I Függelék

Az előadás fóliái

KÖZELÍTŐ ÉS SZIMBOLIKUS SZÁMÍTÁSOK I.

Csendes Tibor
www.inf.u-szeged.hu/∼csendes, csendes@inf.u-szeged.hu

Ez az új tantárgy az elődje, a Numerikus Matematika nyomán valós számokon végzett műveleteket tartalmazó algoritmusok számítógépes megvalósításába és annak használatába ad bevezetést, kitérve a MATLAB és Maple programok használatára. A kredit követelményei:

- Az előadásra járás kötelező, a hiányzásokat (4 fölött) dolgozatírással lehet jóvátenni
- A gyakorlat keretében az első félévben egy MATLAB programot kell önállóan megírni és az erre vonatkozó esszét két részletben leadni határidőre (október 15. és november 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 vizsgáé és az esszéé.
- A korábbiaktól eltérően mindenkinek kell vizsgáznia. Az elérhető 100 pontból 50 kell az elégségeshez, és 80 a jeleshez. A részletes kurzusteljesítési feltételrendszer a Számítógépes Optimalizálás Tanszék vendégoldalán érhető el.

Motiváció: a jelenlévők többsége a végzés után mégis csak számolni fog számítógépen...

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. Érdemes itt kis méretű, áttekinthető példákat használni.

2. A megoldására megírt, felhasznált 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, 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 LaTeX 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-et, Netlib-et vagy Octave-ot használja
- a hallgató javasolhat is esszé formában feldolgozandó témát, de mindenben a gyakorlatvezetővel kell egyeztetnie

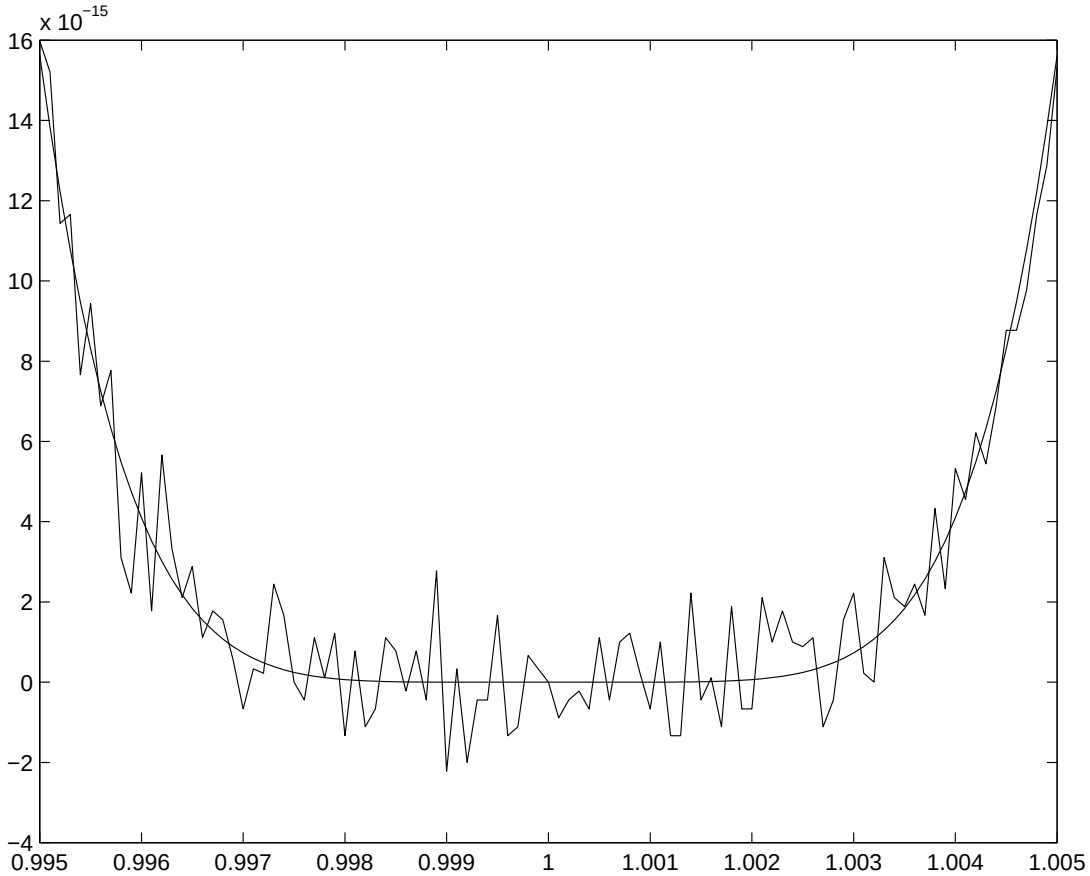
MATLAB MINTA

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 $(1-x)^6$):

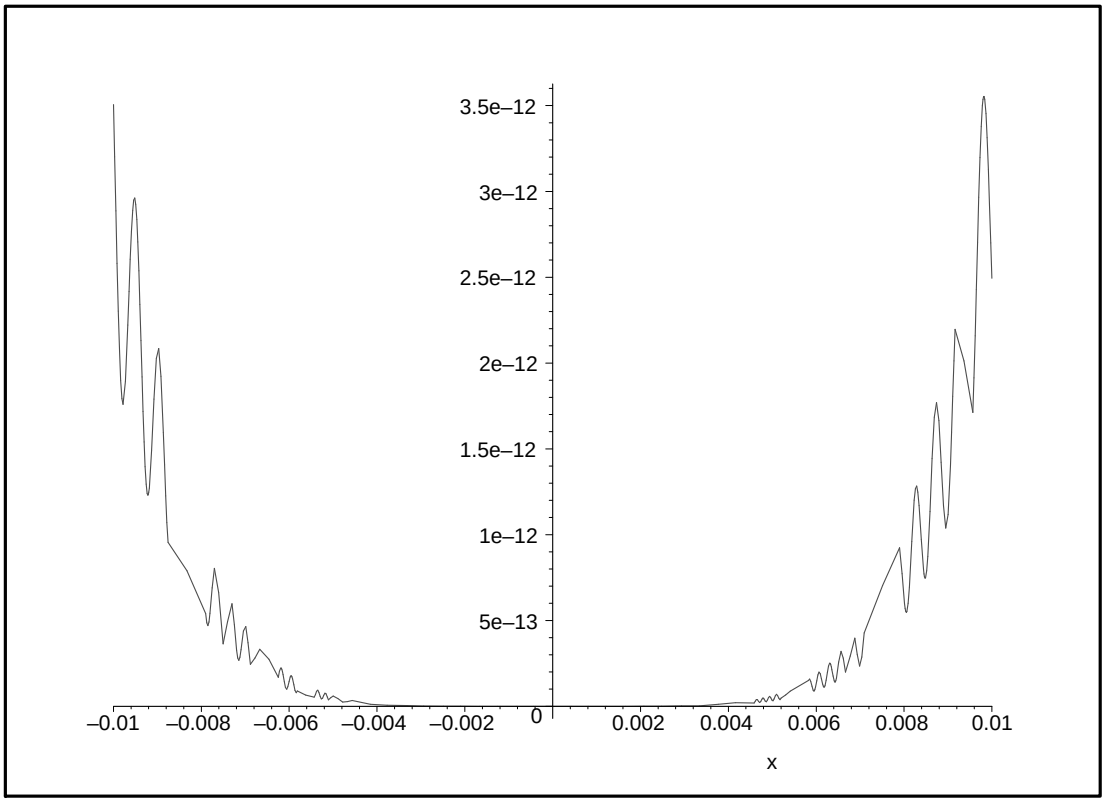


MAPLE MINTA

```
> f(x):=x^6*(sin(1/x)+3);
```

$$f(x) := x \hat{} 6 * (\sin(1/x) + 3)$$

```
> plot(f(x),x=-0.01..0.01);
```




```
> g(x):=diff(f(x),x);
```

$$g(x) := 6 * x \hat{} 5 * (\sin(1/x) + 3) - x \hat{} 4 * \cos(1/x)$$

```
> solve(g(x),x);
```

$$0, 0, 0, 0$$

AJÁNLOTT IRODALOM

Az előadás anyagát a jegyzet tartalmazza. Ezt kiegészíthetik az ezután fontossági sorrendben megadott könyvek:

1. Csendes Tibor: Közelítő és szimbolikus számítások. Polygon, Szeged, 2007
   `http://www.inf.u-szeged.hu/~csendes/koszikicsi.pdf`
2. Virágh János: Numerikus matematika. JATEPress, Szeged, 1977.
3. Gramlich, Günter, und Wilhelm Werner: Numerische Mathematik mit Matlab (Eine Einführung für Naturwissenschaftler und Ingenieure), dpunkt.verlag, Heidelberg, 2000.
4. Mathews, John H. and Kurtis D. Fink: Numerical Methods Using MATLAB, Prentice Hall, Upper Saddle River, 1999.
5. Neumaier, Arnold: Introduction to Numerical Analysis. Cambridge University Press, Cambridge, 2001.
6. Stoyan Gisbert (szerk.): MATLAB (4. és 5. verzió). TypoTEX Kiadó, Budapest, 1999.
7. Móricz Ferenc: Numerikus analízis, I. kötet, Tankönyvkiadó, Budapest, 1977.
8. Móricz Ferenc: Numerikus analízis, II. kötet, Tankönyvkiadó, Budapest, 1976.
9. Higham, Desmond J. and Nicholas J. Highham: MATLAB Guide. SIAM, Philadelphia, 2000.
10. Stoer, J. and R. Bulirsch: Introduction to numerical analysis, Springer, New York, 1992.
11. Westermann, Thomas: Mathematische Probleme lösen mit Maple, Springer, Berlin, 2003.
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`

HIBA

Legyen x a keresett, kiszámítandó pontos érték, és $\tilde{x}$ ennek a becslése.

- Az $\tilde{x}$ *becslés hibája* az $x - \tilde{x}$ érték,
- *abszolút hibája* pedig az $|x - \tilde{x}|$ nemnegatív szám.
- Ez utóbbi érték egy korlátját *abszolút hibakorlát*nak nevezzük. Nyilván a lehetséges értékekből a legkisebb a legjobban használható.
- Azt mondjuk, hogy $\tilde{x}$ az x érték egy század *pontosságú becslés*e, ha 0.01 abszolút hibakorlát.
- A *relatív hiba* az eltérésnek a pontos érték nagyságrendjéhez való viszonyát adja meg: $|x - \tilde{x}|/|x|$,
- A *relatív hibakorlát* pedig ennek egy felső korlátja, amit gyakran százalékban vagy ezrelékben adnak meg.
- A pontos tizedesjegyek számát a relatív hiba negatív tizesalapú logaritmusa adja: $-\log_{10}(|x - \tilde{x}|/|x|)$. Ez persze nem mindig egész szám, de annál pontosabb információt ad.

PÉLDA

Tekintsünk egy egyszerű példát, legyen a $\pi = 3.14159265...$ becslése 3.141626.

Ekkor a becslés hibája -0.000033...

A becslés abszolút hibája 0.000033...

A becslés abszolút hibakorlátja például 0.000034 lehet.

A relatív hiba 0.000010...

A relatív hibakorlát pedig pl. 0.000011.

A pontos tizedesjegyek száma 4.98..., ami összhangban van azzal, hogy az első négy *értékes jegy* megegyezik, de az ötödik már nem.

HIBA (folytatás, 2.)

KÉRDÉS: Jól van-e az, hogy a 2.0 és az $1.\dot{9}$ pontos tizedesjegyeinek száma az előző képlet alapján nagy, bár nincs megegyező értékes jegyük?

A hiba több forrásból származhat, például:

- Ha már a beolvasott adatok is hibásak, akkor erre vonatkozik az *öröklött hiba* elnevezés.
- A számítási módszer tévedéséből származó eltérés neve *képlethiba*.
- A meghatározott új számok számítógépes ábrázolása során csak véges sok lebegőpontos szám közül választhatunk. Az ekkor elkövetett hiba neve *kerekítési hiba*.

A *hibaszámítás alapfeladata*: az öröklött hiba és a számítógépes eljárás ismeretében adjunk az eredményre hibakorlátot. Az összes lehetséges hibaforrást figyelembe kell venni, emiatt ez a feladat nehéz, inkább csak numerikus programkönyvtárak rutinjainak vizsgálata során szokásos.

A differenciálható függvényeknek az argumentumok pontatlanságából adódó hibája a Lagrange tétellel becsülhető: $f(x) - f(\tilde{x}) = f'(\xi)(x - \tilde{x})$ miatt (ξ az x és $\tilde{x}$ között)

$$|f(x) - f(\tilde{x})| \leq |f'(\xi)|\,|x - \tilde{x}| \leq M_1 K_x$$

adódik, ahol M_1 a derivált abszolút értékének, K_x pedig az x becslése hibájának egy felső korlátja.

PÉLDA

Tekintsünk egy olyan feladatot, amikor egy ismeretlen képletű, differenciálható $f(x)$ függvény deriváltját kell meghatározni, és $f(x)$ értékeit méréssel tudjuk megállapítani. Tegyük fel, hogy a derivált becslésére az ún. numerikus differenciálást használjuk, tehát feltesszük, hogy $f'(x) \approx (f(x+h) - f(x))/h$ egy alkalmas kis h pozitív valós számra. Az $f(x) = x^2$ függvényre

$$(1.012 - 0.9900)/0.0010 = 22.00$$

adódik 1% mérési hibával, 4 értékes jegyet tárolva $h = 0.001$ lépésközzel – $f'(1) = 2$ helyett.

SZÁMÁBRÁZOLÁS

A programozási nyelvek és a hardver a nagyobb hatékonyság kedvéért véges, rögzített hosszúságú számábrázolást alkalmaznak. A leggyakoribb két formátum

decimális egész: $\pm xxxxx$, és
decimális lebegőpontos szám: $\pm .xxxxx_{10} \pm xxx$.

Az egész számokkal általában kevesebb numerikus probléma van, kivéve amikor pl. a számlálók értéke meghaladná a lehetséges legnagyobbat, és ezután negatív értéket kapnak (Tetris). Fontos szerepük van több olyan alkalmazásban, amikor csak egész számok jöhetnek szóba, pl. banki műveletek számításakor. Az egész számok 2 vagy 4 byte hosszúak, és ennek megfelelően legnagyobb értékük (az előjelet nem tárolják): 2^{15}, illetve 2^{31}.

A lebegőpontos számban az exponens előtti részt mantisszának hívják. A lebegőpontos számok egyik gyakori alakja az, amelyikben a mantissza első jegye (a tizedespont után) nem nulla. Ezt az alakot normalizáltnak nevezzük. Kis abszolút értékű számok esetén előfordul, hogy nincs normalizált alakjuk, ezeket denormalizáltnak hívjuk (pl. maga a nulla). A lebegőpontos számok belső ábrázolásában 2 hatványait szokás alapnak használni, ez a hardver és a pontosság függvénye. Az általánosított normalizált lebegőpontos szám tehát egy L jegyű mantisszából, és egy $[-E, F]$ intervallumba eső értékű exponensből áll.

A PC-k, és egyes munkaállomások eleget tesznek az IEEE 754-1985, bináris alapú lebegőpontos számokra vonatkozó szabványának. A szimpla pontosságú (real vagy real*4) szám 24 bites mantisszát és 8 bites exponenst használ, és ezzel, az előjelek és a NaN számára fenntartott exponensek figyelembevételével 10^{-38} és 10^{+38} közötti abszolút értékű normalizált számokat lehet ábrázolni. Ez kb. 7 tizedesjegy pontosságnak felel meg. A leggyakrabban használt dupla pontosságú lebegőpontos számokra (double vagy real*8) $L = 52$, $E = 1022 = F - 1$, az ennek megfelelő normalizált abszolút értékek 10^{-308} és 10^{308} közöttiek, és kb. 16 tizedesjegy pontosságúak. 11 bites exponenst hasznal

Számos programozási nyelvben, így a Matlabban és a Fortranban is vannak komplex számok is, a műveletek és a függvények kiterjesztésével együtt. A Matlab bizonyos értelemben nem tesz különbséget egész, valós és komplex számok között. A belső ábrázolása dupla pontosságú, így a szimpla pontosságot nem is lehet vele illusztrálni.

Érdemes még megemlíteni, hogy ezek a számábrázolási formák az elmentett, tárolt számokra vonatkoznak, a processzorokon belüli formátumok ezeknél rendszerint pontosabbak.

KEREKÍTÉS

A *kerekítés* szolgál arra, hogy az adott számítógépes környezetben elérhető ábrázolható számok, az ún. gépi számok közül kiválasszuk azt, amellyel eredményünket azonosítjuk.

Az *optimális kerekítés* az adott számhoz legközelebbi, a számábrázolásban szereplő számot eredményezi.

A *levágás* egyszerűen elhagyja a jegyek egy részét: ha L jegyre kerekítünk, akkor az L-edik utáni jegyeket. Ezt könnyebb megvalósítani, de kevésbé pontos.

Azt mondjuk, hogy egy kerekítés *korrekt*, ha a szám és a kerekített között nincs gépi szám. Mind az optimális kerekítés, mind a levágás korrekt kerekítés.

PÉLDA: A $\pi = 3.14159265...$ négy jegyre való optimális kerekítése 3.142, levágással kapott kerekítése pedig 3.141.

ÁLLÍTÁS. Tegyük fel, hogy az exponens hossza nem korlátos, és a kerekítendő érték nem nulla. Ekkor a B bázisú, L jegyre való korrekt kerekítés relatív pontossága $\epsilon = B^{1-L}$, az optimális kerekítésé pedig $\epsilon = B^{1-L}/2$.

BIZONYÍTÁS. A korrekt kerekítés esetét bizonyítjuk akkor, amikor a kerekítendő x érték pozitív. A többi eset igazolása hasonlóan történik.

Tegyük fel, hogy $B^{e-1} \leq x \leq B^e$, és $x = .x_1x_2 \dots x_Lx_{L+1} \dots B^e$ a normalizált alak (tehát $x_1 \neq 0$). Ekkor a kerekített és a pontos érték eltérése nem lehet nagyobb mint B^{e-L} a kerekítés korrektsége miatt. Ez az eltérés az L. pozíción egy egységnek felel meg. Innen

$$|x - \tilde{x}| \leq B^{e-L} = B^{e-1}B^{1-L} \leq |x|B^{1-L},$$

amivel az állítást igazoltuk. □

A gyakorlatban az exponens hossza persze véges, és számos probléma adódik abból, hogy az ábrázolandó szám abszolút értéke túl nagy, vagy túl kicsi ahhoz, hogy normalizált számmal lehessen reprezentálni. Ekkor beszélünk *felül-*, vagy *alulcsordulás*ról. Az ilyen esetek kezelése a fordítóprogramon múlik. Ha az alulcsordulás miatt denormalizált eredményt kapunk, akkor ez ronthatja a relatív pontosságot. Ezen az előforduló számok nagyságrendjének megváltoztatása, az ún. *skálázás* javíthat.

FELADAT: Határozzuk meg a *gépi pontosság*ot, ϵ-t (azt a legkisebb pozitív számot, amit egyhez adva egynél nagyobb számot kapunk) különböző processzorokra és programozási nyelvekre!

NUMERIKUS STABILITÁS

A korábbi Matlab minta fólián látott két függvényalak *numerikus stabilitás*a eltérő volt. Ez egy kvalitatív fogalom, a numerikus stabilitás azt jelenti, hogy az illető függvény argumentumának kis megváltozása esetén a várható, kis eltérés mutatkozik a függvényértékekben. Az ellentéte a numerikus instabilitás. A célunk nyilván olyan numerikus eljárások létrehozása és használata, amelyek a kiszámítandó függvények numerikusan stabilis változatait használják.

El kell különíteni a *számábrázolás pontosságát* (angolul precision) és egy *közelítő érték pontosságát* (accuracy). Mondhatjuk, hogy x közelítőleg egyenlő y-al: $x \approx y$, és azt hogy x sokkal nagyobb, mint y, azaz $x \gg y$.

Amit kerülni kell numerikus algoritmusokban:

- közel azonos számok kivonását, ez az eredmény relatív hibájának növekedését eredményezi, (ilyenkor maga a kivonás viszonylag pontos ugyan, de a korábbi hibák nagyítódnak fel),
- kis abszolút értékű számmal való osztást vagy nagy abszolút értékű számmal való szorzást, mert ez az abszolút hiba növekedését okozza,
- általában kerülni kell olyan helyzeteket, amikor egy részszámítás eredménye lényegesen nagyobb, vagy kisebb nagyságrendű, mint a végeredmény.

Példák instabil kifejezések stabilizálására:

$$\sqrt{x+1} - \sqrt{x} = \frac{1}{\sqrt{x+1} + \sqrt{x}}, \quad x \gg 1 \ \text{ esetén},$$

$$1 - \cos x = \frac{\sin^2 x}{1 + \cos x} = 2 \sin^2 \frac{x}{2} \ \text{ ha } \ x \approx 0,$$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \frac{-b \mp \sqrt{b^2 - 4ac}}{-b \mp \sqrt{b^2 - 4ac}} = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}},$$

ha $b^2 \gg 4ac$, illetve még jobb

$$q := -(b + \text{sign } b\sqrt{b^2 - 4ac})/2, \quad x_1 := q/a, \ x_2 := c/q.$$

A HIBA TOVÁBBTERJEDÉSE ÉS A KONDÍCIÓSZÁM

A numerikus számítás szempontjából döntő kérdés, hogy a korábbi, esetleg öröklött hibák milyen relatív hibához vezetnek a számítás során. Ahogy majd látni fogjuk, bizonyos feladatok esetén a relatív hibák növekedése elkerülhetetlen. Ezeket a feladatokat *rosszul kondícionált*nak (ill-conditioned) nevezzük.

PÉLDA. Vizsgáljuk az

$$f(x) = \frac{1}{1-x}$$

függvény kiértékelését az $x = 0.999$ pont közelében. $f(x) = 1000$. Ehhez közeli pontokban, $\tilde{x} = 0.999 + \varepsilon$ esetén kis ε-ra

$$f(\tilde{x}) = \frac{1}{1-(0.999+\varepsilon)} = \frac{1000}{1-1000\varepsilon} = 1000(1 + 10^3\varepsilon + 10^6\varepsilon^2 + \ldots).$$

Határozzuk meg most a relatív hibát x-re és $f(x)$-re vonatkozóan:

$$\frac{|x-\tilde{x}|}{|x|} \approx 1.001\varepsilon,$$

illetve

$$\frac{|f(x)-f(\tilde{x})|}{|f(x)|} = 10^3\varepsilon + 10^6\varepsilon^2 + \ldots$$

Más szóval az $f(x)$ kiszámításának módjától függetlenül a relatív hiba kb. ezerszerese lesz az argumentum hibájának. A rosszul kondícionáltság mérésére szolgál differenciálható kifejezések esetén a *kondíciószám*, κ: ez a relatív hiba (aszimptotikus) növekedésének mértéke.

A függvények aszimptotikus, határértékben értelmezett viselkedésének jellemzésére az ún. „kis ordó", o, és a „nagy ordó", $\mathcal{O}$ használatos.

Azt mondjuk, hogy f nagyságrendje g, azaz $f(x) = o(g(x))$, ha

$$\lim_{x \to x^*} \frac{f(x)}{g(x)} = 0.$$

Hasonlóan $f(x) = \mathcal{O}(g(x))$, ha $f(x)/g(x)$ korlátos x^* egy környezetében.

A HIBA TOVÁBBTERJEDÉSE ÉS A KONDÍCIÓSZÁM /2

Tekintsük azt az esetet, amikor az argumentumot egy ε mértékű relatív hiba terheli: $\tilde{x} = (1+\varepsilon)x$. Vegyük $f(\tilde{x})$-nak az x körüli Taylor sorfejtését:

$$f(\tilde{x}) = f(x) + \varepsilon x f'(x) + \mathcal{O}(\varepsilon^2) = f(x)\left(1 + \frac{xf'(x)}{f(x)}\varepsilon + \mathcal{O}(\varepsilon^2)\right).$$

Ekkor f relatív hibája

$$\frac{|f(x) - f(\tilde{x})|}{|f(x)|} = \left|\frac{xf'(x)}{f(x)}\right| |\varepsilon| + \mathcal{O}(\varepsilon^2) = \kappa|\varepsilon| + \mathcal{O}(\varepsilon^2),$$

és ez alapján a kondíciószám:

$$\kappa = \left|\frac{xf'(x)}{f(x)}\right|.$$

Ennek megfelelően tehát az $f(x)$ relatív hibája a fenti becslés alapján az argumentum relatív hibájának közel κ-szorosa az x pont körül. A magasabb rendű tagok elhagyása után azt kapjuk, hogy

$$\frac{|f(x) - f(\tilde{x})|}{|f(x)|} \approx \kappa \frac{|x - \tilde{x}|}{|x|}.$$

$\kappa < 1$ esetén *hibaelnyomás*ról, $\kappa > 1$ esetén *a hiba növelésé*ről lehet beszélni. A rosszul kondícionált feladatokra $\kappa \gg 1$.

PÉLDA. Tekintsük az $f(x) = \sqrt{x^{-1} - 1} - \sqrt{x^{-1} + 1}$ függvényt. Az $x = 0$ pont környezetében $x^{-1} - 1 \approx x^{-1} + 1$, és ezért az elvesztett jegyek miatt a kifejezés numerikusan nem stabilis. Stabilis viszont az $x = 1$ pont körül.

A függvény kondíciószáma

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

alapján

$$\kappa = \frac{xf'(x)}{f(x)} = \frac{1}{2\sqrt{1 - x^2}}.$$

Eszerint viszont f jól kondícionált a 0 körül ($\kappa \approx 0.5$), de rosszul kondícionált az 1 közelében (mert $\kappa \to \infty$). Más szóval a numerikus stabilitás és a jól kondícionáltság nem mindig jár együtt.

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: $A \circ B := \{a \circ b : a \in A, \quad b \in B\}$; $A, B \in \mathbb{I}$, 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:

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

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.

A bizonyítás az állításban szereplő műveletek monotonitásán múlik.

Az alapműveleteken 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 kiterjesztés*nek nevezzük.

Azt mondjuk, hogy egy $f(x)$ valós függvény *befoglaló függvénye* az $F(X)$, ha minden $x \in X$ valósra $f(x) \in F(X)$. 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

- a $+$ 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.

0 Szélesség I => egy valós szám

- 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: $w(A):=\max(w(A_i)\ i=1,\ldots,n)$, ha $A=(A_1,A_2,\ldots,A_n)\in\mathbb{I}^n$. Ekkor teljesülnek a következők:
  1. ha $A\subseteq B$, akkor $w(A)\le w(B)$
  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 $m(A)=(m(A_1),m(A_2),\ldots,m(A_n))$, 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. Ezeket a kerekítési módokat a korábban említett IEEE szabvány biztosítja, de számos programozási nyelv teljes intervallum kiterjesztést ad, pl. C-XSC, INTLAB, PROFIL/BIAS.

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

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

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

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

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

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! h = lépésköz
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 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 helyett használjunk olyan adat-szerkezetet, amely két valós számból áll. Az első felel meg a korábbi változó-értéknek, a második pedig egy derivált-értéknek.
2. Minden változóra ez a második valós legyen kezdetben egy. Minden, a függvény kiszámításához használt konstans új adat-szerkezetében 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 $f(x) = x_1 * x_2$, akkor a szokásos programsor $F = X(1) * X(2)$ lenne. Az automatikus differenciálás megfelelő művelete ezzel szemben

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

és

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

Szabályok néhány alapművelet 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 deriváltfüggvény képlete. Az automatikus differenciálásnak két végrehajtási módja van:

1. a sima, egyszerű változat követi az alapfüggvény kiszámítási 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 egymenetes 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, $L(.)$: az argumentumok meghatározásának műveletigénye a $\{+, -, *, /, \sqrt{}, \log, \exp, \sin, \cos\}$ alapműveletek felett, és $S(.)$: 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.

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 $f'(2) = 2$.

LINEÁRIS EGYENLETRENDSZEREK

Az $Ax = b$ alakú lineáris egyenletrendszerek a numerikus eljárások talán legfontosabb területét adják. Bár a gyakorlati feladatokban csaknem semmi sem lineáris, a konkrét számítógépes algoritmusok szinte csak lineáris problémákat oldanak meg. A tárgyalt feladatok részletes alakja tehát:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n. \end{aligned}$$

Az a_{ij} és a b_i együtthatók általános esetben komplex számok is lehetnek: $a_{ij}, b_i \in \mathbb{C}$. A következő tétel a lineáris algebrából ismeretes.

TÉTEL. Tegyük fel, hogy $A \in \mathbb{C}^{n\times n}$, és $b \in \mathbb{C}^n$. Az $Ax = b$ lineáris egyenletrendszernek pontosan akkor van egyetlen megoldása, ha A nem szinguláris (azaz $\det A \neq 0$, vagyis az A rangja n). Ekkor a megoldás $x = A^{-1}b$. A megoldás $i.$ komponensét megadja a Cramer szabály is:

A-1 = A matrix inverze

b és x is oszlop vektor

$$x_i = \frac{\det A^{(i)}}{\det A},$$

ahol az $A^{(i)}$ mátrixot úgy kapjuk az A mátrixból, hogy annak $i.$ oszlopát kicseréljük a b vektorral.

Gyakorlati szempontból ez a tétel alig alkalmazható, mert az inverz meghatározása indokolatlanul nagy műveletigényű, és a Cramer szabály numerikusan nem stabilis. A nem szinguláris mátrixokat *reguláris mátrix*nak is szokás nevezni.

A lineáris egyenletrendszerek megoldására két módszerosztály használatos. Az elsőbe tartoznak a *direkt módszerek*. Ezek véges sok, meghatározott számú lépés után megadják a megoldást. Ezektől eltérnek az *iterációs módszerek*, amelyek minden iterációs lépésben jobb és jobb közelítését adják a megoldásnak (de azt általában nem érik el véges számú lépesben). Kis vagy közepes méretű mátrixokra, illetve nagy sávmátrixokra a direkt módszerek a jobbak. Ide tartoznak a Gauss elimináció változatai.

A Gauss elimináció a kiindulási általános lineáris egyenletrendszert olyanná alakítja, amelynek mátrixa háromszögmátrix (trianguláris mátrix).

LINEÁRIS EGYENLETRENDSZEREK / PÉLDA

PÉLDA.

Tekintsünk egy egyszerű lineáris egyenletrendszert:

$$2x + 4y = 6$$

$$4x - 2y = 2.$$

Osszuk el az első egyenletet kettővel, így az első együttható egy lesz (ez a lépés nem feltétlenül szükséges, csak az eredmény leolvasását könnyíti meg):

$$x + 2y = 3$$

$$4x - 2y = 2.$$

Ezután vonjuk ki az első egyenlet négyszeresét a másodikból, így abban már csak egy ismeretlen marad:

$$x + 2y = 3$$

$$-10y = -10.$$

Innen világos, hogy $y = 1$, és ezt behelyettesítve az első egyenletbe, azt kapjuk, hogy

$$x + 2 = 3,$$

tehát $x = 1$.

Vegyük észre, hogy a fenti eljárásban az egyenletrendszer ekvivalens alakjait állítottuk elő úgy, hogy a transzformáció végén egy olyan alakot kapjunk, amelyből a megoldás könnyen, illetve könnyebben meghatározható.

Két egyenletrendszert akkor tekintünk *ekvivalensnek*, ha a megoldásaik halmaza megegyezik.

A megengedett transzformációk:

- egy egyenletnek egy nem nulla számmal való beszorzása, és
- egy egyenlet konstansszorosának hozzáadása egy másik egyenlethez.

Az átalakításokkal háromszögmátrixot, vagy diagonális mátrixot hozunk létre.

BEHELYETTESÍTÉS FELSŐ HÁROMSZÖGMÁTRIX ESETÉN

Tekintsük a lineáris egyenletmegoldásnak azt az esetét, amikor az A mátrix felső háromszögmátrix. Ekkor a megoldás komponensei közvetlenül számíthatók. Az általános alak 3×3-as mátrixra:

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

A megoldás ekkor:

$$\begin{aligned} x_3 &= \frac{b_3}{u_{33}} \\ x_2 &= \frac{b_2 - u_{23}x_3}{u_{22}} \\ x_1 &= \frac{b_1 - u_{12}x_2 - u_{13}x_3}{u_{11}}. \end{aligned}$$

Az általános feladatra vonatkozó Matlab program:

```
1  function x = UTriSol(U,b)
2  n = length(b);
3  x = zeros(n,1);
4  for j=n:-1:2
5     x(j) = b(j)/U(j,j);
6     b(1:j-1) = b(1:j-1) - x(j)*U(1:j-1,j);
7  end
8  x(1) = b(1)/U(1,1);
```

Az algoritmus végrehajthatóságának nyilvánvaló feltétele, hogy az U mátrix főátlóbeli elemei ne legyenek nullák.

A ciklusban végzett ún. *vektorizált műveletek* gyorsabban hajthatók végre, mint ciklus formába szervezve.

A ciklus műveletigénye $1+2+\cdots+(n-1) = \frac{n(n-1)}{2}$ darab szorzás és összeadás. Az osztásokból n darab van. A műveletigény tehát $\mathcal{O}(n^2/2)$ összeadás és szorzás.

Az alsó háromszögmátrix esetén a behelyettesítés lényegében azonos módon történik (persze eltérő képletekkel).

ELIMINÁCIÓS MÁTRIXOK

Olyan mátrixot viszonylag egyszerű megadni, amellyel balról való beszorzás azt eredményezi, hogy egy a vektor k-adik alatti együtthatói eltűnnek:

$$M_k a = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -a_{k+1}/a_k & & \\ & & \vdots & \ddots & \\ & & -a_n/a_k & & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \\ a_{k+1} \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Itt az M_k mátrixban minden ki nem írt együttható nulla, és az a_k együtthatót *generálóelem*nek hívjuk, magát az M_k mátrixot pedig *eliminációs mátrix*nak, vagy *Gauss-transzformáció*nak. Az eliminációs mátrixoknak érvényesek a következő tulajdonságai:

- M_k alsó háromszögmátrix, a főátlójában csupa egyessel, tehát reguláris (nem szinguláris) mátrix,
- $M_k = I - me_k^T$, ahol $m = (0, \ldots, a_{k+1}/a_k, \ldots, a_n/a_k)^T$, és e_k az egységmátrix k-adik oszlopa,
- $M_k^{-1} = I + me_k^T$, azaz M_k és az inverze csak a főátlón kívüli elemek előjelében különböznek. Az M_k^{-1} alsó háromszögmátrixot L_k-val is fogjuk jelölni.
- Két eliminációs mátrix szorzatát könnyű megadni a $k < j$ esetre:

  $$M_k M_j = (I - me_k^T)(I - te_j^T) = I - me_k^T - te_j^T + me_k^T te_j^T = I - me_k^T - te_j^T,$$

  mivel $e_k^T t = 0$. Két eliminációs mátrix szorzata tehát a kettő kompozíciója a fenti képlet értelmében.
- Hasonlóan az inverzekre is érvényes $L_k L_j = I + me_k^T + te_j^T$ (ha $k < j$).

PÉLDA. Legyen $a = (1, 2, -3)^T$, ekkor

$$M_1 a = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ -3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

ELIMINÁCIÓS MÁTRIXOK / 2

Az előző oldal példájának Matlab megvalósítása:

```
>> a = [1 2 -3]';
>> m = [0 a(2)/a(1) a(3)/a(1)]';
>> M1 = eye(3);
>> M1 = M1-m*M1(:,1)'
M1 =
     1     0     0
    -2     1     0
     3     0     1
>> M1*a
ans =
     1
     0
     0
```

Itt A' az A mátrix transzponáltját jelöli, a 3×3-as egységmátrixot pedig az `eye(3)` utasítás adja. Az M_1 eliminációs mátrix inverze, L_1 a következő módon határozható meg:

```
>> L1 = M1^-1
L1 =
     1     0     0
     2     1     0
    -3     0     1
```

LU FELBONTÁS, GAUSS ELIMINÁCIÓ

A lineáris egyenletrendszer megoldáshoz az $Ax = b$ egyenletrendszerből eliminációs mátrixokkal olyant szeretnénk kialakítani, amelynek baloldalán egy felső háromszögmátrix van, így a behelyettesítést végre tudjuk hajtani.

$$MAx = M_{n-1} \cdots M_1 Ax = M_{n-1} \cdots M_1 b = Mb.$$

Itt M_1 az A mátrix a_{11} elemére, mint generáló elemre támaszkodik, M_2 az $M_1 A$ mátrix 2., főátlóbeli elemére stb.:

$$\underset{A}{\begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix}} \quad \underset{\rightarrow}{M_1} \quad \underset{M_1 A}{\begin{pmatrix} \times & \times & \times \\ 0 & x & x \\ 0 & x & x \end{pmatrix}} \quad \underset{\rightarrow}{M_2} \quad \underset{M_2 M_1 A}{\begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & x \end{pmatrix}}.$$

Itt $\times$ az épp végrehajtott szorzás előtti mátrix elemeit jelöli, x pedig az utolsó lépésben megváltozott, nem feltétlen nulla együtthatókat.

Vegyük észre, hogy ez az eljárás az eredeti A mátrixot két új, háromszögmátrix szorzatára bontja: $A = LU$, ahol

$$L = M^{-1} = (M_{n-1} \cdots M_1)^{-1} = M_1^{-1} \cdots M_{n-1}^{-1} = L_1 \cdots L_{n-1}.$$

A konstrukcióból adódik, hogy $U = MA$ egy felső háromszögmátrix, az L diagonális elemei pedig egyesek.

Az eredeti feladatunkat most az LU felbontás segítségével egyszerűsíthetjük: $Ax = b$ helyett vegyük az $LUx = b$ egyenletrendszert. Ezt két lépésben oldjuk meg, először az

$$Ly = b$$

egyenletrendszert az új, mesterséges y változóra, majd az

$$Ux = y$$

egyenletet. Vegyük észre, hogy $y = Mb$.

LU FELBONTÁS, GAUSS ELIMINÁCIÓ / 2

Az $Ax = b$ lineáris egyenletrendszer megoldására szolgáló Gauss elimináció formálisan a következő lépésekből áll:

1. Az A mátrix LU felbontása (egy alsó és egy felső háromszögmátrixra),
2. $Ly = b$ megoldása y-ra,
3. $Ux = y$ megoldása x-re.

A megfelelő Matlab program az `LTriSol`, `UTriSol` és az `LU` eljárásokra építve:

```
>> [L,U] = LU(A);     % LU felbontas
>> y = LTriSol(L,b); % elore behelyettesites
>> x = UTriSol(U,y); % hatra behelyettesites
```

A felhasznált LU eljárás Matlab kódja:

```
1 function [L,U] = LU(A)
2 [m,n] = size(A);
3 for k=1:n-1
4    A(k+1:n,k) = A(k+1:n,k)/A(k,k);
5    A(k+1:n,k+1:n) = A(k+1:n,k+1:n)-A(k+1:n,k)*A(k,k+1:n);
6 end
7 L = eye(n,n) + tril(A,-1);
8 U = triu(A);
```

Itt `tril` és `triu` az alsó-, illetve felső háromszögmátrixokat képezik az argumentum mátrixból kivágva. A második argumentum pedig a főátlótól való eltérés mértékét adja, tehát `tril(A,-1)` azt az alsó háromszögmátrixot adja, amelynek főátlójában is nullák vannak.

LU FELBONTÁS, GAUSS ELIMINÁCIÓ / PÉLDA

Tekintsük a következő egyszerű lineáris egyenletrendszert:

$$\begin{aligned} 2x_1 + 4x_2 - 2x_3 &= 2, \\ 4x_1 + 9x_2 - 3x_3 &= 8, \\ -2x_1 - 3x_2 + 7x_3 &= 10. \end{aligned}$$

Ennek szokásos, mátrix formájú alakja:

$$\begin{pmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 8 \\ 10 \end{pmatrix}.$$

Ahhoz, hogy az első oszlopban a főátló alatti elemek eltűnjenek, az első sor kétszeresét kell kivonni a második sorból, és egyszerűen hozzáadni a harmadik sorhoz:

$$M_1 A = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{pmatrix} = \begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{pmatrix},$$

$$M_1 b = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 8 \\ 10 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 12 \end{pmatrix}.$$

Már csak a harmadik sor második elemét kell nullává változtatni, ehhez vonjuk ki a második sort a harmadikból:

$$M_2 M_1 A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{pmatrix} = \begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix},$$

$$M_2 M_1 b = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 4 \\ 12 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 8 \end{pmatrix}.$$

Ezzel megkaptuk az eredeti lineáris egyenletrendszerünk ekvivalens, felső háromszögmátrixos alakját:

$$\begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 8 \end{pmatrix}.$$

LU FELBONTÁS, GAUSS ELIMINÁCIÓ / PÉLDA FOLYTATÁSA

A kapott,

$$\begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 8 \end{pmatrix}$$

egyenletrendszer megoldása visszahelyettesítéssel: $x = (-1, 2, 2)^T$.

Az LU felbontásban az L mátrix:

$$L = L_1 L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix}.$$

Ezzel az A mátrix felbontása:

$$A = \begin{pmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix} = LU.$$

FŐELEMKIVÁLASZTÁS

Az LU felbontás persze csak akkor sikeres, ha az A mátrix nem szinguláris, és minden generáló elem nullától különböző. Ha a generálóelem (*főelem*, pivotelem) nulla, akkor még lehetséges lehet a felbontás – a mátrix átrendezése után, ami szintén ekvivalens feladathoz vezet. Ezt az eljárást *főelemkiválasztás*nak hívjuk.

Például az

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

mátrix reguláris, mégse hajtható végre az LU felbontás közvetlenül. Szabad viszont a sorokat felcserélni (természetesen a lineáris egyenletrendszerben a jobboldali vektor elemei együtt mozognak). A sorcserék utáni mátrix triviális LU felbontása:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = LU,$$

illetve a sorok felcserélését elvégző P, ún. *permutációs mátrix*szal formálisan:

$$PA = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = LU,$$

Jegyezzük meg, hogy ha a főátló alatt az LU felbontás egy fázisában csak nulla elem van, akkor $M_k = I$ választással tovább folytathatjuk a felbontást. Ha ez olyankor következett be, amikor a főátlóbeli elem nulla volt, akkor az így kapott LU felbontás nem lesz reguláris. Ez nem csoda, hiszen az eredeti A mátrix is szinguláris kellett hogy legyen. Tekintsük például a következő esetet:

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = LU.$$

A csak a sorcseréken alapuló főelemkiválasztást *részleges főelemkiválasztás*nak is nevezzük. A hibaforrásoknál elmondottak miatt érdemes nagy abszolút értékű főelemeket választani.

Amikor a főelemet a teljes még szóbajövő részmátrixból választjuk, *teljes főelemkiválasztás*ról beszélünk.

Bár a permutáció elrontja az alsó- és felső háromszögmátrixokat, az egyenletrendszer megoldása szempontjából ezek teljes értékűek maradnak, így a jelölésben is szokás az eredeti LU alakot megtartani.

LU FELBONTÁS MATLABBAN

A Matlab saját `lu` rutinja kezeli a részleges főelemkiválasztást, és a meghívási alakja függvényében különféle formában adja meg az eredményt:

```
>> A=[2 4 -2;4 9 -3;-2 -3 7];
>> [L,U,P]=lu(A)
L =
   1.0000         0         0
  -0.5000    1.0000         0
   0.5000   -0.3333    1.0000
U =
   4.0000    9.0000   -3.0000
        0    1.5000    5.5000
        0         0    1.3333
P =
        0         1         0
        0         0         1
        1         0         0
```

Három kimeneti argumentummal tehát megkapjuk az L, U és P mátrixot is (utóbbi a permutációs mátrix). Ha csak két argumentumot adunk meg, akkor L permutált lesz:

```
>> [L,U]=lu(A)
L =
   0.5000   -0.3333    1.0000
   1.0000         0         0
  -0.5000    1.0000         0
U =
   4.0000    9.0000   -3.0000
        0    1.5000    5.5000
        0         0    1.3333
```

LU FELBONTÁS MATLABBAN / 2

Ha nem adunk meg output argumentumot, akkor egyetlen eredmény mátrixot kapunk:

```
>> lu(A)
ans =
   4.0000   9.0000  -3.0000
  -0.5000   1.5000   5.5000
   0.5000  -0.3333   1.3333
```

Ez a mátrix tömör formában tartalmazza az U felső háromszögmátrixot, és a permutálatlan M mátrixot. Utóbbi tehát a beszorzási együtthatókat adja (ezek az L elemeinek negáltjai). Ez a memóriatakarékos forma jellemző a számítógépes belső tárolásra. A sorvektor cseréket is csak egy segédvektoron szokás átvezetni, a sorok tényleges cseréje nem történik meg.

Az LU felbontás műveletigénye közel $n^3/3$ szorzás és összeadás. A Matlab maga is mérte a végrehajtott lebegőpontos műveletek számát (a 6.5 változat már nem támogatja):

```
>> n = 100;
>> flops(0); % Beallitja a szamlalot 0-ra
>> lu(rand(n));
>> flops
ans =
     661650
```

Ez az érték megfelel a megadott $2/3\ n^3$ összevont műveletigénynek. Ha a lineáris egyenletrendszer jobb oldalán egy egyszerű vektor van (nem több vektor, vagy mátrix), akkor a visszahelyettesítés további n^2 szorzást és összeadást igényel. Emiatt n növekedésével a lineáris egyenletmegoldás műveletigényének mind nagyobb része fordítódik az LU felbontásra.

A felhasznált CPU idő mérésére a következő rövid program használható:

```
>> n = 500;
>> tic % inditja az orat
>> lu(rand(n));
>> toc
elapsed_time =
   0.4110
```

A kapott eredmény (700 Mhz-es Pentium III gépen) 0.41 másodperc volt. Kétszeres méretű feladatra 2.5 másodperc kellett, ez összhangban van a számított műveletigénnyel (`rand(n)` $n \times n$-es véletlen mátrixot ad egyenletes eloszlással).

MÁTRIXINVERTÁLÁS

Az előzőekben bevezetett Gauss-eliminációt sok feladatra lehet használni az egyenletmegoldáson túl, például mátrixok invertálására is. Tekintsük az

$$Ax_i = e_i$$

egyenletrendszerek megoldásait, ahol e_i az egységmátrix i-edik oszlopvektora. Ha az A mátrixot háromszögmátrix alakra hoztuk, akkor az $MA = U$ alakra támaszkodva n darab felső háromszögmátrix alakú egyenletrendszert kell megoldani:

$$Ux_i = Me_i.$$

Egy lineáris egyenlet r. Mvelete igénye n^3

Az x_i vektorokból álló X mátrix lesz A inverze, azaz $AX = I$.

A Matlabban az `inv(A)` adja meg egy A mátrix inverzét, ha az létezik:

```
>> A = [3 4; 4 5];
>> B = inv(A)
B =
     -5      4
      4     -3
```

Ha nem létezik a kért inverz, akkor a következő üzenetet kapjuk:

```
>> A = [3 4; 3 4];
>> B = inv(A)
Warning:  Matrix is singular to working precision.
(Type "warning off MATLAB:singularMatrix" to suppress
this warning.)
B =
    Inf    Inf
    Inf    Inf
```

MÁTRIXOK NORMÁJA

A $||.|| : \mathbb{R}^{n\times n} \to \mathbb{R}$ leképezést mátrixnormának mondjuk, ha a következő tulajdonságok teljesülnek:

(i) $||A|| > 0$, ha A nem a zérusokból álló mátrix, ha zérus mátrix akkor lehet a normája 0

(ii) $||\alpha A|| = |\alpha|||A||$, alfa = szám ha egy számmal beszorzok egy matrixot akkor az a szám abs * matrix norm

(iii) $||A + B|| \leq ||A|| + ||B||$, és

(iv) $||AB|| \leq ||A||\,||B||$.

Az egyes vektornormákhoz mátrixnormák tartoznak a következők szerint:

$$||A|| := \sup_{x\neq 0} \frac{||Ax||}{||x||} = \sup_{||x||=1} ||Ax||.$$

Így minden vektornormához rendeltünk egy mátrixnormát. A definíció miatt ezeknek nehézkes lehet a számítása. Az egyszerűen meghatározható, kis műveletigényű mátrixnormák:

$$||A||_1 = \max_j \sum_{i=1}^{n} |a_{ij}|,$$

$$||A||_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|,$$

$$||A||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2}.$$

Az *egyes norma*, $||A||_1$ tehát az egyes oszlopok abszolútérték-összegének maximuma, *a végtelen norma*, $||A||_\infty$ a sorok abszolútérték-összegének maximuma, a *Frobenius norma* $||A||_F$ pedig a mátrixelemek abszolútértékeinek négyzetösszegének a gyöke.

A Matlab alapból a kettes normát adja (`norm(A)` = `norm(A,2)`):

```
>> A = [3 4; 3 4];
>> norm(A)
ans =
     7.0711
```

MÁTRIXOK KONDÍCIÓSZÁMA

Egy négyzetes *mátrix kondíciószáma*

$$\mathrm{cond}(A) = ||A||\ ||A^{-1}||.$$

Szinguláris mátrixok kondíciószámát végtelennek definiáljuk. A kondíciószám függ a választott normától, így $\mathrm{cond}_p(A)$ a p-normára vonatkozó kondíciószámot jelöli. Másrészt a *jól-* ($\mathrm{cond}(A) \approx 1$), illetve *rosszul kondícionáltság* ($\mathrm{cond}(A) \gg 1$) jórészt független a választott normától.

A rosszul kondícionáltság azzal szokott járni, hogy az érintett mátrix közel szinguláris.

A kondíciószám tulajdonságai:

- $\mathrm{cond}(I) = 1$, tehát az egységmátrix kondíciószáma egy[23].
- $\mathrm{cond}(A) \geq 1$
- $\mathrm{cond}(A) = \mathrm{cond}(A^{-1})$
- ha $c \neq 0$, akkor $\mathrm{cond}(cA) = \mathrm{cond}(A)$
- a P permutációs mátrixokra $\mathrm{cond}(P) = 1$[23]
- a D diagonális mátrixokra $\mathrm{cond}(D) = \max|d_{ii}| / \min|d_{ii}|$ [23]
- $\mathrm{cond}(AB) \leq \mathrm{cond}(A)\ \mathrm{cond}(B)$
- $\mathrm{cond}_\infty(A) = \mathrm{cond}_1(A^T)$
- $\mathrm{cond}_F(A) = \mathrm{cond}_F(A^T)$

PÉLDA. A $0.1I$ $n \times n$-es mátrix determinánsa 10^{-n}, tehát nagy n-re nagyon kicsi lehet, mégis $\mathrm{cond}(0.1I) = 1$.

PÉLDA. Az az A mátrix, amely az egységmátrixtól annyiban tér el, hogy a főátló felett csupa -1 van, olyan, hogy $\det(A) = 1$, és mégis $\mathrm{cond}_1(a) = n2^{n-1}$, tehát nagy n-re csaknem szinguláris.

Az egyes normákon alapuló kondíciószámokat Matlabban a következő utasítással lehet kérni:

`>> cond(A,p)`, ahol `p` pl. `1`, `2`, vagy `inf` lehet

[23] Ha a norma vektornormából származtatott.

CHOLESKY FELBONTÁS[24]

Lineáris egyenletrendszerek megoldása során szokás megkülönböztetni speciális szerkezetű mátrixok alapján néhány típust:

- A szimmetrikus és pozitív definit, x vektor != 0 valós x^TAx>0 x * A * x^T > 0 x != 0 vektor
- az A sávmátrix,
- A ritka mátrix (csak kevés eleme különbözik nullától).

Az első két esetre használatos módszerek a Gauss-elimináció közeli rokonai, a ritka mátrixok esetét bonyolultabb eljárásokkal szokás hatékonyan megoldani. Itt most a valós mátrixokra mutatjuk be a Cholesky felbontást (a komplex eset megoldása hasonlóan történik).

Ha az A mátrix szimmetrikus és pozitív definit, akkor az LU felbontás $U = L^T$ alakban létezik, tehát

$$A = LL^T,$$

ahol L alsó háromszögmátrix, amelynek diagonális elemei pozitív számok (nem feltétlen egyesek). Az ilyen felbontást *Cholesky felbontás*nak hívjuk.

A szimmetrikus pozitív definit mátrixú lineáris egyenletrendszerek megoldása Matlabban:

```
function [x] = LGPD(A, b);
R = chol(A);
y = R'\b;
x = R\y;
```

Itt `chol(A)` adja az A mátrix Cholesky felbontását, és `R\y` adja az $Rx = y$ egyenlet megoldását. Ezt a tömör lineáris algebrai írásmódot baloldali osztásnak (left division) nevezik.

A Cholesky felbontás numerikusan stabilis, műveletigénye $\frac{1}{3}n^3 + \mathcal{O}(n^2)$, feleannyi, mint egy általános mátrix LU felbontásáé.

[24] Kiejtése: 'Koleszki', vö. J. Waldvogel: Pronunciation of Cholesky, Lanczos and Euler, NA-DIGEST, v90n10, 1990.

CHOLESKY FELBONTÁS, PÉLDA

Az előző oldalon bevezetett Cholesky felbontásra tekintsünk egy egyszerű, áttekinthető példát. Legyen a felbontandó mátrix a felbontások nemnulla pozícióival, és annak észrevételével, hogy $L_{11} = \sqrt{A_{11}}$:

$$LL^T = \begin{pmatrix} 2 & 0 & 0 \\ \times & \times & 0 \\ \times & \times & \times \end{pmatrix} \begin{pmatrix} 2 & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{pmatrix} = \begin{pmatrix} 4 & 2 & 4 \\ 2 & 5 & 2 \\ 4 & 2 & 5 \end{pmatrix} = A.$$

A következő lépés L első oszlopának (és így L^T első sorának kitöltése azon az alapon, hogy ehhez az L_{11} értékkel való osztást kell csak elvégezni:

$$LL^T = \begin{pmatrix} 2 & 0 & 0 \\ 1 & \times & 0 \\ 2 & \times & \times \end{pmatrix} \begin{pmatrix} 2 & 1 & 2 \\ 0 & \times & \times \\ 0 & 0 & \times \end{pmatrix} = \begin{pmatrix} 4 & 2 & 4 \\ 2 & 5 & 2 \\ 4 & 2 & 5 \end{pmatrix} = A.$$

Ezután az A_{22} elemre koncentrálunk, ez előáll $A_{22} = L_{21}^2 + L_{22}^2$ alakban, ahonnan az új elem $L_{22} = \sqrt{A_{22} - L_{21}^2}$:

$$LL^T = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ 2 & \times & \times \end{pmatrix} \begin{pmatrix} 2 & 1 & 2 \\ 0 & 2 & \times \\ 0 & 0 & \times \end{pmatrix} = \begin{pmatrix} 4 & 2 & 4 \\ 2 & 5 & 2 \\ 4 & 2 & 5 \end{pmatrix} = A.$$

Az L_{32} értéket az új L_{22} és A_{32} alapján számíthatjuk:

$$LL^T = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ 2 & 0 & \times \end{pmatrix} \begin{pmatrix} 2 & 1 & 2 \\ 0 & 2 & 0 \\ 0 & 0 & \times \end{pmatrix} = \begin{pmatrix} 4 & 2 & 4 \\ 2 & 5 & 2 \\ 4 & 2 & 5 \end{pmatrix} = A.$$

És végül $A_{33} = L_{31}^2 + L_{32}^2 + L_{33}^2$ alapján $L_{33} = \sqrt{A_{33} - L_{31}^2 - L_{32}^2} = \sqrt{5 - 4 - 0}$:

$$LL^T = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 2 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 2 & 4 \\ 2 & 5 & 2 \\ 4 & 2 & 5 \end{pmatrix} = A.$$

CHOLESKY FELBONTÁS, MATLAB PÉLDA

A Cholesky felbontásra tekintsünk egy egyszerű, áttekinthető példát a Matlab használatával. Legyen a megoldandó lineáris egyenletrendszer

$$\begin{pmatrix} 4 & 2 & 3 \\ 2 & 4 & 2 \\ 3 & 2 & 4 \end{pmatrix} x = \begin{pmatrix} 9 \\ 8 \\ 9 \end{pmatrix}.$$

Ennek megoldása az $x = (1,1,1)^T$, és a mátrix szimmetrikus, pozitív definit. A Matlabban a következő lépésekkel oldhatjuk meg a Cholesky felbontás alapján:

```
>> A = [4 2 3; 2 4 2; 3 2 4];
>> b = [9 8 9]';
>> R = chol(A)

R =

     2.0000 1.0000 1.5000
          0 1.7321 0.2887
          0      0 1.2910

>> y = R'\b

y =

     4.5000
     2.0207
     1.2910

>> x = R\y

x =

     1.0000
     1.0000
     1.0000
```

A QR ORTOGONÁLIS FELBONTÁS HÁROMSZÖGMÁTRIX ALAKRA

Egy Q négyzetes mátrixot *ortogonális*nak nevezünk, ha $QQ^T = Q^TQ = I$. Egyseg matrix az 'I'
Egy ortogonális transzformáció megőrzi a vektorok euklideszi normáját:

$$||Qx||_2^2 = (Qx)^TQx = x^Tx = ||x||_2^2.$$

Mivel az ortogonális transzformációk a kettes normát megtartják, ezért numerikusan stabilisak.

Mivel $\text{cond}_2(Q) = 1$, ezért a visszahelyettesítés ugyanúgy kondícionált, mint az eredeti feladat.

Lineáris egyenletrendszer megoldása az $A = QR$ felbontással a következők szerint megy: megoldandó a

$$Rx = Q^Tb$$

felső háromszögmátrixú egyenletrendszer. A Matlab megfelelő:

```
[Q,R] = qr(A,0);
x = R\(Q'*b);
```

reguláris det(A) = 0

TÉTEL. Tetszőleges A négyzetes valós reguláris mátrixnak létezik az $A = QR$ felbontása ortogonális és felső háromszögmátrixra.

BIZONYÍTÁS. Mivel A reguláris, ezért az A^TA mátrix pozitív definit, tehát létezik ennek egy R^TR Cholesky felbontása. Legyen ekkor a Q mátrix egyenlő AR^{-1}-el. Megmutatjuk, hogy QR az A mátrix ortogonális-trianguláris felbontása. R nyilván felső háromszögmátrix, és A egyenlő a két mátrix szorzatával. Azt kell még igazolni, hogy Q ortogonális.

$$\begin{aligned} Q^TQ &= (AR^{-1})^T(AR^{-1}) = (R^{-1})^TA^TAR^{-1} = (R^{-1})^T(R^TR)R^{-1} \\ &= ((R^T)^{-1}R^T)(RR^{-1}) = II = I, \end{aligned}$$

Amivel igazoltuk, hogy Q ortogonális. □

MÁTRIXOK SAJÁTÉRTÉKEI ÉS SAJÁTVEKTORAI

Legyen adott egy A négyzetes mátrix. Adjuk meg a λ számot és az $x \neq 0$ vektort úgy, hogy

$$Ax = \lambda x.$$

lambda az a gorog betu

Itt λ az A mátrix *sajátértéke*, x pedig a *sajátvektora*. Egy hasonló definícióval értelmezhető a baloldali sajátérték és sajátvektor:

lambda irányába megnyújtjuk A matrixot ha lambda >1 akkor megtartjaaz irányát és megnyúl. ha 0>lambda >1 akkor megtartja az irányát és csökken a hossza a vektornak ha lambda < 0 akkor megváltozik az iránya

$$y^T A = \lambda y^T.$$

Mivel y sajátvektora az A^T mátrixnak, ezért a továbbiakban csak az első feladatot vizsgáljuk. Egy mátrix sajátértékei halmazát a mátrix *spektrumá*nak hívjuk, és $\lambda(A)$-val jelöljük. A *spektrálrádiusz* pedig a $\max\{|\lambda| : \lambda \in \lambda(A)\}$ mennyiséget jelenti, és $\rho(A)$-val jelöljük.

az a sugár amit ha az origó körül felveszünk akkor az összes sajátérték ezen a körül belül lesz

A sajátvektorok irányába eső vektorokat az A mátrix megnyújtja az illető sajátvektorhoz tartozó sajátértéknek megfelelően. Így egy mátrixszal reprezentált lineáris transzformáció hatását egyszerűen megadhatjuk a sajátértékei és sajátvektorai ismeretében. Számos fizikai és matematikai feladat (pl. rezonanciavizsgálat, illetve differenciálegyenletek stabilitási kérdései) sajátértékszámítással jól megoldható.

A sajátértékek vizsgálata komplex számokra is kiterjeszthető. Az egyetlen lényeges különbség, hogy akkor az A^T transzponált mátrix helyett a konjugált transzponáltat, A^H-t kell használni.

A sajátértékek és a sajátvektorok nem egyértelműek:

- például az egységmátrixnak az 1 n-szeres sajátértéke,
- egy sajátvektorral együtt annak minden nem nulla számmal szorzottja is ugyanahhoz a sajátértékhez tartozó sajátvektora.

A sajátértékeket, sajátvektorokat meghatározhatjuk az

$$(A - \lambda I)x = 0$$

homogén lineáris egyenletrendszerből. Ennek pontosan akkor van a nulla vektortól különböző megoldása, ha az $A - \lambda I$ mátrix szinguláris. Emiatt a sajátértékeket leírja a

$$\det(A - \lambda I) = 0$$

egyenlet. Ennek baloldalán levő n-edfokú polinomot az A mátrix *karakterisztikus polinom*jának nevezzük.

MÁTRIXOK SAJÁTÉRTÉKEI ÉS SAJÁTVEKTORAI / PÉLDÁK

Néhány mátrix a sajátértékeivel, az azokhoz tartozó sajátvektorokkal és a karakterisztikus polinommal:

$$\begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}: \quad \lambda = 1, x = (1,0)^T, \quad \lambda = 2, x = (0,1)^T, \quad \text{és} \quad (1-\lambda)(2-\lambda),$$

$$\begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}: \quad \lambda = 1, x = (1,0)^T, \quad \lambda = 2, x = (1,1)^T, \quad \text{és} \quad (1-\lambda)(2-\lambda),$$

$$\begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix}: \quad \lambda = 2, x = (1,1)^T, \quad \lambda = 4, x = (1,-1)^T, \quad \text{és} \quad \lambda^2 - 6\lambda + 8,$$

$$\begin{pmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{pmatrix}: \quad \lambda = 2, x = (1,1)^T, \quad \lambda = 1, x = (-1,1)^T, \quad \text{és} \quad \lambda^2 - 3\lambda + 2$$

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}: \quad \lambda = \imath, x = (1,\imath)^T, \quad \lambda = -\imath, x = (\imath,1)^T, \quad \text{és} \quad \lambda^2 + 1,$$

ahol $\imath = \sqrt{-1}$. Ha egy valós mátrixnak van komplex sajátértéke, akkor a sajátérték komplex konjugáltja is sajátérték.

Azt mondjuk, hogy a $||.||_{(v)}$ vektornorma *kompatibilis* az $||.||_{(m)}$ mátrixnormával, ha tetszőleges A négyzetes mátrixra igaz

$$||Ax||_{(v)} \leq ||A||_{(m)}||x||_{(v)}.$$

A származtatott mátrixnormák nyilvánvalóan kompatibilisek az ősükkel, azzal a vektornormával, amellyel definiáltuk őket.

A SAJÁTÉRTÉKEK KORLÁTAI

TÉTEL. Tetszőleges A mátrixra és bármely mátrixnormára

$$\rho(A) \leq ||A||,$$

azaz a spektrálsugár nem nagyobb mint a mátrix normája.

BIZONYÍTÁS. Legyen $||.||_v$ az adott mátrixnormával kompatibilis vektornorma, λ_m az A mátrix maximális abszolút értékű sajátértéke, és x_m a hozzá tartozó sajátvektor. Ekkor

$$\rho(A) = \max_{1 \leq i \leq n} |\lambda_i| = |\lambda_m|,$$

tehát

$$\rho(A)||x_m||_v = |\lambda_m| \; ||x_m||_v = ||\lambda_m x_m||_v = ||Ax_m||_v \leq ||A|| \; ||x_m||_v.$$

Innen a nem nulla $||x_m||_v$-vel leosztva kapjuk az igazolandó egyenlőtlenséget. □

TÉTEL. (Gersgorin) Az A mátrix összes sajátértéke benne van a

$$K_i = \left\{ z \in C \;\middle|\; |z - a_{ii}| \leq \sum_{k=1, k \neq i}^{n} |a_{ik}| \right\}$$

körök $K = \cup_{i=1}^{n} K_i$ egyesítésében.

Érvényes ennek a tételnek egy olyan élesítése, hogy amennyiben a definiált körök egy m, és egy $n - m$ körből álló halmazra bonthatók úgy, hogy a két részhalmaz pontjai diszjunktak, akkor az első körhalmaz multiplicitással számolva pontosan m darab sajátértéket tartalmaz.

PÉLDA. Alkalmazzuk a fenti elméleti eredményeket az alábbi mátrixra:

$$A = \begin{pmatrix} 1 & 1 & -1 \\ 2 & -2 & 0 \\ 1 & 0 & 5 \end{pmatrix}.$$

Ezek szerint minden sajátérték abszolút értéke legfeljebb $||A||_1 = 6$, $||A||_\infty = 6$, és $||A||_F = \sqrt{37}$, azaz $\rho(A) \leq 6$. A sajátértékekből pedig kettő van az 1 középpontú, és 2 sugarú, valamint a -2 középpontú, 2 sugarú körök uniójában, és egy sajátérték van az 5 középpontú és 1 sugarú körben (a komplex síkon).

A HATVÁNYMÓDSZER

A gyakorlatban sok esetben csak bizonyos sajátértékek meghatározására van szükség. A *hatványmódszer* a legnagyobb abszolút értékű sajátérték meghatározására szolgál. Az alapalgoritmus a következő iteráción alapul:

CSAK KÖZELÍTO MEGOLDÁST AD

x-k iterálvektort megszorozzuk az A matrixal ebbl kapunk egy y-k-t a k egy iterációs szám

$$y^k = Ax^k, \; x^{k+1} = y^k/||y^k||.$$

nem lehet 0 vektor(csupa 0 számból álló vektro)

A kiindulási vektor olyan kell hogy legyen, hogy $x^0 \neq 0$, és x^0 nem merőleges a legnagyobb abszolút értékű sajátértékhez tartozó sajátvektorra. A módszer másik neve *von Mieses vektoriterációja*. Ha $|\lambda_1| > |\lambda_i|$ minden $i = 2, \ldots, n$-re, és a λ_1-hez tartozó sajátvektor u_1, akkor

$$\lim_{k\to\infty} \text{sign}(\lambda_1)^k x^k = u_1, \text{ és } \lim_{k\to\infty} ||y^k||/||x^k|| = |\lambda_1|.$$

Ha λ_2 a második legnagyobb abszolútértékű sajátérték, akkor a konvergencia sebességére érvényes:

$$\frac{y_i^k}{x_i^k} = \lambda_1 + \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right), \text{ illetve } \frac{(x^k)^T y^k}{(x^k)^T x^k} = \lambda_1 + \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\right).$$

Itt $((x^k)^T y^k)/((x^k)^T x^k)$ az ún. *Rayleigh-féle hányados*.

Tegyük fel, hogy A reguláris, ekkor minden sajátértéke nullától különböző. Az $Ax = \lambda x$ egyenletből

$$x = A^{-1}(\lambda x) = \lambda A^{-1} x$$

adódik, és innen

$$\lambda^{-1} x = \lambda^{-1}\lambda A^{-1} x = A^{-1} x.$$

Eszerint ha az A mátrix sajátértéke λ, és az ehhez tartozó sajátvektor x, akkor az A^{-1} mátrix egy sajátértéke λ^{-1} az x sajátvektorral. Ezen a felismerésen alapul az *inverz hatványmódszer*, vagy más néven a *Wieland-féle inverz iteráció*:

$$Ay = x^k, \; x^{k+1} = y/||y||,$$

amely tehát a legkisebb abszolút értékű sajátértéket és a hozzá tartozó sajátvektort közelíti meg.

ejsd szigmaAz $A - \sigma I$ mátrixra alkalmazva a hatványmódszert az A mátrixnak a σ számtól legtávolabbi sajátértékét kapjuk, a hozzá tartozó sajátvektorral.

A HATVÁNYMÓDSZER MEGVALÓSÍTÁSA MATLABBAN

A következő egyszerű Matlab program a hatványmódszer segítségével meghatározza a legnagyobb abszolút értékű sajátértéket az iterált véletlen vektor minden komponenséből:

```
function lambda = hatv(A);
x = [rand(1) rand(1) rand(1)]';
for i=1:100
   y = A*x;
   lambda = y./x
   r = (x'*y)/(x'*x)
   x = y/norm(y);
end
```

Most hajtsuk végre a programot a sajátértékek korlátainál vizsgált mátrixra:

$$A = \begin{pmatrix} 1 & 1 & -1 \\ 2 & -2 & 0 \\ 1 & 0 & 5 \end{pmatrix}.$$

A kapott becslések a maximális abszolút értékű sajátértékre a komponensenkénti becsléssel és a Rayleigh hányadossal az első néhány iterációban:

iteráció	1	2	3	4	5
komponensenkénti	40.7831	7.8748	4.9474	-28.6488	4.7033
Rayleigh	-0.4284	-1.1691	0.4980	3.2966	4.1575

iteráció	6	7	8	9	10
komponensenkénti	6.4596	-499.5966	5.0727	7.5500	4.8025
Rayleigh	4.6264	4.6601	4.7092	4.7009	4.7091

A maximális abszolút értékű sajátértékhez tartozó kapott sajátvektor pedig

$$[-0.2807, -0.0826, 0.9562]^T.$$

A helyes sajátérték 4.7066, és a hozzá tartozó sajátvektor

$$[-0.2805, -0.0837, 0.9562]^T.$$

MÁTRIXOK TÍPUSAI

Az A $n \times n$-es mátrix karakterisztikus polinomja egy n-edfokú polinom, így az algebra alaptételéből következik, hogy az A mátrixnak legfeljebb n különböző sajátértéke lehet.

Mátrixok sajátértékének meghatározására szolgáló algoritmusok közötti választást döntően befolyásolják a következő kérdésekre adott válaszok:

- Minden sajátértékre szükség van, vagy csak egy részükre?
- Csak a sajátértékek kellenek, vagy a sajátvektorok is?
- A mátrix valós vagy komplex?
- Milyen a mátrix: inkább kicsi és nemigen van benne nulla, vagy nagy és ritka?
- Vannak-e a mátrixnak olyan speciális tulajdonságai, mint a szimmetria?

A fontosabb mátrix-tulajdonságokat az alábbiakban foglaljuk össze:

A *szimmetrikus*, ha $A = A^T$, pl. $\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$, de nem szimmetrikus pl. $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.

A *ortogonális*, ha $A^T A = I$, pl. $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, de nem ortogonális pl. $\begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$.

A *pozitív definit*, ha szimmetrikus, és $x^T Ax > 0$ minden nem nulla x vektorra. Pl. $\begin{pmatrix} 3 & 2 \\ 2 & 3 \end{pmatrix}$, de nem pozitív definit pl. $\begin{pmatrix} 1 & 2 \\ 2 & 0 \end{pmatrix}$.

A *normális*, ha $AA^T = A^T A$, pl. $\begin{pmatrix} 1 & 1 \\ \imath & 3+2\imath \end{pmatrix}$, de nem normális pl. $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$.

Ezeknek megvan a megfelelőjük a komplex mátrixok körében is, de ott a transzponálás, A^T szerepét átveszi a *konjugált transzponálás*, A^H. A megfelelő nevek ezután: *Hermite-szimmetrikus* (vagy *hermitikus*), *unitér* (vagy *H-ortogonális*), és normális (vagy *H-normális*).

MÁTRIXOK SAJÁTÉRTÉKEI, A HASONLÓSÁGI TRANSZFORMÁCIÓ

Azt mondjuk, hogy az A mátrix *hasonló* a B mátrixhoz, ha létezik olyan T reguláris *hasonlósági mátrix*, hogy

$$B = T^{-1}AT.$$

Tekintsük most a B mátrix egy λ sajátértékét. Erre érvényes, hogy:

$$By = \lambda y \qquad \Rightarrow \qquad T^{-1}ATy = \lambda y \qquad \Rightarrow \qquad A(Ty) = \lambda(Ty),$$

tehát λ sajátértéke A-nak is (még ha más, Ty sajátvektorral is). Eszerint a hasonlósági transzformációk azok, amelyekkel egy mátrixot átalakíthatunk olyan kedvezőbb alakra, amelyről a sajátértékeket már könnyen le tudjuk olvasni.

A fordított összefüggés nem érvényes: ha két mátrixnak megegyeznek a sajátértékei, akkor még nem feltétlenül hasonló mátrixok. Például ez érvényes a következő mátrixokra (ahogy könnyen belátható):

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \text{és} \qquad \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Ebből az is következik, hogy hasonlósági mátrixokkal nem mindig tudunk egy mátrixból hasonlósági transzformációkkal egy diagonális mátrixot kapni. Márpedig nyilvánvaló, hogy a diagonális mátrixok sajátértékei a diagonálisbeli számok, és a sajátvektorok pedig az ezeknek megfelelő egységvektorok.

A sajátvektorok definíciójából adódik, hogy ha az A mátrix sajátvektoraiból állítjuk össze az X mátrixot, akkor például

$$AX = \begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix} = XD,$$

ahol D az a diagonális mátrix, amely A sajátértékeit tartalmazza. Tekintsük általános esetben a következő mátrixokat:

$$D = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n), \quad \text{és} \quad X = (x_1, x_2, \ldots, x_n),$$

ahol λ_i az A mátrix i-edik sajátértéke, és x_i a hozzá tartozó sajátvektor. Ha az X mátrix reguláris, akkor

$$A = XDX^{-1}, \quad \text{és} \quad D = X^{-1}AX.$$

Ebből azonnal adódik a következő

MÁTRIXOK SAJÁTÉRTÉKEI, HASONLÓSÁGI TRANSZFORMÁCIÓ / 2

TÉTEL. Az A mátrix akkor és csak akkor diagonalizálható, ha létezik n elemű lineárisan független sajátvektor-rendszere.

A legtöbb, amit általános esetben is elérhetünk, az az, hogy az A mátrixot hasonlósági transzformációkkal ún. *Jordan alak*ra hozzuk: olyan mátrixot kapunk, amelyben a főátló mellett a felső mellékátló tartalmazhat még nullától különböző elemeket. Sajnos a Jordan alakra hozás numerikusan nem stabilis, ezért nem szokás használni. Emiatt nincs is a Matlabban `jordan` nevű algoritmus.

numerikusan Az viszont stabil eljárás, amely hasonlósági transzformációkkal egy mátrixot felső háromszögmátrixszá alakít (ez az ún. *Schur-féle normálalak*), és ekkor is a főátló elemei adják a sajátértékeket.

Az alábbi táblázat egyes mátrix típusokat tartalmaz a hozzájuk tartozó hasonlósági mátrix fajtával és a transzformált mátrix típusával.

A	T	$B = T^{-1}AT$
páronként eltérő sajátértékek	reguláris	diagonális
szimmetrikus	ortogonális	valós diagonális
Hermite-szimmetrikus	unitér	valós diagonális
normális	unitér	diagonális
tetszőleges	unitér	felső háromszögmátrix
tetszőleges	reguláris	Jordan alakú

Minden esetben a B diagonális elemei a sajátértékek, és az első négy esetben a T oszlopai sajátvektorok.

A Matlabban a sajátértékszámításra a következő fontosabb eljárások állnak rendelkezésre:

`eig`	az összes sajátérték és sajátvektor meghatározása
`hess`	Hessenberg alakra hozás
`schur`	Schur-féle normálalakra (felső háromszögmátrix) hozás
`eigs`	néhány sajátérték és sajátvektor
`condeig`	a sajátértékek kondíciószáma
`poly`	a mátrix karakterisztikus polinomját adja
`eigshow`	illusztrálja a 2 $\times$ 2-es mátrixok sajátértékeit

Keressük ismét az

$$A = \begin{pmatrix} 1 & 1 & -1 \\ 2 & -2 & 0 \\ 1 & 0 & 5 \end{pmatrix}$$

mátrix sajátértékeit és sajátvektorait. Csak a sajátértékeket kapjuk az `eig` utasítással (a sajátérték angolul eigenvalue):

```
>> eig(A)
ans =
     -2.5440
      1.8374
      4.7066
```

Ha a sajátvektorokat is kérjük, akkor a következő formában kell írni:

```
>> [V,D] = eig(A)
V =
      0.2623 -0.8539 -0.2805
     -0.9644 -0.4450 -0.0837
     -0.0348  0.2700  0.9562
D =
     -2.5440        0        0
           0  1.8374        0
           0        0  4.7066
```

Érdemes tudni, hogy az `eig` utasítás 16 különböző algoritmusból választja ki az adott feladathoz legjobban igazodót, amellyel az a leghatékonyabban megoldható.

Az `eig` utasítás egy ügyes alternatívája az `eigs` parancs, amely nagy méretű, vagy ritka mátrixokra lényegesen gyorsabban ad választ – bár alapértelmezésben csak a 6 legnagyobb abszolút értékű sajátértéket, és az azokhoz tartozó sajátvektorokat. Bővebb információt a `doc eigs` utasítás ad.

A jelen példára azonos eredményt kapunk (mert a választott A mátrix kis méretű).

MÁTRIXOK SAJÁTÉRTÉKEI, MATLAB ELJÁRÁSOK / 2

A sajátértékek meghatározásának egy másik útját jelenti, amikor az A mátrix karakterisztikus polinomját határozzuk meg, majd annak gyökeit:

```
>> poly(A)
ans =
     1.0000 -4.0000 -8.0000 22.0000

>> roots(ans)
ans =
      4.7066
     -2.5440
      1.8374
```

Figyeljük meg, hogy az előző utasítás eredményére hivatkozhatunk az `ans` névvel. Az eljárás értékéből levon az, hogy a `poly` használja a `eig` utasítást...

A Schur-féle normálakra is hozhatjuk a mátrixunkat:

```
>>schur(A)
ans =
     -2.5440  0.8748 -0.8048
           0  1.8374  1.8939
           0       0  4.7066
```

Ennek főátlójából olvashatók le a sajátértékek.

SAJÁTÉRTÉKEK KONDÍCIONÁLTSÁGA

A *sajátértékek kondícionáltsága* azt mutatja meg, hogy az adott mátrix együtthatóiban való kis változások milyen hatásúak a sajátértékekre. Ez különbözik attól, hogy az adott mátrix kis eltérései mennyire hatnak a vele felírt lineáris egyenletrendszer megoldásaira. Egy adott mátrix sajátértékei is eltérhetnek a mátrix hibáira vonatkozó érzékenységük szerint.

Az A mátrix λ *sajátértéke kondíciószámát* az

normalizálás
sajátvektor/sajátvektor normája
A kapott sajátvektor normája 1

$$\frac{1}{|y^H x|}$$

A x = Lambda x jobboldali
y^T A = Lambda y^T baloldali

képlet határozza meg, ahol x és y az A mátrix λ sajátértékéhez tartozó bal- és jobboldali normalizált sajátvektorai, tehát

x^h = elemi transzponált

$$x^H x = y^H y = 1.$$

Más szóval a λ sajátérték kondíciószáma a bal- és a jobb sajátvektorai által bezárt szög koszinuszának reciproka. Ha ez a szám nagy, akkor λ rosszul kondícionált.

Egy szimmetrikus, vagy Hermite-szimmetrikus mátrix azonos sajátértékekhez tartozó bal- és jobb sajátvektorai megegyeznek. Emiatt

$$y^H x = x^H x = 1$$

adódik a normalizált sajátvektorokra, tehát a sajátértékek mindig jól kondícionáltak. Általánosabban az is igaz, hogy a normális mátrixok (amelyekre $A^T A = AA^T$) sajátértékei is jól kondícionáltak.

SAJÁTÉRTÉKEK KONDÍCIONÁLTSÁGA / PÉLDA

Vizsgáljuk meg az

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0.999 & 1 \\ 0 & 0 & 2 \end{pmatrix}$$

mátrix sajátértékei kondícionáltságát. A hozzá tartozó normalizált (szokásos, jobb) sajátvektorok 4 jegyre:

$$x_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \qquad x_2 = \begin{pmatrix} \text{-1} \\ 0.0005 \\ 0 \end{pmatrix}, \qquad x_3 = \begin{pmatrix} 0.9623 \\ 0.1923 \\ 0.1925 \end{pmatrix}.$$

Az ezekhez tartozó bal sajátvektorok:

$$y_1 = \begin{pmatrix} 0.0004 \\ 0.7066 \\ \text{-0.7066} \end{pmatrix}, \qquad y_2 = \begin{pmatrix} 0 \\ 0.7075 \\ \text{-0.7068} \end{pmatrix}, \qquad y_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

A sajátértékek persze a diagonálisbeli elemek:

$$\lambda_1 = 1, \;\; \lambda_2 = 0.999 \;\; \text{és} \;\; \lambda_3 = 2.$$

A sajátértékek kondíciószámára a következő értékeket kapjuk rendre:

$$\frac{1}{|y_1^T x_1|} = 2.5 \cdot 10^3,$$

$$\frac{1}{|y_2^T x_2|} = 2.8269 \cdot 10^3,$$

$$\frac{1}{|y_3^T x_3|} = 5.1948.$$

Eszerint a $\lambda_1 = 1$ és a $\lambda_2 = 0.999$ sajátértékek viszonylag rosszul kondícionáltak, míg λ_3 jól kondícionált. A tényleges változások szemléltetéséhez változtassuk meg a mátrixban a_{31} értékét 0.000001-re. Az új mátrix sajátértékei rendre

$$\tilde{\lambda}_1 = 0.9995 + 0.0013\imath, \;\; \tilde{\lambda}_2 = 0.9995 - 0.0013\imath \;\; \text{és} \;\; \tilde{\lambda}_3 = 2.$$

Ez azt jelenti, hogy az első két sajátérték a mátrixbeli kis eltéréshez képest nagyon megváltozott, ráadásul ezek komplex számokká váltak, míg a harmadik sajátérték nem változott.

A mátrixok sajátértékei kondíciószámát a `condeig` Matlab utasítás adja.

SAJÁTVEKTOROK KONDÍCIONÁLTSÁGA

A *sajátvektorok kondícionáltsága* a hozzájuk tartozó sajátérték kondícionáltságától és a többi sajátértéktől mért eltérésétől függ. A többszörös, vagy egymáshoz közeli sajátértékek sajátvektorai rosszul kondícionáltak lehetnek, különösen, ha a mátrix nem reguláris. Utóbbi esetben a mátrix sajátértékfeladatának kondícionáltságát diagonális hasonlósági transzformációkkal lehet javítani. A Matlab ebből a célból a `balance` utasítást ajánlja.

PÉLDA. Vizsgáljuk meg most az

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.99 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

diagonális mátrix sajátvektorai kondícionáltságát. A sajátértékek nyilvánvalóan a főátlóbeli $\lambda_1 = 1$, $\lambda_2 = 0.99$ és $\lambda_3 = 2$. Mivel a mátrix szimmetrikus, és minden sajátérték egyszeres, ezért a sajátértékek jól kondícionáltak. A sajátvektorok:

$$x_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \qquad x_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \qquad x_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Tekintsük most a kissé megváltoztatott újabb mátrixot:

$$\tilde{A} = \begin{pmatrix} 1 & 0.0001 & 0 \\ 0.0001 & 0.99 & 0 \\ 0 & 0 & 2 \end{pmatrix}.$$

Ennek sajátértékei ismét 1, 0.99 és 2. Az új sajátvektorok viszont:

$$\tilde{x}_1 = \begin{pmatrix} -1 \\ -0.01 \\ 0 \end{pmatrix}, \qquad \tilde{x}_2 = \begin{pmatrix} 0.01 \\ -1 \\ 0 \end{pmatrix}, \qquad \tilde{x}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

A harmadik sajátvektor nem változott, de a többi kettő meglehetősen. Ennek az az oka, hogy a megfelelő sajátértékek közel vannak egymáshoz.

MÁTRIXOK SAJÁTÉRTÉKEI, LR TRANSZFORMÁCIÓ

Az *LR transzformáció* a H. Rutishauser által 1959-ben megadott eljárás, amely az LR felbontáson alapulva egy olyan mátrixsorozatot ad meg, amellyel meghatározhatók az A mátrix sajátértékei.

Az eljárás az $A_1 = A$ mátrixból indul, és a következő két lépésből áll:

$$A_k = L_k R_k,$$

$$A_{k+1} = R_k L_k.$$

Tehát az aktuális mátrixot felbontjuk egy alsó- és egy felső háromszögmátrixra, majd a következő iterált mátrixot úgy kapjuk, hogy ezeket fordított sorrendben összeszorozzuk. Az L_k alsó háromszögmátrix főátlójában egyesek vannak.

Legyen $T_k = L_1 L_2 \cdots L_k$ és $U_k = R_k R_{k-1} \cdots R_1$. Ekkor érvényes a következő

SEGÉDTÉTEL. Ha minden $k = 1, 2, \ldots$ indexre létezik a fent definiált A_k mátrix, akkor

i, az A mátrix hasonló A_k-hoz,

ii, $A_k = (L_1 L_2 \cdots L_{k-1})^{-1} A_1 (L_1 L_2 \cdots L_{k-1}) = T_{k-1}^{-1} A_1 T_{k-1}$,

iii, $A_1^k = T_k U_k$.

A^k = A matrix k adik hatványa
T^k T mátrix k adik iterátora

BIZONYÍTÁS. Az i, állítás adódik az ii,-ből. A többire alkalmazzunk teljes indukciót. A $k = 2$ esetben mindkettő nyilvánvalóan teljesül. Vegyük először az ii, állítást. Mivel L_k reguláris, ezért $A_{k+1} = L_k^{-1} A_k L_k$, és az indukciós feltevés miatt

$$\begin{aligned} A_{k+1} &= L_k^{-1} (L_1 L_2 \cdots L_{k-1})^{-1} A_1 (L_1 L_2 \cdots L_{k-1}) L_k \\ &= (L_1 L_2 \cdots L_k)^{-1} A_1 (L_1 L_2 \cdots L_k) \\ &= T_k^{-1} A_1 T_k. \end{aligned}$$

Tehát az ii, állítás igaz a $k+1$ esetre is. Tekintsük most az iii, pontot hasonló gondolatmenetet követve:

$$\begin{aligned} A_1^{k+1} &= A_1 A_1^k = A_1 T_k U_k = A_1 (L_1 L_2 \cdots L_k)(R_k R_{k-1} \cdots R_1) \\ &= (L_1 L_2 \cdots L_k) A_{k+1} (R_k R_{k-1} \cdots R_1) \\ &= (L_1 L_2 \cdots L_k) L_{k+1} R_{k+1} (R_k R_{k-1} \cdots R_1) = T_{k+1} U_{k+1}. \end{aligned}$$

Amivel a harmadik részt is igazoltuk. □

MÁTRIXOK SAJÁTÉRTÉKEI, LR TRANSZFORMÁCIÓ / FOLYTATÁS

Az LR algoritmus konvergencia-tétele:

TÉTEL. Ha az A valós négyzetes mátrix teljesíti az alábbi feltételeket:

1. az $A_k = L_k R_k$ trianguláris felbontás létezik minden $k \geq 1$-re,
2. az A sajátértékeit alkalmasan indexelve érvényes

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n| > 0$$

(amiből adódik, hogy A_1 reguláris és sajátértékei mind valósak és egyszeresek), és

3. megadható az A mátrixot diagonalizáló olyan X mátrix, amellyel egyrészt

$$X^{-1} A_1 X = D = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n),$$

másrészt létezik mind az $X = \tilde{L}\tilde{R}$, mind az $X^{-1} = \bar{L}\bar{R}$ trianguláris felbontás,

akkor az $\{A_k\}$, $\{R_k\}$ és az $\{L_k\}$ mátrix sorozatok mind konvergensek:

$$\lim_{k\to\infty} A_k = \lim_{k\to\infty} R_k = \begin{pmatrix} \lambda_1 & \mathrm{x} & \cdots & \mathrm{x} \\ 0 & \lambda_2 & \cdots & \mathrm{x} \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix} \qquad \text{és} \qquad \lim_{k\to\infty} L_k = I.$$

Az első két sorozat tehát egy olyan mátrixhoz tart, amelynek főátlójában az A_1 mátrix sajátértékei vannak nagyság szerint rendezve. Az LR algoritmus legfőbb hátránya, hogy még erős feltételek teljesülése esetén is előfordulhat, hogy valamely k-ra az A_k iterált mátrix nem bontható fel $L_k R_k$ alakban.

Az iteráció megállási feltétele lehet például, hogy az iterált A_k mátrix főátló alatti elemei négyzetösszege kisebb egy előre megadott kis $\epsilon > 0$ számnál.

Általános A mátrixra egy LR iteráció műveletigénye $\mathcal{O}(n^3)$, felső Hessenberg alakú A mátrixra $\mathcal{O}(n^2)$, és tridiagonális kiindulási mátrixra (amelynek a főátló mellet legfeljebb a mellékátlókban vannak nem nulla elemei) pedig $\mathcal{O}(n)$.

LR TRANSZFORMÁCIÓ / MATLAB

A Matlab maga nem tartalmazza az LR transzformációs iterációt, de nem nehéz ennek egy egyszerű programját megírni:

```
function R = lr(A);
for k=1:10
     [L,R] = lu(A)
     A = R*L
end
```

Vizsgáljuk meg, hogy az LR transzformáció hogyan működik a korábban már megismert mátrixra:

$$A = \begin{pmatrix} 1 & 1 & -1 \\ 2 & -2 & 0 \\ 1 & 0 & 5 \end{pmatrix}.$$

Emlékeztetésül: ennek sajátértékei -2.5440, 1.8374 és 4.7066. A programunk minden iterációban három mátrixot ír ki, az L, R és A mátrixokat. Meglepetéssel láthatjuk, hogy a kapott L mátrixok nem mindig lesznek alsó háromszögmátrixok – de ennek csak az az oka, hogy az alkalmazott LU felbontás a kért formátum kedvéért az L mátrixba integrálta a P permutációs mátrix hatását is (lásd az LU felbontás részletes tárgyalását).

Lássuk, hogy hogyan alakulnak az R_k mátrixok főátlóbeli elemei és a főátló alatti elemek négyzetösszege (eltérés):

iteráció	R_{11}	R_{22}	R_{33}	eltérés
1	2.0000	2.0000	5.5000	17.3750
2	2.7500	3.0000	-2.6667	20.3343
3	6.0000	-2.5556	1.4348	5.0446
4	5.3333	-2.5761	1.6013	0.5999
5	4.8125	-2.5159	1.8170	0.2339
6	4.8052	-2.5719	1.7801	0.0513
7	4.7135	-2.5217	1.8509	0.0187
8	4.7259	-2.5622	1.8169	0.0052
9	4.7052	-2.5301	1.8480	0.0019
10	4.7109	-2.5546	1.8281	0.0006

Megállapíthatjuk, hogy már 5 iteráció után felismerhetők a sajátértékek, és 10 iteráció alatt meglehetős pontossággal állnak már rendelkezésre. Az is jól látszik, hogy a sajátértékek az abszolút értékeik csökkenő sorrendje szerint rendezettek.

EGYENLETRENDSZEREK ITERÁCIÓS MÓDSZEREI

Ahogy korábban már tárgyaltuk, kis és közepes lineáris egyenletrendszerekre, továbbá a nagyméretű sávmátrixokra a direkt módszerek ajánlhatók, a nagy méretű, de nem feltétlen ritka mátrixokkal rendelkező lineáris egyenletrendszerek megoldására iterációs eljárások használatosak. Az utóbbi feladatokra a korábban bemutatott algoritmusok használhatatlanul nagy műveletigényűek lehetnek.

PÉLDA. Tekintsünk egy egyszerű lineáris egyenletrendszert:

$$\begin{aligned} 4x_1 - x_2 + x_3 &= 4 \\ 2x_1 - 4x_2 + x_3 &= -1 \\ -2x_1 + x_2 + 5x_3 &= 4. \end{aligned}$$

Az egyenletrendszer megoldása az $x = (1,1,1)^T$ vektor. Ezeket az egyenleteket írhatjuk olyan alakban, hogy minden egyenlet bal oldalára rendezünk egy-egy változót:

$$\begin{aligned} x_1 &= \frac{4 + x_2 - x_3}{4} \\ x_2 &= \frac{1 + 2x_1 + x_3}{4} \\ x_3 &= \frac{4 + 2x_1 - x_2}{5}. \end{aligned}$$

Most vegyünk egy $x_0 = (1,2,3)^T$ indulóvektort, és azt az utóbbi egyenletrendszer jobboldalába helyettesítve határozzuk meg a baloldalon álló változók új értékeit, a következő iterált vektort. Az iteráció a következő vektorokat adja:

k	x_1	x_2	x_3
0	1.0000	2.0000	3.0000
1	0.7500	1.5000	0.8000
2	1.1750	0.8250	0.8000
3	1.0063	1.0375	1.1050
4	0.9831	1.0294	0.9950
5	1.0086	0.9903	0.9874
6	1.0007	1.0011	1.0054
7	0.9989	1.0017	1.0001
8	1.0004	0.9995	0.9992

JACOBI ITERÁCIÓ

Ezt a bevezetett eljárást *Jacobi iteráció*nak nevezzük. Bár az előző példa ígéretes, mégis vannak esetek, amikor a Jacobi iteráció nem konvergál. Tekintsük például a vizsgált lineáris egyenletrendszer következő átrendezését (csak annyi történt, hogy az utolsó sort vettük előre):

$$\begin{aligned} -2x_1 + x_2 + 5x_3 &= 4 \\ 4x_1 - x_2 + x_3 &= 4 \\ 2x_1 - 4x_2 + x_3 &= -1 \end{aligned}$$

Az ehhez tartozó iterációs egyenletek:

$$\begin{aligned} x_1 &= \frac{-4 + x_2 + 5x_3}{2} \\ x_2 &= -4 + 4x_1 + x_3 \\ x_3 &= -1 - 2x_1 + 4x_2. \end{aligned}$$

Használjuk ismét az $x_0 = (1, 2, 3)^T$ indulóvektort, ekkor az

```
>> y = [(-4+x(2)+5*x(3))/2 -4+4*x(1)+x(3) -1-2*x(1)+4*x(2)]
>> x = y
```

iterációs lépések ismétlése a következő divergáló vektorsorozatot adja:

k	x_1	x_2	x_3
0	1.0000	2.0000	3.0000
1	6.5000	3.0000	5.0000
2	12	27	-2
3	6.5	42	83
4	226.5	105	154
5	435.5	1056	-34
6	441	1704	3352
7	9230	5112	5933
8	17387	42849	1987
9	26390	71529	136622
10	377317.5	242178	233335

A JACOBI ITERÁCIÓ KONVERGENCIÁJA

Írjuk fel az $Ax = b$ lineáris egyenletrendszer Jacobi iterációját a következő alakban (feltéve, hogy a diagonális elemek nem nullák):

$$x^{(k+1)} = D^{-1}b - D^{-1}(A - D)x^{(k)},$$

ahol D az A mátrix diagonális elemeiből álló mátrix. Ugyanerre az egyenletre használjuk a következő, egyszerűbb jelölést:

$$x^{(k+1)} = Bx^{(k)} + c,$$

ahol tehát $B = -D^{-1}(A - D)$ és $c = D^{-1}b$. Az általános iterált vektor ekkor felírható

$$x^{(k)} = c + B(c + B(c + \ldots x^{(0)}) \ldots) = c + Bc + B^2c + \cdots + B^{k-1}c + B^kx^{(0)}$$

alakban. Vizsgáljuk meg két egymást követő iterált vektor eltérését:

$$||x^{(k+1)} - x^{(k)}|| = ||B^kc + (B^{k+1} - B^k)x^{(0)}|| \leq ||B^k||\ ||c + (B - I)x^{(0)}||.$$

Innen látszik, hogy az $\{x^{(k)}\}$ vektorsorozat konvergenciája és a $||B||$ norma egynél kisebb volta összefügg. Utóbbi azt jelenti, hogy a konvergenciához

$$||D^{-1}(A - D)|| < 1$$

szükséges. Vegyük most a végtelen normát. A

$$||B||_\infty = ||D^{-1}(A - D)||_\infty = \max_i \sum_{j=1}^{n} |b_{ij}| = \max_i \sum_{j=1, j\neq i}^{n} |a_{ij}/a_{ii}| < 1$$

összefüggésekből következik, hogy

$$\sum_{j=1, j\neq i}^{n} |a_{ij}| < |a_{ii}|$$

teljesül minden $i = 1, 2, \ldots, n$-re (ahol A az $n \times n$-es mátrix). Ha az utóbbi egyenlőtlenség teljesül egy mátrixra, akkor azt mondjuk, hogy az *szigorúan diagonális domináns*.

A fentiek alapján a diagonális dominancia elegendő a Jacobi iteráció konvergenciájához. Nézzük az összefüggéseket részletesebben.

AZ ITERÁCIÓS MÓDSZEREK KONVERGENCIÁJA

Vizsgáljuk meg az $x^{(k+1)} = Bx^{(k)} + c$ iteráció által definiált $\{x^{(k)}\}$ sorozat konvergenciáját. Jelöljük az eredeti egyenletrendszerünk megoldását x^*-al. Az $e_k = x^{(k)} - x^*$ eltérésre a következő állítás érvényes:

ek eltérés vektor
xk közelít megoldás
x* valódi megoldás

TÉTEL. Tetszőleges $x^{(0)}$ kezdővektor esetén a k-adik közelítés eltérése az x^* megoldástól $e_k = B^k e_0$.

BIZONYÍTÁS. Használjunk teljes indukciót. $k = 0$-ra nyilvánvalóan igaz az állítás, mert ekkor $e_0 = B^0 e_0 = Ie_0$ adódik. Tegyük fel most, hogy egy adott k-ra teljesül az összefüggés, és vizsgáljuk meg a $k+1$ esetét.

$$\begin{aligned} e_{k+1} &= x^{(k+1)} - x^* = (Bx^{(k)} + c) - (Bx^* + c) = B(x^{(k)} - x^*) \\ &= Be_k = B(B^k e_0) = B^{k+1} e_0 \end{aligned}$$

B=-D^-1 * (A-D)
c=D^-1 * b
D = n*n diag(A)

ek eltérés vektor
xk közelít megoldás
x* valódi megoldás

Ami igazolja a tétel állítását. □

A tétel következménye, hogy ha a B mátrix *nilpotens* (azaz van olyan j index, amire $B^j = 0$), akkor $B^j e_0 = 0$, tehát az iterációs eljárás véges sok lépésben megtalálja a megoldást. Ez a feltétel teljesül pl. akkor, ha B *szigorúan trianguláris*, tehát olyan háromszögmátrix, amelynek főátlójában csak nullák vannak.

Akkor mondjuk, hogy egy iterációs sorozat *globálisan konvergens*, ha minden indulóvektorral ugyanazt a megoldást kapjuk.

Mivel a globális konvergencia esetén tetszőleges kezdővektorból indulva megkapjuk a megoldást, ezért szokásos kezdővektornak a c vektort használni.

Iterációs eljárásokat használunk olyan esetben is, amikor az eliminációs eljárással kapott, kerekítési hibákkal terhelt közelítő megoldást kell pontosítani.

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

AZ ITERÁCIÓS MÓDSZEREK KONVERGENCIÁJA / 2.

A következő elméleti állítás azon a lineáris algebrai tényen alapul, hogy egynél kisebb spektrálsugárú mátrixhoz létezik olyan mátrixnorma, amelyik a mátrixra egynél kisebb értéket ad.

TÉTEL. Az $x^{(k+1)} = Bx^{(k)} + c$ iteráció akkor és csak akkor globálisan konvergens, ha $\rho(B) < 1$.

BIZONYÍTÁS. Tegyük fel először, hogy az eljárás globálisan konvergens, és mégis $\rho(B) \geq 1$. Ekkor a maximális abszolút értékű λ sajátértékre $|\lambda| \geq 1$. Válasszuk ezután az $x^{(0)}$ indulóvektort úgy, hogy a λ sajátértékhez tartozó x normált sajátvektor legyen az e_0 eltérésvektor. Mivel az előző tétel szerint ebben az esetben $e_k = B^k e_0 = \lambda^k e_0$, ezért

$$||e_k|| = ||\lambda^k e_0|| = \left|\lambda^k\right| ||e_0|| = |\lambda|^k 1 \geq 1$$

azaz e_k nem konvergál nullához. Ez pedig ellentmond annak, hogy a módszer globálisan konvergens.

Induljunk ki most abból, hogy $\rho(B) < 1$. Ekkor van olyan $||.||$ mátrixnorma, hogy $||B|| < 1$. Ezzel pedig tetszőleges kezdővektorra és vektornormára igaz, hogy

$$||e_k|| = ||B^k e_0|| \leq ||B^k|| \, ||e_0|| \leq ||B||^k \, ||e_0|| \to 0.$$

Tehát az eljárás globálisan konvergens. □

Ha megadható olyan $||.||$ mátrixnorma, amelyre $||B|| < 1$, akkor érvényesek a következő hibabecslések:

$$||e_{k+1}|| \leq ||B|| \, ||e_k||,$$

$$||e_k|| \leq ||B||^k ||e_0||,$$

$$||e_k|| \leq \frac{||B||}{1 - ||B||} ||x^{(k)} - x^{(k-1)}||,$$

$$||e_k|| \leq \frac{||B||^k}{1 - ||B||} ||x^{(1)} - x^{(0)}||,$$

$$||e_k|| \leq \frac{||B||^{k+1}}{1 - ||B||} ||c||, \text{ ha } x^{(0)} = c.$$

Ha az A mátrix diagonális domináns, akkor a hozzá tartozó B mátrixra a spektrálsugár kisebb mint egy, tehát a belőle adódó Jacobi iteráció globálisan konvergens.

GAUSS-SEIDEL ITERÁCIÓ

A Jacobi iteráció tanulmányozása során felmerülhet, hogy mivel egy a megoldáshoz közelítő vektorsorozatot akarunk generálni, ezért talán javítani lehet a konvergencia sebességén, ha az első új vektorkomponensek meghatározása után az iterációs képlet jobb oldalán már ezeket az új értékeket használjuk. Ez a *Gauss-Seidel iteráció.* A kapcsolódó képletek az eredeti feladatra:

$$\begin{aligned} x_1^{k+1} &= \frac{4 + x_2^k - x_3^k}{4} \\ x_2^{k+1} &= \frac{1 + 2x_1^{k+1} + x_3^k}{4} \\ x_3^{k+1} &= \frac{4 + 2x_1^{k+1} - x_2^{k+1}}{5}. \end{aligned}$$

Ismételjük meg most a Jacobi iterációval végzett első kísérletet. Az eredménye:

k	x_1	x_2	x_3
0	1.0000	2.0000	3.0000
1	0.7500	1.3750	0.8250
2	1.1375	1.0250	1.0500
3	0.9938	1.0094	0.9956
4	1.0034	1.0006	1.0013
5	0.9998	1.0002	0.9999
6	1.0001	1.0000	1.0000
7	1.0000	1.0000	1.0000

Bár néhány iterált értékéből nemigen lehet megállapítani a konvergencia sebességét, azt mégis leszögezhetjük, hogy a kapott eredményünk lényegesen jobb, mint ami a Jacobi iterációval adódott.

Vigyázat, az

```
y = [(4+x(2)-x(3))/4  (1+2*y(1)+x(3))/4  (4+2*y(1)-y(2))/5]
```

Matlab utasítás ugyanazt az eredményt adja, mint amit a Jacobi iterációval kaptunk, mert a Matlab a jobboldal kiértékelése során az `y` vektor korábbi értékeivel számol, nem egyszerűen balról jobbra határozza meg az `x` vektort!

A JACOBI ITERÁCIÓ A MATLABBAN

Az alábbi Matlab program az `AX = B` lineáris egyenletrendszert oldja meg Jacobi iterációval az `X = P` kezdeti vektorból indulva. Egy olyan sorozatát generálja a `P` vektoroknak, amelyek a megoldáshoz konvergálnak, feltéve, hogy `A` szigorúan diagonális domináns.

```
function X = jacobi(A,B,P,delta,max1)
N = length(B);
for k=1:max1
  for j=1:N
   X(j)=(B(j)-A(j,[1:j-1,j+1:N])*P([1:j-1,j+1:N]))/A(j,j);
  end
  err = abs(norm(X'-P));
  relerr = err/(norm(X)+eps);
  P = X';
  if (err < delta)|(relerr < delta)
    break
  end
end
X = X'
```

- Itt `A` egy `N x N`-es nemszinguláris mátrix, `B`, `X` és `P` pedig `N` hosszú vektorok, `delta` a `P` toleranciája, és `max1` az iterációk megengedett maximális száma.
- Az `A(j,[1:j-1,j+1:N])` az `A` mátrix `j`-edik sorának főátlón kívüli elemeiből álló sorvektor tömör írásmódja.
- Figyeljük meg a gépi pontosság, az `eps` használatát!
- Miért kell a vektorok transzponálása?
- Milyen meglepő dolgot lehet még találni a programszövegben?

A GAUSS-SEIDEL ITERÁCIÓ A MATLABBAN

Az alábbi Matlab program az `AX = B` lineáris egyenletrendszert oldja meg Gauss-Seidel iterációval az `X = P` kezdeti vektorból indulva. Egy olyan sorozatát generálja a `P` vektoroknak, amelyek a megoldáshoz konvergálnak, feltéve, hogy `A` szigorúan diagonális domináns.

```
function X = gseidel(A,B,P,delta,max1)
N = length(B);
for k=1:max1
  for j=1:N
    if j==1
        X(1)=(B(1)-A(1,2:N)*P(2:N))/A(1,1);
      elseif j==N
        X(N)=(B(N)-A(N,1:N-1)*(X(1:N-1))')/A(N,N);
      else
        X(j)=(B(j)-A(j,1:j-1)*(X(1:j-1))'
             -A(j,j+1:N)*P(j+1:N))/A(j,j);
    end
  end
  err = abs(norm(X'-P));
  relerr = err/(norm(X)+eps);
  P = X';
  if (err < delta)|(relerr < delta)
    break
  end
end
X = X'
```

- Itt `A` ismét egy `N x N`-es nemszinguláris mátrix, `B`, `X` és `P` pedig `N` hosszú vektorok, `delta` a `P` toleranciája, és `max1` az iterációk megengedett maximális száma.
- A Gauss-Seidel iteráció programja csak a belső ciklus magjában tér el a Jacobi iteráció programjától.

MÁTRIXOK REGULÁRIS SZÉTVÁGÁSAI

Az iteratív eljárásokhoz az eredeti lineáris egyenletrendszert ekvivalens átalakításokkal $x = Bx + c$ alakra kell hozni. A következő módon járhatunk el: az A együtthatómátrixot bontsuk fel $A = R - S$ alakban, ahol R egy reguláris mátrix. Az ilyen felbontásokat az A mátrix *reguláris szétvágásá*nak hívjuk.

A reguláris szétvágás segítségével könnyen meg lehet adni az iterációs alakot:

$$\begin{aligned} (R-S)x &= b \\ Rx &= Sx + b \\ x &= R^{-1}Sx + R^{-1}b \end{aligned}$$

és innen

$$x^{(k+1)} = Bx^{(k)} + c,$$

ahol $B = R^{-1}S = I - R^{-1}A$ és $c = R^{-1}b$. A továbbiakban feltesszük, hogy A *erősen reguláris*, azaz A reguláris, és $a_{ii} \neq 0$ teljesül minden $i = 1, \ldots, n$-re.

Tekintsük az A mátrix következő felbontását: $A = D - L - U$, ahol D az A mátrix diagonális elemeiből álló diagonális mátrix, U szigorúan felső-, L pedig szigorúan alsó háromszögmátrix. Használni fogjuk az $\hat{U} = D^{-1}U$ és az $\hat{L} = D^{-1}L$ jelölést is.

Vegyük először az $R = D$, $S = L + U$ reguláris szétvágást. Ekkor a Jacobi iterációra

$$B = D^{-1}(L + U) = \hat{L} + \hat{U}$$

az iterációs mátrix, és

$$c = D^{-1}b.$$

Az $R = D - L$, $S = U$ reguláris szétvágásra

$$B = (D - L)^{-1}U = (I - \hat{L})^{-1}\hat{U}$$

és

$$c = (I - \hat{L})^{-1}D^{-1}b$$

a Gauss-Seidel iterációt adja.

MÁTRIXOK REGULÁRIS SZÉTVÁGÁSAI /2

A reguláris szétvágással definiált iterációkra érvényes a következő

TÉTEL. Tetszőleges erősen reguláris együtthatómátrixú $Ax = b$ egyenletrendszerhez tartozó $x^{(k+1)} = Bx^{(k)} + c$ iteráció

i, pontosan akkor konvergens, ha $\rho(B) < 1$,

ii, pontosan akkor konvergens, ha van olyan mátrixnorma, amelyre $||B|| < 1$.

TÉTEL. Ha megadható olyan $||.||$ mátrixnorma, amelyre $||\hat{L}|| + ||\hat{U}|| < 1$, akkor mind a Jacobi iteráció, mind a Gauss-Seidel iteráció konvergens.

BIZONYÍTÁS. Alkalmazzuk a háromszög-egyenlőtlenséget a Jacobi iteráció B mátrixára:

$$||B|| = ||\hat{L} + \hat{U}|| \leq ||\hat{L}|| + ||\hat{U}|| < 1.$$

Ebből és az előző tételből következik a Jacobi iteráció globális konvergenciája.

A Gauss-Seidel iteráció definíciójából kiindulva végezzük el a következő azonos átalakításokat:

$$\begin{aligned} x^{(k+1)} &= Bx^{(k)} + c \\ x^{(k+1)} &= (I - \hat{L})^{-1}\hat{U}x^{(k)} + c \\ (I - \hat{L})x^{(k+1)} &= \hat{U}x^{(k)} + (I - \hat{L})c \\ x^{(k+1)} &= \hat{L}x^{(k+1)} + \hat{U}x^{(k)} + (I - \hat{L})c. \end{aligned}$$

Mivel az x^* megoldás is kielégíti az utolsó egyenletet, ezért

$$x^* = \hat{L}x^* + \hat{U}x^* + (I - \hat{L})c,$$

és innen a képlethiba

$$e_{k+1} = \hat{L}e_{k+1} + \hat{U}e_k,$$

illetve átrendezés után

$$||e_{k+1}|| \leq \frac{||\hat{U}||}{1 - ||\hat{L}||}||e_k||.$$

A tétel kiinduló feltételéből következik, hogy az $||e_k||$ szorzója egynél kisebb, amiből adódik a Gauss-Seidel iteráció konvergenciája. □

KONVERGENCIA DIAGONÁLIS DOMINÁNS EGYÜTTHATÓMÁTRIXRA

Könnyen belátható, hogy minden diagonális domináns együtthatómátrix erősen reguláris.

A korábbiakban beláttuk, hogy az együtthatómátrix akkor és csak akkor diagonális domináns, ha $||B||_\infty < 1$ teljesül a Jacobi iteráció B mátrixára.

Ha az A együtthatómátrix diagonális domináns, akkor mind a Jacobi- mind a Gauss-Seidel iteráció konvergens.

KONJUGÁLT GRADIENS MÓDSZER

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

Legyen A egy szimmetrikus, pozitív definit mátrix, akkor a

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

q(x) = segéd függvény melynek egyetlen minimuma x* ezt a + definitsége biztosítja
A = n * n -es matrix
b = n komponensu vektor

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*: $-\nabla q(x) = b - Ax = r$.

Több változós függvény elso deriváltja adja a gradienst

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:

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

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

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

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

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

r˘k = reziduális vektro
s˘k = keresesi irany
alfa = lépésköz

KONJUGÁLT GRADIENS MÓDSZER / 2

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

1. $\alpha_k = (r_k^T r_k)/(s_k^T A s_k)$ (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 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 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:

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

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

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

Tehát a lineáris egyenletrendszer bal- és jobb oldalának eltérése már a számábrázolás pontossága 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.	prekondícionált 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 = solver(A,b),
```

ahol `solver` a táblázatban szereplő egyik eljárás neve. Ha a megállási feltételben a toleranciát módosítani szeretnénk, akkor a hívási forma

```
x = solver(A,b,tol),
```

ahol a `tol` érték az a szám, amellyel a `norm(b-A*x) <= tol*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
```

EGYENLETRENDSZEREK ITERÁCIÓS MÓDSZEREI / MATLAB / 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 prekondícionált 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 = A \ b` pedig hétszer tovább.

EGYENLETRENDSZEREK ITERÁCIÓS MÓDSZEREI / MATLAB / 3

Az iteratív eljárások a hatékony működéshez általában prekondícionálást 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 prekondícionáló mátrixok meghatározása nehéz feladat, és általában az adott alkalmazás ismeretét kívánja meg, amiből a lineáris egyenletrendszer származik.

Az általunk vizsgált A mátrixnak egy jó prekondícionálója az

```
M = diag(diag(A))
```

mátrix, az A mátrix főátlója elemeiből álló diagonális mátrix. Ezzel mint ötödik argumentummal felhívva a `pcg` eljárást, lényegesen 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.

A POLINOMOK ZÉRUSHELYEI

A $p(x) = a_0x^n+a_1x^{n-1}+\cdots+a_n$ alakú *polinom*oknak, ahol a_i az i-edik (általános esetben komplex) polinom-együttható, és a_0 nem nulla, fontos szerepük van a matematikában. A tudományos számításokban pedig bonyolultabb függvényeket szokás polinomokkal közelíteni. A polinomokat egyszerű deriválni és integrálni, nem nagyon nehéz a zérushelyeiket (gyökeiket) közelíteni. Numerikus nehézségek magasabb fokú polinomokkal kapcsolatban szoktak fellépni.

A polinomok kiértékelésére a *Horner-elrendezés* használatos:

$$p(x) = (\dots((a_0x + a_1)x + a_2)\,x + \cdots + a_{n-1})\,x + a_n.$$

Ennek a kiértékelési sémának a műveletigénye n szorzás és n összeadás, míg a szokásos polinomalak szerinti kifejtés ennél lényegesen több számítást igényelne.

Az x_0 komplex számot a $p(x)$ polinom *zérushely*ének, vagy *gyök*ének nevezzük, ha $p(x_0) = 0$. Ez akkor és csak akkor teljesül, ha igaz a

Olyan argumentum ahol a gyök értéke 0

$$p(x) = (x - x_0)q(x)$$

egyenlőség valamely $q(x)$ polinommal. Azt mondjuk, hogy az x_i gyök *multiplicitása* ν_i, ha $p(x)$ felírható $(x - x_i)^{\nu_i}r(x)$ alakban, ahol $r(x)$ olyan polinom, amelynek nem gyöke x_i. A gyökökre érvényesek a következő állítások:

ejsd mu"

i, Az n-edfokú $p(x)$ polinomnak multiplicitással számolva n darab komplex gyöke van.

ii, A $p(x)$ polinom a gyökök sorrendjétől eltekintve egyértelműen felírható $p(x) = a_0(x - x_1)(x - x_2)\cdots(x - x_n)$ *gyöktényezős alak*ban.

iii, Ha a $p(x)$ valós együtthatós polinomnak van egy x_i komplex gyöke, akkor annak komplex konjugáltja, $\overline{x}_i$ is gyöke, éspedig azonos multiplicitással.

iv, Nincs olyan megoldóképlet, amely tetszőleges polinom gyökeit meg tudná adni az együtthatókat tartalmazó, véges sok aritmetikai művelettel, és gyökvonással felírt kifejezések formájában.

v, Ha a $p(x)$ polinom gyökei $x_1, x_2, \dots, x_n$, akkor a $q(x) = p(-x)$ polinom gyökei $-x_1, -x_2, \dots, -x_n$.

vi, Az $r(x) = x^np(1/x)$ polinom gyökei $1/x_1, 1/x_2, \dots, 1/x_n$, feltéve hogy $p(0) \neq 0$.

RUFFINI-SOROZAT

A Horner-elrendezés kiszámításához hasonló a számítógépen könnyen implementálható, az $\hat{x}$ helyhez tartozó *Ruffini-sorozat*:

x kalap egy kiértékelési hely a ruffini sorozathoz szükségünk van az elz sorozat a-k együtt hatóira

$$b_0 = a_0,$$

$$b_k = b_{k-1}\hat{x} + a_k,$$

ahol $k = 1, 2, \ldots, n$. Erre érvényes a következő

SEGÉDTÉTEL. Jelölje $q(x) = b_0 x^{n-1} + b_1 x^{n-2} + \cdots + b_{n-2}x + b_{n-1}$ a $\{b_k\}$ sorozat elemeivel felírt $n-1$-edfokú polinomot. Ekkor

i, $p(x) = (x - \hat{x})q(x) + b_n$,

ii, $p(\hat{x}) = b_n$, és Ha x helyére behelyettesítjük x kalapot akkor adja magát az elso állításból

iii, $p'(\hat{x}) = q(\hat{x})$.

BIZONYÍTÁS. Az első állítás abból adódik, hogy a Ruffini-sorozat képletei a $p(x) : (x - \hat{x})$ polinomosztás végrehajtásából következnek. Ha most az i, egyenletbe $x = \hat{x}$-et helyettesítjük, akkor pontosan az ii, állítást kapjuk. Az i, egyenlet differenciálása és az előző helyettesítés pedig az iii, egyenletet adja. □

A Ruffini-sorozat fölhasználásával közvetlenül belátható, hogy tetszőleges n-edfokú $p(x)$ polinomra és $\hat{x}$ valós számra érvényes, hogy

- a $p(\hat{x})$ értéke $\mathcal{O}(n)$ művelettel, n darab szorzással és n összeadással meghatározható,
- az a kérdés, hogy $\hat{x}$ gyöke-e a $p(x)$ polinomnak, az $\mathcal{O}(n)$ művelettel eldönthető.

ITERÁLT HORNER-ELRENDEZÉS

Ha a polinom mellett annak deriváltjai helyettesítési értékeire is szükség van, akkor az *iterált Horner-elrendezés*t használhatjuk. Ennek az az alapja, hogy a Ruffini-sorozatból kapott $q(x)$ polinomra $p'(\hat{x}) = q(\hat{x})$.

Legyen most $p_0(x)$ az eredeti polinom, $p_1(x)$ az első Ruffini-sorozat együtthatóival felírt polinom. Vegyük most a p_0-hoz tartozó $q(x)$ polinomot (n együttható), majd az ehhez tartozó, a Ruffini-sorozat alapján meghatározott polinom legyen a következő polinom. Ezzel egy $n+1$ polinomból álló sorozatot határoztunk meg ($a_{0,j} = a_j$, $0 \le j \le n$):

$$a_{i,0} = a_{i-1,0},$$

$$a_{i,k} = a_{i,k-1}\hat{x} + a_{i-1,k},$$

ahol $1 \le i \le n+1$, és $1 \le k \le n-i+1$. Minden sorban tehát a kiszámolt együtthatókból egyet elhagyunk, és az így kapott polinomból lépünk tovább.

A kiszámolt együtthatókat táblázatba rendezhetjük:

polinom	együtthatók				
$p_0(x)$:	$a_{0,0}$	$a_{0,1}$	$\cdots$	$a_{0,n-1}$	$a_{0,n}$
$p_1(x)$:	$a_{1,0}$	$a_{1,1}$	$\cdots$	$a_{1,n-1}$	$a^*_{1,n}$
$p_2(x)$:	$a_{2,0}$	$a_{2,1}$	$\cdots$	$a^*_{2,n-1}$	
$\vdots$	$\vdots$	$\vdots$			
$p_n(x)$:	$a_{n,0}$	$a^*_{n,1}$			
$p_{n+1}(x)$:	$a^*_{n+1,0}$				

A *-al jelzett együtthatók már nem tartoznak az illető sorban lévő polinomhoz.

SEGÉDTÉTEL. A *-al megjelölt együtthatók az eredeti polinom $(x-\hat{x})$ hatványai szerint átrendezett alakját adják:

$$p_0(x) = a_{n+1,0}(x-\hat{x})^n + a_{n,1}(x-\hat{x})^{n-1} + \cdots + a_{2,n-1}(x-\hat{x}) + a_{1,n}.$$

BIZONYÍTÁS. A $p_0(x)$ polinom fokszáma szerinti indukcióval bizonyítunk. Ha a p_0 fokszáma legfeljebb 1, akkor az állítás nyilvánvalóan teljesül. Tegyük fel most, hogy az egyenlőség érvényes minden $n-1$-edfokú polinomra. Ha a $p_0(x)$-re fölírjuk a táblázat első két sorát, a Ruffini-sorozatokra vonatkozó előző segédtétel alapján $p_0(x) = (x-\hat{x})p_1(x) + a_{1,n}$. Az indukciós feltevés alapján ezután

$$p_1(x) = a_{n+1,0}(x-\hat{x})^{n-1} + a_{n,1}(x-\hat{x})^{n-2} + \cdots + a_{2,n-1}.$$

Ezt visszahelyettesítve $p_1(x)$ helyére az előző egyenletbe, épp a bizonyítandót kapjuk. □

ITERÁLT HORNER-ELRENDEZÉS / 2

Az iterált Horner-elrendezésre vonatkozó segédtételből adódik az alábbi

KÖVETKEZMÉNY. Tetszőleges $p(x)$ valós együtthatós n-edfokú polinomra és $\hat{x}$ valós számra érvényesek a következő állítások:

i, A $p(x)$ polinom $x-\hat{x}$ hatványai szerint átrendezett alakja $\mathcal{O}(n^2)$ művelettel megkapható.

ii, A $p(\hat{x}), p'(\hat{x}), \ldots, p^{(n)}(\hat{x})$ polinom- és deriváltértékek összesen $\mathcal{O}(n^2)$ művelettel számíthatók.

iii, A $p(x)$ polinom tetszőleges x_i gyökének multiplicitása $\mathcal{O}(n^2)$ művelettel meghatározható.

BIZONYÍTÁS. Az i, állítás igazolásához elegendő az előző táblázat egyes sorainak műveletigényét összegezni.

ii, Hasonlítsuk össze az előző Segédtételben megadott $p_0(x)$ polinomot a $p(x)$ polinom $\hat{x}$ körüli n-edfokú Taylor-polinomjával:

$$p(x) = \frac{p^{(n)}(\hat{x})}{n!}(x-\hat{x})^n + \frac{p^{(n-1)}(\hat{x})}{(n-1)!}(x-\hat{x})^{n-1} + \cdots + p'(\hat{x})(x-\hat{x}) + p(\hat{x}).$$

A Taylor-polinom unicitása miatt

$$\frac{p^{(n)}(\hat{x})}{n!} = a_{n+1,0}, \quad \frac{p^{(n-1)}(\hat{x})}{(n-1)!} = a_{n,1}, \ \ldots, \ p'(\hat{x}) = a_{2,n-1}, \quad p(\hat{x}) = a_{1,n}.$$

Innen az összes deriváltérték kifejezhető $\mathcal{O}(n)$ művelet árán.

iii, Egy x_i gyök multiplicitása meghatározásához elegendő az előbbi derivált-sorozat végén a nullákat $\mathcal{O}(n)$ további művelettel összeszámolni. □

A deriváltakra vonatkozó eredmények javíthatók, megmutatható, hogy az n darab derivált értékének kiszámításához elegendő $\mathcal{O}(n)$ művelet. Viszont nincs olyan eljárás, amely tetszőleges n-edfokú polinom helyettesítési értékét megadná $\mathcal{O}(n)$-nél kevesebb művelettel.

Ha tehát valamely polinomot nagyon sok helyen kell kiértékelni, például előfordul egy gyakran hívott szubrutinban, akkor a fentiek értelmében érdemes azt olyan alakban felírni, amelynek kisebb műveletigénye lényegesen csökkentheti a teljes futási időt. A kapott műveletigények érvényesek komplex együtthatós polinomok esetén azonos számú komplex műveletre.

A POLINOMOK A MATLABBAN

A polinomokat a Matlab a koefficienseiket indexeik szerint növekvő sorrendben tartalmazó sorvektorral reprezentálja. Eszerint tehát a vektor első komponense a polinom legmagasabb fokú tagjának együtthatója. Például az $2x^3+2x+3$ polinomot a következő Matlab utasítás adja meg (persze a nulla együtthatókat is be kell vinni):

```
>> p1 = [2 0 2 3]
p1 =
     2      0      2      3
```

Ahogy azt már korábban, a sajátértékeknek az adott mátrix karakterisztikus polinomjának ismeretében való meghatározásánál láttuk, a polinomok gyökeit a `roots(.)` utasítás adja meg oszlopvektor formájában:

```
>> r = roots(p1)
r =
     0.4306 + 1.2475i
     0.4306 - 1.2475i
    -0.8612
```

Ha fordítva, egy polinommal kapcsolatban csak a gyökeit ismerjük, és a polinom együtthatóit szeretnénk meghatározni, akkor a `poly` utasítást kell használnunk.

```
>>poly(r)
ans =
     1.0000     -0.0000      1.0000      1.5000
```

Vegyük észre, hogy mivel az azonos gyökökkel rendelkező minimális fokszámú polinomokból végtelen sok van, a Matlab ezek közül azt adta meg, amelyiknek a legmagasabb fokú tagjának együtthatója egy (és ennek megfelelően az eredmény eltér a kiindulási polinomtól).

A POLINOMOK GYÖKE A MATLABBAN

A gyököket meghatározó Matlab program olyan egyszerű és meglepő, hogy érdemes a program szövegét a magyarázó megjegyzések nagy része nélkül, a

`c:\MATLAB6p5\toolbox\matlab\polyfun\roots.m`

állománynak megfelelően bemutatni, még ha minden részlete nem is érthető az előzőek alapján:

```
function r = roots(c)
if size(c,1)>1 & size(c,2)>1
    error('Must be a vector.')
end
c = c(:).'; n = size(c,2); r = zeros(0,1);
inz = find(c); if isempty(inz),
    % All elements are zero
    return
end
% Strip leading zeros and throw away.
% Strip trailing zeros, but remember them as roots.
nnz = length(inz); c = c(inz(1):inz(nnz));
r = zeros(n-inz(nnz),1);
% Polynomial roots via a companion matrix
n = length(c); if n > 1
    a = diag(ones(1,n-2),-1);
    a(1,:)  = -c(2:n) ./ c(1);
    r = [r;eig(a)];
end
```

Itt `size(c,i)` a `c` mátrix `i`-edik dimenziójának a méretét adja, a `find` utasítás az argumentum nem nulla elemei indexét adja vissza, `length` az argumentum vektor hosszát. Az összeállított $(n-1)\times(n-1)$-es mátrix olyan, hogy a főátló alatti mellékátlóban egyesek vannak, az első sorban pedig a $-a_2/a_1, -a_3/a_1, \ldots,$ $-a_n/a_1$ elemek. Ezeken kívül a mátrix minden eleme nulla.

Érdekes a `c = c(:).'` utasítás: ez felsorolja a `c` vektor elemeit, ez oszlopvektort ad. Ezután veszi ennek (nem konjugált) transzponáltját.

A program a nyilvánvaló esetek elkülönítése után egy megfelelően összeállított mátrix sajátértékei meghatározására vezette vissza a gyökkeresési feladatot.

A POLINOMOKKAL VÉGZETT MÛVELETEK A MATLABBAN

A polinomok összeadására és kivonására a Matlab nem tartalmaz külön utasítást, ezekre a műveletekre egyszerűen a megfelelő sorvektorokra vonatkozó műveletek kell használni. Arra viszont a felhasználónak kell ügyelni, hogy a sorvektorok hossza megfelelő legyen:

```
>> p1 = [1 2 -3];
>> p2 = [-1 3 4];
>> p3 = p1+p2
p3 =
     0      5      1
>> p4 = [2 -1 3 4];
>> p5 = [0 p1]+p4
p5 =
     2      0      5      1
```

Ezt az eljárást azért viszonylag egyszerűen lehet automatizálni a Matlabban, ezt valósítja meg a következő egyszerű Matlab program. Figyeljük meg a részleteket!

```
function p1esp2 = addpoly(p1,p2)
if nargin < 2
   error('Keves argumentum.')
end
p1 = p1(:)'; % azt biztositja, hogy p1 sorvektor
p2 = p2(:)'; % azt biztositja, hogy p2 sorvektor
lp1 = length(p1); % a p1 hossza
lp2 = length(p2); % a p1 hossza
p1esp2 = [zeros(1,lp2-lp1) p1] + [zeros(1,lp1-lp2) p2];
```

Az előző feladatra alkalmazva új programunkat, ismét a helyes eredményt kapjuk:

```
>> addpoly(p1,p4)
ans =
     2      0      5      1
```

A POLINOMOKKAL VÉGZETT MÛVELETEK A MATLABBAN / 2

A polinomok szorzását a `conv` Matlab utasítás végzi:

```
>> p1 = [1 2 -3]
>> p2 = [-1 3 4]
>> conv(p1,p2)
ans =
     -1      1     13     -1    -12
```

az osztást pedig a `deconv` parancs, mégpedig maradékos osztásként, ha az alábbiak szerint két polinomba kérjük az eredményt:

```
>> [q,r] = deconv(p1,p2)
q =
     -1
r =
     0      5      1
```

A polinomok deriválását is külön Matlab utasítás végzi:

```
>> p = [2 -1 3 -4];
>> dp = polyder(p);
dp =
     6     -2      3
```

Vegyük észre, hogy ez a deriválás lényegében numerikus jellegű, tehát nem szimbolikus műveletekről van szó. A Matlab Symbolic Math Toolbox nevű csomagjára ehhez nem volt szükség, csak a polinomokra vonatkozó deriválási szabályokat alkalmaztuk az együtthatók konkrét értékeire.

A polinomok adatszerkezete miatt is szükség van külön kiértékelő szubrutinra, a `polyval` nevűre, amely a Horner-elrendezés alapján működik:

```
>> p = [1 -1 0 0];
>> polyval(p,0.5)
ans =
     -0.1250
```

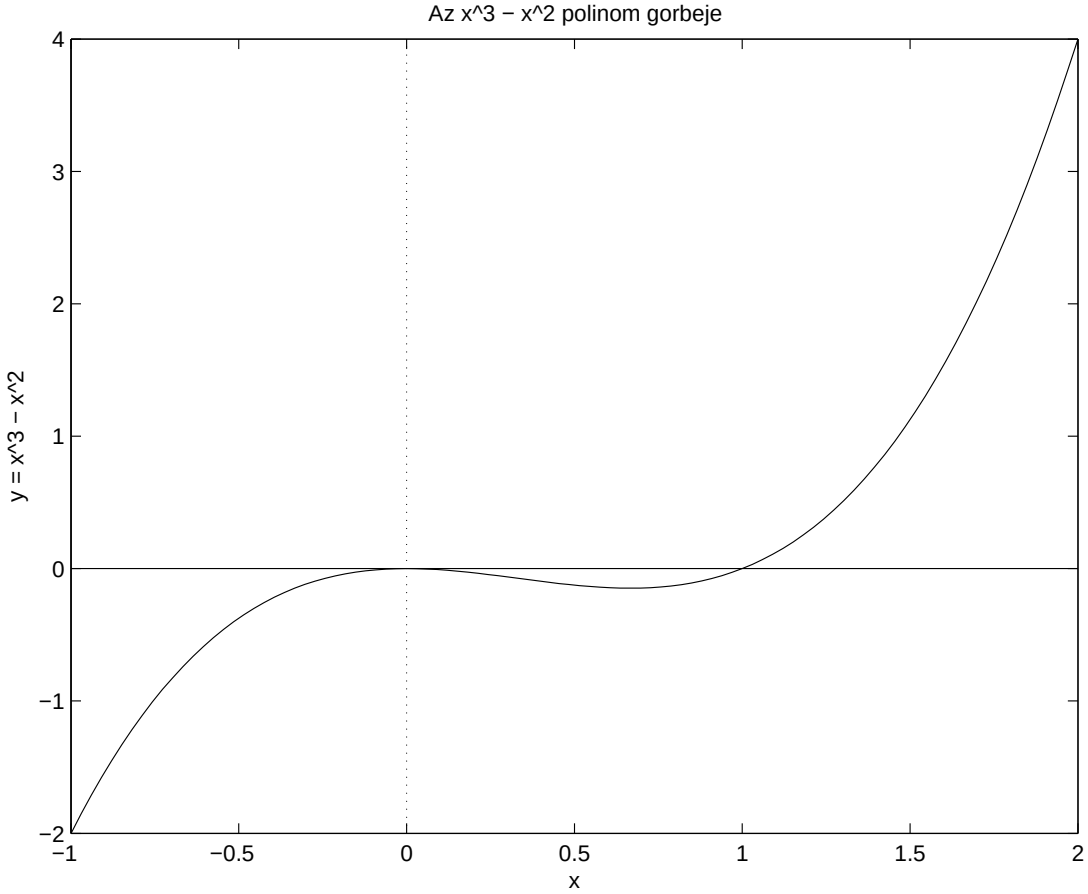
A POLINOMOKKAL VÉGZETT MÛVELETEK A MATLABBAN / 3

Ha a második argumentum mátrixot tartalmaz, akkor a kiértékelés ennek megfelelően történik.

A polinomok kiértékelése gyakran szükséges azok ábrázolása során. Figyeljük meg a 100 alappont kijelölésére használható ügyes `linspace` utasítást!

```
>> p = [1 -1 0 0];
>> x = linspace(-1,2);
>> y = linspace(-2,4);
>> px = polyval(p,x);
>> plot(x,px,x,x-x,0,y)
>> title('Az x^3 - x^2 polinom gorbeje')
>> xlabel('x')
>> ylabel('y = x^3 - x^2')
```

A kapott szép ábra:



FÜGGVÉNYKÖZELÍTÉSEK

Az előző fejezetben láttuk, hogy a polinomokkal való műveletek gyorsan és hatékonyan végrehajthatók számítógépen. Ez, és a polinomoknak több kedvező tulajdonsága (pl. differenciálhatóság) indít arra, hogy bonyolultabb függvények közelítésére polinomokat használjunk.

Azt a feladatot, amely azt tűzi ki, hogy adott $(x_i, y_i),\ i = 1, 2, \ldots, m$ pontsorozathoz állítsuk elő azt a függvényt, amely egy adott függvényosztályba tartozik, és minden ponton átmegy, *interpoláció*nak nevezzük. Az, hogy az illető függvény adott pontokon áthalad, azt jelenti, hogy az x_i argumentumokat véve a függvény a megfelelő y_i értékeket veszi fel. Ha a keresett $f(x)$ függvény polinom, akkor *polinominterpoláció*ról beszélünk. Ha a közelítő függvényeket a két polinom hányadosából álló racionális függvények körében keressük, akkor *racionális interpoláció*ról van szó.

Az interpoláció egy másik jelentése az, hogy a közelítő függvény segítségével az eredeti $f(x)$ függvény értékét egy olyan $\hat{x}$ pontban becsüljük az interpoláló $p(x)$ polinom $p(\hat{x})$ helyettesítési értékével, amelyre

$$\hat{x} \in [\min(x_1, x_2, \ldots, x_m), \max(x_1, x_2, \ldots, x_m)].$$

Ha ezzel szemben

$$\hat{x} \notin [\min(x_1, x_2, \ldots, x_m), \max(x_1, x_2, \ldots, x_m)]$$

teljesül, akkor *extrapoláció*ról van szó.

A *spline interpoláció* azt a feladatot jelöli, amelyik több alacsony fokszámú polinomból összerakott függvényt keres úgy, hogy az adott pontokon való áthaladás megkövetelése mellett az is elvárás, hogy a szomszédos polinomok a csatlakozási pontokban előírt differenciálhatósági feltételeknek is eleget tegyenek. Az alkalmazott polinomok fokszáma alapján beszélünk kvadratikus vagy köbös splineról.

A polinom interpoláció esetén a polinom fokszáma, n egyenlő $m-1$-el (tehát eggyel kevesebb, mint a pontok száma). Spline alkalmazásakor a fokszám lényegesen kisebb, mint az alappontok száma. Amennyiben egy olyan polinomot illesztünk, amelynek fokszáma kisebb, mint $m-1$, akkor *görbeillesztés*ről beszélünk. Ez a polinom persze nem feltétlen megy át minden alapponton.

LANGRANGE INTERPOLÁCIÓ

Abban az esetben, amikor az interpoláló alapfüggvények polinomok, a következő érvelés alkalmazható. Tegyük fel, hogy az alappontok páronként különbözők. Ez annál is jogosabb, mert azt követeljük meg, hogy az interpoláló polinom menjen át az adott ponton – márpedig adott x-re nem mehet át a polinom két $y = f(x)$ értékhez tartozó ponton.

Írjuk fel az interpolációs feltételeket a következő alakban:

$$\sum_{k=0}^{n} a_k x_i^k = y_i, \qquad i = 0, 1, 2, \ldots, n.$$

Ez egy lineáris egyenletrendszert ad meg a keresett a_k együtthatókra vonatkozóan, ami a *Vandermonde-mátrix*szal adható meg:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

Erre a mátrixra érvényes, hogy

$$\det \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} = \prod_{i>j} (x_i - x_j).$$

Innen az adódik, hogy amennyiben az alappontok páronként különbözők, akkor pontosan egy interpolációs polinom létezik, amely az adott pontokon áthalad.

A *Lagrange interpoláció* az interpoláló polinomokat

$$p_n(x) = \sum_{i=0}^{n} f(x_i) L_i(x)$$

alakban adja meg, ahol

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} = \frac{(x - x_0)(x - x_1) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_0)(x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)}.$$

LANGRANGE INTERPOLÁCIÓ / 2

TÉTEL. Legyenek adottak az $x_0, \ldots, x_n$ páronként különböző alappontok. Ekkor az $f(x_i),\ i = 0, 1, \ldots, n$ függvényértékekhez egyértelműen létezik olyan legfeljebb n-edfokú interpoláló polinom, amely megegyezik a Lagrange interpolációs polinommal.

BIZONYÍTÁS. A definíció alapján az L_i polinomok mindegyike n-edfokú polinom, amelyekre

$$L_i(x_j) = \begin{cases} 0 & \text{ha } i \neq j, \\ 1 & \text{ha } i = j. \end{cases}$$

Ebből adódik, hogy $p_n(x) = \sum_{i=0}^{n} f(x_i)L_i(x)$ egy legfeljebb n-edfokú polinom, amelyik kielégíti a $p_n(x_j) = f(x_j)$ feltételt.

Az unicitás bizonyításához feltesszük, hogy p egy legfeljebb n-edfokú polinom, amelyre $p(x_j) = f(x_j),\ j = 0, \ldots, n$. Ekkor a $p_n - p$ polinom foka is $\leq n$, és legalább $n+1$ darab páronként eltérő gyöke van: $x_0, \ldots, x_n$. Ebből az következik, hogy $p_n - p$ osztható az $(x - x_0)(x - x_1) \cdots (x - x_n)$ szorzattal. Mivel az osztó fokszáma $> n$, így ebből az adódik, hogy $p_n(x) - p(x)$ azonosan nulla, vagyis $p = p_n$. □

A Lagrange interpolációs polinom együtthatóinak kiszámítására szolgáló egy eljárás a Matlabban:

```
function [C,L] = lagran(X,Y)
w = length(X);
n = w-1;
L = zeros(w,w);
for k=1:n+1
    V = 1;
    for j=1:n+1
       if k ~= j
          V = conv(V,poly(X(j)))/(X(k)-X(j));
       end
    end
    L(k,:)  = V;
end
C = Y*L;
```

LAGRANGE INTERPOLÁCIÓ, PÉLDA

Tekintsük a következő adatsort: (0, 0), (1, -1), (2, 0), (3, 1) és (4, 0). Illesszünk erre interpolációs polinomot, és ez alapján határozzuk meg az 5 pontban az extrapolált értéket. A definíciót használva:

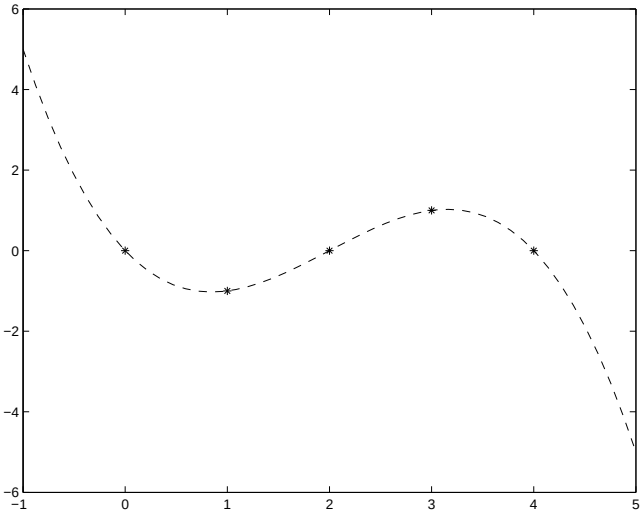
$$\begin{aligned} p(x) &= 0\frac{(x-1)(x-2)(x-3)(x-4)}{(0-1)(0-2)(0-3)(0-4)} - 1\frac{(x-0)(x-2)(x-3)(x-4)}{(1-0)(1-2)(1-3)(1-4)} + \\ & 0\frac{(x-0)(x-1)(x-3)(x-4)}{(2-0)(2-1)(2-3)(2-4)} + 1\frac{(x-0)(x-1)(x-2)(x-4)}{(3-0)(3-1)(3-2)(3-4)} + \\ & 0\frac{(x-0)(x-1)(x-2)(x-3)}{(4-0)(4-1)(4-2)(4-3)}. \end{aligned}$$

Innen egyszerűsítve:

$$\begin{aligned} p(x) &= -1\frac{(x-0)(x-2)(x-3)(x-4)}{(1-0)(1-2)(1-3)(1-4)} + 1\frac{(x-0)(x-1)(x-2)(x-4)}{(3-0)(3-1)(3-2)(3-4)} \\ &= \frac{(x)(x-2)(x-3)(x-4)}{6} - \frac{(x)(x-1)(x-2)(x-4)}{6} \\ &= \frac{1}{6}x(x-2)(x-4)\left((x-3)-(x-1)\right) = -\frac{1}{3}x(x-2)(x-4). \end{aligned}$$

A $p(5)$ érték ebből $-15/3 = -5$.

Az interpoláló polinomot az extrapolált függvényértékkel együtt láthatjuk a következő ábrán:



LAGRANGE INTERPOLÁCIÓ, PÉLDA / 2

Induljunk most ki az eredeti interpolációs feltételekből ($\sum_{k=0}^{n} a_k x_i^k = y_i, \quad i = 0, 1, 2, \ldots, n$):

$$\begin{aligned}
a_0 + a_1 0 + a_2 0^2 + a_3 0^3 + a_4 0^4 &= 0 \\
a_0 + a_1 1 + a_2 1^2 + a_3 1^3 + a_4 1^4 &= -1 \\
a_0 + a_1 2 + a_2 2^2 + a_3 2^3 + a_4 2^4 &= 0 \\
a_0 + a_1 3 + a_2 3^2 + a_3 3^3 + a_4 3^4 &= 1 \\
a_0 + a_1 4 + a_2 4^2 + a_3 4^3 + a_4 4^4 &= 0
\end{aligned}$$

Egyszerűsítések után (pl. $a_0 = 0$ adódik mindjárt az első egyenletből):

$$\begin{array}{lllll}
 & a_1 + & a_2 + & a_3 + & a_4 = -1 \\
2 & a_1 + 4 & a_2 + 8 & a_3 + 16 & a_4 = 0 \\
3 & a_1 + 9 & a_2 + 27 & a_3 + 81 & a_4 = 1 \\
4 & a_1 + 16 & a_2 + 64 & a_3 + 256 & a_4 = 0.
\end{array}$$

A Gauss-elimináció lépéseit követve

$$\begin{array}{llll}
a_1 + & a_2 + & a_3 + & a_4 = -1 \\
 & a_2 + 3 & a_3 + 7 & a_4 = 1 \\
 & + 6 \; a_2 + 24 & a_3 + 78 & a_4 = 4 \\
 & + 12 \; a_2 + 60 & a_3 + 252 & a_4 = 4,
\end{array}$$

majd

$$\begin{array}{llll}
a_1 + & a_2 + & a_3 + & a_4 = -1 \\
 & a_2 + 3 & a_3 + 7 & a_4 = 1 \\
 & + 6 & a_3 + 36 & a_4 = -2 \\
 & + 24 & a_3 + 168 & a_4 = -8,
\end{array}$$

és

$$\begin{array}{llll}
a_1 + & a_2 + & a_3 + & a_4 = -1 \\
 & a_2 + 3 & a_3 + 7 & a_4 = 1 \\
 & + & a_3 + 6 & a_4 = -1/3 \\
 & & + 24 & a_4 = 0.
\end{array}$$

Ebből már adódik az előző eljárással is megkapott $a_4 = 0$, $a_3 = -1/3$, $a_2 = 2$, $a_1 = -8/3$, és $a_0 = 0$, azaz az interpoláló polinom.

A LAGRANGE INTERPOLÁCIÓ HIBÁJA

Az interpolációs feladat megfogalmazásában eddig nem kellett semmilyen kikötést tennünk az interpolált $f(x)$ függvényre. A hibavizsgálathoz viszont feltesszük, hogy az $f(x)$ függvény $n+1$-szer differenciálható azon a legszűkebb $[a,b]$ intervallumon, amelyben benne van az $n+1$ alappont, és az $\hat{x}$ interpolációs vagy extrapolációs pont is.

Vezessük be a

$$g(x) = f(x) - p_n(x) - \frac{\omega_n(x)}{\omega_n(\hat{x})}(f(\hat{x}) - p_n(\hat{x}))$$

jelölést, ahol $p_n(x)$ a Lagrange-féle interpolációs polinom, $\omega_n(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$. Ez a $g(x)$ függvény nyilván legalább $n+1$-szer differenciálható az $[a,b]$ intervallumon, és az $n+1$-edik deriváltja:

$$g^{(n+1)}(x) = f^{(n+1)}(x) - \frac{(n+1)!}{\omega_n(\hat{x})}(f(\hat{x}) - p_n(\hat{x})),$$

mivel a legfeljebb n-edfokú $p_n(x)$ polinom deriváltja nulla, az egy főegyütthatós $\omega_n(x)$-é pedig $(n+1)!$.

A definíció szerint $g(x)$ minden alappontban nulla, és $g(\hat{x}) = 0$ is teljesül, tehát $g(x)$-nek $[a,b]$-ben legalább $n+2$ különböző zérushelye van. A kalkulusban megismert Rolle tétel szerint tetszőleges két zérushely között van olyan pont, ahol a $g'(x)$ derivált is felveszi a nulla értéket. Emiatt az $[a,b]$ intervallumban $g'(x)$-nek legalább $n+1$ zérushelye van. A gondolatmenetet megismételve az adódik, hogy akkor $g''(x)$-nek $[a,b]$-ben legalább n zérushelye van stb. Így belátható, hogy van olyan $\xi \in [a,b]$, hogy

$$g^{(n+1)}(\xi) = f^{(n+1)}(\xi) - \frac{(n+1)!}{\omega_n(\hat{x})}(f(\hat{x}) - p_n(\hat{x})) = 0,$$

és innen a képlethibára azt kapjuk, hogy

$$f(\hat{x}) - p_n(\hat{x}) = \frac{\omega_n(\hat{x})}{(n+1)!} f^{(n+1)}(\xi).$$

Amennyiben $f^{(n+1)}(x)$ korlátos az $[a,b]$-n, és felső korlátja μ_{n+1}, akkor a teljes $[a,b]$ intervallumon érvényes a következő hibabecslés:

$$|f(\hat{x}) - p_n(\hat{x})| \le \frac{|\omega_n(\hat{x})|}{(n+1)!}\mu_{n+1} \le \frac{\max_{x\in[a,b]}|\omega_n(x)|}{(n+1)!}\mu_{n+1}.$$

Az utóbbi becslések élesek abban az értelemben, hogy az $f(x)$ függvény és az alappontok megválasztásával elérhető az egyenlőség mindkét esetben.

INTERPOLÁCIÓ A MATLABBAN

A Matlab egy- két- és magasabb dimenziós interpolációt támogat. A legegyszerűbb esetben az

```
y0 = interp1(x,y,x0)
```

utasítás az `x(i), y(i)` pontokra vonatkozó interpolációs polinomot meghatározza, kiértékeli azt az `x0` pontban, és az eredményt értékül adja az `y0` változónak. Az `x(i)` értékeknek szigorúan növekvő sorozatot kell alkotniuk. A negyedik argumentumban megadhatjuk az interpoláció típusát aposztrófok között:

`'cubic'`	köbös interpoláció,
`'linear'`	lineáris interpoláció (alapértelmezés),
`'nearest'`	a legközelebbi szomszédos interpoláció,
`'spline'`	köbös spline interpoláció.

A lineáris interpoláció egyenes szakaszokból álló törtvonalat hoz létre, a legközelebbi szomszéd az adott ponthoz legkisebb távolságú alappont függvényértékét adja.

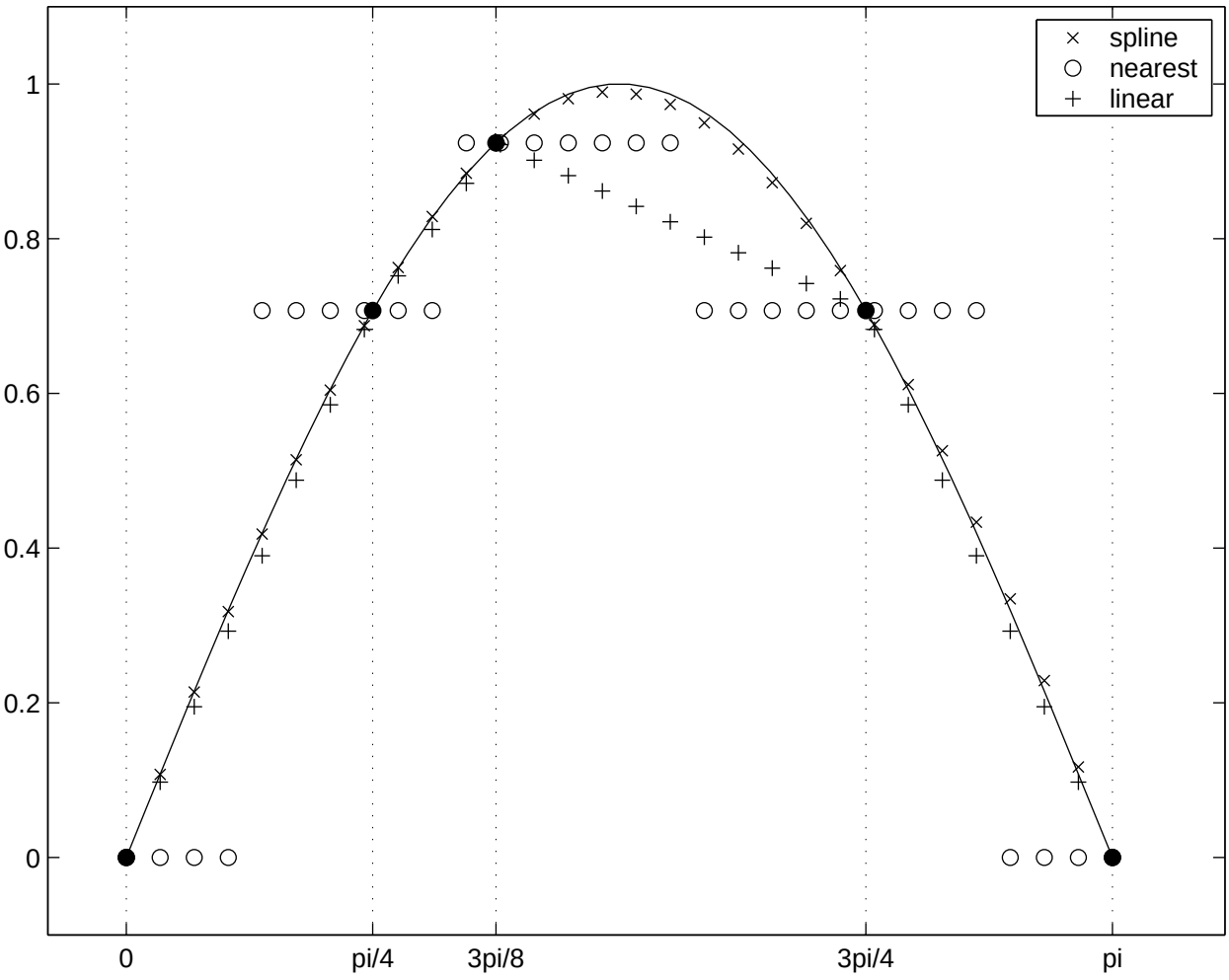
A következő Matlab példa a szinusz függvényt közelíti a $[0, \pi]$ intervallumban, 5 alappontra támaszkodva. A módszerekből a köbös interpoláció hiányzik, mert az ábrán nem lenne megkülönböztethető a spline interpolációtól. Az interpolációs polinomokat 30 pontban értékeljük ki, és jelenítjük meg. Az ábrán a teli körök jelzik az adatpontokat, és az egyéb jelek az interpolált értékeket.

A nearest interpoláció ennek alig nevezhető, durva közelítést ad. Ezzel szemben a spline jó leírást ad a függvényről, és a lineáris spline közelítés is csak a nagyon görbült részen tér el lényegesen.

INTERPOLÁCIÓ A MATLABBAN / 2

```
>> x = [0 pi/4 3*pi/8 3*pi/4 pi]; y = sin(x);
>> xi = linspace(0,pi,30);
>> ys = interp1(x,y,xi,'spline');
>> yn = interp1(x,y,xi,'nearest');
>> yl = interp1(x,y,xi,'linear');
>> xx = linspace(0,pi,50);
>> plot(xx,sin(xx),'-',x,y,'.','MarkerSize',20), hold on
>> set (gca,'XTick',x), set (gca,'XGrid','on')
>> set (gca,'XTickLabel','0|pi/4|3pi/8|3pi/4|pi')
>> h = plot(xi,ys,'x',xi,yn,'o',xi,yl,'+');
>> axis([-0.25 3.5 -0.1 1.1])
>> legend(h,'spline','nearest','linear'), hold off
```

Magyarázzuk meg az egyes közelítő függvények közti eltéréseket!



TÖBBDIMENZIÓS INTERPOLÁCIÓ A MATLABBAN

A Matlabban két eljárás is van kétdimenziós interpolációra: a `griddata` és az `interp2`. Az előbbi utasítás szintaxisa

```
ZI = griddata(x,y,z,XI,YI),
```

ahol az `x`, `y` és `z` vektorok tartalmazzák az adatot, `ZI` az interpolált függvényértékek mátrixa az `XI` és `YI` mátrixoknak megfelelően — amelyeket rendszerint a `meshgrid` paranccsal töltünk ki. A hatodik, szöveges argumentum adja meg az alkalmazott közelítő módszert:

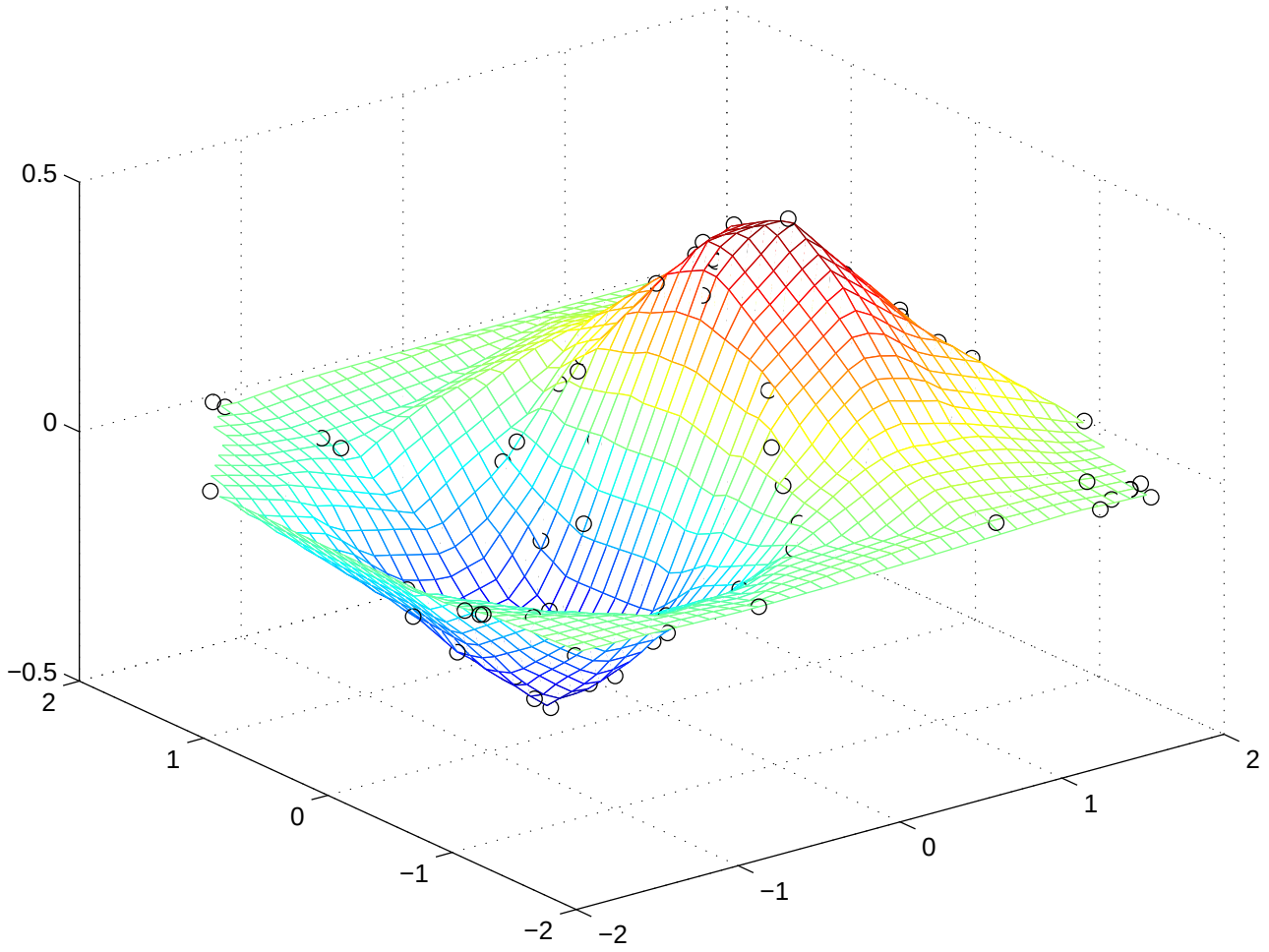
`'linear'`	háromszög alapú lineáris interpoláció (alapértelmezés),
`'cubic'`	háromszög alapú köbös interpoláció,
`'nearest'`	a legközelebbi szomszédos interpoláció.

Az `interp2`-nek hasonló argumentum-listája van, de azt igényli, hogy `x` és `y` monoton mátrixok legyenek, amilyent a `meshgrid` is előállít.

```
>> x = rand(100,1)*4-2; y = rand(100,1)*4-2;
>> z = x.*exp(-x.^2-y.^2);
>> hi = -2:.1:2;
>> [XI,YI] = meshgrid(hi);
>> ZI = griddata(x,y,z,XI,YI);
>> mesh(XI,YI,ZI), hold
>> plot3(x,y,z,'o'), hold off
```

TÖBBDIMENZIÓS INTERPOLÁCIÓ A MATLABBAN / 2

Az alábbi ábrán az eredeti adatpontokat körök jelzik, az interpolált felületet pedig a rács.



Az `interp3` és `interpn` eljárások három-, illetve n-dimenziós interpolációra használhatók.

A NEWTON-MÓDSZER

Tegyük fel, hogy az $f(x) = 0$ egyenlet x^* egyszeres, izolált zérushelyét akarjuk meghatározni, és hogy ennek egy környezetében $f(x)$ differenciálható. Válasszunk ki ebből a környezetből egy x_0 kezdőértéket, majd képezzük az

az x egyszeres zérushelye f -nek, ha f (x) != 0
az x izolált zérushelye f -nek, ha f (x) = 0 és van olyan epszilon > 0, hogy
az (x −epszilon , x +epszilon) intervallumban
x az egyetlen zérushely

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

iterációs sorozatot. Ezt az eljárást *érintő*-, vagy *Newton-módszer*nek nevezzük. Az iterációs képlet geometriai értelmezése az, hogy az aktuális x_k pontban meghatározzuk az $f(x)$ függvény és deriváltja értékét, ezekkel képezzük az adott ponthoz húzott érintőt, és következő iterációs pontnak azt határozzuk meg, amelyben az érintőnek zérushelye van.

SEGÉDTÉTEL. Ha az $f(x) = 0$ egyenlet x^* egyszeres zérushelyének egy környezetében $f(x)$ kétszer differenciálható, akkor megadható olyan $0 < \delta$, delta hogy ha $x_k \neq x^*$ benne van az x^* körüli δ sugarú $S_\delta(x^*)$ környezetben (intervallumban), akkor létezik az $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$ közelítés, és érvényes az

$$|e_{k+1}| = \left|\frac{f''(\eta_k)}{2f'(x_k)}\right| e_k^2$$

hibabecslés valamely olyan η_k-ra, amely benne van az x_k és x^* nyitott burkában.

BIZONYÍTÁS. Fejtsük $f(x)$-et x_k körül Taylor-sorba:

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{f''(\eta_k)}{2!}(x - x_k)^2.$$

Helyettesítsük most be az x^* értéket:

$$0 = f(x^*) = f(x_k) + f'(x_k)(x^* - x_k) + \frac{f''(\eta_k)}{2!}(x^* - x_k)^2.$$

Az iterációs képletből

$$f(x_k) = f'(x_k)(x_k - x_{k+1}).$$

Az utolsó két egyenletből pedig az adódik, hogy

$$0 = f'(x_k)(x^* - x_{k+1}) + \frac{f''(\eta_k)}{2!}(x^* - x_k)^2.$$

Ez az összefüggés már közvetlenül a segédtétel állítását igazolja. □

A NEWTON-MÓDSZER / 2

Az előző segédtétel alapján bizonyítható, hogy ha az $f(x)$ függvény kétszer folytonosan differenciálható az x^* zérushely egy környezetében, akkor van olyan pont, ahonnan indulva a Newton-módszer kvadratikusan konvergens sorozatot ad meg:

$$|x^* - x_{k+1}| \leq C|x^* - x_k|^2$$

valamely pozitív C konstanssal.

Ha az eljárást az $f(x)$ függvény deriváltjára alkalmazzuk, akkor optimalizálási módszert kapunk. Ennek az algoritmusnak van többváltozós változata is. Ott a deriválttal való osztás helyett a második parciális deriváltakat tartalmazó Hesse mátrix inverzével kell szorozni:

$$x_{k+1} = x_k - H^{-1}(x_k)\nabla f(x_k).$$

PÉLDA. A zérushely keresésre való Newton-módszer sebességének bemutatására tekintsük az $f(x) = x^2 - x$ függvényt. Ennek két zérushelye van, a nulla és az egy. Válasszuk indulópontnak az $x_0 = 2$-t. Az $f(x)$ függvény deriváltja $f'(x) = 2x - 1$. Ezzel az iterációs egyenlet:

$$x_{k+1} = x_k - \frac{x_k^2 - x_k}{2x_k - 1}.$$

Az iterációs sorozatot az alábbi táblázat tartalmazza:

iteráció	közelítés
0	2.00000000000000
1	1.33333333333333
2	1.06666666666667
3	1.00392156862745
4	1.00001525902190
5	1.00000000023283
6	1

Figyeljük meg, hogy az eredményen szépen látszik, ahogy a pontos jegyek száma minden lépésben közel megduplázódik, és az is, hogy az utolsó iterált már megkülönböztethetetlen a pontos megoldástól.

A SZELŐMÓDSZER

Legyen x^* az $f(x) = 0$ egyenlet egyszeres gyöke. Válasszunk alkalmas x_0 és x_1 kezdőértékeket, és ezekből indulva hajtsuk végre azt az iterációt, amit a következő képlet definiál:

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} = \frac{f(x_k)x_{k-1} - f(x_{k-1})x_k}{f(x_k) - f(x_{k-1})} \quad k = 1, 2, \ldots .$$

Könnyen belátható, hogy x_{k+1} nem más, mint az $(x_k, f(x_k))$ és az $(x_{k-1}, f(x_{k-1}))$ pontokon átmenő egyenes és az x tengely metszéspontja x koordinátája. Az iterációs képlet első alakja azt is megmutatja, hogy itt a Newton-módszer olyan módosításáról van szó, amikor $f'(x_k)$ helyett annak közelítéseként a numerikus deriválť,

$$\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

szerepel. Ez alapján a *szelőmódszer* tehát olyan iterációs eljárás, amely az $f(x) = 0$ egyenlet megoldásához csak az $f(x)$ függvényt kiszámító szubrutinra támaszkodik.

Amennyiben az $f(x)$ függvény kétszer folytonosan differenciálható az x^* gyök egy $S_\delta(x^*)$ környezetében, akkor vannak olyan indulóértékek, amelyekkel a szelőmódszer az x^* megoldáshoz konvergál, és a közelítés hibájára érvényes, hogy

$$|e_k| \leq \frac{2\mu_1}{\mu_2}|e_{k-1}|\ |e_{k-2}|,$$

ahol $\mu_1 \leq |f'(x)|$ és $f''(x) \leq \mu_2$ az $S_\delta(x^*)$ környezetbeli minden pontra.

A szelőmódszert szokás olyan kezdőértékekkel indítani, amelyek közrefogják a keresett x^* gyököt. Másrészt ha $f'(x^*) > 0$, és $f''(x^*) > 0$, akkor x^*-nál nagyobb (de ahhoz közeli) kezdőértékekkel szigorúan monoton konvergencia érhető el.

Azt az algoritmus-változatot, amely felteszi, hogy a kezdeti x_0, x_1 pontokban az $f(x)$ függvény ellentétes előjelű, és $f(x_{k+1})$ előjele függvényében a megelőző két pontból azt választja a következő iterációs lépéshez, amelyikkel ez a tulajdonság fennmarad, *húrmódszer*nek hívjuk.

A SZELŐMÓDSZER – PÉLDA

Tekintsük ismét a Newton-módszerrel már vizsgált feladatot: $f(x) = x^2 - x$. Két kezdőpont párral indítsuk az eljárást: 2 és $1.\dot{3}$ legyen az egyik, és 1.5, 0.5 a másik.

A Matlabbal kapott eredményeket a következő táblázat tartalmazza.

iteráció	1. közelítés	2. közelítés	húrmódszer
0	2.00000000000000	1.50000000000000	1.50000000000000
1	1.33333333333333	0.50000000000000	0.50000000000000
2	1.14285714285714	0.75000000000000	0.75000000000000
3	1.03225806451613	1.50000000000000	0.90000000000000
4	1.00392156862745	0.90000000000000	0.96428571428571
5	1.00012208521548	0.96428571428571	0.98780487804878
6	1.00000047683739	1.00413223140496	0.99590163934426
7	1.00000000005821	0.99984760743676	0.99863013698630
8	1	0.99999937277492	0.99954296160878
9		1.00000000009560	0.99984760743676
10		1	0.99994919731762
11			0.99998306519898
12			0.99999435500260
13			0.99999811832712
14			0.99999937277492
15			0.99999979092489
16			0.99999993030829
17			0.99999997676943
18			0.99999999225648
19			0.99999999741883
20			0.99999999913961

A pontos jegyek számának duplázódását később ugyan, de meg lehet figyelni a szelőmódszer esetén is. Mivel a Newton-módszer értékesebb információt is igényel, ezért természetes, hogy annak konvergenciája gyorsabb volt. A húrmódszer lassabb konvergenciája az egyik alappont változatlanul maradása miatt adódott.

LEGKISEBB NÉGYZETEK MÓDSZERE

Vizsgáljuk meg most azt az esetet, amikor több feltételünk van, mint együtthető ezek kielégítéséhez. Általában persze ilyen feladat nem oldható meg minden pontban való illeszkedéssel, de feltehető az a kérdés, hogy mi az a megoldás, amely a lehető legkisebb eltérést ad a kitűzött egyenlet két oldala között.

Tekintsünk például olyan $Ax = b$ egyenletet, amelyben a b vektor magasabb dimenziójú, mint az x. Gyakori eset az, amikor az adatvektort egy véletlen modell generálja, azaz $y = Ax + \epsilon$, ahol ϵ a zajt reprezentáló vektor, ismert statisztikai jellemzőkkel. Amennyiben az ϵ komponensei független valószínűségi változók nulla várható értékkel és konstans varianciával, a Gauss-Markov tétel állítása szerint a legjobb torzítatlan lineáris becslést az x vektorra úgy kaphatjuk meg, ha a reziduál $||y - Ax||_2$ kettes normáját minimalizáljuk.

Mivel ebben az esetben az

$$||y - Ax||_2^2 = ||\epsilon||_2^2 = \sum_i \epsilon^2$$

is minimális, így az x vektor meghatározásának ezt a módját a *legkisebb négyzetek módszeré*nek, az ennek megfelelő x vektort pedig *legkisebb négyzetes megoldás*nak nevezzük. A megoldás előállítása módját adja meg a következő

TÉTEL. Tegyük fel, hogy $A \in \mathbb{C}^{m\times n}$, $b \in \mathbb{C}^m$, és $A^H A$ nem szinguláris. Ekkor $||b - Ax||_2$ a minimumát abban az x^* vektorban veszi fel, amelyikre az

$$A^H A x^* = A^H b$$

normálegyenlet teljesül.

BIZONYÍTÁS. Legyen $r = b - Ax$, és $r^* = b - Ax^*$. Ekkor a normálegyenletből az adódik, hogy $A^H r^* = 0$, és emiatt

$$(r^* - r)^H r^* = (A(x - x^*))^H r^* = (x - x^*)^H A^H r^* = 0.$$

Ebből az következik, hogy

$$\begin{aligned} ||r||_2^2 &= r^H r + r^{*H}(r^* - r) = r^{*H} r^* - (r^* - r)^H r \\ &= r^{*H} r^* + (r^* - r)^H (r^* - r) = ||r^*||_2^2 + ||r^* - r||_2^2. \end{aligned}$$

Azaz $||r||_2^2 \geq ||r^*||_2^2$, és az egyenlőség pontosan akkor teljesül, ha $r^* = r$. Ebben az esetben azonban

$$A^H A(x - x^*) = A^H (r - r^*) = 0,$$

és az $A^H A$ regularitása miatt a minimum csak az $x^* = x$ pontban áll elő. □

LEGKISEBB NÉGYZETEK MÓDSZERE / 2

A normálegyenlet együtthatómátrixa, $A^H A$ Hermite-szimmetrikus, mivel

$$A^H A = A^H (A^H)^H = (A^H A)^H.$$

Ezen túl pozitív szemidefinit is, mivel $x^H A^H A x = ||Ax||_2^2 \geq 0$ teljesül minden x-re. Ha az A rangja n, akkor $A^H A$ pozitív definit, mert $x^H A^H A x = 0$-ból $Ax = 0$ következik, és így $x = 0$.

Amennyiben a normálegyenlet együtthatómátrixa jól kondícionált (mint pl. a spline közelítések esetén), akkor ebből közvetlenül meghatározható a legkisebb négyzetes megoldás. Ennek az a módja, hogy képezzük a Cholesky-felbontást: $A^H A = LL^H$, majd a megfelelő háromszögmátrixokkal megoldjuk a kapott egyenleteket.

Gyakran azonban a kapott normálegyenletek rosszabban kondícionáltak, mint a kiindulási optimalizálási feladat, a kondíciószám lényegében a négyzete lesz a korábbinak. Görbeillesztési feladatokban a közelítő függvényeket szokás alapfüggvények lineáris kombinációjaként megadni. Amikor ezek az alapfüggvények hasonló jellegűek, akkor az ezekből adódó egyenletek rendszerint rosszul kondícionáltak.

LEGKISEBB NÉGYZETEK MÓDSZERE / PÉLDA

PÉLDA. Határozzuk meg tízes alapú, 5 hosszú mantisszával és optimális kerekítéssel a legjobban illeszkedő egyenest a (3.32, 4.32), (3.33, 4.33) és (3.34, 4.34) pontokhoz. A megoldás nyilvánvalóan az egy meredekségű, és egy tengelymetszetű egyenes. A feladat az $s = x_1 t + x_2$ egyenlet együtthatóinak meghatározása, annak az $Ax = b$ egyenletnek a megoldása, ahol

$$A = \begin{pmatrix} 3.32 & 1 \\ 3.33 & 1 \\ 3.34 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 4.32 \\ 4.33 \\ 4.34 \end{pmatrix}.$$

Az $A^H Ax = A^H b$ normálegyenletre azt kapjuk, hogy

$$A^H A \approx \begin{pmatrix} 33.267 & 9.99 \\ 9.99 & 3 \end{pmatrix}, \quad A^H b \approx \begin{pmatrix} 43.257 \\ 12.99 \end{pmatrix}.$$

A Cholesky felbontás az $A^H A \approx LL^T$ eredményt adja, ahol:

$$L \approx \begin{pmatrix} 5.7678 & 0 \\ 1.7320 & 0.014142 \end{pmatrix},$$

az első egyenletrendszerből azt kapjuk, hogy

$$L^{-1} A^H b \approx \begin{pmatrix} 7.4997 \\ 0.0010 \end{pmatrix},$$

és ebből a hamis megoldás:

$$x = \left(L^H\right)^{-1} L^{-1} A^H b \approx \begin{pmatrix} 1.2790 \\ 0.070817 \end{pmatrix}.$$

A probléma oka az $A^H A$ mátrix rosszul kondícionáltsága: $\mathrm{cond}_\infty(A^H A) \approx 3.1\ 10^6$ mellett nem lehetett pontos értékes jegyet várni. Ha a számításokat a Matlab alapbeállításával hajtottuk végre, akkor már 11 jegyre helyes eredményt kaptunk: $x = 1.00000000000349, 0.99999999998837$. Ez megint csak összhangban van a fenti kondíciószámmal.

Jegyezzük meg, hogy jobban jártunk volna, ha a megoldást $s = x_1(t - 3.33) + x_2$ alakban keressük. Különben pedig a korábban megállapítottak értelmében rosszul kondícionált mátrixra érdemesebb a QR felbontást használni.

LEGKISEBB NÉGYZETEK MÓDSZERE A MATLABBAN

Polinomok legkisebb négyzetek módszere alapján való illesztésével a Matlabban a `polyfit` utasítás foglalkozik. Feltesszük, hogy az interpolációs alappontok páronként különbözők, és hogy egy n-edfokú $p(x)$ polinomot szeretnénk megkeresni az $\{x_i, y_i\}_{i=1}^m$ pontokhoz úgy, hogy $p(x_i) \approx y_i$ teljesüljön. Konkrétan az utasítás olyan polinomot határoz meg, amelyre a

$$\sum_{i=1}^{m} (p(x_i) - y_i)^2$$

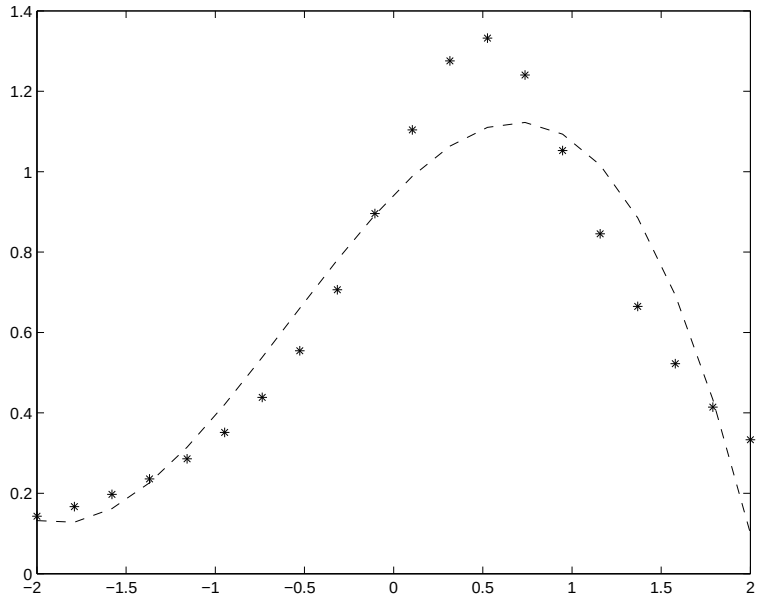
függvény minimális. A szintaxisa

`p = polyfit(x,y,n).`

Ha az `n` paramétert úgy adjuk meg, hogy $n \geq m - 1$, akkor interpolációs polinomot kapunk. Másrészt magas fokszámú interpolációs polinomok nagyon oszcillálók lehetnek, ezért általában alacsony fokszámú polinommal való közelítést szoktak előnyben részesíteni.

A következő ábra az $1/(x + (1 - x)^2)$ függvénynek a [-2, 2] intervallumban 20 egyenletesen elhelyezkedő mintapontjához illeszt köbös polinomot.

```
>> x = linspace(-2,2,20);
>> y = 1./(x+(1-x).^2);
>> p = polyfit(x,y,3);
>> plot(x,y,'*',x,polyval(p,x),'--')
```



SPLINE PÉLDA

Vegyük a korábban interpolációval vizsgált adatok első felét: $(0,0)$, $(1,-1)$, $(2,0)$. Illesszünk ehhez kvadratikus spline-t a csatlakozási pontban megegyező első derivált értékkel:

$$p_1(0)=0, \quad p_1(1)=-1, \quad p_2(1)=-1, \quad p_2(2)=0, \quad p_1'(1)=p_2'(1).$$

Ez kifejtve a $p_1(x)=a_0+a_1x+a_2x^2$, és a $p_2(x)=b_0+b_1x+b_2x^2$ polinomokra:

$$\begin{aligned} a_0+a_1 0+a_2 0^2 &= 0\\ a_0+a_1 1+a_2 1^2 &= -1\\ b_0+b_1 1+b_2 1^2 &= -1\\ b_0+b_1 2+b_2 2^2 &= 0\\ a_1 1+2a_2 1 &= b_1+2b_2 1. \end{aligned}$$

Innen azonnal adódik, hogy $a_0=0$. Mivel egyel kevesebb egyenlet van, mint meghatározandó együttható, ezért valamely további megkötést tehetünk. Legyen ez most az, hogy $p_1'(0)=2a_2 0+a_1=a_1=0$. Ekkor a maradék feltételünk:

$$\begin{aligned} a_1 &= 0\\ a_1+a_2 &= -1\\ b_0+b_1+b_2 &= -1\\ b_0+2b_1+4b_2 &= 0\\ a_1+2a_2 &= b_1+2b_2. \end{aligned}$$

Innen most gyorsan meghatározhatók az első polinom együtthatói: $a_0=0$, $a_1=0$, $a_2=-1$. Ezután

$$\begin{aligned} b_0+b_1+b_2 &= -1\\ b_0+2b_1+4b_2 &= 0\\ b_1+2b_2 &= -2. \end{aligned}$$

Innen Gauss eliminációval

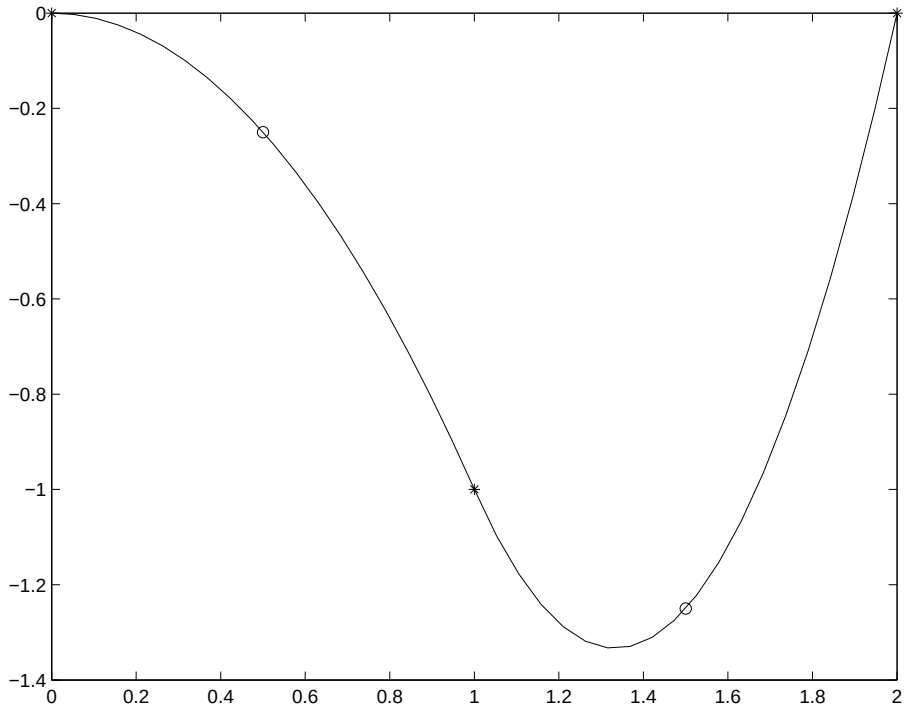
$$\begin{aligned} b_0+b_1+b_2 &= -1\\ b_1+3b_2 &= 1\\ -b_2 &= -3, \end{aligned}$$

valamint $b_2=3$, $b_1=-8$, és $b_0=4$ adódik.

SPLINE PÉLDA / 2

Határozzuk meg a 0.5 és az 1.5 pontokban a közelítés értékét. Mivel 0.5 a [0, 1] intervallumban van, ezért az interpolált értékhez a p_1 polinomot használjuk: $f(0.5) \approx p_1(0.5) = -1(0.5)^2 = -0.25$. Az 1.5 pontra pedig $f(1.5) \approx p_2(1.5) = 4 - 8(1.5) + 3(1.5)^2 = -1.25$.

Ábrázoljuk a spline közelítést a korábbiakhoz hasonlóan. A grafikonon csillag jelzi az adatpontokat, és karika az interpolációs pontokat.



Az ábrát előállító Matlab program (a rövid sorok kedvéért kicsit terjengősen írva):

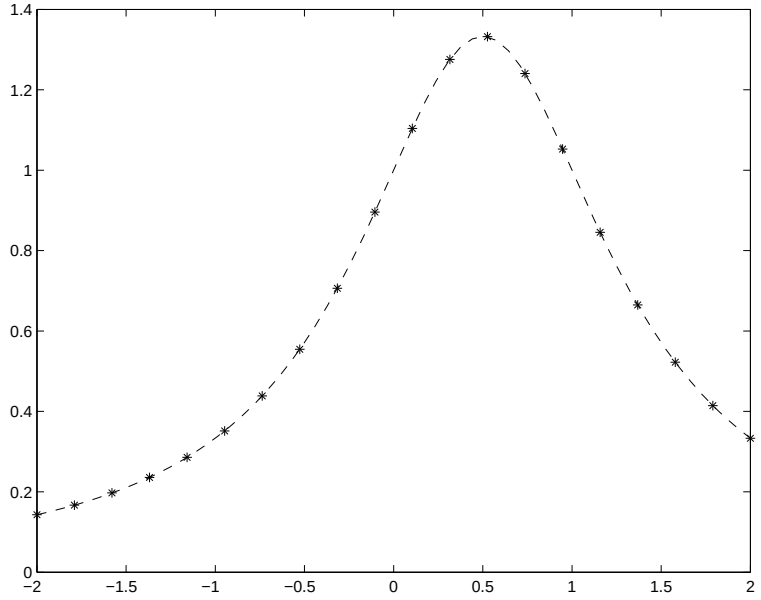
```
>> xm = [0 1 2];
>> ym = [0 -1 0];
>> x1 = linspace(0,1,20);
>> x2 = linspace(1,2,20);
>> p1 = [-1 0 0];
>> p2 = [3 -8 4];
>> xo = [0.5, 1.5];
>> yo = [-0.25, -1.25];
>> pa = polyval(p1,x1);
>> pb = polyval(p2,x2);
>> plot(xm,ym,'*',x1,pa,'-',x2,pb,'-',xo,yo,'o')
```

SPLINE A MATLABBAN

A korábbi példát oldjuk most meg köbös spline-okkal:

```
>> x = linspace(-2,2,20);
>> y = 1./(x+(1-x).^2);
>> xx = linspace(-2,2,60)
>> yy = spline(x,y,xx);
>> plot(x,y,'*',xx,yy,'--')
```

A kapott ábra érthető módon (mivel sokszor annyi paraméteres közelítőfüggvényt használtunk) lényegesen jobb approximációt ad:



Lehetőség van a spline függvény együtthatóival való közvetlen számolásra is. A `ppval` utasítást lehet használni az együtthatók interpretálására. A

```
>> pp = spline(x,y);
>> plot(x,y,'*',xx,ppval(pp,xx),'--')
```

utasítások is az előbbi ábrát eredményezik. A spline-ok alacsony szintű manipulálására szolgálnak az `mkpp` (a közelítőfüggvény összeállítása) és `unmkpp` (a spline-ok paraméterei kivonása) parancsok. A `pp` ezekben a nevekben a piecewise polynomial-ra utal.

NUMERIKUS INTEGRÁLÁS, KVADRATÚRA-FORMULÁK

A kvadratúra a numerikus integrálás szinonimája, amikor az

$$\int_a^b f(x)\,dx = F(b) - F(a)$$

határozott integrál közelítése a feladat. Itt $F(x)$ az $f(x)$ integrálandó függvény primitív függvénye. Ez utóbbi nem minden esetben áll rendelkezésre, sőt sokszor (mint például az $f(x) = e^{x^2}$ esetben) nem is elemi függvény, nem adható meg zárt alakban.

A határozott integrálokat szokás

$$\int_a^b f(x)\,dx \approx Q_n(f) = \sum_{i=1}^{n} w_i f(x_i)$$

alakban közelíteni, ahol $Q_n(f)$-et *kvadratúra-formulának* nevezzük. Általában feltesszük, hogy $x_i \in [a,b]$ teljesül az x_i *alappontok*ra, és ezek páronként különbözők. A w_i számokat *súly*oknak hívjuk. ejsd vevé

Az $\int_a^b f(x)\,dx$ integrál és a $Q_n(f)$ kvadratúra-formula homogén és additív leképezés, azaz érvényesek

$$\int_a^b f(x)+g(x)\,dx = \int_a^b f(x)\,dx + \int_a^b g(x)\,dx,$$

$$Q_n(f+g) = \sum_{i=1}^{n} w_i(f(x_i)+g(x_i)) = \sum_{i=1}^{n} w_i f(x_i) + \sum_{i=1}^{n} w_i g(x_i) = Q_n(f) + Q_n(g),$$

$$\int_a^b \alpha f(x)\,dx = \alpha \int_a^b f(x)\,dx,$$

$$Q_n(\alpha f) = \sum_{i=1}^{n} w_i \alpha f(x_i) = \alpha \sum_{i=1}^{n} w_i f(x_i) = \alpha Q_n(f).$$

Az integrál fontos további tulajdonsága a határok szerinti additivitás:

$$\int_a^b f(x)\,dx = \int_a^{z_1} f(x)\,dx + \cdots + \int_{z_{m-1}}^{z_m} f(x)\,dx + \int_{z_m}^b f(x)\,dx,$$

és az, hogy ha az $f(x)$ folytonos függvény nem azonosan nulla, és jeltartó az $[a,b]$ intervallumon, akkor $\int_a^b f(x)\,dx \neq 0$.

NUMERIKUS INTEGRÁLÁS, KVADRATÚRA-FORMULÁK / 2

Érvényesek továbbá a következő állítások is az integrálokra:

1. Legyen $f(x)$ és $g(x)$ két, az $[a,b]$ intervallumon integrálható függvény. Ha $g(x)$ jeltartó $[a,b]$-n, akkor

$$\int_a^b f(x)g(x)\,dx = \mu \int_a^b g(x)\,dx, \text{ ahol } \inf_{x\in[a,b]} f(x) \le \mu \le \sup_{x\in[a,b]} f(x).$$

2. Ha még azt is tudjuk, hogy $f(x)$-nek megvan a Darboux-tulajdonsága (például mert folytonos) $[a,b]$-n, akkor érvényes

$$\int_a^b f(x)g(x)\,dx = f(\gamma)\int_a^b g(x)\,dx$$

valamely $\gamma \in [a,b]$-re.

A *kvadratúra-formula képlethibáját* az $R_n(f) = \int_a^b f(x)\,dx - Q_n(f)$ kifejezéssel definiáljuk. Akkor mondjuk, hogy egy *kvadratúra-formula pontos* $f(x)$-re, ha $R_n(f) = 0$. Amennyiben az $f(x)$ függvény, vagy olyan tulajdonságai, mint a deriválhatóság, illetve a deriváltak korlátai ismertek, a kvadratúra-formula hibája jól becsülhető. Sokszor viszont az $f(x)$ függvénynek csak az értékei ismertek bizonyos helyeken. A kvadratúra-formulák pontosságának jellemzésére azoknak a polinomokon való képlethibáját szokás vizsgálni.

Azt mondjuk, hogy a $Q_n(f)$ kvadratúra-formula *(pontossági) rendje* az r természetes szám, ha az pontos az $1, x, x^2, \ldots, x^r$ hatványfüggvényekre (azaz $R_n(x^k) = 0$ minden $0 \le k \le r$-re), de nem pontos x^{r+1}-re.

A rend meghatározása ekvivalens a

$$\begin{aligned} w_1 + w_2 + \cdots + w_n &= \int_a^b dx, \\ w_1x_1 + w_2x_2 + \cdots + w_nx_n &= \int_a^b x\,dx, \\ &\vdots \\ w_1x_1^r + w_2x_2^r + \cdots + w_nx_n^r &= \int_a^b x^r\,dx \end{aligned}$$

egyenletek megoldásával. Ha az alappontokat és a súlyokat ismeretlennek tekintjük, akkor ez egy $r+1$ egyenletből álló algebrai-, de nem lineáris egyenletrendszer $2n$ darab ismeretlennel. Ha viszont rögzítjük az alappontokat, akkor a súlyokra már lineáris egyenletrendszert kapunk.

NUMERIKUS INTEGRÁLÁS, KVADRATÚRA-FORMULÁK / 3

TÉTEL. A Q_n n alappontos kvadratúra-formula rendje legfeljebb $2n-1$ lehet.

BIZONYÍTÁS. Tekintsük a $q(x) = (\omega_{n-1}(x))^2$ $2n$-edfokú polinomot, ahol $\omega_{n-1}(x)$ az n alappontra felírt $(x-x_1)(x-x_2)\cdots(x-x_n)$ polinom. Erre

$$0 < \int_a^b q(x)\,dx = \int_a^b (\omega_{n-1}(x))^2\,dx \neq Q_n(q) = \sum_{i=1}^n w_i(\omega_{n-1}(x_i))^2 = 0,$$

tehát erre a polinomra nem pontos a formula. □

Amennyiben legalább 0 rendű formulát akarunk előállítani, akkor annak az $f(x)=1$ függvényre pontosnak kell lennie, azaz a

$$w_1 + w_2 + \cdots + w_n = \int_a^b dx = b - a$$

feltétel adódik a súlyokra. A továbbiakban ezt mindig feltételezzük.

Nagyszámú alappont esetén gondot jelenthet a kvadratúra-formulák öröklött hibája. Ha a pontos $f(x_i)$ függvényértékek helyett az $f^*(x_i)$ közelítő értékeket használjuk, akkor a velük adódó $Q_n^*(f) = \sum_{i=1}^n w_i f^*(x_i)$ öröklött hibája:

$$\begin{aligned} |Q_n(f) - Q_n^*(f)| &= \left|\sum_{i=1}^n w_i f(x_i) - \sum_{i=1}^n w_i f^*(x_i)\right| = \left|\sum_{i=1}^n w_i (f(x_i) - f^*(x_i))\right| \\ &\leq \sum_{i=1}^n |w_i|\,|(f(x_i) - f^*(x_i))| \leq \epsilon \sum_{i=1}^n |w_i|. \end{aligned}$$

Itt $\epsilon = \max_{1\leq i\leq n} |(f(x_i) - f^*(x_i))|$ a függvények abszolút hibakorlátja. Mivel Q_n pontossági rendje legalább 0, és $\sum_{i=1}^n w_i = b-a$, ezért

$$\epsilon(b-a) \leq \epsilon \sum_{i=1}^n |w_i|,$$

és az egyenlőség pontosan akkor teljesül, ha minden w_i súly pozitív. Az ilyen tulajdonságú kvadratúra-formulákat *pozitív kvadratúra-formulák*nak hívjuk. Az öröklött hiba abszolút hibakorlátja tehát ilyen formulákra minimális. Ezek a korlátok élesek abban az értelemben, hogy bizonyos esetekben a tényleges öröklött hiba megegyezik a megadottakkal.

INTERPOLÁCIÓS KVADRATÚRA-FORMULÁK

Azt mondjuk, hogy $Q_n(f) = \sum_{i=1}^n w_i f(x_i)$ egy *interpolációs kvadratúra-formula*, ha ez előáll az alappontokra felírt Lagrange polinom integrálásával:

$$\int_a^b f(x)\,dx \approx \int_a^b p_{n-1}(x)\,dx = \int_a^b \sum_{i=1}^n f(x_i)L_i(x)\,dx = \sum_{i=1}^n f(x_i)\int_a^b L_i(x)\,dx,$$

ahonnan $w_i = \int_a^b L_i(x)\,dx$. Ebben az esetben az alappont az interpolációra és a kvadratúrára is vonatkozik.

TÉTEL. Minden n alappontra épülő Q_n interpolációs kvadratúra-formula rendje legalább $n-1$.

BIZONYÍTÁS. A Lagrange interpolációs polinom unicitása miatt minden legfeljebb $n-1$-edfokú $p(x)$ polinom megegyezik a hozzá tartozó $p_{n-1}(x)$ interpolációs polinommal. Emiatt

$$\int_a^b p(x)\,dx = \int_a^b p_{n-1}(x)\,dx = Q_n(f),$$

tehát a kvadratúra-formula pontos $p(x)$-re, vagyis a rendje legalább $n-1$. □

TÉTEL. Ha egy Q_n kvadratúra-formula rendje legalább $n-1$, akkor az interpolációs kvadratúra-formula.

BIZONYÍTÁS. Legyen $p_{n-1}(x) = \sum_{i=1}^n L_i(x)f(x_i)$ az $f(x)$ függvénynek a $Q_n(f)$ alappontjaira vonatkozó Lagrange interpolációs polinomja. Mivel Q_n rendje legalább $n-1$, ezért az pontos az $n-1$-edfokú $L_i(x)$ polinomokra. Ekkor

$$\int_a^b p_{n-1}(x)\,dx = \sum_{i=1}^n \int_a^b L_i(x)\,dx f(x_i) = \sum_{i=1}^n Q_n(L_i)f(x_i) =$$

$$\sum_{i=1}^n \sum_{j=1}^n w_j L_i(x_j)f(x_i) = \sum_{i=1}^n w_i f(x_i) = Q_n(f),$$

mivel $L_i(x_j) = \delta_{ij}$. □

E két tétel szerint tehát minden legalább $n-1$-edrendű kvadratúra-formula előáll interpolációs kvadratúra-formulaként.

INTERPOLÁCIÓS KVADRATÚRA-FORMULÁK HIBÁJA

Tegyük fel, hogy az $f(x)$ függvény n-szer folytonosan differenciálható az $[a,b]$ intervallumon. Ez ahhoz kell, hogy a Lagrange interpolációs polinomokra korábban igazolt hibakorlátokat alkalmazni tudjuk. Ekkor emiatt

$$f(x) - p_{n-1}(x) = \frac{\omega_{n-1}(x)}{n!} f^{(n)}(\xi),$$

ahonnan

$$\int_a^b f(x)\ dx - \int_a^b p_{n-1}(x)\ dx = \int_a^b \frac{\omega_{n-1}(x)}{n!} f^{(n)}(\xi)\ dx,$$

amiből a képlethiba:

$$R_n(f) = \int_a^b f(x)\ dx - Q_n(f) = \int_a^b \frac{\omega_{n-1}(x)}{n!} f^{(n)}(\xi)\ dx.$$

Ha $\omega_{n-1}(x)$ jeltartó az $[a,b]$ intervallumon, akkor bár ξ függ x-től, a megfelelő középértéktétellel elérhető a következő alak ($\gamma \in [a,b]$):

$$R_n(f) = f^{(n)}(\gamma) \int_a^b \frac{\omega_{n-1}(x)}{n!}\ dx.$$

f nek az n edik deriváltja a gamma helyen

Ha $\omega_{n-1}(x)$ nem jeltartó az $[a,b]$ intervallumon, akkor

$$|R_n(f)| = \left| \int_a^b \frac{\omega_{n-1}(x)}{n!} f^{(n)}(\xi)\ dx \right| \le \int_a^b \frac{|\omega_{n-1}(x)|}{n!} |f^{(n)}(\xi)|\ dx$$

$$\le \frac{||f^{(n)}(x)||_\infty}{n!} \int_a^b |\omega_{n-1}(x)|\ dx,$$

ahol $||f^{(n)}(x)||_\infty$ az $|f^{(n)}(x)|$ maximuma $[a,b]$-n. Ez a hibakorlát akkor a legkisebb, ha $\int_a^b |\omega_{n-1}(x)|\ dx$ minimális:

ÁLLÍTÁS. Az $\int_{-1}^1 |\omega_{n-1}(x)|\ dx$ integrál értéke pontosan akkor minimális, ha alappontoknak az

$$U_0(x) = 1, \quad U_1(x) = 2x, \quad \dots \quad U_n(x) = 2xU_{n-1}(x) - U_{n-2}(x)$$

rekurzióval definiált n-edik másodfajú Csebisev-polinom zérushelyeit vesszük.

VÉGES DIFFERENCIÁK

Számos gyakorlati feladat megoldása során szokás egyenlő lépésközzel, ekvidisztáns alappontokkal közelítést adni. Ekkor tehát a szomszédos alappontok távolsága állandó: $h = x_{i+1} - x_i$. Az interpolációs alappontok tehát $x_i = x_0 + ih,\ i = 0, \ldots, n-1$.

Az adott x_k alappontokhoz és $f_k = f(x_k)$ függvényértékekhez tartozó $\Delta^i f_k$ *i-edrendű véges differenciák*at a következő kettős rekurzióval definálják:

$$\begin{aligned} \Delta^0 f_k &= f_k, \\ \Delta^i f_k &= \Delta^{i-1} f_{k+1} - \Delta^{i-1} f_k. \end{aligned}$$

A véges differenciákat célszerű az i szerint sorba rendezve meghatározni, ismételt kivonásokkal.

A természetes számokra értelmezett binomiális együtthatók általánosításaként vezessük be a

$$\binom{t}{j} = \frac{t(t-1)\cdots(t-j+1)}{j!}$$

jelölést a $t = (x - x_0)/h$ transzformációhoz.

A véges differenciákkal felírt Lagrange interpolációs polinom:

$$p_{n-1}(x_0 + th) = f_0 + \binom{t}{1}\Delta f_0 + \binom{t}{2}\Delta^2 f_0 + \cdots + \binom{t}{n-1}\Delta^{n-1} f_0 = \sum_{i=0}^{n-1}\binom{t}{i}\Delta^i f_0.$$

A $\binom{t}{j} = \frac{t-j+1}{j}\binom{t}{j-1}$ rekurzióval a $p_n(x)$ helyettesítési értéke az $x_0 + ht$ helyen $\mathcal{O}(n)$ művelettel meghatározható, ha a $\Delta^i f_0$ véges differenciák ismertek.

VÉGES DIFFERENCIÁK, PÉLDA

Az $f(x) = x^2 - x$ függvénynek a Lagrange interpoláció véges differenciákkal való felírására tekintsük a 0, 1 és 2 pontokra támaszkodó interpolációs polinomját. Ekkor $x_0 = 0$, $x_1 = 1$, $x_2 = 2$; valamint $f_0 = 0$, $f_1 = 0$ és $f_2 = 2$. Ebből a véges differenciák:

$$\begin{array}{lll} \Delta^0 f_0 = f_0 = 0, & \Delta^1 f_0 = \Delta^0 f_1 - \Delta^0 f_0 = 0, & \Delta^2 f_0 = \Delta^1 f_1 - \Delta^1 f_0 = 2 \\ \Delta^0 f_1 = f_1 = 0, & \Delta^1 f_1 = \Delta^0 f_2 - \Delta^0 f_1 = 2 & \\ \Delta^0 f_2 = f_2 = 2, & & \end{array}$$

A h lépésköz ez esetben 1, a $t = (x - x_0)/h = (x - 0)/1 = x$ új változóra felírva az interpolációs polinom:

$$p_2(x_0 + th) = f_0 + \binom{t}{1}\Delta f_0 + \binom{t}{2}\Delta^2 f_0 = 0 + \binom{t}{1}0 + \binom{t}{2}2 = 2\binom{t}{2}.$$

Tekintsük most a $\binom{t}{2}$ előállítását. A $\binom{t}{j} = \frac{t-j+1}{j}\binom{t}{j-1}$ rekurzióval:

$$\begin{aligned} \binom{t}{1} &= t \\ \binom{t}{2} &= \frac{t-2+1}{2}\binom{t}{1} = \frac{t-2+1}{2}t = \frac{t^2-t}{2}. \end{aligned}$$

Innen az interpolációs polinom: $x^2 - x$. Mivel 3 alappontra illesztettük az interpolációs polinomot, így érthető, hogy visszakaptuk a kiindulási függvényt.

NEWTON-COTES FORMULÁK

Az interpolációs kvadratúra-formulák egy régi osztályát adják a *Newton-Cotes formulák*, amelyek annak a speciális esetnek felelnek meg, amikor ekvidisztáns alappontokat használunk. Legyen a kiválasztott n alappont $a \leq x_0 < x_1 < \cdots < x_{n-1} \leq b$. Ha az integrálás határai szerepelnek az alappontok között, akkor *zárt-*, ha a határok nem alappontok, akkor *nyitott formulá*ról beszélünk.

A zárt formulákra tehát:

$$h = \frac{b-a}{n-1}, \quad a = x_0, \quad b = x_{n-1}, \quad \text{és} \quad x_i = x_0 + ih \quad 0 \leq i \leq n-1,$$

míg a nyitott formulákra:

$$h = \frac{b-a}{n+1}, \quad a = x_0 - h, \quad b = x_{n-1} + h, \quad \text{és} \quad x_i = x_0 + ih \quad 0 \leq i \leq n-1.$$

Az n-edik Newton-Cotes formulát a következő egyenlet definiálja a $t = (x - x_0)/h$ új változóval kifejezve:

$$\int_a^b p_{n-1}(x_0 + th)\, dx = \int_a^b \sum_{i=0}^{n-1} \binom{t}{i} \Delta^i f_0\, dx = \sum_{i=0}^{n-1} \Delta^i f_0 \int_a^b \binom{t}{i}\, dx.$$

A t szerinti integrálásra már két eset van. Ha a formula zárt:

$$\sum_{i=0}^{n-1} \Delta^i f_0 \int_a^b \binom{t}{i}\, dx = h \sum_{i=0}^{n-1} \Delta^i f_0 \int_0^{n-1} \binom{t}{i}\, dt,$$

ha pedig nyitott, akkor

$$\sum_{i=0}^{n-1} \Delta^i f_0 \int_a^b \binom{t}{i}\, dx = h \sum_{i=0}^{n-1} \Delta^i f_0 \int_{-1}^{n} \binom{t}{i}\, dt.$$

Adott i indexre az utolsó integrál könnyen számítható.

NEWTON-COTES FORMULÁK / 2

Az első négy zárt Newton-Cotes formula:

$\int_{x_0}^{x_1} f(x)\,dx \approx \frac{h}{2}(f_0 + f_1)$	trapéz szabály
$\int_{x_0}^{x_2} f(x)\,dx \approx \frac{h}{3}(f_0 + 4f_1 + f_2)$	Simpson-szabály
$\int_{x_0}^{x_3} f(x)\,dx \approx \frac{3h}{8}(f_0 + 3f_1 + 3f_2 + f_3)$	Simpson 3/8-os szabálya
$\int_{x_0}^{x_4} f(x)\,dx \approx \frac{2h}{45}(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4)$	Bool-szabály

Amennyiben a megfelelő deriváltak folytonosak $[a, b]$-n, akkor a fenti szabályok hibája rendre:

$$\frac{h^3}{12}f^{(2)}(c), \qquad \frac{h^5}{90}f^{(4)}(c), \qquad \frac{3h^5}{80}f^{(4)}(c), \qquad \frac{8h^7}{945}f^{(6)}(c),$$

ahol $c \in [a, b]$.

Az említett szabályok pontossági rendje sorrendben: 2, 3, 3 és 5.

NUMERIKUS INTEGRÁLÁS A MATLABBAN

A Matlabnak alapvetően két függvénye van a numerikus integrálásra, a `quad` és a `quadl`. Mindkettő megköveteli, hogy a $\int_a^b f(x)dx$ határozott integrál alsó és felső végpontja véges, és hogy az integrálandó függvénynek nincs szingularitása az $[a, b]$ intervallumon. Amennyiben ezek a feltételek nem teljesülnek, akkor a feladatot alkalmasan át kell alakítani, pl. a részenként való integrálással.

Az eljárásokat `q = quad(fun,a,b,tol)` alakban kell hívni (`quadl`-re hasonlóan). Itt a `fun` egy olyan integrálandó függvény, amely az argumentumok egy vektorát veszi inputként, és a függvényértékek egy vektorát adja vissza függvényértékként. A `tol` toleranciaérték abszolút toleranciát jelent, amelynek alapértelmezése kicsit nagyobb, mint az `eps` szorozva az integrál becsült értékével.

Tekintsük a következő egyszerű példát a használatra:

```
function f = fxlog(x)
f = x.*log(x);
```

legyen az $x \log(x)$ függvényünket kiszámító eljárás. A `quad` alkalmazása:

```
>> quad(@fxlog,2,4)
ans =
     6.7041
```

Figyeljük meg a függvény felírásában az x változó vektorként való kezelését. A skaláris függvényhívások számát a második output paraméterben kaphatjuk meg:

```
[q,count] = quad(fun,a,b)]
```

A `quad` eljárás a Simpson szabályon alapul, ami egy 3 pontra támaszkodó Newton-Cotes kvadratúra szabály (tehát pontos legfeljebb harmadfokú polinomokra), míg `quadl` a Gauss-Lobatto szabályt használja a 7-pontos Kronrod kiterjesztéssel (ötöd, illetve kilencedfokú polinomokra pontos változatokkal). Mindkét eljárás adaptív kvadratúrát alkalmaz, a teljes integrálási tartományt a feladat jellegének megfelelően felosztják részintervallumokra: ott a legsűrűbb a beosztás, ahol az integrálandó függvény a leggyorsabban változik. Hibaüzenetet kapunk, ha a részintervallumok túl kicsik, vagy ha túl sok függvényhívásra van szükség. Mindkettő az integrálandó függvény szingularitására utal.

NUMERIKUS INTEGRÁLÁS A MATLABBAN / 2

Az adaptív numerikus integrálás illusztrálására tekintsük a Matlab `humps` függvényét, számítsuk ki ennek a következő integrálját:

$$\int_0^1 \left(\frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.9)^2+0.04} - 6 \right) \, dx.$$

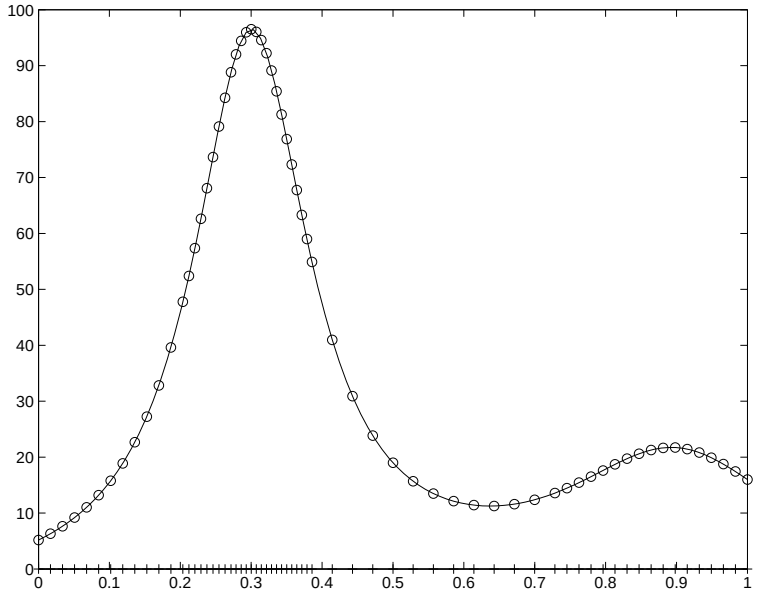
Adjuk meg a függvény `.m` kódját, beépítve az ábrában a kiértékelési pontok (karika), és azok x argumentumának (vessző) megjelenítését:

```
function [y] = h(x)
y = 1 ./ ((x-.3).^2 + .01) + 1 ./ ((x-.9).^2 + .04) - 6;
plot(x,0,'+'),plot(x,y,'o')
```

Mentsük el a definiált függvényt `h.m` néven. Az ábrát három Matlab sorral generálhatjuk (v.ö. `quaddemo.m`):

```
>> x = linspace(0,1,100);
>> plot(x,h(x),'-'), hold on
>> quad(@h,0,1,0.0001), hold off
```

Az alábbi ábráról szépen leolvasható, hogy az adaptív numerikus integrálás a görbe olyan szakaszán, ahol az érintő gyorsan változott, sok függvénykiértékelést igényelt.



Az integrál értéke pedig 29.8581.

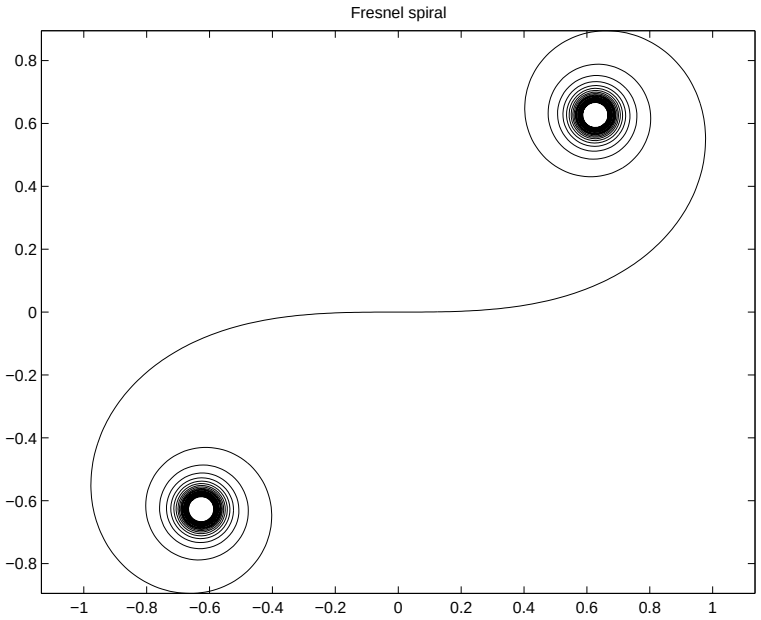
NUMERIKUS INTEGRÁLÁS A MATLABBAN / 3

Tekintsünk egy bonyolultabb integrálási példát, a Fresnel integrálokat:

$$x(t) = \int_0^t \cos(u^2)\, du, \quad y(t) = \int_0^t \sin(u^2)\, du.$$

Az ezekkel definiált görbe látható az alábbi ábrán. A t értékére a $[-4\pi, 4\pi]$ intervallumban 2001 egyenlő távolságra lévő pontban értékeltük ki a függvényeket.

```
>> n = 1000;
>> x = zeros(1,n); y = x;
>> t = linspace(0,4*pi,n+1);
>> for i=1:n
x(i) = quadl(inline('cos(x.^2)'),t(i),t(i+1),1e-3);
y(i) = quadl(inline('sin(x.^2)'),t(i),t(i+1),1e-3);
end
>> x = cumsum(x); y = cumsum(y);
>> plot([-x(end:-1:1) 0 x], [-y(end:-1:1) 0 y])
>> axis equal
>> Title('Fresnel spiral')
```



Figyeljük meg, hogy a hatékonyság kedvéért (így is másodpercekig számolt a program) kihasználtuk a szimmetriát, és a részintervallumokon való integrálás után a teljes összeget képeztük a `cumsum` utasítással.

NUMERIKUS INTEGRÁLÁS A MATLABBAN / 4

Egy további egyszerű numerikus integrálásra való utasítás a `trapz`. Ez az ismételt trapéz-szabályt valósítja meg. Abban is különbözik a `quad` és a `quadl` eljárásoktól, hogy nem egy függvényt kell használatához megadni, hanem az x_i és $f(x_i)$ értékeket tartalmazó vektorokat. Emiatt aztán nem is lehet adaptív az algoritmus. Példa a használatára:

```
>> x = linspace(0,2*pi,10);
>> f =sin(x).^2./sqrt(1+cos(x).^2);
>> trapz(x,f)
ans =
     2.8478
```

Ebben a példában a kiszámított integrál hibája 10^{-7} nagyságrendű, ami lényegesen jobb, mint amit a trapéz-szabály hibabecslése alapján várnánk. Ennek az a magyarázata, hogy periodikus függvényt integrálunk a teljes perióduson, és ebben pontos az ismételt trapéz-szabály. Mégis, amennyiben az integrálandó függvény képlete rendelkezésre áll, akkor érdemesebb a `quad` vagy a `quadl` eljárásokat használni.

A kettős integrálokat a `dblquad` utasítással lehet kiszámolni. Tekintsük a következő példát:

$$\int_4^6 \int_0^1 (y^2 e^x + x \cos y)\, dx\, dy.$$

Írjuk fel a függvényt:

```
function out = fxy(x,y)
      out = y^2*exp(x)+x*cos(y);
```

ezután kiadhatjuk a megfelelő utasítást:

```
>> dblquad(@fxy,0,1,4,6)
ans =
     87.2983
```

A `dblquad` által használt függvény képes kell hogy legyen egy `x` vektor és egy `y` skalár fogadására, és az eredményt egy vektorba kell adnia. A `dblquad` további argumentumai adhatják meg a toleranciát, és az integrálási eljárást (az alapértelmezés `quad`).