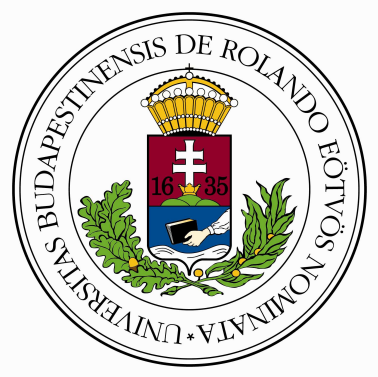
MATLAB 2013-2014

Bevezetés használatába, lineáris algebra, grafika, optimalizálás

Stoyan Gisbert

Magyarország, 2014. október
Javított verzió 2016. november



A tananyag (1-60. oldal) a TÁMOP-4.1.2.A/1-11/1-2011-0052 ELTE - PPKE informatika tananyagfejlesztési projekt, ill. (61-144. oldal) az ELTE Informatikai Karának 2014. évi Jegyzetpályázatának támogatásával készült.

A jegyzet előzményének, egy szerzői team „Matlab" könyvének ([36], Typotex kiadó) lektorai:
Dr. Fazekas István, Túri József, KLTE (az egész könyv, kivéve Statisztika fejezet)
Dr. Móri Tamás, ELTE (Statisztika fejezet)
Dr. Hegedűs Csaba, ELTE (Lineáris algebra).

A jelenlegi jegyzet lektora: Dr. Baran Ágnes, Debreceni Egyetem, Informatikai Kar

A javított verzióhoz:
Az eredeti kiadás után 2 éve elkészült a javítása, melyben nemcsak az elírásokkal foglalkoztunk, hanem a tartalommal is. Ez látszik mindenekelőtt a 4.4.2 alpontban, amelyet alaposan átdolgoztunk, beleértve programjait, és ezekkel kiterjedt új számításokat is végeztünk. Az indíték ehhez szolgált két előadásunk a Numerikus Analízis Tanszék numerikus szemináriumában.

Budapest 2016 november Stoyan Gisbert

Tartalomjegyzék

1. Bevezetés **5**
1.1. A MATLAB lényeges vonásai 5
1.2. A jegyzetről 6
1.3. A MATLAB-termékek áttekintése 6

2. Lineáris algebra **9**
2.1. Mátrixok 9
2.2. Mátrixok szerkesztése, megszerzése 10
2.3. További adattípusok 13
2.4. Fontos lineáris algebrai műveletek 17

3. A MATLAB egyes grafikus lehetőségei **21**
3.1. A `plot` utasítás 21
3.2. A `bar` utasítás 34
3.3. A `pie` utasítás 39
3.4. `surf` és `contourf` 45
3.5. A grafikai paraméterek 48
3.6. Grafikák mentése, exportálása 49
3.7. Mozgóképek és videok 49
3.8. „Graphical User Interface” (`GUI`) létrehozása 57

4. Optimalizálási feladatok **59**
4.1. Bevezetés az optimalizálásba 59
4.1.1. Lineáris legkisebb négyzetek, totális legkisebb négyzetek . 59
4.1.2. Gradiens módszerek 66
4.1.3. Newton-féle módszerek 70
4.1.4. Optimalizálási feladatok érzékenységéről és a minimumhely hibabecsléséről 75
4.2. Hátizsák feladatok 81
4.3. Az utazó ügynök 85
4.4. Modellezés differenciálegyenletekkel 97
4.4.1. Négy modellparaméter meghatározása 4 közönséges differenciálegyenletből álló rendszerben mérési adatokból . . . 97

4.4.2. Függvény meghatározása közönséges differenciálegyenletekből álló rendszerben mérési adatokból 107
4.4.3. Három modellparaméter meghatározása egy parciális differenciálegyenletben mérési adatokból 137
4.4.4. Parciális differenciálegyenletekben szereplő függvények meghatározásáról mérési adatokból 147

5. Irodalomjegyzék **149**

6. Tárgymutató **152**

1. fejezet

Bevezetés

1.1. A MATLAB lényeges vonásai

A MATLAB olyan nagyméretű programcsomag, amely főként a műszaki számításokat támogatja, és ehhez hatékony matematikai algoritmusokat használ. Erőssége az eredmények vizuális megjelenítése is, és a nagy adattömegek biztos, gyors kezelése. A MATLAB használatával lényegesen rövidül a programok elkészítésének ideje vektor- és mátrix-műveleteinek és sok matematikai, számos mérnöki toolboxainak („szerszámládák", azaz speciális témájú programgyűjtemények) révén. Ha a MATLAB-program elkészült, és futási ideje nem megfelelő, akkor segíthet a MATLAB C-compiler és a párhuzamosítási parancsai.

Alapvető adattípusa a *mátrix* – neve a MATrix LABoratory rövidítése – amely mátrixokat (és más változókat) nem kell deklarálni (bár speciáls helyzetben célszerű ezt megtenni, de csak a helyfoglalás céljából). Mivel a feladatokat mátrix- és vektorműveletekkel oldja meg, ciklusokat ritkán kell használni. Amellett a numerikus számításokat szimbolikus módon is végzi, nem áll meg egy nullával való osztás miatt (ennek eredménye általában `inf`, és `1/inf` eredménye 0, míg pl. `0*inf` eredménye `NaN` – „not a number" –, és a `NaN` hozzárendelhető változókhoz – ami még előnyös is lehet, ld. [36], 14.3.5. rész).

A MATLAB felhasználói felületeket (GUI – „graphical user interface") is tud gyártani, a gyorsan megírható MATLAB program lefordítható C nyelvre, és C, C++, Fortran programokkal tud együttműködni. A MATLAB nem mereven lezárt programcsomag, mindenki ki tudja bővíteni saját új modulok megírásával, úgynevezett m-fájlokkal.

A MATLAB fejlesztését Cleve Moler amerikai matematikus kezdte Fortranban, hozzá csatlakoztak Steve Bangert és Jack Little mérnökök, majd C nyelvre írták át.

Ma a MathWorks Inc. cég fejleszti tovább, Cleve Moler még mindig aktív [25], kb. egy millió felhasználója van a MATLAB-nak 175 országban, sok egyetemen ezzel tanítják a Numerikus analízist és pl. az automatikus szabályozást és a jelfeldolgozást.

Eredetileg a LINPACK és az EISPACK programcsomagokból nőtt ki. Mostanában még több ilyen csomag áll a MATLAB hátterében, pl. a PARPACK és az UMFPACK [38]. Ezeket a (mérnöki feladatok megoldásakor fontos) matematikai csomagokat pl. akkor mind külön kell installálnunk, ha a (nyilván) nem ingyenes MATLAB helyett az ingyenes „Octave" csomagot, úgyszólván a MATLAB öcsét, akarjuk installálnunk. Minden informatikusnak érdekes lehet az ingyenes, objektum orientált „Python" [20] is, amely ugyancsak több más programcsomag előzetes installációját tételezi fel.

1.2. A jegyzetről

A jegyzetben több helyen „magyar Matlab-könyv" címen a Typotex kiadónál, a GAMAX Kft., a MATLAB disztributorának a támogatásával megjelentetett [36] könyvre hivatkozunk majd, amelyet egy magyar szerzői team készített, az 5.1-es, ill. 7-es MATLAB-verzió alapján. Ennek megismerését javasoljuk az Olvasónak, hiszen a MATLAB-nak kitűnő help-rendszere van ugyan (ez interneten is elérhető, ld. lejjebb), de egy nyomtatott könyvnek megvannak a maga előnyei. A jelenlegi jegyzet ezt a magyar Matlab-könyvet veszi alapul, egyes témákat abból újra felvesz és tovább fejleszti a 2013-ban aktuális R2013b verzió alapján, amelyet a szerző a GAMAX Kft-től megvásárolt.

Világos, hogy egyetemi jegyzetben csak egyszerű példákon, kis programokkal fogjuk bemutatni a MATLAB lehetőségeit (kivétel ez alól viszont a 4.4.2. alpont).

1.3. A MATLAB-termékek áttekintése

A MATLAB értékét nagyban növeli a már említett kb. 40 „toolbox", amelyek közül – néhány inkább matematikai tartalmú mellett (mint a symbolic toolbox, görbeillesztési toolbox, parciális differenciálegyenletek vagy a statisztika toolbox) – főként mérnöki feladatok megoldását segítőek vannak (pl. jelfeldolgozás, vezérlési és rendszer identifikációs toolboxok), de adatbázis toolbox és párhuzamos számítás toolbox is van.

A mérnöki modellezést nagyban segíti a MATLAB „rokona", a Simulink rendszer, amit ugyanaz a cég fejleszt.

Részletesebben ld.

http://www.mathworks.com/products/

Összességében a következő területeken hasznos a MATLAB :

Műszaki számítások (technical computing), beágyazott rendszerek (embedded systems), vezérlési rendszerek (control systems) digitális jelfeldolgozás (digital signal processing) kommunikációs rendszerek (communications systems), kép és video feldolgozás (image and video processing), FPGA dizájn (FPGA design and codesign), mechatronika (mechatronics), kísérlet tervezése, mérések (test and measurement), számítógépes biológia (computational biology), számítógépes pénzügy (computational finance).

Hasznos internet címek:
A MATLAB „Documentation Center”:
http://www.mathworks.com/help/matlab/
A MATLAB-fórumokhoz el lehet jutni a következő címen keresztül:
http://www.mathworks.com/matlabcentral/newsreader/
Hasznos fájlok találhatóak és letölthetőek itt:
http://www.mathworks.com/matlabcentral/fileexchange/

Még a fentiekben nem érintett kérdésekről is lehet tájékoztatást kapni, ld.
http://undocumentedmatlab.com/
(vagy írjuk be a keresőbe : Undocumented Matlab)

Figyelmeztetjük az Olvasót, hogy mivel rengeteg irodalom található a MATLAB-ról az interneten, mindig nézze meg, milyen évből való, ill. melyik MATLAB-verzióra vonatkozik.

Még néhány tanácsunk:

Rendezzük be saját könyvtáraink mellett saját MATLAB-könyvtárunkat (pl. „ml” nevén) is! Ez MATLAB-ból a `pathtool` hívásának a segítségével a leggyorsabban csinálható meg.

Továbbá, mielőtt egy `xyz.m` m-fájlt vagy programot futtatnánk, használjuk a MATLAB szintaktikai ellenőrzéjét:

```
checkcode xyz
```

ld. `help checkcode` vagy a MATLAB-dokumentációt (a korábbi `mlint` már nem ajánlott).

Emellett legyünk figyelemmel azokra a kicsi barna és piros jelekre a MATLAB-editorban, amelyek már az m-fájlok megírásákor jelennek meg: ha egy sor befejezése után megmaradnak, akkor figyelmeztetésről ill. hibáról van szó, és az egérrel ezen jeleknek a közelébe jutva, még információt is kapunk, mi ott a gond!

Végül, amennyiben futtatáskor olyan hibajelzést kapnánk a MATLAB-tól, amivel nem tudunk mit kezdeni, kérdezzük meg valamelyik internet-keresőt, az egész hibajelzést belemásolva a keresőbe.

A MATLAB-bal kaptunk egy kísérleti eszközt a kezünkbe. Tehát kísérletezzünk !
Legfeljebb visszaszól nekünk angolul. (Arra érdemes időt áldozni, hogy az üzeneteket és a részletes help-szövegeket megértsük, inkább csak az ismeretlen szavakat fordítatva, nem egy egész szöveget a translator-ba másolva.)
Szintaktikailag hibás sorokat úgyis jelez, és segít a `checkcode`. Ha ilyen problémánk nincsen, de a program működése nem tetszik, változtassuk meg és próbáljuk ujra ! És ha gyanúsan sokáig fut, még mindig húzhatjuk `Control-C`-vel a vészféket ! **(:-)**

2. fejezet

Lineáris algebra

2.1. Mátrixok

A mátrix a MATLAB legfontosabb adattípusa, elemei lehetnek komplex számok, indexei mindig nagyobb 1-nél vagy egyenlők 1-gyel.

Kisméretű mátrixokat elemenként adhatunk meg, mégpedig a sor komponenseit üres hellyel vagy vesszővel, az egyes sorokat ,,;'' jellel választjuk el egymástól, pl.

```
A=[i 0 -3.1 \pi; 4 2.3 0 1; sqrt(2) 0 1/3 2*i-4];
```

ami a következő mátrixot adja, `format long`-ban kinyomtatva:

$$A = \begin{pmatrix} i & 0 & -3.1 & 3.141592653589793 \\ 4 & 2.3 & 0 & 1 \\ 1.1414213562373095 & 0 & 0.333333333333333 & 2i-4 \end{pmatrix}$$

A mátrix elemeinek tárolása a Fortran oszlopfolytonos konvenciójának megfelelően történik.

A mátrix elemeire így hivatkozunk: `A(i,j)`, pl.

```
>> s=A(1,3)
s=
   -3.1
```

a fenti mátrix esetén. Ha egy index nem egész, a MATLAB hibát jelez. Hibát kapunk akkor is, ha olyan mátrixelemet akarunk felhasználni, amely nem kapott értéket, viszont mivel előzetes deklaráció (azaz helyfoglalás) nem kell, írhatunk valamilyen, eddig fel nem használt helyre, pl. a fenti mátrix esetén

```
>> A(4,5)=5.2
```

amire az eredmény

$$A = \begin{pmatrix} i & 0 & -3.1 & 3.1415926...93 & 0 \\ 4 & 2.3 & 0 & 1 & 0 \\ 1.1414213...95 & 0 & 0.3333333...33 & 2i-4 & 0 \\ 0 & 0 & 0 & 0 & 5.2 \end{pmatrix}$$

Skalárok, vektorok, szövegek ábrázolása ugyanazzal a struktúrával történik. Így a skalárnak, mint mátrixnak, csak egy sora és egy oszlopa van.

Mátrixelemekre hivatkozhatunk egyetlen egy indexszel is, az oszlopfolytonos tárolásnak köszönhetően. Pl. az utolsó mátrixszal:

```
>> s=A(6)
s=
   2.3
```

is adja a fenti mátrix $(2,2)$-poziciójú elemét, mert az oszlopfolytonos tárolásnál ez a hatodik elem.

2.2. Mátrixok szerkesztése, megszerzése

A fenti mód mátrixok szerkesztésére csak kis méret esetén alkalmas. Azt még lehet fokozni. Ha pl.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 6 & 7 \\ -1 & 0 & 5 \end{pmatrix}$$

akkor

```
B=[A -A; 2-A A^2]
```

is szintaktikailag helyes, eredménye

```
B =

     1     2     3    -1    -2    -3
     4     6     7    -4    -6    -7
    -1     0     5     1     0    -5
     1     0    -1     6    14    32
    -2    -4    -5    21    44    89
     3     2    -3    -6    -2    22
```

és pl. az $A \otimes I$ Kronecker–szorzat is nagyobb mátrixokra vezet (a MATLAB-ban az egységmátrix `eye(n)` utasítással kapható):

```
C=kron(A,eye(3))
```

adja azt, hogy

```
C =

     1     0     0     2     0     0     3     0     0
     0     1     0     0     2     0     0     3     0
     0     0     1     0     0     2     0     0     3
     4     0     0     6     0     0     7     0     0
     0     4     0     0     6     0     0     7     0
     0     0     4     0     0     6     0     0     7
    -1     0     0     0     0     0     5     0     0
     0    -1     0     0     0     0     0     5     0
     0     0    -1     0     0     0     0     0     5
```

Hasonlóan hat az `A=repmat(B,m,n)` parancs: a `B` mátrixot rakja `m`-szer egymás alá, majd a kapott blokk-oszlopot `n`-szer egymás mellé. Pl.

```
B=[-1 2;3 -4];
A=repmat(B,4,3)
```

azt adja, hogy

```
A =

    -1     2    -1     2    -1     2
     3    -4     3    -4     3    -4
    -1     2    -1     2    -1     2
     3    -4     3    -4     3    -4
    -1     2    -1     2    -1     2
     3    -4     3    -4     3    -4
    -1     2    -1     2    -1     2
     3    -4     3    -4     3    -4
```

Amennyiben viszont `B` skalár, az `A` mátrix `B*E` lesz, ahol `ones(m,n)=:E` a lineáris algebrában elterjedt jelöléssel a csak egyeseket tartalmazó `m`×`n`-es mátrix.

Összefoglalva, ezzel a jelöléssel érvényes `repmat(A,m,n)=kron(E,A)`.

A nagyobb méretű, akár háromdimenziós mátrixoknál inkább valamilyen szabály adja meg elemeit. Ekkor ciklusok segíthetnek, vagy például az alkalmazásokban (így a parciális differenciálegyenletek diszkretizációjánál, ld. [34]) fontos, sok nullát tartalmazó ritkamátrixoknál az átlónkénti megadásra hasznos a `spdiags` függvény, ld. a magyar MATLAB-könyvet. Itt csak egy példával mutatjuk ezen függvény működését:

```
>> n=10; % A meret kicsi, hogy a kesz matrix attekinthet"o legyen
>> e=(1:n)'; % Oszlopvektor az 1 ... n szamokbol
>> B=e*[1 1 2 1 1]; % B 4 oszlopa e, 3. oszlopa 2*e
>> A=spdiags(B,[-round(n/2),-1,0,1,round(n/2)],n,n);
```

Az ekkor keletkező ritkamátrixot kinyomtatva, csak a „foglalt” (tehát csak a nemnulla elemeit érintő) három oszlopvektort kapjuk: a nemnulla elemek sor-

és oszlopindexeit, illetve az értékeit tartalmazzák. Ennél jobban mutatja a ritkamátrix (ez csak kis méret esetén értelmes, de a számításoknál elkerülendő) teljes alakját a `full` függvény:

$$full(A) = \begin{matrix} 2 & 2 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 3 & 0 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 4 & 0 & 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 3 & 8 & 5 & 0 & 0 & 8 & 0 & 0 \\ 1 & 0 & 0 & 4 & 10 & 6 & 0 & 0 & 9 & 0 \\ 0 & 2 & 0 & 0 & 5 & 12 & 7 & 0 & 0 & 10 \\ 0 & 0 & 3 & 0 & 0 & 6 & 14 & 8 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 7 & 16 & 9 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 & 8 & 18 & 10 \\ 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 9 & 20 \end{matrix}$$

Tehát a főátló feletti mellékátlók *eleje* pozíciójának megfelelően le lesz vágva, a főátló alatti mellékátlók *vége* pozíciójának megfelelően lemarad (ahol azokat a pozíciókat a `spdiags` második argumentuma adja).

Magában a MATLAB-ban beépített mátrixoknak egész sora van, amikről a `help gallery` paranccsal kérhetünk listát, vagy kereshetjük a 'gallery' szót a help-ablakban. Így például adott $n \geq 1$ egészszámmal

```
[V b s]=gallery('house',[1:n]',0)
```

adja a lineáris algebrából ismert `H` Householder–matrixot az `eye(n)-b*V*V'` formában, amelynek tulajdonsága az, hogy az `x` oszlopvektort (esetünkben `=[1:n]'`) tükrözi `s*`e_1-be (ld. [32]), ahol e_1 az `eye(n)` $n \times n$-es egységmátrix első oszlopa.

Az interneten készen kaphatóak egész mátrix-gyűjtemények, pl. Timothy A. Davis munkája (University of Florida Sparse Matrix Collection):

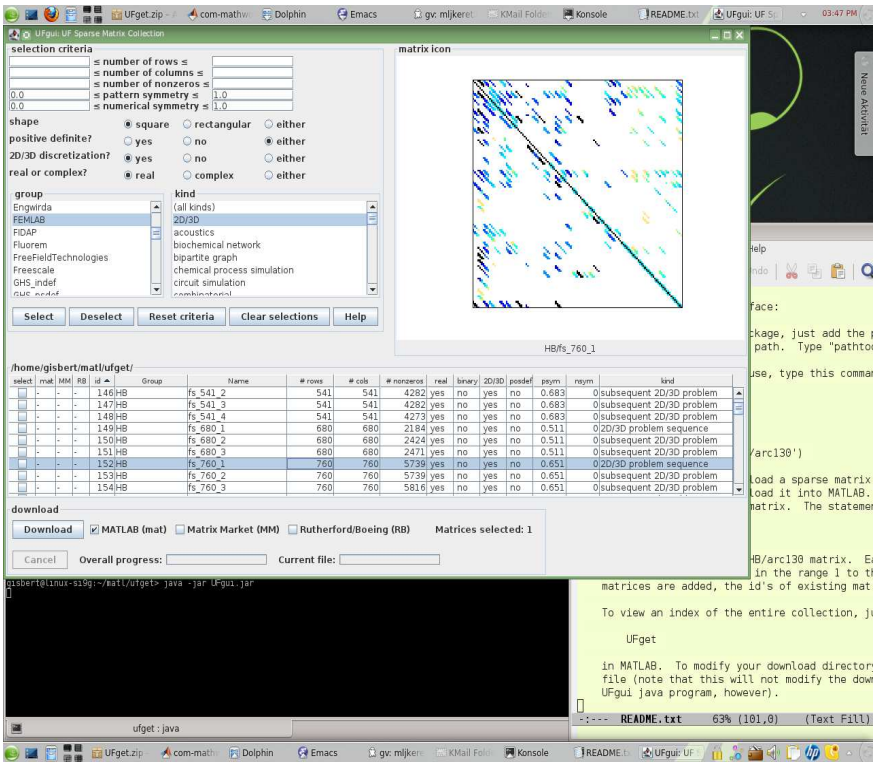
http://www.cise.ufl.edu/research/sparse/mat/UFget.html

cím alatt. Ez előnyös lehet, ha valamilyen új algebrai algoritmust akarunk kipróbálni. A mátrixok megszerzése innen azzal indul, hogy az ott látható `UFget.zip` könyvtárat letöltjük, kibontjuk, és annak README.txt fájlja alapján kiadjuk a könyvtárában a

java -jar UFgui.jar

parancsot. Erre megjelenik egy ablak (lényegében a lent mutatott 2.1 ábra) és a program először a mátrixok ikonjainak letöltésével foglalkozik (amelyek nagyjából a MATLAB `spy` ábráinak – vö. a [36] könyvvel – felelnek meg) – ami elég sokáig tarthat. Ezután az ábra alsó részében választhatjuk ki a letöltendő mátrixot (ebben segíthet, ha ott a mátrix jellemzését elolvassuk), ld. a 2.1. ábrát. Példáként azt a mátrixot választjuk, amely az `UFget` program `HB` csoportjának `fs_541_4` nevű eleme.

Viszont kényelmesebb lehet a MATLAB-ból – `UFget` letöltése után – a `UFget_example` utasítást kiadni, és a megjelenő ablakban a minden mátrixot



2.1. ábra. *Az UFget program ablaka: egy mátrix kiválasztása*

tartalmazó listát választani és azon belül a `fs_541_4` mátrixot, a letöltési módnál a „MAT”-ot jelölve. Ezután a mátrix letöltődik, pontosabban egy „Problem”, ami egy `record`. Most a

```
>> load fs_541_4
>> Problem % ez a parancs mutatja a record felepiteset
>> A=Problem.A
```

utasítások után rendelkezésünkre áll a mátrix – amelyre 2.4-ben visszatérünk.

2.3. További adattípusok

Miután a MATLAB számára egy valós szám is mátrixnak értelmezhető, vö. a következő utasításokkal (a `size` MATLAB-függvény adja a mátrix sor- és oszlopindexét):

```
>> s=pi;size(s)
```

```
ans=
    1     1
```

a más adattípusoknál mindenekelőtt a karakterláncok („string”-ek) említendők. Részletesen ld. a magyar MATLAB-könyvet.

Az alábbiakban csak egy fontos kérdéssel foglalkozzunk röviden, a helyközökkel, és hogy mi az üres helyek (a „space”-ek) sorsa karakterláncok egyesítésénél. Szemléltetésnek erre nézzük a következő példákat:

```
>>  s1='Informatikai';s2='Kar';

>> strcat(s1,s2),strcat(s1,' ',s2),[s1 s2],[s1,' ',s2]

ans =

InformatikaiKar

ans =

InformatikaiKar

ans =

InformatikaiKar

ans =

Informatikai Kar
```

Tehát a `strcat` – vagyis „string concatenation” *nem* veszi figyelembe a közbe iktatott helyközt, de az egyszerű egyesítés a szögletes zárójelekkel *igen*. Ennek megerősítéseként egy kicsit variáljuk a példát:

```
>> s1='Informatikai ';s2='Kar ';

>> strcat(s1,s2),strcat(s1,' ',s2),[s1 s2],[s1,' ',s2]

ans =

InformatikaiKar

ans =

InformatikaiKar

ans =
```

```
Informatikai Kar

ans =

Informatikai  Kar

>> length(ans)

ans =

    18
```

A szabály tehát az, hogy a `strcat` mindig kihagyja a karakterláncok végén lévő helyközöket, az egyszerű egyesítés nem: egyébként a `length` eredménye, vagyis a karakterlánc hossza az utolsó példában 16 lenne. Azt is látjuk, hogy egyik művelet sem mutatja kinyomtatásnál a karakterláncot kitüntető felső vesszőt az elején és a végén.

A fenti `' '` helyközt nem lenne érdemes felváltani az egyenértékű `blanks(1)`-gyel, de ha lenne 10 helyköz, akkor a `blanks(10)` már kényelmesebb, és a `blanks(10)'` azt eredményezi, hogy a kurzor 10 sorral mozog lefelé.

A `length` előfordulása a fenti példánkban arra is utal, hogy (sor)vektorként kezeli a MATLAB a karakterláncokat. Ezt arra használhatjuk, hogy az „elszúrt" `InformatikaiKar` karakterláncba beszúrunk egy helyközt:

```
>> s='InformatikaiKar';

>> ls=length(s);s(ls+1:-1:ls-1)=s(ls:-1:ls-2);s(13)=' ';s

s =

Informatikai Kar

>> length(s)

ans =

    16
```

Végül, több helyen találkozhatunk `cell array` (azaz „cellatömb") típussal, ld. pl. a 3.1. részben az egyszerű példát a 26. oldalon ilyen típusra, valamint a 26. oldalt, vagy a 3.3. részben a `pie33` m-fájlt (41. oldal, a program 35-38. sora). A cellatömbök elemei lehetnek számok, mátrixok, karakterláncok – sőt cellatömbök is. Mi karakterláncok tárolására használjuk majd.

Még a típusokhoz egy példa: az `input` MATLAB-függvény hatása az, hogy a program várakozik, míg a felhasználó be nem ad egy értéket. Az alábbiakban mindig az `1` értéket adjuk be:

» x=1;ma=input('choose'),if x==ma,'yes',else 'no',end

```
choose1
ma =
1
ans =
yes
» x=1;ma=input('choose ','s'),if x==ma,'yes',else 'no',end
choose 1
ma =
1
ans =
no
» isnumeric(ma),ischar(ma)
ans =
0
ans =
1
» x=1;ma=input('choose','s'),if x==str2num(ma),'yes',else 'no',end
choose1
ma =
1
ans =
yes
» x=1;ma=input('choose '),if int2str(x)==ma,'yes',else 'no',end
choose 1
ma =
1
ans =
no
» x=1;ma=input('choose'),if x==str2num(ma),'yes',else 'no',end
choose1
ma =
1
Error using str2num (line 33) Requires string or character array input.
```

Most adjuk be az `'1'` értéket:

```
» x=1;ma=input('choose ','s'),if x==str2num(ma),'yes',else 'no',end
choose '1'
ma =
'1'
ans =
no
» x=1;ma=input('choose ','s'),if x==ma,'yes',else 'no',end
choose '1'
ma =
'1'
ans =
no
```

2.4. Fontos lineáris algebrai műveletek

A standard lineáris algebrai feladat az

$$Ax = b, \quad b \in I\!R^n, \;\; A \in I\!R^{n\times n}$$

egyenletrendszer megoldása. Ez a „backslash" operátor segítségével történik:

```
x=A\b
```

Az egyszerű képlet mögött egy bonyolult program áll, amely nem az inverz mátrixot számítja ki (mert az rendszerint túl költséges), hanem a mátrix LU-felbontásán alapszik, több évtizedes munka eredménye, és amely a mátrix tulajdonságait (esetleges szimmetriáját, pozitív definitségét, ritkaságát) messzemenően vizsgálja és lehetőség szerint kihasználja. De komplex esetben is működik.

Egy példával rávilágítunk arra, hogy a méret mellett a mátrix tulajdonságaira reagál a „backslash" operátor.

A MATLAB ismeri az (n-nel gyorsan növekvő) kondiciószáma miatt hírhedt de szimmetrikus és pozitív definit Hilbert–mátrixot: `A=hilb(n)`. Ez a mátrix „telt": minden eleme különbözik nullától. A nagy kondiciószáma kapcsolatos a legkisebb sajátértékével, amely n-nel gyorsan tart nullához (míg legnagyobb sajátértéke csak lassan növekszik, úgy, mint `log n`). Így az egységmátrix hozzáadásával keletkező `A=eye(n)+hilb(n)` már jól kondicionált, és továbbra szimmetrikus és pozitív definit.

Az alábbi kísérletekben a következő mátrixokat használjuk:

```
A=eye(n)+hilb(n);              % szim. es pozitiv definit: "szpd"
A=-(eye(n)+hilb(n));           % szim. es negativ definit: "sznd"
A=eye(n)+hilb(n);A(1,n)=-1;    % aszimmetrikus, indefinit: "aind"
A=-(eye(n)+hilb(n));A(1,n)=1; % aszimmetrikus, indefinit: "-aind"
A=i*eye(n)+hilb(n);A(1,n)=-i; % komplex, indefinit: "kind"
```

Az következő táblázat adja az `x=A\b` futási idejét másodpercekben:

mátrix	$n = 3000$	$n = 4000$	$n = 5000$	$n = 6000$	$n = 7000$
szpd	1.28	3.26	4.84	8.72	14.32
sznd	1.57	5.94	7.43	12.40	19.77
aind	2.27	5.24	9.05	16.08	25.38
-aind	2.24	4.99	9.30	15.71	25.19
kind	9.02	19.02	34.56	58.16	–

A „–" a végén azt jelzi, hogy itt a MATLAB szabálytalanul fejezte be futását, mert már a mátrix előállításához is kevés volt a memória (2 Gb-os volt a RAM, és csupán a mátrix elemei 7000^2 komplex szám, ami megfelel $2 * 49 * 10^6$ valós számnak, azaz durván 784 Mb-nak. Ez kevés, viszont a MATLAB ezen célokra csak összefüggő tárhelyet foglal le. Valószínűleg ilyen nem volt itt.).

A lineáris algebra másik fontos művelete a **sajátértékek kiszámítása** (bár az nemlineáris feladat). Erre adunk példát a 13. oldalon nyert `A` mátrixszal:

```
>> format long;eig(A)

Error using eig

For standard eigenproblem EIG(A) when A is sparse, A must be
real and symmetric. Use EIGS instead.

>> eigs(A)

ans =

   1.0e+04 *

   4.527411651036773
   3.645096636409842
   3.237605752626074
   3.091388572876291
   2.758044190772034
   2.664632606643540
```

Az `eigs` standard outputja a 6 legnagyobb abszolútértékű sajátérték. De alkalmas paraméterekkel kiegészítve pl. a 10 legnagyobb abszolútértékű komplex sajátérték is kapható:

```
>> eigs(A,10,'LI')

Warning: Only 3 of the 10 requested eigenvalues converged.

> In eigs>processEUPDinfo at 1302

  In eigs at 335

ans =

   1.0e+04 *

                  0
                  0
                  0
                  0
                  0
                  0
                  0
   2.758044190772038
   3.091388572876285
   4.527411651036773
```

Ahogyan látjuk, a művelet nem volt igazán sikeres (amelynek háttere az, hogy – míg az `eig`-ben az eltolásos QR-algoritmus (ld. [32]) működik – itt, az `eigs`-ben a MATLAB egy iteratív módszer, az Arnoldi–iteráció [23] segítségével próbálja a feladatot megoldani).

Ezért inkább a telt mátrixhoz megyünk át (ami az 541-es méret mellett nem probléma), a következő scriptet használva:

```
% script az UFget-tel kapott matrix sajatertekeinek
% kiszamitasara, abrazolasara

load fs_541_4
Problem % ez a parancs mutatja a record felepiteset
A=Problem.A;

d=sort(eig(full(A)));dr=real(d);di=imag(d);
x=1:length(d);plot(x,di,'r*',x,dr*1e-4,'b.')
title(['Az fs-41-4-es matrix sajatertekeinek valos reszek'...
       ' szer 1e-4' es imaginarius reszek'],'fontweight','bold')
xlabel('A (rendezett) sajatertekek sorszama','fontweight','bold')
ylabel('piros: imag. resz, kek: valos resz szer 1e-4',...
       'fontweight','bold')
```

Ezen script outputját mutatja a 2.2. ábra a 20. oldalon.

Összehasonlításképpen megmutatjuk `d` (az abszolútérték nagysága szerint rendezett) utolsó 5 sajátértékét (mindig 1e-4-gyel szorozva, itt nemcsak a valós, hanem az imaginárius rész is! Elől viszont a sorszám áll):

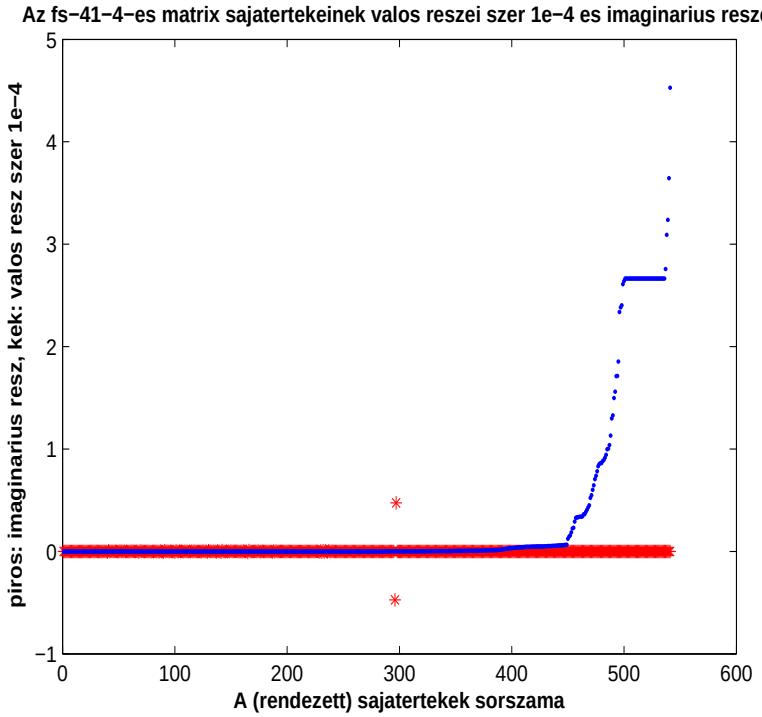
```
537  2.758044190772056 + 0.000000000000000i
538  3.091388572876280 + 0.000000000000000i
539  3.237605752626077 + 0.000000000000000i
540  3.645096636409863 + 0.000000000000000i
541  4.527411651036783 + 0.000000000000000i
```

Ezek tehát valójában valósak. Azért álljon itt a mátrix még néhány komplex sajátértéke is: (*1e-4)

```
229  0.000127472421008 - 0.000000565487246i
230  0.000127472421008 + 0.000000565487246i
231  0.000127529895404 - 0.000000579158399i
232  0.000127529895404 + 0.000000579158399i
233  0.000127557769128 - 0.000000584555407i
234  0.000127557769128 + 0.000000584555407i
```

és befejezésül a 2 maximális abszolútértékű komplex sajátérték: (*1e-4)

```
296  0.000342551634282 - 0.000047333589543i
297  0.000342551634282 + 0.000047333589543i
```



2.2. ábra. *Az* `A` *mátrix sajátértékei (valós részük* $*10^{-4}$*)*

Ezen számszerű eredmények fényében (ld. a 2.2. ábrát is) már jobban érthető, hogy az `eigs(A)` iterációja nem tudott megbirkózni a több nagyságrendbeli különbséggel, amely a legnagyobb valós és a legnagyobb abszolútértékű komplex sajátértékek között áll fenn.

3. fejezet

A MATLAB egyes grafikus lehetőségei

A MATLAB programcsomag sokoldalú 2- és 3-dimenziós ábrázolási lehetőségekkel rendelkezik.

Mint máshol is ebben a jegyzetben, a magyar Matlab-könyvet ismertnek tesszük fel, és itt csak néhány fontos, ill. ott nem tárgyalt esettel foglalkozunk.

A MATLAB-helprendszerében és az interneten ezenkívül sok segítség megtalálható, pl. a következő „szakács-könyvnek" egy része:

http://www.packtpub.com/matlab-graphics-and-data-visualization-cookbook/book

3.1. A `plot` utasítás

Ha néhány görbét szereténk kirajzoltatni és ehhez vannak vektoraink, amelyek az egyes görbe pontjainak x- és y-koordinátáit felsorolják, akkor a `plot` utasítás a megfelelő. Ennek alapformája pl.

```
plot(x,u,y,v,z,w)
```

ahol `x,u` az első görbe (azonos hosszúságú) vektorok, amelyek az első görbe x- és y-koordinátáit tartalmazzák, `y,v` és `z,w` pedig a 2. és 3. görbe adatait. Míg az `x` és `u` vektorok hossza egyenlő kell legyen, és hasonlóan az `y,v` és `z,w` párok vektoraié is, az egyes párok hosszai különbözhetnek.

Amennyiben a néhány görbe x-koordinátai megegyeznek és az `x` vektorban vannak, y-koordinátai viszont egy `A` mátrix sorait vagy oszlopait alkotják, akkor a `plot` megfelelő hívása

```
plot(x,A)
```

Itt és az előző példában felmerül, hogyan fogjuk az egyes görbéket megkülönböztetni egymástól?

A MATLAB különböző színekkel rajzolja az egyes görbéket, mégpedig egymás után, sorban használva a következő hét színt:

kék (b), zöld (g), piros (r), cián (kékes zöld, c), magenta (lila, m), sárga (y), fekete (k).

Ha több, mint 7 görbénk van, akkor elölről kezdi a színek felhasználását.

Fent zárójelben adtuk meg az angol szó első betűjét (ill. a fekete esetén, az utolsót), amely a szín rövid jelölésére használható megfelelő MATLAB-utasításokban. Ezenkívül a fehérnek is van ilyen rövid jelölése, w, ami hasznos nem fehér háttér esetén.

De egyébként a színskála minden színe összekeverhető az RGB (piros, zöld, kék) összetevői alapján, ami a MATLAB-ban egy háromdimenziós vektor, amelynek mindhárom komponense $[0, 1]$-ből való. Pl. a fekete `[0 0 0]`, a fehér meg `[1 1 1]`.

A fentiekre lássunk egy példát (amelyhez a 3.1. ábra tartozik) !

```
% script plott: rajzol nehany els"o Bessel-f"uggvenyt     %  0
x=0:0.1:6;                                                %  1
for i=0:9                                                 %  2
    A(i+1,:)=besselj(i,x)                                 %  3
end                                                       %  4
plot(x,A,'linewidth',2)                                   %  5
set(gca,'fontweight','bold','fontsize',14)                %  6
xlabel('x')                                               %  7
ylabel('y')                                               %  8
title('A Bessel-fele J f"uggveny')                        %  9
legend(' \nu=0',' \nu=1',' \nu=2',' \nu=3',' \nu=4',...   % 10
' \nu=5',' \nu=6',' \nu=7',' \nu=8',' \nu=9')             % 11
```

Ehhez illik néhány magyarázat, amelynek során a % kommentár jel mögötti sorszámra hivatkozunk:

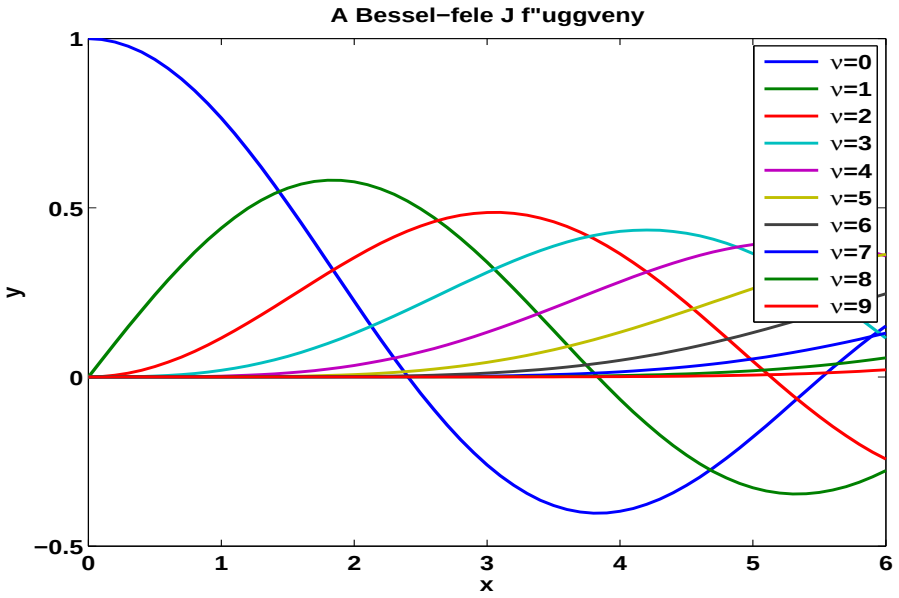
Az 1. sorban létrehozzuk az `x` sorvektort;

a 2-4. sorban az `A` mátrix sorait töltjük a (hengerkoordinátákkal kapcsolatos differenciálegyenletekben fontos, speciális) `i`-edik Bessel-féle `J`-függvény értékeivel; ez egy példa vektor-mátrix műveletekre: az `x` vektornak megfelel egy sor a mátrixból (és itt `x` oszlopvektor is lehetne!). Emlékezzünk, hogy vektorok és mátrixok legkisebb indexe 1-nél kisebb nem lehet;

az 5. sorban a `plot` utasítást azzal egészítettük ki, hogy minden görbe vonalvastagsága 2 méretű legyen. Az alapérték az 1, a 2 itt és a képernyőn esetleg csúnyának tűnik, de segíti a színek felismerését, és az ábra projekciójánál jobban láthatóvá teszi a görbét. Prezentációknál ez döntő;

a 6. sor `set`-utasítása azt eredményezi, hogy az ábrán látható koordinátarendszernek a feliratai (a skálabeosztás számai, a tengelyek és a `legend` feliratai) félköver jelekkel legyenek szedve és méretük 14-es legyen (ahol az alapérték a 12);

a 7. és 8. sor adja ezeket a tengely-feliratokat;



3.1. ábra. *1. példa a plot függvény használatára*

a 9. sor az ábra feletti címét adja;

a 10. sor sorolja az egyes, eddig csak színekkel megkülönböztetett görbék értelmét (ill. nevét). Esetünkben ez a Bessel–függvény rendszáma, és mivel egyetlen sorral ezt nem tudtuk kiírni, folytatási jellel: `...` végződik a sor, majd a 11. sorban következik a folytatás. E két sor azt is mutatja, hogy az ilyen feliratokban bizonyos Latex-parancsokat lehet használni, tehát a híres szövegszerkesztési nyelv (amelyben ez a jegyzet is íródott) parancsaiból néhányat, pl. az összes görög kis betűt (esetünkben a ν-t, de az „omikron” nincs már a Latex-ben sem, mert nem különbözik az „o”-tól).

Következőnek egy valamivel összetettebb példát mutatunk a színek és vonalfajták vezérlésére. Van három görbénk, `(x,y1),(x,y2),(x,y3)`, megfelelő `x` és `y1,y2,y3` vektorokkal, amelyekből kettőt a `plot` alapvető lehetőségeivel szeretnénk ábrázolni kék, ill. piros színben, szimpla vonalvastagságban és kihúzott módon. Viszont a 3. vonalat képzeljük sötétebb zöld színnel (mert az jobban látszik) és szaggatott módon (mert ez a görbe másképpen készült, mint a többi).

Először külön, a `plot(x,y3,'color',[0.1,0.4,0.1])` utasítással tesztelj ük ki, hogy ezen RGB-vektor színe jobban megfelel, mert valóban sötétebb, mint a `'g'`-vel kapható standard zöld. Ezután viszont a logikusnak tűnő

```
plot(x,y1,'b-',x,y2,'r-',x,y3,'color',[0.1,0.4,0.1],'linestyle','--')
```

nem lesz jó, ugyanis a végén álló `'color'` és `'linestyle'` megadása mind a három görbére érvényes lesz ! Emiatt így érjük el a kívánt eredményt :

```
plot(x,y1,'b-',x,y2,'r-')
hold on
plot(x,y3,'color',[0.1,0.4,0.1],'linestyle','--')
hold off
```

A kész ilyen ábra a 4.13. a 128. oldalon. Egyébként 2 különböző RGB-vektoros `plot`-utasítás a `hold on ... hold off` között nem adja a kívánt eredményt.

Tekintsünk most olyan esetet, amikor érdemes a görbéket másképpen jellemezni.

Vannak mérési adataink, átlaghőmérsékletek, amelyek változnak idővel. Ez ad egy első görbét, és ehhez számítunk egy, az adatokra illeszkedő 2. görbét.

Vegyük például a madridi maximális $\tilde{T}$ hőmérsékleteket (C°-ban, forrás: http://worldweather.wmo.int/083/c00195.htm, 30 éves átlagok):

i	1	2	3	4	5	6	7	8	9	10	11	12
$\tilde{T}_i$	9.7	12.0	15.7	17.5	21.4	26.9	31.2	30.7	26.0	19.0	13.4	10.1

Itt `i` adja a hónap sorszámát.

Azt kívánjuk, hogy ezekre az adatokra illeszkedjen a következő képlet

$$\begin{aligned} T(t) &= a + b * cos(\frac{2\pi}{365} * t) + c * sin(\frac{2\pi}{365} * t) = a + d * sin(\frac{2\pi}{365} * t + \alpha), \\ \sin\alpha &= \frac{b}{\sqrt{b^2+c^2}}, \quad d = \sqrt{b^2+c^2}, \end{aligned}$$

ahol t az idő, napokban mérve. Ekkor a lesz a középhőmérséklet és $2d$ a teljes (átlagos) hőmérsékleti ingadozás.

Az illesztést úgy fogjuk érteni, hogy minél jobban teljesüljön

$$T(t_i) \approx \tilde{T}_i \quad \text{minden } i\text{-re},$$

ahol t_i az i-edik hónap 15. napja.

Ezt szolgáltatja a következő m-fájl a legkisebb négyzetek értelmében. Ehhez a témához ld. részletesebben 4.1.1. vagy [32].

```
function [c,dd]=lsq(v,s,t)                                    % 1
% Kiszamitja a legkisebb negyzetek ertelmeben a               % 2
% c parameter vektorral adott k"ozelitest.                    % 3
% Hivasa:                                                     % 4
% [c,dd]=lsq(v,s,t)                                           % 5
% ahol                                                        % 6
% v - 12-dimenzios sorvektor a h"omersekleti adatokkal        % 7
% s - sztring, a hely neve                                    % 8
% t - sztring, a h"omersekleti egyseg nevevel                 % 9
% c - a legjobb k"ozelites parameter vektora                  %10
```

```
% dd - a honapok k"ozeps"o napjat tartalmazo vektora     %11
                                                         %12
if length(v)~=12                                         %13
   disp('12 honapra szamitja a k"ozelitest!')            %14
   return                                                %15
end                                                      %16
                                                         %17
A=[];                                                    %18
d=[31,28,31,30,31,30,31,31,30,31,30,31];                 %19
ddd=0;                                                   %20
for i=1:12                                               %21
    dd(i)=ddd+d(i)*0.5;                                  %22
    A=[A; 1 cos(dd(i)/365*2*pi) sin(dd(i)/365*2*pi)];    %23
    ddd=ddd+d(i);                                        %24
end                                                      %25
                                                         %26
c=(A'*A)\A'*v';                                          %27
disp(['A legjobb k"ozelites parameterei: ',num2str(c')]) %28
disp(['a h"omersekleti ingadozas: ',...                  %29
      num2str(2*sqrt(c(2)^2+c(3)^2))])                   %30
x=0:365;y=c(1)+c(2)*cos(x/365*2*pi)+c(3)*sin(x/365*2*pi);%31
yd=c(1)+c(2)*cos(dd/365*2*pi)+c(3)*sin(dd/365*2*pi);     %32
disp(['Az elteres: ',num2str(norm(v-yd))])               %32
                                                         %33
plot(dd,v,'*r',x,y,'k-')                                 %34
title([s,'i adatok'],'fontweight','bold','fontsize',14)  %35
xlabel('napok','fontsize',14)                            %36
ylabel(['fok ', t],'fontsize',14)                        %37
                                                         %38
sd=[];                                                   %39
for i=1:length(dd)                                       %40
    sd=[sd;sprintf('%5.1f',dd(i))];                      %41
end                                                      %42
cdd=cellstr(sd)                                          %43
set(gca,'xtick',dd)                                      %44
set(gca,'xticklabel',cdd,'fontsize',9)                   %45
```

Magyarázatok:

1. sor: az m-fájl nevét és paramétereit definiálja. (Pontosabban: az itteni `lsq` név miatt az MATLAB azt fogja javasolni, hogy a fájlt `lsq.m` nevén mentsük el. Ettől el lehet térni, de nem érdemes: a MATLAB valójában azzal a névvel hívja majd, amely alatt elmentettük.)

2-11. sor: az m-fájl „helpje”: ez jelenik meg, ha a parancssorba írjuk `help lsq`.

13-16. sor: Mindig jó, a paraméterek néhány alapvető tulajdonságát ellenőrizni. Ennél rövidebb és azonos eredményű

```
if length(v)~=12
   error('12 honapra szamitja a k"ozelitest!')
end
```

18-25. sor: a túlhatározott mátrix soronkénti szerkesztése, a `dd` output-vektor kiszámítása (amelynek i-edik komponense adja az i-edik hónap középső napját).

27. sor: a modell paramétereinek kiszámítása a Gauss-féle normálegyenletek alapján, tehát a döntő sor. (A futás során itt simán lehetne probléma: az A^TA mátrix lehet numerikusan szinguláris vagy rosszul kondicionált, de ez tipikusan akkor történik, amikor az A-t definiáló matematikai modell vagy a mérési helyek – itt *dd* – alkalmatlanok. Az ilyen problémákra a 4.1.1. pontban térünk vissza.) A 27. sor helyett, ha A ritkamátrix (és különösen, ha nagyméretű), alkalmazhatjuk a Q-nélküli QR-felbontást is, ld. `help qr`. Ugyanis az `R = qr(A)` utasítás az `A'*A` szorzat Cholesky–felbontásának felső háromszög szorzóját allítja elő : `A'*A=R'Q'*QR=R'*R`. Ekkor a 27. sor helyett írjuk :

```
R=qr(sparse(A));                          %27a
c=R\(R'\(A'*v'));                         %27b
```

28-30. sor: az eredmények kinyomtatása.

31-33. sor: Az eltérés kiszámítása (a *dd* helyeken) és kinyomtatása.

34-45. sor: a rajz elkészítése. Ezen belül különösen érdekes a 39-43. sor, mert ezzel azt érjük el, hogy az x-tengelyen *hol* (ezt a 44. sorban az `xtick` utáni paramétere, a `dd` jelöli meg) és *mit* fogunk mutatni (ezt a 45. sorban az `xticklabel` utáni paraméter, tehát a `cdd` adja meg). Az utóbbi sorok közül a 45. a problémás (a `dd`-vel már előbb rendelkeztünk): az `xticklabel` paramétere egy **cell array** („cellatömb") lehet, de ezt nem lehet egyszerűen a `cellstr(dd)` utasítással elkészíteni (ami hibajelzést váltana ki). Viszont a 39-42. sor után a `cellstr`, vagyis karakterláncok → cell array művelet már lehetséges. A 41. sorban fontos a `sprintf`-ben szereplő – C-ből ismert – `'%5.1f'` formátum, amely 5 helyen összesen, abból 1 helyen a decimális pont után, adja a számokat. A 41. sorban szereplő „ ; " jel miatt a ciklusban egy oszlopvektor készül (és emiatt fontos, hogy az ottani karakterláncok hossza egyenlő legyen). Lehetne „ , " használatával sorvektort is előállítani, de akkor a grafikonon az x-tengely feliratai összemosódnának.

Amennyiben egy grafikonon pl. a $2, \pi, 4, \pi^2$ számokat szeretnénk kinyomtatni az x-tengely mentén, akkor a 39-45. sor helyett a következőket írnánk:

```
set(gca,'xtick',[2,pi,4,pi^2])
set(gca,'xticklabel',{'2','\pi','4','\pi^2'},'fontsize',9)
```

Itt `{'2','`π`','4','`π^2`'}` a cell array.

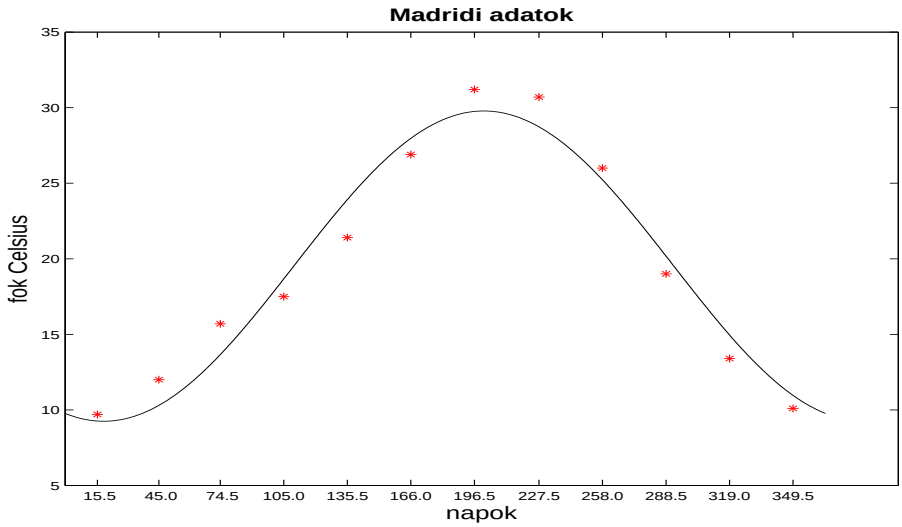
Összehasonlítás céljából még azt mutatjuk meg, hogy `sd = num2str(dd)` eredménye:

15.5 45 74.5 105 135.5 166 196.5 227.5 258 288.5 319 349.5

Viszont a 39-42. sor a következőt adja:

' 15.5'

' 45.0'



3.2. ábra. *3-paraméteres Fourier–illesztés a havi átlag hőmérsékleti adatokra*

’ 74.5’
’105.0’
’135.5’
’166.0’
’196.5’
’227.5’
’258.0’
’288.5’
’319.0’
’349.5’

Csak ezzel a verzióval kapjuk az `lsq(v,’Madrid’,’Celsius’)` hívásának jó eredményét, a 3.2. ábrát. A középhőmérséklet $a = 19.52\,C^\circ$ lett, és a hőmérsékleti ingadozázás $2d = 20.53\,C^\circ$. A négyzetösszeges eltérés viszont 5.22-nek adódott.

A fent említett cell array-nek van több más alkalmazása is. Ha pl. egy felirat (akár a `title`-ban, vagy `xlabel`-ben vagy `ylabel`-ben) egy sorban nem fér el, akkor a következőket tehetjük:

1. Cell array-be rakjuk a feliratot, pl. így:

```
ylabel({’Ez egy tul’,’hosszu felirat’})
```

vagy pedig

2. speciális karaktert használunk:

```
title([’Ez egy tul’,char(10),’hosszu felirat’])
```

Térjünk vissza alapvető témánkhoz, a görbék megkülönböztetéséhez! A következő további lehetőségek vannak erre. Egyrészt különféle vonalfajtákat alkalmazhatunk:

A már említett kihúzott vonal (jele „-") ill. szaggatott-pontozott vonalon (jele „-.") kívül még pontozott (jele „:") és szaggatott (jele „- -"), és ha semmi ilyet nem írunk elő, akkor töröttvonallal lesznek összekötve a pontok (tehát az eredmény ugyanaz, mint a "-" esetén).

Viszont a pontok feltűnő jelölésére („marker"ek segítségével) több lehetőségünk van:

jel	.	o	x	+	*
jelentés	pont	kör	x-jel	plusz	csillag
jel	d	p	h	s	
jelentés	gyémánt	ötszög	hatszög	négyzet	
jel	v	^	<	>	
háromszög	lefelé	felfelé	balra	jobbra	

A színek, vonalfajták és pont-jelölések felhasználása úgy történik, hogy a korábbi

```
plot(x,u,y,v,z,w)
```

alapverzió helyett minden adat vektor-párhoz teszünk 1-1 karakterláncot, pl.:

```
plot(x,u,'r-',y,v,'g-.',z,w,'*b')
```

Tehát itt az első adat vektor-pár piros törött, kihúzott vonallal jelenik meg, a második zöld „-." (vonalkázott, pontozott), a harmadiknál pedig az adatpontok kék csillagok, és nem lesznek összekötve (de a kék csillagos „markerek"et is összeköthetjük, pl. kék kihúzott vonallal, használva a '*-b' karakterláncot). Ehhez tartózik a 3.3. „akadémiai" példaábra, amikor speciálisan

```
x=0:.1:6;y=sin(x);z=cos(x);w=0.5*cos(2*x);
```

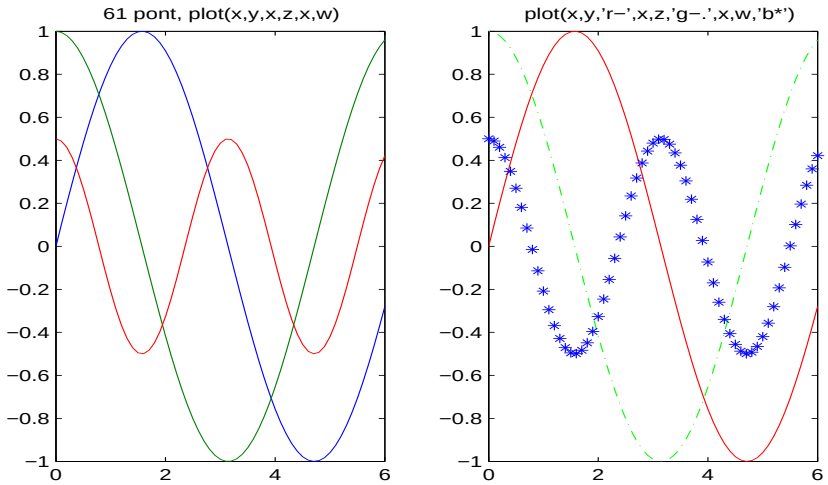
Egy gyakorlati példa: használjuk a MATLAB `polyfit` függvényét, hogy a fenti madridi hőmérsékleti adatokra polinomiális illesztést számítsunk ki. Legyen tehát

xd=[15.5 45.0 74.5 105.0 135.5 166.0 196.5 227.5 258.0 288.5 319.0 349.5],

ami az előző `lsq.m` program `dd` outputja, és

yd=[9.7 12.0 15.7 17.5 21.4 26.9 31.2 30.7 26.0 19.0 13.4 10.1].

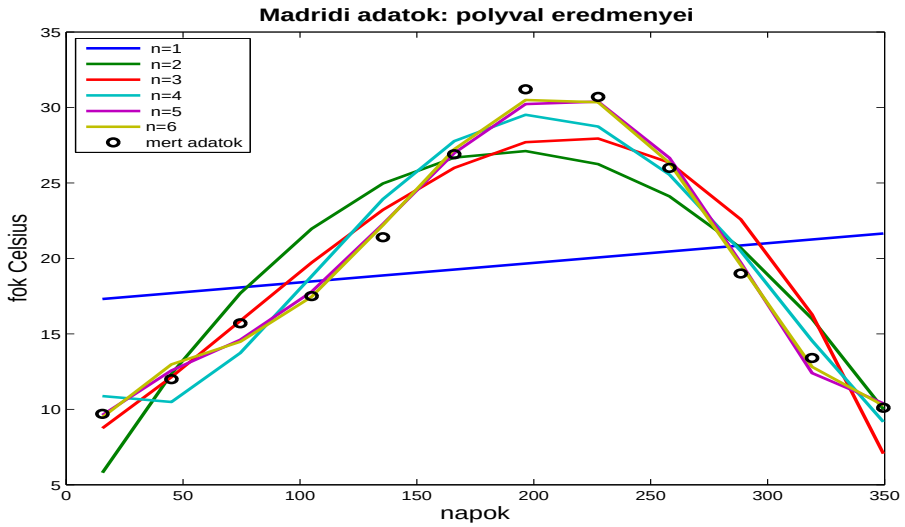
```
function polyf(xd,yd,n)                                          % 1
% Hasznalja a polyfit f"uggvenyt a polinomialis illesztesre      % 2
% Hivasa:                                                        % 3
```



3.3. ábra. *Különféle görbe-ábrázolási lehetőségek*

```
% polyf(xd,yd,n)                                                    % 4
% ahol                                                             % 5
% xd - az x-adatok vekta                                           % 6
% yd - az y-adatok vektora                                         % 7
% n - a polinomialis illesztes maximalis fokszama                  % 8
                                                                   % 9
if length(xd)~=length(yd)                                          %10
   error('Egyenl"o hosszusagu adatvektorok legyenek!')             %11
end                                                                %12
for i=1:n                                                          %13
    [p,s]=polyfit(xd,yd,i)                                         %14
    y(i,:)=polyval(p,xd);                                          %15
    [i,norm(y(i,:)-yd)]                                            %16
end                                                                %17
plot(xd,y,xd,yd,'ok','linewidth',2)                                %18
title('Madridi adatok: polyval eredmenyei',...                     %19
'fontweight','demi','fontsize',14)                                 %20
xlabel('napok','fontsize',14)                                      %21
ylabel('fok Celsius','fontsize',14)                                %22
legend(' n=1',' n=2',' n=3',' n=4',' n=5',...                      %23
'n=6','meresek','Location','NorthWest')                            %24
```

Magyarázatok:
1-7. sor: Az m-fájl neve és paraméterei.
10-12. sor: a paramétereknek egy alapvető tulajdonságának ellenőrzése.



3.4. ábra. *Polinomiális illesztés $n = 1 \ldots 6$ fokszámmal a havi átlag hőmérsékleti adatokra*

13-17. sor: a polinomiális illesztés kiszámítása, a kapott eredmények eltérésének kinyomtatása.

18-24. sor: A rajz elkészítése. A 18. sorban a `'linewidth'` tulajdonság 2-re való beállítása, a 20. sor `'fontweight','demi'`, majd a 20. és 21. sor `'fontsize'` tulajdonság 14-re való rögzítése mind prezentáció esetén a jó láthatóságát szolgálják. Ezenkívül a 23-24. sorban álló `legend` utasítás a görbe színeit az illesztés fokszámához rendeli hozzá. Az ezt tartalmazó doboz `''default''` helye a jobb felső sarok, de esetünkben jobbnak bizonyult a bal felső sarok, amit a `'location'` paraméter megadásával értük el.

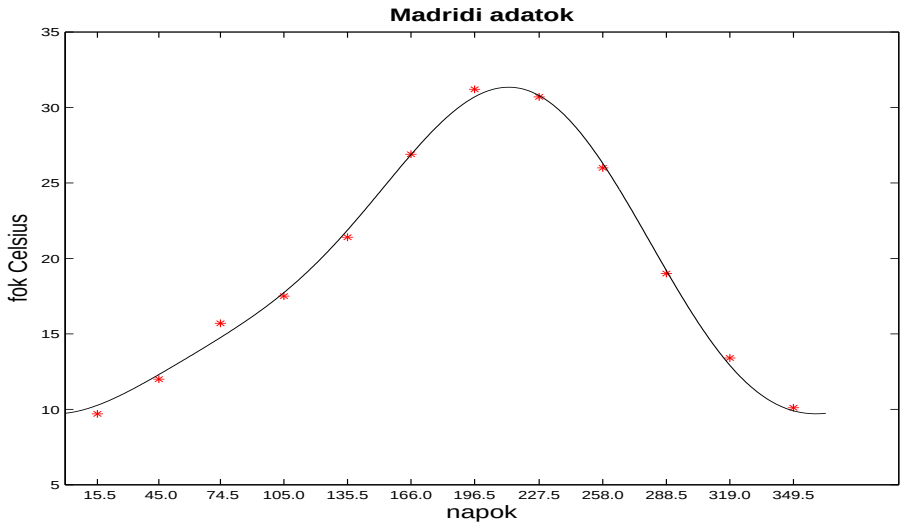
A fenti `polyf.m` programmal a következő eltéréseket kaptuk :

n	1	2	3	4	5	6
norm	25.24	10.08	7.77	5.26	2.34	2.16

Az eredő ábra a képernyőn és könyvben bizony nem szép a többi, ezekhez az adatokhoz ebben a pontban közölt képekkel összehasonlítva, de mást fog mondani prezentációkor a terem utolsó sorában ülő nezőnk.

Igaz, hogy az $n = 5$-höz és 6-hoz tartozó illesztések jobbak, mint a korábbi, a Fourier–sor elejével készített illesztés, de már $n > 3$-ra a `polyfit` program rosszul kondicionált feladatot jelez, és még az $n = 4$-hez tartozó polinomiális illesztés normája is nagyobb, mint a három paraméteres Fourier–sor esetén.

Emiatt a Fourier–soros illesztést ismételtük 5 paraméterrel, a fenti `lsq.m` program 23., 30. és 31. sorát megfelelően változtatva:



3.5. ábra. *5-paraméteres Fourier–illesztés a havi átlag hőmérsékleti adatokra*

```
    A=[A; 1 cos(dd(i)/365*2*pi) sin(dd(i)/365*2*pi) ...
     cos(dd(i)/365*4*pi) sin(dd(i)/365*4*pi)];            %23
x=0:365;y=c(1)+c(2)*cos(x/365*2*pi)+c(3)*sin(x/365*2*pi) ...
       +c(4)*cos(x/365*4*pi)+c(5)*sin(x/365*4*pi);       %30
yd=c(1)+c(2)*cos(dd/365*2*pi)+c(3)*sin(dd/365*2*pi) ...
       +c(4)*cos(dd/365*4*pi)+c(5)*sin(dd/365*4*pi);     %31
```

úgy, hogy most a közelítés az alábbi képletnek felel meg:

$$\begin{aligned} T(t) &= a + b * cos(\frac{2\pi}{365} * t) + c * sin(\frac{2\pi}{365} * t) \\ &\quad + d * cos(\frac{4\pi}{365} * t) + e * sin(\frac{4\pi}{365} * t). \end{aligned}$$

Ekkor eltérésként 1.5036-ot kaptuk (jobb, mint a 6-paraméteres polinomiális esetben), ld. a 3.5. ábrát is.

Végeredményben a polinomiális illesztés itt nem olyan sikeres, mint a Fourier–soros illesztés, aminek hátterében esetünkben egy fontos modellezési különbség is all: nemcsak a megszokott évszakok változása támogatja a Fourier–sorok által megvalósított *periódikus* modellt, hanem az (itt) harmincéves átlagolás is az ennek megfelelő adatokat hoz létre! Közben a polinomok a vizsgált intervallumon kívül gyorsan elszállnak – ami nem elhessegethető érv a rögzített intervallumra való, jogosnak tűnő hivatkozás mellett, hanem nyilvánvalóvá teszi a modellezési hibát.

Fent említettük annak a lehetőségét, hogy többsoros feliratokat készítsünk. Viszont a `text` utasítással bárhová az ábra környékén szöveget helyezhetünk el, pl. így:

```
text(x,y,'egy fontos tovabbi sz"oveg','fontsize',14)
```

ahol `x,y` a felirat kezdetének koordinátái (a rajzon közölt ábra koordinátarendszerének szempontjából). Ha rajzunk koordinátáira pl. érvényes $0 \leq x \leq 2$, $0 \leq y \leq 3$, akkor a

```
text(0,-1,'egy fontos tovabbi sz"oveg','fontsize',14)
```

a rajz alatt jelenik meg majd.

Mìg fent a madridi hőmérsékleteknél csak kis mennyiségű és a harmincéves átlagolással már előzetesen feldolgozott adatokról volt szó, következőnek közepes mennyiségű, feldolgozatlan adatokkal foglalkozzunk, kb. 7 évnyi Forint/Euro árfolyamokkal: 2006.04.30.-2013.05.02. Forrásunk a Magyar Nemzeti Bank

http://www.portfolio.hu/history/adatletoltes.php

alatt található adatai. Ezeknek formátuma

```
EUR (2006-04-30 - 2013-05-03)

d'atum     'ert'ek
2006-05-02 263.6100
2006-05-03 262.4000
... stb ...
2013-04-30 300.0300
2013-05-02 297.9000
```

Az adott címről akár egy grafikont is letölthetünk, de azt majd magunk hozzuk létre, így szövegfájlba kérjük az adatokat. Ennek neve legyen `eurft0613.txt`, amit a MATLAB-program „Import Data" nevezetű gombbal máris be lehet olvasni. De a dátumokban lévő „-" kötőjelek miatt ebből leginkább cell array lesz. Emiatt először egy csereparanccsal kitöröljük az összes ilyen zavaró jelt, amiután már mátrixot készíthet az „Import Data". Az „Import Selection" gombot megnyomva, feltűnik a „workspace"-ben egy 1753×2-méretű mátrix az `eurft0613` név alatt. Hogy később is kényelmesen használhassuk, adjuk neki az `A` nevet és mentsük ki a mátrixot:

```
A=eurft0613
save eurft A
```

A `save-load` parancs-párral kapcsolatban egy **súlyos lehetséges hibá**ra hívjuk fel a figyelmet : Ha vissza akarjuk nyerni az `eurft` tartalmát (pl. a munka megszakítása után) és nem azt írjuk be, hogy `load eurft`, hanem hibásan azt, hogy `save eurft`, akkor ezzel megsemmisítjük a régi `eurft.mat` fájlt és helyettesítjük egy új, de lényegében üres ugyanilyen nevű fájllal !

A fenti `A`-nak a tartalma most a „format long"-ban

```
A =

   1.0e+07 *

2.006050200000000  0.000026361000000
2.006050300000000  0.000026240000000
... stb ...
2.013043000000000  0.000030003000000
2.013050200000000  0.000029790000000
```

Ezen adatok feldolgozására és ábrázolására a következő programot írjuk:

```
% script eurft06_13
load eurft % Ebben az A tartalmazza a datumokat (1. oszlop)
% es az MNB Forint / Euro k"ozep arfolyamat (2. oszlop)

x=num2str(A(:,1)-2e7); % Elt"unik a "20" a datumok elejer"ol
t=str2num(x(:,5:6));   % az adatokbeli napok sora
lm=[31,28,31,30,31,30,31,31,30,31,30,31]; % a honapok hossza

T=size(A,1);
yy=A(:,2);
xx=zeros(T,1);% helyfoglalas xx-re yy-nal azonos meretre
e=2006;k=1;j=5;                                          % 12. sor
xx(1)=0;              % e szamolja az eveket, j a honapokat,
for i=1:T-1           % xx(i) adja az i-edik adatnap tavolsagat
    dt=t(i+1)-t(i);   % az 1. naptol, 2006.05.02-t"ol
    if dt>0
       xx(i+1)=xx(i)+dt;
    else
        dt=lm(j)+dt;j=j+1;
        if j==13,j=1; % ha ev vege van
           in(k)=i;k=k+1;e=e+1;                          % 21. sor
        end
        xx(i+1)=xx(i)+dt;
        if j==2
            if e==2008 | e==2012 % ha a februar valtoevben van
                xx(i+1)=xx(i+1)+1;
            end
        end
    end
end

[p1,s]=polyfit(xx,yy,1) % polinomialis illesztes
y1=polyval(p1,xx);
[p2,s]=polyfit(xx,yy,2)
y2=polyval(p2,xx);
```

```
B=ones(T,5)/5;B=spdiags(B,[-2,-1,0,1,2],T,T);
y3=B*yy;y3=y3(3:T-2); % szokasos 5-pontos mozgo-ablakos simitas
x3=xx(3:T-2);                                          % 39. sor

plot(xx,yy,'b-',xx,y1,'g-',xx,y2,'k-',x3,y3,'-r')
axis tight
ylabel('Forint','fontweight','bold')
xlabel('evek','fontweight','bold')
title(['Forint / Euro MNB k"ozeps"o arfolyama',char(10),...
      '2006.04.30.-2013.05.02.'],'fontweight','bold')
legend('arfolyam','linearis regr.','2.-foku regr.',...
       'mozgo ablak','location','NorthWest')
ss={'2007';'2008';'2009';'2010';'2011';'2012';'2013'};% 49. sor
set(gca,'xtick',xx(in(1:k-1)),'xticklabel',ss)
```

Felhívjuk a figyelmet arra, hogy itt a `plot`-ban szereplő vektorok hossza különböző, ld. a 39. sort. Ezenkívül az `ss` cell array-t nem kellene kiírni, hanem a programmal hasonlóan el lehetne készíteni, mint az `lsq`-programban, ld. a megjegyzéseket a 26. oldalon:

```
ss=[];               % Ezt tegy"uk hozza a 12. sorhoz,
ss=[ss;int2str(ss)]; % a 21. sorhoz,
ss=cellstr(ss);      % a 49. sor helyett.
```

A program eredménye a lenti 3.6. ábra, amely mutatja egyrészt, hogy számolhattunk 2014-ben az árfolyam további növekedésével, másrészt azt is, hogy a „mozgó-ablakos simítás" hatása főként a csúcsoknál érezhető (azok az eredeti adatok kék színét mutatják).

3.2. A `bar` utasítás

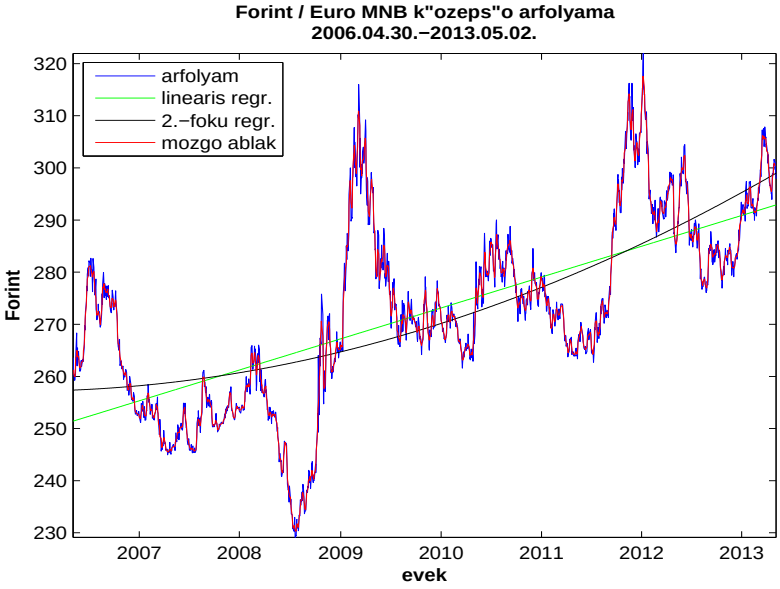
Most forduljunk a közgazdasági szakmában kedvelt `bar`-ábrázoláshoz, ahol téglalap formában mutatják a számokat. Azt a feladatot kaptuk, hogy az OECD-országokról készítsünk egy, a gazdasági súlyát bemutató `bar`-ábrát, tehát a GDP-t („Gross Domestic Product", bruttó hazai össztermék) nem osztjuk majd a lakosok számával.

Ehhez a forrásunk:

http://stats.oecd.org/Index.aspx?QueryId=350#

(Szakmai információ: Quarterly National Accounts, GDP - expenditure approach, measure: CPCARSA: millions of US dollars, current prices, seasonally adjusted, 2012 year, 4 quartals.)

Az adott internet oldalon már elő van készítve az adatok ábrázolása is, csak itt túl sok adat van egy „bar"-ábrázoláshoz. Ugyancsak elő van készítve az exportálás, mint Excel-fájl. Az utóbbit választva, a (`.mat`-kiterjesztésű fájlokra



3.6. ábra. *A Forint-Euro árfolyamának ábrázolása, különféle illesztése és simítása*

számító) `load` utasítás már nem működik, még az `-ascii` opcióval sem, de van egy külön parancs erre a helyzetre:

```
[A,CA]=xlsread('oecd2012.xls')
```

Fent `A` egy, a numerikus adatokat tartalmazó mátrix és `CA` egy cell array a szövegekkel (részletesebben ld. a MATLAB-helpjét).

Erre sajnos kapunk egy hibajelzést:
„Error using xlsread (line 247)
Unable to read XLS file /home/abc/oecd2012.xls. File is not in recognized format."

Ezt még ki tudjuk védeni, holott éppen Linux-gépen dolgozunk, és nem Windows-gépen: beolvassuk a fájlt a LibreOffice-be és kimentjük a megkívánt legújabb Windows-formátumban `oecd2012.xlsx`-ként. Ezután használjuk a fenti utasítás részletesebb formáját, tudva, hogy itt csak **egy** „worksheet"-ről van szó és a számunkra főként érdekes adatok az Excel-fájl C,D,E,F oszlopában és a 8-46. sorában vannak:

```
[A,CA]=xlsread('oecd2012.xlsx',1,C8:F46,'basic')\index{xlsread}
```

Itt a `'basic'` a nem Windows- – hanem Linux- – gép miatt szerepel.

De megint jön egy hibajelzés:

„Error using xlsread (line 247) Subscript indices must either be real positive integers or logicals."

Nyilván valójában nem erről van szó, hanem vagy a Linux-gépen mégsem megy a művelet, vagy a táblázatban csak pontokkal szereplő Görögország miatt.

Ekkor Görögországot töröljük, majd a megmaradt fájlt szétválasztjuk a szövegekre és a számokra még a LibreOffice-ben és kimentjük (az A mátrixba, ill. a CA cell array-ba). Ezután beolvassuk a kapott két tömböt az „Import Data" gombbal a MATLAB-ba és egész dollar-ra kerekítjük az A-beli számokat (hasonló problémakról lesz majd szó a 3.3 részben). Ennek eredménye:

ország / negyedévek	n1	n2	n3	n4
Australia	1009450	1020042	1018517	1023602
Austria	361668	364153	366464	368214
Belgium	433099	432267	434188	436680
Canada	1475310	1479588	1493320	1500722
Chile	383370	389996	398195	411115
Czech Republic	280924	279262	277402	277774
Denmark	230442	229758	233254	232063
Estonia	29974	30522	31243	31616
Finland	206783	206343	207136	207200
France	2335092	2346432	2359072	2361383
Germany	3286654	3308168	3327454	3316579
Hungary	212361	215190	217324	215780
Iceland	12198	11438	12310	12117
Ireland	193843	196570	196569	196220
Israel	231210	235625	239250	241576
Italy	1982195	1984163	1983907	1973846
Japan	4534260	4513612	4468604	4454344
South Korea	1533185	1531606	1534906	1545155
Luxembourg	46346	46692	46634	48018
Mexico	1989306	2005369	2034319	2031295
Netherlands	720868	723082	717195	720357
New Zealand	140069	143330	141620	141115
Norway	317259	316085	312893	316947
Poland	834788	842733	848701	850310
Portugal	272903	266794	265997	260739
Slovak Republic	133183	134280	135405	135904
Slovenia	56009	55541	55227	54727
Spain	1493087	1488362	1490890	1476963
Sweden	398271	401382	402394	402035
Switzerland	416039	415014	417205	417680
Turkey	1280636	1297832	1308656	1337536
United Kingdom	2240208	2244822	2290762	2283243
United States	15478300	15585600	15811000	15864100

és folytatása:

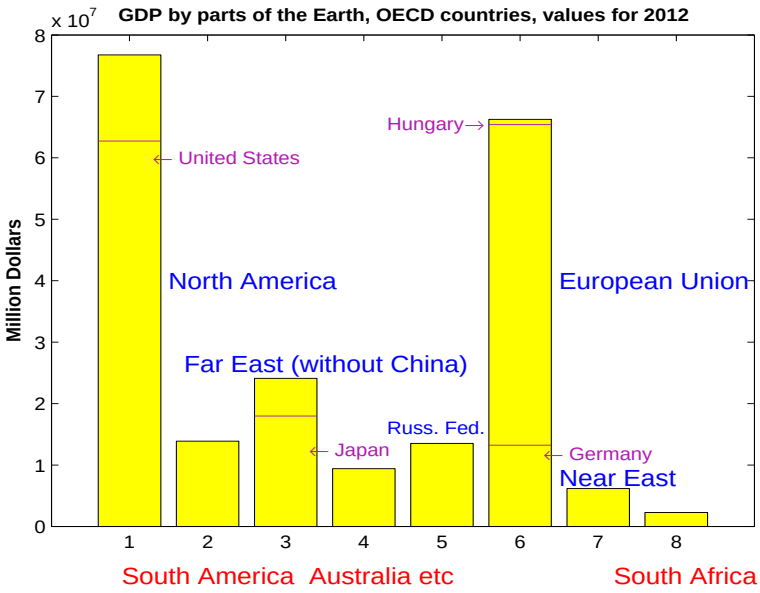
ország / negyedévek	n1	n2	n3	n4
Argentina	692096	714364	749964	788966
Brazil	2293042	2320529	2338313	2400244
Indonesia	1159707	1181985	1196653	1225368
Russian Federation	3254844	3327144	3416089	3521966
South Africa	554974	561996	572360	588718

Az `A, CA` adatainkat későbbi használatra kimentjük a MATLAB-ból:

```
save oecdat A CA
```

Most ezen kimentett adatok megjelenítésére írjuk a következő `szkript`-et, tehát paraméter nélküli m-fájlt:

```
% script oecdbar az OECD adatok abrazolasara

load oecdat
% ez tartalmazza az A matrixot a negyedevenkenti 2012-es
% GDP-adatokat, es a CA cell array-t, az OECD-orszagok neveivel

d=A*ones(4,1); % GDP-"osszeg a 4 2012es egyedevb"ol
[d,in]=sort(d);
ca=CA(in,:);

cont(1)=d(38)+d(31)+d(27);    % Eszak amerika
cont(2)=d(33)+d(22)+d(16);    % Del amerika
cont(3)=d(37)+d(29);          % tavol keleti orszagok
cont(4)=d(24)+d(25)+d(6);     % Ausztralia es k"ornyezete
cont(5)=d(36);                % Orosz federacio
cont(6)=d(35)+d(34)+d(32)+d(30)+d(28)+d(23)+d(18)+d(19)+d(21) ...
        +d(12)+d(13)+d(14)+d(15)+d(17)+d(7)+d(8)+d(9)+d(10)...
        +d(1)+d(2)+d(3)+d(4)+d(5); % Europai orszagok
cont(7)=d(26)+d(11);          % K"ozelkeleti orszagok
cont(8)=d(20);                % Del-Afrika

bar(cont,'y','edgecolor','k')
title('GDP by parts of the Earth, OECD countries, year 2012',...
      'fontweight','b')
text(1.5,4e7,'North America','color','b','fontsize',13)
text(6.5,4e7,'European Union','color','b','fontsize',13)
text(1.7,2.65e7,'Far East (without China)','color','b',...
                'fontsize',13)
text(6.5,0.8e7,'Near East','color','b','fontsize',13)
text(4.3,1.6e7,'Russ. Fed.','color','b')
text(1.1,-0.8e7,'South America','color','r','fontsize',13)
text(3.3,-0.8e7,'Australia etc','color','r','fontsize',13)
text(7.2,-0.8e7,'South Africa','color','r','fontsize',13)
```



3.7. ábra. *OECD-országcsoportok az egyes földrészeken*

```
dm=[0.7 0.1 0.7];            % s"otet lila szin
line([0.6 1.4],[d(38) d(38)],'color',dm)
text(1.3,6e7,'\leftarrow United States','color',dm)
line([5.6 6.4],[d(35) d(35)],'color',dm)
text(6.3,0.9*d(35),'\leftarrow Germany','color',dm)
line([2.6 3.4],[d(37) d(37)],'color',dm)
text(3.3,0.7*d(37),'\leftarrow Japan','color',dm)
line([5.6 6.4],[cont(6)-d(9) cont(6)-d(9)],'color',dm)
text(4.3,cont(6)-0.7*d(9),'Hungary\rightarrow','color',dm)
```

Eredményünket mutatja a 3.7. ábra, amelyen az OECD-országokat a következőképpen csoportosítottuk:

1. North America: United States, Canada, Mexico;

2. South America: Brazil, Argentina, Chile;

3. Far East: Japan, Korea;

4. Australia etc: Indonesia, Australia, New Zealand;

5. Russian Federation;

6. European Union: Germany, France, United Kingdom, Italy, Spain, Poland, Netherlands, Belgium, Switzerland, Sweden, Austria, Norway, Czech Republic, Portugal, Denmark, Hungary, Finland, Ireland, Slovak Republic, Slovenia, Luxembourg, Estonia, Iceland;

7. Near East: Turkey, Israel;

8. South Africa.

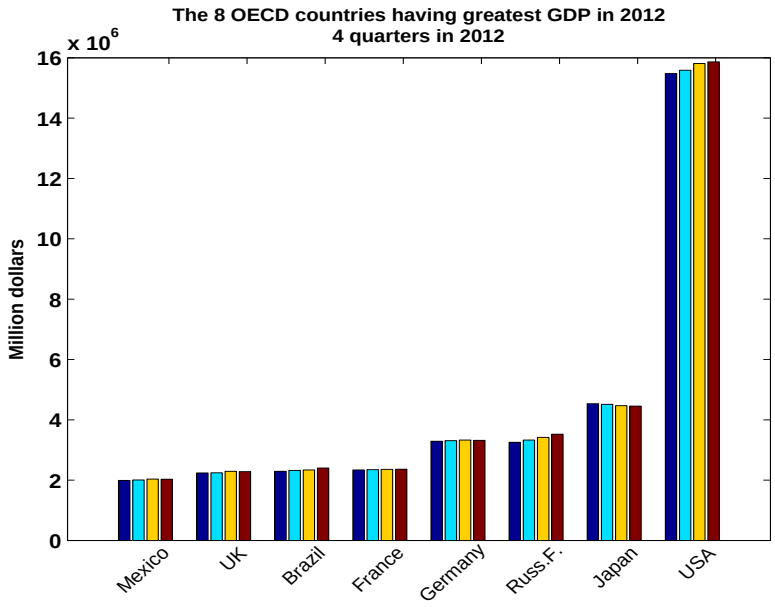
Most a 3.8. ábrán az OECD gazdaságilag 8 első országának mutatjuk a teljesítményét. Az egyes országok nevét azért 45° fokos szög alatt helyeztük el, mert máskülönben összeolvadnának (mint a 3.2. és 3.5. ábrákon az x-tengely mentén a napok, ha nem választottuk volna ott a hátrányos kis betűméretet). Az xticklabel-féle adatok elforgatása egyébként a MATLAB beépített rotateXLabels függvényével nem sikerült, a hibajelzés egy (a MATLAB-ban ritkán előforduló) típushibát említett, de a MATLAB-File Exchange Center-ből letölthető rotateticklabel nevű m-fájl tette a dolgát, ld. az alábbi programot és a 3.8. ábrát.

```
% script oecdbargr az els"o 8 OECD-orszag adatainak abrazolasara
load oecdat
% contains matrix A of quarterly GDP n 2012,
% and CA, cell array of names of OECD-countries

d=A*ones(4,1); % GDP-sum of the 4 quartals in 2012
[d,in]=sort(d);
ca=CA(in,:);
A8=A(in,:);A8=A8(31:38,:)

bar(A8,'grouped') % Itt kapjuk az oszloponkenti csoportokat
title(['The 8 OECD countries having greatest GDP in 2012',...
       char(10),'4 quarters in 2012'],'fontweight','bold')
ylabel('Million dollars','fontweight','bold')

ca8={'Mexico ';'UK ';'Brazil ';'France ';'Germany';'Russ.F. ';...
    'Japan ';'USA '}
set(gca,'fontsize',11,'xtick',[1:8]+.3,'xticklabel',ca8,...
    'fontweight','bold','fontsize',11)
% rotateXLabels(gca,45) % nem m''uk''od''ott
rotateticklabel(gca,45)
% Ez szarmazik az internetb"ol: ML File Exchange Center
```

A 3.8. ábrához az a következtetés tartozik (mivel az OECD az adatokból a szezonális hatásokat már eliminálta), hogy az Egyesült Államok és az Orosz Federáció (kisebb mértékben Brazilia is) erősödött 2012 folyamán, míg a többi ország lényegében stagnált, Japán viszont kissé rosszabbodott.

3.3. A pie utasítás

Ez az utasítás akkor jó, ha egy egységnek a felosztását össztevőire akarjuk megmutatni. A pie3 forma ugyanazt nyújtja, de van, akinek az jobban tetszik (ld.



3.8. ábra. *A 8 legerősebb OECD-ország 4 negyedévnyi teljesítményét mutatjuk 2012-ben*

lejjebb). Válasszuk az utolsó évek magyarországi GDP-nek a lényeges részei szemléltetésére. Adataink forrása a Központi Statisztikai Hivatal:

http://www.ksh.hu/docs/hun/xstadat/xstadat_evkozi/e_qpt002i.html

ahonnan Excel-fájlt le lehet tölteni. Ezután pl. ennek utolsó, „CM" jelzésű oszlopában lévő, az egész 2012-es évre vonatkozó adatait akarjuk kinyerni, amire

```
A12=xlsread('gdp9512.xls',1,CM1:CM19,'basic')
```

szolgálhatna (vö. a 35. oldallal). De a korábban elmondottak alapján ne csodálkozzunk, hogy erre egy hibajelzést kapunk: `Undefined function or variable 'CM1'`.

(A fájl direkt, Linux alatti `A12=xlsread('gdp9512.xls')` utasítással való beolvasásával nem is probálkoztunk már, egyébként az
„Error using xlsread (line 247)
File contains unexpected record length. Try saving as Excel 98."
hibajelzést okozná.)

Ezután az Excel-fájlt a LibreOffice-be olvassuk be és ott jelöljük ki azt az oszlopot, majd a jobb egérgombra kattintva válasszuk a `Format Cells` lehetőséget. A megjelenő új ablakban kérjük a `Number` átalakítását a `General` formátumba. Ezzel az oszlopbeli vesszők eltűnnek (a karakterláncok, amelyek addig ott voltak, számok lesznek), és most kimentjük az oszlopot egy új fájlba, amelynek formátumát `Text CSV`-nek választjuk, nevéként `gdp12`-t vesszük, és a kimentése során javasolt `Unicode (UTF-8)` formátumot elfogadjuk. Ezután a MATLAB

„Import Data” gombot megnyomva, kérjük az átalakítást mátrixba és nyomjuk az „Import Selection” gombot, az ott felkínált lehetőségek közül az „Import Data”-t választva. Ekkor a „workspace”-ben a `gdp12` mátrix jelenik meg.

Éppúgy járunk el összehasonlításul a 2010 évvel, amiután következik a 2 adat-oszlop kimentése `A02` mátrixként:

```
A10=gdp10;
A12=gdp12;
A02=[A10 A12]
save kshgdp A02
```

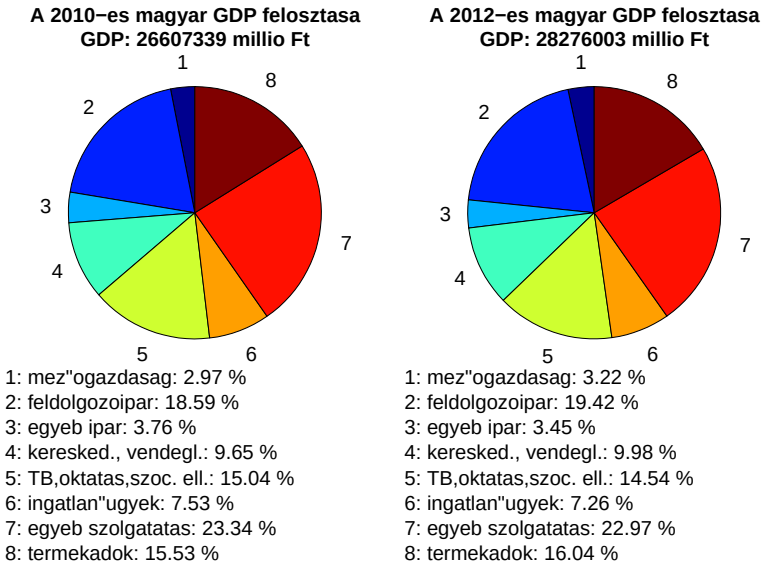
Most foglalkozzunk a `pie` utasítás programozásával, a `subplot` segítségével szembe állítva a 2 adatsort. Ennek során arra ügyeljünk, hogy az adatok között részösszegek is vannak, és csak a lényegesebb adatokra koncentrálunk:

```
function pie33(st,A0,ev)
% Az st fajl alapjan rajzol "pie" hivasaval torta-felosztast
% Hivasa:
% pie33(st,A0,ev)
% ahol
% st - fajlnev (tartalmazza oszloponkent a magyar GDP-szamokat)
% A0 - az st-beli matrix neve
% ev - az els"o ev szama

load(st) % tartalmaz nehany oszlopot (A0 matrix neven)
[i1 i2]=size(A0);
if i1~=15
    error('Oszlophossza nem 15!')
end

la1=8;                          % ennyi csoportra koncentralunk

for j=1:i2
    if i2>1
       subplot(1,2,j)
    end

    x(1)=A0(1,j);               % mez"og. stb
    x(2)=A0(3,j);               % feldolg. ip.
    x(3)=A0(2,j)-A0(3,j);       % egyeb ip.
    x(4)=A0(6,j);               % keresk.
    x(5)=A0(12,j);              % TB, oktat. szoc. el.
    x(6)=A0(10,j);              % ingatl."ugyek
    x(7)=A0(5,j)-sum(A0([6,10,12],j)); % egyeb szolg.
    x(8)=A0(14,j);              % termekado
    x=x(1:la1);
```

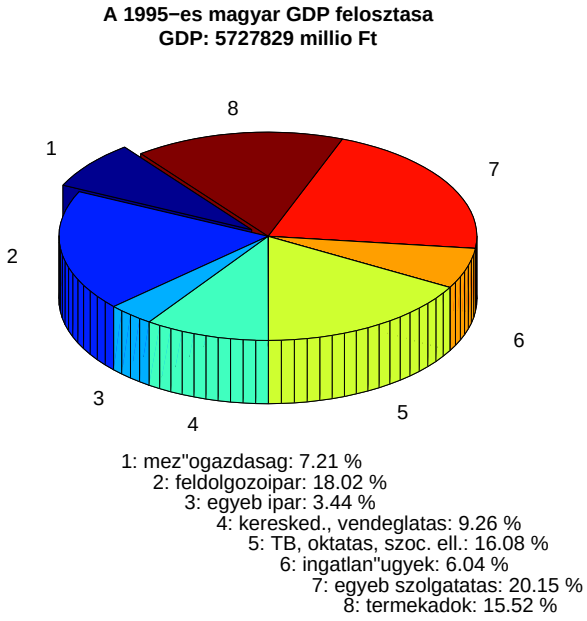
```
    xs=A0(15,j);p=100*x/xs;   % p tartalmazza a szazalekokat
    s=[];                     % 34. sor
    for i=1:la1
        s=[s;sprintf('%2d',i)];
    end
    cs=cellstr(s);
    pie(x,zeros(1,la1),cs)    % 39. sor
    title(['A ',int2str(ev+2*(j-1)),...
           '-es magyar GDP felosztasa',char(10),'GDP: ',...
           num2str(xs),' millio Ft'],'fontweight','bold')
    z=-0.5*ones(1,8);         % 43. sor
    text(-1.5,z(1)-0.8,['1: mez"ogazdasag: ',...
                        num2str(p(1),'%5.2f'),' %'])
    text(-1.5,z(2)-1.0,['2: feldolgozoipar: ',...
                        num2str(p(2),'%5.2f'),' %'])
    text(-1.5,z(3)-1.2,['3: egyeb ipar: ',...
                        num2str(p(3),'%5.2f'),' %'])
    text(-1.5,z(4)-1.4,['4: keresked., vendegl.: ',...
                        num2str(p(4),'%5.2f'),' %'])
    text(-1.5,z(5)-1.6,['5: TB,oktatas,szoc.ell.: ',...
                        num2str(p(5),'%5.2f'),' %'])
    text(-1.5,z(6)-1.8,['6: ingatlan"ugyek: ',...
                        num2str(p(6),'%5.2f'),' %'])
    text(-1.5,z(7)-2.0,['7: egyeb szolgatatas: ',...
                        num2str(p(7),'%5.2f'),' %'])
    text(-1.5,z(8)-2.2,['8: termekadok: ',...
                        num2str(p(8),'%5.2f'),' %'])
    colormap jet
    set(gcf,'color',[0.96,0.98,0.99])  % vilagos sz"urke hatter
end
```

A programhoz jegyezzük meg, hogy az `x` vektort az Excel-fájl „A” oszlopának az alapján állítottuk össze, továbbá, hogy a `text`-utasításokban egy, a százalékoknak megfelelő formátumot vettünk (`'%5.2f'`), a 34-38. sorhoz ld. a 26. és 34. oldalt, végül, hogy a `pie`-utasítások 2. paramétere arra ad lehetőséget, hogy a „tortaszeletek” (a „pie” az amerikai angolban éppen a tortát jelenti) közül valamelyiket kiemeljük: azt, amelynek számának megfelelően abban a 2. argumentumban a komponenst 1-nek választjuk (ld. lent).

Az utolsó sor `set(gcf)`-utasítása a hátteret festi, de ennek (a `figure`-nek) vannak további beállítható paraméterei, ld. a „Figure Properties” lehetőségeit az ábra „Edit” leguruló ablaka alatt.



3.9. ábra. *A magyar 2010-es és 2012-es GDP-adatok felosztása*



3.10. ábra. *A magyar 1995-ös GDP-adatok felosztása*

A 3.9. ábra azonnal mutatja, hogy lényeges változás nem történt a magyar 2010-es és 2012-es GDP-adatok felosztásánál, említésre legfeljebb a feldolgozó ipar 1 %-os és a termékadók fél %-os növekedése érdemes, meg a TB-nél, oktatásnál a fél %-os csökkenése.

Egyébként, a MATLAB képernyőjén a `pie`-kör valóban körként jelenik meg, de ha az ábrát szövegbe helyezzük, akkor ott a kör ellipszissé válhat, ha az ábrának nem jó méreteket választunk.

Az itt hasznos `subplot` utasításhoz tartozik még az a megjegyzés, hogy a részábrák száma nyilván nem lehet tetszőleges, mivel a MATLAB annak függvényében átméretezi a részábrák méreteit. Tanácsunk, hogy ne vegyünk többet, mint 4-6 részábrát (és ha pl. 4-et, akkor inkább 2×2 formában) !

Most a `pie3` utasítással foglakozzunk, ehhez a KSH 1995-ös adatokat bemutatva és a `pie33` m-fájlban a következő változásokat végrehajtva:

```
pie3(x,[1 zeros(1,la1-1)],cs)   % lesz az uj 39. sor
z=-[0:0.05:0.4];                % lesz az uj 43. sor
```

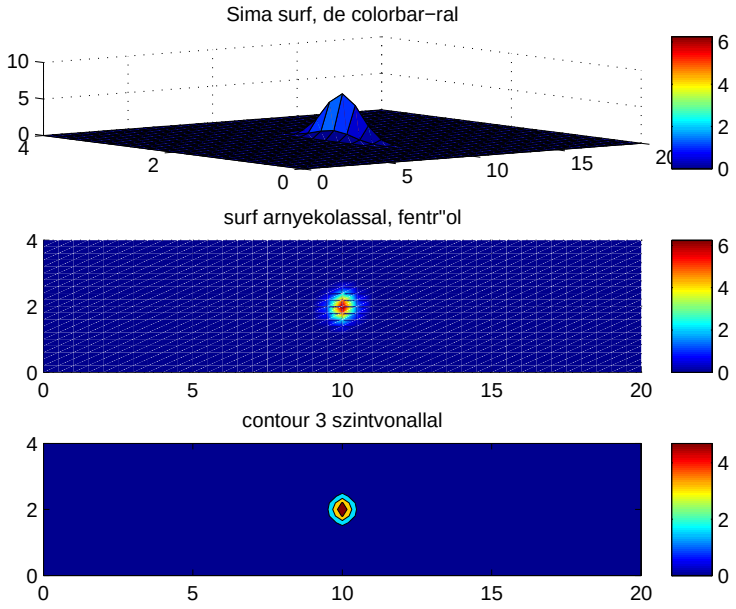
A 3.10. ábra mutatja, hogy mi a lényeges különbség a 2010-es és 2012-es GDP-adatok felosztásához képest: a mezőgazdaság (és beleértve a halászatat és erdészetet) több, mint kétszer nagyobb részesedése 1995-ben!

Megjegyezzük, hogy az 1-től 8-ig sorolt magyarázatokat lehetett volna (a z vektorral) egyenesbe terelni, de így hagytuk azért, mert világossá teszi, hogy az ábrán használt perspektíva erre a szövegre is kiterjed.

3.4. surf és contourf

Itt csak röviden hasonlítjuk össze a két ábrázolási módot egy nehezebb esetben, egyébként ld. a MATLAB-könyvet és a MATLAB-dokumentációt: egy Gauss-görbet kell ábrázolni. A görbe képlete $z = exp(-((x-a)^2 + (y-b)^2)/s^2)/s^2$, és kicsi s-re nem egyszerű azt bemutatni. A következő m-fájl segíthet megfelelő ábrázolási módot kiválasztani:

```
function surfcont(s,n)
% surf es contour "osszehasonlitasa
% Hivasa:
% surfcont(s,n)
% ahol
% s - a Gauss-g"orbe parametere
% n - a szintvonalak szama

a=10;b=2;
x=[0:a/20:2*a];
y=[0:b/10:2*b];

[xx,yy]=meshgrid(x,y);
z=exp(-((xx-a).^2+(yy-b).^2)/s^2)/s^2; % a Gauss-g"orbe
size(z)
minz=min(min(z))
maxz=max(max(z))

subplot(3,1,1)
surf(xx,yy,z)
title('Sima surf, de colorbar-ral')
colorbar('FontSize',12,'Location','EastOutside');

subplot(3,1,2)
surf(xx,yy,z)
shading interp
set(gca,'CameraPosition',[a,b,10*maxz])
colorbar('FontSize',12);
title('surf arnyekolassal, fentr"ol')

subplot(3,1,3)
contourf(xx,yy,z,n)
title(['contour ',int2str(n),' szintvonallal'])
colorbar('FontSize',12);
```



3.11. ábra. `surf` *és* `contourf` *összehasonlítása*

Az m-fájl `surfcont(0.4,3)` hívásának eredménye a 3.11. ábra.

A „sima” `surf` hátránya nyilvánvaló: egyrészt a „csúcs” értéke 5-nek tűnik az ábrán, a `colorbar` szerint 1 is lehetne, holott 6.25. Másrészt a `colorbar` összeolvadt a rajz feliratával (annak 20-asával), és ezt nem tudtuk kivédeni a `Location EastOutside` paraméter megadásával.

A fentről mutatott felület elég jó (a csúcs magasságát itt a legjobban lehet kivenni), de jobb ezt az előző lehetőséggel (ott ekkor a `colorbar` megtakarítható) együtt használni.

A `contourf` részábrája valamivel rosszabb a 2. részábránál, de problémás a szintvonalaknak a számát megadni csak az adatok alapján, azaz ha nem interaktívan dolgozunk. Már `n=5` szinte felismerhetetlenné teszi a csúcs alakját. (Megjegyezzük, hogy a `contourf`-hez választhatunk – a szintvonalak számán vagy értékein kívül – vonalfajtát és vonalszínt, ld. 3.1-ben a 28. és 22. oldalt (például `':b'`), míg az erre előre beállított érték `'-k'`, tehát a kihúzott vonal és a fekete. Ekkor, ha a fenti Gauss–görbés példánkban sok a vonal, azok egyetlenegy fekete folttá tömörülnek, de más szín is zavaró lehet, hiszen magassági értelme van, közben pl. térképek esetén nem szokás a pontozott szintvonal, és általában nehéz a helyzettől függően egyetlen alkalmas színt és vonalfajtát kiszámítani.)

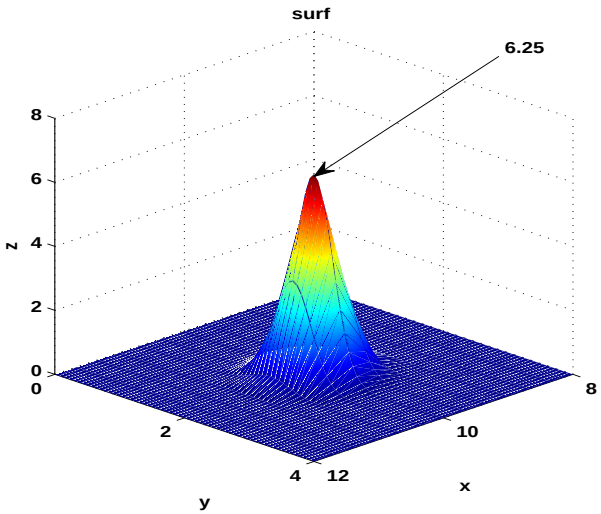
Bár igaz az érv, hogy a fenti problémák enyhülnek, hogyha finomabban osztjuk fel a [0,2a] és [0,2b] intervallumokat (pl. felezve az x,y lépéstávolságait) a fájl elején: a „sima" surf ekkor mutat egy sárga csúcsot (más látható változás viszont nincs). De már az s csökkentése 0.2-re ezt a hatást is eltünteti.

Befejezésül mutatunk egy tetszetős megoldást a fenti problémára a surf segítségével, amikor a kész ábrát még változtatjuk:

```
function surfarr(s,n)
% surf a Gauss-g"orben, utolagos nyil hozzadasaval az "insert"-tel
%
% Hivasa
% surfarr(s,n)
% ahol
% s - a Gauss-g"orbe parametere
% n - a felosztas finomsaga

a=10;b=2;
xx=[a-2:a/n:a+2];
yy=[b-2:b/n:b+2];

[xx,yy]=meshgrid(xx,yy);
z=exp(-((xx-a).^2+(yy-b).^2)/s^2)/s^2; % a Gauss-g"orbe
maxz=max(max(z))

surf(xx,yy,z)
shading interp
axis square
set(gca,'cameraposition',[a+maxz,b+maxz,2*maxz],...
    'fontweight','bold')
title('surf ','fontweight','bold')
xlabel('x','fontweight','bold')
ylabel('y','fontweight','bold')
zlabel('z','fontweight','bold')
```

Ezzel az m-fájllal gyártjuk le az ábrát, és amikor az megjelenik az ablakában, ott klikkeljük az insert lehúzható ablakot, válasszuk a text arrow lehetőséget, a megjelenő célkereszttel válasszunk egy alkalmas helyet a szélén, és tovább lenyomva tartva az egérgombot, induljunk a felület csúcsa felé, ahol leengedjük a gombot. Ezután a (fehér) szövegablakba írjuk azt, hogy \bf 6.25, azt a maximális felületi értéket, amelyet a programunk kinyomtatta. Ezen műveletek eredménye a 3.12 ábra.

Az „insert text arrow" lehetőséggel azért vigyázni kell: ha az ablak aktív marad és egy újabb rajzot ábrázolunk benne, akkor a nyíl ott marad!



3.12. ábra. `surf` *és utólagosan hozzáadott, a maximumra mutató nyíl*

3.5. A grafikai paraméterek

Gyakran szemben álltunk már azzal a problémával, hogy egy grafikánk valamelyik részlete nem tetszik, de nem tudjuk esetleg, hogyan lehetne ezt megváltoztatni. Ilyen például a rajz vonalvastagsága, ld. a 3.1 részt, amelyet a `plot` utasítás részeként lehet beállítani, ahogyan azt ott megmutattuk azzal, hogy a `plot` parancs végén megmondjuk, hogy melyik paramétert mire szeretnénk állítani, tehát konkrétan `'linewidth'` a paraméter és 2-re akarjuk előírni, együttesen tehát `plot(...,'linewidth',2)`. De pl. a `legend` utasítás esetén ez nem megy: ekkor csak együttesen a többi, a koordináta rendszerre vonatkozó paraméterekkel tudjuk azt befolyasolni

```
set(gca,'linewidth',2)
```

formában, vö. a 22. oldalon a `plott` m-fájl 6. sorával, ahol erre hasonlóan a vonal súlyát (félkövérségét) és betűi nagyságát írtuk elő.

Az alapvető problémánk tehát: mikor milyen grafikai paraméterek („ graphical properties") vannak, és hogyan lehet őket befolyásolni?

A kísérletezés és `help`-olvasás mellett 2 lehetőség van – ha már megvan a (nem igazán megfelelő) ábránk :

a) adjuk ki a `get(gca)` parancsot (ill. `get(gcf)`, ha a koordináta rendszeren kívüli rész, a „figure" érdekel), mert erre minden ilyen paramétert ki fog listázni;

b) az ábrán kattintsuk az „edit" gombot, és ott válasszuk, hogy a „Figure Properties" vagy az „Axis Properties" kellenek. Ezzel ugyanis a „Property Editor"-ba kerülünk, ahol a megadható paraméterek és jelenlegi beállításuk láthatóak, de van ott még egy „more properties" gomb is. Ha pl. az így elérhető

„GridLineStyle” paraméter jelentését, lehetőségeit nem tudjuk, akkor a főablak „help” gombját klikkeljük, beírjuk a „Search Documentation”-ba a szót és nyomjuk az „enter”-t. Aztán a „grid - Grid lines for 2-D and 3-D plots” lehetőséget választjuk, és részletes információt kapunk, példákkal együtt.

3.6. Grafikák mentése, exportálása

Közismerten a programozás interaktív folyamat, de ez többszörösen is így van grafikák kidolgozásakor. Ezeket nemcsak a képernyőn érdemes megnézni és megfelelően megváltoztatni, hanem mentés után is – ha nem a MATLAB-ábrák „fig” formátumát használtuk, hanem publikációra gondolva a `print`-parancshoz olyan opciót, mint „-depsc” vagy „-dwinc”, és végül, prezentációk készítésekor még harmadik helyen is érdemes a grafikát ellenőrizni: a teremben, leginkább ott, ahol majd vetítjük.

Parancssorból írva

```
print -depsc name
% name a kesz"ul"o fajl neve, ehhez j"on a ".eps"
```

elég jó eredményeket kapunk, de programból ennek megfelelője inkább ne legyen a szintaktikailag helyes, minimális

```
print('-depsc','name') % vagy -depsc helyett pl. -dwinc stb.
```

utasítás, hanem helyette jobb

```
print(gcf,'-depsc','-zbuffer','name')
```

A rövidebb verzió esetén ugyanis előfordulhat, hogy pl. a koordináta-rendszeren kívüli szín eltűnik, amelyet a képernyőn a

```
set(gcf,'color',[1 1 0]) % "gcf" az aktualis grafikai "figure"
```

utasítással narancssárgára változtattuk, vagy pl. a `text`-utasítással elhelyezett szöveg már nem pontosan a tervezett helyen jelenik meg, és akár el is tűnhet, mert a vágási téglalapon kívülre került.

A MATLAB helpje adja azt a magyarázatot, hogy a nagyobb hatékonyság érdekében különféle ábrázolásokat alkalmaz aszerint, melyik `print`-utasítást használunk.

3.7. Mozgóképek és videok

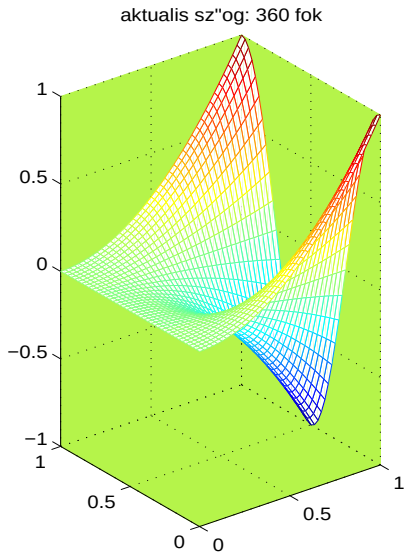
Amiután egy ciklusban több képet (a leendő mozgókép egy kockáját avagy `frame`-jét) állítottunk elő, ha egy `frame`-mátrixba gyüjtöttük ezeket, többször is meg tudjuk nézni mozgó formában, filmként, választható sebességgel. Ezt mutatjuk be a következő egyszerű példán:

```
function F=meshframe(n,dw,gw,str)
% mesh es frames: mozgokepek gyartasa
% Hivasa:
% F=meshframe(n,dw,gw,str)
% ahol
% n - a fel"ulet finomsaga (pl. 40)
% dw - a sz"og n"ovekmenye (pl. 1)
% gw - a vegs"o sz"og      (pl. 360)
% str- karakterlanc a "title"-hez
% F - frame-matrix (valojaban egy record)

if n*dw*gw==0
    error('inputhiba: n*dw*gw=0')
end

x=0:1/n:1;[x,y]=meshgrid(x);
z=x.*x.*cos(2*pi*y);
mesh(x,y,z) % Peldakent egy egyszer"u fel"ulet letrehozasa
axis equal  % probaljuk ki 'axis' nelk"ul is
[az,el]=view;
rot=0:dw:gw;

for i=1:length(rot) % A fel"ulet forgatasi ciklusa
    view([az+rot(i),el])
    set(gca,'Color',[sin(rot(i)/gw)^2 sin(pi/4+rot(i)/gw)^2 ...
        cos(rot(i)/gw)^2]) % id"ovel valtozo hatterszin
% Hozzateve 'Visible','off' a koordinata-rendszer lekapcsolhato !
    title([str,':  aktualis sz"og: ',num2str(rot(i)),' fok'])

    F(i)=getframe(gcf);
end
```

Hívjuk ezt az m-fájlt a 40, 2, 360 paraméterekkel (ha a `dw` növekmény `dw=1`, akkor elég nagy fájl keletkezik, ami magában nem veszélyes, de késlelteti a lejjebb következő `mlmovie` működését). Már a fájl futása során láthatjuk a mozgóképet. Ennek utolsó kockáját mutatja a 3.13. ábra.

Most a kapott `F` frame-mátrixot **mozgókép**ként lejátszhatjuk megadható sebességgel:

```
function mlmovie(F,nrep,fpm)
% m-fajl a movie lejatszasara
% Hivasa:
% mlmovie(F,nrep,fpm)
% ahol
```



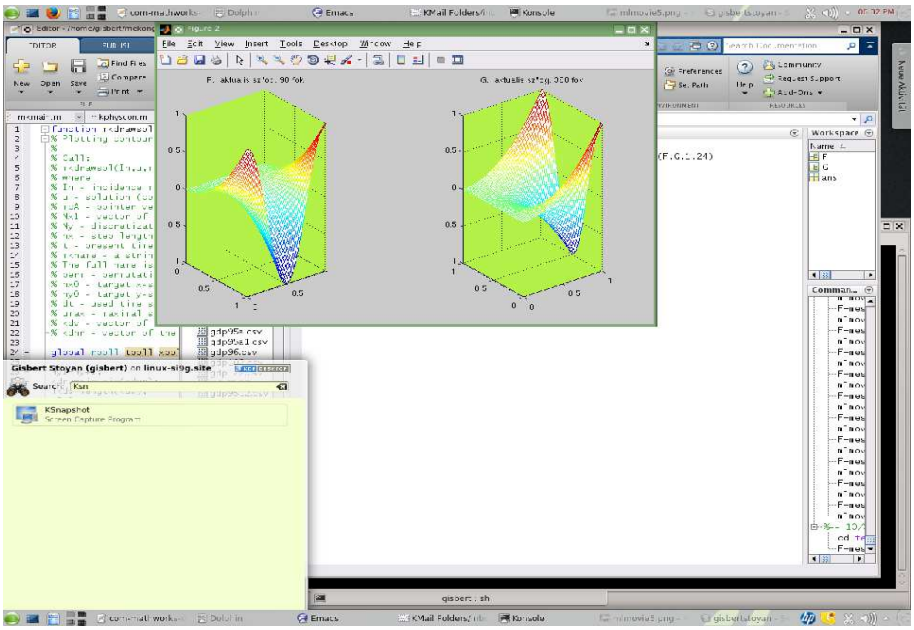
3.13. ábra. *A* `meshframe(40,2,360)` *hívásának utolsó kockája*

```
% F - a frame-matrix
% nrep - hanyszor akarjuk lejatszani
% fpm - a mozgokep sebessege: frame per masodperc

[h, w] = size(F(1).cdata); % Az 1. frame adja a dimenziokat
hf = figure;
set(hf, 'position', [100 100 w+40 h]);
movie(hf,F,nrep,fpm)
```

Ez az m-fájl a `[h,w]=...` és `set(hf,...` sorai nélkül is működik, de velük elkerülhetjük, hogy a `movie` ablaka hirtelen pl. a képernyő egész szélességét foglalja el.

Annak érdekében, hogy a `movie` lehetőségeit jobban bemutassuk, az `mlmovie` m-fájlnak olyan verzióját adjuk meg, amely egymásután mutat 2 frame-mátrixot egy ábrán:

```
function mlmovie2(F,G,nrep,fpm)
% m-fajl 2 movie lejatszasara
% Hivasa:
% mlmovie2(F,G,nrep,fpm)
% ahol
% F - az 1. frame-matrix
% G - a 2. frame-matrix
% nrep - hanyszor akarjuk lejatszani
```



3.14. ábra. *Az* `F=meshframe(40,1,90)`, `G=meshframe(40,2,360)`, *majd* `mlmovie2(F,G,1,24)` *hívások eredményeképpen a képernyő*

```
% fpm - a mozgokep sebessege: frame per masodperc

hf = figure('Position',[210 500 700 420]); % ld. "help figure"

axis off
movie(hf,F,nrep,fpm,[-150 0 0 0])          % ld. "help movie"
movie(hf,G,nrep,fpm,[240 0 0 0])
```

Ebben a fájlban figyelemre méltóak a `figure`- és `movie`-utasítások.

A fájlt alkalmazva két frame-mátrixra, `F=meshframe(40,1,90,'F')`-re és `G=meshframe(40,2,360,'G')`-re, a végén a 3.14. ábrát kaptunk. Viszont ezzel elértük a MATLAB határait: az ábrát nem tudtuk kimenteni, emiatt a képernyőt mutatjuk meg, ezenkívül a részábrák megegyező színskálában jelennek meg, az `F` (utólag) nem abban, amelyben a `meshframe` futása végén láthattuk, hanem a (másodikként aktív) `G`-nek megfelelő színskálában.

A videok készítését mutatjuk először ugyanazzal a példával.

A videok létrehozása a `VideoWriter` hívásával történik. Annak használata előtt érdeklődjünk a lehetséges formátumok iránt az alábbi módon és pl. a következő eredménnyel:

```
>> profiles = VideoWriter.getProfiles()
  Summary of installed VideoWriter profiles:

       Name                                        Description
  ---------------- ------------------------------------------------
  Archival         Video file compression with JPEG 2000 codec
                   with lossless mode enabled.
  Grayscale AVI    An AVI file with Grayscale Video Data
  Indexed AVI      An AVI file with Indexed Video Data
  Motion JPEG 2000 Video file compression with JPEG 2000 codec.
  Motion JPEG AVI  An AVI file with Motion JPEG compression
  Uncompressed AVI An AVI file with uncompressed RGB24 video data
```

Ha itt pl. az utolsó bejegyzésre, az `Uncomppressed AVI`-re klikkelünk, akkor részletesebb információt is kapunk:

```
 audiovideo.writer.ProfileInfo

  ProfileInfo Properties:

     Name:                   'Uncompressed AVI'
     Description:            'An AVI file with uncompressed RGB24
                             video data'
     FileExtensions:         {'.avi'}
     ColorChannels:          3
     FrameRate:              30
     VideoBitsPerPixel:      24
     VideoCompressionMethod: 'None'
     VideoFormat:            'RGB24'
```

A MATLAB mint 1. példát video készítésére a „peaks” nevű felület „lengetését” adja. `Script`-ként felírva:

```
% Script az alapvet"o video-utasitasok bemutatasara
writerObj = VideoWriter('peaks.avi'); % A VideoWriter hivasa
open(writerObj);                      % A video megnyitasa

Z = peaks;surf(Z);                    % A "peaks" fel"ulete
axis tight                            % 5
set(gca,'nextplot','replacechildren');% 6
set(gcf,'Renderer','zbuffer');        % 7

for k = 1:20
    surf(sin(2*pi*k/20)*Z,Z)          % 10
    frame = getframe;                 % A filmkocka megszerzese
    writeVideo(writerObj,frame);      % A kocka beirasa a videoba
end
```

```
close(writerObj);                       % A video lezarasa
```

Ebből a `script`-ből az 1. sorban a „peaks.avi” fájlnév megváltoztatható, az 5-7. sor elhagyható. a `k`-ciklus pl. `k=1:200`-ra is futhat, és a lengetés amplitudója pl. 1.2 is lehetne a 10. sorban, azaz ott `surf(1.2*sin(2*pi*k/20)*Z,Z)` is írható, továbbá egy szöveg, valamint egy felirat is elhelyezhető. Ekkor a `script`:

```
% Script az alapvet"o video-utasitasok bemutatasara
writerObj = VideoWriter('peaks3.avi');% A VideoWriter hivasa
open(writerObj);                        % A video megnyitasa

Z = peaks;surf(Z);                      % A "peaks" fel"ulete

for k = 1:200
    surf(1.2*sin(2*pi*k/20)*Z,Z)
    title('A peaks fel"ulet')
    text(-15,-15,['ido: ',int2str(k),' perc'])
    frame = getframe;                   % A filmkocka megszerzese
    writeVideo(writerObj,frame);        % A kocka beirasa a videoba
end

close(writerObj);                       % A video lezarasa
```

Ennek pl. 5. `frame`-je a 3.15 képen látható.

A keletkező `avi`-fájl mérete 1.5 Mb.

De a program érzékeny további változtatásokra, pl. gyakran egy hibajelzés jelenik meg, hogy a `frame` mérete nem megfelelő.

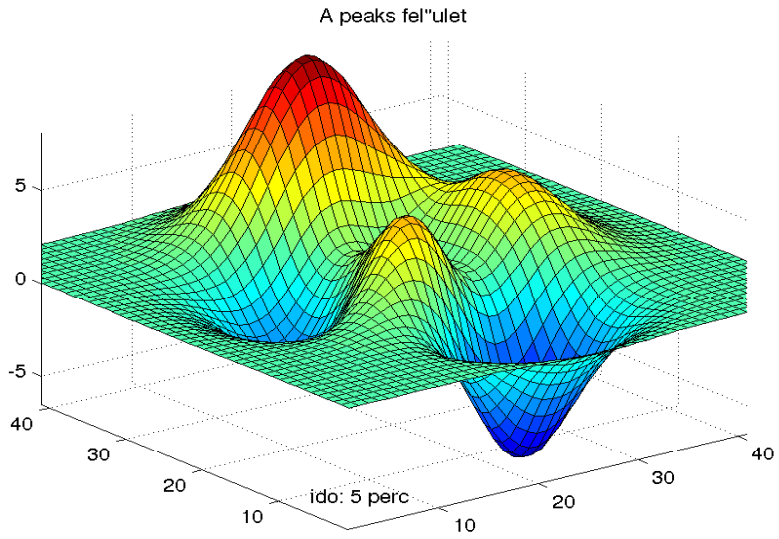
Erre egy példa az alábbi `vigomb2` m-fájl, amennyiben ott a sima `fri=getframe;` utasítást használnánk, mert akkor a reakció:

```
>> vigomb2(30,1,20)
Error using VideoWriter/writeVideo (line 383)
Frame must be 306 by 343

Error in vigomb2 (line 40)
    writeVideo(writerObj,fri);
```

Emiatt lent szerepel a 37. sorban látható összetett `getframe`-utasítás.

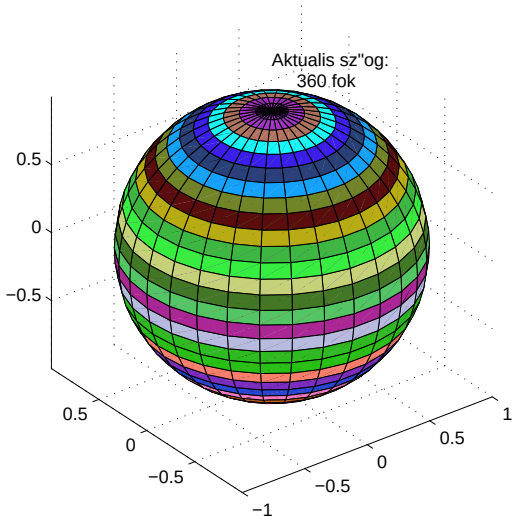
Hasonló hiba történhet, ha a (forgó és színét véletlenszerűen változtató) gömb merőleges átmerőjét túlságosan növeljük. Továbbá, a `text` helyett egy `title` bár a képernyőn jobban mutatna, de lemarad a videoról.

```
function vigomb2(n,dw,gw)
% mesh-g"omb: video gyartasa
% Hivasa:
```



3.15. ábra. *A lengetett „peaks” felület 5.* `frame`*-je*

```
% vigomb2(n,dw,gw)
% ahol
% n - a fel"ulet finomsaga (pl. 40)
% dw - a sz"og n"ovekmenye (pl. 1)
% gw - a vegs"o sz"og      (pl. 360)

if n*dw*gw==0
    error('inputhiba: n*dw*gw=0')
end

writerObj = VideoWriter('vigomb2.avi');
writerObj.FrameRate = 24;
open(writerObj);

[x,y,z]=sphere(n);
surf(x,y,z)    % Peldakent egy g"ombfel"ulet letrehozasa
axis equal

[az,el]=view;
rot=0:dw:gw;

for i=1:length(rot) % A fel"ulet forgatasa
    zz=z*(1+0.5*sin(4*pi/360*rot(i)));    % A g"omb pulzalasa
```



3.16. ábra. *A forgó és pulzáló gömb utolsó* `frame`*-je*

```
    surf(x,y,zz)
    axis equal

    text(0,0,1.3*(1+0.3*sin(4*pi/360*rot(i))),...
    ['Aktualis sz"og:',char(10),'     ',num2str(rot(i)),' fok'])

    view([az+rot(i),el])
    colormap(rand(64,3))

    fri = getframe(gca,[0 0 306 343]);    % 37. sor
    writeVideo(writerObj,fri);
end

close(writerObj);
```

Ezen fájl (valahányadik) `vigomb2(30,1,360)` hívása utolsó filmkockaként adja a 3.16. ábrát, a hozzátartozó `avi`-fájl mérete 12.1 Mb.

Általában is, a videok készítése hatalmas `avi`-fájlokat generálhat, amelyek viszont Windows- és Linux-gépeken (utóbbiakon a `vlc` program installációja után: ehhez írjuk a kereső ablakba azt, hogy ,,`vlc avi Linux`'') egyaránt megnézhetőek.

3.8. „Graphical User Interface" (GUI) létrehozása

Egy új programozási nyelvvel való találkozásakor az informatikus egyik első kérdése: Hogyan lehet itt ablakokat kreálni? A MATLAB esetén a válasz: a GUI-editorral, a `guide` paranccsal.

Mivel a `guide` használata eleinte elég bonyolult (mégpedig annak `fig`-formátumú részével ugyan boldogulnánk: húzzuk az ilyen-olyan gombot („static text" – felirat, „edit text" – beviteli ablak stb.) az „építő" mezőbe, javítsuk helyzetüket stb., csak semmi nem reagál), a következő utat javasoljuk:

Klikkeljük a `help`-et, a leguruló ablakban a `Dokumenation` részre kattintsunk, majd a `Search Dokumentation` sorába írjuk be azt, hogy `simple gui` és nyomjuk az entert. Válasszuk a felkínált listából először azt, hogy `About the Simple GUI`, majd ezen egy oldal elolvasása után és ugyanennek alján kattintsuk a `Lay Out the Simple GUI in GUIDE` szavakat. A leírást követve (amire bizony egy jó angol tudás elkel), szerkesztjük az előírt `GUI`-t. Itt a döntő az, hogy lássuk a `fig`-formátumú oldalt működtető lépéseket, amelyek annak hozzátartozó m-fájljában kellenek.

Amikor ezzel sikeresen végeztünk, csináljuk az egészet (a „simple" GUI létrehozását) másodszor is, de most eltérve, variálva az előírtakat: az elhelyezéseket, a feliratokat, a függvényeket, amelyeket majd ábrazol a `GUI` – és mentsük el új név alatt.

Most a `help`-be menjünk újra, annak `Dokumenation` részének `Search Dokumentation` sorába írva azt, hogy `Examples of GUIDE`, és nyomjuk az entert. A felkínált listából válasszuk az elsőt, `Examples of GUIDE GUIs`, és nézzük végig. Amelyik példa a legközelebb saját problémánkhoz, elképzelésünkhöz, abból indulunk ki, azt variálva, ahol kell.

A jelenlegi témához érdemes a magyar MATLAB-könyv 11.5.6-8 részét elolvasni, ahol az alapvető fogalmak leírását találjuk és mintapéldát, valamint a 13.1.5. részét is.

4. fejezet

Egyes optimalizálási feladatok megoldása

4.1. Bevezetés az optimalizálásba

Ez a téma igen fontos a mindennapi élet szempontjából : pl. szennyeződések és veszteségek minimalizálása, profitok vagy szavazatok maximalizálása, optimális utak és technológiák meghatározása. Itt is feltesszük, hogy az Olvasó hozzá tud férni a [36] magyar MATLAB-könyvhöz, továbbá a [26] műhöz (a feltételek nélküli optimalizálásról ld. [32]-t is).

Ha megvan a MATLAB-unkban az `optimization toolbox`, írjuk be a MATLAB parancssorába azt, hogy `optimtool`, továbbá, a lehetőségekbe való bevezetésként olvassuk el a ,,`Choose a solver`'' részt.

4.1.1. Lineáris legkisebb négyzetek, totális legkisebb négyzetek

A legkisebb négyzetek módszere lineáris illesztési feladatok megoldását szolgálja, ld. részletesen [32]. Itt arról van szó, hogy a b_i, $i = 1, \ldots, m$ hibákkal terhelt mérési adatokat szeretnénk egy

$$F(t) := \sum_{j=1}^{n} x_j \varphi_j(t) \approx b(t), \quad t = t_i,\, b(t_i) =: b_i\,, \;\; i = 1, \ldots, m, \tag{4.1}$$

alakú modellel visszaadni, ahol $m \gg n$. Ez egyrészt a vizsgált folyamat jobb megértését szolgálhatná, másrészt adattömörítésnek is felfogható.

A megfigyelés a diszkrét t_i időpontokban történik (digitálisan). Ilyenkor kézenfekvő az a gondolat, hogy a mérési hibák jelenléte miatt az $\{x_j\}$ paraméterek akkor határozhatók meg nagyobb biztonsággal, ha lényegesen több mérést végzünk (m db.), mint amennyi paraméterünk (n db.) van, és ekkor — a Gauss-tól származó ötletet alkalmazva — ezeket a paramétereket abból számítjuk ki, hogy

a

$$\sum_{i=1}^{m}(F(t_i)-b_i)^2=\sum_{i=1}^{m}\left(\sum_{j=1}^{n}x_j\varphi_j(t_i)-b(t_i)\right)^2\to\min_{\{x_j\}}! \tag{4.2}$$

minimum feladatot megoldjuk. Ugyanis arra általában nincs kilátás, hogy az

$$\sum_{1}^{n}x_j\varphi_j(t_i)=b(t_i),\quad i=1,\dots,m, \tag{4.3}$$

egyenletrendszer megoldható legyen, amikor $m\gg n$.

Legyen most

$$A=(a_{ij})=(\varphi_j(t_i))\in\mathbb{R}^{m\times n},$$
$$b=(b(t_1),\dots,b(t_m))^T,\quad x=(x_1,\dots,x_n)^T.$$

Ekkor (4.3)-at röviden felírhatjuk mint

$$Ax=b, \tag{4.4}$$

azaz lineáris, túlhatározott (téglalap alakú) egyenletrendszer alakjában, és a (4.2) minimum feladatot így tudjuk leírni:

$$J(x):=\|Ax-b\|_2^2\to\min_x!\,, \tag{4.5}$$

ahol $\|\cdot\|_2$ az $\mathbb{R}^m$ euklideszi normája és $A\in\mathbb{R}^{m\times n}$, $x\in X=\mathbb{R}^n$, $Ax\in\mathbb{R}^m=Z$. A továbbiakban (4.4) helyett írjuk

$$Ax\cong b. \tag{4.6}$$

Ezzel akarjuk aláhúzni, hogy (4.4)-nek általában nincs megoldása, és egyben emlékeztetni akarunk arra is, hogy (4.4) megoldását a (4.5) értelmében keressük.

Legyen $y_b\in Y:=R(A)$ az adott b legjobb közelítése Y-ban. Ez az y_b létezik és egyértelműen meghatározott, ld. [32]. Amikor $m=n$ és A reguláris, akkor $Z=Y=X$ és maga $b(=Ax)$ lesz a legjobb közelítés Y-ban. Ekkor az x is egyértelműen meghatározott. Általában viszont végtelen sok olyan $\overline{x}$ tartozik az $y_b\in R(A)$ legjobb közelítéshez, hogy $A\overline{x}=y_b$. Legyen x_b egy ilyen vektor. Az $Ax_b=y_b$-t jellemző ortogonalitási tulajdonság az, hogy az euklideszi skalárszorzatban

$$0=(y,b-y_b)\quad\text{minden }\ y\in Y\text{-ra},$$

és mivel $Y=R(A)$, ez azzal ekvivalens, hogy teljesül

$$0=(Ax,b-y_b)=(Ax,b-Ax_b)\quad\text{minden }\ x\in X\text{-re},$$

vagyis

$$0=(x,A^Tb-A^TAx_b)\quad\text{minden }\ x\in X\text{-re}.$$

Ez azt jelenti, hogy x_b eleget tesz az

$$A^T Ax = A^T b \tag{4.7}$$

egyenletnek. Más szóval ennek az egyenletnek mindig van megoldása. Fordítva is igaz a levezetés : ha x_b (4.7)-nek egy tetszőleges megoldása, akkor ez egyben (4.5) megoldása is, és Ax_b a b egyértelműen meghatározott y_b legjobb közelítése Y-ban. Az A mátrix mint X-ből Y-ba ható leképezés tehát eltűnteti az esetleges többértelműséget (ugyan nem (4.5)-ből, de) a (4.7) megoldásaiból. Ezt közvetlenül beláthatjuk a következő azonosságból :

$$\begin{aligned} J(x) &= \|b - Ax\|_2^2 = \|b - Ax_b + (Ax_b - Ax)\|_2^2 \\ &= \|b - Ax_b\|_2^2 + \|A(x_b - x)\|_2^2 + 2(b - Ax_b, A(x_b - x)) \\ &= J(x_b) + \|A(x_b - x)\|_2^2 \geq J(x_b), \end{aligned}$$

amely igaz, mert x_b a (4.7) egy megoldása.

A (4.7) rendszert *Gauss-féle normálegyenlet(ek)*nek hívjuk. Használatához ld. pl. az `lsq` program 27. sorát a 24. oldalon.

$A^T A$ pontosan akkor lesz pozitív definit, amikor oszlopai lineárisan függetlenek.

Tegyük fel, hogy ez igaz. Ekkor megoldhatjuk a (4.7) egyenletrendszert az LDL^T-felbontással is. Az x_b megoldás leírható a következő alakban :

$$x_b = (A^T A)^{-1} A^T b. \tag{4.8}$$

Ezen az úton a megoldás kényelmesen készíthető el MATLAB-ban, ld. a 26. oldalon a 27. sorhoz fűzött megjegyzéseket is.

De lehetnek már kicsi n mellett is rosszul kondicionált feladatok, és ekkor nagy pontossági veszteség, hogy a *Gauss-féle normálegyenletek* $A^T A$ *mátrix*ának a kondiciószáma az A kondiciószámának négyzete. Ez a veszély nem áll fenn a QR-felbontás esetén (amely viszont kétszer annyi művelettel jár, ha $m \gg n$), és A felbontását a Q ortogonális és az R felső háromszög mátrixra szolgálja : $A = Q * R$.

A QR-felbontás is alapművelete a MATLAB-nak : $[Q, R] = qr(A)$, és a `qr` MATLAB-függvény több, mint 10-féle módon hívható. Ezek közül az $Ax \cong b$ legkisebb négyzetek feladat tényleges megoldásának a szempontjából a legügyesebb a következő :

```
B=sparse(A);
[C,R]=qr(B,b);
x=R\C;
```

mert ez még akkor is jól működik, amikor a feltétel az A teljes rangjáról nem igaz (vö. a 26. oldalon a 27. sorhoz fűzött megjegyzéseket : az illesztésre használt függvényrendszer a mérési pontok szempontjából lehet alkalmatlan : az ott használt trigonometrikus függvényekre ilyen példa az a triviális eset, amikor $\sin k\pi t$ szerepel a rendszerben, és t_j mérési pontjaink éppen $0, 1, 2, \ldots$).

Az algoritmus hátterében a főoszlopválasztásos QR-felbontás áll, ld. a [32] 2.5. pontját.

Most adunk egy példát a QR-felbontás MATLAB-beli használatához. A (4.1) függvény rendszerünk és t_i megfigyelési helyeink legyenek

$$\{\varphi_1(t),\ldots,\varphi_5(t)\} = \{1,t,\sin(\pi*t),\exp(-2*t),\sin(2*\pi*t)\},$$
$$\{t_i\}_{i=1}^{15} = \{0,1,\ldots,14\},$$

tehát 15 pont a $[0,14]$ intervallumból.

A (szimulált) mérési adatokat így kapjuk :

$$b_i = 1-7*t_i+9*\sin(\pi*t_i)-4*\exp(-2*t)-11*\sin(2*\pi*t_i)+randn(1).$$

Ekkor az $A=(\varphi_j(t_i))$ kísérleti mátrix (`format short, format compact`-ben) :

```
A =
    1.0000         0         0    1.0000         0
    1.0000    1.0000    0.0000    0.1353   -0.0000
    1.0000    2.0000   -0.0000    0.0183   -0.0000
    1.0000    3.0000    0.0000    0.0025   -0.0000
    1.0000    4.0000   -0.0000    0.0003   -0.0000
    1.0000    5.0000    0.0000    0.0000   -0.0000
    1.0000    6.0000   -0.0000    0.0000   -0.0000
    1.0000    7.0000    0.0000    0.0000   -0.0000
    1.0000    8.0000   -0.0000    0.0000   -0.0000
    1.0000    9.0000    0.0000    0.0000   -0.0000
    1.0000   10.0000   -0.0000    0.0000   -0.0000
    1.0000   11.0000    0.0000    0.0000   -0.0000
    1.0000   12.0000   -0.0000    0.0000   -0.0000
    1.0000   13.0000   -0.0000    0.0000    0.0000
    1.0000   14.0000   -0.0000    0.0000   -0.0000
```

A 3. oszlop nullái előtti minuszjelek sejtetik, hogy ott kerekítési hibák miatt valójában nem nullák állnak. Így pl. az oszlop utolsó eleme `format long`-ban -0.000000000000002. Ezenkívül *rejtett számjegyei* is vannak a MATLAB-nak, pl. az utóbbi szám csak a $\sin(14*\pi)$ MATLAB-beli közelítése, és ha ennek utánanézünk, akkor azt kapjuk, hogy

```
>> (0.000000000000002-2.854944811937057e-16)+sin(14*pi)
ans =
     0
>> (0.000000000000002-2.85494481193706e-16)+sin(14*pi)
ans =
    -3.944304526105059e-31
```

A fenti A mátrix a `B=sparse(A);[C,R]=qr(B,b)` utasításokkal előállított QR-felbontásából először az R ritkamátrixot mutatjuk :

```
R =
   (1,1)      -3.8730
   (1,2)     -27.1109
   (2,2)      16.7332
   (1,3)       0.0000
   (2,3)      -0.0000
   (1,4)      -0.2986
   (2,4)      -0.4730
   (3,4)       0.8401
   (1,5)       0.0000
   (2,5)      -0.0000
   (3,5)       0.0000
```

majd ezután a megoldást :

```
Warning: Rank deficient, rank = 3, tol =  2.829653e-12.
> In qerr at 26
x =
   1.000713779476178
  -7.000051078724917
                   0
  -4.000631114273324
                   0
```

Ez tiszta beszéd : A jelenlegi mérési pontok mellett a sin-függvények feleslegesek a modellben, amely csak 3 bázisfüggvényt tartalmaz és az `x` eredményünk szerint

$$F(t) = 1.000713779476178 - 7.000051078724917t - 4.000631114273324\exp(-2t).$$

Érdekes a ,,`tol = 2.829653e-12`'' üzenet, amely nyilván a MATLAB `qr`-algoritmusának a beépített küszöbe a nulloszlopok megkülönböztetésére : ez a szám több, mint 10-szer nagyobb a `max(max(abs(A)))*m*n*eps` számnál (nálunk `m=15, n=5`, és `eps` a MATLAB relatív pontossági állandója). Ha viszont `max(max(abs(A)))*m*n*eps` helyett a `norm(A,'fro')*sqrt(m*n)*eps` számra gondolunk (az `A` *Frobenius–normá*jával), akkor ennél a `tol` tolerancia több, mint 45-szer nagyobb.

Megismételjük a feladatunkat abban az esetben, amikor csak a mérési pontokat választjuk másképpen, mégpedig úgy, hogy

$$\{t_i\}_{i=1}^{30} = \left\{ \frac{14 * (i-1)}{29} \right\}_{i=1}^{30},$$

tehát most 30 pontot a $[0, 14]$ intervallumból. Ekkor

```
A =
    1.0000         0         0    1.0000         0
    1.0000    0.4828    0.9985    0.3808    0.1081
    1.0000    0.9655    0.1081    0.1450   -0.2150
```

```
    1.0000    1.4483   -0.9868    0.0552    0.3193
    1.0000    1.9310   -0.2150    0.0210   -0.4199
    1.0000    2.4138    0.9635    0.0080    0.5156
    1.0000    2.8966    0.3193    0.0030   -0.6052
    1.0000    3.3793   -0.9290    0.0012    0.6877
    1.0000    3.8621   -0.4199    0.0004   -0.7622
    1.0000    4.3448    0.8835    0.0002    0.8277
    1.0000    4.8276    0.5156    0.0001   -0.8835
    1.0000    5.3103   -0.8277    0.0000    0.9290
    1.0000    5.7931   -0.6052    0.0000   -0.9635
    1.0000    6.2759    0.7622    0.0000    0.9868
    1.0000    6.7586    0.6877    0.0000   -0.9985
    1.0000    7.2414   -0.6877    0.0000    0.9985
    1.0000    7.7241   -0.7622    0.0000   -0.9868
    1.0000    8.2069    0.6052    0.0000    0.9635
    1.0000    8.6897    0.8277    0.0000   -0.9290
    1.0000    9.1724   -0.5156    0.0000    0.8835
    1.0000    9.6552   -0.8835    0.0000   -0.8277
    1.0000   10.1379    0.4199    0.0000    0.7622
    1.0000   10.6207    0.9290    0.0000   -0.6877
    1.0000   11.1034   -0.3193    0.0000    0.6052
    1.0000   11.5862   -0.9635    0.0000   -0.5156
    1.0000   12.0690    0.2150    0.0000    0.4199
    1.0000   12.5517    0.9868    0.0000   -0.3193
    1.0000   13.0345   -0.1081    0.0000    0.2150
    1.0000   13.5172   -0.9985    0.0000   -0.1081
    1.0000   14.0000   -0.0000    0.0000   -0.0000
```

azaz eltűntek a nulloszlopok, továbbá most

```
R =
   (1,1)     -5.477225575051661
   (1,2)    -38.340579025361620
   (2,2)     22.886526679430325
   (1,3)      0.000000000000000
   (2,3)     -0.322889106626797
   (3,3)      3.794172192299891
   (1,4)     -0.294848375488358
   (2,4)     -0.472995415319220
   (3,4)      0.050539794599631
   (4,4)      0.925404547815998
   (1,5)      0.000000000000002
   (2,5)     -0.016583074284161
   (3,5)     -0.001411241706849
   (4,5)      0.014990367882579
   (5,5)      3.807820675782260
```

vagyis A teljes rangú hiszen $i = 1, \ldots, 5$-re `R(i,i)`$\neq$ `0`, sőt `|R(i,i)|` jóval nagyobb, mint `max(max(abs(A)))*eps`, a MATLAB `eps` állandójával. Az x_j keresett együtthatók `format short`-ban, sorvektorként :

```
x =
  1.0000   -7.0000    9.0001   -3.9995  -10.9997
```

és részletesen, `format long`-ban a modellbe helyettesítve,

$$F(t) = 0.999278612142624 - 6.999921983825049t + 9.000048745094798\sin(\pi t) \\ -3.998880150713579\exp(-2t) - 10.999860811148945\sin(2\pi t).$$

Megjegyezzük, hogy a fent használt `[C,R]=qr(B,b)` utasításnak van egy másik verziója is :

```
[C,R,p]=qr(B,b,'vector');
x(p)=R\C;
```

amelynek használatákor R-ben a felesleges, csak kerekítési hulladékot tartalmazó oszlopok hátra lesznek rendezve (a p vektor a permutációról őrzi az információt). Ekkor érdemes az $R(1:3, 1:3)$ döntő részmátrixot `format long`-ban mutatni :

```
 -3.872983346207417 -27.110883423451920  -0.298611571330939
                  0  16.733200530681508  -0.472988300683720
                  0                   0   0.840101516036823
```

A p vektor viszont `[ 1 2 4 3 5 ]`.

A vázolt két út a legkisebb négyzetek problémája megoldásához a következőképpen értékelhető:

míg az n kicsi (mint gyakran mérési feladatok esetén), addig nem érdemes a QR-felbontást bevetni (és annál inkább nem a szinguláris felbontást (MATLAB-ban `[U,D,V]=svd(A)`)), bár ez a felbontás adja a teljes információt a legkisebb négyzetek feladatáról, ld. [32]), hanem a *Gauss-féle normálegyenlete*ket.

De ha rosszul kondicionált a feladat, akkor a lehetséges nagy pontossági veszteség miatt, vagy azért, mert nagy n mellett esetleg az A^TA nem is kiszámítható, inkább a hagyományos vagy a Q-nélküli QR-felbontást alkalmazzuk, ld. 26 oldal. Továbbá, a szinguláris felbontás gyakran túl sok számítási időbe kerül, és csak a QR-felbontás marad.

A *totális legkisebb négyzetek módszere* abból indul ki, ld. pl. [14], hogy hibák A-ban is lehetnek. Ilyen helyzettel gyakran lehet találkozni. Könyvünkben pl. a 4.4.3 részben közel vagyunk ehhez, hiszen ott a (4.41)-(4.43) parciális differenciálegyenletekből álló rendszert véges differenciák segítségével lineáris algebrai rendszerrel helyettesítettük (bár ott végülis nem az egyenlet y_h-nak nevezett megoldását keressük, hanem a rendszer egyes – nemlineárisan kiható – paramétereit). És az ottani rendszerben még a differenciáloperátor is csak közelítés lehet, pl. $d\frac{\partial^2 u}{\partial x^2}$ helyett valójában a $\frac{\partial}{\partial x}\left(d(x)\frac{\partial u}{\partial x}\right)$ kifejezés kellene a fizikai folyamat jobb leírásához.

Az ilyen problémák tipikusan diszkretizációval keletkeznek differenciálegyenletekből és általában még érzékenyebbek, mint a szokásos legkisebb négyzetek feladatai. Emiatt ekkor (4.6) minimum-megfogalmazása nem (4.5), hanem

$$\mathcal{L}(E,x,\gamma) := \|(A+E)x-b\|^2 + \|E\|_F^2 + \gamma(\|Lx\|_2^2 - \delta^2) \to \min_x !\,, \qquad (4.9)$$

ahol γ a Lagrange-multiplikátort és F a *Frobenius-normát* jelöli, tehát a mátrix elemei négyzetösszegének a gyökét. Továbbá, E az A mátrix hibáit adja, míg L az $\|Lx\|_2 \le \delta$ mellékfeltételből származik, amelynek jelenléte gyakran azt biztosítja, hogy a feladat egyáltalán megoldható legyen, és ebben L legtöbbször a számítógépes grafikából is ismert diszkrét Laplace–operátor, δ viszont hibaparaméter, amely néha fizikai meggondolásokból kapható, máskor értéke a feladat része.

Ld. [10], és [12], [13]. Ezek a művek a lineáris, igen rosszul kondicionált feladatokról szólnak, és a [13] alatt található programcsomag tartalmaz egy csomó hasznos m-fájlt a (4.9)-cel kapcsolatos feladatok megoldásához. A sokféle alkalmazás, amely ide sorolható, pl. a visszafelé megoldandó hővezetési egyenlet (ld. 4.4.4. pont), vagy bizonyos (ún. elsőfajú) integrálegyenletek megoldása, vagy torzított (digitális) ábrák feljavítása („deblurring").

Az utóbbi alkalmazásra gondolnak [22]-ben is, ahol (4.9) megoldását lineáris egyenletek egy sorozatára meg kétváltozós Newton–iterációkra vezetik vissza.

4.1.2. Gradiens módszerek

Amennyiben egy f valós függvényt kell pl. minimalizálni, és van okunk feltenni, hogy f folytonosan differenciálható, akkor a deriváltja bizony segítségünkre lehet, hiszen amikor $|\delta x| \to 0$, akkor

$$f(x+\delta x) = f(x) + (f'(x) + O(\omega(|\delta x|)))\delta x = f(x) + f'(x)\delta x + o(|\delta x|)$$

(ahol ω az f folytonossági modulusa) alapján várhatjuk, hogy egy adott x-nél $f(x+\delta x)$ kisebb lesz $f(x)$-nél elég kicsi $|\delta x|$ és $f'(x)\delta x < 0$ esetén (vagy ha pl. $\delta x = -t\cdot f'(x)$ és $t>0$ elég kicsi). Így indulhatunk egy $\delta x = \delta x_0$-val, és ha $f(x+\delta x_0) \ge f(x)$, akkor vesszük δx_0 helyett $\delta x_0/2$ és próbálkozunk újra stb.

Ezen eljárás általánosítása a fontosabb n-változós esetre, amikor $x = (x_1,\ldots,x_n)^T$, $\delta x = (\delta x_1,\ldots,\delta x_n)^T$ és $f : \mathbb{R}^n \to \mathbb{R}$, azon alapszik, hogy ekkor folytonosan differenciálható f-re igaz

$$\begin{aligned} f(x+\delta x) &= f(x) + \frac{\partial f}{\partial x_1}\delta x_1 + \ldots + \frac{\partial f}{\partial x_n}\delta x_n + o(||\delta x||) \\ &=: f(x) + (\operatorname{grad} f)^T(x)\cdot\delta x + o(||\delta x||), \end{aligned}$$

ahol $||\cdot||$ valamelyik $\mathbb{R}^n$-beli norma és $(\nabla f)(x) := (\frac{\partial f(x)}{\partial x_1},\ldots,\frac{\partial f(x)}{\partial x_n})^T = (\operatorname{grad} f)(x)$ az f gradiense – amely f növekedésének irányába mutat (tehát ekkor előnyös $\delta x = -t\cdot(\operatorname{grad} f)(x)$ elég kicsi $t>0$-ra). Ezért a gradiens módszer így néz ki : Egy x_0 pontból és $t>0$ értékből kiindulva számítjuk $\ell = 0,1,\ldots$-re

$$f(x_\ell),\ \ (\operatorname{grad} f)(x_\ell),\ \ x_{\ell+1} = x_\ell - t*(\operatorname{grad} f)(x_\ell), \quad \text{ha } f(x_{\ell+1}) < f(x_\ell),$$

másképpen t helyett próbálkozunk pl. $t/2$-vel stb.

Ezen módszer előnye egyszerűsége, és az első néhány lépése gyakran gyorsan csökkenti az f értékét, továbbá, az egyik nagyon gyakori mellékfeltétel esetén, hogy van egy korlátozó hipersík, mondjuk adott α egységvektorral és adott H_0 értékkel a mellékfeltétel (használva az $(\cdot,\cdot)$ euklideszi skalárszorzatot)

$$H_\alpha(x) := (\alpha, x) - H_0 = \sum_{i=1}^{n} \alpha_i x_i - H_0 \geq 0,$$

könnyű dolgoznunk : ha a következő $x_{\ell+1} = x_\ell - t * g_\ell,\ g_\ell := (\operatorname{grad} f)(x_\ell)$ lépésben az aktuális $t > 0$-val a hipersíkon túllépnénk, tehát $H_\alpha(x_\ell) > 0$, de $H_\alpha(x_{\ell+1}) < 0$ (és $f(x_\ell) > f(x_{\ell+1})$), akkor a

$$H_\alpha(x_\ell - s * g_\ell) = H_\alpha(x_\ell) - s(\alpha, g_\ell) = 0$$

feltételből meghatározhatjuk azt az $s \in (0, t)$ értéket, amivel éppen elérjük a hipersíkot, mert ekkor $t > 0$ és $0 < H_\alpha(x_\ell) - H_\alpha(x_{\ell+1}) = t(\alpha, g_\ell)$ mutatják, hogy $(\alpha, g_\ell) > 0$. A hipersík elérése után azonnal tehetünk egy lépést a hipersíkon belül : a g_ℓ gradiens helyett annak a hipersíkba való $g_\ell - (\alpha, g_\ell)\alpha$ projekciójával, vagyis a

$$x_{\ell+1} := x_\ell - s * g_\ell,\ \ x_{\ell+2} := x_{\ell+1} - t * (g_\ell - (\alpha, g_\ell)\alpha)$$

lépésekkel és megfelelő új $t > 0$-val.

Általában a gradiens módszer problémája viszont a t helyes választása. Továbbá, ha a minimalizálandó függvény domborzatának mély völgyei vannak (ami nem is ritka eset), akkor a programunk gyorsan lelassulhat és akár végtelen ciklusba kerülhet.

Egy további probléma maga a gradiens. Ha lehetséges, számítsuk ki analitikusan. De ha f bár analitikusan adott csak nincsen a MATLAB `symbolic toolbox`-a, vagy ha már f is egy összetett program eredménye, akkor a gradiens legtöbbször csak véges differenciákkal közelíthető, pl.

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x + h_i e_i) - f(x)}{h_i},$$

ahol $h_i > 0$ alkalmas szám és e_i az i-edik koordináta egységvektor. Amennyiben itt a differenciák h_i lépestávolsága túl nagy, akkor rossz az approximáció, ha meg túl kicsi, akkor a kerekítési hibák miatt az eredménynek semmi köze az igazi gradienshez.

A véges differenciák használatával általában n f-kiértékeléssel növekszik a munka, és tulajdonképpen azoknak a deriváltak nélkül dolgozó programoknak a közelébe kerültünk, amelyek ciklusban váltják a koordináta irányokat és 1-1 lépésben azt nézik, hogy balra vagy jobbra lépve csökken-e a függvényérték? (Megfelelő lépéshossz stratégia mellett ez egyébként elég hatékony is lehet.)

A t lépésparaméternek van értelmes választása, ha több információ áll rendelkezésünkre.

Vegyük elsőnek azt az egyszerű esetet, amikor f másodfokú polinom :

$$f(x) = ax^2 + bx + c, \quad \text{ahol } a > 0, \quad \text{a minimumhely tehát } x_* = -\frac{b}{2a}, \quad (4.10)$$

és x_0 a minimumhely közelítése, pl. $x_0 < x_*$. Ekkor $g_0 = 2ax_0 + b = (2ax_* + b) + 2a(x_0 - x_*) = 2a(x_0 - x_*) < 0$ és

$$x_1 = x_0 - tg_0 = x_0 - 2at(x_0 - x_*) = x_* \,,$$

amennyiben $t = 1/(2a)$. Ennek általánosítása az n-változós esetre, amikor x, δx oszlopvektorok és f kétszer folytonosan differenciálható vektorfüggvény, $H(x)$ a (szimmetrikus) *Hesse–mátrix*, $H(x) = (h_{ij}(x))_{i,j=1}^n = \left(\frac{\partial^2 f}{\partial x_i \partial x_j}(x)\right)_{i,j=1}^n$, akkor

$$f(x + \delta x) = f(x) + ((\operatorname{grad} f)(x), \delta x) + \frac{1}{2}(H(x)\delta x, \delta x) + o(||\delta x||^2) \qquad (4.11)$$

a megszakított Taylor–sor. A $(\operatorname{grad} f)(x_0) = g_0$ jelöléssel kapjuk

$$f(x_0 - tg_0) = f(x_0) - t||g_0||^2 + \frac{t^2}{2}(H(x_0)g_0, g_0) + o(t^2).$$

Álljon rendelkezésünkre az a további információ, hogy a Hesse–mátrix pozitív definit is minden x-re (ami a minimum létezését biztosítja), valamint hogy $\lambda(x)$ legnagyobb sajátértékének ismerünk egy $\overline{\lambda}$ felső becslését :

$$\lambda(x) \le \overline{\lambda}, \quad \text{minden } x \in \mathbb{R}^n\text{-re.}$$

Ekkor $t \le 1/(2\overline{\lambda})$ választással, ha $\overline{\lambda}$ elég nagy, közelebb kerülünk a minimumhelyhez és nem lépünk túl rajta.

Befejezésül közöljük a klasszikus *golden sectioning* módszer `goldensearch` programját (ez a módszer a keresési intervallumot mindig az aranymetszet szerint osztja fel, ld. [7], 53. o.), amelyben a gradiens módszernek minden $x_\ell \to x_{\ell+1} = x_\ell + t * v_\ell$ lépéséhez a v_ℓ keresési iránynak a $v_\ell := -(\operatorname{grad} f)(x_\ell)$ negatív gradiens aktuális értékét vesszük és a t paramétert optimálisan választjuk meg :

```
function [x,fval,t,funev]=goldensearch(fname,x,v,dt0,epst,epsf)
% vonalas kereses a g(t)=fname(x+t*v) nemnegativ f"uggvenyre aranymetszettel
%
% Hivasa :
% [x,fval,t,funev]=goldensearch(fname,x,v,dt0,epst,epsf)
% ahol bemeneten:
% fname - a minimalizalando f"uggveny neve (karakterlanc!)
% x - a kereses kezdeti erteke (f argumentuma, x: n-dim., f: m-dim.)
% v - keresesi irany
% dt0 - a lepestavolsag kezdeti erteke (pl. 1e-3, vagy 1e-4 stb.)
% epst - kilepesi feltettel: abs(dt)<epst*(1+abs(t))
```

```
% epsf - kilepesi feltettel: abs(f)<epsf*(1+abs(f0))
% a kimeneti parameterek viszont:
% x - az optimalis x+v*t
% fval - a hozzatartozo f-ertek
% t - az optimalis t
% funev - a f"uggveny kiertekeleseknek a szama

t0=0;
f(1)=norm(feval(fname,x+v*t0));f00=f(1);
disp(['f0=',num2str(f00,'%1.17g')])
t1=t0+dt0;
f(2)=norm(feval(fname,x+v*t1));
f0=max(f);

if f(2)>f(1)
   s(1)=t1;s(2)=t0;
   dt=-dt0;t=t0;
   ft=f(2);f(2)=f(1);f(1)=ft;
else
    dt=dt0;
    t=t1;
    s(1)=t0;s(2)=t1;
end
ga=(1+sqrt(5))*0.5;
for i=3:50
    dt=dt*ga;
    t=t+dt;
    f(i)=norm(feval(fname,x+v*t));
    s(i)=t;
    if f(i)>=f(i-1)
       break
    end
end

f3=f(i);f2=f(i-1);f1=f(i-2);
s3=s(i);s2=s(i-1);s1=s(i-2);
sg=sign(dt);

t=t-dt;
ga1=(sqrt(5)-1)*0.5;
dt=ga1^2*dt;

while abs(s3-s1)>=epst*(1+abs(s3)+abs(s1)) && min([f1,f2,f3])>=epsf*(1+f0)
   t=t+dt;
   ft=norm(feval(fname,x+v*t));
   i=i+1;
```

```
   if ft<=f2
       if (t-s2)*sg>0
          f1=f2;s1=s2;
       else
          f3=f2;s3=s2;
       end
       f2=ft;s2=t;
       dt=sg*abs(dt)*ga1;
   else
       if (t-s2)*sg>0
          f3=ft;s3=t;
       else
          f1=ft;s1=t;
       end
       t=s2;
       dt=-sg*abs(dt)*ga1;
   end
end
s=[s1,s2,s3];
f=[f1,f2,f3];
[f,in]=sort(f);
t=s(in(1));
x=x+v*t;
fval=f(1);
funev=i;
disp(['f=',num2str(fval,'%1.17g'),', rel. javitas: ',...
      num2str((f00-fval)/f00),', x=:'])
disp(num2str(reshape(x',4,4),'%1.15g')) % specialisan 16 x-komponensre
```

A végén szereplő `reshape` utasítás megfelelően megváltoztatható, ha nem pl. 16, hanem más a mindenkori `x`-vektor dimenziója : a cél egy kompakt és könnyen áttekinthető mező a képernyőn.

4.1.3. Newton-féle módszerek

A Newton–módszer egy valós f függvény gyökének meghatározására jól ismert. Egy x_0 közelítésből kiindulva lépünk így :

$$x_{\ell+1} = x_\ell - \frac{f(x_\ell)}{f'(x_\ell)}, \quad \ell = 0, \ldots, \tag{4.12}$$

és néhány feltétel teljesülése esetén, mint $f'(x) \neq 0$ (ld. pl. [32], 6.4.1), ez konvergál is a gyökhöz.

Amikor ezt a módszert f' gyökeinek meghatározására alkalmazzuk,

$$x_{\ell+1} = x_\ell - \frac{f'(x_\ell)}{f''(x_\ell)}, \quad \ell = 0, \ldots, \tag{4.13}$$

akkor a $-\frac{f'(x_\ell)}{f''(x_\ell)}$ változást az x_ℓ approximációban nevezzük *Newton–lépés*nek. A (4.13) módszerhez mindjárt hozzá kell tenni, hogy $f'(x) = 0$ csak szükséges feltétele a minimumnak, mert maximumot és inflexiós pontot is jellemez ez a feltétel, és mellékfeltétel jelenlétében (mint ha pl. pereme van a keresési tartománynak : $x \geq 0$) még $f'(x) \geq 0$ is lehet érvényes a minimumhelyen. Emiatt a talált pont környezetét kell vizsgálnunk, hogy tisztázzuk ezeket az eshetőségeket, pl. $f''(x)$ vizsgálatával.

A többváltozós esetben a Newton–módszerek alkalmazása minimumkeresésére is a (4.11) összefüggésen alapszik. Ehhez ott a $o(||\delta x||^2)$ tagot elhanyagoljuk és a többi tagot a $\delta x = (\delta x_1, \ldots, \delta x_n)^T$ vektor komponensei szerint deriváljuk. De először próbaképpen vizsgáljuk az

$$\begin{aligned} f(x_1, x_2) &= c + \left(\begin{pmatrix} g_1 \\ g_2 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right) + \frac{1}{2} \left(\begin{pmatrix} h_{11}x_1 + h_{12}x_2 \\ h_{21}x_1 + h_{22}x_2 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right) \qquad (4.14) \\ &= c + \sum_{i=1}^{2} g_i x_i + \frac{1}{2} \sum_{j=1}^{2} (h_{j1}x_1 + h_{j2}x_2)x_j \\ &= c + (g_1x_1 + g_2x_2) + \frac{1}{2} \left[(h_{11}x_1 + h_{12}x_2)x_1 + (h_{21}x_1 + h_{22}x_2)x_2 \right] \end{aligned}$$

másodfokú függvény (amelyben $c = f(0,0)$) deriválását

x_1 szerint :

$$\frac{\partial f}{\partial x_1} = 0 + g_1 + \frac{1}{2} \left[(2h_{11}x_1 + h_{12}x_2) + (h_{21}x_2) \right] = g_1 + \left[h_{11}x_1 + h_{12}x_2 \right],$$

H szimmetriája alapján ($h_{12} = h_{21}$);

és x_2 szerint :

$$\frac{\partial f}{\partial x_2} = 0 + g_2 + \frac{1}{2} \left[(h_{12}x_1) + (h_{21}x_1 + 2h_{22}x_2) \right] = g_2 + \left[h_{21}x_1 + h_{22}x_2 \right],$$

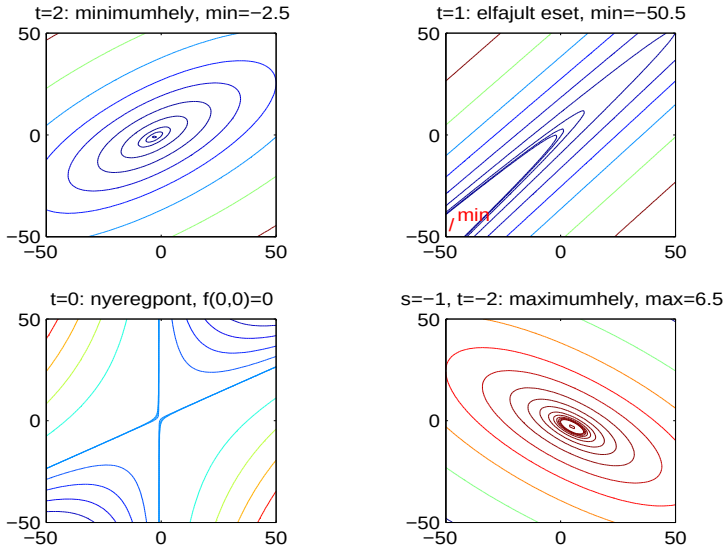
itt is felhasználva a szimmetriát. A két relációt érdemes vektoralakban összefoglalni :

$$\nabla f = g + Hx, \;\; x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \; g = \begin{pmatrix} g_1 \\ g_2 \end{pmatrix}.$$

Így $\nabla f = \operatorname{grad} f = 0 \in \mathbb{R}^n$, $n = 2$, ha $Hx = -g$. Ez a reláció tehát jellemzi a minimumhelyeket is, és általános n-re is fennáll. Ha pl. $n = 1$, akkor ld. (4.10). Emellett a (4.14)-ből való levezetés maximumhelyekre és nyeregpontokra is vezethet, és a másodfokú függvényeknek az így kapott minimuma, ill. maximuma globális szélsőértékei.

Szemléltetésként annak, hogy $\operatorname{grad} f = 0 \in \mathbb{R}^n$ nemcsak minimumhelyet határozhat meg, nézzük a következő paraméteres esetet, ahol $n = 2$ és $c = f(0,0) = 0$:

$$f(x) = (g, x) + \frac{1}{2}(H_{s,t}x, x), \;\; x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \; g = \begin{pmatrix} 2 \\ -1 \end{pmatrix}, \; H_{s,t} = \begin{pmatrix} s & -1 \\ -1 & t \end{pmatrix}. \quad (4.15)$$



4.1. ábra. *A (4.15) függvény viselkedése az origó környezetében (s=1 fent és a bal lenti részábrán)*

Ehhez ld. a 4.1 ábrát. Megemlítjük elsőnek, hogy a színskála `jet` volt, azaz a kék alacsony értéket mutat, a piros magas értéket. Ezzel a minimumhely, a nyeregpont és a maximumhely már jobban elképzelhető. De az elfajult eset további magyarázatot kíván.

Ott tehát $s = 1, t = 1$, vagyis $H_{s,t} = H_{1,1} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$ szinguláris (és a lent közölt MATLAB-szöveg teljesen értelmesen itt azt mondja, hogy $x = (Inf, Inf)^T$). Ha $x_* = (1,1)^T$, akkor $H_{1,1}x_* = \binom{0}{0}$, más szóval, a zéró sajátértékhez x_* a sajátvektor, amelynek $r*x_*$ irányában $f(r*x_*) = r(g, x_*) = r$ (mert esetünkben $(g, x_*) = 1$), és így, ha $-\infty$-hez tart r, akkor $f(r * x_*)$ is ezt teszi. Ezért ekkor a függvény minimuma az ábrázolási tartomány (vagy a mellékfeltétel) peremén fekszik. A minimumértékének és -helyének az $x_1 = -50$, $x_2 \geq -50$ és $x_2 = -50$, $x_1 \geq -50$ félegyeneseken utánaszámolva kiderül, hogy az érték az ábrán egzakt, a hely pedig $(-50, -49)$.

Végül, a nyeregpont esetén vannak maximumok és minimumok az ábrázolási tartomány peremén, de azok megint nem kapcsolódnak $\operatorname{grad} f = 0 \in \mathbb{R}^n$-hez : ez a reláció csak az origóban, a nyeregpontban áll fenn.

Most az `x` szélsőérték helyeket számíttatjuk ki MATLAB-bal a 4.1 ábra négy esetében :

```
>> s=1;g=[2 -1]';for t=2:-1:0,H=[s -1;-1 t],la=eig(H),x=-H\g,end
```

```
H =
     1    -1
    -1     2

la =
   0.381966011250105
   2.618033988749895

x =
    -3
    -1

H =
     1    -1
    -1     1

la =
     0
     2

Warning: Matrix is singular to working precision.

x =
   Inf
   Inf

H =
     1    -1
    -1     0

la =
  -0.618033988749895
   1.618033988749895

x =
    -1
     1

>> s=-1;g=[2 -1]';t=-2,H=[s -1;-1 t],la=eig(H),x=-H\g
t =
    -2

H =
    -1    -1
    -1    -2
```

```
la =
  -2.618033988749895
  -0.381966011250105

x =
     5
    -3
```

Eszerint az $x = (-3, -1)$-beli minimumhely globális, és értékét kiszámolva $s = 1$, $t = 2$ mellett kapjuk $f(-3, -1) = -2.5$, mint az első részábrán, míg $s = -1$, $t = -2$ mellett adja az $f(5, -3) = 6.5$ globális maximumot, ld. az utolsó részábrát.

Most, ha figyelembe vesszük, hogy általános f függvény esetén H és g függnek x-től, akkor (4.13) mintájára kapjuk, hogy

$$x_{\ell+1} = x_\ell - H^{-1}(x_\ell)g(x_\ell), \quad \ell = 0, \dots . \tag{4.16}$$

A $\delta x := -H^{-1}(x_\ell)g(x_\ell)$ vektort ekkor joggal nevezhetjük *Newton–lépés*nek, vö. (4.13)-mal (δx nyilván függ ℓ-től, de azt nem jelöljük).

(4.16)-ban H^{-1} persze nem az inverz mátrix kiszámítását, hanem a MATLAB „backslash" operátor használatát jelenti. (4.16) levezetéséhez újra (4.11)-hez fordulunk (ahol a $o(||\delta x||^2)$ tagot elhanyagoljuk). Ezt δx komponensei szerint deriválva kapjuk a $\nabla f = 0 \in \mathbb{R}^n$ szükséges minimumfeltételt akkor, amikor $\delta x = -H^{-1}g$, és visszahelyettesítve (4.11)-be (ott a $o(||\delta x||^2)$-et elhanyagoltuk), $x = x_\ell$-re látjuk, hogy $x_{\ell+1} := x_\ell + \delta x$-re éppen (4.16) adódik.

A Newton–módszer előnye, hogy bizonyos feltételek teljesülése mellett másodrendű konvergenciát biztosít (ld. [32], 6.4.1-2.). Hátránya az, hogy általában csak lokálisan, azaz elég jó kiindulási x_0 mellett konvergál. Ekkor segíthet egy $t > 0$ paraméter bevezetése (4.16)-ba :

$$x_{\ell+1} = x_\ell - t \cdot H^{-1}(x_\ell)g(x_\ell), \quad \ell = 0, \dots , \tag{4.17}$$

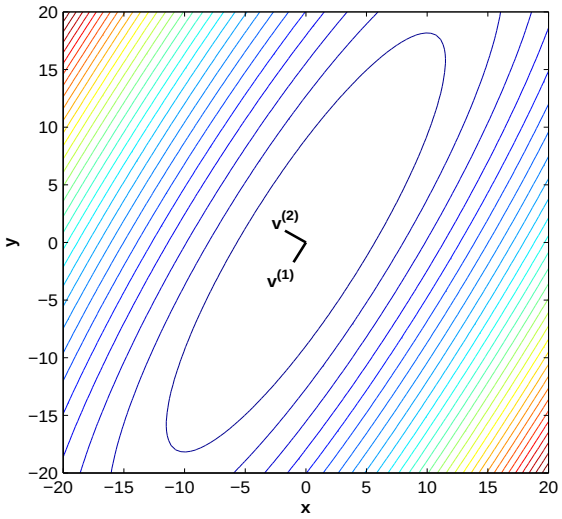
ahol a t paramétert elég kicsinek választjuk, vagy pedig minden ℓ-re egydimenziós minimum kereséssel határozzuk meg, mint a gradiens módszereknél. Még hatékonyabb lehet a Newton–módszer kombinációja a gradiens módszerrel :

$$x_{\ell+1} = x_\ell - s \cdot \operatorname{grad} f(x_\ell) - t \cdot H^{-1}(x_\ell)g(x_\ell), \quad \ell = 0, \dots ,$$

ahol most a két s, t paramétert kell lehetőleg jól meghatározni. Ezt teszik lokálisan egy megbízhatónak ítélt környezetben a „trust region"-módszerek, ld. [26], [7], amelyek a megbízható környezetet minden lépésben korrigálják.

A Newton–módszer nyilvánvaló másik hátránya, hogy itt, minimum keresés esetén, még a második deriváltak is kellenek, vö. a gradiens módszereknél az első deriváltakról mondott szöveggel, ld. 67. oldal.

A (4.16) Newton–módszer használatához tartozik az a tapasztalat, hogy (akár az egyváltozós esetben, ld. [32]) nem előnyös, túl nagy Newton–lépést megengedni, pl. azért, mert az f közelítése másodfokú függvénnyel várhatóan csak



4.2. ábra. *A* $10x^2 - 11xy + 4y^2$ *függvény szintvonalai, Hesse–mátrix sajátértékei* $\lambda_1 \approx 1.47004$, $\lambda_2 \approx 26.52996$, *sajátvektorai* $v^{(1)} \approx \binom{-0.51046}{-0.85990}$, $v^{(2)} \approx \binom{-0.85990}{0.51046}$

x_ℓ kicsi környezetében jó. Emiatt is javasolható a (4.17) csillapított Newton–módszer.

A gradiens módszereknél említett mély völgyekben a Newton–módszer egyébként képes egy nagyobb lépést megtenni a völgy lejtésének irányába (amire a következő pontban visszatérünk), de esetleg nem jut a völgy aljára. Emiatt is gyakran érdemes a két módszert kombinálni.

4.1.4. Optimalizálási feladatok érzékenységéről és a minimumhely hibabecsléséről

Ez a fontos téma arról szól, hogy az optimalizálandó változók közül melyik milyen mértékben hat ki a célfüggvény értékére (illetve a mellékfeltételekre). Ez bonyolult lehet, ha nem éppen a lineáris legkisebb négyzetekről van szó. Amikor a célfüggvény (mint (4.2) esetén) négyzetes polinom, az érzékenységet könnyű elmagyarázni, ld. a 4.2 ábrát.

A döntő a Hesse–mátrix teljes sajátérték feladata.

Mivel minimumot keresünk, kizárhatjuk azt az esetet, amikor minden sajátérték negatív (tehát amikor a Hesse–mátrix negatív definít). Ha negatív és pozitív sajátértékek is vannak, akkor nyeregponton vagyunk és a negatív sajátértékekhez tartozó sajátvektorok irányaiba kell menni a minimumkereséshez. Mindkét eset nem fordul elő a legkisebb négyzetek feladatánál (de általános függvény minimalizálasa közben nem ritkák ezek az esetek).

Amennyiben a Hesse–mátrix szemidefinít, akkor a nulla sajátértékhez tartozó sajátvektorok irányaiban a célfüggvény nem változik, a minimumfeladatnak végtelen sok megoldása van. Végül, ha a Hesse–mátrix pozitív definít, akkor a minimum feladat érzékenysége értelmezhető :

a legkisebb sajátértékhez tartozó sajátvektor irányában a célfüggvény a leglassabban változik, tehát abban az irányban a legkevésbé érzékeny (ld. $v^{(1)}$-et a 4.2 ábrán, közben ebben az irányban nehéz lehet a minimumot pontosan meghatározni). Megfordítva, a legnagyobb sajátértékhez tartozó sajátvektor irányában a célfüggvény a leggyorsabban változik (ld. $v^{(2)}$-t a 4.2 ábrán), tehát abban az irányban a leginkább érzékeny a célfüggvény (közben ebben az irányban könnyű a minimumot meghatározni, pontosabban : a völgy aljára jutni).

Az ilyen helyzet akkor problémás, amikor a Hesse–*mátrix rosszul kondicionált*, tehát az abszolútértékben legkisebb és legnagyobb sajátértékek több nagyságrendben különböznek. A gradiens módszereknek ehhez is köze van (anélkül, hogy a Hesse–mátrixot használnák fel), mert a rosszul kondicionált Hesse–mátrix azt jelenti, hogy a függvény domborzatán ekkor azok az utolsó két pontban említett mély völgyek vannak (amiről a 4.2 ábránk csak enyhe benyomást ad), és ekkor a gradiens leggyakrabban közel merőleges a völgy aljának irányára, azaz az ábrán $v^{(2)}$ irányába mutat.

Viszont a Newton–módszer éppen a Hesse–mátrix inverzét használja, amelynek sajátértékei az eredeti mátrix reciprokai, így felerősíti nagy mértékben a gradiensben lévő, a völgy irányába (az ábrán $v^{(1)}$ irányába) mutató komponenseket.

A MATLAB-nak nincs kész programja érzékenységi vizsgálatra a fenti értelemben, ld. pl. 2012-ből a [24] hivatkozást (az ott említett `fmincon` program az `optimization toolbox` elsőnek javasolt programja). (De ld. a [9] internetoldalt, ha differenciálegyenletek rendszereinek a paraméterei szerinti érzékenységéről van szó).

Az irodalom alapján elmondható, hogy az érzékenységi vizsgálat megfelelően kezelhető a legkisebb négyzetekre (ahol A^TA a Hesse–mátrix) és a lineáris programozásra, ld. pl. [28].

Az általános nemlineáris minimalizálás esetén érdemes nehezségek feltűnésekor – a nem kielégítő, itt most az x_0 oszlopvektorral jelölt végső „megoldás" alapján – a neki megfelelő $H(x_0)$ lokális Hesse–mátrixot pontosan vagy közelítőleg kiszámítani. Nagyméretű feladatok esetén ez problémás lehet, hiszen $n(n-1)/2$ másodrendű deriváltról van szó, majd $H(x_0)$ teljes sajátérték feladatát kell megoldani.

Ezután a sajátvektorokat érdemes lehet 5 csoportra osztani:

– negatív sajátértékekhez tartozó sajátvektorokra;

– azokra, amelyek a legkisebb pozitív sajátértékekhez tartoznak (ezek a leginkább érzékeny irányok, közben ezekre a célfüggvény a legkevésbé érzékeny);

– a közepes pozitív sajátértékekhez tartozókra;

– azokra, amelyek a legnagyobb pozitív sajátértékekhez tartoznak (ezek a legkevésbé érzékeny irányok, közben ezekre a célfüggvény a legérzékenyebb);

– végül, a nulla sajátértékekhez tartozó sajátvektorokra.

A pozitív sajátértékek három csoportra való beosztása azt a célt szolgálja, hogy a hozzátartozó alterekben futó minimalizálási feladatok kondicionáltsága lényegesen javuljon.

Amikor minimalizálási feladatunk nemlineáris legkisebb négyzetek alakjában meg van fogalmazva :

$$f := \frac{1}{2}\sum_{i=1}^{m} f_i(x)^2, \quad f_i : \mathbb{R}^n \to \mathbb{R}, \tag{4.18}$$

akkor az érzékenységi vizsgálatnak egy lehetséges programjához fontos tudni, hogy itt is a Hesse–mátrix szimmetrikus, de két részből áll (ld. [32]) :

$$H(x) = G + f''(x)\cdot f(x),$$
$$G(x) = (f'(x))^T f'(x), \qquad f''(x)\cdot f(x) = (\phi_{k\ell})_{k,\ell=1}^{n}, \;\; \phi_{k\ell} := \sum_{i=1}^{m} \frac{\partial^2 f_i}{\partial x_k \partial x_\ell} f_i(x).$$

Ez magyarázza a következő programot :

```
function [V,S,neg,pos1,pos2,pos3,nul,g,delta]=sensib(fname,x,epsi)
% Az "fname" f"uggveny erzekenysege x-t"ol
%
% Hivasa :
% [V,S,neg,pos1,pos2,pos3,nul,g,delta]=sensib(fname,x,epsi)
% ahol
% fname - a f"uggveny neve, sztring
% x - az aktualis argumentum
% epsi - a veges differenciak lepesk"oze
% V - a n"ovekv"o sajatertekek szerint rendezett sajatvektorok matrixa
% S - a n"ovekv"oen rendezett sajatertekek
% neg - a negativ s.e.-k indexei
% nul - a nulla s.e.-k indexei
% pos1 - a kis pozitiv s.e.-k indexei
% pos2 - a k"ozeps"o pozitiv s.e.-k indexei
% pos3 - a nagy pozitiv s.e.-k indexei
% g - fname gradiense x-ben
% delta - a Newton-lepes, |delta| a minimumhely hibaindikatora
% (ha a Hesse-matrix poz.def.), vagy -1

f0=feval(fname,x);
m=length(f0);
n=length(x);

G=zeros(m,n);
F=zeros(n,n);
H=zeros(n,n);
g=zeros(n,1);
```

```
for i=1:n
    dx=epsi*max(epsi,abs(x(i)));
    xp=x;
    xm=x;
    xp(i)=xp(i)+dx;
    fp=feval(fname,xp);
    xm(i)=xm(i)-dx;
    fm=feval(fname,xm);
    G(:,i)=0.5*(fp-fm)/dx; % G i-edik oszlopa
    g(i)=G(:,i)'*f0;
    Hii=(fp-2*f0+fm)/(dx^2);
    F(i,i)=Hii'*f0;        % f"f f"oatloja
    for j=i+1:n            % f"f fels"oharomsz"ogenek kiszamitasa
        dxj=epsi*max(epsi,abs(x(j)));
        xm=x;
        xm(j)=xm(j)+dxj;
        fj=feval(fname,xm);
        xm=xp;
        xm(j)=xm(j)+dxj;
        fm=feval(fname,xm);
        Hij=(fm-fj-fp+f0)/(dx*dxj);
        H(i,j)=Hij'*f0;
    end
end
F=F+H+H';
G=G'*G;
disp(['A Hesse-matrix 2 reszenek normai: ||G||=',num2str(norm(G),' %0.5g'),...
      ', ||F||=',num2str(norm(F),' %0.5g')])
[V,D]=eig(G+F);            % G+F a Hessematrix
[S,in]=sort(diag(D));
V=V(:,in);
epsnm=10*eps*max(abs(S));  % nullanak tekintett sajatertekek k"usz"obe
neg=find(S<=-epsnm);
disp(['A negativ sajatertekek: ',num2str(S(neg)',' %0.5g')])
nul=find(abs(S)<epsnm);
disp(['A nulla k"or"uli sajatertekek: ',num2str(S(nul)',' %0.5g')])
pos=find(S>=epsnm);        % azzal szamolunk, hogy mindig van poz. sajatertek!
disp(['A pozitiv sajatertekek: ',num2str(S(pos)',' %0.5g')])
np=length(neg)+length(nul);
pos1=find(S(pos)<100*S(pos(1)))+np;
pos2=find(S(pos)>=100*S(pos(1)) & S(pos)<=0.01*S(n))+np;
pos3=find(S(pos)>0.01*S(n))+np;
posdef=find(S>0);

if length(pos1)==length(pos3)
   if pos1==pos3 & length(neg)>0  % ritka eset
```

```
       if sum(S(pos1))>1000*abs(sum(S(neg)))
          pos1=[];
       else
          pos3=[];
       end
   end
end
if length(posdef)==n       % pozitiv definit eset
    delta=(G+F)\g;
else
    delta=-1;
end

plot(0:n+1,zeros(n+2,1),'-k',1:n,S,'*r')
set(gca,'fontweight','bold')
axis([0 n+1 20*S(1) 1.05*S(n)])
title('''Erz''ekenys''egi saj''at''ert''ekek','fontweight','bold')
xlabel('saj''at''ert''ek indexe','fontweight','bold')
```

Ezen program $G(:,i)$-t kiszámító sorához azt jegyezzük meg, hogy a derivált ezen közelítése a szimmetrikus differenciahányadossal másodrendű, ld. [33], 11.4.1. pontban (11.34), és kell is, hogy az legyen : máskülönben még abban az alapvető esetben, amikor egy parabolikus felület minimumpontjában vagyunk, nem kapunk zérus gradienst!

Az `epsnm` küszöbbértékben az `eps` a gépi epszilon.

A `find` parancsnak (ld. a magyar MATLAB-könyvet) több alkalmazását is tartalmazza a program, és a `plot` előtti `if length(pos)==n ... end` sorok a szűkebb értelemben vett érzékenységi vizsgálatnak felelnek meg, ld. lent (4.20).

Ha a sajátvektorok (mint oszlopvektorok) mátrixa V, akkor feltehetjük, hogy $V = [V_1, \ldots, V_5]$ (hiszen V-vel együtt VP is a sajátvektorok mátrixa, ahol P egy megfelelő permutációs mátrix), és ahol V_i a fenti i-edik csoport sajátvektorainak felel meg, pl. V_1 a negatív sajátértékekhez tartozó sajátvektorok (rész)mátrixának, stb., V_5 a nulla(-nak tekintett) sajátértékekhez tartozó sajátvektorok (rész)mátrixának.

Mivel Hesse–mátrixunk szimmetrikus, érvényes

$$V^{-1} = V^T = (V_1, \ldots, V_5)^T .$$

A kapott információ kétféle hasznosítását látjuk, amikor továbbra is x_0-val jelöljük azt a pontot, amely a minimalizálás numerikus eredménye és amelyben az érzékenységi vizsgálatot folytattuk le a $H(x_0)$ Hesse–mátrix alapján a `sensib` programmal :

1) Nem kívánjuk az egész minimalizálási feladatot úgy átfogalmazni, hogy az optimalizálás csak valamelyik i-edik, $i = 1, 2, 3, 4$, altérben fusson, mert az gyakran bonyolult lenne (holott ezzel a döntéssel lehet kapcsolatos némely hatékonysági veszteség is).

V_5-tel nem érdemes foglalkozni, de ha ez az altér többször ugyanazokkal a sajátvektorokkal fordul elő, akkor ez azt jelenti, hogy az ismeretlenek ezzel adott kombinációitól nem függ az `f` és az optimalizálás!

Az $x_0 \in \mathbb{R}^n$ helyett egyszerűen vesszük a minimalizálandó függvény argumentumaként $V_i(V_i^T x_0)$. Ekkor a program kap egy altérbeli n-dimenziós kezdeti vektort, éppen $x = V_i(V_i^T x_0)$-t, munka közben kiléphet az altérből, de amikor valamelyik $z \in \mathbb{R}^n$ pontban függvényértéket kér, azt először $x = V_i(V_i^T z)$ segítségével visszavetítjük az altérbe. Ez egyszerűbben programozható. Képletben:

$$\text{az eredeti } x \;\rightarrow\; F(x) \;\text{ behelyettesítés változik } y = V_i(V_i^T x) \;\rightarrow\; F(y)\text{-ra.}$$

Az i indexet nyilván ciklikusan változtatjuk, először a negatív sajátékeknek megfelelő V_1-gyel definiált alteret véve, ott lefut egy teljes minimalizálás, majd folytatjuk V_2-vel és V_3-mal, végül V_4-gyel. Ezután lehetne elölről kezdeni, de ne felejtsük, hogy a $H(x_0)$ Hesse–mátrix csak lokális információt hordoz.

2. A V_i-vel lehet a $v = V_i(V_i^T x_0)$ irányt definiálni és pl. a `goldensearch` programmal (ld. 68. oldalt) a v-irányú egydimenziós minimum keresést indítani :

$$||f(x_0 + t \cdot V_i(V_i^T x_0))|| \;\rightarrow\; \min_t !\,, \quad i = 1, \ldots, 4, \tag{4.19}$$

persze mindig az x_0-át felújítva az éppen optimális $x_0 + t \cdot V_i(V_i^T x_0)$ eredménnyel, hiszen az optimalizálás a főfeladatunk.

Az ilyen teljes „V-ciklus" után (beleértve, hogy ha valamelyik i. altér üres, akkor arra a konkrét i-re nem teszünk semmit) nem érdemes egy azonnali újabb érzékenységi vizsgálat, hanem inkább az optimalizáló programot indítsuk be újra.

A fenti ciklusokra egy alkalmazást adunk a 4.4.2. pontban, kezdve a 119. oldallal, ahol viszont a V_i altereket, miután a g gradiens is rendelkezésünkre áll, még a $V_0 := \text{span}\{g\}$ altérrel kiegészítve, a $V_0, \ldots, V_4$ altér-sorozattal dolgozunk.

Amennyiben a Hesse–mátrix pozitív definít, akkor négyzetes f esetén az x_ℓ közelítő minimumhely távolsága az x_* pontos minimumhelytől a Newton–lépésből adódik : $||\delta x||$. Tehát ekkor az abszolútértékek $|\delta x|$ vektora a **minimumhely hibabecslése** koordinátánként, vö. (4.16)-tal. Amikor f általános függvény, akkor δx képlete

$$\delta x = H^{-1}(x_\ell) g(x_\ell), \tag{4.20}$$

és mivel a Newton–lépés általában sem visz a minimumba és x_ℓ-től is függ, $|\delta x|$ legtöbbször nem hibabecslés, hanem csak **hibaindikátor** az adott helyen, azaz a hiba lehetséges nagyságrendjére mutathat rá.

Teszteljük ezt a `sensib` eljárást a 4.2 ábrán vizsgált függvényen, azaz amikor (4.18)-ben $f_1 = \sqrt{2}(3x - by)$, $f_2 = \sqrt{2}(x - dy)$, ahol $d = (11 - 3\sqrt{39})/20$, $b = (33 + \sqrt{39})/20$ és közelítésünk a minimumhelyhez $[0.1,\, 0.2]'$, továbbá `epsi=1e-4`. Ekkor a sajatértékek $\approx 0.735018,\, 13.264982$, és a (4.20) szerinti $\delta x \approx (0.10000000171,\, 0.20000000139)$.

Mivel csak hibaindikátorról van szó, azért érdemes, a (4.17) szerinti csillapításra gondolni, ill. méginkább (ha a $H^{-1}(x_\ell) > 0$ feltétel fennáll) a V-ciklus legelső helyére egy δx-irányú vonalkeresést bekapcsolni.

Ld. a 4.4.3. pont végét is, ahol (4.20)-at egy paraméter meghatározási feladatra alkalmazzuk.

4.2. Hátizsák feladatok

Most tekintsünk egy ún. „hátizsák" feladatot, ld. több példát, algoritmust és az elméletet [40]-ben.

Munkahelyünk és számlánk van, de olyan helyen kellene fizetnünk, ahol csak készpénzt fogadnak el, esetünkben összesen kb. 40 eFt-ról van szó. A pénzautomata kezelése előtt gondolkodjunk röviden : ha pontosan 40000-et gépelünk be, akkor lehet, hogy két 20 eFt-os bankjegyet kapunk, ami esetleg problémát okoz majd. Inkább vegyünk ki 39 eFt-ot (míg címletek után érdeklődő automáták és Eurók még nincsenek), akkor biztosan apróbb pénzünk is lesz.

Ezzel saját hátizsák problémánkat oldottuk meg. Most jön a pénzautomata (aránylag egyszerű) problémája :

Van pénze $i = 1, \ldots, n$ címletben, amelynek értéke $c_i > 0$, abból adhat $x_i \geq 0$ darabot (ezeket a számokat keresi) annak érdekében, hogy kifizesse a kívánt $s > 0$ összeget. Mind ezek a számok egészek. Emellett azt a stratégiát követi az automata, hogy minél kevesebb bankó kerüljön kifizetésre, vagyis egy **egészszámú optimalizálási feladat**tal áll szemben :

$$\begin{aligned} \sum_{i=1}^{n} x_i &= \min_{x_i,\, i=1,\ldots,n} ! \\ \sum_{i=1}^{n} x_i c_i &= s, \\ x_i &\geq 0, \;\; i = 1, \ldots, n. \end{aligned} \tag{4.21}$$

Ez tehát egy tipikus hátizsák feladat, amit még bonyolíthatunk azzal az n mellékfetétellel, hogy b_i nemnegatív felső korlátok is vannak az x_i-kre (például kevés 2000 forintosa van a minket remélhetőleg kiszolgáló automátának) :

$$b_i \geq x_i, \;\; i = 1, \ldots, n. \tag{4.22}$$

Az `intlinprog` függvény, amely az R2014a MATLAB-verzió kidolgozásakor az *optimalizálási toolbox* része lett, oldja meg a hátizsák és ennél általánosabb feladatokat. Alapesetben hívása a következő :

```
[x,fval] = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub);
```

Ezt implementáljuk a következő m-fájllal, amelyből a paraméterek jelentése is látható :

```
function [x,fval]=mljegesz(s,ub)
% ATM feladata
%
% Hivasa :
```

```
% [x,fval]=mljegesz(s,ub)
% ahol
% s - a kivant "osszeg
% ub - a fels"o korlatok vektora
% x - a megoldasi vektor (a kifizetend"o cimletek darabszama)
% fval - az egyenl"osegi feltetelre kapott ertek

if s<=0
    error('Adj be pozitiv kifizetend"o "osszeget!')
end

f=[20 10 5 2 1]; % cimletek eFt
n=length(f);
intcon=1:n;      % Minden keresett valtozo egeszszam
A=[];b=[];       % Nincsenek egyenl"otlensegi mellekfeltetelek
Aeq=f;beq=s;     % Az egyenl"osegi mellekfeltetel Aeq*x=beq
lb=zeros(n,1);   % Az also korlatok

[x,fval] = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub);

disp(['A megoldas:  ',num2str(x','%3.0f '),' darab'])
disp(['A cimletek: ',int2str(f),' eFt'])
disp(['A mellekfeltetelek teljes"ulnek: norma(Aeq*x-beq)=',...
      num2str(norm(Aeq*x-beq))])
x=x';
```

Most mutatunk egy egyszerű példát az m-fájl alkalmazására :

```
>> x=mljegesz(39,inf(5,1));

LP:                Optimal objective value is 39.000000.

Cut Generation:    Applied 1 gomory cut.

Lower bound is 39.000000.

Branch and Bound:

   nodes     total   num int        integer       relative

explored  time (s)  solution           fval        gap (%)

       2      0.00         1   0.390000E+02   0.000000E+00

Optimal solution found.

Intlinprog stopped because the objective value is within a gap
```

```
tolerance of the optimal value;

options.TolGapAbs = 0 (the default value). The intcon variables
are integer within tolerance,

options.TolInteger = 1e-05 (the default value).

A megoldas:  1   1   1   2   0 darab
A cimletek: 20  10   5   2   1 eFt
A mellekfeltetelek teljes"ulnek: ||Aeq*x-beq||=0
```

Az Olvasónak világos, hogy esetünkben a megoldás egyértelmű és gyorsan kiszámítható (4.21) miatt. De (4.22) bonyolítja a helyzetet. Lássunk néhány példát erre :

```
>> x=mljegesz(39,[inf inf 0 0 1])

No feasible solution found.

Intlinprog stopped because no point satisfies the constraints.
```

Világos, hogy nem tudja kifizetni a 39 eFt-ot, mert csak 20 eFt-os és 10 eFt-os és egyetlen 1000 forintos bankója van.

```
>> x=mljegesz(39,[inf inf 1 0 inf])

LP:                Optimal objective value is 39.000000.

Cut Generation:    Applied 1 gomory cut.

Lower bound is 39.000000.

Relative gap is 0.00%.

Optimal solution found.

Intlinprog stopped at the root node because the objective value is
within a gap tolerance of the optimal value;

options.TolGapAbs = 0 (the default value). The intcon variables are
integer within tolerance,

options.TolInteger = 1e-05 (the default value).

A megoldas:  1   1   1   0   4 darab
A cimletek: 20  10   5   2   1 eFt
A mellekfeltetelek teljes"ulnek: ||Aeq*x-beq||=7.1054e-15
```

Az `intlinprog` program által kinyomtatott információból látjuk, hogy a vizsgált, aránylag egyszerű feladatunk megoldása során már néhány alapvető fogalom és algoritmus szükségeltetett.

Lássuk, hogy ezek mit jelentenek.

LP : *linearis programozás*, ld. pl. [27]. Ez a feladat nincs messze (4.21)-től, de a korlátozás egészszámokra hiányzik. Ehelyett valós számokkal foglalkozik. Gyakran így fogalmazzák meg :

$$\begin{aligned} c^T x = \sum_{j=1}^{n} c_j x_j &= \max_{x_j,\, j=1,\ldots,n} ! \\ (Ax)_i = \sum_{j=1}^{n} a_{ij} x_j &\leq b_i, \;\; i = 1, \ldots, m \\ x_j &\geq 0, \;\; j = 1, \ldots, n. \end{aligned} \tag{4.23}$$

Itt is az x vektor komponensei keresettek, adottak a b és c vektorok komponensei és az A mátrix elemei. A $c^T x$ szám gyakran valamilyen termelési vagy szervezési folyamat nyereségét adja, ez a *célfüggvény*.

Elsőnek Kantorovics [17] foglalkozott ilyen feladattal, Dantzig javasolta a Gauss eliminációval kapcsolatban álló *szimplex algoritmus*t (amely híres, mert a legtöbb alkalmazott lineáris programozási feladaton gyorsan adja a megoldást és csak egyes speciális problémákon nem képes erre), Karmarkar-ra [18] mennek vissza a modern **belső pont módszerek**, egy alapvető mű az egész témakörről itt [5];

Cut Generation : vágások előállítása, ld. [40], p. 105;

gomory cut : *Gomory-vágás*, ld. [40], p. 106. Ez a módszer az egészszámú programozási feladatot egy sorozat, kisebb dimenziójú, olyan lineáris programozási problémákra vezeti vissza (hipersíkok segítségével), amelyekben már nincs az egész számokra való korlátozás (Gomory javasolta), így bevethető a szimplex vagy a belsőpontos algoritmusok. Más ilyen vágásokat is használ az `intlinprog`, pl. a MIR-t („mixed integer rounding") vagy a „flow cover"-t;

Branch and Bound : korlátozás és szétválasztás, ld. [40], p. 169 : egy, a Gomory-vágáshoz hasonló, de általánosabb elv optimalizálási feladatok megoldására, nemcsak egészszámú feladatokra. Utóbbiaknál azt jelenti, hogy az egészszámú korlátozás nélküli feladat megoldása pontos alsó korlátot ad a célfüggvény optimális értékére, ezenkívül szétbontja részfeladatokra arra ügyelve, hogy a részfeladatok mellékfeltételeinek uniója szűkebb legyen, mint az eredeti feladaté.

A jelenlegi pontban tárgyalt feladatok köre a pénautomátáénál jóval fajsúlyosabb kérdéseket is tartalmaz. Pl. indulna egy expedíció (akár egy hadsereg) és felszerelések egész sorát kellene vinnie, amelyekből mindegyiknek bizonyos fontossága (értéke) van : ez c_i, közben súlya a_i is van, $i = 1, \ldots, n$. Ugyanakkor van egy felső korlát az összsúlyra : b. Előáll tehát a döntési kényszer : adott

tárgyat vigyük ($x_i = 1$) vagy pedig nem ($x_i = 0$)? Matematikailag :

$$\begin{aligned} \sum_{i=1}^{n} c_i x_i &= \max_{x_i,\, i=1,\ldots,n} ! \\ \sum_{i=1}^{n} x_i a_i &\leq b, \\ x_i &\in \{0,1\},\ \ i = 1,\ldots,n. \end{aligned}$$

Világos, hogy a maximumot úgy tudjuk keresni, hogy $-\sum_{i=1}^{n} c_i x_i$ minimumát próbáljuk meghatározni.

4.3. Az utazó ügynök

Ez egy hasonló, de gyakran lényegesen nehezebb feladat, mint az előző, 4.2-beli.

A 7 legnagyobb magyar város (forrásunk : hu.wikipedia.org/wiki, 2013-as kimutatás) mindegyikébe egyszer kell utaznia és kiindulási pontjához visszatérnie egy ügynöknek, mégpedig kocsival és úgy, hogy az össztávolság (km-ben, forrásunk az egyes városok közti távolságokra : maps.google.com) minimális legyen. Az alábbi távolságok természetesen kerekített számok, hiszen a városok nem pontok, és pl. Budapesten belül is lehet 20 kilométert egy irányban kocsikázni.

város/km	B.pest	Debr.	Misk.	Szeged	Pécs	Győr	Ny.háza
Budapest	0	230	183	172	201	121	234
Debrecen	230	0	113	218	445	352	50
Miskolc	183	113	0	343	397	304	116
Szeged	172	218	343	0	189	286	395
Pécs	201	445	397	189	0	301	448
Győr	121	352	304	286	301	0	355
Nyíregyháza	234	50	116	395	448	355	0

Mindenki megérti, hogy a feladat mintha igen egyszerű lenne : elég az íróasztalon végig próbálni minden lehetőséget és a megfelelő kilométereket összeadni. De sajnos van $n!$ lehetőség és közismert, hogy ez gyorsan növekvő szám (közelítőleg úgy, mint $\left(\frac{n}{e}\right)^n$, vagyis gyorsabban nő n minden polinomjánál), amely $n = 7$-re még csak 5040, de $n = 10$-re már 3628800. Az egyre gyorsabb számítógépekben bízva, elsőként felmerülhet a kérdés, hogyan állítsuk elő egyáltalán azt az $n!$ lehetőséget, a városok számoknak megfeleltetett összes permutációit?

Az utóbbi kérdésre a válasz a következő m-fájl. Itt lesz nyilvánvaló egy másik probléma : az m-fájl $n * n!$ tárhelynél többet foglal le! Ez kérdésessé teszi, hogy n=12-re még le tud futni a program, hiszen ekkor már $n * n! \approx 5.748e + 9$. (A szerző gépén már az $n = 11$ eset sem futott le, de $n = 10$ igen.) Ehhez a témához ld. [19]-et is.

```
function A=faculty(n)
% Kiszamitja az (1,...,n) szamok n! permutacioit es A-ba rakja.
% Hivasa :
% A=faculty(n)
% ahol n vilagos, >=2
% A - olyan [nxn!]-meret"u matrix, amelynek oszlopaiban allnak a
% permutaciok

A=[1 2;2 1]; % a matrix n=2-re
if n==2
   return
end
nf=2; % a permutaciok szama az elejen
for m=3:n;
    m1=m-1;m2=m1-1;
    B=A;
    for i=1:m1
        B=[B,A]; % m-szer rakjuk egymas melle az el"oz"o matrixot
    end;
    ek=m*ones(1,nf); % az m szam meg nem szerepelt
    A(1:m,1:nf)=[ek;B(:,1:nf)]; % az 1. B-resz ele rakjuk ek-t
    for i=1:m2 % 1-1 B-resz sorai k"oze rakjuk ek-t
        A(:,nf*i+1:nf*i+nf)=[B(1:i,nf*i+1:nf*i+nf);ek;...
                             B(i+1:m1,nf*i+1:nf*i+nf)];
    end;
    nfa=nf-1;
    nf=nf*m; % a faktorialis kiszamitasa
    A(:,nf-nfa:nf)=[B(:,nf-nfa:nf);ek]; % az utolso B-resz m"oge
end                                     % is rakjuk ek-t
```

Ezután már nem nehéz a dolgunk : a permutációk birtokában kiszámoljuk a hozzátartozó össztávolságot (ld. lent (4.27)). Így $n = 7$-re egy pillanat alatt megvan a megoldás:

1 - 6 - 5 - 4 - 2 - 7 - 3 - 1, 1178 km, vagyis Budapest - Győr - Pécs - Szeged - Debrecen - Nyíregyháza - Miskolc - Budapest a legjobb választás - amit egy tapasztalt logisztikus szakember várhatóan számolás nélkül is mondana. Mivel itt egy körutazásról van szó, Budapest helyett az adott körön bármely más városból is indulhatna az ügynök.

Feladat : Vegyük hozzá a következő 3 várost, Kecskemétet, Székesfehérvárat és Szombathelyt, és erre oldjuk meg a feladatot!

A fent vázolt megoldási út előnye, hogy kis n-re gyorsan adja a pontos megoldást, ami nyilvánvalóan nem az egyetlen, mert ha fordított sorrendben vesszük a városokat, akkor ugyanaz a minimális össztávolság jön ki, de más permutáció tartozik hozzá.

Annak érdekében, hogy a MATLAB `intlinprog` programot be tudjuk vetni és nagyobb n-re, ha esetleg nem is optimális, de elég jó megoldást kapjunk,

teszünk egy első próbálkozást arra, hogy megfogalmazzuk a feladatot a (4.21)-(4.22) hátizsák feladat mintájára, a fenti táblázat számait az $A = (a_{(i,j)}) \in \mathbb{R}^{n\times n}$ mátrixba rendezve (példánkban persze eddig $n \leq 7$) :

$$\sum_{i=1}^{n!} x_i \sum_{j=1}^{n} a_{(\nu_j^{(i)},\nu_{j+1}^{(i)})} = \min_{x_i,\, i=1,\ldots,n!}! \tag{4.24}$$

$$\sum_{i=1}^{n!} x_i = 1, \tag{4.25}$$

$$x_i \in \{0,1\}. \tag{4.26}$$

Itt x_i döntési változó, $\nu_{(i)} = (\nu_1^{(i)}, \ldots, \nu_n^{(i)})$ az i-edik permutáció, $\nu_{n+1}^{(i)} := \nu_1^{(i)}$,

$$f_i := \sum_{j=1}^{n} a_{(\nu_j^{(i)},\nu_{j+1}^{(i)})} \tag{4.27}$$

az össztávolság adott $\nu_{(i)}$ esetén.

Most emlékezzünk a MATLAB `intlinprog` programjának hívására, ld. a 81. oldalt :

```
[x,fval] = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub);
```

Ebből ugyan látjuk, hogy a (4.24)-(4.26) feladat megoldására alkalmazható a program, de mivel az $n!$ szerepel benne (pl. `f` hossza ekkora, és az összes permutációt ismernünk kell), nem lesz hasznunkra a program, ha n nagy. De azért nézzük meg, mit nyújt $n = 7$ esetén : az eredmény a pontos megoldás (a sorrend ciklikus permutációja nem számít),

x=[7 3 1 6 5 4 2] a sorrend, az optimális össztávolság : fval=1178 km.

Az $n = 10$ eseten már egy picit „gondolkodik" a program, kb. 7.8 másodpercet, de megint adja a pontos megoldást, míg az `f` vektor és minimális komponensének kiszámítása csak kb. 3.0 másodperc.

A fentiek után világos, hogy nagyobb n-re ($n >$ 10-re) szükségünk van a feladatnak olyan megfogalmazására, amely nem tartalmazza közvetlenül az $n!$ számot és az összes permutációt. Ez pl. a következőképpen lehetséges :

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_{(i,j)} x_{ij} = \min_{\{x_{ij},i,j=1,\ldots,n\}}! \tag{4.28}$$

$$x_{ij} \in \{0,1\},\ \ i,j = 1,\ldots,n, \tag{4.29}$$

$$\sum_{i=1,i\neq j}^{n} x_{ij} = 1,\ \ j = 1,\ldots,n, \tag{4.30}$$

$$\sum_{j=1,j\neq i}^{n} x_{ij} = 1,\ \ i = 1,\ldots,n, \tag{4.31}$$

Itt az x_{ij} számok keresettek, az optimális utat adják meg, mert definíciójuk, hogy

$$x_{ij} = \begin{cases} 1, & \text{ha az optimális út vezet az } i\text{-edikből a } j \neq i\text{-edik városba,} \\ 0, & \text{máskülönben.} \end{cases} \tag{4.32}$$

Így (4.30) azt jelenti, hogy minden j-edik városba az út vezet valamelyik (egyetlen) i-edik városból, és hasonlóan (4.31) azt jelenti, hogy minden i-edik városból az út folytatódik valamelyik (egyetlen) j-edik városba.

(Zárójelben megjegyezzük, hogy bevezetve az $n \times n$ permutációs mátrixokat és $\mathcal{P}_n$-nel jelölve az összes ilyen mátrix halmazát – ha pl. $n = 3$, akkor az összes permutáció a

$$B = \begin{pmatrix} 3 & 3 & 1 & 2 & 1 & 2 \\ 1 & 2 & 3 & 3 & 2 & 1 \\ 2 & 1 & 2 & 1 & 3 & 3 \end{pmatrix}$$

mátrix oszlopaiban található és a $(3,1,2)$ permutációnak megfelel a $P_1 := \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$, a $(2,1,3)$-nak a $P_{n!} = P_6 := \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, és $\mathcal{P}_n = \{P_1, \ldots, P_{n!}\}$ –, akkor a (4.28)-(4.31) feladat egyetlen sorral helyettesíthető:

$$trace\{AP^T\} = \min_{P \in \mathcal{P}_n} !$$

A kapcsolat (4.28)-(4.32)-vel ekkor az, hogy $P = (x_{ji})$, és a minimumkövetelményből a transzponálás elhagyható, mert legfeljebb fordított sorrendben keressük fel az n várost.)

A (4.28)-(4.32) feltételeket az `intlinprog` program paramétereivé alakítva, nagyon gyorsan választ kapunk, és aszerint az optimális megoldási út hossza 899 km – ami gyanús, hiszen tudjuk, hogy az optimális össztávolság 1178 km. Kiderül : nem zártuk ki annak a lehetőségét, hogy az optimális út több kisebb körből áll, esetünkben : 1 - 6 - 1, 2 - 3 - 7 - 2, 4 - 5 - 4.

Ez ellen segítenek a következő egyenlőtlenségek (ld. `http://en.wikipedia.org/wiki/Travelling_salesman_problem`, de részletesebben érdemes a [4] cikket elolvasni vagy [1] magyar fordítását, ahol a téma történelmét bemutatja a szerző és 145 fontosabb irodalmi hivatkozást is ad), amelyekben az u_i-k mesterséges, ún. „slack” változók :

$$u_i \geq 0, \quad i = 1, \ldots, n-1, \tag{4.33}$$

$$u_i - u_j + nx_{ij} \leq n-1, \quad 1 \leq i \neq j \leq n-1. \tag{4.34}$$

Most minden készen áll az `intlinprog` sikeres bevetésére. Paramétereinek készítésénél a következőket fogjuk figyelembe venni :

1. A (4.28)-(4.33) megfogalmazás egyszerűsödik, ha $x_{ii} = 0$-t hozzátesszük, $i = 1, \ldots, n$;

2. A felső és alsó korlátok egybe is tudnak esni (pl. éppen x_{ii}-nél);

3. a mátrixokat (kivéve a távolsági A mátrixot) érdemes ritkamátrixként megadni.

Kezdjük a (4.28) bal oldalával. Az A és $X = (x_{ij})$ mátrixokat `reshape`-pel f és x sorvektorokba rendezve (utóbbit csak gondolatokban, és ennek n^2 komponenseihez még az n slack-változót kell betervezni), kapjuk a szükséges összeget, pl. $A \to f$ esetén :

```
nn=n*n;
f=reshape(A,1,nn);
```

A (4.29) feltételt majd az alsó és felső korlátok megadásánál biztosítjuk, és azzal, hogy az `intcom` paraméternél az első n^2 változót egésznek jelentjük be.

Most nézzük meg (4.30) átírását. Itt jól jön a *Kronecker–szorzat*, pl. :

```
n=3;kron(eye(n),ones(1,n))

ans =

     1     1     1     0     0     0     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     0     0     0     1     1     1
```

Ez általában már $n^2 * n$ elemet fog adni, és még az n slack-változók nincsenek figyelembe véve. Ezért a fentiek helyett azt írjuk majd az általános esetben, hogy

```
A1=[kron(speye(n),ones(1,n)),sparse(n,n)];
```

emlékezve arra, hogy a ritka, $n \times n$-es nullmátrix (tréfásan lehetne mondani : a „létező semmi") éppen `sparse(n,n)`, és ahol `speye(n)` a ritka egységmátrix. Ekkor csak n^2 tárhelyet foglal el $A1$. A fenti $n = 3$ esetben szorzata az $(x_{11}, x_{2,1}, x_{31}, \ldots, x_{33})^T$ oszlopvektorral (vagy általában az $(x_{11}, \ldots, x_{n1}, \ldots, x_{nn})^T$ vektorral) láthatóan

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, \ldots, n, \quad (x_{jj} = 0),$$

tehát (4.30).

Hasonlóan kapjuk (4.31)-et :

```
n=3;kron(ones(1,n),eye(n))

ans =

     1     0     0     1     0     0     1     0     0
     0     1     0     0     1     0     0     1     0
     0     0     1     0     0     1     0     0     1
```

amely mátrixnak a szorzata az $(x_{11}, x_{2,1}, x_{31}, \ldots, x_{33})^T$ vektorral $n = 3$-ra

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, \ldots, n, \quad (x_{ii} = 0),$$

tehát (4.31). Általános n-re a ritkamátrixú megfogalmazás :

```
A2=[kron(ones(1,n),speye(n)),sparse(n,n)];
Aeq=[A1;A2];
```

ami összesen $2 * n^2$ tárhelyet jelenti. Most fordulunk a (4.34) – x-ben lineáris – egyenlőtlenségek mátrixához ((4.33) biztosítása az alsó és felső korlátok dolga). Ehhez az x vektorban elsőnek rendezzük az x_{ij} elemeket (és az $i = j$ esetekre is írunk – akár üres – egyenlőtlenségeket), majd az u_i-ket :

```
nn=n*n;n1=n-1;
  . . .

n11=n1-1;
C=[speye(n11),-ones(n11,1)];C(n1,n1)=0;
B=C;
for i=1:n1-2
    C=[[C(2:n1,2:n1),sparse(n11,1)];sparse(1,n1)];
    B=[C;B];
end
B=[zeros(n1,n1);B];
C=rot90(B,2); % Ehelyett flipup(fliplr(B)) lehetseges, de lassubb
B=B+C; % A slack valtozokkal foglalkozo matrix

n1n1=n1*n1;
C=[n*speye(n1n1),zeros(n1n1,1)];
for i=1:n1
    C=[C(:,1:(n-i)*n1),zeros(n1n1,1),C(:,(n-i)*n1+1:n1n1+i)];
end
A=[C,zeros(n1n1,n1),B]; % Az egyenl"otlensegi feltetelek matrixa
```

Ennek megértéséhez talán elég $n = 4$ esetén a 2. i-ciklus utáni (és `A=[C,...` előtti) `B,C` mátrixokat telt mátrixként megmutatni:

```
B =
     0     0     0
    -1     1     0
    -1     0     1
     1    -1     0
     0     0     0
     0    -1     1
     1     0    -1
```

```
     0     1    -1
     0     0     0

C =
     4     0     0     0     0     0     0     0     0     0     0     0     0
     0     4     0     0     0     0     0     0     0     0     0     0     0
     0     0     4     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     4     0     0     0     0     0     0     0     0
     0     0     0     0     0     4     0     0     0     0     0     0     0
     0     0     0     0     0     0     4     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     4     0     0     0     0
     0     0     0     0     0     0     0     0     0     4     0     0     0
     0     0     0     0     0     0     0     0     0     0     4     0     0
```

Feladat : a fenti program 2. i-ciklusát írjuk át úgy, hogy ott a `diag` utasítást használjuk (ld. a magyar MATLAB-könyvet).

Ezután közöljük a teljes programot, a távolsági mátrixot most inkább `T`-vel jelölve :

```
function [x,fval]=mljutazo(T)
% Az utazo "ugyn"ok feladata
%
% Hivasa :
% [x,fval]=mljutazo(T)
% ahol
% T - a varosok k"ozti tavolsagok matrixa
% x - a megoldasi permutacio
% fval - a minimalis "osszhossz

n=size(T,1);
if n<3
    error('Legyen T merete legalabb 3x3!')
end
if T~=T'
   error('T nem szimetrikus!')
end
tic
   nn=n*n;n1=n-1;
   f=[reshape(T,1,nn),zeros(1,n1)];
   On=sparse(n,n1);
   A1=[kron(speye(n),ones(1,n)),On];
   A2=[kron(ones(1,n),speye(n)),On];
   Aeq=[A1;A2]; % Az egyenl"osegi mellekfeltetel matrixa

   n11=n1-1;
   C=[speye(n11),-ones(n11,1)];C(n1,n1)=0;
   B=C;
```

```
for i=1:n1-2
    C=[[C(2:n1,2:n1),sparse(n11,1)];sparse(1,n1)];
    B=[C;B];
end
B=[zeros(n1,n1);B];
C=rot90(B,2); % Ehelyett flipup(fliplr(B)) lehetseges, de lassubb
B=B+C;

n1n1=n1*n1;
C=[n*speye(n1n1),zeros(n1n1,1)];
for i=1:n1
    C=[C(:,1:(n-i)*n1),zeros(n1n1,1),C(:,(n-i)*n1+1:n1n1+i)];
end
A=[C,zeros(n1n1,n1),B]; % Az egyenl"otlensegi feltetelek matrixa

intcon=1:n1n1; % Az x vektor komonensei mind egeszszamuak
               % - csak a slack valtozok esetleg nem
b=n1*ones(n1n1,1); % Az egyenl"otlensegi mellekfeltetelek jobboldala
beq=ones(n+n,1);   % Az egyenl"osegi mellekfeltetel Aeq*x=beq
lb=zeros(1,nn+n1); % Az also korlatok
ub=ones(n,n);ub=[reshape(ub-eye(n),1,nn),inf(1,n1)]; % Fels"o korlatok

[x,fval,exitf] = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub);
if exitf<1
    error('No solution found')
end
xx=reshape(xx(1:nn),n,n);
disp('Az optimalis sorrend :')
for j=1:n
    pj=find(xx(:,j)>0);
    disp([int2str(pj),' ---> ',int2str(j)])
end
toc

disp(['A tavolsag: ',int2str(fval),' km'])
disp(['A mellekfeltetelek teljes"ulnek: norma(Aeq*x-beq,inf)=',...
      num2str(norm(Aeq*x-beq,inf))])
```

Most mutatjuk az `intlinprog` eredményeit néhány n-re.

$n = 4$:

```
LP:                Optimal objective value is 628.000000.

Cut Generation:    Applied 1 flow cover cut, and 1 mir cut.
                   Lower bound is 628.000000.
```

```
Branch and Bound:

   nodes     total   num int        integer       relative

explored  time (s)  solution           fval        gap (%)

       2      0.01         1   0.686000E+03   0.843336E+01

       6      0.01         1   0.686000E+03   0.000000E+00

Optimal solution found.

Intlinprog stopped because the objective value is within a gap
tolerance of the optimal value; options.TolGapAbs = 0 (the de-
fault value). The intcon variables are integer within toler-
ance, options.TolInteger = 1e-05 (the default value).

Az optimalis sorrend :
3 ---> 1
4 ---> 2
2 ---> 3
1 ---> 4

Elapsed time is 0.026530 seconds.

A tavolsag: 686 km

A mellekfeltetelek teljes"ulnek: norma(Aeq*x-beq)=6.2804e-16
```

Az optimális sorrend tehát 3 - 1 - 4 - 2 - 3.

$n = 7$ (itt elhagyjuk a program közléseinek egy részét) :

```
Optimal solution found.

Intlinprog stopped because the objective value is within a gap
tolerance of the optimal value;

options.TolGapAbs = 0 (the default value). The intcon variables
are integer within tolerance,

options.TolInteger = 1e-05 (the default value).

Az optimalis sorrend :
6 ---> 1
7 ---> 2
1 ---> 3
```

```
2 ---> 4
4 ---> 5
5 ---> 6
3 ---> 7
Elapsed time is 0.033191 seconds.
A tavolsag: 1178 km
A mellekfeltetelek teljes"ulnek: norma(Aeq*x-beq)=7.6919e-16
```

Tehát ez a már ismert optimális sorrend megfordítottja $n = 7$-re.

$n = 10$: Itt érdekes a numerikus eredmény, 10×10-es mátrixként :

```
xx =

     0   0   1   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   1
     0   0   0   0   0   0   0   0   0   0
     0   1   0   0   0   0   0   0   1   0
     1   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   1   0   0
     0   1   0   0   0   0   0   0   0   0
     0   0   0 0.2 0.8   0   0   0 0.8 0.2
     1   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   1   0   0   0   0
```

A jobb olvashatóság érdekében itt már kerekítettük az értékeket (pl. 0.2 eredetileg 0.199999999999999 volt stb.). Ehhez az `intlinprog` az alábbiakat közölte :

```
LP:                Optimal objective value is 1078.200000.

Cut Generation:    Applied 2 gomory cuts, 6 mir cuts,

                   and 1 flow cover cut.

                   Lower bound is 1091.833333.

Heuristics:        Found 3 solutions using rss.

                   Upper bound is 1379.000000.

                   Relative gap is 20.80%.

Cut Generation:    Applied 1 gomory cut,

                   and 1 flow cover cut.

                   Lower bound is 1091.833333.
```

```
                Relative gap is 20.80%.

Branch and Bound:

   nodes     total   num int        integer       relative

explored  time (s)  solution           fval        gap (%)

      83      0.03         4   0.130300E+04   0.121845E+02

     278      0.05         4   0.130300E+04   0.683118E+01

     462      0.06         4   0.130300E+04   0.322653E+01

     658      0.08         4   0.130300E+04   0.327464E+00

     682      0.08         4   0.130300E+04   0.000000E+00

Optimal solution found.

Intlinprog stopped because the objective value is within a gap
tolerance of the optimal value;

options.TolGapAbs = 0 (the default value). The intcon variables
are integer within tolerance,
options.TolInteger = 1e-05 (the default value).
```

A MATLAB arra hivatkozik, hogy nem egész x_{ij}-k előfordulhatnak (ekkor a j-ciklusban még hiba is keletkezhet, mert pj vektor lesz), és hogy az `options.TolInteger = 1e-05` volt. Ilyenkor javasolja a kapott megoldás kerekítését és a mellékfeltételek ellenőrzését. Ezen tanács követése vezetett a program alábbi végéhez (az utolsó j-ciklus előtt) :

```
xx=round(x(1:nn));
r=max(abs(xx-x(1:nn)));
if r>1e-8
   disp(['Kerekites maximuma: ',num2str(r)])
   disp(['pozitiv (A*x-b)-komponensek: ',int2str(find(A*x-b>0)')])
   disp('max(A*x-b) (ez legyen nempozitiv), a kerekites utan :')
else
   disp('max(A*x-b) (ez legyen nempozitiv) :')
end
disp(num2str(max(A*x-b)))

disp([char(10),'Az optimalis sorrend :'])
xx=reshape(xx,n,n);
```

(míg a j-ciklus, a `toc` és az azutáni sorok változatlanok). Ekkor az output :

```
Kerekites maximuma: 0.2
pozitiv (A*x-b)-komponensek: 16  19  35  57  65
max(A*x-b) (ez legyen nempozitiv), a kerekites utan :
3.5527e-15   % Tehat az ertek pozitiv, de kerekitesi nagysagrend"u!

Az optimalis sorrend :
9 ---> 1                        10 ---> 6
7 ---> 2                         3 ---> 7
1 ---> 3                         2 ---> 8
8 ---> 4                         6 ---> 9
4 ---> 5                         5 ---> 10

Elapsed time is 0.104397 seconds.
A tavolsag: 1303 km
A mellekfeltetelek teljes"ulnek: norma(Aeq*x-beq,inf)=6.6613e-16
```

Tehát az optimális sorrend (ami a képernyőn valójában egyetlen oszlopba van rendezve) : 9 - 1 - 3 - 7 - 2 - 8 - 4 - 5 - 10 - 6 - 9.

$n = 13$-ra csak a végső eredményeket mutatjuk (amiután még hozzávettük Szolnokot, Tatabányát és Kaposvárt, és $n = 11, 12$-re nem tapasztaltuk, hogy szükséges a kerekítés):

```
Optimal solution found.

Intlinprog stopped because the objective value is within a gap
tolerance of the optimal value;
options.TolGapRel = 0.0001 (the default value). The intcon variables
are integer within tolerance,
options.TolInteger = 1e-05 (the default value).

max(A*x-b) (ez legyen nempozitiv) :
1.4211e-14

Az optimalis sorrend :
3 ---> 1              4 ---> 8
11 ---> 2             1 ---> 9
7 ---> 3              6 ---> 10
5 ---> 4              8 ---> 11
13 ---> 5             9 ---> 12
12 ---> 6            10 ---> 13
2 ---> 7

Elapsed time is 0.749424 seconds.
A tavolsag: 1348 km
A mellekfeltetelek teljes"ulnek: norma(Aeq*x-beq,inf)=6.6613e-16
```

Az útvonal tehát Miskolc - Budapest - Székesfehérvár - Tatabánya - Győr -

Szombathely - Kaposvár - Pécs - Szeged - Kecskemét - Szolnok - Debrecen - Nyíregyháza - Miskolc.

Az eredményt így is lehet kommentálni : amelyik feladatot a pont elején bemutatott primitív módszernek semmi kilátása sem volt megoldani ($n = 13$), azt az `intlinprog` kb. négyszer gyorsabban oldotta meg, mint amit az egyszerű módszer még meg tudott oldani ($n = 10$), holott $n!$-ban mérve az utóbbi feladat több, mint ezerszer kisebb, mint az előző.

Befejezésül megemlítjük, hogy a tárgyalt feladat nemcsak utazások kapcsán fontos, hanem pl. optimális technológiák meghatározása esetén is. Gondoljunk csak arra, hogy egy lemeznek (így egy chip-nek) sok-sok pontja mindegyikében egy a lemez felett mozgó gépnek néhány egyenlő műveletet el kell végeznie.

4.4. Modellezés differenciálegyenletekkel

Természetes vagy gazdasági folyamatok elemzésénél, új termékek fejlesztése során gyakran támaszkodnak olyan fizikai modellekre, amelyek közönséges vagy parciális differenciálegyenletekből állnak. Ezekben tipikusan lesz néhány közvetlenül nem vagy csak nehezen, pontatlanul mérhető paraméter (amelyek konstansok vagy akár függvények is lehetnek). Ekkor a remény az, hogy közvetve, a folyamat megfigyeléséből, mégis csak meghatározhatóak a paraméterek. Matematikai szóhasználattal ezek *inverz feladatok*, mert nem a megoldás keresett, hanem egy vagy néhány, a modellt lezáró paraméter.

Erre adunk konkrét példákat a következő alpontokban. Az első alpontban pl. egy mozgó egyszerű, de csak távolból látható szerkezet két tömegét és két további számot kellene találni csupán az ismert mozgás alapján, a másodikban megfigyelhetjük egy kocsi útját, de nem tudjuk, hogy a vezető hogyan forgatta a kormányt.

Bevezetésként a témakörben adódó numerikus problémákba, ha megvan az `optimization toolbox`, írjuk be a MATLAB parancssorába azt, hogy `optimtool`, valamint a kereső ablakába azt, hogy `optimizing ode`.

4.4.1. Négy modellparaméter meghatározása 4 közönséges differenciálegyenletből álló rendszerben mérési adatokból

Elsőnek tekintjük a következőkben a „csúszós” inga egyenleteit, ld. [31]. Ez egy egyszerű, de nemlineáris mechanikai rendszer, amely abból áll, hogy egy vízszintes sínen csúszhat egy m_1 ([kg]-ban mérve) tömegű test az $x_1 = x(t)$, $y_1 = 0$ útjával ([m]-ben) és $\dot{x}(t)$ sebességével ([m/sec]). Erre a testre fel van akasztva egy merev ℓ [m] hosszú ingán egy másik m_2 tömegű test, amelyre hat a Föld tömegvonzása (tudjuk : $g = 9.81$ [m/sec^2] a gyorsulás) és amelynek helyzete $(x_2(t), y_2(t))$, [m]-ben. Ezeket az $x_2(t), y_2(t)$ koordinátákat ki tudjuk számítani, ha ismerjük az első testből lefelé induló félegyenesből az ingáig vett $\alpha(t)$ (radiánban, dimenziótlan) szöget.

A rendszer mozgását meghatározzák az $\alpha(t)$, $\dot{\alpha}(t)$, $x(t)$, $\dot{x}(t)$ függvények, amelyekre [31]-ben a következő egyenleteket vezetik le (2 másodrendű egyenlet, amelyek ekvivalensek négy elsőrendű egyenlettel).

Ezekhez az egyenletekhez mi még hozzátettünk súrlódási tagokat, az egyszerűség kedvéért egyetlen r súrlódási együtthatóval, az x-egyenletben r-rel, az α-egyenletben csak $\frac{r}{2}$ részével, ahol az $\frac{1}{2}$ szorzó is inverz feladat tárgya lehetne, elegendő mérési adatok esetén :

$$\ell(m_1 + m_2 \sin^2\alpha)\ddot{\alpha} + r\frac{\ell}{2}\dot{\alpha} = -\sin\alpha[g(m_1+m_2) + m_2\cos\alpha\ell\dot{\alpha}^2], \quad (4.35)$$

$$(m_1 + m_2\sin^2\alpha)\ddot{x} + r\dot{x} = m_2\sin\alpha[\ell\dot{\alpha}^2 + g\cos\alpha]. \quad (4.36)$$

Itt érdemes az egyes tagok dimenzióit vizsgálni: az $\ell(m_1 + m_2\sin^2\alpha)$ [m·kg], az $\ddot{\alpha}$ [1/sec^2], a kettő szorzata tehát [m·kg/sec^2], ami egy erő, ahogyan ez megfelel a Newton-féle mechanikának. Ugyanannyi kell, hogy legyen az $r\frac{\ell}{2}\dot{\alpha}$ súrlódási tag dimenziója, amiből következik r dimenziója, mint [kg/sec], és meggyőződhetünk arról, hogy az egyenlet jobboldalának a dimenziója is ennyi.

A második egyenlet bal oldalán ugyan hiányzik az ℓ az [m] dimenziójával, de van $\ddot{x}$, amelynek [m/sec^2] a dimenziója.

A fentiek szerinti *súrlódás modellezése* nem a talaj okozta súrlódásnak felel meg, hanem – ami feladatunkban kézenfekvő – egy másik közegben történő súrlódásnak, mint a levegőben, vagy a vízben. Megjegyezzük, hogy erre a célra a lineáris $r\frac{\ell}{2}\dot{\alpha}$, $r\dot{x}$ tagok mellett nemlineáris tagok is használatosak, ld. [33], 10.1.1. Viszont a talajjal való súrlódás esetén $r\dot{x}$ helyett a nemlineáris $mg \cdot r\,\mathrm{sign}(\dot{x})$ lenne megfelelő, ld. [31], ahol az utóbbi r dimenzió nélküli együttható.

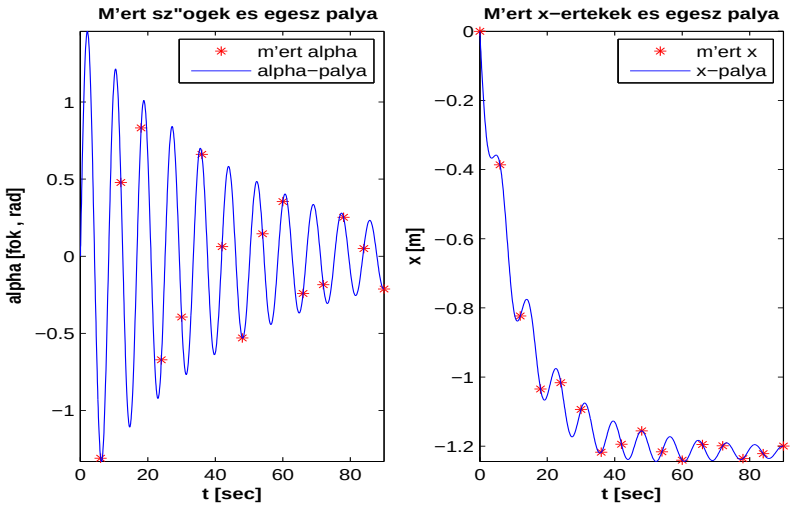
Az alábbi főprogramban a (4.35)-(4.36) egyenleteket szokásos módon elsőrendű rendszerré alakítottuk ($z = [\alpha, \dot{\alpha}, x, \dot{x}]$ a rendszer megoldási vektora), és ezzel oldjuk meg a *direkt feladat*ot : a $p = (\ell, m1, m2, r)$ paraméterekből és a kezdeti értékekből meghatározni az $\alpha(t)$, $x(t)$ függvényeket :

```
function pendulum(ell,m1,m2,r)
% a csuszosan felf"uggesztett pendulum dif.-egyenletek megoldasa,
% abrazolasa
%
% Hivasa :
% pendulum(ell,m1,m2,r)
% ahol
% ell - a pendulum hossza
% m1 - az els"o t"omeg
% m2 - a 2. t"omeg
% r - a surlodasi egy"utthato

fok=180/pi;
epsr=1e-5;
a=0;b=90;
ab=linspace(a,b,b/6+1);
```

```
z0=[0,0.02,0,-0.2];
opts=odeset('reltol',epsr,'abstol',epsr*1e-3);
[t,z]=ode45(@(t,z) pendulumf(t,z,ell,m1,m2,r),ab,z0',opts);
[tt,zz]=ode45(@(t,z) pendulumf(t,z,ell,m1,m2,r),[a b],z0',opts);

subplot(1,2,1)
plot(t,fok*z(:,1),'r*',tt,fok*zz(:,1),'b-')
axis tight
title('  M''ert sz"ogek es egesz palya','fontweight','bold')
xlabel('t [sec]','fontweight','bold')
ylabel('alpha [fok , rad]','fontweight','bold')
legend('m''ert alpha','alpha-palya','location','northeast')

subplot(1,2,2)
plot(t,z(:,3),'r*',tt,zz(:,3),'b-')
axis tight
title('M''ert x-ertekek es egesz palya','fontweight','bold')
xlabel('t [sec]','fontweight','bold')
ylabel('x [m]','fontweight','bold')
legend('m''ert x','x-palya','location','northeast')
```

A jobboldali függvények kiszámítása :

```
function f=pendulumf(t,z,ell,m1,m2,r)
% a csuszosan felf"uggesztett inga dif.-egyenletek jobboldala
%
% Hivasa :
% f=pendulumf(t,z,ell,m1,m2,r)
% ahol
% t - az id"o
% z - 4-dimenzios vektor az alpha, alpha-dot, x, x-dot komponensekkel
% ell - az inga hssza
% m1 - a fels"o t"omeg
% m2 - az also t"omeg
% r - a surlodasi egy"utthato
% f - a 4-dimenzios oszlopvektor, mint output

g=9.81; % [m/sec^2]
si=sin(z(1));co=cos(z(1));
m1m2=1/(m1+m2*si^2);

f(1,1)=z(2);
f(2,1)=-(r*0.5*z(2)+si/ell*(g*(m1+m2)+m2*ell*co*z(2)^2))*m1m2;
f(3,1)=z(4);
f(4,1)=(-r*z(4)+m2*si*(ell*z(2)^2+g*co))*m1m2;
```



4.3. ábra. *A „csúszós" inga mért szögei, a felső tömeg elmozdulásai (piros csillagok) és egész pályája (kék vonalok)*

Ismerjük a kezdeti értékeket : $\alpha(0) = 0$, $\dot{\alpha}(0) = 0.02$, $x(0) = 0$, $\dot{x}(0) = -0.2$.

Ehhez a mérési adataink (amelyeket lejjebb a `pendulfopt` m-fájl is tartalmazza jobban hasznosítható formában és amelyeket speciális, nem egész `ell, m1, m2, r` értékekkel kaptuk `ode45`-ből), ahol $\alpha(t_i) = \alpha_i^*$, $x(t_i) = x_i^*$:

$t, \alpha, x_i \setminus$ i	1	2	3	4	5
t_i	6.00	12.00	18.00	24.00	30.00
α_i^*	-0.0229	0.0084	0.0145	-0.0117	-0.0069
x_i^*	-0.3862	-0.8232	-1.0346	-1.0162	-1.0936
t_i	36.00	42.00	48.00	54.00	60.00
α_i^*	0.0115	0.0011	-0.0093	0.0025	0.0062
x_i^*	-1.2174	-1.1941	-1.1556	-1.2160	-1.2404
t_i	66.00	72.00	78.00	84.00	90.00
α_i^*	-0.0042	-0.0032	0.0044	0.0009	-0.0037
x_i^*	-1.1950	-1.1987	-1.2357	-1.2214	-1.1994

Az *inverz feladat* a következő : az inga hosszát, a felső és az alsó tömeget, valamint a súrlódási együtthatót nem ismerjük. Ezt a 4 paramétert, vagyis a p vektort határozzuk meg a 30 mérési adatból és a 4 kezdeti értékből.

A feladathoz tehát 34 adatot ismerünk és 4 paramétert keressünk. Vagyis kb. 8-szor annyi az ismert adat, mint a keresett – ami elég jó viszony.

Az inverz feladat megoldását úgy kapjuk meg, hogy sokszor oldjuk meg a direkt feladatot, újabb és újabb p paraméter vektorral indítva a direkt feladatot, amihez egy optimalizáló program választja ki mindig a következő 4 paramétert az előző eredményeknek a mérési adatoktól való távolsága alapján.

Az optimalizáló program az `lsqnonlin` program lesz az `optimization toolbox`-ból, ld. a magyar MATLAB-könyvet vagy `help lsqnonlin`-t, amely a következő négyzetösszeget igyekszik minimalizálni :

$$\begin{aligned} f(p) &= \sum_{i=1}^{2n} f_i^2(p), \;\; p = (\ell, m1, m2, r), \\ i = 1, \ldots, n \; : &\qquad f_i(p) = \alpha(t_i, p) - \alpha_i^* \, , \\ i = n+1, \ldots, 2n \; : &\qquad f_i(p) = x(t_i, p) - x_i^* . \end{aligned}$$

Itt nálunk $n = 15$ és $t_i = 6i$ másodperc.

Az `lsqnonlin` megfelelő hívását mutatja az alábbi m-fájl :

```
function p=pendulopt(p)
% Szamitja az optimalis 4 inga-parametert p(1) ... p(4)
% Hivja maga: optimset, lsqnonlin, pendulfopt, pendulum
%
% Hivasa :
% p=pendulopt(p)
% ahol
% p - input parammeterek vektora
% p - az optimalis parameterek vektora (output)

% a tablazat meresi alpha es x ertekei:
z0al=[-0.0229,0.0084,0.0145,-0.0117,-0.0069,...
       0.0115,0.0011,-0.0093,0.0025,0.0062,...
      -0.0042,-0.0032,0.0044,0.0009,-0.0037];
z0x=[-0.3862,-0.8232,-1.0346,-1.0162,-1.0936,...
     -1.2174,-1.1941,-1.1556,-1.2160,-1.2404,...
     -1.1950,-1.1987,-1.2357,-1.2214,-1.1994];
z0=[z0al';z0x'];

opt=optimset('tolfun',1e-8,'tolx',1e-8,'maxfunevals',2000,...
             'maxiter',1000);
lb=zeros(4,1);
ub=500*ones(4,1);

for i=1:3
    [p,~,~,eflag]=lsqnonlin(@pendulfopt,p,lb,ub,opt);
    f=pendulfopt(p);
    disp(['normf=',num2str(norm(f),'%1.15g'),', eflag=',int2str(eflag)])
    if eflag==1,break,end % megszakitas, ha megoldast talalt
end
```

```
pendulum(p(1),p(2),p(3),p(4)) % Ezt mar ismerj"uk: rajzokat is keszit
```

Itt használtuk az `optimset` függvényt, amellyel nemcsak az `lsqnonlin`, hanem más optimalizáló programok speciális paramétereit lehet előzetesen beállítani, mint fent pl. a `tolfun`, tehát a függvényértékek toleranciáját, vagy a `maxfuneval`, tehát a maximális függvény-kiértékeléseknek a számát. Ezekkel a paraméterekkel lehet ismerkedni a `help optimset` segítségével, vagy futás közben az optimalizáló program üzeneteit elemezve.

Most mutatjuk a jobboldalt kiszámító m-fájlt :

```
function f=pendulfopt(p)
% a csuszosan felf"uggesztett pendulum dif.-egyenletek megoldasa
%
% Hivasa :
% [t,z]=pendulfopt(ell,m1,m2,r)
% ahol
% ell - a pendulum hossza
% m1 - az els"o t"omeg
% m2 - a 2. t"omeg
% r - a surlodasi egy"utthato
% t - az id"ok vektora
% z - az al,x koordinatai
g=9.81; % [m/sec^2] F"old gyorsulasa, f(16)-ban

z0al=[-0.0229,0.0084,0.0145,-0.0117,-0.0069,...
       0.0115,0.0011,-0.0093,0.0025,0.0062,...
      -0.0042,-0.0032,0.0044,0.0009,-0.0037];
z0x=[-0.3862,-0.8232,-1.0346,-1.0162,-1.0936,...
     -1.2174,-1.1941,-1.1556,-1.2160,-1.2404,...
     -1.1950,-1.1987,-1.2357,-1.2214,-1.1994];
zz=[z0al';z0x'];

ell=p(1);
m1=p(2);
m2=p(3);
r=p(4);
epsr=1e-5;
a=0;b=90;
ab=linspace(a,b,b/6+1);
z0=[0,0.02,0,-0.2];
opts=odeset('reltol',epsr,'abstol',epsr*1e-3);
[~,z]=ode45(@(t,z) pendulumf(t,z,ell,m1,m2,r),ab,z0',opts);
f=zz-[z(2:16,1);z(2:16,3)];
%f(16)=10*(pi^2/25*ell*(m1+m2/2)-g*(m1+m2));
```

Itt az utolsó előtti sorban azért zártuk ki az 1. komponenseket `z(:,1),z(:,3)`-ból, mert azok a kezdeti értékeket tartalmazzák. Előbb, `ode45` hívásánál, `t`

kell, hogy legyen az első output paraméter, de nincs szükségünk rá. Ezért ott áll a `[~,z]` jelsorozat (hasonlóan, mint korábban is már a `pendulopt` program `[p,~,~,eflag]=...` utasításaiban).

Most minden készen áll az inverz feladat megtámadására. A `pendulopt`-ban megadott `optimset`-paraméterekkel hívjuk ezt a programot a következőképpen, mert a keresett ℓ, `m1`, `m2`, `r` nagyságrendjéről nincs információnk:

```
tic;for i=0:2,p=pendulopt(10^i*ones(1,4)),end;toc
```

Ennek eredménye, részben kerekítve :

```
i=0: normf=0.125242, eflag=3
p =[6.076367, 9.354079, 37.559267, 0.786833]
i=1: normf=0.173727, eflag=3
p =[13.5360349, 88.489249, 62.239846, 6.304001]
i=2: normf=0.178757, eflag=3
p = 1.0e+02 *[0.0685642, 2.687939, 4.965444, 0.2459457]
Elapsed time is 143.531380 seconds.
```

Ezek az eredmények mások, mint amit vártunk, ld. lejjebb a 4.4 ábrát is.

Mielőtt ezen probléma megoldásával foglalkoznánk, hadd említsük, hogy a `pendulopt` program kísérletezés eredménye :

1. Először gondoltunk, hogy nagyon okos lesz elsőnek az első paramétert meghatározni (a többi paramétert addig szűk határok közé szorítva), majd lépésenként 1-1 paramétert hozzávenni, ld. a program következő részletét :

```
lb=zeros(4);
lb(2:4,1)=p(2:4)-2e-6;lb(3:4,2)=p(3:4)-2e-6;lb(4,3)=p(4)-2e-6;
ub=500*ones(4);
ub(2:4,1)=p(2:4)+2e-6;ub(3:4,2)=p(3:4)+2e-6;
ub(4,3)=p(4)+2e-6;

for j=1:4
    [p,~,~,eflag]=lsqnonlin(@pendulfopt,p,lb(:,j),ub(:,j),opt);
    f=pendulfopt(p);
    disp(['normf=',num2str(norm(f),'%1.15g'),', eflag=',int2str(eflag)])
    if eflag==1,break,end % megszakitas, ha megoldast talalt
end
[p,~,~,eflag]=lsqnonlin(@pendulfopt,p,zeros(4,1),500*ones(4,1),opt);
```

Részletezzük a fent programozott `lb,ub` alsó és felső korlátok tervezett értelmét : Először `lb` 4×4-es nullmátrix (a keresett fizikai konstansok nemnegatívak). Ezután az alsó háromszögének (i,j) elemét felülírjuk a p(i)-2e-6 értékkel. Hasonlóan `ub` minden eleme először 500, majd alsó háromszögének (i,j) elemét felülírjuk a p(i)+2e-6 értékkel. Az erre következő `lsqnonlin`-hívását tartalmazó ciklus j-edik menetében `lb(:,j)`, ill. `ub(:,j)`, tehát `lb`, ill. `ub` j-edik oszlopa adja a minimum-keresésnek a korlátját. Ezzel az 1...j-edik paraméter 0 és 500 között mozoghat, míg a többi paraméter lényegében az input által adott helyén

fog maradni. Megjegyezzük, hogy `lsqnonlin` nem engedi a felső és alsó határok egybeesését, ami segíthetné az optimalizálást (a korábban, 4.2. pontban tárgyalásra került `intlinprog` megengedi). Mégpedig azzal, hogy ilyen esetben egyes változókat ki lehet zárni az optimalizálásból.

Kiderült viszont, hogy az a fenti programverzió rossz azért, mert a szűk határok figyelembe vétele rengeteg időbe került. Az eredmények, megint részben kerekítve, ugyanis :

```
i=0: normf=0.470113, eflag=3
p =[0.000300752, 5.810356, 0.999998, 0.999998]
i=1: normf=0.470100, eflag=3
p =[0.00116877, 58.100230, 10.000001, 9.999996]
i=2: normf=0.470107, eflag=3
p = 1.0e+02*[0.00000692844, 4.999999, 0.643236, 0.860480]
Elapsed time is 8756.974050 seconds
```

tehát kb. 2 óra, 26 perc kellett a külső i-ciklusra, és a végén normf csak 0.470100 lett. Közben az i=2 eredményen látszik az 500 felső határ is, míg i=0,1 esetén p(3:4) lényegében még a kezdetiértékén van.

2. Az előzők miatt hagyjuk el a határok lépésenkénti bővítését. Ekkor a fenti sorok helyett a következőket vesszük :

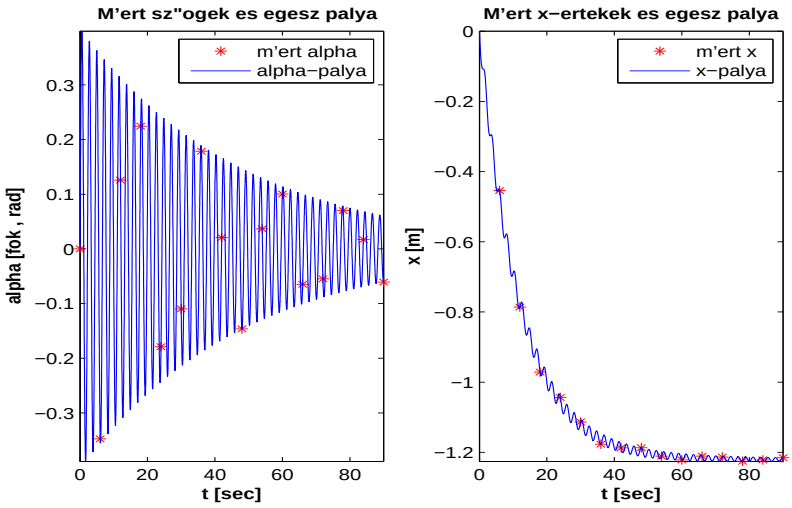
```
lb=zeros(4,1);
ub=500*ones(4,1);

for j=1:4
    [p,~,~,eflag]=lsqnonlin(@pendulfopt,p,lb,ub,opt);
    f=pendulfopt(p);
    disp(['normf=',num2str(norm(f),'%1.15g'),', eflag=',int2str(eflag)])
    if eflag==1,break,end % megszakitas, ha megoldast talalt
end

[p,~,~,eflag]=lsqnonlin(@pendulfopt,p,lb,ub,opt);
```

Ez a verzió jól néz ki, gyorsan futott le (175.7 másodperc), és feltűnt, hogy a j-ciklus `j=1:3` formában is bőven elég lenne. Akkor még a befejező `lsqnonlin`-hívás is megtakarítható, amivel a javasolt programot kapjuk.

3. Az előzőhöz képest csak az `ub`-t változtatjuk meg `ub=Inf*ones(4,1)`-re : 1429.8 másodperc és pl. i=1-re `normf`≈`0.097646`, p =[7.895284, 32.961811, 52.804034, 2.774536] lett.

4. Ugyanezeket az eredményeket kaptuk, amikor az `ub`-t `ub=[]`-re változtatjuk. Nem csoda, mert ez az `ub` az előző 3. lehetőségnek csak egy másik felírása, csak a számítási idő lényegtelenül változott : 1444.8 másodperc alatt i=2-ig `normf`≈`0.150302`-re lefut az i-ciklus.

Most térjünk vissza az előző oldal furcsa, nyilván az i-ciklus i változójától függő eredményeinkhez! Forduljunk a (4.35)-(4.36) egyenletekhez, azokat elemezve – amit már az elején kellett volna megtennünk.



4.4. ábra. *Az első (i=0) optimalizálás szerinti $\alpha(t)$ és $x(t)$, és a mérési eredmények : piros csillagok. A baloldal tisztán mutatja a magasabb frekvenciát 4.3-hoz képest.*

1) Ha az egyenleteket m_2-vel elosztjuk, akkor az eredmény

$$\begin{aligned}\ell(\frac{m_1}{m_2}+\sin^2\alpha)\ddot{\alpha}+\frac{r}{m_2}\frac{\ell}{2}\dot{\alpha} &= -\sin\alpha[g(\frac{m_1}{m_2}+1)+\cos\alpha\ell\dot{\alpha}^2],\\ (\frac{m_1}{m_2}+\sin^2\alpha)\ddot{x}+\frac{r}{m_2}\dot{x} &= \sin\alpha[\ell\dot{\alpha}^2+g\cos\alpha].\end{aligned}$$

Ez azt mutatja, hogy a megoldások a kezdeti értékeken kívül csak az ℓ, $\frac{m_1}{m_2}$ és $\frac{r}{m_2}$ számoktól függnek. Más szóval, **ha nem variáljuk a kezdeti értékeket, akkor csak az ℓ hosszat és az $\frac{m_1}{m_2}$ és $\frac{r}{m_2}$ viszonyokat tudjuk meghatározni**.

2) A kiszámított lengések frekvenciája nyilván mindegyik esetben hibás, elég az i=0-ás számításhoz tartozó 4.4. ábrát megnézni, amely az `1,1,1,1` paraméterekből induló optimalizálás eredményeit mutatja (rosszul látható az α-skála szorzója : 10^{-3}).

A hibás frekvencia ügyében segíthet a kis $\alpha, \dot{\alpha}$ közelítésének feltevése (4.35)-ben : ekkor, elhanyagolva az $\dot{\alpha}$, $\sin^2\alpha$ mennyiségeket (valamint a súrlódást is) és $\sin\alpha$-t helyettesítve α-val, kapjuk (az α kezdeti értékét is figyelembe véve)

$$\ell m_1\ddot{\alpha} = -\alpha[g(m_1+m_2)],\ \ \alpha(0)=0,$$

amely kezdeti érték feladatnak a megoldása jól ismert:

$$\alpha(t)=\sin\omega t,\quad \omega^2=\frac{g(m_1+m_2)}{\ell m_1}.$$

Ekkor egy periódus hossza $2\pi/\omega$, ami durván 10 másodperc a 4.3 ábra szerint, vagyis

$$100 \approx \frac{4\pi^2}{\omega^2} = 4\pi^2 \frac{\ell m_1}{g(m_1+m_2)}. \tag{4.37}$$

Így közelítőleg $\ell = 25g(m_1+m_2)/(\pi^2 m_1)$, amely relációt a `pendulfopt` m-fájl végéhez tesszük az `f(16)=10*(π^2/25*ell*m1-g*(m1+m2));` formában (a 10-es szorzó súlynak szolgál). Ezután, mint első lépés, a `pendulopt(10*ones(1,4))` hívásra kaptuk a következő eredményt (megint részben kerekítve) :

```
lsqnonlin stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 2000 (the selected value).

normf=0.284281082500339, eflag=0
p = 1.0e+02*[0.256163; 1.319143; 0.0407331; 0.207498]
```

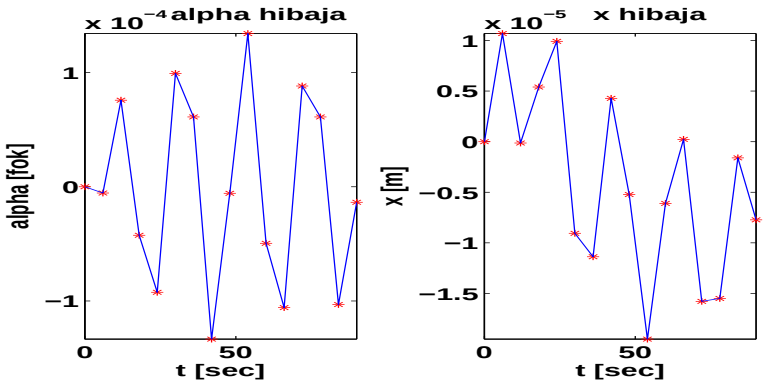
Most, mint második lépés, kikommentáltuk az `f(16)`-ot `pendulfopt`-ból, hiszen a (4.37) reláció csak közelítőleg érvényes, és már megtette a dolgát, ezenkívül az opciókat `pendulopt`-ban a következőkre változtattuk : `'tolfun',1e-11,'tolx',1e-11`. A kapott `p` vektorral újra beindítva az optimalizálást, az eredmény az alábbi lett :

```
lsqnonlin stopped because the size of the current step is less than
the selected value of the step size tolerance.

<stopping criteria details>

normf=0.000136778197887654, eflag=2
p = 1.0e+02*[0.220687; 1.306073; 0.350116; 0.114505]

Optimization stopped because the norm of the current step, 3.304478e-12,
is less than options.TolX = 1.000000e-11.

Optimization Metric                                  Options
norm(step) =   3.30e-12                      TolX =   1e-11 (selected)
```

A fentiek szerint az inverz feladat közelítő eredménye (és ez nem változik, ha első lépésként `pendulopt(ones(1,4))`-gyel indítunk, csak lényegesen több számítási időt követel)

$$\ell = 22.069, \ \frac{m_1}{m_2} = 3.7304, \ \frac{r}{m_2} = 0.32705.$$

Mivel ehhez olyan ábra tartozik, amelyet alig lehet megkülönböztetni a 4.3 ábrától, inkább az eltéréseket mutatjuk meg, ld. a 4.5 ábrát.



4.5. ábra. *A sikeres optimalizálás adta $\alpha(t)$ és $x(t)$ eltéréseit, és a mérési helyek : piros csillagok*

4.4.2. Függvény meghatározása közönséges differenciálegyenletekből álló rendszerben mérési adatokból

Ebben a pontban a következő „egyszerű kocsi"-modellel foglalkozunk (forrásunk : http://planning.cs.uiuc.edu/node658.html, avagy [21], 13.1.2.1. Részletesebben kocsi-modellekről ld. [16]) :

$$\dot{x} = u_s \cos\theta, \tag{4.38}$$

$$\dot{y} = u_s \sin\theta, \tag{4.39}$$

$$\dot{\theta} = \frac{u_s}{L} \tan u_\phi \,, \tag{4.40}$$

ahol a modellparaméterek

L – az elülső és a hátsó tengely (rögzített) távolsága: 2.6 m;
$u_s \in [u_{min}, u_{max}]$ – a kocsi sebessége, $u_{min} < 0 < u_{max}$;
u_ϕ – a kormány forgatási szöge, $|u_\phi| \leq \phi_{max} < \frac{\pi}{2}$;

és $x = x(t)$, $y = y(t)$ a hátsó tengely-középpontjának a t időben változó koordinátai, $\theta = \theta(t)$ pedig az elülső és hátsó tengely-középpontokra illeszkedő egyenes szöge az x-koordináta tengellyel. Ponttal jelöltük a t szerinti deriváltat.

u_s és u_ϕ változhatnak idővel, ezekről dönt a kocsivezető.

A fenti kocsimodellt implementálják az alábbi m-fájlok. A kocsi először tesz egy félkört, majd igyekszik $\mathtt{u}_\phi$-nek egy bizonyos választásával a $-x$-irányban és az $y \approx 12$ egyenes közelében menni. Míg u_s-t meg fogjuk adni t és $z = (x, y)$ függvényében, egy speciális $\mathtt{u}_\phi$-választás (egy $t \rightarrow u_\phi$ függvény) meghatározása mérési adatok alapján lesz az *inverz feladat* célja.

a) a kocsimodell főprogramja :

```
function [x,y]=mljkocsi(a,b,z0,epsr)
% Egyszer"u kocsi modellje, a kocsiut rajzolasaval.
% Hivja az ode45 Runge-Kutta-Fehlberg, es az mjlkocsif programot
```

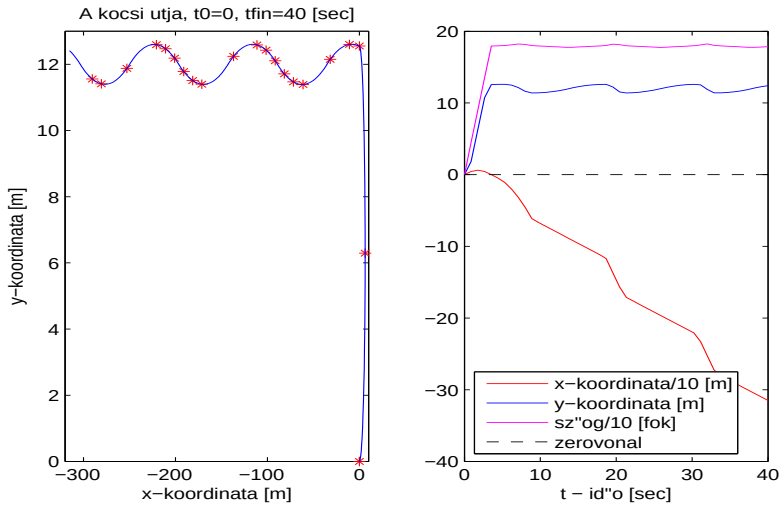
```
% amely a differencialegyenletek jobboldalat szamitja
%
% Hivasa :
% [x,y]=mljkocsi(a,b,z0,epsr)
% ahol
% a - kezdeti id"opont [sec]
% b - vegs"o id"opont [sec]
% z0 - kezdeti ertekek: [x0,y0,th0] ([m,m,rad])
% epsr - a relativ hiba toleranciaja ode45 szamara
% x - szimulalt meresi helyek x-koordinatai
% y - szimulalt meresi helyek y-koordinatai

opts=odeset('reltol',epsr,'abstol',epsr*1e-3);
[~,zz]=ode45(@mljkocsif,[a,b],z0,opts);
ab=linspace(a,b,b-a+6);
[t,z]=ode45(@mljkocsif,ab,z0,opts);
p=1:2:size(z,1);p=p(1:21);
z2=z(p,1:2);
x=z2(:,1);
y=z2(:,2);

subplot(1,2,1)
plot(x,y,'r*',zz(:,1),zz(:,2),'b-')
axis([-320 10 0 13])
title(['A kocsi utja, t0=',num2str(a),', tfin=',num2str(b),' [sec]'])
xlabel('x-koordinata [m]')
ylabel('y-koordinata [m]')
subplot(1,2,2)
plot(t,z(:,1)/10,'-r',t,z(:,2),'b-',t,z(:,3)*18/pi,'m-',t,...
     zeros(length(t),1),'--k')
xlabel('t - id"o [sec]')
legend('x-koordinata/10 [m]','y-koordinata [m]','sz"og/10 [fok]',...
       'zerovonal','location','southwest')
```

Ebben az m-fájlban a döntő hívás az `ode45` programé, amelyet azért választottuk, mert itt merevséggel nem kell számolnunk (vö. [33], 11.3 és 11.5 résszel).

A 4.4.2. alpunkt programjairól: Az itt vizsgált feladat programjainak kidolgozása sok kísérletezéssel járt. A közölt m-fájlok szövege a végső állapotnak felel meg, éppúgy, mint a végső eredmények. De egyes közbülső eredmények lehet, hogy keletkeztek a programok közbülső állapotából.

b) a jobboldali függvény :

```
function f = mljkocsif(t,z)
% Az egyszer"u kocsimodell jobboldala. Hivja: mlju, mljp
```



4.6. ábra. *Az egyszerű kocsi útja és az* (x_i^*, y_i^*) *mérési eredmények : piros csillagok*

```
L=2.6; % L - tengelytavolsag [m]

ut=mlju(t,z);       %   u_s megadasa
pt=mljp(t,z);       %   u_phi megadasa

f(1,1)=ut*cos(z(3));
f(2,1)=ut*sin(z(3));
f(3,1)=ut/L*tan(pt);
```

Itt látjuk, hogy az `ode45` jobboldalként oszlopvektort vár, míg a `z0` kezdeti érték sorvektor is lehet.

c) a sebesség megadása `t` és `z` függvényében :

```
function u = mlju(t,z)
% A kocsi sebesseget ([m/sec]) adja t es z f"uggvenyeben

if z(3)<pi
u=2e4/3600; % tehat 20 km/h, mig meg nem fordult a kocsi
elseif t>4
       u=min(8e4,(2e4+1e4*(t-4)))/3600; % max. sebesseg: 80 km/h
end
```

d) egy bizonyos ϕ-vezérlést megvalósító `mljp`-ből indultunk ki, de ennek (akár közelítő) meghatározása éppen a feladatunk.

A 4.6 ábra mutatja a kocsiútat és az azt leíró függvényeket. Ehhez a főprogram hívása :

```
mljkocsi(0,40,[0,0,0],1e-9)
```

Csillagokkal jelöltük az ábrán az inverz feladat mérési adatait. Ezen pontok (x_i^*, y_i^*) koordinátáit (a vessző utáni 3 számjegyre kerekített formában) tartalmazza a táblázat. Ezen adatokra, mint az `mljdat` vektorra, hivatkozunk a későbbiekben, az x_i^* és y_i^* számokat x^* és y^* oszlopvektorokba rendezve : `mljdat=[`x^*`;`y^*`]`.

$x^*, y^* \setminus i$	1	2	3	4	5	6	7
x_i^*	0	6.277	-0.033	-10.781	-31.626	-61.243	-71.905
y_i^*	0	6.294	12.555	12.605	12.159	11.392	11.467
i	8	9	10	11	12	13	14
x_i^*	-81.778	-91.647	-101.518	-111.393	-136.368	-171.557	-181.433
y_i^*	11.707	12.102	12.428	12.590	12.269	11.392	11.498
i	15	16	17	18	19	20	21
x_i^*	-191.305	-201.174	-211.046	-220.921	-251.153	-281.085	-290.961
y_i^*	11.767	12.172	12.468	12.602	11.991	11.399	11.534

Inverz feladatunk megoldásának első lépése annak eldöntése, hogyan tudjuk az $u_\phi(t)$ függvényt lehetőleg kevés paraméterrel leírni? Más szóval : hogyan modellezzük $u_\phi(t)$-t?

A függvény modellezése

Nagyon meggondolandó kérdés, hogyan írjuk le a keresett függvényt lehetőleg kevés paraméterrel? (Az, hogy mi itt a „kevés”, az a tapasztalat szerint a mérési adatok számától függ : legyen legalább kétszer annyi mérési adatunk, mint keresett paraméter, de általában nem kell, hogy tízszer annyi legyen.)

Ha véges intervallumon keressük a függvényt, akkor jöhetnek szóba a polinomiális interpoláció lehetőségei. (Végtelen intervallumon ezek nem alkalmasak, ekkor racionális függvények javasolhatók, vagy pedig exponenciálisan véges határértékhez, leginkább nullához tartozó függvények jók.) Viszont ha periodikus folyamatokra lehet gondolni, akkor a véges Fourier–sorok megfelelőek, amelyekre esetünkben is kézenfekvő gondolni, ld. lejjebb a 4.6 ábrát a mérési eredményekkel.

A felsorolt lehetőségek közvetetten az alkalmas függvénytér kiválasztását jelentik. Ez a választás viszont fontos általános matematikai probléma is. A jelenlegi alpontunkban pl. közönséges differenciálegyenletek rendszeréről van szó, amelyben keressük a független változónak egy függvényét a differenciálegyenlet jobb oldalán. Itt emlékeztetünk arra a tételre, amely szerint a megoldás létezésének (majdnem minimális) követelménye, hogy a jobboldal legyen Lipschitz-folytonos a megoldásban és mérhető (pl. négyzetesen integrálható) a független változóban.

Amellett a függvénytér kiválasztása egy fizikai modell felállításának is megfelel, és ez a vizsgált alkalmazás szempontjából értelmes kell, hogy legyen. Azonkívül, hogy természetesen jó megegyezést akarunk elérni a mérési adatokkal, egy további mérlegelendő aspektus lehet, hogy egy-egy függvényosztály hogyan hat ki a számítási időre?

A szakaszoson konstans függvény választása

Ez a legegyszerűbb lehetőség, és megfelel a négyzetes integrálhatóságnak.

Mivel az elején a félkör elő van írva és erre 4 másodperc kell a megadott sebesség mellett, a következő m-fájl lehet alkalmas :

```
function p = mljphi(t,z,phi)
% A kormany sz"oget [radiansban] adja t,z es phi f"uggvenyeben
if t<4                          % a felk"or sz"ogei
   if z(3)<pi-0.2e-1
      p=pi/8;
   elseif z(3)>pi+1e-2
      p=-pi/18;
   elseif z(3)<pi-1e-2
       p=pi/18;
   else
       p=0;
   end
else                            % a felk"or utani resz
    i=min(floor((t-4)/2)+1,18); % a 18 kes"obb 16-ra valtozik
    p=phi(i);
end
```

Az m-fájl vége mutatja, hogy a `t` változó $[4, 40]$ intervallumát 17 darab $[t_i, t_{i+1})$ részintervallumra és a $t = 40$ pontra osztjuk fel és mindegyik részintervallumban a megfelelő `phi`-komponens adja a szöget, míg `t=40`-ben a $\texttt{phi}_{18}$ értéke áll.

A következő lépés az inverz feladat megoldásához azon út megtalálása, hogyan tudjuk eljutatni a keresett függvény paramétereit (esetünkben a `phi` (oszlop)vektor értékeit, amelyek az elején az optimalizáló program kezdeti értékei, majd később ezeket az optimalizáló program fogja előírni) a paramétereket felhasználó m-fájlhoz? Itt ez a fenti `mljphi`.

Előbb nyilván a differenciálegyenleteket megoldó `ode45` programot kell hívnunk, az hívja a jobboldali függvényeket kiszámító m-fájlt (amely hasonló lesz a fenti az `mljkocsif`-hez), és csak az utóbbi hívja az `mljphi`-t. A megoldás itt egy *anonim függvény* (avagy *pointer-függvény*) használata :

```
function xy=mljv(phi)
% A phi szakaszoson konstans sz"ogek kihatasa a kocsiutra,
% ezt hivja az optimalizalo program, maga hivja az ode45-"ot
%
% Hivasa :
```

```
% xy=mljv(phi)
% ahol
% phi - 18-dimenzios oszlopvektor a sz"ogekkel - kes"obb a dimenzio 16
% xy - a kocsi koordinatai (1-21: x, 22-42: y)

epsr=1e-9;
z0=[0,0,0];
opts=odeset('reltol',epsr,'abstol',epsr*1e-3);
a=0;
b=40;
ab=linspace(a,b,46);
[t,z]=ode45(@(t,z) mljfphi(t,z,phi),ab,z0,opts);
p=1:2:size(z,1);p=p(1:21);
z2=z(p,1:2);
xy=[z2(:,1);z2(:,2)];
```

Itt tehát a

```
[t,z]=ode45(@(t,z) mljfphi(t,z,phi),ab,z0,opts);
```

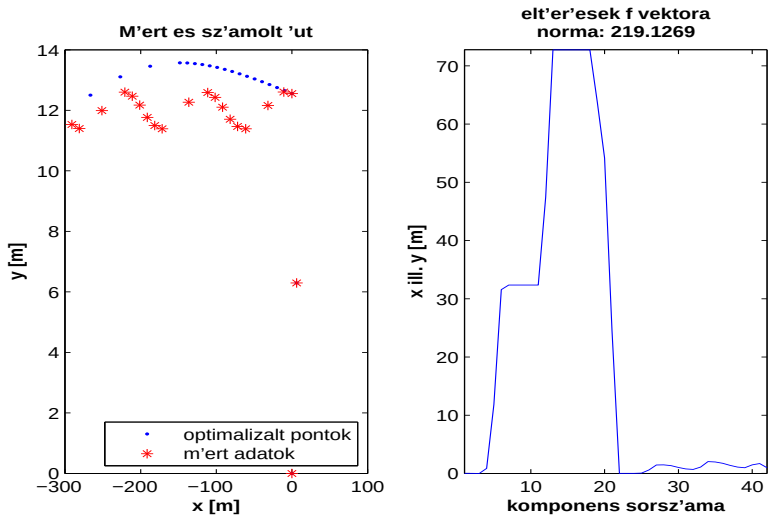
a döntő sor : az `ode45` program első paramétere olyan függvény kell legyen, amely t és a 3-dimenziós z vektor alapján a (4.38)-(4.40) rendszer jobboldalát számítja ki (oszlop)vektorként (ld. fent a b részbeli m-fájlt). De most a `phi`-vektort is kell közvetítenie. Emiatt az első input paraméter `@(t,z) mljfphi(t,z,phi)`. Ez az utóbbi m-fájl szinte ugyanaz, mint az `mljkocsif`, csak 3. argumentuma `phi` – és benne a `pt`-vektor kiszámítása változik :

`pt=mljp(t,z);` helyett `pt=mljphi(t,z,phi);`

Az `ode45` 2. output paramétere egy ponthalmaz és nem egy intervallum, ezért a program csak a ponthalmazhoz tartozó megoldási értékeket fogja kiszámítani.

Harmadik lépésként, mielőtt az optimalizáló programot futtatnánk, érdemes egy alkalmas kezdeti `phi`-vektort meghatározni. Az a választás pl., hogy `phi=zeros(18,1)`, kevésbé jó eredményt adott (a pontok y-koordinátai szinte lineárisan tartottak 15-höz). Néhány további, a fenti `mljv` m-fájlra támaszkodó számítás után már elfogadhatóbbnak tűnt az alábbi, ld. ehhez a 4.7 ábrát. Az `mljdat` vektort a 110. oldalon definiáltuk.

```
xy=mljv(linspace(0,0.0005,18)'); % linear. valtozo phi-ertekek
xyd=mljdat;
plot(xy(1:21),xy(22:42),'b.',xyd(1:21),xyd(22:42),'r*')
title('Probaszamitas mljv-vel','fontweight','bold')
xlabel('x [m]','fontweight','bold')
ylabel('y [m]','fontweight','bold')
legend('probaszamitas','meresi eredmenyek','location','south')
```



4.7. ábra. *A kocsi útja enyhén növekvő* `phi` *szögek esetén (az f vektor 1-21. komponense x-koordináta, 22-42. komponense y-koordinata)*

A negyedik lépés az optimalizáló program kiválasztása és a neki megfelelő m-fájl, amelynek outputját fogja minimalizálni. Mi itt újra az `lsqnonlin` programot választjuk, ld. a 4.4.1. részben a 101. oldalt.

Ez a program nem egészen azt várja, amit az `mljv` nyújt, a mindenkori `phi`-szögekhez tartozó, a kocsiutat jellemző x- és y-koordinátákat (ezeket az `xy(phi)=[x;y]` vektorba gyűjtjük), hanem azoknak eltérése a fenti táblázatban adott mérési eredményektől (amelyeket az `xy0=[x*;y*]` vektorba gyűjtjük). Ezen két vektor eltérését adja az alábbi m-fájl output-paraméterként :

```
function f=mljf(phi)
% A phi sz"ogek kihatasa a kocsiutak elteresere; hivja: ode45-"ot
% (szamitott, ill. m'ert pozicio)
%
% Hivasa :
% f=mljf(phi)
% ahol
% phi - oszlopvektor a sz"ogekkel
% f=xy-xy0 - a kocsi szamitott, ill. m'ert koordinatai (1-21: x, 22-42: y)

epsr=1e-9;                                          % kes"obb epsr=1e-10
z0=[0,0,0];
opts=odeset('reltol',epsr,'abstol',epsr*1e-3);
a=0;
b=40;
ab=linspace(a,b,46);
```

```
[~,z]=ode45(@(t,z) mljfphi(t,z,phi),ab,z0,opts);
p=1:2:size(z,1);p=p(1:21);                              %          ***
z2=z(p,1:2);
xy=[z2(:,1);z2(:,2)];
xy0 =[0;6.277;-0.033;-10.781;-31.626;-61.243;-71.905;-81.778;-91.647;...
    -101.518;-111.393;-136.368;-171.557;-181.433;-191.305;-201.174;...
    -211.046;-220.921;-251.153;-281.085;-290.961;0;6.294;12.555;...
    12.605;12.159;11.392;11.467;11.707;12.102;12.428;12.590;12.269;...
    11.392;11.498;11.767;12.172;12.468;12.602;11.991;11.399;11.534];
f=xy-xy0; % az x- es y-komponenseket sulyozassal k"ul"onb"ozhetn'enk meg
```

és az `lsqnonlin` a $\|f\|^2 = \sum_{i=1}^{42} f_i^2$ négyzetösszeget fogja minimalizálni. Ezt szolgálja a következő m-fájl :

```
function phi=mljopt(phi)
% Szamitja az optimalis phi-vektort, hivja: lsqnonlin-t, ezzel mljf-et
%
% Hivasa :
% phi=mljopt(phi)
% ahol
% phi - input sz"ogek vektora
% phi - az optimalis sz"ogek oszlopvektora (output)

% phi=linspace(0.0005,0,18)'; % a kezdeti phi-ertekek

phi0=phi;
normf(1)=norm(mljf(phi));
disp(['Kezdet: |f|=',num2str(normf(1),'%1.15g')])
opt=optimset('tolfun',1e-12,'tolx',1e-12,...          % kes"obb tolx=1e-13
             'maxfunevals',15000,'maxiter',3000);
lb=-pi/4*ones(1,18);
ub=pi/4*ones(1,18);
for i=1:3
    disp(['i=',int2str(i),':'])
    [phi,~,~,eflag]=lsqnonlin(@mljf,phi,lb,ub,opt);
    normf(i+1)=norm(mljf(phi));normfi=normf(i);
    v=(normfi-normf(i+1))/normfi;
    disp(['utolso 2 f-ertek: ',num2str(normfi,'%1.15g'),', ',...
          num2str(normf(i+1),'%1.15g'),...
          '; rel. javitas: ',num2str(v),', eflag=',int2str(eflag)]);
    % megszakitas, ha megoldast talalt vagy semmivel nem lepett el"ore:
    if eflag==1 || normf(i+1)==normfi,break,end
end
[phi,~,~,eflag]=lsqnonlin(@mljf,phi,...
                -pi/4*ones(1,18),pi/4*ones(1,18),opt);
normfi=norm(mljf(phi));
v=(normf(1)-normfi)/normf(1);
```

```
disp(['1. es utolso f-ertek: ',num2str(normf(1),'%1.15g'),', ',...
      num2str(normfi,'%1.15g'),...
      '; rel. "osszes javitas: ',num2str(v),', eflag=',int2str(eflag)]);

tt=1:18;                                % kes"obb ebb"ol lesz a tt=1:16;
plot(tt,phi0(tt),'b.',tt,phi(tt),'m+')
title('Kezdeti es optimalizalt sz"ogek','fontweight','bold')
xlabel(['phi-vektor komponensei, |f|=',num2str(normfi)],'fontweight','bold')
ylabel('sz"ogek [rad]','fontweight','bold')
legend('kezdeti sz"ogek','optim. sz"ogek','location','best')
```

Ebben az m-fájlban a `[phi,~,~,eflag]=...` utasítások azért szerepelnek (hasonlóan, mint korábban a `pendulopt` programban, vö. a 103. oldallal), mert az `lsqnonlin` program leállásakor `eflag`-ben közli a leállás okát (a paraméter angol neve „exit flag”, kimeneti jelző), és ez kell, hogy a 4. output-paraméter legyen – de a 2. és 3. output-paraméter nekünk most nem fontos (a 2. paraméter például a négyzetösszeg, de helyette inkább annak gyöke, az $\|f\|$ érdekel).

Ahogyan említettük a fenti 3. lépéssel kapcsán, a

```
phi=linspace(0,0.0005,18)';
```

kezdeti értékek (és az abból `mljv`-vel kapott `xy` vektor) jól szerepeltek a $\|f\|$ szempontjából az előzetes próbálkozások során. Ugyanakkor itt az `lsqnonlin` program csak lényegtelen csökkenést tudott elérni. Ezzel szemben a

```
tic;phi=mljopt(linspace(0.0005,0,18)'),toc
```

választás nagyobb, kb. $\|f\| = 478.3$ kezdetiértéket adott ugyan, de az ezzel induló optimalizáló program ezt gyorsan csökkentette kb. 200-ra. Innen az $||f||$ célfüggvény értékét sikerült tovább csökkentetnünk :

```
lsqnonlin stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 15000 (the selected value).
i=1:
utolso 2 f-ertek: 478.27022231675, 199.957148118603; rel. javitas: 0.58192, eflag=0

Solver stopped prematurely.

lsqnonlin stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 15000 (the selected value).
i=2:
utolso 2 f-ertek: 199.957148118603, 116.90105441243; rel. javitas: 0.41537, eflag=0

Solver stopped prematurely.

lsqnonlin stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 15000 (the selected value).
i=3:
```

```
utolso 2 f-ertek: 116.90105441243, 116.90105441243; rel. javitas: 0, eflag=0

Solver stopped prematurely.

lsqnonlin stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 15000 (the selected value).
i-ciklus utani lsqnonlin-hivas :
|f|=116.90105441243, eflag=0
Elapsed time is 7002.292435 seconds.
```

Tehát csak kétszer sikerült az `f`-értéket javítani, a jelentős számítási idő ellenére.

Ekkor előnyösnek gondoltuk a {phi$_1$,...,phi$_{18}$} csak egy részét, pl. csak egy harmadát optimalizálni, a többi változót szűk határok közé szorítva, a következőképpen :

```
function phi=mljopt(phi,kkk)
...
% kkk - kapcsolo az optimalizando ismeretlenek k"oz"ott:
%       1: els"o harmad, 2: 2. harmad, 3: 3. harmad
...
lb=-45*pi/180*ones(1,18);
ub=45*pi/180*ones(1,18);
s=3e-8;
switch kkk
   case 1
   lb(7:18)=phi(7:18)-s;ub(7:18)=phi(7:18)+s;
   case 2
   lb([1:6,13:18])=phi([1:6,13:18])-s;ub([1:6,13:18])=phi([1:6,13:18])+s;
   case 3
   lb(1:12)=phi(1:12)-s;ub(1:12)=phi(1:12)+s;
end
...
```

míg az i-ciklus és a későbbi utasítások nem változnak. Célünk ekkor, hogy a 18-dimenziós optimálizálási probléma dimenzióját lényegében 6-dimenziósra csökkentsük. Példának említjük, hogy ekkor `phi=mljopt(phi,1)` azt fogja eredményezni, hogy az 1-6. változó szerint szabadon fut az optimalizálás, míg a 7-18. változók lényegében le vannak rögzítve. (Már rámutattunk a 4.4.1. pontban arra, hogy az `lsqnonlin` program nem engedi az alsó és felső korlátok megegyezését.) Ugyanakkor az `mljopt` program negyedik `lsqnonlin`-hívása nem tartalmazza a határok szűkítését : minden `phi`-értékre hat.

A kkk=3 esetben érdemes lehet néha az indextartományt az aktuális közbülső eredmények alapján megváltoztatni (ld. 126. oldalt is).

Erre a fenti új `mljopt`-verzióra és a

```
tic;for kkk=1:3
```

```
disp(['kkk=',int2str(kkk),':']);
phi=mljopt(phi,kkk)
end;toc
```

hívásra kaptuk a követező eredméyeket (|f| kerekített) :

```
kkk=1:
|f|=115.504644, eflag=0
kkk=2:
|f|=108.589047, eflag=0
kkk=3:
|f|=77.107322, eflag=0
Elapsed time is 21623.466266 seconds
```

ahol az `eflag=0` mindig azt jelenti, hogy a függvénykiértékeléseknek maximális száma, tehát 15000, túl kicsi volt. Emiatt nem értelmes, az `optimset` függvénynek `tolfun` vagy `tolx` paramétereit csökkenteni.

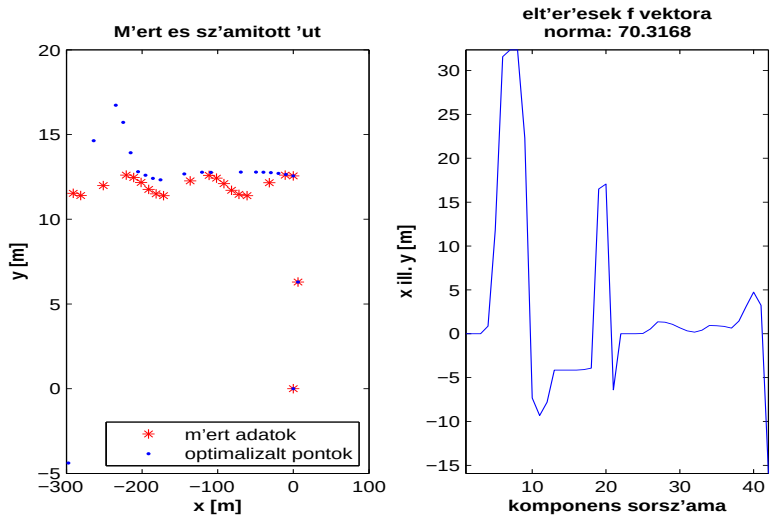
Úgy tűnik, hogy lehetőségként nekünk csak a fenti kkk-ciklus ismétlése marad. De ekkor már az `f` csökkenésének sebessége gyakran lelassul :

```
kkk=1:
|f|=76.979798, eflag=0
kkk=2:
|f|=76.906843, eflag=0
kkk=3:
|f|=71.479574, eflag=0
Elapsed time is 18188.637000 seconds.
kkk=1:
|f|=71.418168, eflag=0
kkk=2:
|f|=71.085878, eflag=0
kkk=3:
|f|=70.316769, eflag=0
Elapsed time is 21880.854187 seconds.
```

A 4.8 ábra az utóbbi számítási eredményt mutatja. (Emlékeztetésül : az `f` vektor 1-21. komponense az eltérés x-, a 22-42. komponense annak y-komponenseit tartalmazza.)

Erre az ábrára nézve és abból kiindulva, hogy a $\texttt{phi}_i$ értéke körülbelül a trajektóra i-edik pontjára hat ki (hiszen van 18 phi-értékünk és kevéssel több, 21 egyenletesen elosztott mérési helyünk, később viszont csökken $\texttt{phi}_i$ kihatása), azt gondoljuk, hogy leginkább az 5-8. és a 16-18. phi-értéket kellene jobban meghatározni. Emiatt az `mljopt`-beli kapcsolót a következőre allítjuk át :

```
switch kkk
    case 1
```



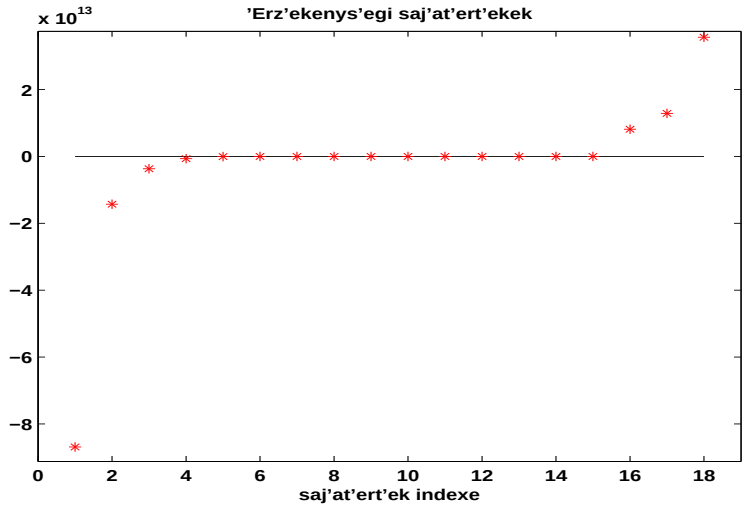
4.8. ábra. *Az optimalizálásnak egy közbülső eredménye,* $\|f\| \approx 71.0859$*-ből kiindulva. Balra : a mért és számított út. Jobbra : az* $f = xy - xy0$ *vektor komponensei és euklideszi normája.*

```
    lb([1:4,9:15])=phi([1:4,9:15])-s;ub([1:4,9:15])=phi([1:4,9:15])+s;
    case 2
    lb([5:8,16:18])=phi([5:8,16:18])-s;ub([5:8,16:18])=phi([5:8,16:18])+s;
    case 3
    lb([1:4,9:15])=phi([1:4,9:15])-s;ub([1:4,9:15])=phi([1:4,9:15])+s;
end
```

Kis elméletünk részben bejött :

```
tic;for kkk=1:3,disp(['kkk=',int2str(kkk),':']),phi=mljopt(phi,kkk),end;toc
kkk=1:
|f|=64.405161, eflag=0
kkk=2:
|f|=64.387974, eflag=0
kkk=3:
|f|=64.205801, eflag=0
Elapsed time is 18041.553518 seconds.
```

Itt a lényeges előrelépés csak `kkk=1`-nél történt meg, és a kkk-ciklus utáni, 4.8-nak megfelelő ábra alig változott, szinte csak a 11. `f`-komponensnél látható minimum eltűnt. („Kis elméletünk" persze javításra szorul, bár a folyamat lassulását nem okozza : az `mljf`-beli, csillagozott sor, ld. a 113. oldalt, mutatja, hogy $\mathtt{f}_j$-hez tartozik a `t=16(j-1)/9` időpont, és a `mljphi`-beli `i=min(floor((t-4)/2)+1,18)` képlettel kombinálva azt kapjuk, hogy először is `j`$\geq$`4` kell legyen (mert csak `t`$\geq$`4`-re érvényesés a képlet), továbbá, az $\mathtt{f}_{21}$-hez



4.9. ábra. *A közbülső,* `|f|=64.205801`*-nek megfelelő adatokkal készített érzékenységi vizsgálat sajátérték eredménye. A skálaszorzó* 10^{10}.

tartozó idő 320/9, amelynek i indexe 16. Így hiába van még $\texttt{phi}_{17}$ és $\texttt{phi}_{18}$, csak azoknak nincsen kihatása az $\texttt{f}_j$-kre.)

Számításaink során a célfüggvény értékét túl lassan csökkenőnek találva (sőt néha teljes akadozást is észlelve), emlékszünk a 4.1.4-pontbeli *érzékenységi vizsgálat*ra, amely itt, ahol 18 optimalizálandó paraméterrel állunk szemben, különösen érdekes lehet, és nem követel túl sok gépidőt.

Alkalmazva a `sensib` programot az `mljf` függvényre, amikor a hívás alakja (ahol `phi` a fenti, utolsónak kapott `|f|=64.205801`-nek felelt meg)

```
[V,L,neg,pos1,pos2,pos3,nul,g,delta]=sensib('mljf',phi,1e-4)
```

a 4.9 ábrát kaptuk az `L`-ben tárolt sajátértékek alapján (`epsi=1e-4` magyarázata, hogy `mljf`-ben a pontosság `epsr=1e-9`, és [32], 0.4.3 szerint a derivált kiszámításakor differencia képletből az optimális lépéstávolság `epsr` gyöke.

A Hesse-matrix 2 reszenek normai : ||G||=3.3575e+13, ||F||=8.9237e+13

A döbbenetes az, hogy 10 negatív sajátérték van : -8.6867e+13, -1.4264e+13, -3.6616e+12, -6.5897e+11, -1.8587e+10, -4.3254e+09, -2.8828e+08, -6.7264e+07, -4.8355e+07, -6.597e+06. De emlékszünk, ezek hasznunkra válhatnak : a hozzájuk tartozó sajátértékek lehetnek leereszkedési irányok. A két (egzaktul) zérus sajátértéken nem kell csodálkoznunk, és hozzájuk tartozik a 17., ill. a 18. koordináta egységvektor, egyhangban a fent elmondattokkal $\texttt{phi}_{17}$ és $\texttt{phi}_{18}$-ról. A 6 pozitív sajátérték között három a nagyságrendjével magaslik ki : 8.1496e+12, 1.2899e+13, 3.5668e+13.

A folytatásban a (4.19) V-ciklussal dolgoztunk (80. o.), az elején próbaképpen csak a legelső, legnagyobb abszolútértékű negatív sajátértékhez tartozó sajátvektort használva, majd sorban a többi alteret `goldensearch`-ben (68. o.) :

```
[phi,f]=goldensearch('mljf',phi,V(:,1)*V(:,1)'*phi,1e-2,1e-8,1e-8)
f0=64.205800824458791
f=64.201626599522243, rel. javitas: 6.5013e-05
[phi,f]=goldensearch('mljf',phi,V(:,2:10)*V(:,2:10)'*phi,1e-2,1e-8,1e-8)
f0=64.201626599522243
f=64.200596895473339, rel. javitas: 1.6039e-05
[phi,f]=goldensearch('mljf',phi,V(:,pos1)*V(:,pos1)'*phi,1e-2,1e-8,1e-8)
f0=64.200596895473339
f=64.200529059428561, rel. javitas: 1.0566e-06
[phi,f]=goldensearch('mljf',phi,V(:,pos2)*V(:,pos2)'*phi,1e-2,1e-8,1e-8)
f0=64.200529059428561
f=64.20052837635906, rel. javitas: 1.064e-08
[phi,f]=goldensearch('mljf',phi,V(:,pos3)*V(:,pos3)'*phi,1e-2,1e-8,1e-8)
f0=64.20052837635906
f=64.200511779306126, rel. javitas: 2.5852e-07
```

Itt tehát két `goldensearch`-hívással figyelembe vettük a negatív sajátértékekhez tartozó V_1 alteret, azt szétbontva : V_1={V(:,1),V(2:10)}. Az egész V-ciklusból valóban a V(:,1) hozta a legjobb eredményt, bár ennél is kicsiny volt a javulás. További kísérletezéssel találtuk, hogy a V_1 ilyen szétbontása valójában nem érdemes. (Megjegyezzük, hogy a `goldensearch` programban az itt nem mutatott szögek kinyomtatását (ld. 70. o.) most a `disp(num2str(reshape(x',6,3),'%1.15g'))` formába írtuk át.) Így az `mljopt` és a `sensib`-V-ciklus ($i = 0, \ldots, 4$-re szóló V_i-vel) kombinációt folyamatosan alkalmazva haladtunk tovább.

Egyébként, az érzékenységi sajátértékek maguk is elég érzékenyek, mint ahogyan kísérletekkel találtuk. Amikor pl. nem egy `mljopt`-optimalizálás végén még a gépben lévő `phi` értékekből indultunk ki, hanem a `format long`-gal kinyomtatott értékeket újabb `sensib`-számítás kezdetének vettük (ezzel tehát a `phi` komponenseinek rejtett számjegyeit elvesztettük – ami nem történt volna, ha `save`-vel kimentjük és, a munka folytatásakor, `load`-dal visszahozzuk a vektort), akkor már a (negatív) sajátértékek, sőt a száma is változhatott. Hasonlóan `sensib` eredményei változhatnak, hogyha az `epsi` paramétere nem `1e-4`, hanem pl. `1e-5`.

Megjegyezzük, hogy [7]-ben található (ott a 49-51. oldalon) a szigorú Wolfe-Powell–lépésstratégia algoritmusa és elmélete, amely a `goldensearch`-beli keresési intervallum folyamatos felosztása helyett addig lép a minimumkeresés érdekében, míg a célfüggvény eleget csökkent, és ehhez követeli a gradiens kiszámítását minden lépésben (úgy, hogy ez a Wolfe-Powell–algoritmus javasolható, ha a gradiens analitikusan megvan).

A `goldensearch` vonatkozásában az derült ki célszerűnek, ha a hívását néha úgy változtatjuk meg, hogy a `phi` vektort nem azonnal felülírjuk, hanem variálva a negyedik, fent 1e-2 értékű `dt0` paramétert (pl. `dt0=1e-2,1e-1,1e-0`

stb., de ha az `1e-1` eredménye rosszabb az `1e-2`-höz tartozónál, akkor folytatva `1e-3,1e-4` stb-vel), csak azt fogadva el új `phi`-nek, amelynek $\|f\|$-értéke a legkisebb. Továbbá, a negatív sajátértékek feltűnése bár megakadályozza a (4.20) értelemben vett érzékenységi vizsgálatot, de a minimalizálás szempontjából előny, hiszen (a negatív gradiens mellett) leereszkedési irány létét jelenthetnek.

Azt is tapasztaltuk, hogy gyakran nem érdemes egy „érzékenységi" $V_0 \to V_1 \to V_2 \to V_3 \to V_4$ ciklust azonnal ismételni. Itt leereszkedési módszerről van szó, és mint ahogyan a gradiens módszereknél már elmondtuk, általában a legnagyobb haladás az elején figyelhető meg. Érdemes viszont a V-ciklus és az `mljopt` váltása.

Az optimalizálás lassúsága arra késztetett, hogy elérve az |f|≈52.568295-höz, a [0,40] intervallumról átváltottunk a [0,320/9]-re, mert ezután a számítás valójában már felesleges, hiszen nincs mérési adatunk a (320/9,40] intervallumban. Ez azt jelenti, hogy a 45 lépés helyett csak 40-et tesz az `ode45` program, ami kb. 9% nyeremény, továbbá, csökken a `phi`-vektor dimenziója 16-ra és eltűnnek a nulla sajátértékek az érzékenységi vizsgálatnál.

Ílymódon folytatva, elértük a 4.10 ábrán látható eredményt. Ehhez az optimalizálást (kisebb megszakításokkal) négy hónapig kellett futtatnunk, mindig váltakozva egy kkk-ciklust (ld. 117. oldalt fent) és egy érzékenységi ciklust. (Achi Brandt [2], a híres, parciális differenciálegyenletek megoldásával kapcsolatos többrácsos algoritmus bár nem feltalálója, de sikeres fejlesztője szerint **„stalling numerical processes must be wrong"**, vagyis, szabadon lefordítva : akadozó algoritmusokat le kell váltani, jobbakkal helyettesíteni. Esetünkben : mivel? Lejjebb fogjuk látni, hogy a spline-ok használata lényegesen javítja a helyzetet.)

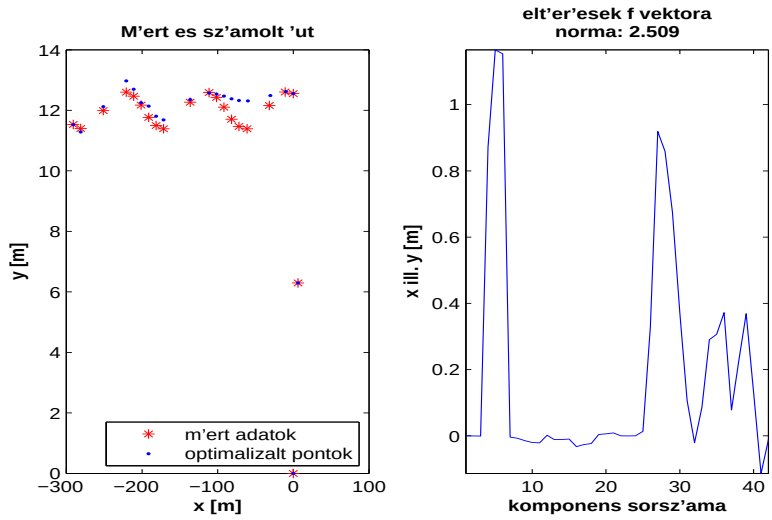
Az ekkor abbahagyott számítás eredménye elfogadhatónak tűnik. Végül is, az elején a célfüggvény értéke (`f` euklideszi normája) nem kisebb volt 2.51-nél, mint most, hanem nagyobb 478-nál – ill., ld. a 4.7 ábrát, nagyobb 219-nél és ott az `f` eltérések vektora maximumnormája nagyobb, mint 72, most 1 körül, amikor a mérési adatokban látható ingadozás is több, mint 1.
A 4.10 ábra mutatja az eredményünk teljesítményét. Emlékeztetünk, hogy csak az első 16 komponensnek van kihatása. (A bal részábra x-, ill. y-tengelyének nagyságrendje különbözik!)

Néhány modell összehasonlítása

A szakaszosan konstans függvénnyel kapcsolatos óriási számítási idő bizonyosan azzal kapcsolatos, hogy a szakadékok környezete igen pici lépésekre kényszeríti az `ode45` programot, különösen az `mljf` programunk opciói mellett (ld. a 114. oldalon az `epsr=1e-9`–t stb.).

Matematikailag : ezt az `ode45` programot nem ilyen jobboldalra találták ki, alkalmazása a megoldásnak legalább 4 folytonos deriváltját tételezi fel, azaz a jobboldalnak 3 folytonos deriváltját!

Az alábbiakban a célba vett 4 függvényosztályból : szakaszosan konstans, első- és harmadfokú spline, véges Fourier–sor, csak az utolsó felel meg ennek a simasági feltételnek.



4.10. ábra. *A szakaszosan konstans függvényű optimalizálás (egyelőre) végeredménynek elfogadott adatainak ábrája, most* $\|f\| \approx 2.51$ *(amikor* $\|f\| \approx 478.3$*-ból indultunk ki). Baloldalt : a mért és számolt út, jobboldalt : az* $f = xy - xy0$ *vektor komponensei és euklideszi normája.*

Nézzük meg, hogy ez a gyakorlatban, a számítógépen hogyan néz ki : hasonlítsuk össze a 4 függvényosztályt abból a szempontból, hogy velük magának az `ode45`-nek mennyi számítási ideje kell, változatlan opciók mellett. (Az elsőfokú spline és a véges Fourier–sor előállítását mutatjuk a 129., ill. a 125. oldalon.)

A következő táblázatban d a folytonos deriváltak száma, és mutatjuk a különböző függvények,

d=-1 : szakaszosan konstans;

d=0 : elsőfokú spline (vagyis a töröttvonal);

d=2 : harmadfokú spline;

d=∞ : véges Fourier–sor;

a t számítási idejét másodpercekben, amikor n-szer hívjuk az `mljf` m-fájlt az előző szakaszban kapott `phi`-vektor (első 16, ha d>-1) komponenseivel :

n \ d	-1	0	2	∞
100	14.174	6.732	5.833	7.334
200	28.576	13.514	11.667	14.664
400	56.940	26.902	23.349	29.312

A táblázat eredményei különböző n értékekhez mutatják, hogy itt megbízható tendenciáról van szó.

A simább jobboldallal a differenciálegyenletek valóban gyorsabban integrálhatóak az `ode45`–tel. Más szóval, a simaság, azaz a differenciálhatóság, nem csak a matematikában fontos, hanem kihatása van a számítási időre is.

Emellett a (véges) Fourier–sorok esetén a trigonometrikus függvények kiszámításának költsége érezhető.

Figyelmeztetünk, hogy az „eredeti" függvény milyen simasággal rendelkezett, azt nem tudjuk. Ez azt nyomatékosítja, hogy a d-vel jellemzett függvényosztályokból egyet választva, modellt választunk.

Egy gyakorlati esetben lehetnek háttérinformációk a függvényosztályról.

A harmadfokú spline

Ezen spline programozási problémája az, hogy a függvényértékek és a spline-t leíró együtthatók lineáris egyenletrendszeren keresztül állnak kapcsolatban, bár rögzített számú, ekvidisztáns x_i alappontok esetén (16 ilyen pontra gondolunk, amelyekben a $phi(i) = \varphi_i$ függvényértékeket majd az optimalizáló program fogja előírni), a lineáris rendszer A mátrixának LU-felbontását csak egyszer, a számítás legelején, el kell készítenünk, majd eljutatnunk az `mljf` m-fájlba, a függvényértékeket valamint a spline y_i együtthatókat az `mljphi`-be (i=1,. . . ,16).

[32]-ből vesszük az $S(x)$ harmadfokú spline-nak azt a verzióját, amikor az i-edik $[x_i, x_{i+1}]$ intervallumon érvényes $S(x) = s_i(x)$, a perem-intervallumokon $(i = 1, n-1)$ teljesül

$$s_1''(x_1) = 0, \quad s_{n-1}''(x_n) = 0,$$

továbbá, a $h_i = x_{i+1} - x_i$ lépéstávolságok ekvidisztánsok, $h_i = h$, és a spline lokális képlete i=1,. . . ,15-re

$$s_i(x) := \varphi_i v_i(x) + \varphi_{i+1} w_i(x) + [(v_i(x))^3 - v_i(x)]y_i + [(w_i(x))^3 - w_i(x)]y_{i+1}\,.$$

Itt a következő jelöléseket használtuk :

$$v_i(x) := \frac{x_{i+1} - x}{h} = 1 - w_i(x), \quad w_i(x) := \frac{x - x_i}{h}.$$

Ekkor a kapcsolat a $\overline{\varphi} := (\varphi_2, \ldots, \varphi_{n-1})^T$ rész-függvényérték vektor és a $\overline{c} := (c_2, \ldots, c_{n-1}))^T$ rész-együttható vektor között $A\overline{y} = \overline{c}$, míg a teljes vektorok $c = (0, c_2, \ldots, c_{n-1}, 0)^T$, $y = (y_1, \ldots, y_n)^T$. Itt az A mátrix konstans és tridiagonális,

$$A := \mathrm{tridiag}(1, 4, 1) \in \mathbb{R}^{(n-2)\times(n-2)}, \text{ és } c_i := \varphi_{i-1} - 2\varphi_i + \varphi_{i+1}\,.$$

Most nézhetjük ezen képletek programozását. Kezdjük az A mátrixszal és LU-felbontásával, amikor `n` továbbra is az alappontok és keresett paraméterek száma :

```
function [L,U] = spline3(n)
% Szamitja a harmadfoku spline matrixanak LU-felbontasat
```

```
% Peremfeltetelek: s"=0 az els"o es utolso pontban
% Az s spline alappontjai ekvidisztansok
%
% Hivasa :
% [L,U] = spline3(n)
% ahol
%  n - az alappontok szama
%  L,U - az LU-felbontas ritkamatrixai
global L U

n2=n-2;
e=ones(n2,1);
A=spdiags([e,4*e,e],[-1,0,1],n2,n2);
[L,U]=lu(A);
end
```

Ezután ezt az információt el kell jutatnunk az `mljf`-be :

```
function f=mljf(phi)
...
% f=xy-xy0 - a kocsi ered"o - eredeti koordinatai (1-21: x, 22-42: y)
% %{
global L U
n=length(phi);
y=phi;
for k=2:3    % differenciak szamitasa
    for i=n:-1:k
        y(i)=y(i)-y(i-1);
    end
end
y=y(3:n);
y=L\y;y=U\y;
y=[0;y;0];   % elkesz"ultek a spline-egy"utthatok
phi=[phi;y]; % idaig a 3-adfoku spline
% %}
epsr=1e-9;
...
```

Itt tehát a `phi`-vektort a spline-együtthatókkal bővítettük ki, és így jutnak el az `mljphi`-be is.

Ott a spline behelyettesítési értékét készítsük el, de most egyben arra rendezkedünk be, hogy a tényleges számítási intervallum a $[4, 320/9]$ (mert `mljkocsi`-ban és `mljf`-ben korlátoztuk a `p`-vektort [1:21]-re, ld. a 113. oldalon a ***-sort) :

```
function f=mljphi(t,z,phi)
...
else
```

```
   h=284/135;i=min(floor((t-4)/h)+1,15); % h=(320/9-4)/15
   xi=4-h+i*h;
   v=i-(t-4)/h;w=1-v;                    % intervallumunk [4,320/9]
   p=phi(i)*v+phi(i+1)*w+phi(i+16)*(v^3-v)+phi(i+17)*(w^3-w);
end
```

A véges Fourier–sor használata

Itt is 16 paraméterre számítunk és ennek megfelelően `mljphi`-ben $t \geq 4$ esetén a szöget keressük a következő alakban :

$$p(t) = \varphi_1 + \sum_{n=1}^{7} \left\{ \varphi_{2n} \sin n\pi \frac{t-4}{36} + \varphi_{2n+1} \cos n\pi \frac{t-4}{36} \right\} + \varphi_{16} \sin 8\pi \frac{t-4}{36}$$

Az `mljphi` programban ezért szerepelnek a következő sorok :

```
   tpi=pi*(t-4)/36;
   p=phi(1)+phi(16)*sin(8*tpi);              % 16-tagu Fourier-sor
   for n=1:7
       p=p+phi(2*n)*sin(n*tpi)+phi(2*n+1)*cos(n*tpi);
   end
```

A harmadfokú spline használata az optimalizaciós problemánkban

A 122. oldalon közölt táblázatból kiindulva kézenfekvő, hogy a harmadfokú spline-t használjuk függvényként, a szakaszosan konstans függvény helyett, és paramétereit határozzuk meg optimálisan. Várható, hogy ezzel nagyjából a felére csökken a nagy számítási időnk.

Kisebb nyeremény keletkezhet abból, hogy az `mljf` programunk (113. oldal) integrálási intervalluma $[0, 40]$, de a csillagozott sorban a megoldási vektorból a $(\frac{320}{9}, 40]$ részt hagyjuk el (amitől a phi-vektor 16 dimenziós lesz). Az intervallumot mindjárt $[0, \frac{320}{9}]$-re korlátozva, további kb. 11 százalék nyerhető.

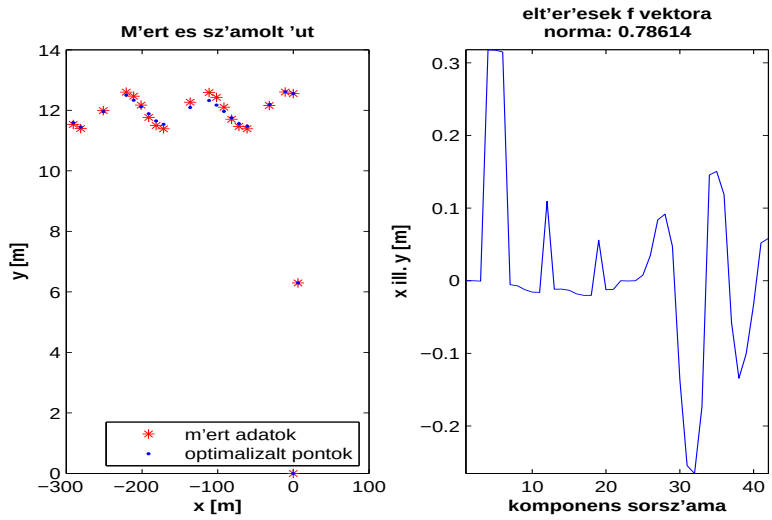
De esetleg lesz még egy további nyereményünk azáltal, hogy a simább jobboldali függvénnyel együtt a probléma célfüggvénye is simább lesz, megkönnyítendő az optimalizálást.

A tényleges számítást most értelemszerűen a

```
phi=linspace(0.0005,0,16)';
```

utasítással beindítva, kapjuk az $||f|| \approx 464.5$ eredményt, de már egy napos futtatás után $||f|| \approx 0.786$, tehát kb. egy negyede a korábbi, több hónapos számítással elért utolsó eredményünknek, vö. a 4.10 ábrát a mostani 4.11 ábrával.

A számítási idő ezek szerint két nagyságrenddel jobban csökkent, mint csak a felére. Ez csak úgy lehetséges, hogy a három fent felsorolt okból a harmadiknak lényeges szerepe volt.



4.11. ábra. *A harmadfokú spline-os számítás egy napos eredménye*

De eközben, a most is alkalmazott érzékenységi vizsgálat során, a gradiens negatív irányú keresésnél, új probléma akadt `ode45`–ben :

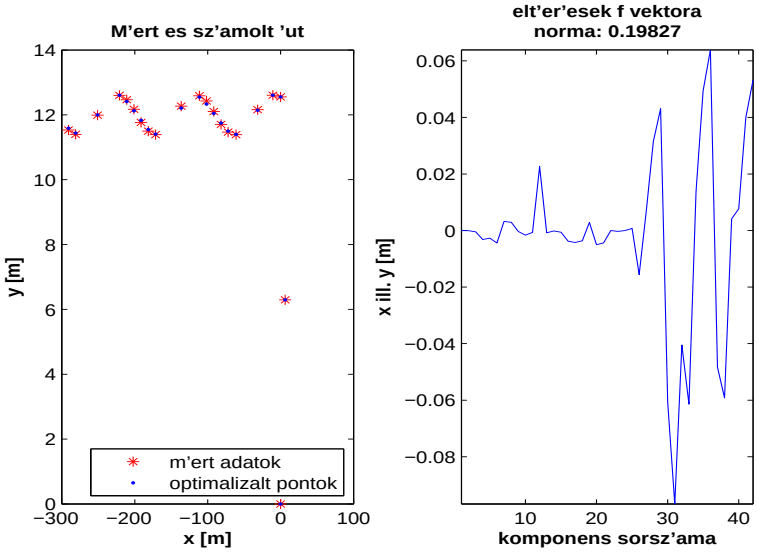
```
Warning: Failure at t=4.005968e+00.  Unable to meet integration
tolerances without reducing the step size below the smallest value
allowed (1.421085e-14) at time t.
 In ode45 at 309
 In mljf at 29
 In goldensearch at 22
```

És ettől megszakadt a számítás. Ennek okaként azt találtuk, hogy ekkor a gradiens normája $||g|| \approx 1e4$, míg a `phi`–komponensek nagyságrendje 1e-3 körül volt. Emiatt `dt0`–ként `goldensearch`–ben (68. oldal) nem vehettünk 1e-2 standard (kezdeti) értékünket (ld. 120. o.), hanem érezhetően kisebbet, pl. 1e-5. Ezzel az említett probléma elhárult.

Felhasználva az `mljopt` program 2. inputparaméterét (kkk, ld. 117. oldalt) és az érzékenységi vizsgálattal együtt kiszámított gradienst, a `kkk=3` esetben az `lb, ub` aktuális indextartományt a mindenkori gradiens néhány, abszolútértékben legnagyobb komponensei alapján határoztuk meg : ezeknek indexeit szabadon hagytuk, tehát ezeket nem tartottuk (lényegében) konstansnak. Hasonló céllal lehet az eltérések `f` vektorából kiindulni, mert a spline együtthatók bár globálisan hatnak ki, de a legerősebben a saját helyén.

A 0.786 célfüggvényi érték elérése után a célfüggvény további csökkenése érezhetően lelassult. Két napos számítás után a mért és számított pontok maximális eltérése már csak 22 cm körül volt.

Hat napos számítással kaptunk egészen elfogadható eredményt – bár a



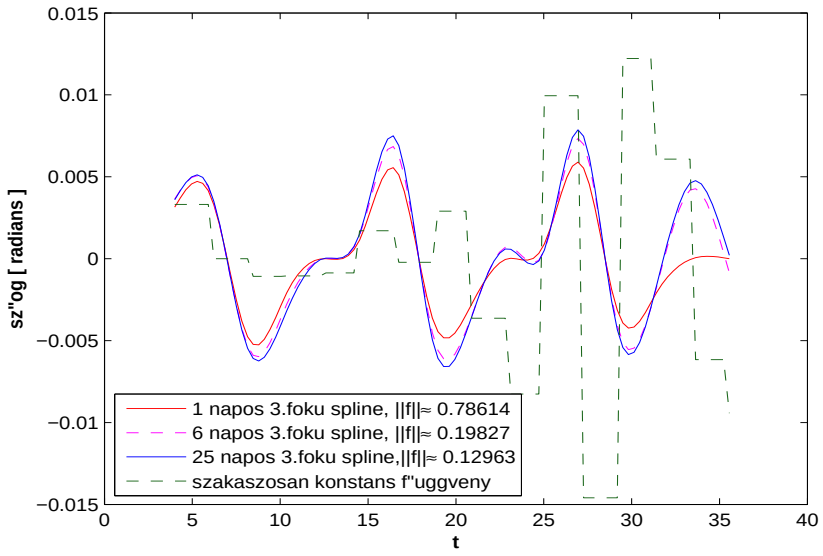
4.12. ábra. *A harmadfokú spline-os számítás hat napos eredménye (emlékeztetésül: az f vektor 1-21. komponense x-koordináta, 22-42. komponense y-koordinata)*

`lsqnonlin` program még nem közölte, hogy „local minimum found" (de már sűrűen azt, hogy „local minimum possible"), és az érzékenységi vizsgálat szerint van ekkor 5 negatív sajátérték: ≈ -1.3367e+08, -4.444e+07, -9.9572e+06, -4.8009e+06 és -6.6027e+05 (amikor a maximális sajátérték ≈ 2.0096e+10). De ne felejtsük el, hogy az ehhez szükséges deriváltakat véges differenciákból kaptuk, ld. a `sensib` programot a 77. és az azt következő oldalon.

Viszont a számított pontok x-komponensei már csak maximálisan 2.3 cm-rel térnek el a mért értékektől, az y-komponensei legfeljebb 9.7 cm-rel, így a megegyezés jónak mondható, ld. a 4.12 ábrát is (ahol jellegzetes eltérés látszik az x- és y-komponensek hibája között, amelyre az y-komponensek nagyobb súlyozásával az `f` vektorban, ld. 114. o., lehet válaszolni. Javasoljuk, hogy az Olvasó ezt tegye meg.). A célfüggvény értéke már a két napos eredménynek kevesebb, mint a fele : $||f|| \approx 0.1983$.

Érdekességként még említjük, hogy a végén a Hesse-mátrix két része, ld. 77. oldalt stb., a következő euklideszi normát adott : $||G|| \approx$ 1.996e+10, $||F|| \approx$ 1.6454e+09, amikor a Gauss-Newton–módszer (ld. [32], 6.5.3.) alapja az, hogy F elhanyagolható G-hez képest. Ez az egész számítás alatt nem állt fenn (leggyakrabban volt $||F|| > ||G||$).

Makacsságból folytattuk a számítást még 19 napig. Ettől bár kaptuk a $||f|| \approx 0.12963$ célfüggvény-értéket, de ekkor a harmadfokú spline ábrája alig különbözött a hat napostól, ld. a 4.13 ábrát, csak csúcsosabb lett a spline, és a maximális eltérést a hat naposhoz képest (≈ 0.001) a végpontban vette fel. A



4.13. ábra. *A harmadfokú spline és (egyelőre) optimális szögei : (egy, hat és húszonöt napos eredmény), és szakaszosan konstans függvény (az ábrához ld. a 23-26. oldalt is)*

kapott `phi`-vektor komponensei :

```
phi=[0.003593845017114; 0.003927436343051;-0.005634133963673;...
    -0.003335343795770;-0.000000211987503; 0.001509381220455;...
     0.007251242694501;-0.005228008581288;-0.003747175734278;...
     0.000618363126633; 0.000527728832149; 0.007749567479152;...
    -0.004606040314916;-0.001939794293025; 0.004723261207415;...
     0.000213235255651]
```

Az egyelőre optimálisnak elfogadott szögeket (a 25 napos számításból) hasonlítsuk össze a 4.13. ábrán a korábbi spline-eredményekkel, és a szakaszosan konstans függvény szögeivel is.

Egyébként a számítás ezen folytatása alatt sem tűnt fel tisztán pozitív spektrum, amivel el lehetett volna elvégezni a (4.20) szerinti szűkebb értelem vett érzénységi vizsgálatot.

Következtetésünk az, hogy jóval korábban abba kellett volna hagyni a számítást a szakaszosan konstans függvénnyel és jobban elgondolkodni a modell választásán.

Az elsőfokú spline használata

A 4.13 ábrán megfigyelhető jelenség, hogy a harmadfokú spline eltérése a mérési adatoktól akkor javul, ha a spline csúcsosabbá válik, arra a gondolatra jutat, hogy ki kellene még probálni a töröttvonalat is. A számítási időket összehasonlító táblázat szerint (122. o.) ez nem járhat túlzottan nagy ráfordítással.

Először megmutatjuk, hogyan állíthatjuk elő az elsőfokú spline-t $t \geq 4$-re, tehát az `mljphi` programunk `else` és `end` közötti sorai mire változnak (míg a $t < 4$ eset sorai maradnak, és nyilván `mljf`-ből a harmadfokú spline-nal kapcsolatos sorokat kommentáljuk ki) :

```
h=284/135;i=min(floor((t-4)/h)+1,15);  % h=(320/9-4)/15
xi=4-h+i*h;
p=((t-xi)*phi(i+1)+(xi+h-t)*phi(i))/h; % az els"ofoku spline erteke
```

A számítást most is beindítottuk a

```
phi=linspace(0.0005,0,16)';
```

utasítással. Ezután addig használtuk az `mljopt` programot, míg tudott javítani az eredményen (ill., ha az `lsqnonlin` utolsó két, `mljopt`-beli hívása már csak $3 \cdot 10^{-6}$ alatti relatív javulást hozott). Ha nem, akkor következett egy érzékenységi számítás. Az, hogy ekkor a megfelelő H Hesse–mátrix (numerikusan) pozitív definit volt, csak háromszor fordult elő. Az első ilyen esetben az

```
f1=norm(mljf(phi-delta))
[phi2,f2]=goldensearch('mljf',phi,-delta,1e-2,1e-8,1e-8)
```

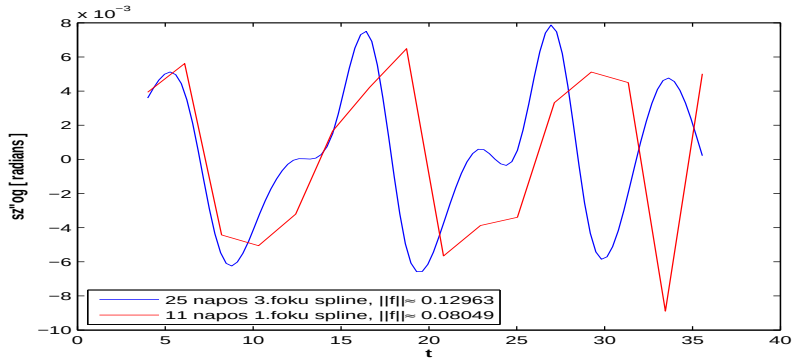
sorokkal döntettük el, vajon a `phi-delta` tiszta Newton–lépés vagy a (4.17) csillapított Newton–módszernek egy lépése volt előnyös, és ennek megfelelően helyettesítettük az aktuális `phi`-vektort `phi-delta`-val, ill. itt : `phi2`-vel. Ez az ||`f`||-érték csökkenését ≈ 0.083973-ról ||`f`||≈ 0.083554-re hozta, ami akkor egy eléggé nagy (≈ 0.00498) relatív nyeremény volt.

Ezután következett egy V-ciklus, majd az `mljopt` stb. A következő alpontban gyüjtettünk össze a további, már itt kikísérelt és alkalmazott észszerűsítéseket.

Amikor az ||`f`||≈ 0.083462 eredményt értük el, az `XTOL` opciót 10^{-12}-ről 10^{-13}-ra állítottuk át, mert az `lsqnonlin` már általában arra hivatkozott kiszálláskor, hogy ezen opció értéke túl nagy. De ez a változás nem sokat segített az `lsqnonlin` eredményességén, több előrehaladást az érzékenységi vizsgálattal kapcsolatos vonalkeresések hoztak.

Később, amikor már ||`f`||≈ 0.0823097 volt, újra lett H (numerikusan) pozitív definit és ekkor az ||`f1`||$\approx$0.0822188 lett, míg az ||`f2`||-t eredményező vonalkeresés nem hozott jobb eredményt, mint ||`f`||-et. Ekkor tehát `phi-delta`-val tovább számítottunk, a relatív csökkenés ≈ 0.001104 lett.

Harmadszor, ||`f`||≈ 0.0816016-kor, a csillapított Newton-lépés volt előnyösebb, amelyhez ||`f2`||$\approx$0.0814043 tartozott és a ≈ 0.002418 relatív csökkenés.



4.14. ábra. *Harmadfokú spline szögei (25 napos eredmény) és elsőfokú spline (töröttvonal) szögei (11 napos eredmény)*

Az a benyomásunk, hogy a töröttvonal használata és a Hesse–mátrix (numerikusan) pozitív definitségének a kihasználása nagyban felgyorsította a számítást :

A végeredmény 11 nap után lett $||\mathtt{f}|| \approx 0.0804874$, azután, hogy úgy az `lsqnonlin`, mint az összes V-ciklusbeli `goldensearch` nem hozott már semmi javulást, és ekkor volt

```
phi=[0.003933679854987; 0.005614400720542;-0.004425670701041;...
    -0.005058964161408;-0.003206490600249; 0.001655708117753;...
     0.004219950985189; 0.006486379486348;-0.005658778381571;...
    -0.003874169035540;-0.003390551529458; 0.003319468291191;...
     0.005119451021992; 0.004496717282768;-0.008889355020975;...
     0.004810894469547]
```

ld. a 4.14 ábrát. Azt is mutatjuk, hogy az ekkor kapott elsőfokú spline szögei, amelyeket egyelőre optimálisnak elfogadjuk, hogyan hasonlítanak a harmadfokú spline 25 napos eredményeihez.

További észszerűsítések

A számítási intervallum átállítása [0,40]-ről [0,320/9]-re csökkenti a számítási időt és két helyen követel változásokat.

a) `mljf`-ben, vö. a 113. oldallal : ott `b` és `ab` megadása változik a következőre :

```
b=320/9;
ab=linspace(a,b,41);
```

b) `mljphi(t,z,phi)`-ben változik az utolsó (13. sorbeli) `else` és a befejező `end` közötti rész, vö. 111. oldallal. Itt most azonnal az összes lehetőségre gondolunk :

```
else
% 16 komponens"unk van
 %{
   h=284/135; % ez (320/9-4)/15, a lepes [4,320/9]-en 15 reszinterv.-hoz
   i=min(floor((t-4)/h)+1,15);
   xi=4-h+i*h;
   p=((t-xi)*phi(i+1)+(xi+h-t)*phi(i))/h; % t"or"ottvonal
 %}
 %{
   h=284/135;
   i=min(floor((t-4)/h)+1,15);
   p=phi(i);                        % szakaszosan konstans fgv
 %}
 %{
   tpi=pi*(t-4)/36;
   p=phi(1)+phi(16)*sin(8*tpi);    % 16-tagu Fourier-sor [0,40]-en
   for n=1:7
       p=p+phi(2*n)*sin(n*tpi)+phi(2*n+1)*cos(n*tpi);
   end
 %}
% %{
   h=284/135;
   i=min(floor((t-4)/h)+1,15);
   v=i-(t-4)/h;w=1-v;               % harmadfoku spline
   p=phi(i)*v+phi(i+1)*w+phi(i+16)*(v^3-v)+phi(i+17)*(w^3-w);
% %}
end
```

A harmadfokú spline esetén gondoljunk arra is, hogy ekkor `mljf`-ben be kell kapcsolnunk a 124. oldalon mutatott, `global L U`-val kezdődő részt.

A `sensib` m-fájlban (77. o.) a nullának tekinthető sajátértékek küszöbe eredetileg

```
epsnm=100*eps*n*m;
```

volt, de ezzel a választással, kombinálva a `nul=find(abs(S)<epsnm);` utasítással csak a zérus sajátértékeket sikerült megkülönböztetni a többi sajátértéktől. Sikeresebbnek bizonyult

```
epsnm=10*eps*max(abs(S));
```

mert így a maximális abszolútértékű sajátértékhez képest elhanyagolható sajátértékek irányai sikerült kizárni a V-ciklusokból. Ezzel is növeltük az optimalizálás hatékonyságát.

További észszerűsítés jelenti, ha a V-ciklus során használt `goldensearch`-hívásokat egy m-fájlba rakjuk (vö. a 120. oldallal) és a `sensib` eredményeinek csak egy részét kinyomtatjuk :

```
function phi=golden(phi,dfac,gfac)
% Szamitja az optimalis phi-vektort minden iranyu arany metszettel.
```

```
%
% Hivasa :
% phi=golden(phi,dfac,gfac)
% ahol
% phi - aktualis es ered"o phi-vektor
% dfac - delta szorzoja
% gfac - g szorzoja
tic
[V,~,neg,pos1,pos2,pos3,~,g,delta]=sensib('mljf',phi,1e-4);
disp(['neg=[',int2str(neg'),'], pos1=[',int2str(pos1'),...
      '], pos2=[',int2str(pos2'),'], pos3=[',int2str(pos3'),']'])
disp(['|g|=',num2str(norm(g)/4),', g:']) % 16 komponensre szamitva
disp(num2str(g,'%1.15g'))
disp('delta:')
disp(num2str(delta,'%1.15g'))

if delta~=-1
   [phi,~]=goldensearch('mljf',phi,-dfac*delta,1e-2,1e-8,1e-8);
end
[phi,~]=goldensearch('mljf',phi,-gfac*g,1e-2,1e-8,1e-8);
[phi,~]=goldensearch('mljf',phi,V(:,neg)*V(:,neg)'*phi,1e-2,1e-8,1e-8);
[phi,~]=goldensearch('mljf',phi,V(:,pos1)*V(:,pos1)'*phi,1e-2,1e-8,1e-8);
[phi,~]=goldensearch('mljf',phi,V(:,pos2)*V(:,pos2)'*phi,1e-2,1e-8,1e-8);
[phi,~]=goldensearch('mljf',phi,V(:,pos3)*V(:,pos3)'*phi,1e-2,1e-8,1e-8);
toc
```

Végül, itt is a kkk-ciklust hasznaltuk `mljopt`-ban, vö. a 114–117. oldalakkal (ahol a `plot`-utasítás előtt persze már a `tt=1:16;` áll). Mint ahogyan több kísérlettel kiderítettük, a jelen (töröttvonalas) esetben a leghatékonyabb volt az alábbi programrészlet ki nem kommentált, „átfedéses” (kkk=1:3-nak megfelelő) verzió :

```
s=3e-8;
switch kkk
    case 1
        lb(9:16)=phi(9:16)-s; ub(9:16)=phi(9:16)+s; % itt kkk=1:3
       % lb(5:16)=phi(5:16)-s; ub(5:16)=phi(5:16)+s;
       % lb(7:16)=phi(7:16)-s; ub(7:16)=phi(7:16)+s;
    case 2
        lb([1:4,13:16])=phi([1:4,13:16])-s; ub([1:4,13:16])=phi([1:4,13:16])+s;
       % lb([1:4,9:16])=phi([1:4,9:16])-s; ub([1:4,9:16])=phi([1:4,9:16])+s;
       % lb([1:4,11:16])=phi([1:4,11:16])-s; ub([1:4,11:16])=phi([1:4,11:16])+s;
    case 3
        lb(1:8)=phi(1:8)-s; ub(1:8)=phi(1:8)+s;
       % lb([1:8,13:16])=phi([1:8,13:16])-s; ub([1:8,13:16])=phi([1:8,13:16])+s;
       % lb(1:8,15:16)=phi(1:8,15:16)-s; ub(1:8,15:16)=phi(1:8,15:16)+s;
    case 4
```

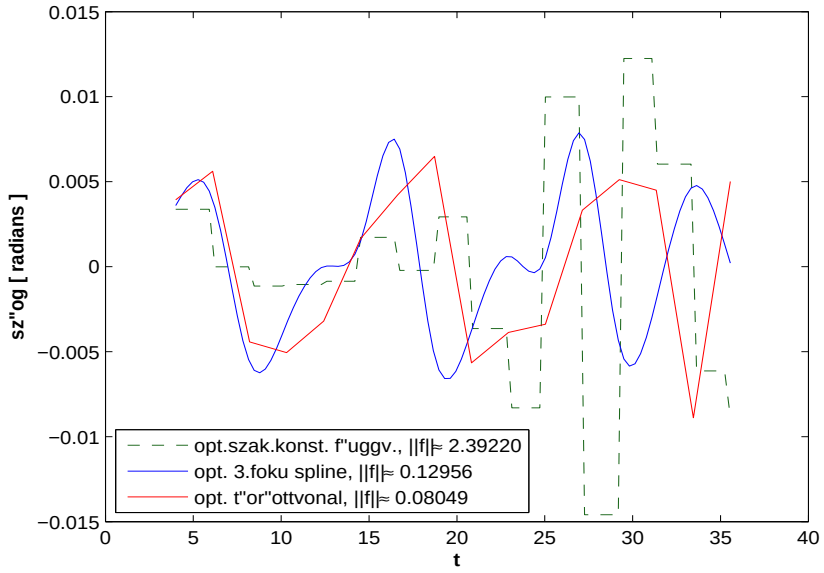
```
       % ez az eset kkk=1:3 hasznalatakor nem lep fel
       % lb(1:12)=phi(1:12)-s; ub(1:12)=phi(1:12)+s;  % itt kkk=1:4
       % lb(1:10)=phi(1:10)-s; ub(1:10)=phi(1:10)+s;
end
```

Az ezzel a részlettel változtatott `mljopt` programot valamint a `golden` programot a következőképpen hasznosítottuk, kiindulva egy `s1`-nevű mat-fájlban lévő kezdeti `phi`-közelítésből és kimentve az eredményt az `s2` mat-fájlba :

```
function kocsiopt(s1,s2)
% kocsiopt: kkk-ciklus es golden hivasa, idomeressel es phi mentesevel
% Hivasa :
% kocsiopt(s1,s2)
% ahol
% s1 - a kiindulasi phi-vektort tartalmazo mat-fajl, sztring
% s2 - az ered"o phi-vektort viszi az s2 mat-fajlba, sztring
load(s1,'phi')
tic;
for kkk=1:3
   disp(['kkk=',int2str(kkk),':'])
   phi=mljopt(phi,kkk);
end;
save(s2,'phi')         % biztonsagi mentes
toc
phi=golden(phi,1,1e-3) % a gfac parameter a gradiens nagysagrendjenek
                       % reciproka
phi=golden(phi,1,1e-3)
save(s2,'phi')
```

A 4.15 ábra hasonlítja össze a 11 napos töröttvonalat a 25 napos harmadfokú spline-nal és az (egyelőre) optimális szakaszonként konstans függvénnyel. Az ábrát a következő m-fájl állítja elő.

```
function spline43(L,U,phi1,phi3,phi0)
% szakasz.kons.f., t"or"ottvonal es harmadfoku spline "osszehasonlitasa
%
% Hivasa :
% spline43(L,U,phi1,phi3,phi0)
% ahol
% L,U - a 16 alapponthoz tartozo LU-felbontas (spline3 eredmenye)
% phi1 - az optimalis t"or"ottvonal sz"ogei
% phi3 - az optimalis harmadfoku spline sz"ogei
% phi0 - az optimalis szakaszosan konstans f"uggveny sz"ogei

n=100;
y3=spline3y(L,U,phi3);[f3,xf3]=spline3f(phi3,y3,4,320/9,n);
```



4.15. ábra. *Az (egyelőre) optimális szögek : harmadfokú spline, szakaszosan konstans és szakaszosan lineáris függvény*

```
y0(1)=phi0(1);j=1;
for i=2:n
    if xf3(i)<4+j*284/135                              % (320/9-4)/15
        y0(i)=phi0(j);
     else
        j=min(j+1,16);y0(i)=phi0(j);
    end
end

plot(xf3,y0,'color',[0.1,0.4,0.1],'linestyle','--') % szak.konstans fgv.
hold on
plot(xf3,f3,'b-',linspace(4,320/9,16),phi1,'r-')    % spline-ok
hold off

xlabel('t','fontweight','bold')
ylabel('sz"og [ radians ]','fontweight','bold')
legend('opt.szak.konst. f"uggv., ||f||\approx 2.39220',...
       'opt. 3.foku spline, ||f||\approx 0.12956',...
       'opt. t"or"ottvonal, ||f||\approx 0.08049','location','southwest')
```

Maga `spline3` már szerepelt a 123. oldalon. Eredményeink alapján egyelőre a töröttvonalat mint optimális modellt fogadhatjuk el, az inverz feladatunk megoldásaként.

Az optimalizálás befejezése

Az előző pontban részletezett programokkal mégegyszer nekiláttunk az inverz feladat megoldásának. Ekkor a szakaszosan konstans függvény (vö. annak `||f||`-értékével a 4.10 ábrán a 128. oldalon), az elsőfokú és harmadfokú spline esetén (tehát a 122. oldal táblázata szerint d=-1,0,2) azt kaptuk, hogy (az ||f|| értékeit kerekítve)

d	-1	0	2
||`f`||	2.3243304	0.0786379	0.0702474

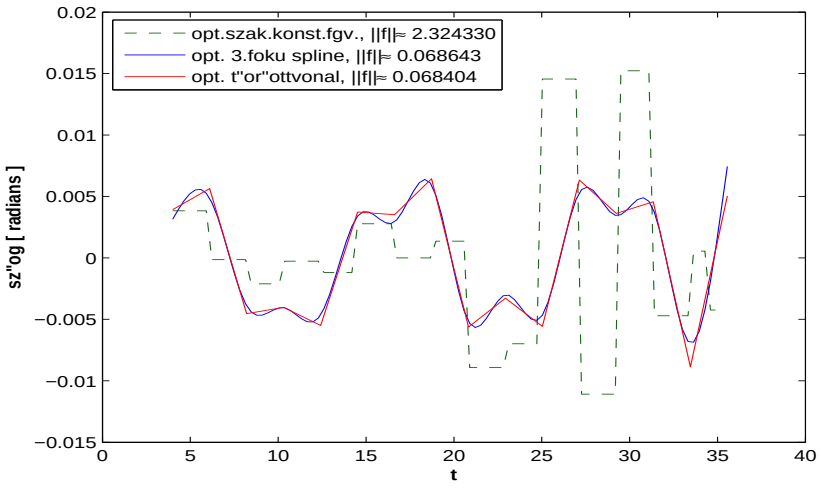
Viszont ezen eredmények birtokában azt találtuk, hogy a spline függvények jobb simasága miatt érdemes két programban az opciókat javítani : `mljopt`-ban a `tolx` opciót 1e-13-ra (a töröttvonalra ezt már előbb jeleztük) és `mljf`-ben az opciókat megadó `epsr` paramétert 1e-10-re csökkenteni. Ekkor többször is kihasználhattuk a Newton–lépést definiáló `delta` vektort, ld. a `sensib` programot. A számításokat mindig addig folytattuk, míg úgy a `kocsiopt`-beli `mljopt` kkk-ciklusa, mint a két `golden`-hívása már nem hozott semmi javulást. Így a következő eredményeket értük el :

d	-1	0	2
||`f`||	2.32433039596	0.06840372768	0.06864319395

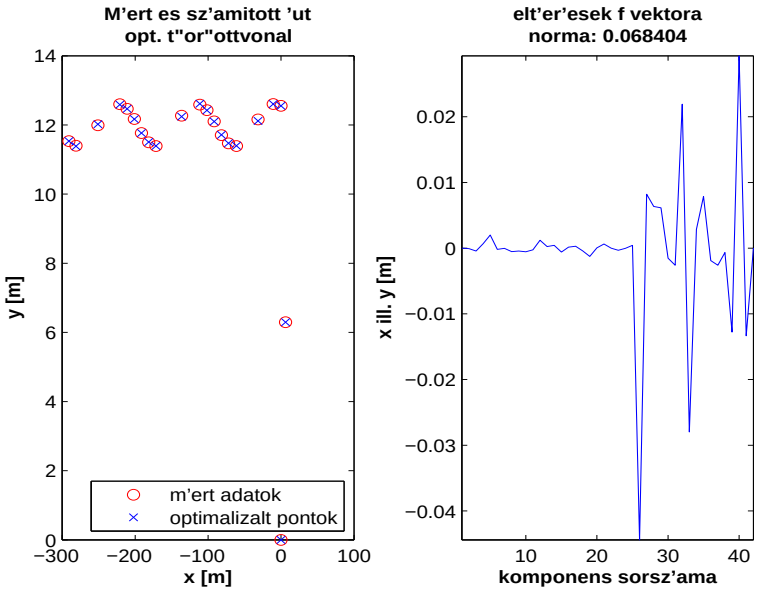
Ez azt jelenti, hogy az elsőfokú és a harmadfokú spline modellje szinte egyenértékű. Közöljük az elsőfokú spline, a töröttvonal optimális `phi`-vektorát, az inverz feladat megoldását :

```
phi =[0.003147829973014; 0.004863450520700;-0.003905648886108;...
     -0.004043098436020;-0.004649109542159; 0.003401827461738;...
      0.002924604690334; 0.005948025800297;-0.005171252748068;...
     -0.003007040144189;-0.004667295480040; 0.005308359295283;...
      0.003433227254492; 0.003937619062295;-0.006955495119365;...
      0.007445706348610]
```

A 4.16 ábra mutatja a három utolsónak vizsgált modell `phi`-függvényét, a 4.17 ábra viszont az optimális töröttvonal teljesítményét.



4.16. ábra. *Az optimális szögek : szakaszosan konstans függvény, elsőfokú és harmadfokú spline*



4.17. ábra. *Az optimális töröttvonalas számítás eredménye*

A 4.17. jobb ábrán látható csúcsok (mindenek előtt a 26. `f`-komponensnél, ami az y-vektor 5. komponense, ahol `f(26)` ≈ -0.0444) a négyzetes normák elég tipikus velejárója. Ha ilyen csúcsokat nem szeretünk, akkor alapvetően a maximumnormához kellene átmennünk, vagy pedig, a Gauss-Newton–módszer keretében (vö. a (6.51) képlettel [32]-ben) `mljf` végén az `f` vektor komponensenkénti páros hatványát képezni, vagy egyszerűen az `f(26)`-ot egy 2-es súllyal megszorozni.

Ettől az utóbbi ötlettől egyébként a maximális eltérés valóban csökken, mégpedig $|\texttt{f(26)}| \approx 0.0381$-re, de a feladat megoldása nehezebb és a végén $||\texttt{f}|| \approx 0.07328791338$ lett (ezen érték kiszámításakor, az optimalizálás után, a 2-es szorzót nem alkalmaztuk, az összehasonlítás végett).

4.4.3. Három modellparaméter meghatározása egy parciális differenciálegyenletben mérési adatokból

Feladatunk az, hogy három pozitív konstanst keressünk, amelyek egy rúd hővezetési tulajdonságaival állnak kapcsolatban. Ezen számok meghatározására $n = 10$ mérési adat áll rendelkezésünkre (ezeket egy $\overline{y}_n^*$ vektorba fogjuk rendezni).

A *hővezetési folyamat* fizikai modellje a hővezétési egyenlet (ld. [34]), amelynek egyértelmű megoldásához a $Q_T := (0, L) \times (0, T]$ derékszögben az alábbi kezdeti- és peremértékeket adjuk meg, mégpedig bal- és jobboldalt 3-adfajú (ott *Newton-féle turbulens hőcsere* zajlik a külvilággal, ld. [33], 11.2.), szakadékos kezdeti függvénnyel (a rúd bal fele meleg : 30 C°, jobb fele hideg : 0 C°):

$$\begin{aligned} \frac{\partial u}{\partial t} &= d\frac{\partial^2 u}{\partial x^2}, \quad 0 < x < L = 1, \ 0 < t \le T; \quad (4.41) \\ d &:= \frac{k}{\rho c_p}; \end{aligned}$$

$$t = 0, \ 0 \le x \le 1 : \qquad u(x,t) = u_0(x,0) := \begin{cases} 30, & 0 \le x \le \frac{1}{2}, \\ 0, & \frac{1}{2} < x \le 1; \end{cases}$$

$$t > 0, \ x = 0 : \ -d\frac{\partial u}{\partial x} + \alpha_0 u = \alpha_0 T_0, \ T_0 \equiv 0; \quad (4.42)$$

$$t > 0, \ x = 1 : \ d\frac{\partial u}{\partial x} + \alpha_1 u = \alpha_1 T_0. \quad (4.43)$$

Itt u a rúd hőmérséklete C°-ban, x a helykoordináta (méterben), t az idő (másodpercekben), k a hővezetési együttható, ρ a közeg sűrűsége, c_p a hőkapacitás (konstans nyomás esetén). d-t hőállandónak nevezhetjük (dimenziója $[m^2/sec]$), α_j-t hőcsereegyütthatónak (j=0,1, $[m/sec]$). Inverz feladatunk a d, α_0, α_1 meghatározása adott hőmérsékleti adatok segítségével.

Sajnos ugyanis úgy van, hogy d összetevőiből a k direkt nem mérhető, és ha a rúd anyaga valójában nem homogén, akkor egy *effektív* hővezetési együtthatót kell használni, amely az anyag konkrét felépítésétől függ. Közben a csereegyütthatókra esetleg ismertek empirikus (nemlineáris!) képletek, amelyek használatához további információ kellene, mint a rudat körülvevő közeg sebessége és hőmérséklete, amelyek nem mindig állnak rendelkezésre.

Fizikailag arról van szó, hogy az 1 méter hosszú rudat a nulla időpontban összerakjuk egy 30 C° meleg és egy 0 C° hideg részből. A peremfeltételekben viszont a külvilág hőmérsékletét a példa kedvéért $T_0 = 0\,C^\circ$-nak vettük, és lehet $\alpha_0 \neq \alpha_1$ (pl. egy épület esetén ez akkor történik, ha a szél erősebb az egyik oldalon, mint a másikon). Továbbá, érvényes $d > 0$, $\alpha_0 \geq 0$, $\alpha_1 \geq 0$ (és ha pl. $\alpha_0 = 0$, akkor a bal oldalon a rúd hőszigetelt).

Mivel nincsenek belső hőforrások és a külső hőmérséklet nulla, azért a test idővel nullára hűl le (kivéve, ha $\alpha_0 = \alpha_1 = 0$, tehát ha a rúd mindkét oldalán hőszigetelt). Ez akkor azt eredményezi, hogy nagy T-re nehéz lesz az inverz feladatot megoldani, a három konstanst meghatározni. Közben a szakadékos u_0 kezdeti függvény jelenléte a tapasztalatok szerint segíti a d paraméter meghatározhatóságát.

Feladatunk most konkrétabban abból áll, hogy írjunk egy MATLAB-programot a fenti ún. „vegyes" (és egyben direkt) feladat közelítő megoldására és használjuk a már ismert `lsqnonlin` MATLAB-programot arra, hogy a d, α_0, α_1 konstansok optimalizálásával az $x_j \in [0,1]$ helyeken az $u(x_j,T) \approx u_j$ számított (közelítő) értékek a mért $\overline{y}_j$ adatokat (ld. lejjebb a táblázatot) minél jobban közelítsék ($j = 1,\ldots,10$).

A feladat része az, hogy $t = T = 0.1$ másodperc esetén biztosítsuk, hogy az $u(x_j,T) \approx u_j$ közelítés elfogadható legyen (pl. abban az értelemben, hogy az eredmények negyedik számjegye legfeljebb 3 egységgel változzon). Továbbá, az `lsqnonlin` opcionális paramétereinek megfelelő választásával és beállításának segítségével próbáljuk elérni, hogy még a nehezebb $T = 10$ másodperc esetben is elfogadható (a $T = 0.1$ értékhez kapott eredményekhez közeli) d, α_0, α_1 konstansokat kapjunk.

Szimulált, tehát előzetes számítással kapott „mérési" eredményekről lesz szó. Ehhez felosztjuk a $Q_T = (0,L) \times (0,T]$ megoldási tartományt $h := \delta x = L/N$, $\tau = \delta t = T/m$ hosszúságú téglalapokra, és a (4.41) egyenletben valamint (4.42)-(4.43) peremfeltételeiben szereplő parciális deriváltakat véges differenciákkal helyesítjük (ld. [33], 11.4. és [34], 16.4.) :

$$\overline{\omega}_h := \{x_i := ih,\ \ i = 0,1,\ldots,N\},$$

a rács felező pontjait $x_{i+1/2} = x_i + \frac{h}{2}$-vel jelöljük és $u(x_i,t)$ helyett u_i-t írunk stb. (A t-től való függésre hamarosan visszatérünk.)

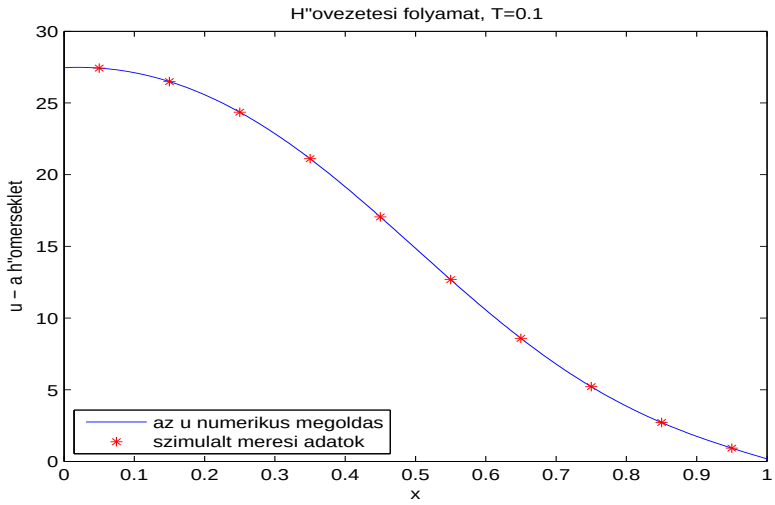
Ekkor a *haladó differenciahányados* képlete

$$\frac{u_{i+1} - u_i}{h} =: u_{x,i} = u_i' + \frac{h}{2}u_i'' + O(h^2). \tag{4.44}$$

Ez a reláció következik a Taylor–formulából:

$$u_{i+1} = u_i + hu_i' + \frac{h^2}{2}u_i'' + \frac{h^3}{6}u_i'''(\vartheta_i h), \quad \vartheta_i \in (0,1),$$

amihez fel kell tenni, hogy $u \in C^3[0,L]$.



4.18. ábra. *A szimulált mérési eredmények*

A térbeli $\overline{\omega}_h$ rács mintájára bevezetjük az $\overline{\omega}_\tau$ időbeli rács ponthalmazát is,

$$\overline{\omega}_\tau := \{t_j,\ j = 0, 1, \ldots, m\}, \tag{4.45}$$

ahol $\tau > 0$ az időlépés. Ekkor a jelölésünk $u(x_i, t_j) =: u_i^j,\ \ x_i \in \overline{\omega}_h,\ \ t_j \in \overline{\omega}_\tau$.

Most a t-től való függésre koncentrálunk. Másodrendű hibatagot a (4.44)-nek megfelelő $(u_i^{j+1} - u_i^j)/\tau =: u_{t,i}^j$ differenciahányadoshoz akkor kapunk, ha a $t_{j+1/2}$ helyén sorba fejtjük a haladó differenciát, és ha t szerint $u \in C^3[0,T]$ (u függőségét x_i-től elhagyva és deriváltját t szerint $\dot{u}$-tal jelölve, stb.):

$$\begin{aligned} u^{j+1} &= u^{j+1/2} + \frac{\tau}{2}\dot{u}^{j+1/2} + \frac{\tau^2}{8}\ddot{u}^{j+1/2} + \frac{\tau^3}{48}\dddot{u}(\zeta_j^+), \\ u^{j} &= u^{j+1/2} - \frac{\tau}{2}\dot{u}^{j+1/2} + \frac{\tau^2}{8}\ddot{u}^{j+1/2} - \frac{\tau^3}{48}\dddot{u}(\zeta_j^-), \end{aligned}$$

ahol $\zeta_j^\pm \in (t_j, t_{j+1})$. Ezért kivonás és τ-val való osztás után

$$u_{t,i}^j = \dot{u}_i^{j+1/2} + \frac{\tau^2}{24}\dddot{u}(x_i, t_{j+1/2} + \vartheta_j\tau), \quad |\vartheta_j| < \frac{1}{2}, \tag{4.46}$$

hiszen az $\dddot{u}$ folytonos, és így van olyan $t_{j+1/2} + \vartheta_j\tau \in (t_j, t_{j+1})$, hogy

$$\frac{1}{2}\left[\dddot{u}(\zeta_j^+) + \dddot{u}(\zeta_j^-)\right] = \dddot{u}(x_i, t_{j+1/2} + \vartheta_j\tau).$$

A (4.44) alatti és (4.46)-beli ϑ-k persze általában különböznek.

Érdemes lesz még egy t szerinti *súlyozás*t bevezetnünk, amelyben $\sigma \in [0,1]$ a súly:

$$u^{(\sigma)} := \sigma u^{j+1} + (1-\sigma)u^j = u^{j+\sigma} + O(\tau^2) = u^{j+1/2} + O(|\tfrac{1}{2} - \sigma|\tau + \tau^2). \tag{4.47}$$

Ez is a Taylor–sor alapján jön ki, ha t szerint $u \in C^2[0,T]$. (4.47)-ben $u^{j+\sigma} := u(x,(j+\sigma)\tau)$.

Még nem foglalkoztunk a (4.41)-beli másodrendű deriválttal. Az x szerinti központi *másodrendű differenciahányados* képlete

$$u_{\overline{x}x,i} = \frac{1}{h^2}(u_{i+1} - 2u_i + u_{i-1}). \tag{4.48}$$

Amennyiben $u \in C^4[0,L]$, akkor a Taylor–formula alapján

$$u_{\overline{x}x,i} = u''(x_i) + \frac{h^2}{12}u^{(4)}(x_i + \vartheta_i h), \quad |\vartheta_i| < 1, \quad 0 < i < N. \tag{4.49}$$

Ha $u \in C^2[0,L]$, akkor csak azt tudjuk, hogy $u_{\overline{x}x,i} \to u''(x_i)$, $h \to 0$, de ha $u \in C^3[0,L]$, akkor $u_{\overline{x}x,i} = u''(x_i) + O(h)$, [34] szerint.

Most minden parciális deriváltat (4.41)-ben és (4.42)-(4.43) peremfeltételeiben differenciahányadossal helyettesíthetünk. Ehhez az alábbi $(N+1)\times(N+1)$-méretű mátrixot vezetjük be ($N+1$ az $\overline{\omega}_h$ rács pontszáma), amelyben a d, α_0 , α_1 konstansok fontosak :

$$(A_h y_h)_i = (A_h(d,\alpha_0,\alpha_1)y_h)_i = \begin{cases} -\frac{2}{h}(dy_{x,0} - \alpha_0 y_0), & \text{ha} \quad i = 0, \\ -dy_{\overline{x}x,i}, & \text{ha} \quad 1 \le i \le N-1, \\ \frac{2}{h}(dy_{\overline{x},N} + \alpha_1 y_N), & \text{ha} \quad i = N. \end{cases} \tag{4.50}$$

Itt $y_h = y_h^j$ rögzített j-re az $y_i^j \approx u(x_i,t_j)$ értékek (N+1)-dimenziós oszlopvektora.

A (4.50) mátrix felépítése a következő :

$$A_h := \frac{1}{h^2}\begin{pmatrix} 2d+2h\alpha_0 & -2d & 0 & . & . & . & . & 0 \\ -d & 2d & -d & 0 & . & . & . & 0 \\ 0 & -d & 2d & -d & 0 & . & . & 0 \\ 0 & 0 & -d & 2d & -d & 0 & . & 0 \\ . & . & . & . & . & . & . & . \\ 0 & . & . & . & -d & 2d & -d & 0 \\ 0 & . & . & . & . & -d & 2d & -d \\ 0 & . & . & . & . & . & -2d & 2d+2h\alpha_1 \end{pmatrix}. \tag{4.51}$$

Ez a mátrix tridiagonális, tipikus ritkamátrix, és mivel $d > 0$, $\alpha_0 \ge 0$, $\alpha_1 \ge 0$, a [33]-beli 11.7. lemma alapján pozitív definit, ha teljesül $\alpha_0 + \alpha_1 > 0$. Ezzel egy ún. *M-mátrix*, mert elemeinek előjeleloszlása (főátlója pozitív, mellékátlói negatívak) is megfelelő (ld. [32], 1.3.4.). Ekkor igaz, hogy a $||\cdot||_{0,h}$ „diszkrét" normában (ld. [33], 11.4.6.) és a hozzátartozó skalárszorzatban eleget tesz az

$$(A_h(d,\alpha_0,\alpha_1)y_h, y_h)_{0,h} \ge M(\alpha_0,\alpha_1)||y_h||_{0,h}^2,\ M(\alpha_0,\alpha_1) := 1\Big/\left(1 + \frac{1}{\alpha_0+\alpha_1}\right)$$

becslésnek, h-tól függetlenül, ami $||A_h^{-1}||_{0,h}$ korlátosságát és a stabilitást jelenti.

Ezután az ún. *súlyozott differenciaséma* képletei (ha (4.41)-et még a nagyobb használhatóság kedvéért egy $f(x,t)$ jobboldali függvénnyel egészítjük ki, amelyet a $t_{j+\sigma} = (j+\sigma)\tau$ pillanatban értékeljük ki), az y_h keresett vektorral :

$$\begin{aligned} y_{t,i} + (A_h(d,\alpha_0,\alpha_1)y_h^{(\sigma)})_i &= \varphi_i^j\,, \quad j = 0,1,\ldots,m-1, && (4.52)\\ \varphi_i^j &= f_i^{j+\sigma}\,, \quad i = 0,1,\ldots,N-1, && (4.53)\\ \varphi_0^j := f_0^{j+\sigma} + \frac{2}{h}g_0^{j+\sigma},\ \varphi_N^j &:= f_N^{j+\sigma} + \frac{2}{h}g_1^{j+\sigma}, && (4.54)\\ y_i^0 &= u_0(x_i), \quad 0 \le i \le N. && (4.55) \end{aligned}$$

A súly speciális értékei esetén a (4.52)-(4.55) súlyozott sémára a következő elnevezések használatosak:

$$\sigma = \begin{cases} 0, & \textit{explicit séma, avagy explicit Euler–módszer},\\ \frac{1}{2}, & \textit{Crank-Nicolson–séma},\\ 1, & \textit{(tisztán) implicit séma, avagy retrográd Euler–módszer}. \end{cases}$$

Ha (4.52)-(4.55)-be az u pontos megoldást helyettesítjük be, akkor általában az egyenletek nem teljesülnek. Az eltérést *képlethibá*nak hívjuk. A (4.44)-(4.49) relációk alapján a

$$\psi_i^j := u_{t,i} + (A_h(d,\alpha_0,\alpha_1)u_h^{(\sigma)})_i - \varphi_i^j$$

képlethibára azt kapjuk, hogy

$$\psi_i^j = O(\tau|\sigma - \tfrac{1}{2}| + \tau^2 + h^2), \qquad (4.56)$$

ha $u \in C^{4,3}(\overline{Q}_T)$.

Ennek speciális esetei, hogy a *Crank–Nicolson séma* képlethibája $O(\tau^2+h^2)$, ha $u \in C^{4,3}(\overline{Q}_T)$, míg az explicit és implicit séma képlethibája $O(\tau + h^2)$, és ehhez elegendő, ha $u \in C^{4,2}(\overline{Q}_T)$. A képlethiba nagyságrendje megegyezik a séma pontosságának nagyságrendjével, ha a séma stabil, ld. [34].

A séma használatához fontos tudni a következőket:

1. Az explicit esetben be kell tartani a $\tau \le h^2/(2d)$ eléggé korlátozó (stabilitási) feltételt. Ha ezt nem tesszük, akkor eredményeink heves oszcillácókat mutathatnak, és gyorsan túlcsordulással végződhet a program futása. Ez különösen bosszantó a mi esetünkben, hiszen d-t nem is ismerjük.

2. A Crank-Nicolson séma az egyetlen másodrendű ebben a séma-családban, így gyakran használják. Viszont nálunk a másodrendűség folytonossági előfeltételei nem teljesülnek, mert az u_0 kezdeti függvény szakadékos. Ekkor előfordulhat, hogy az eredmények megint oszcillációkat mutatnak, bár azok ebben az esetben idővel csillapodnak, mert a séma azért stabil.

3. Az implicit séma nemcsak stabil, hanem oszcillációkat sem produkál. Sajnos csak elsőrendű.

4. A szakadékos kezdeti függvény jelenlétében a számítást két részben kell megvalósítani : 2 lépést az implicit eljárással tesszük, aztán a többit a Crank-Nicolson sémával. (Ez nem tapasztalati trükk, hanem matematkai eredmény : ekkor nem lesznek oszcillációk. Ha egyébként ezt a módszert alkalmazzuk akkor, amikor a kezdeti függvény nem szakadékos, akkor bár a végső hiba általában valamivel növekszik, de másdrendű marad, mert az implicit Euler–módszer 2 lépése még nem rontja a másodrendűséget.)

Az előzőek alapján a *direkt feladat*ot már meg tudjuk oldani, vagyis : adott d és α-k esetén közelítőleg kiszámítani a megoldást. Ehhez érdemes átmenni a MATLAB-nak jól fekvő ritkamátrixú megfogalmazáshoz.

Elsőnek a (4.52)-(4.55) séma mátrixalakjával foglalkozzunk, amelyben a (4.50)-(4.51) mátrix a lényeges szerepet játszik. Ehhez a sémát τ-val szorozzuk, majd rendezzük a keresett y_h^{j+1} vektorral kapcsolatos tagokat a bal, a többi (ismert) tagokat a jobb oldalra:

$$(I_h + \sigma\tau A_h)y_h^{j+1} = (I_h - (1-\sigma)\tau A_h)y_h^j + \tau\varphi_h^j =: r_h\,.$$

Ha adottak a d, α_0, α_1 konstansok, akkor innen az y_h^{j+1} vektort a MATLAB segítségével azonnal ki tudjuk számítani, és ehhez az időciklus előtt egyszer az $I_h + \sigma\tau A_h$ mátrix *LU-felbontás*át állítjuk elő, majd a cikluson belül írjuk $y = L\backslash r_h$, $y = U\backslash y$. Akkor $y = y_h^{j+1}$ lesz a keresett új vektor. (Az $I_h + \sigma\tau A_h$ inverze bár létezik, hiszen $\sigma\tau \geq 0$ miatt A_h-val együtt $I_h + \sigma\tau A_h$ is M-mátrix, sőt még akkor is az, amikor $\alpha_0 = \alpha_1 = 0$ miatt A_h szinguláris, de mivel $\sigma\tau > 0$, $\alpha_0 + \alpha_1 > 0$ esetén telt az inverz mátrix, kiszámítása általában hátrányos.)

Az $y_h^j = y$ vektort egyébként mindig felül lehet írni, mert csak a végső, $t = T$-nek megfelelő y_h^m érték érdekel.

A direkt feladat megoldása most világos. Ne felejtsük el, hogy $j = 1$, 2-re $\sigma = 1$-et kell választanunk, utána pedig $j = 3, \ldots, m$-re $\sigma = 0.5$-öt.

Az így kapott y_h^m vektort az

$$R_{N+1}^n : \quad \mathbb{R}^{N+1} \to \mathbb{R}^n$$

vetítés segítségével az y_h^m-t tartalmazó $\mathbb{R}^{N+1}$ térből az $y_n = (y_1^{(n)}, \ldots, y_n^{(n)})$ vektorba és ezzel az $\mathbb{R}^n$ térbe visszük (amely tér az $\overline{y}^*$ mérési vektort is tartalmazza). Összességben a *direkt feladat* tehát jellemezhető a következő módon:

$$d,\, \alpha_0,\, \alpha_1 \implies y_h^m \implies y_n\,, \quad \text{avagy rövidebben } d,\, \alpha_0,\, \alpha_1 \implies y_n\,.$$

Ezen az úton nyertük az $\overline{y}_n^*$ szimulált mérési eredményeket is bizonyos d^*, α_0^*, α_1^* paraméterekkel (amelyekről csak annyit árulunk el, hogy pozitívak, de nem egészek, hiszen éppen ezeknek meghatározása a feladat). Ekkor viszont különböző h, τ-párokat kellett választanunk úgy, hogy a numerikus úton kapott „mérési" eredmények a korábban említett pontossági követelménynek feleljenek meg (a 4. számjegye már csak 3 egységgel változzon). Kiderült, hogy – a Q_T téglalap diszkretizációját felezéssel „finomítva", (kezdve a $N = 200$, $m = 40$

számokkal, és ezeket folyamatosan duplázva) – $T = 0.1$ esetén ehhez elég $N = 800,\ m = 160$.

Tehát $h = L/800,\ \tau = T/160$ esetén kaptuk a direkt (diszkrét) feladat megoldásával az inverz feladat alapját képező $\overline{y}_n^* = \overline{y}_n(d^*, \alpha_0^*, \alpha_1^*)$ vektort is, amelynek értékeit mutatja a 4.18 ábra és az alábbi táblázat $n = 10$ esetén (a függőséget (h, τ)-tól nem jelöljük) :

T=0.1	x=0.05	0.15	0.25	0.35	0.45
$\overline{y}_n^*$	$\overline{y}_1^{(n,*)}$	$\overline{y}_2^{(n,*)}$	$\overline{y}_3^{(n,*)}$	$\overline{y}_4^{(n,*)}$	$\overline{y}_5^{(n,*)}$
	27.4639	26.5315	24.4104	21.1808	17.1301
	x=0.55	0.65	0.75	0.85	0.95
$\overline{y}_n^*$	$\overline{y}_6^{(n,*)}$	$\overline{y}_7^{(n,*)}$	$\overline{y}_8^{(n,*)}$	$\overline{y}_9^{(n,*)}$	$\overline{y}_{10}^{(n,*)}$
	12.7614	8.65696	5.26658	2.74803	0.936593

Az *inverz feladat*ot vissza fogjuk vezetni direkt feladatok sorozatának megoldására, amire most rátérünk.

A legegyszerűbb ötlet itt is az, hogy minimalizáljuk a

$$\mathcal{J}(d, \alpha_0, \alpha_1) := ||R_{N+1}^n y_h^m(d, \alpha_0, \alpha_1) - \overline{y}_n^*||_2^2 \tag{4.57}$$

funkcionált, amelyben $R_{N+1}^n y_h^m = \overline{y}_n$ és $||\cdot||_2^2$ az $\mathbb{R}^n$-beli euklideszi norma négyzete. Kiindulva egy kezdeti $d^0,\ \alpha_0^0,\ \alpha_1^0$ választásból, egy optimalizálási program fogja megadni a következőnek kipróbálandó $d^k,\ \alpha_0^k,\ \alpha_1^k$ számhármast (ahol $k > 0$ jelöli az optimalizáció lépését), és a kapott $\mathcal{J}(d^k, \alpha_0^k, \alpha_1^k)$ érték alapján továbblép :

$$\ldots\ d^{k-1},\ \alpha_0^{k-1},\ \alpha_1^{k-1} \Longrightarrow \overline{y}_n^{k-1} \Longrightarrow \mathcal{J}(d^{k-1}, \alpha_0^{k-1}, \alpha_1^{k-1}) \Longrightarrow d^k,\ \alpha_0^k,\ \alpha_1^k\ \ldots$$

Mivel (4.57) az $y_i^{(n)} - y_i^{(n,*)}$ számok négyzetösszege, és a (4.41)-(4.43) feladat, valamint a (4.52)-(4.55) séma megoldása nemlineáris módon függ az `x=(d,` α_0, α_1`)` paraméter vektortól, újra a MATLAB `lsqnonlin` programja (tehát a nemlineáris legkisebb négyzetek feladata) lehet alkalmas a $\mathcal{J}(d, \alpha_0, \alpha_1)$ minimalizálására. Hívására több lehetőség van, pl.:

```
[x,resnorm,residual]= lsqnonlin(fun,x0,lb,ub,options)
```

Itt `fun` olyan m-fájl, amely az

$$\mathbb{R}^3 \ni x \ \rightarrow\ y_n - \overline{y}_n^* \in \mathbb{R}^{10}$$

kiszámítását végzi el;

`x0=(d`$^0, \alpha_0^0, \alpha_1^0$`)` tartalmazza a paraméterek kezdeti értékeit;

`lb,ub` két vektor, amely tartalmazza az alsó ill. felső x-korlátjait (nálunk ez `lb=[0 0 0], ub=[]` lesz, mert pozitív `(d,` α_0`,` α_1`)` paramétereket keressük, de felső korlát nincsen);

`options` egy struktúra (opcionális paraméter!), amellyel az `lsqnonlin` futását tudjuk befolyásolni, amire visszatérünk;

`x` a bal oldalon a (remélhetőleg) kisebb $\mathcal{J}$-értéket adó `(d,`α_0`,`α_1`)` eredmények vektora;
`resnorm` a végső $\mathcal{J}$-érték;
`residual` az $y_n - \overline{y}_n^*$ vektor végső komponensei.

Mivel semmi információval nem rendelkezünk a keresett paraméterek nagyságrendjéről sem, az `lsqnonlin` optimalizálási programot először az `x=[1,1,1]` számokkal indítjuk be, abban bízva, hogy a $T = 0.1$ mellett nem lesz annyira nehéz az inverz feladat. Esetleg be kell állítani `optimset`-tel alkalmas opciókat, vagy majd segít a program ismételt hívása.

```
x=ones(1,3);T=0.1;
x=lsqnonlin(@(x) mljhovez(x,T),x,zeros(1,3))
```

Itt a `zeros(1,3)` azt jelenti, hogy a keresett paraméterek legyenek nemnegatívak. Erre a hívásra egyebek mellett a következő információkat is kapjuk:

```
Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to
its initial value is less than the default value of the function tolerance.

<stopping criteria details>
```

Kattintva a „stopping criteria details”-ra, olvassuk:

```
<stopping criteria details>
Optimization stopped because the relative sum of squares (r) is changing
by less than options.TolFun = 1.000000e-06.

Optimization Metric                                   Options
relative change r =   3.24e-07              TolFun =   1e-06 (default)
```

Emiatt mégegyszer hívjuk a programot, most a `TolFun` opciót kisebb értékre állítva (egyben lehetett volna a kapott `x` vektort is beírni az `[1,1,1]` helyett):

```
x=ones(1,3);T=0.1;
opts=optimset('tolfun',1e-10);
x=lsqnonlin(@(x) mljhovez(x,T),x,zeros(1,3),[],opts)
```

Itt a `',[]'` azert szerepel, mert nincsenek felső korlátok, de máskülönben `opts` kell, hogy az `lsqnonlin` 7. paramétere legyen.

Ezen újabb hívás ugyanazokat a paramétereket adja, és ezenkívül azt olvashatjuk a „stopping criteria” alatt:

```
Optimization stopped because the relative sum of squares (r) is changing
by less than options.TolFun = 1.000000e-10.

Optimization Metric                                   Options
relative change r =   7.10e-12              TolFun =   1e-10 (selected)
```

Most térjünk rá a nehezebb $T = 10$ esetre, újabb d^*, α_0^*, α_1^* paraméterek mellett. Hogy a feladat életszerűbb legyen, itt is kiinduljunk az `x=[1,1,1]` kezdeti vektorból.

A feladat sikeres megoldásához a következő tanácsokat tesszük hozzá, amelyek más optimalizálási feladatok megoldásakor is hasznosak lehetnek :

1. A keresett paraméterek ismeretlen nagyságrendje miatt lehet, hogy jó lesz Gauss-Seidel–módon úgy eljárni, hogy ugyan `x=[1,1,1]`-gyel indulunk, de először csak az 1. paramétert optimalizáljuk. Majd ezt tartjuk rögzítettnek a kapott értéknek szűk környezetében (hiszen az alsó és felső korlátok különbözőek kell, hogy legyenek) és a csak a 2. paraméter szerint fut az optimalizálás stb.

2. Amikor az `lsqnonlin` programot már az összes paraméterre alkalmazzuk, javasoljuk, hogy minden futás után nézzük meg a végső információkat, a „stopping criteria"-kat, és esetleg megfelelő opciókat állítsunk be.

3. Egy-egy `lsqnonlin`-futás eredményét rögtön használjuk a következő futás kezdeti értékeként, pl. :

   ```
   x=lsqnonlin(@(x) mljhovez(x,T),x0,zeros(1,3),[],opts)
   x=lsqnonlin(@(x) mljhovez(x,T),x,zeros(1,3),[],opts)
   ```

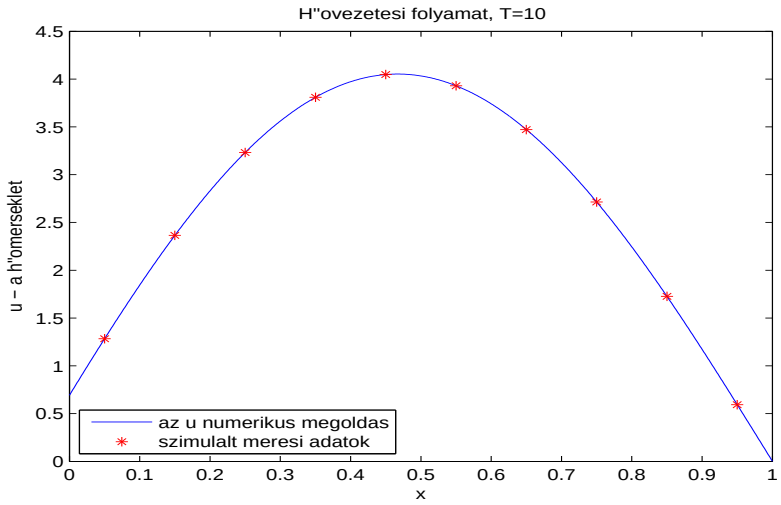
   stb., míg nem jön az „Initial point is a local minimum" jelentés vagy megnyugtatóak a „stopping criteria details".

Megmutatjuk az előző tanácsok hatását, amikor a mérési eredmények előállításához a következő numerikus paramétereket bizonyultak hasznosnak : $h = 1/1600, \tau = 10/320$. Ezzel kaptuk a 4.19 ábrát és az alábbi táblázatot.

T=10	x=0.05	0.15	0.25	0.35	0.45
$\overline{y}_n^*$	$\overline{y}_1^{(n,*)}$	$\overline{y}_2^{(n,*)}$	$\overline{y}_3^{(n,*)}$	$\overline{y}_4^{(n,*)}$	$\overline{y}_5^{(n,*)}$
	1.28425	2.36655	3.23461	3.81212	4.05065
	x=0.55	0.65	0.75	0.85	0.95
$\overline{y}_n^*$	$\overline{y}_6^{(n,*)}$	$\overline{y}_7^{(n,*)}$	$\overline{y}_8^{(n,*)}$	$\overline{y}_9^{(n,*)}$	$\overline{y}_{10}^{(n,*)}$
	3.93313	3.47406	2.71674	1.72845	0.594282

Itt most a következő programrészlet volt sikeres :

```
x=ones(1,3);
op=optimset('tolfun',1e-12);
x=lsqnonlin(@(x) mljhovez(x,T),x,[0,1,1],[inf,1+2e-6,1+2e-6],op)
x=lsqnonlin(@(x) mljhovez(x,T),x,[x(1)-2e-6,0,1],[x(1)+2e-6,inf,1+2e-6],op)
x=lsqnonlin(@(x) mljhovez(x,T),x,[x(1)-2e-6,x(2)-2e-6,0],...
            [x(1)+2e-6,x(2)+2e-6,inf],op)
x=lsqnonlin(@(x) mljhovez(x,T),x,zeros(1,3),[],op)
```



4.19. ábra. *Szimulált mérési eredmények T=10-re.*

Míg az 1-3. `lsqnonlin` hívás eredménye volt

```
x =
   0.016636953039027   1.000000000000022   1.000000999999978
x =
   0.016637953039005   0.764572395444631   1.000000999999978
x =
   1.0e+03 *
   0.000016636953039   0.000764571559082   3.347014480445593
```

a 4. hívás már teljes sikerrel végződött:

```
x =
   0.018315638928425   0.318309887941509  97.408848091357129

<stopping criteria details>

Optimization stopped because the relative sum of squares (r) is changing
by less than options.TolFun = 1.000000e-12.

Optimization Metric                                  Options
relative change r =   1.07e-17              TolFun =   1e-12 (selected)
```

Eredményünk eltérése a (szerző számára ismert) helyes értékektől :

$$dx = 10^{-3} * (0.000000039691,\ 0.000001757718,\ 0.242942645286).$$

Ezekután kézenfekvő ezt összehasonlítani az érzékenységi vizsgálattal (vö. 77., 119. o.), amelyhez nem kell a helyes eredmény ismerete. A `sensib` program hívásában `x` az inverz feladatunk fenti eredménye és 10 a `T` aktuális értéke :

```
 [~,~,~,~,~,~,~,~,d]=sensib(@(z) mljhovez(z,10),x',1e-4)
A Hesse-matrix 2 reszenek normai: ||G||=4.8679e+05, ||F||=0.0018665
A negativ sajatertekek:
A nulla k"or"uli sajatertekek:
A pozitiv sajatertekek: 7.8587e-10     8.1822 4.8679e+05

d =
   1.0e-03 *
   0.000000039644404
   0.000001757683053
   0.242972200546089
```

Az itt kapott, (4.20) szerinti `d` hibaindikátor több számjegyre egybeesik a fenti dx vektor komponenseinek értékeivel. Az `x` vektor az `lsqnonlin` újabb hívásával vagy a (4.19)-alakú ciklussal még javítható.

4.4.4. Parciális differenciálegyenletekben szereplő függvények meghatározásáról mérési adatokból

Mint ahogyan azt a 4.4.2. pont alapján sejthetjük, parciális differenciálegyenletek esetén egy függvény meghatározása nehéz lehet. Sajnos többről van szó : rendszerint az ilyen feladat *inkorrekt*, vagyis ilyen függvény vagy nincs, vagy végtelen sok van, vagy van, de az nem függ stabil módon az adatoktól.

Közben az ilyen feladatok igen érdekesek lehetnek. Pl. : határozzuk meg a Föld 4.5 milliárd évvel ezelőtti hőeloszlását, ismertnek tekintve a mai hőeloszlását és, a Föld külső peremén, a hőáramokat !

Ekkor matematikailag arról van szó, hogy a hővezetési egyenlet kezdeti értékét akarjuk meghatározni egy későbbi állapotból.

Ezt le is programozhatjuk, akár az egydimenziós esetben, és közelítőleg megpróbálhatjuk megoldani, mint a 4.4.3. pontban, véges differenciák segítségével, de eredményként hatalmas oszcillációkat fogunk kapni, vagy túlcsordulást.

Ezzel a témakörrel elsőként Andrej Nyikolajevics Tyihonov foglalkozott sikeresen (az angol irodalomban Tikhonov-nak írják), és két megállapításra jutott :

A keresésre megengedett (végtelen) F függvényteret egy G kompakt részhalmazra kell korlátozni (pl. véges sok függvény lineáris kombinációira, vö. a 4.4.2. ponttal), és a szokásos

$$J(f) := ||f - u||^2 \ \rightarrow \ \min_{f \in G}$$

legkisebb négyzetek funkcionálját (amelyben u a mérési adatokat reprezentálja és $||\cdot||$ az L_2-norma) ki kell egészíteni egy „regularizációs" taggal :

$$J_\alpha(f) := ||f - u||^2 + \alpha ||f||_1^2 \ \rightarrow \ \min_{f \in G} \qquad (\alpha > 0),$$

ahol $||\cdot||_1$ legtöbbször az L_2-norma, vagy pedig $||f||_1^2 = (-\Delta f, f)$, a Δ (diszkrét) Laplace–operátorral. Ezt az eljárást Tyihonov–regularizációnak hívják.

Nyilván az eredmény α-tól függ, és annak kiválasztására sok dolgozat készült. Ezt a „regularizációs paramétert" leginkább az u adatok pontosságával kell kapcsolatba hozni.

A további részletekért az irodalomra hivatkozunk, numerikus vonatkozásban ld. pl. [10], [12]-[14], [22], ezenkívül arra mutatunk rá, hogy parciális differenciálegyenletekből sok van, még ha csakis másodrendűekre korlátozunk magunkat, mint [34]-ben. Ezeknek mind megvan a saját specialitása, akár inkorrekt, akár direkt vagy inverz feladatok szempontjából nézzük, az utóbbihoz ld. [15].

Irodalomjegyzék

[1] Bernáth A., A kombinatorikus egészértékű programozás ötvenegynéhány éve. Alk. Mat. Lapok 30 (2013), 23–80.

[2] Brandt, A., Livne, O.E., *Multigrid Techniques. 1984 Guide with Applications to Fluid Dynamics. Revised Ed.*. Classics in Applied Mathematics 67. SIAM, Philadelphia 2011.

[3] Calveti, D., Somersalo, E., *Computational Mathematical Modeling. An Integrated Approach Across Scales*. Cambridge Univ. Press, Cambridge 2013.

[4] Cook, W., Fifty-plus years of combinatorial integer programming. In: Jünger, M., et al. (eds.), *50 Years of Integer Programming 1958-2008*. Springer-Verlag, Berlin 2010.

[5] Dantzig, G.B., Thapa, M.N., *Linear programming 1: Introduction*. Springer-Verlag, New York 1997. Dantzig, G.B., Thapa, M.N., *Linear Programming 2: Theory and Extensions*. Springer-Verlag, New York 2003.

[6] Duff, I.S., Koster, J., On algorithms for permuting large entries to the diagonal of sparse matrices. SIAM J. Matrix Anal. Appl. 22 (2001), 973–9996.

[7] Geiger, C., Kanzow, Chr., *Numerische Verfahren zur Lösung von unrestringierten Optimierungsaufgaben*. Springer-Verlag, Berlin 1999.

[8] Geiger, C., Kanzow, Chr., *Numerische Verfahren zur Lösung von restringierten Optimierungsaufgaben*. Springer-Verlag, Berlin 2002.

[9] *GSAT* programcsomag:

http://www.mathworks.com/matlabcentral/fileexchange/40759-global-sensitivity-analysis-toolbox

[10] Golub, G.H., Hansen, P.C., O'Leary, D.P., Tikhonov regularization and total least squares, SIAM J. Matrix Anal. Appl. 21 (2000), 185–194.

[11] Golub, G.H., van Loan, C.F., *Matrix Computations*. John Hopkins University Press, Baltimore, 3rd ed. 1996, 4th ed. 2012.

[12] Hansen, P.C., Regularization tools: A Matlab package for the analysis and solution of discrete ill-posed problems. Numer. Algorithms 6 (1994), 1–39.

[13] Hansen, P.C., regtools: Analysis and solution of discrete ill-posed problems. Version 4.1., 1998, updated 2008.

http://www.mathworks.com/matlabcentral/fileexchange/52-regtools

http://www.imm.dtu.dk/∼pcha/Regutools/

[14] van Huffel, S., Vandewalle, J., *The Total Least Squares Problem. Computational Aspects and Analysis*. SIAM, Philadelphia 1991.

[15] Isakov, V., *Inverse Problems for Partial Differential Equations*. Springer Science+Business Media, New York 2006.

[16] Jazar, R.N., *Vehicle Dynamics. Theory and Application*, 2nd ed., eBook. Springer Science+Business Media, New York 2014.

[17] L.V. Kantorovich: A new method of solving some classes of extremal problems, Doklady Akad. Sci. USSR, 28 (1940), 211-214 (oroszul).

[18] Karmarkar M., A new polynomial time algorithm for the linear programming problem. Combinatorica 4 (1984), 373–395.

[19] Knuth, D.E., *The Art of Computer Programming 1 and 4, 2. fasc.* Addison-Wesley, Reading 1997, 2011. Ford. magyarra 1987 és 2008.

[20] Langtangen, H.P., *A Primer on Scientific Programming with Python*. 3rd ed., Springer-Verlag, Berlin 2012.

Ld. a következő internet címet is a „Python versus MATLAB" témáról:

http://www.pyzo.org/python_vs_matlab.html

[21] LaValle S.M., *Planning Algorithms*. Cambridge University Press, Cambridge 2006.

[22] Lee, G., Fu, H., Barlow, J.L., Fast high-resolution image reconstruction using Tikhonov regularization based total least squares. SIAM J. Sci. Comput. 35,1 (2013), B275–B290.

[23] Lehoucq, R.B., Sorensen, D.C., Deflation techniques for an implicitly restarted Arnoldi iteration. SIAM J. Matrix Anal. Appl. 17, No.4 (1996), 789–821.

[24] MATLAB Newsgroup: `fmincon`: sensitivity analysis. See

http://www.mathworks.com/matlabcentral/newsreader/view_thread/321332

[25] Moler, C., *Numerical Computing with Matlab*. SIAM Books, Philadelphia 2004.

[26] Nocedal, J., Wright, S.J., *Numerical Optimization*. Springer, New York 2000.

[27] Prékopa A., *Lineáris programozás*, Bolyai Társulat, Budapest 1968.

[28] Roos, K., Terlaky, T., Vial, J., *Theory and algorithms for linear optimization. An interior point approach*. Wiley, Chichester 1997.

[29] Rózsa P., *Lineáris algebra és alkalmazásai*. 6. kiadás, Tankönyvkiadó, Budapest 1991.

[30] Saad, Y., *Iterative Methods for Sparse Linear Systems*. 2nd ed., Cambridge Univ. Press, Cambridge 2003.

[31] Samarszkij A.A., Mihailov A.P., *Matematikai Modellezés*. 2. kiadás, FIZMATLIT, Moszkva 2002 (oroszul).

[32] Stoyan G., Takó G., *Numerikus Módszerek I* (alapfogalmak, lineáris és nemlineáris egyenletek, sajátértékek, interpoláció, integrálás, optimalizálás). 3. kiadás, Typotex, Budapest 2010.

[33] Stoyan G., Takó G., *Numerikus Módszerek II* (közönséges differenciálegyenletek). 2. kiadás, Typotex, Budapest 2012.

[34] Stoyan G., Takó G., *Numerikus Módszerek III* (parciális differenciálegyenletek). 2. kiadás, Typotex, Budapest 2011.

[35] Stoyan G., *Numerikus matematika. Mérnököknek és programozóknak*. Typotex, Budapest 2007.

[36] Stoyan G. (szerk.), *Matlab. Frissített kiadás*. Typotex, Budapest 2005.

[37] The MathWorks, *Building a Graphical User Interface*. Natick, Massachusets 1993.

[38] Timothy D.A., Algorithm 832: UMFPACK V4.3 – an unsymmetric-pattern multifrontal method. ACM Trans. Math. Softw. 30, No. 2 (2004), 196–199. Online:

http://www.cise.ufl.edu/research/sparse/umfpack.

[39] van der Vorst, H.A., A fast and smoothly converging variant of BICG for the solution of nonsymmetric linear systems. SIAM J. Sci. Stat. Comp. 13 (1990), 631–644.

[40] Vizváry B., *Egészszámú programozási feladatok*. Typotex, Budapest 2006.

Tárgymutató

Megjegyzés [1]

A

anonim függvény, 111
ascii, 35

B

backslash operátor, 17
bar, 34
Bessel-féle J-függvény, 22
blanks, 15
Branch and Bound, 84

C

C++, 5
cameraposition, 47
célfüggvény, 84
cell array, 15, 26
cellstr, 26
checkcode, 7
colorbar, 46
contourf, 45
Crank-Nicolson–séma, 141
csillapított Newton–módszer, 75

D

deblurring, 66

E

eig, 17
eigs, 18
eps, 63
error, 25
érzékenységi vizsgálat, 75, 76, 119, 147
Euler–módszer, explicit, 141
Euler–módszer, implicit, 141
Examples of GUIDE GUIs, 57
Excel-fájl, 34, 40
eye, 10

F

factoriális, 85
feladat, direkt, 98, 142
feladat, inkorrekt, 147
feladat, inverz, 100, 143
felirat, túl hosszú, 27
figure, 52
find, 79, 95
fontsize, 30
fontweight, 30, 47
Format Cells, 40
format long, 9, 32
Fourier–sor, véges, 123, 125
frame, 49, 54
frame-mátrix, 49
Frobenius–norma, 63, 66

G

gallery, 12
GAMAX, 6
Gauss-féle normálegyenletek, 26, 61, 65

[1] Mivel a MATLAB utasításainál, szakszavainál nem tesz különbséget kis és nagy betűk között, és elfogadja pl. a hivatalos `CameraPosition` mellett a `cameraposition` formát is, az alábbiakban gyakran a kisbetűs forma szerepel. De arra hívjuk fel a figyelmet, hogy a MATLAB változóira ez nem érvényes, pl. `a` és `A` különböző változók.

Gauss-féle normálegyenletek mátrixa, 61
gépi epszilon, 79
get(gca), 48
get(gcf), 48
getframe, 54
golden sectioning, 68
Gomory-vágás, 84
gradiens módszerek, 66
GUI, 57
guide, 57

H

harmadfokú spline, 123
hátizsák feladat, 81
help, 25
Hesse–mátrix, 68
hibaindikátor, 80
Hilbert–mátrix, 17
Householder–mátrix, 12
hővezetési folyamat, 137

I

illesztés, 24
Import Data, 32
Import Selection, 32
inf, 5, 82
input, 15
insert, 47
intlinprog, 81

J

jet, 72

K

kron, 10, 69
Kronecker–szorzat, 89

L

legend, 22
length, 15, 29
lineáris legkisebb négyzetek, 24, 59
lineáris programozás, 84
linewidth, 30
load, 35
location, 30, 46
lsqnonlin, 101, 104, 108, 113, 115, 143, 144
LU-felbontás, 142

M

M-mátrix, 140
markerek, 28
MATLAB egy szabálytalan befejezése, 17
MATLAB-File Exchange Center, 39
mátrix, 9
modellezési hiba, 31
mozgó-ablakos simítás, 34
movie, 52
m-fájl, 5

N

NaN, 5
Newton-féle hőcsere, turbulens, 137
Newton–lépés, 71, 74
Newton–módszerek, 70

O

ode45, 99, 102, 108
ones, 11
optimalizálási toolbox, 81, 101
optimset, 102, 144

P

pathtool, 7
pie, 39
pie3, 44
plot, 21
pointer-függvény, 111
polinomiális illesztés, 28
polyfit, 28
print, 49
Property Editor, 48

Q

QR-felbontás, 61
Q-nélküli QR-felbontás, 26

R

rejtett számjegyek, 62, 120
repmat, 11
reshape, 89
RGB, 22
RGB-vektor, 23, 24
ritkamátrix, 11, 140
rosszul kondicionált mátrix, 76
rot90, 92
rotateticklabels, 39
rotateXLabels, 41

S

save, 32, 37
save-load probléma, 32
script, 37
set(gca), 22, 26, 47, 48
set(gcf), 42
simítás, mozgó-ablakos, 35
Simple GUI, 57
Simulink, 6
size, 13
sparse, 11, 89
speye, 89
sprintf, 26
str2num, 16
strcat, 14
string, 14
subplot, 41, 44
súlyozott differenciaséma, 141
surf, 45
súrlódás modellezése, 98
svd, 65

Sz

szigorú Wolfe-Powell–lépésstratégia, 120
szimplex algoritmus, 85
színek jelölése, 22
színek vezérlése, 23

T

text, 32, 49
text arrow, 47
Tihonov–regularizáció, 148
title, 27
totális legkisebb négyzetek módszere, 65

U

UFget, 12
Uncompressed AVI, 53
Unicode (UTF-8), 40

V

V-ciklus, 80
video, 52
VideoWriter, 52
vlc, 58
vonalfajták, 28

X

xlabel, 27
xtick, 26
xticklabel, 39

Y

ylabel, 27