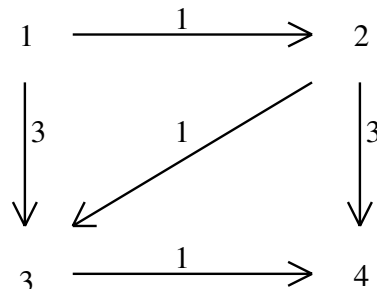
CPM, A KÉSEI IDŐZÍTÉS MEGHATÁROZÁSA

A kései időzítés meghatározását visszafelé, az utolsó, a befejezési csúcsból kiindulva kezdjük. Ebben az eljárásban ha egy i csúcsot több esemény követ, az utóbbiakra korábban számított LT értékekből le kell vonni az i csúcsból ezekhez vezető él hosszát. Az így kapott időpontok közül a legkorábbira teljesülnie kell az i eseménynek, hiszen csak ebben tartható a kitűzött határidő a teljes projekt teljesítésére.

Általában a befejezési csúcs kései időzítésének olyan időpontot szokás választani, amely alatt az mindenképpen elérhető az egyes csúcsból. Amennyiben az $LT(j)$ kései időzítési értékek mind ismertek a $j > i$ indexekre, az $LT(i)$ a következő eljárással számítható:

1. lépés Keressük meg azokat a csúcsokat, amelyekbe megy él az i csúcsból. Ezek az események az i esemény *közvetlen követői, utódai*.
2. lépés Az i esemény minden közvetlen utódának LT értékéből vonjuk le az i -ből az utódba vezető élhez tartozó tevékenység időtartamát.
3. lépés $LT(i)$ egyenlő az előző lépésben számított értékek minimumával.

PÉLDA. Határozzuk meg a kései időzítés értékeit a jól ismert irányított gráfunkra:



Legyen az $LT(4)$ érték 4, hiszen ekkorra a projekt befejezhető. Innen $LT(3) = 4 - 1 = 3$, hiszen a 3. csúcsnak csak a 4. az utódja. Ezután $LT(2) = \min\{LT(4) - 3, LT(3) - 1\} = \min\{4 - 3, 3 - 1\} = \min\{1, 2\} = 1$. Hasonlóan $LT(1) = \min\{LT(3) - 3, LT(2) - 1\} = \min\{3 - 3, 1 - 1\} = \min\{0, 0\} = 0$. Eszerint legkésőbb 4 időegységgel kell a projektet kezdeni a tervezett teljes készülségi esemény előtt. $\square$

Amennyiben egy i csúcsra $ET(i) = LT(i)$, akkor az illető csúcsban való minden késedelem a projekt határidejének lekésését jelenti. Esetünkben a $(2, 4)$ él 2-re változtatása azt eredményezné, hogy a 2. esemény 1 egységgel késletne a projekt befejezésének késedelme nélkül: ekkor $ET(2) = 1$, és $LT(2) = 2$.

CPM, A TŰRÉSHATÁR

A projekt tervezésekor fontos ismeret az, hogy az egyes tevékenységek mekkora késedelmre nem veszélyeztetni még a teljes projekt határidőre való befejezését. Egy tevékenység, illetve az ennek megfelelő (i, j) él *tűrőshatára* az a $TH(i, j)$ szám, amennyivel a tevékenység kezdete a legkorábbi lehetséges időponttól eltolódhat anélkül, hogy a projekt befejezése elkésne – a többi tevékenység pontos végrehajtását feltételezve.

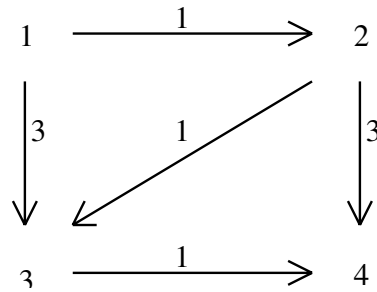
Legyen t_{ij} az (i, j) tevékenység hossza. Az (i, j) tevékenység késése mellett akkor tartható a projekt eredeti határideje, ha az i . esemény korai időzítése + az (i, j) tevékenység hossza + a k tűréshatár még mindig a j . esemény kései időzítésénél nem későbbi időpontot ad. Eszerint

$$ET(i) + t_{ij} + k \leq LT(j),$$

amiből a tűréshatárra

$$TH(i, j) = LT(j) - ET(i) - t_{ij}$$

adódik.



A példánkra adódó tűréshatárok:

$$TH(1, 2) = LT(2) - ET(1) - 1 = 1 - 0 - 1 = 0,$$

$$TH(1, 3) = LT(3) - ET(1) - 3 = 3 - 0 - 3 = 0,$$

$$TH(2, 3) = LT(3) - ET(2) - 1 = 3 - 1 - 1 = 1,$$

$$TH(2, 4) = LT(4) - ET(2) - 3 = 4 - 1 - 3 = 0,$$

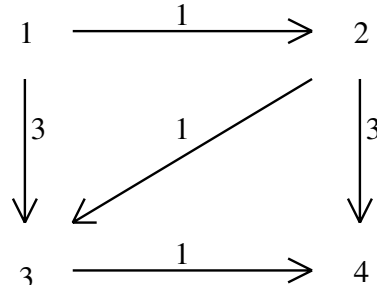
$$TH(3, 4) = LT(4) - ET(3) - 1 = 4 - 3 - 1 = 0.$$

Ennek megfelelően csak a $(2, 3)$ tevékenység halasztható (egy egységgel) anélkül hogy ez a teljes projekt csúszását okozná.

CPM, A KRITIKUS ÚT, MOZGÁSHATÁR

A nulla tűréshatárú tevékenységek bármilyen elhúzódása késlelteti a projekt befejezését. Ezért az ilyen éleket *kritikus tevékenység*nek nevezzük. A csak kritikus tevékenységekből álló, a kezdés csúcsból a befejezés csúcsba vezető utat *kritikus útnak* hívjuk.

A vizsgált példánkon az (1, 2) és (2, 4) tevékenységek például egy kritikus utat adnak meg, mert ezekre a tűréshatár nulla volt.



Természetesen egy projekt gráfjában több kritikus út is lehet. A tevékenységek flexibilitásának a tűréshatár mellett további mérőszáma a *mozgáshatár*:

Egy tevékenység, illetve az ezt reprezentáló (i, j) él mozgáshatára az az időtartam, amennyivel a tevékenység kezdete (vagy hossza) elhúzódhat anélkül, hogy ezzel bármely későbbi tevékenység kezdési időpontja a korábbi kezdési időpontjánál későbbre tolódna. A jelölése $MH(i, j)$.

A mozgáshatár meghatározását a tűréshatárhoz hasonlóan tehetjük meg: tegyük fel, hogy az i . esemény bekövetkezése, illetve az (i, j) tevékenység hossza k időegységnyit tolódik. Ebben az esetben a j esemény akkor nem csúszik az (i, j) tevékenység miatt, ha

$$ET(i) + t_{ij} + k \leq ET(j).$$

Ebből a mozgáshatár definíciója:

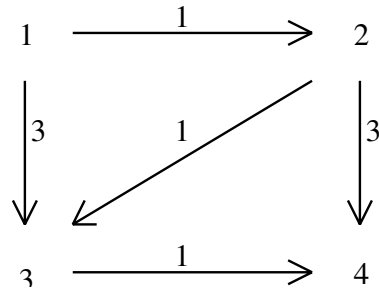
$$MH(i, j) = ET(j) - ET(i) - t_{ij}.$$

Mivel a példánkra az $ET(i)$ értékek megegyeznek a megfelelő $LT(i)$ értékekkel, ezért a mozgáshatárok is megegyeznek a korábbi tűréshatárokkal.

CPM, A KRITIKUS ÚT MEGHATÁROZÁSA

A kritikus út meghatározásának egyik lehetséges módja az ismertetett módszer a tűréshatár értékek kiszámításával, majd ezek ismeretében a nulla tűréshatárú élekből álló út megkeresése a kezdés csúcsból a befejezés csúcsig.

Emellett a kritikus út meghatározására lineáris programozási feladatot is fel lehet írni, amely tükrözi az eredeti probléma sajátosságait. Tekintsük példafeladatunkat ismét:



Jelölje x_i az i . esemény bekövetkeztének időpontját. Minden (i, j) tevékenységre érvényes, hogy a j esemény előtt be kell következnie az i -nek, és az (i, j) tevékenységnek is be kell fejeződnie. A kritikus út meghatározása a $z = x_F - x_1$ függvény minimalizálását jelenti, ahol F a befejezés csúcs. A példánkra adódó lineáris programozási feladat innen:

$$\min z = x_4 - x_1,$$

feltéve, hogy

$$x_2 \geq x_1 + 1,$$

$$x_3 \geq x_1 + 3,$$

$$x_3 \geq x_2 + 1,$$

$$x_4 \geq x_2 + 3,$$

$$x_4 \geq x_3 + 1.$$

A változókra érdemes nemnegativitási feltételt alkalmazni, sőt, $x_1 = 0$ természetes választás lehet. Az utóbbi feltételek nélkül az Excel a -2, -1, 1, 2 értékeket adta, és a kritikus út hosszára 4-et kaptunk. Ha x_1 értékét rögzítettük nullára, akkor az optimális megoldás megfelelően a 0, 1, 3, 4 értékekre módosult.

A kritikus út meghatározására felírt lineáris programozási feladatnak általában sok optimális megoldása van (főleg ha több tűréshatár pozitív).

CPM, A PROJEKT LERÖVIDÍTÉSE

Reális és gyakori feladat az, hogy egy projekt-hálózat ismert kritikus út megoldását módosítani kell úgy, hogy a teljes projekt időtartamát kell csökkenteni minimális költséggel – feltételezve, hogy minden tevékenység hossza csökkenthető egy ismert költség fejében.

Módosítsuk a példánkat annyiban, hogy minden tevékenység hossza csökkenthető féllal a következő költségek megfizetése esetén: 1, 2, 3, 4, 5. Legyenek az új változók, amelyek az egyes tevékenységek lerövidítésének mértékét adják, rendre A , B , C , D és E . A cél azt meghatározni, hogy hogyan kell ütemezni a projektet ahhoz, hogy az eredeti 4 hosszú végrehajtás 3.5-re módosuljon, és a tevékenységek gyorsításának összköltsége minimális legyen. Az ennek megfelelő lineáris programozási feladat:

$$\min z = 2A + 4B + 6C + 8D + 10E,$$

feltéve, hogy

$$A \leq 0.5, \quad B \leq 0.5, \quad C \leq 0.5, \quad D \leq 0.5, \quad E \leq 0.5,$$

$$x_2 \geq x_1 + 1 - A,$$

$$x_3 \geq x_1 + 3 - B,$$

$$x_3 \geq x_2 + 1 - C,$$

$$x_4 \geq x_2 + 3 - D,$$

$$x_4 \geq x_3 + 1 - E,$$

$$x_4 - x_1 \leq 3.5.$$

Itt $x_1 = 0$, és A, B, C, D, E nemnegatív. Az optimális megoldás

$$x_1 = 0, \quad x_2 = 0.5, \quad x_3 = 2.5, \quad x_4 = 3.5, \quad A = 0.5, \quad B = 0.5, \quad C = D = E = 0.$$

Ez azt jelenti, hogy az $(1, 2)$ és az $(1, 3)$ tevékenységeket kell postamunkában végezni.

PROGRAM KIÉRTÉKELÉS ÉS FELÜLVIZSGÁLAT, PERT

A CPM, a kritikus út módszere azt feltételezi, hogy a tevékenységek időtartama ismert. A projekt lerövidítési vizsgálat kivételével a munkafolyamatok hossza rögzített. Ezzel szemben a PERT megközelítés a tevékenységek időtartamában a bizonytalanság figyelembevételét is lehetővé teszi.

A PERT egy munkafolyamat időtartamáról három adatot vesz számításba:

- a = a tevékenység legrövidebb lehetséges időtartama,
- b = a tevékenység leghosszabb lehetséges időtartama, és
- m = a tevékenység időtartama legvalószínűbb értéke.

Jelölje a T_{ij} valószínűségi változó az (i, j) munkafolyamat időtartamát. A PERT megközelítés felteszi, hogy ezek a valószínűségi változók béta eloszlásúak. Amennyiben a T_{ij} valószínűségi változó béta eloszlást követ, akkor várható értéke és szórásnégyzete (varianciája):

$$E(T_{ij}) = \frac{a + 4m + b}{6},$$
$$\text{var } T_{ij} = \frac{(b - a)^2}{36}.$$

A PERT feltételezi azt is, hogy a munkafolyamatok időtartamai egymástól független valószínűségi változók. Ebben az esetben a projekt-hálózat egy útjának időtartama mint valószínűségi változó

$$\sum_{(i,j) \in \text{út}} E(T_{ij}),$$

az út teljes idejének varianciája pedig

$$\sum_{(i,j) \in \text{út}} \text{var } T_{ij}.$$

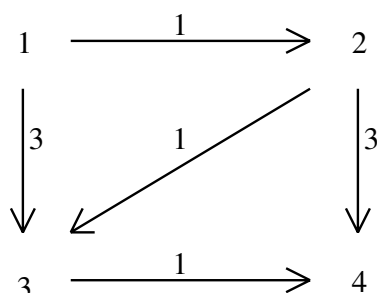
Jelölje most a CP valószínűségi változó a CPM által meghatározott kritikus út tevékenységeinek teljes időtartamát. A PERT feltételezi, hogy a kritikus útban elegendően sok tevékenység szerepel ahhoz, hogy alkalmazni lehessen a centrális határeloszlás tételt, és így megállapíthatjuk, hogy a

$$CP = \sum_{(i,j) \in \text{kritikus út}} T_{ij}$$

valószínűségi változó közelítőleg normális eloszlású.

PROGRAM KIÉRTÉKELÉS ÉS FELÜLVIZSGÁLAT, PERT PÉLDA

Tekintsük ismét a korábbi projekt-hálózatunkat:



Ehhez most meg kell adni az egyes munkafolyamatok időtartamának eloszlás-paramétereit:

Tevékenység	a	b	m
(1,2)	0.5	1.5	1.0
(1,3)	2.0	4.0	3.0
(2,3)	0.5	1.5	1.0
(2,4)	2.0	4.0	3.0
(3,4)	0.5	1.5	1.0

Vegyük észre, hogy a legvalószínűbb végrehajtási időeknek minden élre a korábbi rögzített értékeket választottuk. Ezekkel a számokkal meghatározhatjuk az egyes tevékenységek várható időtartamát és varianciáját:

$$E(T_{12}) = \frac{0.5 + 4 * 1 + 1.5}{6} = 1, \quad \text{var } T_{12} = \frac{(1.5 - 0.5)^2}{36} = \frac{1}{36} = 0.028,$$

$$E(T_{13}) = \frac{2 + 4 * 3 + 4}{6} = 3, \quad \text{var } T_{13} = \frac{(4 - 2)^2}{36} = \frac{4}{36} = 0.111,$$

$$E(T_{23}) = \frac{0.5 + 4 * 1 + 1.5}{6} = 1, \quad \text{var } T_{23} = \frac{(1.5 - 0.5)^2}{36} = \frac{1}{36} = 0.028,$$

$$E(T_{24}) = \frac{2 + 4 * 3 + 4}{6} = 3, \quad \text{var } T_{24} = \frac{(4 - 2)^2}{36} = \frac{4}{36} = 0.111,$$

$$E(T_{34}) = \frac{0.5 + 4 * 1 + 1.5}{6} = 1, \quad \text{var } T_{34} = \frac{(1.5 - 0.5)^2}{36} = \frac{1}{36} = 0.028.$$

Vegyük észre, hogy paraméterezésünk mellett a kapott várható értékek megegyeznek a korábbi rögzített időtartamokkal. A fiktív élekre a várható érték és a variancia is nulla lenne.

PROGRAM KIÉRTÉKELÉS ÉS FELÜLVIZSGÁLAT, PERT PÉLDA II.

Mivel a feltételezés szerint az egyes projekt szakaszok hosszai független valószínűségi változók, ezért a kritikus út teljes ideje várható értéke és annak szórásnégyzete a kritikus út szakaszaira kapott értékek összege lesz. Tekintsük az (1,2) és (2,4) szakaszokból álló kritikus utat:

$$\sum_{(i,j) \in \text{kritikus út}} E(T_{ij}) = E(T_{1,2}) + E(T_{2,4}) = 1 + 3 = 4,$$

a teljes idő varianciája pedig

$$\sum_{(i,j) \in \text{kritikus út}} \text{var } T_{ij} = \text{var } T_{12} + \text{var } T_{24} = 0.028 + 0.111 = 0.139.$$

Ugyanezeket az értékeket kapjuk a másik, (1,3), (3,4) kritikus útra is. A varianciából a megfelelő szórás $\sqrt{0.139} = 0.373$. Eszerint a kritikus utakon a várható értéktől való eltérés várható értéke ez a 0.373.

Feltételezve, hogy a teljes kritikus út megtételéhez szükséges idő normális eloszlású (ez példánkra aligha teljesülhet), meghatározhatjuk annak a valószínűségét, hogy a teljes projekt befejeződik adott idő, mondjuk 5 nap alatt.

Ez a valószínűség $P(CP \leq 5)$. Standardizáljuk a normális eloszlású valószínűségi változónkat:

$$P(CP \leq 5) = P\left(\frac{CP - 4}{0.373} \leq \frac{5 - 4}{0.373}\right) = P(Z \leq 2.681) = 0.996.$$

Itt az $F(2.681) = 0.996$ értéket a standard normális eloszlás táblázatából olvastuk ki². Itt $F(x)$ a standard normális eloszlás eloszlásfüggvénye. A PERT szerint tehát a megadott feltevések mellett annak a valószínűsége, hogy a teljes projekt 5 nap alatt befejeződik, 99.6%.

²Elérhető a jegyzet függelékében is, de az interneten is pl. a „normal distribution table” szavakra keresve.

PROGRAM KIÉRTÉKEELÉS ÉS FELÜLVIZSGÁLAT, PERT PÉLDA III.

A standard normális eloszlás megfelelő értékét a táblázat használata helyett megtudhatjuk:

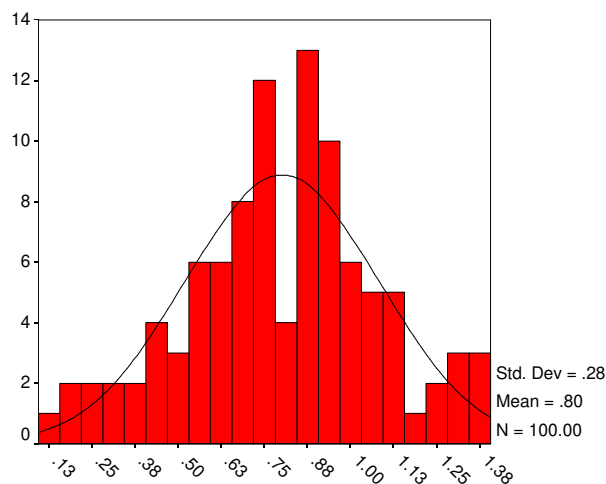
- az Excel `NORM.ELOSZL` függvényét használva,
- az SPSS nevű statisztikai program `CDF.NORMAL(2.681, 0, 1)` utasításával,
- a Matlab `0.5*erfc(-2.681/1.414)` parancsával (ehelyett persze a Matlab statisztikai csomagja `normcdf` utasítása a jobb megoldás – már ha az elérhető),
- a Maple pedig a `stats[statevalf, cdf, normald](2.681)` ; utasítással adja a kért valószínűségi értéket.

A PERT alkalmazása során tett feltevések nehezen teljesíthetők.

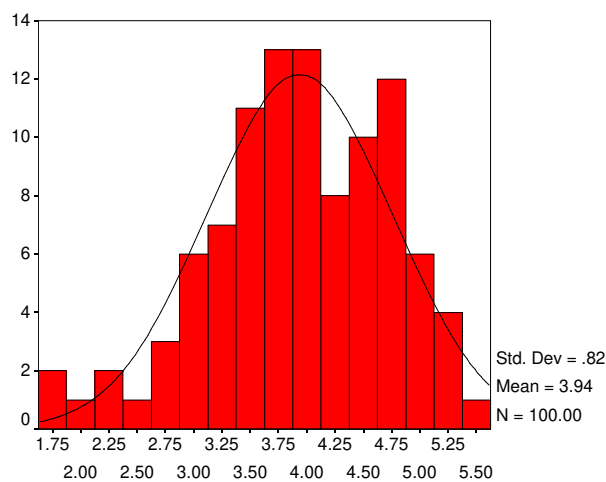
- Így nem könnyű igazolni, hogy az egyes tevékenységek időtartamai egymástól függetlenek.
- Sokszor a munkafolyamatok ideje nem béta eloszlást követ.
- A centrális határeloszlás tétel feltételei teljesüléséhez lehet, hogy nincs elegendő tevékenység a vizsgált útban.
- Gyakran előfordul, hogy a várható időtartamokra alkalmazott CPM eljárás eredménye nem marad kritikus út a véletlen események hatására.

Az utolsó probléma megoldására alkalmazhatunk Monte Carlo szimulációt annak meghatározására, hogy az egyes tevékenységek milyen valószínűséggel kritikusak, illetve hogy mennyi lehet a teljes projekt teljesülése idejének várható értéke és szórása.

BÉTA ELOSZLÁSÚ VALÓSZÍNŰSÉGI VÁLTOZÓK ÖSSZEGE



KETTO



TIZ

Itt az első ábra mutatja két béta eloszlással generált ($RV.BETA(2,3)$) véletlen szám összegének eloszlását az SPSS nevű program hisztogram rajzoló utasítása eredményeként. Be van rajzolva az illeszkedő normális eloszlás sűrűségfüggvénye is. A következő ábra tíz béta eloszlású véletlen szám összegének eloszlását mutatja. Mindkét megjelenített eloszlás eltér a normálistól.

SZTOCHASZTIKUS KÉSZLETMODELLEK, AZ ÚJSÁGÁRUS PROBLÉMA

Számos közgazdasági probléma fogalmazható meg, vagy vezethető vissza arra a feladatra, amikor azt szeretnénk meghatározni, hogy mekkora raktárkészletet tartsunk fenn, illetve rendeljünk meg, ha mind a raktáron tartásnak, mind a túl kicsi készletnek ismert költsége van. A döntéshozó célja nyilván a lehetséges legnagyobb nyereség elérése, illetve a költségei minimalizálása.

Az *újságárus problémának* azokat a feladatokat nevezzük amelyek elegettesznek a következő feltételeknek:

- Arról kell dönteni, hogy mennyi árut rendeljünk. Legyen ennek mennyisége q .
- A d egységnyi kereslet a beszerzett árura egy ismert, $p(d)$ valószínűséggel fordulhat elő. Itt d nemnegatív szám, és a D valószínűségi változó reprezentálja a keresletet.
- A d és q értékekre vonatkozóan egy $c(d, q)$ költség merül fel.

Egy újságárus valóban a fenti problémával szembesül. Ha az adott napi forgalmat alulbecsli, akkor egyes vevőket nem tud kiszolgálni, így lehetséges nyereségtől esik el. Másrészt ha túl sokat rendel, akkor annak a költségeit fizetnie kell, és az el nem adott példányok alapján nyilván nem jut nyereséghez. Ha a kereslet diszkrét valószínűségi változó, akkor a *diszkrét keresletű újságárus problémáról* van szó.

A költségfüggvény alakja arra az esetre, amikor a készlet nem kisebb, mint az igény ($d \leq q$):

$$c(d, q) = c_o q + a,$$

ahol c_o az *egységnyi túlkészletezés* költségét. Itt tehát c_o az a pozitív költség, amivel számolni kell akkor, ha a már így is túlzott készletet még egy egységgel növeljük. a a q -tól nem függő tagokat jelzi. A később bemutatott eljárás miatt ennek részletezése nem szükséges.

Abban az esetben, amikor a megrendelt áru mennyisége kisebb, mint az igény ($d \geq q + 1$), akkor a költségfüggvény:

$$c(d, q) = -c_u q + b,$$

ahol c_u a pozitív *egységnyi alulkészletezési költség*. Ez az az összeg, amivel a költségeinket csökkenteni tudjuk, ha egy egységgel több a készletünk. Itt b a q -tól nem függő tagokat jelzi.

AZ ÚJSÁGÁRUS PROBLÉMA II.

Az újságárus probléma elemzése során tekintsük most a költségfüggvény változását. Ha $E(q)$ a költségfüggvény várható értéke abban az esetben, ha q egységnyt rendeltünk az áruból, akkor a feladat olyan q^* optimális rendelés meghatározása, amely minimalizálja E értékét.

Amennyiben az $E(q)$ konvex függvény, akkor elegendő az $E(q+1) - E(q)$ értékeket meghatározni. Konvex függvény esetén ugyanis a feladatunk annak a legkisebb q -nak a megkeresésére egyszerűsödik, amelyre $E(q+1) - E(q)$ pozitív. Azt az eljárást, amely ez alapján az $E(q+1) - E(q)$ ismételt kiértékelésével határozza meg a keresett optimális megrendelést, *határelemzésnek* nevezik. Alkalmazásának feltétele, hogy a célfüggvény konvex legyen, és hogy az említett különbség könnyen számítható legyen.

Az $E(q+1) - E(q)$ különbség meghatározásához két esetet kell megvizsgálni:

1. Amikor $d \leq q$. Ekkor egy újabb egység megrendelése további túlkészletezést okoz. Ez c_o -al növeli a költséget. Ennek az esetnek a bekövetkezési valószínűsége $P(D \leq q)$, ahol D a kereslet valószínűségi változója.

2. A másik eset az, amikor $d \geq q+1$. Ekkor egy további egység megrendelése csökkenti a hiányt, és így csökkenti a költséget is c_u -val. A második eset bekövetkeztének valószínűsége $P(D \geq q+1) = 1 - P(D \leq q)$.

A fentiek alapján tehát az összes eset $100 \cdot P(D \leq q)$ százalékában a $q+1$ egység megrendelése c_o -val kerül többbe, mint q egység rendelése, és az esetek $100 \cdot (1 - P(D \leq q))$ százalékában a $q+1$ egységnyi készlet költsége c_u -val kevesebbe kerül, mint q egységnyié. Ezek alapján átlagosan a $q+1$ áruegység megrendelése

$$E(q+1) - E(q) = c_o P(D \leq q) - c_u (1 - P(D \leq q)) = (c_o + c_u) P(D \leq q) - c_u$$

költséggel kerül többbe, mint a q egység rendelése. Mivel a $P(D \leq q)$ valószínűség q növekedésével nő, ezért ha $c_o + c_u$ nemnegatív (ez az esetek többségében reális feltételezés), akkor a fenti különbség monoton nőni fog, így teljesül a határelemzés feltétele.

AZ ÚJSÁGÁRUS PROBLÉMA, PÉLDA

Ha $E(q+1) - E(q) \geq 0$, akkor $(c_o + c_u)P(D \leq q) - c_u \geq 0$ -ból $P(D \leq q) \geq c_u/(c_o + c_u)$ adódik. Ha $F(q) = P(D \leq q)$ a kereslet eloszlásfüggvénye, akkor azt a minimális q értéket keressük, amire még teljesül

$$F(q) \geq \frac{c_u}{c_o + c_u}.$$

Tekintsük azt a feladatot, amikor egy jegyzet kiadásakor a nyomtatott példányszámról kell dönteni. A hallgatóság létszáma ismeretében a várható igényeket többé-kevésbé jól lehet becsülni. A kérdés nyilván az, hogy hány példányban készüljön a jegyzet ahhoz, hogy például a három éven belüli teljes költség minimális legyen.

A konkrét szám adatok legyenek a következők: az előállítási költség 900 Ft, az eladási ár 1500 Ft. A harmadik év végén a teljes maradék készlettel megszabadulunk féláron: eladjuk őket egy nagykereskedőnek (750 Ft). A három éven belül eladható példányok valószínűsége:

eladott jegyzet	valószínűség
50	0.30
100	0.20
150	0.20
200	0.10
250	0.10
300	0.10

A jelölésünk kövesse a bevezetőt: legyen q a megrendelt jegyzetek száma, d pedig a három éven belül ténylegesen eladottak száma. Amennyiben az eladott példányok száma kevesebb, mint a megrendeltké ($d < q$), akkor a teljes költség a következők szerint alakul:

tevékenység	költség
q darab jegyzet vásárlása	$900 q$
d darab jegyzet eladása	$-1500 d$
$q - d$ jegyzet eladása féláron	$-750 (q - d)$
teljes költség	$150 q - 750 d$

A további vizsgálatunk szempontjából ebből az a fontos, hogy a q egységnyi növelése 150 Ft-nyi költségnövekedést jelent (hiszen 900-ért állítják elő a jegyzetet, és a fölös példányoktól csak 750-ért tudunk megszabadulni).

AZ ÚJSÁGÁRUS PROBLÉMA, PÉLDA II.

Vegyük észre, hogy a jegyzet előállításának a példányszámtól nem függő költségei az optimális példányszámot nem befolyásolják, hiszen ezeket minden q -ra azonos módon kell megfizetni.

Amennyiben az igényelt példányok száma legalább annyi, mint a megrendeltéké ($d \geq q$), akkor a teljes költség a következők szerint alakul (a negatív költséget nyereségként értelmezzük):

tevékenység	költség
q darab jegyzet vásárlása	$900 q$
q darab jegyzet eladása	$-1500 q$
teljes költség	$-600 q$

Ebben az esetben a q egységnyi növelése a költségeket 600 forinttal csökkenti (a nyereséget 600 Ft-al növeli). Az eddigi költségelemzés alapján a túlkészletezési költség $c_o = 150$, az alulkészletezési költség pedig $c_u = 600$.

Alkalmazzuk most az optimális megrendelésre vonatkozó levezetett feltételt:

$$F(q) \geq \frac{c_u}{c_o + c_u} = \frac{600}{150 + 600} = \frac{600}{750} = 0.8.$$

A megadott valószínűségek alapján példánkban 200 darab jegyzet megrendelése az optimális, mert erre adódik először a kapott $0.3+0.2+0.2+0.1 = 0.8$ eloszlásfüggvény érték.

A kapott eredmény illusztrálásaként határozzuk meg az egyes költség eltéréseket

$$E(q+1) - E(q) = (c_o + c_u)P(D \leq q) - c_u$$

alapján az egymásra következő eltérések rendre:

$$E(51) - E(50) = (150 + 600) \cdot 0.3 - 600 = 225 - 600 = -375,$$

$$E(101) - E(100) = 750 \cdot 0.5 - 600 = 375 - 600 = -225,$$

$$E(151) - E(150) = 750 \cdot 0.7 - 600 = 525 - 600 = -75,$$

$$E(201) - E(200) = 750 \cdot 0.8 - 600 = 600 - 600 = 0,$$

$$E(251) - E(250) = 750 \cdot 0.9 - 600 = 675 - 600 = 75,$$

$$E(301) - E(300) = 750 \cdot 1.0 - 600 = 750 - 600 = 150.$$

AZ ÚJSÁGÁRUS PROBLÉMA, FOLYTONOS KERESLET

Bizonyos értelemben az előző példában is feltételeztük a kereslet finomabb felbontását, mint az épp definiáltat. Mégis, az az eset külön tárgyalandó, amikor a D keresletet egy folytonos valószínűségi változóval reprezentáljuk. Az előző modellre használt határelemzési eljárást ennek megfelelően módosítani kell.

Eszerint a döntéshozó várható költségét az a q^* megrendelés minimalizálja várható értékben, ahol q^* az a legkisebb megrendelési érték, amelyre teljesül

$$P(D \leq q) = \frac{c_u}{c_o + c_u}.$$

A feltételben az egyenlőséget az tette lehetővé, hogy a kereslet most folytonos valószínűségi változó. Egyszerűen belátható, hogy a fenti feltétel ekvivalens az

$$P(D \geq q) = \frac{c_o}{c_o + c_u}$$

egyenlettel.

PÉLDA. A légitársaságok nemrég még bevett gyakorlatavolt, hogy a legnagyobb lehetséges nyereség elérése érdekében több jegyet adtak el, mint ahány utas az adott gépre felfér – arra számítva, hogy nem minden utas fog ténylegesen utazni, egyesek lemondják az utat különböző okok miatt. Vizsgáljuk meg az újságáros probléma modelljével, hogy egy adott esetben hogyan határozható meg az optimális *túlfoglalás*.

A Fokker F70 gép 79 utast tud szállítani. Tegyük fel, hogy egy járatra a jegy ára 40 eFt, a túlfoglalás miatt a gépről lemaradó utas kárpótlás és egy másik járat drágább ára miatt 20 eFt többletköltséget jelent (és visszatérítik a teljes jegyárat). A tapasztalatok szerint a jeggyel rendelkező, de meg nem jelent utasok száma közel normális eloszlást követ 10 várható értékkel és 3 szórással.

Legyen most q a légitársaság által az adott járatra eladott jegyek száma, d pedig a meg nem jelent utasok száma (ez eltér a korábban szokásos jelentéstől). Ekkor $q - d$ lesz a ténylegesen utazásra jelentkezők száma. Ha $q - d \leq 79$, akkor mindenki utazhat, és ekkor a légitársaság költsége $-40(q - d)$ (ezer Forintban). Ha $(q - d) > 79$, akkor 79 utas lesz a gépen (ennek költsége $-79 \cdot 40$), és $q - d - 79$ utas kap kárpótlást fejenként 20 eFt értékben. Ekkor tehát a légitársaság teljes költsége $20(q - d - 79) - 40 \cdot 79 = 20q - 20d - 60 \cdot 79 = 20q - 20d - 4740$.

A FOLYTONOS KERESLETŰ ÚJSÁGÁRUS PROBLÉMA, PÉLDA

Amennyiben $q - 79$ a döntési változónk, akkor egy folytonos keresletű újságárus problémát kell megoldani. A fenti adatok alapján $c_u = 40$, és $c_o = 20$. Az optimális döntés feltétele:

$$P(D \leq q - 79) = \frac{c_u}{c_o + c_u} = \frac{40}{20 + 40} = \frac{2}{3}.$$

Standardizáljuk a D valószínűségi változónkat a 10 várható érték és a 3 szórás felhasználásával:

$$P\left(\frac{D - 10}{3} \leq \frac{q - 79 - 10}{3}\right) = 0.6.$$

Bevezetve a $Z = (D - 10)/3$ standard normális eloszlású valószínűségi változót, optimalitási feltételnek azt kapjuk, hogy

$$P\left(Z \leq \frac{q - 79 - 10}{3}\right) = 0.6.$$

A standard normális eloszlás táblázatából megtudhatjuk, hogy $P(Z \leq 0.43) = 0.6664$ (és a következő értékre, 0.44-re már 0.67 valószínűség adódik). Innen a közelítő optimális megoldás

$$0.43 = \frac{q - 79 - 10}{3},$$

azaz

$$q = (0.43 \cdot 3) + 89 = 90.29.$$

Eszerint a légitársaságnak az adott feltételek mellett a legelőnyösebb 90 vagy 91 jegyet eladnia a 79 tényleges férőhelyre. Természetesen ha az igény ennél kisebb, akkor annyi jegyet érdemes kiadni. $\square$

Ahogy az újságáros probléma példáján is látszott, egy *sztochasztikus programozási probléma* szokás szerint egy alapul szolgáló *determinisztikus feladatra* épül. Miután kiderült, hogy valamely változó valójában egy valószínűségi változóval adható meg, egy újabb, *sztochasztikus optimalizálási modellt* fogalmazunk meg.

A *modell* és *feladat* szavakat szinonimaként használhatjuk. Szigorúbb értelemben a modell adja meg a feladatra vonatkozó feltételezéseket, és határozza meg azokat a matematikai objektumokat, amik a rendszerünk paramétereinek megfelelnek. Ez alapján aztán felírhatjuk a konkrét megoldandó feladatot, majd az eredményt alkalmazhatjuk leírási vagy működtetési céljainkra.

Abban az esetben, amikor a modellre vonatkozó döntést a véletlen esemény előtt kell meghozni, *statikus modellről* beszélünk. Ide tartozik az újságáros probléma is. Ezzel szemben *dinamikus modellnek* nevezünk olyan modelleket, amelyek olyan rendszerekre vonatkoznak, amelyek állapotaikat időben változtatják. Amennyiben a rendszer viselkedését nem befolyásolják véletlen folyamatok, akkor az *optimális vezérlést* a rendszer indulása előtt meg lehet határozni.

A *sztochasztikus dinamikus rendszer* esetén az optimális működtetéshez szükség van a valószínűségi változók aktuális értékeinek megfigyelésére, és a rendszer működésére való hatásuk megállapítására.

Az alapul vett determinisztikus feladatokra a következő két példa szolgál kiindulási pontként:

Meghatározandó olyan x , hogy

$$g_1(x, \xi) \geq 0, \quad g_2(x, \xi) \geq 0, \quad \dots, \quad g_r(x, \xi) \geq 0,$$

$$x \in D,$$

ahol D egy adott, nem véletlen halmaz, amit gyakran véges sok x -re vonatkozó egyenlőtlenség határoz meg. A ξ szimbólum olyan vektort jelöl, amelynek komponensei majd valószínűségi változók lesznek.

SZTOCHASZTIKUS PROGRAMOZÁSI MODELLEK II.

A második determinisztikus feladat egy szélsőérték keresés:

$$\min h(x, \xi)$$

feltéve, hogy

$$g_1(x, \xi) \geq 0, \quad g_2(x, \xi) \geq 0, \quad \dots, \quad g_r(x, \xi) \geq 0,$$

$$x \in D,$$

ahol D ismét egy adott, nem véletlen halmaz, amit gyakran véges sok x -re vonatkozó egyenlőtlenség határoz meg. A ξ szimbólum itt is olyan vektort jelöl, amelynek komponensei majd valószínűségi változók lesznek.

Ezeknek a feladatoknak fontos speciális esetei az alábbi problémák:

Meghatározandó olyan x , hogy

$$Tx \geq \xi, \quad (\text{vagy} \quad Tx = \xi),$$

$$Ax = b \quad \text{és} \quad x \geq 0,$$

illetve az, hogy

$$\min h(x, \xi)$$

feltéve, hogy

$$Ax = b \quad \text{és} \quad x \geq 0,$$

ahol esetleg nem csak ξ , de a T mátrixok egyes elemei is véletlen valószínűségi változók.

A legelső determinisztikus problémának felel meg például a következő *valószínűség maximalizálási feladat*:

$$\max P(g_1(x, \xi) \geq 0, \quad g_2(x, \xi) \geq 0, \quad \dots, \quad g_r(x, \xi) \geq 0),$$

úgy, hogy $x \in D$.

KORLÁTOZÁS NÉLKÜLI NEMLINEÁRIS OPTIMALIZÁLÁS

A *korlátozás nélküli nemlineáris optimalizálási feladatok* gyakorlati problémákban gyakran lépnek fel. Általános alakjuk

$$\min f(x),$$

ahol a célfüggvény kétszer folytonosan differenciálható ($f \in C^2$), $f : \mathbb{R}^n \rightarrow \mathbb{R}$. A megoldás egyik módszere az, hogy az eredeti nemlineáris függvényt a megoldáshoz tartó pontokban kvadratikus függvényekkel közelítjük, és a következő iterált a közelítésből kapott megoldás lesz.

A feladat megoldásához *helyi kereső eljárásokat* szokás használni, amelyek az indulópont-hoz tartozó *helyi minimumpont* megkeresésére vállalkoznak, általában monoton nem növekvő célfüggvényérték mellett.

Amennyiben a második derivált, a $H_{ij}(x) = \partial^2 f(x) / \partial x_i \partial x_j$ képlettel adott Hesse mátrix ismeretes, akkor a leghatékonyabb a *Newton módszer*. Ez az *érintő-módszer* alapján működik, ami egydimenziós nemlineáris egyenletet old meg az $x_{k+1} = x_k - f(x_k) / f'(x_k)$ iterációs képlettel. A többdimenziós optimalizálási feladatra ennek a következő formula felel meg:

$$x_{k+1} = x_k - H^{-1}(x_k) \nabla f(x_k),$$

ahol $\nabla f(x_k)$ az $f(x)$ függvény gradiense az x_k pontban.

A korszerű számítógépes megvalósításokban a megadottnál kisebb lépést szokás tenni. Az ilyen Newton módszerre bizonyos feltételek teljesülése esetén *kvadratikus konvergencia* érvényes, azaz egy megfelelő x^* helyi minimumpontra

$$\|x^* - x_{k+1}\| \leq C \|x^* - x_k\|^2$$

érvényes egy alkalmas pozitív C konstansra.

Gyakran a Hesse mátrix nem állítható elő egyszerűen, vagy pedig numerikus differenciálással jól közelíthető. Az erre az esetre módosított Newton módszernek *quasi-Newton eljárás* a neve. Erre kvadratikus konvergencia már nem érvényes, csak a *szuperlineáris konvergencia*, azaz teljesül

$$\lim_{k \rightarrow \infty} \frac{\|x^* - x_{k+1}\|}{\|x^* - x_k\|} = 0.$$

GRADIENS MÓDSZER

Amennyiben csak a gradiensértékre lehet támaszkodni, akkor olyan eljárást is fel lehet építeni, amely az adott iterációs pontból a negatív gradiens irányában lép tovább:

$$x_{k+1} = x_k - \lambda \nabla f(x_k),$$

ahol λ a lépésköz. Az olyan módszereket, amelyek keresési iránya a negatív gradienssel pozitív belső szorzatot ad, *gradiens módszereknek* nevezzük.

A gradiens módszernek is vannak olyan változatai, amelyek nem igénylik a célfüggvény deriváltjának ismeretét, ennek megfelelő közelítését maga az eljárás állítja elő.

A gradiens módszer családjában általában csak lineáris konvergenciát mutat, de a konjugált gradiens módszerrel bizonyos feladatosztályon el lehet érni a szuperlineáris konvergenciát.

PÉLDA. Tekintsük az $f(x) = (x_1 - 1)^2 + (x_1 + x_2 - 2)^2$ függvényt. Ennek a célfüggvénynek a minimuma az $x_1 = 1$, $x_2 = 1$ pontban van, értéke 0. A gradiens

$$\nabla f(x) = (2(x_1 - 1) + 2(x_1 + x_2 - 2), 2(x_1 + x_2 - 2))^T,$$

a Hesse mátrix és inverze pedig

$$H(x) = \begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}, \quad \text{illetve} \quad H(x)^{-1} = \begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 1.0 \end{bmatrix}.$$

Ennek alapján a Newton módszer adta iteráció az $x_0 = (3, 3)^T$ pontból indulva:

$$x_1 = x_0 - H^{-1}(x_0) \nabla f(x_0) = \begin{pmatrix} 3 \\ 3 \end{pmatrix} - \begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 1.0 \end{bmatrix} \begin{pmatrix} 2 \cdot 2 + 2 \cdot 4 \\ 2 \cdot 4 \end{pmatrix}.$$

Innen

$$x_1 = \begin{pmatrix} 3 \\ 3 \end{pmatrix} - \begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 1.0 \end{bmatrix} \begin{pmatrix} 12 \\ 8 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix} - \begin{pmatrix} 6 - 4 \\ -6 + 8 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix} - \begin{pmatrix} 2 \\ 2 \end{pmatrix}.$$

Tehát ez alkalommal egy lépésben megkaptuk az $(1, 1)^T$ optimális megoldást. Ez a jelenség a lényegében kvadratikus függvényekre fordulhat elő.

GRADIENS MÓDSZER, PÉLDA

Tekintsük az előző példát, és oldjuk meg a gradiens módszerrel! Minimalizálandó tehát az $f(x) = (x_1 - 1)^2 + (x_1 + x_2 - 2)^2$ függvény. Ennek minimuma az $x_1 = 1$, $x_2 = 1$ pontban van, értéke 0. A gradiens

$$\nabla f(x) = (2(x_1 - 1) + 2(x_1 + x_2 - 2), 2(x_1 + x_2 - 2))^T.$$

Vegyünk egy viszonylag kis lépésközt, $\lambda = 0.1$ -et, és ismét az $x_0 = (3, 3)^T$ indulópontot. Az iterációs sorozat első lépései ezzel az

$$x_{k+1} = x_k - \lambda \nabla f(x_k)$$

képlet alapján:

$$x_1 = x_0 - \lambda \nabla f(x_0) = \begin{pmatrix} 3 \\ 3 \end{pmatrix} - 0.1 \begin{pmatrix} 12 \\ 8 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix} - \begin{pmatrix} 1.2 \\ 0.8 \end{pmatrix} = \begin{pmatrix} 1.8 \\ 2.2 \end{pmatrix}.$$

A következő iterált pontok a Matlab

```
» x = x - 0.1 * [ 2 * (x(1) - 1) + 2 * (x(1) + x(2) - 2) ; 2 * (x(1) + x(2) - 2) ]
```

utasításával kiszámítva:

$$x_2 = \begin{pmatrix} 1.24 \\ 1.80 \end{pmatrix}, \quad x_3 = \begin{pmatrix} 0.984 \\ 1.592 \end{pmatrix}, \quad x_4 = \begin{pmatrix} 0.8720 \\ 1.4768 \end{pmatrix}.$$

A következő néhány kiválasztott közelítő vektor:

$$x_{10} = \begin{pmatrix} 0.8359 \\ 1.2722 \end{pmatrix}, \quad x_{20} = \begin{pmatrix} 0.9245 \\ 1.1221 \end{pmatrix}, \quad x_{50} = \begin{pmatrix} 0.9930 \\ 1.0113 \end{pmatrix}, \quad x_{100} = \begin{pmatrix} 0.9999 \\ 1.0002 \end{pmatrix}.$$

Az iterált vektorok lineáris konvergenciára utalnak.

KONJUGÁLT GRADIENS MÓDSZER

A *konjugált gradiens módszer* optimalizálásra 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

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

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

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

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

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

Balról beszorozva s_k^T -vel, és megoldva ezt 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 $k = 1, 2, \dots$ értékekre az alábbi lépéseket, amíg a megállási feltételek nem teljesülnek:

1. $\alpha_k = (r_k^T r_k) / (s_k^T A s_k)$ (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. $\beta_{k+1} = (r_{k+1}^T r_{k+1}) / (r_k^T r_k)$ (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 gradiensmódszerek 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 megoldás felé. A konjugált gradiens módszer ezzel szemben a lépésenkénti megfelelő 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 gradiensekre és a Hesse mátrixra vagy ezek közelítésére támaszkodva).

A KONJUGÁLT GRADIENS MÓDSZER A MATLABBAN

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:

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

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.

EGYENLETRENDSZEREK ITERÁCIÓS MÓDSZEREI / MATLAB

A lineáris egyenletrendszerek megoldására szolgáló iterációs Matlab eljárásokat összegeztük az alábbi táblázatban:

függvény	mátrix típus	módszer
bicg	általános	bikonjugált gradiens módszer
bicgstab	általános	stabilizált bikonjugált gradiens módszer
cgs	általános	négyzetes konjugált gradiens módszer
gmres	általános	általánosított minimum-reziduál módszer
minres	Hermite-szimmetrikus	minimum-reziduál módszer
lsqr	általános	konjugált gradiens normális egyenletekre
pcg	Herm. poz. def.	prekondicionált konjugált gradiens
qmr	általános	kvázi-minimál reziduál módszer
symmlq	Hermite-szimmetrikus	szimmetrikus LQ módszer

Ezek a függvények (a gmres kivételével) azonos hívási formátumot használnak. A legegyszerűbb hívási mód az

$$x = \text{solver}(A, b),$$

ahol solver a táblázatban szereplő egyik eljárás neve. Ha a megállási feltételben a toleranciát módosítani szeretnénk, akkor a hívási forma

$$x = \text{solver}(A, b, tol),$$

ahol a tol érték az a szám, amellyel a $\text{norm}(b - A \cdot x) \leq tol \cdot \text{norm}(b)$ feltétel teljesülését követeljük meg. A tolerancia alapbeállítása $1e-6$.

Egy adott $n \times n$ -es A mátrix nemnulla elemei százalékos arányát a következő Matlab utasítás adja:

$$\gg \text{nnz}(A) / n^2$$

A gyakrabban használatos, érdekes mátrixok, vektorok közvetlenül is elérhetők a Matlabban:

```
» b = ones(n,1);
```

az egyesekből álló n hosszú oszlopvektort adja.

```
» A = gallery('wathen',12,12); n = length(A)
n =
    481
» nnz(A)/n^2
ans =
    0.0301
```

a 481×481 -es Wathen mátrixot generálja, amelynek rögzített ritkasági szerkezete van véletlen elemekkel. A nemnulla elemek aránya kb. 3%. A prekondicionált konjugált gradiens módszer a következő eredményt adja a fentiekben definiált lineáris egyenletrendszerre.

```
»x = pcg(A,b);
pcg stopped at iteration 20 without converging to the
desired tolerance 1e-006 because the maximum number of
iterations was reached.
The iterate returned has relative residual 0.063
```

Ez azt jelenti, hogy az előírt megállási feltétel nem teljesült még a reziduálra, több iteráció végrehajtását kell ehhez engedélyezni.

```
x = pcg(A,b,1e-6,100);
pcg converged at iteration 86 to a solution with relative
residual 8.8e-007
```

Nagyon tanulságos a 12 helyett nagyobb paraméterrel futtatni a fenti utasításokat. 40 esetén az n már közel 5000, a nem nulla mátrixelemek aránya 3 ezrelék. Erre a feladatra a `pcg` eljárás kb. 6 másodpercig futott, $x = b \setminus A$ pedig hétszer tovább.

Az iteratív eljárások a hatékony működéshez általában prekondicionálást igényelnek, az eredeti

$$Ax = b$$

egyenlet helyett az M_1 és M_2 mátrixokkal, illetve az $M = M_1 M_2$ mátrixszal a következő egyenleteket fogják használni:

$$M_1^{-1} A M_2^{-1} \cdot M_2 x = M_1^{-1} b,$$

vagy pedig

$$M^{-1} A x = M^{-1} b.$$

Az átalakítás célja az, hogy az eredménymátrix bizonyos értelemben közel legyen az egységmátrixhoz. A jó prekondicionáló mátrixok meghatározása nehéz feladat, és általában az adott alkalmazás ismeretét kívánja meg, amiből a lineáris egyenletrendszer származik.

Az általunk vizsgált A mátrixnak egy jó prekondicionálója az

$$M = \text{diag}(\text{diag}(A))$$

mátrix, az A mátrix főátlója elemeiből álló diagonális mátrix. Ezzel mint ötödik argumentummal felhívva a `pcg` eljárást, lényegesebben gyorsabban kapunk a megállási feltételnek megfelelő megoldást:

```
» [x,flag,relres,iter] = pcg(A,b,1e-6,100,diag(diag(A)));
» flag, relres, iter
flag =
    0
relres =
    9.0568e-007
iter =
    28
```

Vegyük észre, hogy amikor egynél több eredmény-argumentumot kérünk, akkor nem jönnek üzenetek. A `flag` nulla értéke azt mutatja, hogy a megoldás teljesíti az előírt megállási feltételt `iter` darab iterációs lépés után.

AUTOMATIKUS DIFFERENCIÁLÁS / MOTIVÁCIÓ

A numerikus algoritmusok gyakran támaszkodnak valamely függvény deriváltjára. A következő eljárásokat szokás használni:

1. Az illető függvény "kézzel", papíron 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: $f'(x) \approx (f(x+h) - f(x))/h$. 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.

A numerikus derivált használatának előnye (+) és hátránya (–):

- + nincs előzetes munkaráfordítás a deriváltak "kézzel" történő előállítására,
- + emiatt javítani sem kell az azok programozása során elkövetett hibákat, és
- + akkor is működik, ha az illető függvény képletét nem ismerjük, csak a kiszámolására szolgáló szubrutin adott.
- 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 „kézzel” való deriválás és a megfelelő rutinok megadása előnye és hátránya:

- + a levágási hiba nem jelentkezik, a kiszámított deriváltértékek általában csak nagyon kis kerekítési hibával terheltek, és
- + a gyorsan változó deriváltértékek is jól meghatározhatók.
- 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.

AUTOMATIKUS DIFFERENCIÁLÁS

A szimbolikus deriválási lehetőségek mellett (amelyek a képletet igénylik), olyan módszer is kellett, amely az előző módszerek előnyeit képes egyesíteni a hátrányok elhagyásával, tehát:

- + lényegében nem igényel előzetes ráfordítást a deriváltak "kézzel-" vagy akár számítógépes algebrarendszerrel, szimbolikus manipulációval való meghatározására,
- + emiatt nem is kell a megfelelő szubrutinokat programozni és javítani,
- + akkor is működik, ha csak az illető függvény kiszámolására szolgáló szubrutin adott, de a függvény képlete nem ismert,
- + a levágási hiba miatt nem vesznek el értékes jegyek,
- + a gyorsan változó deriváltak meghatározására is alkalmas, és
- + a deriváltak kiszámításának műveletigénye általában kisebb, mint a numerikus deriválásé, illetve az analitikus deriváltakat kiszámító szubrutinoké.

A MÓDSZER:

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.

1. A függvény minden változója és konstansa helyett használjunk olyan adatszerkezetet, amely két valós számból áll. Az első felel meg a korábbi változóértéknek, a második pedig egy deriváltértéknek.
2. Minden változóra ez a második valós legyen kezdetben egy. Minden, a függvény kiszámításához használt konstans új adatszerkezetében a második érték legyen nulla.
3. Ezután csak olyan szabályokra van szükség, amelyek minden műveletre megadják a megfelelő operációt az első tagon, és a deriválási szabályoknak megfelelő lépést a második tagon.

AUTOMATIKUS DIFFERENCIÁLÁS / MŰVELETIGÉNY

Például, ha $f(x) = g(x) * h(x)$, akkor a szokásos programsor $F = G * H$ lenne. Az automatikus differenciálás megfelelő művelete ezzel szemben

$$F(1) = G(1) * H(1),$$

és

$$F(2) = G(1) * H(2) + G(2) * H(1).$$

Szabályok néhány alpművelet és elemi függvény differenciálásához:

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

Vegyük észre, hogy a derivált értékének kiszámítása során sehol se jelenik meg a derivált-függvény képlete. Az automatikus differenciálásnak két végrehajtási módja van:

1. a sima, egyszerű változat követi az alapfüggvény kiszámításának sorrendjét, az argumentumoktól halad a függvényérték felé,
2. a fordított módszer ezzel szemben először meghatározza a függvény kiszámítási fáját, majd ennek ismeretében, a redundanciák kihasználásával fordított sorrendben haladva határozza meg a függvény és deriváltja értékét.

A fordított algoritmus előnyének az az ára, hogy a tárigénye magasabb, és a sima algoritmus egy menet végrehajtásával szemben két menetet igényel.

A fontosabb automatikus differenciálási feladatok művelet- és tárigénye:

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

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, $L(\cdot)$: az argumentumok meghatározásának műveletigénye a $\{+, -, *, /, \sqrt{\cdot}, \log, \exp, \sin, \cos\}$ alpműveletek felett, és $S(\cdot)$: az argumentumok meghatározásának tárigénye.

AUTOMATIKUS DIFFERENCIÁLÁS / PÉLDA

Az automatikus differenciálás számítógépes megvalósítását könnyíti, hogy a sima változat ideálisan alkalmas objektum orientált környezetben operátor túltöltéssel való elegáns megoldásra. Számos professzionális megvalósítás született, pl. a PASCAL-XSC, C-XSC, ADOL-C, ADIFOR, JAKEF rendszerek. Megemlítendő, hogy a legkorszerűbb jelenlegi megoldások több százezer soros programokkal megadott függvények differenciálását teszik lehetővé ezen a módon.

PÉLDA.

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 $f'(2) = 2$.

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* 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.

Gyakran hasznos a lehetséges megoldások L halmazát befoglalni egy könny-nyebben kezelhető halmazba, és a felosztást azon végigkövetni.

KORLÁTOZÁS ÉS SZÉTVÁLASZTÁS MÓDSZERE, II.

A korlátozás és szétválasztás (angolul branch-and-bound, B&B) módszere egy *leszámlálási* fát épít fel a következők szerint:

- 0. lépés** Az előkészítés során határozzuk meg a $g(L)$ értéket, és legyen L a leszámllálási fa gyökere. Legyen $k = 1$. Címkezzük 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 ($L' = \{\bar{x}\}$), akkor vége az eljárásnak, $\bar{x}$ 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 címkéként a megfelelő levelekhez. Növeljük a k iterációs számot 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 $f(\bar{x}) \leq f(x)$ adódik az $\bar{x}$ vektorra. Ez pedig pontosan azt jelenti, hogy $\bar{x}$ optimális megoldás.

Érdemes az alaplódszer indítása során egy lehetséges megoldásra vonatkozó (és ezért pontos) felső korlátot adni az optimum értékére. Ennek segítségével a számontartott részfeladatok számát csökkenteni lehet.

KORLÁTOZÁS ÉS SZÉTVÁLASZTÁS MÓDSZERE, PÉLDA

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

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

feltéve hogy a

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

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

Ebben az esetben a lehetséges megoldások halmaza: $L = \{(0, 0, 0, 0), (0, 0, 0, 1), (0, 0, 1, 0), (0, 0, 1, 1), (0, 1, 0, 0), (0, 1, 0, 1), (0, 1, 1, 0)\}$.

Az L -en a célfüggvény alsó korlátjának vegyük azon 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 $\bar{x} = (1, 0, 0, 0)$ 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.

INTERVALLUM ARITMETIKA

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 eredménye tartalmazza az intervallumokban lévő valós értékekre vonatkozó műveletek értékét is.

HALMAZELMÉLETI DEFINÍCIÓ: $A \circ B := \{a \circ b : a \in A, b \in B\}$; $A, B \in \mathbb{I}$, ahol $\mathbb{I}$ a valós kompakt intervallumok halmaza (azaz olyan $[i, j]$ intervallumoké, amelyekre $i, j \in \mathbb{R}$, és $i \leq j$).

ARITMETIKAI DEFINÍCIÓ:

$$\begin{aligned} [a, b] + [c, d] &= [a + c, b + d], \\ [a, b] - [c, d] &= [a - d, b - c], \\ [a, b] * [c, d] &= [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)], \\ [a, b] / [c, d] &= [a, b] * [1/d, 1/c], \text{ ha } 0 \notin [c, d]. \end{aligned}$$

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 intervallumaritmetika ebben az értelemben pontos.

A bizonyítás az állításban szereplő műveletek monotonitásán múlik.

Az alpműveleteken túl a standard függvényeket is ki lehet terjeszteni intervallumokra. Azt az eljárást, amelynek során egy valós függvény műveleteit és standard függvényeit rendre intervallumos megfelelőjükre cseréljük, *természetes intervallum kiterjesztésnek* nevezzük.

Azt mondjuk, hogy egy $f(x)$ valós függvény *befoglaló függvénye* az $F(X)$, ha minden $x \in X$ valósra $f(x) \in F(X)$, ahol $X \in \mathbb{I}^n$. A természetes intervallum kiterjesztés befoglaló függvényt ad. Az $f(x)$ függvény *értékkészletét* az X intervallumon $f(X)$ -el jelöljük.

PÉLDA. Az $x - x^2$ értékkészlete a $[0, 2]$ intervallumon $[-2, 0, 25]$. Ezzel szemben az intervallum-kiterjesztéssel adódó intervallum $[-4, 2]$.

AZ INTERVALLUM-ARITMETIKA ALGEBRAI TULAJDONSÁGAI

- az $+$ és a $-$, illetve a $*$ és az $/$ 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 $a \in \mathbb{R}$ 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 $A \in \mathbb{I}^n$ intervallum szélességét a koordinátánkénti intervallumok szélességének maximumaként: $w(A) := \max(w(A_i) \mid i = 1, \dots, n)$, ha $A = (A_1, A_2, \dots, A_n) \in \mathbb{I}^n$. 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 következők szerint: $m(A) = (a + b)/2$, ha $A \in \mathbb{I}$, és $m(A) = (m(A_1), m(A_2), \dots, m(A_n))$, ha $A \in \mathbb{I}^n$. Ekkor $m(A \pm B) = m(A) \pm m(B)$, ha $A, B \in \mathbb{I}^n$.

A gépi számokkal végzett intervallumos műveletek során gondoskodni kell arról, hogy a kerekítés során a befoglalási tulajdonság ne vesszen el. Ehhez elegendő az ún. kifelé kerekítést alkalmazni, azaz az eredmény intervallumok alsó végpontját lefelé, a felsőt felfelé kell kerekíteni. Ezek a kerekítési módokat egy IEEE szabvány biztosítja, de számos programozási nyelv teljes intervallum kiterjesztést ad, pl. C-XSC, INTLAB, PROFIL/BIAS.

PÉLDA AZ INTERVALLUM-ARITMETIKÁN ALAPULÓ KORLÁTOZÁS ÉS SZÉTVÁLASZTÁS MÓDSZERRE

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 legígéretebb 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:

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

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

$$f([0, 25, 1]) = [0, 0625, 1], \quad f([1, 4]) = [1, 16],$$

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

$$f([0, 25, 1]) = [0, 0625, 1].$$

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

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' részintervallumot 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 $v_i > \tilde{f}$ (ahol $\tilde{f}$ adott eljárás-paraméter, általában a globális minimum legjobb ismert felső korlátja) 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.

INTERVALLUM-FELOSZTÁSI ALGORITMUS II.

Az 5a pontbeli monotonitási teszt:

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

akkor töröl valamely részintervallumot, 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:

töröljük a (V_i, v_i) párt, ha $v_i > \tilde{f}$ (ahol $\tilde{f}$ adott eljárás-paraméter, általában a globális minimum legjobb ismert felső korlátja), $i = 1, 2$

olyan részintervallumokat dob el, amelyekre az $f(x)$ függvény lehetséges legkisebb értéke is nagyobb, mint $\tilde{f}$.

Az $\tilde{f}$ értékét megválaszthatjuk a feladatra vonatkozó előzetes információink alapján, de adaptív módon is: kezdetben legyen $\tilde{f} = \max F(X)$, majd minden vágásnál $\tilde{f} = \min(\tilde{f}, \max F(V_1), \max F(V_2))$. Algoritmusunk 5b lépése az utóbbi eljárással biztos nem dob ki olyan részintervallumot, amelyben globális minimumpont van. Szokás $\tilde{f}$ javítása helyi kereső módszerrel (és utána a kapott közelítő optimum intervallumos kiértékeléssel).

INTERVALLUMOS NEWTON MÓDSZER

Tekintsük most az $f(x) = 0$ egydimenziós feladatot. Feltételezzük, hogy $f'(x)$ folytonos függvény az $[a, b]$ intervallumon, és

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

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

$$X_{k+1} := \left(c(X_k) - \frac{f(c(X_k))}{F'(X_k)} \right) \cap X_k.$$

Itt $c(X)$ 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:

$$X_{k+1} := \left(c(X_k) - \frac{f'(c(X_k))}{F''(X_k)} \right) \cap X_k.$$

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.

Amennyiben a célfüggvény többdimenziós, akkor az intervallumos Newton lépés:

$$X_{k+1} := \left(c(X_k) - H^{-1}(X_k) \nabla f(c(X_k)) \right) \cap X_k,$$

ahol $H(X_k)$ az $f(x)$ Hesse mátrixa befoglalása az X_k argumentum intervallumra.

KORLÁTOZÁS ÉS SZÉTVÁLASZTÁS MÓDSZERE, 2. 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

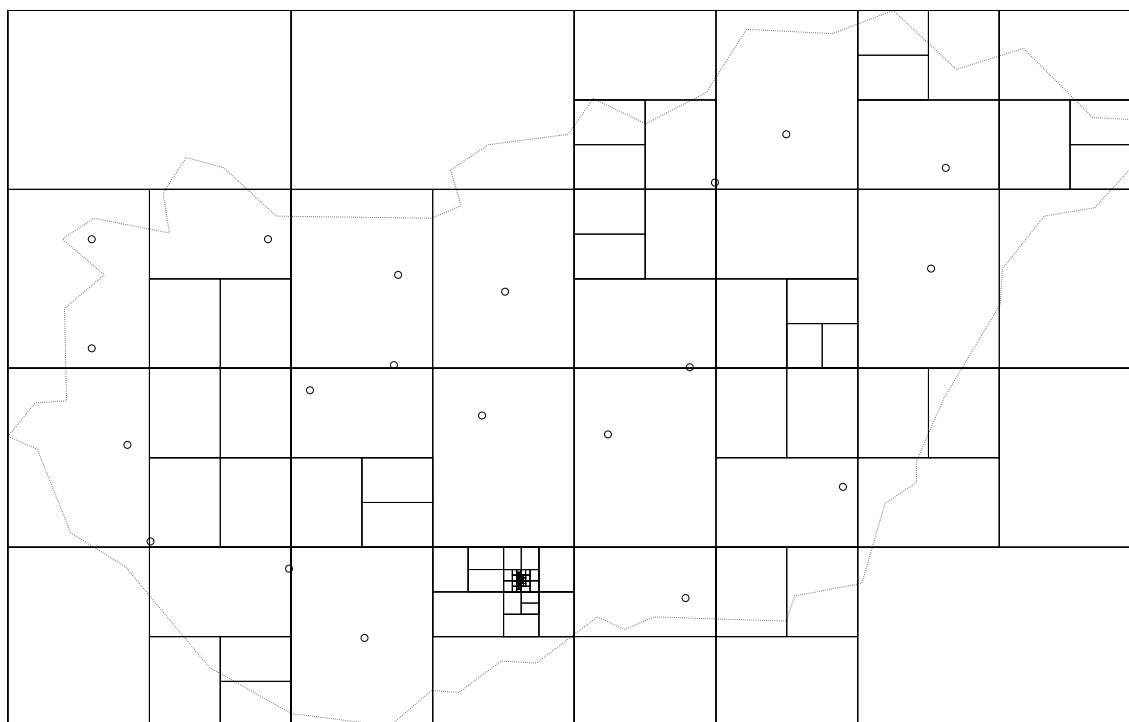
$$f(x) = \sum_i \frac{l_i}{(x - x_i)^2 + (y - y_i)^2},$$

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, 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.

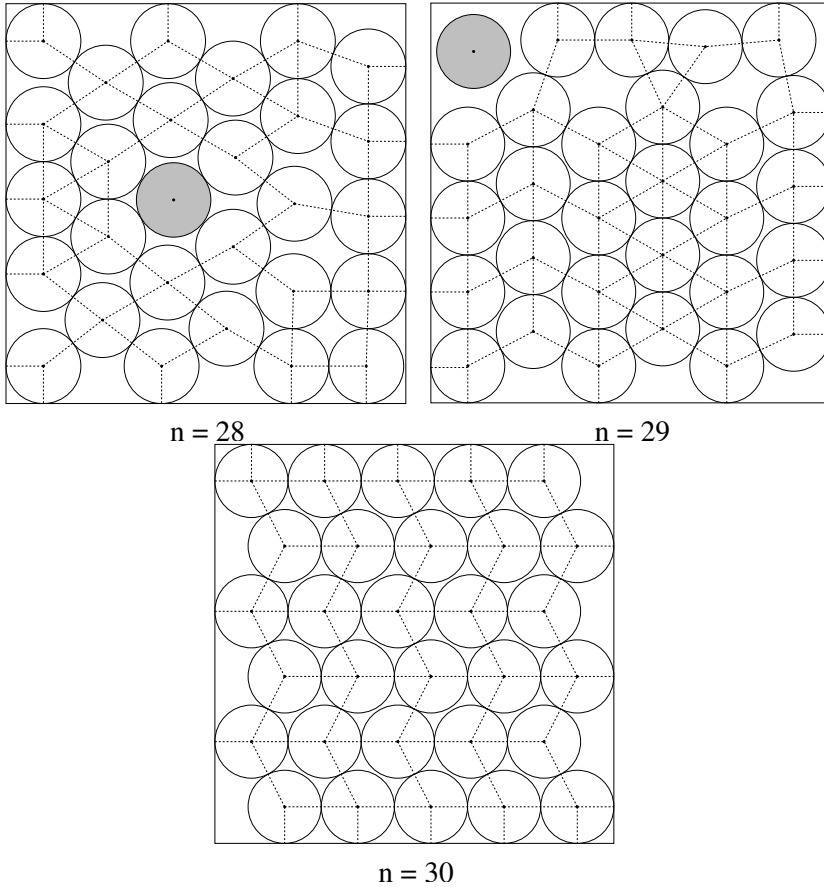
KÖRPAKOLÁSI FELADATOK MEGOLDÁSA B&B MÓDSZERREL

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.

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

$$\text{ahol } 0 \leq x_i, y_i \leq 1, \quad i = 1, 2, \dots, n.$$



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

Hardver: PC, Pentium IV 1.8 GHz processzor, 1 GB RAM. Szoftver: Linux, GNU C/C++, C–XSC Toolbox, PROFIL/BIAS. A középpontok távolságára kapott korlátok:

$$F_{28}^* = [0.2305354936426673, 0.2305354936426743], \quad w \approx 7 \cdot 10^{-15},$$

$$F_{29}^* = [0.2268829007442089, 0.2268829007442240], \quad w \approx 2 \cdot 10^{-14},$$

$$F_{30}^* = [0.2245029645310881, 0.2245029645310903], \quad w \approx 2 \cdot 10^{-15}.$$

A teljes futási idők: ≈ 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.

G. függelék

Kiegészítő 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, amennyivel 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 épülfélben 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. Érdekességgént 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 $(3, 4, \dots)$ 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 befér! 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 $90^\circ = 91^\circ$. Vegyünk egy AB szakaszt, mérjük fel az A pontban 91° -ot, a B pontban pedig 90° -et (úgy, hogy a félegyenese az AB szakasz azonos oldalára essenek). Mérjük fel ugyanazt a távolságot a félegyenésekre: $AD = BC$. Legyen az AB felezőpontja X , a CD felezőpontja Y . Húzzuk meg ezekben a szakaszfelező merőlegesseket. 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 $90^\circ = 91^\circ \dots$
11. Szorzás orosz módra (vagy orosz parasztszorzás, 'multiplication a la Russe'): tegyük fel, hogy csak kettővel tudunk szorozni és osztani. Pl. a 27×13 helyett így 54×6 -ot írunk, és megjegyezzük, hogy a 27-el volt valami bajunk, hiszen a 13 nem osztható kettővel. A következő lépés: 54×6 helyett 108×3 , majd 216×1 , 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 $b_i, i = 1, 2, \dots, n$. 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?

16. Tekintsük azt a helyzetet, amikor egy 100 ülőhellyel rendelkező repülőgépre minden jegyet eladtak, az utasok beszállásra jelentkeznek, de az első beszállónak nincs meg a beszállókártyája, és így nem tudja hova is kell ülnie. Ezek után ő egy egyenletes eloszlással véletlenül választott helyre ül. Ezt követően minden utas a saját helyére ül ha az üres, illetve egy egyenletes eloszlással választott még szabad helyre, ha a sajátja foglalt. A kérdés az, hogy az utolsó utas milyen valószínűséggel tud a saját helyére ülni?

"Tekintsünk egy gömbalakú tehenet."

(Egy amerikai egyetem alkalmazott matematikai intézete által egy rosszul tejelő tehenekkel megvert gazdálkodó megbízására írt tanulmány első mondata.)

H. függelék

A 2002-es SIAM numerikus verseny feladatai

2002-ben a SIAM News folyóirat 100 dollár, 100 decimális jegy címmel versenyt írt ki numerikus feladatok megoldására¹. A maximális pontszámot minden feladatra 10-10 pontos jegy megadásával lehetett elérni. Bár a határidő rég lejárt, érdemes a kitűzött feladatokat megnézni.

1. Mi az értéke a $\lim_{\epsilon \rightarrow 0} \int_{\epsilon}^1 x^{-1} \cos(x^{-1} \log x) dx$ integrálnak?
2. Egy foton mozog 1-es sebességgel az $x-y$ síkon, a $t = 0$ időpontban indul az $(x, y) = (0.5, 0.1)$ pontból keletnek. Minden egész koordinátájú (x, y) pont körül van egy $1/3$ sugarú köralakú tükör. Milyen messze van a foton az origótól a $t = 10$ időpontban?
3. Az A végtelen mátrix, amelynek komponensei $a_{11} = 1$, $a_{12} = 1/2$, $a_{21} = 1/3$, $a_{13} = 1/4$, $a_{22} = 1/5$, $a_{31} = 1/6$ stb., korlátos operátor ℓ^2 -ben. Mennyi $\|A\|$?
4. Mi a globális optimuma a következő függvénynek:

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

5. Legyen $f(z) = 1/\Gamma(z)$, ahol $\Gamma(z)$ a gamma függvény, és legyen $p(z)$ egy köbös polinom, amely a legjobban közelíti $f(z)$ -t az egység sugarú körön a $\|\cdot\|_{\infty}$ supremum normában. Mennyi $\|f - p\|_{\infty}$?
6. Egy bolha a $(0, 0)$ pontból indul a kétdimenziós egész rácson, és egy torzított véletlen sétát hajt végre: minden lépésben északra vagy délre egyaránt $1/4$ valószínűséggel ugrik, keletre $1/4 + \epsilon$ -nal, nyugatra pedig $1/4 - \epsilon$ -nal. Annak a valószínűsége, hogy a bolha visszatér a $(0, 0)$ pontba, $1/2$. Mennyi ϵ ?
7. Legyen A az a 20000×20000 -es mátrix, amelynek elemei nullák a főátlóban lévő $2, 3, 5, 7, \dots, 224737$ prímek, és azon egyesek kivételével, amelyek olyan a_{ij} pozíciókban vannak, amelyekre $|i - j| = 1, 2, 4, 8, \dots, 16384$. Mi lesz az A^{-1} mátrix bal felső eleme?
8. A $[-1, 1] \times [-1, 1]$ négyzetes lemez $u = 0$ hőmérsékletű. A $t = 0$ időpontban a hőmérsékletet egyik oldalán $u = 5$ -re emeljük, míg a többi oldalt változatlanul $u = 0$ hőmérsékleten tartjuk. A meleg a lemez belsejébe áramlik $u_t = \Delta u$ szerint. Mikor éri el a hőmérséklet az $u = 1$ értéket a lemez középpontjában?

¹Nick Trefethen: A Hundred-Dollar, Hundred-digit Challenge. SIAM News 35(2002)

²Ezt a feladatot egy intervallum aritmetikán alapuló korlátozás és szétválasztás algoritmus 0.26 másodperc alatt, 1975 függvény-, 1158 gradiens- és 92 Hesse-mátrixhívás áran megoldotta: az optimum befoglalása a következőnek adódott: $[-3.306868647475316, -3.306868647475196]$

9. Az $I(\alpha) = \int_0^2 [2 + \sin(10\alpha)] x^\alpha \sin(\alpha/(2-x)) dx$ integrál függ az α paramétertől. Milyen $\alpha \in [0, 5]$ értékre lesz $I(\alpha)$ maximális?
10. Egy részecske Brown-mozgást követ a 10×1 téglalap közepéből (azaz 2D véletlen sétát végtelenül kicsi lépésközzel), amíg el nem éri annak határát. Mi a valószínűsége annak, hogy az alsó vagy felső határra ér ki, és nem valamelyik oldalsó szakaszra?

I. függelék

A standard normális eloszlás táblázata

$$P(0 \leq x \leq a)$$

a	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.0000	0.0040	0.0080	0.0120	0.0160	0.0199	0.0239	0.0279	0.0319	0.0359
0.1	0.0398	0.0438	0.0478	0.0517	0.0557	0.0596	0.0636	0.0675	0.0714	0.0753
0.2	0.0793	0.0832	0.0871	0.0910	0.0948	0.0987	0.1026	0.1064	0.1103	0.1141
0.3	0.1179	0.1217	0.1255	0.1293	0.1331	0.1368	0.1406	0.1443	0.1480	0.1517
0.4	0.1554	0.1591	0.1628	0.1664	0.1700	0.1736	0.1772	0.1808	0.1844	0.1879
0.5	0.1915	0.1950	0.1985	0.2019	0.2054	0.2088	0.2123	0.2157	0.2190	0.2224
0.6	0.2257	0.2291	0.2324	0.2357	0.2389	0.2422	0.2454	0.2486	0.2517	0.2549
0.7	0.2580	0.2611	0.2642	0.2673	0.2704	0.2734	0.2764	0.2794	0.2823	0.2852
0.8	0.2881	0.2910	0.2939	0.2967	0.2995	0.3023	0.3051	0.3078	0.3106	0.3133
0.9	0.3159	0.3186	0.3212	0.3238	0.3264	0.3289	0.3315	0.3340	0.3365	0.3389
1.0	0.3413	0.3438	0.3461	0.3485	0.3508	0.3531	0.3554	0.3577	0.3599	0.3621
1.1	0.3643	0.3665	0.3686	0.3708	0.3729	0.3749	0.3770	0.3790	0.3810	0.3830
1.2	0.3849	0.3869	0.3888	0.3907	0.3925	0.3944	0.3962	0.3980	0.3997	0.4015
1.3	0.4032	0.4049	0.4066	0.4082	0.4099	0.4115	0.4131	0.4147	0.4162	0.4177
1.4	0.4192	0.4207	0.4222	0.4236	0.4251	0.4265	0.4279	0.4292	0.4306	0.4319
1.5	0.4332	0.4345	0.4357	0.4370	0.4382	0.4394	0.4406	0.4418	0.4429	0.4441
1.6	0.4452	0.4463	0.4474	0.4484	0.4495	0.4505	0.4515	0.4525	0.4535	0.4545
1.7	0.4554	0.4564	0.4573	0.4582	0.4591	0.4599	0.4608	0.4616	0.4625	0.4633
1.8	0.4641	0.4649	0.4656	0.4664	0.4671	0.4678	0.4686	0.4693	0.4699	0.4706
1.9	0.4713	0.4719	0.4726	0.4732	0.4738	0.4744	0.4750	0.4756	0.4761	0.4767
2.0	0.4772	0.4778	0.4783	0.4788	0.4793	0.4798	0.4803	0.4808	0.4812	0.4817
2.1	0.4821	0.4826	0.4830	0.4834	0.4838	0.4842	0.4846	0.4850	0.4854	0.4857
2.2	0.4861	0.4864	0.4868	0.4871	0.4875	0.4878	0.4881	0.4884	0.4887	0.4890
2.3	0.4893	0.4896	0.4898	0.4901	0.4904	0.4906	0.4909	0.4911	0.4913	0.4916
2.4	0.4918	0.4920	0.4922	0.4925	0.4927	0.4929	0.4931	0.4932	0.4934	0.4936
2.5	0.4938	0.4940	0.4941	0.4943	0.4945	0.4946	0.4948	0.4949	0.4951	0.4952
2.6	0.4953	0.4955	0.4956	0.4957	0.4959	0.4960	0.4961	0.4962	0.4963	0.4964
2.7	0.4965	0.4966	0.4967	0.4968	0.4969	0.4970	0.4971	0.4972	0.4973	0.4974
2.8	0.4974	0.4975	0.4976	0.4977	0.4977	0.4978	0.4979	0.4979	0.4980	0.4981
2.9	0.4981	0.4982	0.4982	0.4983	0.4984	0.4984	0.4985	0.4985	0.4986	0.4986
3.0	0.4987	0.4987	0.4987	0.4988	0.4988	0.4989	0.4989	0.4989	0.4990	0.4990

Értelemszerűen, ha a $P(x \leq a)$ valószínűséget keressük, akkor a táblázatbeli értékekhez 0.5-et hozzá kell adni.

J. függelék

Kötelező program témák

1. a hátizsák feladat megoldása teljes leszámolással
2. a hátizsák feladat megoldása közvetett leszámolással
3. a hajórakodási feladat megoldása teljes leszámolással
4. a hajórakodási feladat megoldása közvetett leszámolással
5. a fix költség feladat megoldása rácsmenti kereséssel *
6. az utazó ügynök feladat megoldása teljes leszámolással
7. az utazó ügynök feladat megoldása szimplex módszerrel *
8. az utazó ügynök feladat megoldása közelítése a hozzárendelési feladat megoldásával (Magyar módszer) *
9. az utazó ügynök feladat megoldása a legközelebbi város beillesztése heurisztikával
10. az utazó ügynök feladat megoldása a legközelebbi város hozzáadása heurisztikával
11. az utazó ügynök feladat megoldása a legközelebbi város beszúrása heurisztikával
12. az utazó ügynök feladat megoldása a legtávolabbi város beszúrása heurisztikával
13. az utazó ügynök feladat megoldása a legolcsóbb beszúrás heurisztikával
14. az oszlopgenerálás módszere a szabási feladatra *
15. rendezés nélküli heurisztikák a szabási feladatra
16. rendezéses heurisztika a szabási feladatra
17. online szabási heurisztika (saját szakirodalom feldolgozással) *
18. a legrövidebb út feladat megoldása a Dijkstra algoritmussal
19. a maximális folyam feladat megoldása a Ford-Fulkerson eljárással
20. a CPM eljárás megvalósítása
21. a projekt lerövidítési feladat megoldása
22. a PERT eljárás megvalósítása

23. a diszkrét keresletű újságárus probléma megoldása
24. a folytonos keresletű újságárus probléma megoldása
25. egyes stochasztikus modellek feladatai megoldása Monte Carlo módszerrel és szimulációval *
26. a Newton módszer megvalósítása nemlineáris feltétel nélküli optimalizálási feladatok megoldására
27. nemlineáris feltétel nélküli optimalizálási feladatok megoldása gradiens módszerrel
28. a konjugált gradiens módszerek összevetése nemlineáris feltétel nélküli optimalizálási feladatokon
29. automatikus differenciálás az INTLAB csomaggal *
30. korlátozás és szétválasztás módszere nemlineáris feltételes optimalizálási feladatok megoldására
31. az intervallum aritmetika használata nemlineáris optimalizálásra az INTLAB csomaggal *

* Az így megjelölt feladatok megítélésem szerint nehezebbek: aki jobb jegyet ambicionál, annak érdemes ilyent választania, bár az egyszerűbbeket is lehet gyönyörűre megírni...

Tárgymutató

- ET*, 142
LT, 142
MH, 144
TH, 144
érintőmódszer, 157
újságáros probléma, 152
út¹, 136
- Activity On Arc, 141
all cities, 125
alulkészletezés, 153
AOA hálózathoz, 141
automatikus differenciálás, 102
- béta eloszlás, 146
befejezés csúcs, 141
befoglaló függvény, 95
best fit algoritmus, 133
- centrális határeloszlás tétel, 147
CPM, 141
Critical Path Method, 141
- determinisztikus feladat, 151
Dijkstra algoritmus, 136
dinamikus modell, 151
diszkrét kereslet, 152
- Early Event Time, 142
egységnyi alulkészletezés, 153
egységnyi túlkészletezés, 152
ekvivalens mátrix, 120
előremenő él, 139
előzmény, 141
esemény, 141
Excel, 148
- független valószínűségi változó, 146
feladat, 151
fiktív tevékenység, 142
first fit módszer, 133
- folytonos keresletű újságáros probléma, 155
forrás, 137
- gradiens módszer, 157
gradiensmódszer, 160
- hálózat egy vágása, 140
hátramenő él, 139
határelemzés, 153
helyesség, 165
helyi kereső eljárás, 157
helyi minimumpont, 157
heurisztika aszimptotikus hányadosa, 124
heurisztikus közelítő eljárás, 122
- időtartam csökkentés, 145
intervallum-aritmetika, 94
irányított gráf, 136
- járatütemezési probléma, 119
- közvetlen előzmény, 142
közvetlen követő, 143
kései időzítés, 142
keresési irány, 159
konjugált gradiens módszer, 159
korai időzítés, 142
korlátozás és szétválasztás módszer, 122
korlátozás és szétválasztás módszere, 164
korlátozás nélküli nemlineáris optimalizálási feladat, 157
korlátozó függvény, 164
kritikus út, 144
kritikus út módszer, 141
kritikus tevékenység, 144
kvadratus konvergencia, 157
- ládapakolási feladat, 124
lánc, 136
lépésköz, 159
Late Event Time, 142
legközelebbi város beillesztése, 122
legközelebbi város beszúrása, 125
legközelebbi város hozzáadása, 123
legolcsóbb beszúrás, 125

¹Az abc-sorrend sajnos nem az igazi: az ékezetes betűket és a matematikai jeleket érdekes helyre teszi a $\LaTeX$...

legtávolabbi város beszúrása, 125
leszámlálási fa, 164

Maple, 148

Matlab, 148

|, 182
~=, 182
~, 182
, 180
*, 177
+, 177
-, 177
., 178
/, 177
0, 182
1, 182
:, 180
<=, 182
<, 182
==, 182
>=, 182
>, 182
Display, 200
MaxGunEvals, 200
MaxIter, 200
NaN, 178
TolFun, 200
TolX, 200
&, 182
^, 178
abs, 178
acos, 178
bicgstab, 161
bicg, 161
break, 183
cgs, 161
computer, 185
cosh, 178
cos, 178
defaultopt, 200
diag, 162
disp, 183
else, 183
end, 183
eps, 185
exit, 177
exp, 178
ezcontour, 192
feval, 179
flag, 162

floor, 178
fminbnd, 193
fminsearch, 193
format, 178
for, 183
fplot, 181
function, 178
fzero, 193
gallery, 161
gmres, 161
if, 183
iter, 162
i, 178
kg, 160
log10, 178
log, 178
lsqr, 161
meshgrid, 182
mesh, 182
minres, 161
nnz, 161
notify, 200
ones, 161, 179
pasc, 184
pcg, 161
plot3, 182
plot, 181, 182
print, 184
qmr, 161
quit, 177
realmax, 185
realmin, 185
rem, 184
sin, 178
spy, 184
sqrt, 178
symmlq, 161
tanh, 178
tan, 178
tol, 161
wathen, 161
while, 183
zeros, 179
maximális folyam probléma, 137
mesterséges él, 138
modell, 151
Monte Carlo szimuláció, 148
mozgáshatár, 144

nemlineáris optimalizálási feladat, 157

Newton módszer, 157
normális eloszlás, 147
numerikus differenciálás, 103
nyelő, 137

offline szabási feladat, 133
optimális vezérlés, 151

PERT, 141, 146
Program Evaluation and Review Technique, 141
program kiértékelési és felülvizsgálati technika, 141

quasi-Newton eljárás, 157

reziduális vektor, 159

SPSS, 148
standard normális eloszlás, 148
statikus modell, 151
szétválasztási függvény, 164
szórás, 146
szabási feladat, 124
szimmetrikus költségmátrix, 122
sztochasztikus optimalizálási modell, 151
sztochasztikus programozási probléma, 151

táblázat a standard normális eloszláshoz, 305
távolságvektor, 123
túlkészletezés, 152
tűréshatár, 144

utód, 143

vágás, 140
vágás kapacitása, 140
végeesség, 165
valószínűség maximalizálási feladat, 152
variancia, 146

worst fit eljárás, 133

Irodalomjegyzék

- [1] D. Knuth. A számítógép programozás művészete 2, Szeminumerikus algoritmusok. Műszaki Könyvkiadó, Budapest, 1994.
- [2] Gács Péter, Lovász László. Algoritmusok. Tankönyvkiadó, Budapest, 1991.
- [3] A.G. Kuros. Felsőbb algebra. Tankönyvkiadó, Budapest, 1978.
- [4] G. Laakmann McDowell. Cracking the coding interview (5th edition). CareerCup LLC, Palo Alto, CA, USA, 2014.
- [5] A. Aziz, T-H. Lee, A. Prakash. Element of programming interviews (version 1.4.6). <http://elementsofprogramminginterviews.com/>
- [6] Richard P. Brent, Paul Zimmermann. Modern Computer Arithmetic. <https://arxiv.org/abs/1004.4710>
- [7] Jonathan M. Borwein, David H. Bailey, Roland Girgensohn. Experimentation in Mathematics: Computational Paths to Discovery. CRC Press, 2004.
- [8] Márton Gyöngyvér. Kriptográfiai alapismeretek. Scientia Kiadó, Kolozsvár, 2008.
- [9] Lax Péter. Lineáris algebra és alkalmazásai. Akadémiai Kiadó, Budapest, 2008.
- [10] Johan Hastad. Advanced algorithms. KTH, 2000. <https://www.nada.kth.se/~johanh/algnotes.pdf>
- [11] Richard Liska, Ladislav Drska, Jiri Limpouch, Milan Sinor, Michael Wester, Franz Winkler. Computer Algebra, Algorithms, Systems and Applications. 1999. <http://www-troja.fjfi.cvut.cz/~liska/ca/>
- [12] Joachim von zur Gathen and Jürgen Gerhard. Modern Computer Algebra. Cambridge University Press, 1999.
- [13] Bajalinov Erik és Imreh Balázs: Operációkutatás. Polygon Jegyzettár, Szeged, 2001.
- [14] Bazaraa, M.S., H.D. Sherali, and C.M. Shetty: Nonlinear Programming – Theory and Algorithms, Wiley, 1993.
- [15] Bomze, I.M., T. Csendes, R. Horst, and P.M. Pardalos (eds.): Developments in Global Optimization. Kluwer, Dordrecht, 1997.
- [16] Csallner András Erik: Intervallum-felosztási eljárások a globális optimalizálásban. PhD disszertáció, Szeged, 1999. Elérhető a <http://www.jgytf.u-szeged.hu/~csallner> internetes címen.
- [17] Csernyák László (szerk.): Operációkutatás II., Nemzeti Tankönyvkiadó, Budapest, 1999.
- [18] Dixon, L.C.W., G.P. Szegő (eds.): Towards Global Optimisation. North-Holland, Amsterdam, 1974.
- [19] Dixon, L.C.W., G.P. Szegő (eds.): Towards Global Optimisation 2. North-Holland, Amsterdam, 1978.

- [20] Forgó Ferenc: Nemkonvex és diszkrét programozás: korszerű ismeretek gazdasági szakemberek számára, Közgazdasági és Jogi Könyvkiadó, Budapest, 1978.
- [21] Gáspár László és Temesi József: Lineáris programozási gyakorlatok. Nemzeti Tankönyvkiadó, Budapest, 1999.
- [22] Gáspár László és Temesi József: Matematikai programozási gyakorlatok. Nemzeti Tankönyvkiadó, Budapest, 1999.
- [23] Gerencsér László: Nemlineáris programozási feladatok megoldása szekvenciális módszerekkel, SZTAKI Tanulmányok 49/1976, Budapest.
- [24] Gill, P. E., W. Murray, M.H. Wright: Practical Optimization. Academic Press, London, 1981.
- [25] GNU OCTAVE vendégoldal: www.che.wisc.edu/octave
- [26] Hansen, E.: Global Optimization Using Interval Analysis. Marcel Dekker, New York, 1992.
- [27] Higham, Desmond J. and Nicholas J. Higham: MATLAB Guide. SIAM, Philadelphia, 2000.
- [28] Hillier, F.S., G.J. Lieberman: Bevezetés az operációkutatásba. LSI Oktatóközpont, Budapest, 1994.
- [29] Horst, R., P.M. Pardalos, and N.V. Thoai: Introduction to Global Optimization, Kluwer, Dordrecht, 1995.
- [30] Imreh Balázs: Kombinatorikus optimalizálás. Novadat Kiadó, Győr.
- [31] Komlósi Sándor: Az optimalizáláselmélet alapjai. Dialóg Campus Kiadó, Budapest, Pécs, 2001.
- [32] Kósa András: Optimumszámítási modellek, Műszaki Könyvkiadó, Budapest, 1979.
- [33] Krekó Béla: Optimumszámítás. Nemlineáris programozás, Közgazdasági és Jogi Könyvkiadó, Budapest, 1972.
- [34] Martos Béla: Nemlineáris Optimalizálás, Akadémiai Könyvkiadó, Budapest, 1974.
- [35] MATLAB online kézikönyvek:
www.mathworks.com/access/helpdesk/help/fulldocset.shtml
- [36] Neumaier, Arnold globális optimalizálási portálja számos további linkkel, forráskódokkal, tesztfeladatokkal:
<http://www.solon.math.univie.ac.at/globopt>
- [37] NETLIB vendégoldal: www.netlib.org
- [38] Pintér, J.D.: Global Optimization in Action, Kluwer, Dordrecht, 1996.
- [39] András Prékopa: Stochastic Programming. Kluwer, Dordrecht, 1995.
- [40] Raffai Mária (szerk.): Döntéselőkészítés – Operációkutatási Módszerek. Novadat Kiadó, Győr, 2000.
- [41] Ratschek, H., J. Rokne: Computer Methods for the Range of Functions. Ellis Horwood, Chichester, 1984.
- [42] Ratschek, H., J. Rokne: New Computer Methods for Global Optimization. Ellis Horwood, Chichester, 1988.
- [43] SCILAB vendégoldal: www-rocq.inria.fr/scilab/scilab.html
- [44] Tóth Irén (szerk.): Operációkutatás I., Nemzeti Tankönyvkiadó, Budapest, 1999.
- [45] Törn, A., A. Žilinskas: Global Optimization. (Lecture Notes in Computer Science No. 350, G. Goos and J. Hartmanis, Eds.) Springer, Berlin, 1989.
- [46] Winston, W.L.: Operációkutatás I-II. Módszerek és alkalmazások. Aula Kiadó, Budapest, 2003.

Az irodalomjegyzék persze nem teljes, csak néhány fontosabb, illetve a tananyaghoz közvetlenül kapcsolódó könyvet, jegyzetet adunk meg. Az itt megadott 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.