Dr. Mihálykó Csaba, Pozsgai Tamás:

Scilab programozás és a numerikus algoritmusok Scilab-ban

A felsőfokú informatikai oktatás minőségének fejlesztése, modernizációja

TÁMOP-4.1.2.A/1-11/1-2011-0104

Főkedvezményezett:
Pannon Egyetem
8200 Veszprém
Egyetem u. 10.

Kedvezményezett:
Szegedi Tudományegyetem
6720 Szeged
Dugonics tér 13.

2014

A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Scilab programozás és a numerikus algoritmusok Scilabban
1. előadás
Bevezetés a Scilab program használatába

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

1 Általános tudnivalók

2 Telepítés

3 A Scilab program

4 A SciNotes alkalmazás

5 Scilab használata

Félév felépítése

- Scilab program használata
- Iterációs algoritmusok
- Lineáris egyenletrendszer megoldó algoritmusok
- Közelítő integrálszámítás
- Interpoláció
- Szélsőértékszámítás

Ajánlott irodalom

- Hartung Ferenc, Bevezetés a numerikus analízisbe, Veszprémi Egyetemi Kiadó, Veszprém, 2011
- Molnárka Győző, Gergó László, Wettl Ferenc, Horváth Attila, Kallós Ferenc, A Maple V és alkalmazásai, Springer Hungarica Kiadó Kft, Budapest, 1996
- Stoyan Gisbert, Takó Galina: Numerikus módszerek I-II.,ELTE-TypoTeX, Budapest,1993,1995.

Ajánlott internetoldalak

- www.scilab.org
- hu.wikipedia.org/wiki/Numerikus_analízis

A Scilab program

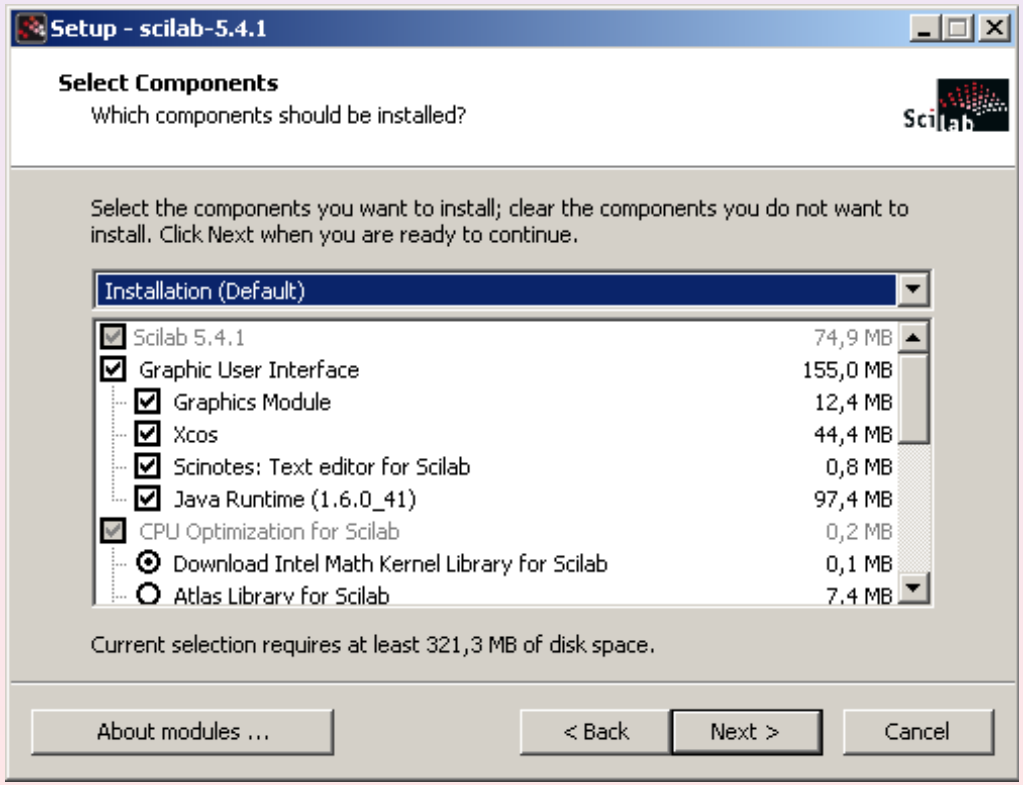
A Scilab az INRIA-nál Franciaországban kifejlesztett a -Matlab-hoz hasonló képességekkel rendelkező ingyenesen terjeszthető matematikai szoftver. A Scilab segítségével mátrixműveleteket végezhetünk, folyamatokat szimulálhatunk, grafikai képességének köszönhetően a folyamatokat megjeleníthetjük, grafikonokon ábrázolhatunk függvényeket 2D és 3D formában.
Letölthető a www.scilab.org internetes oldalról.
A jegyzet elkészítés során Windows 7 operációs rendszerben telepített Scilab 5.4.1. változatot használtunk.

Scilab telepítése

A letöltött scilab-5.4.1_x64.exe fájlt futtatva indul el a Scilab telepítése:

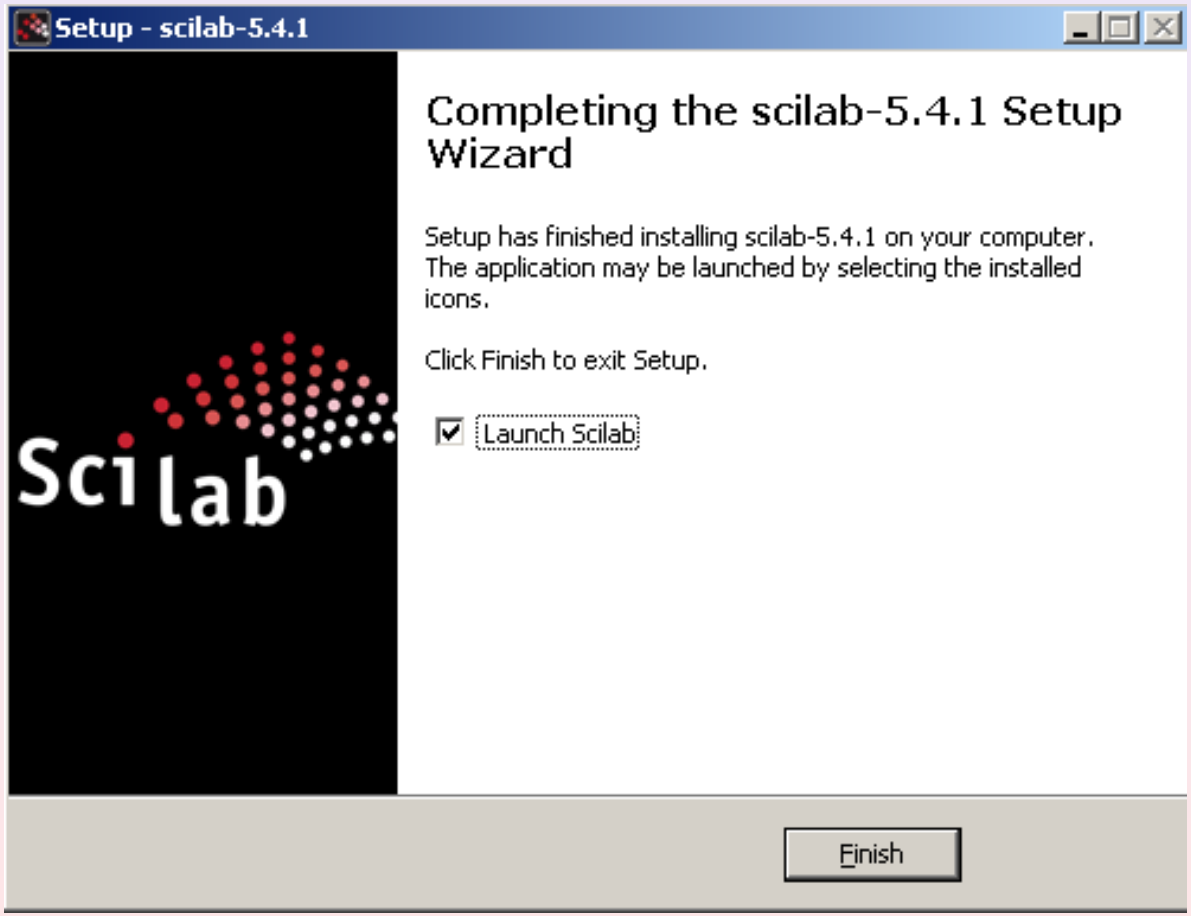
Telepítés

A telepítés sikeres végrehajtása néhány "Next" gombra kattintással történik. A második lépésben választhatjuk ki a számunkra szükséges programkomponenseket.



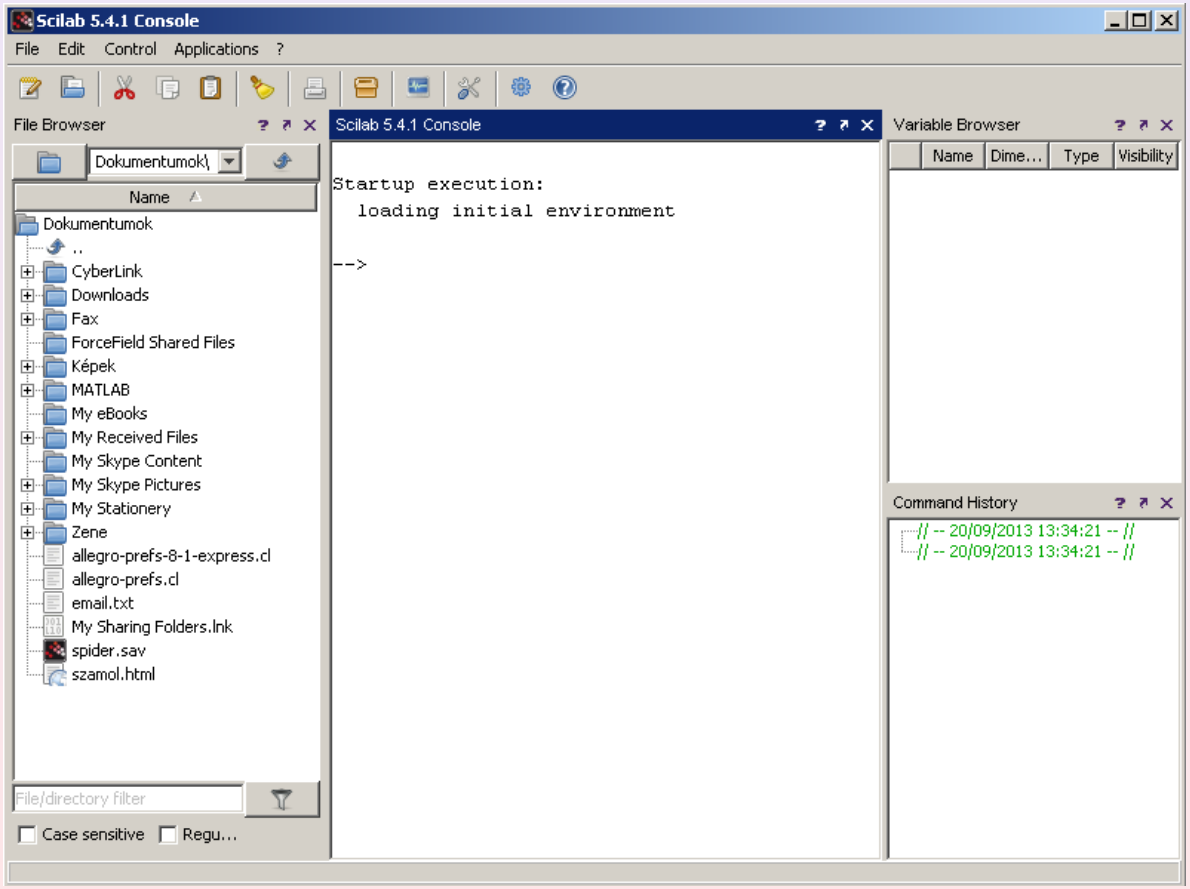
Telepítés befejezése

A sikeres telepítés végén a következő ablakot kapjuk:



Induló képernyő

A program első indításakor a következő ablakot látjuk



Ablakok

A program alapesetben az alábbi négy részre osztódik fel:

- File Browser - az aktuális könyvtár tartalma. Alapértelmezetten a Dokumentumok könyvtár.
- Scilab 5.4.1 Console - konzol ablak, az utasítások lefuttatása, változók létrehozása történhet itt.
- Variable Browser - változók megjelenítése, változtatása táblázatos formában.
- Command History - az előzőleg végrehajtott utasítások listája

Menüsor

A Scilab menüsora az alábbi elemeket tartalmazza:

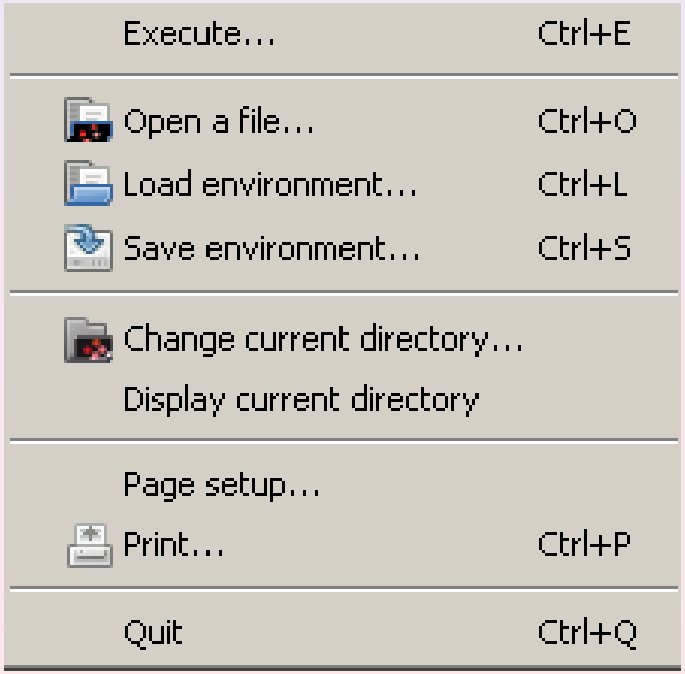
- File
- Edit
- Control
- Application
- ?

File menü

- Execute - végrehajtja a kiválasztott fáljban található utasításokat.
- Open a file - szerkesztésre megnyitja a kiválasztott fáljt.
- Load environment... - előzőleg elmentett Scilab környezetet tölt be.
- Save environment... - elmenti az aktuális környezetet.
- Change current directory ... - megváltoztatja az aktuális könyvtárat.
- Display current directory ... - külön ablakban megjeleníti az aktuális könyvtár tartalmát.
- Page setup - nyomtatási beállítások.
- Print - aktuális környezet nyomtatása.
- Quit - kilépés a programból.

File menü

A menüpontok és a hozzájuk tartozó gyorsbillentyűk a következők:

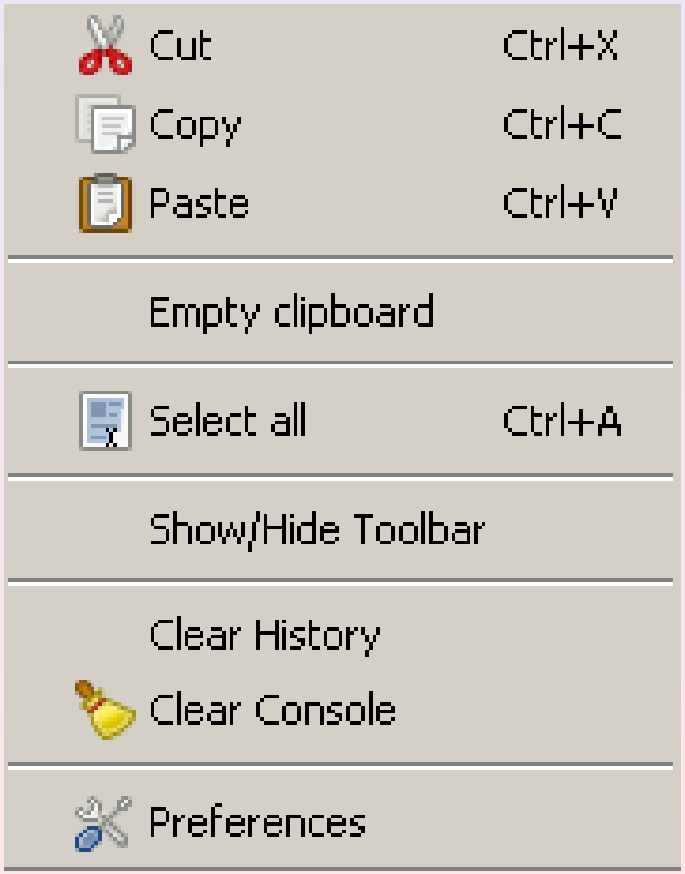


Edit menü

- Cut - kivágja a kijelölt szövegrészt.
- Copy - másolja a kijelölt szövegrészt.
- Paste - beilleszti a kijelölt szövegrészt.
- Empty clipboard- törli a vágólapot.
- Select all - mindent kijelöl a konzolon.
- Show/Hide Toolbar - megjeleníti/elrejti az eszköztárat.
- Clear History - törli az előzményeket.
- Clear Console - törli a konzol tartalmát.
- Preferences - beállítások.

Edit menü

A menüpontok és a hozzájuk tartozó gyorsbillentyűk a következők:



Control menü

A menüben már futó algoritmusok, programok megállítására, újraindítására van lehetőségünk.

- Resume - kiválasztott algoritmus, program futtatása.
- Abort - futtatás leállítása.
- Interrupt - futtatás megszakítása.

Applications menü

- SciNotes - beépített szövegszerkesztő.
- Xcos - tervező alkalmazás.
- Matlab to Scilab translator - Matlab eljárásokat fordít le a Scilab nyelvére.
- Module manager - ATOMS - letölthető modulok kezelése.
- Variable Browser - változók kezelésére szolgáló alkalmazás.
- Command History - az előzőleg végrehajtott utasítások listáját kezelő alkalmazás.
- File Browser - fájlok kezelésére szolgáló alkalmazás.

"?" menü

- Scilab Help - Scilab súgó.
- Scilab Demonstrations - demo programok.
- Links - ajánlott internetes oldalak.
- Scilab Enterprises - fejlesztési weboldalra mutató link.
- About Scilab - a Scilab névjegye.

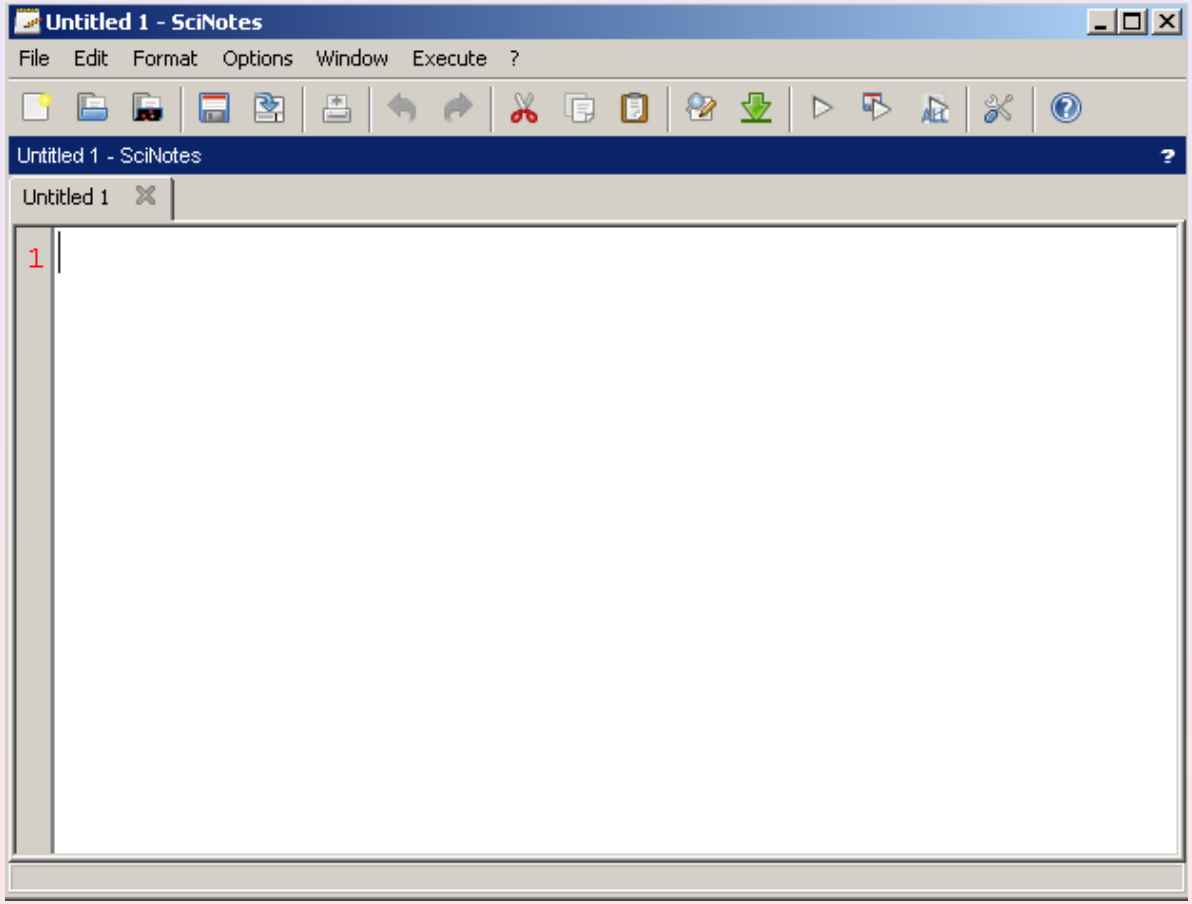
SciNotes alkalmazás

A Scilab beépített szövegkezelő programja a SciNotes.
Segítségével

- az algoritmusok, alkalmazások külön fáljokban készíthetők el.
- betölthetjük a már elkészült eljárásokat, fájlokat.

Indítása a Scilab Applications menüjéből lehetséges, de az Eszköztár első ikonja - az üres lap - is ezt az alkalmazást indítja.

SciNotes üres fájl



File menü

A főbb menüpontok:

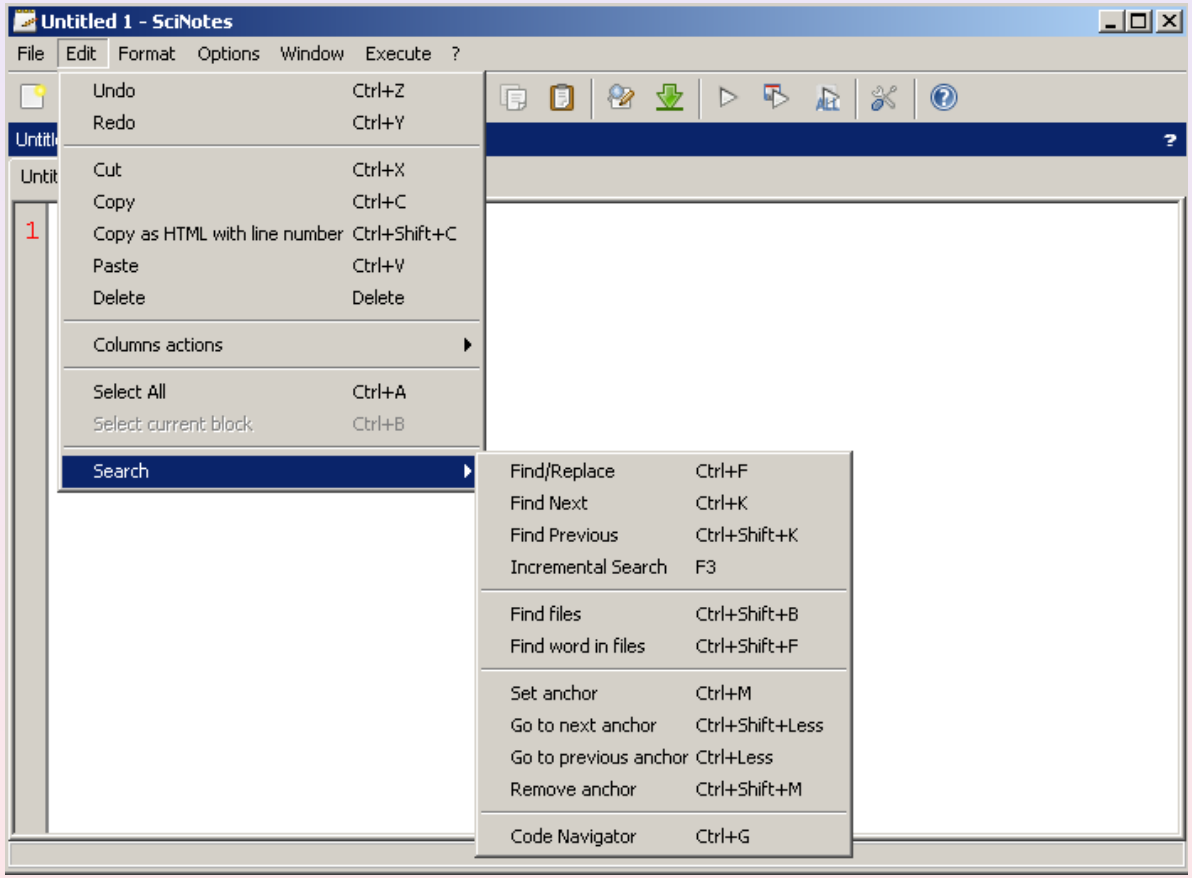
- New - új fájl megnyitása.
- Open - meglévő fájl megnyitása.
- Open function source file - beépített függvények forrásának megnyitása .
- Save, Save as, Save All - fájlok mentésére szolgáló menüpontok.
- Export - a szöveg exportálása .pdf, .ps, .eps, .rtf tipusú fájlokba.
- Print - fájl nyomtatása.
- Close - fájl bezárása.
- Exit Scinotes - kilépés a programból.

Edit menü

A főbb menüpontok:

- Undo, Redo - legutolsó művelet visszavonása, illetve visszaállítása.
- Cut - kijelölt rész kivágása.
- Copy - kijelölt rész másolása.
- Paste - beillesztés vágólapról.
- Delete - kijelölt rész törlése.
- Select All - teljes szöveg kijelölése.
- Search - keresés almenü megnyitása, keresés, csere.

Edit menü


Untitled 1 - SciNotes
File
Edit
Format
Options
Window
Execute
?
Undo Ctrl+Z
Redo Ctrl+Y
Cut Ctrl+X
Copy Ctrl+C
Copy as HTML with line number Ctrl+Shift+C
Paste Ctrl+V
Delete Delete
Columns actions
Select All Ctrl+A
Select current block Ctrl+B
Search
Find/Replace Ctrl+F
Find Next Ctrl+K
Find Previous Ctrl+Shift+K
Incremental Search F3
Find files Ctrl+Shift+B
Find word in files Ctrl+Shift+F
Set anchor Ctrl+M
Go to next anchor Ctrl+Shift+Less
Go to previous anchor Ctrl+Less
Remove anchor Ctrl+Shift+M
Code Navigator Ctrl+G

Format menü

A főbb menüpontok:

- Shift Right - jobbra tabulálás.
- Shift Left - balra tabulálás.
- Remove trailing spaces - felesleges szóközök kivágása.
- Comment Selections - a kijelölt részt megjegyzésbe teszi.
- Uncomment Selections - a kijelölt megjegyzést visszahelyezi a szövegbe.

Options menü

- Current file encoding - megnyitott fájl karakterkódolása állítható be.
- Line endings - sortörés típusa állítható be.
- Preferences - beállítások almenü megnyitása.
  - Set font - betütípus, méret stb. beállítása.
  - Set color - színek beállítása.
  - Scinotes General settings - általános beállítások.
- Auto Competition on - automatikus kiegészítési lehetőségek.
- Line numbering - sorok számozási beállításai.

Window menü

- Copy tag in new window - aktuális fülről másolatot készít új ablakba.
- Detach tab in new window - az aktuális fület új ablakba helyezi.
- Split view - ablakok elrendezése.

Execute menü

- ... file with no echo - betölti a Scilab konzolba az aktuális fájlban található függvényeket, de a fájl tartalmát nem írja ki a képernyőre.
- ... file with echo - betölti a Scilab konzolba az aktuális fájlban található függvényeket, és a fájl tartalmát kiírja a képernyőre.
- Save and execute - elmenti a fájl tartalmát és végre is hajtja a benne szereplő utasításokat.
- Save and execute all files - az összes fül tartalmát betölti és végrehajtja.

"?" menü

- Scinotes Help - SciNotes súgó.
- About - a SciNotes névjegye.

Számológép

A Scilab konzol használható számológépként is.

```
5+4  //összeadás
3*4  //szorzás
4^2  //hatványozás
3!  //faktoriális
```

Beépített függvények

A teljesség igénye nélkül az alábbi matematikai függvények használhatók:

- Logaritmus függvények: `log`, `log2`, `log10`.
- Exponenciális függvény: `exp`.
- Gyökfüggvények: `sqrt`, `nthroot`.
- Trigonometrikus függvények: `sin`, `cos`, `tan`, `cotg`.
- Arcus függvények: `asin`, `acos`, `atan`, `acotg`

Saját függvény definiálása

Konzolon a következő módon hozhatunk létre saját függvényt:

```
deff('y=fgv(x)','y=x^2+3*x-2')
```

Az utasítás hatására létrejön az fgv változóban a függvényünk. Használata:

```
fgv(3)
```

Saját függvény definiálása

Külön fájlban elkészített függvényeket a következő szintaxissal készíthetünk. A fájlt a SciNotes alkalmazással hozzuk létre:

```
function y=fgv(x)
y=x^2+3*x-2
endfunction
```

Az elkészült és elmentett fájlt be kell töltenünk a Scilabba. Ezt a SciNotes Execute menüjében tudjuk megtenni. A betöltés után az "fgv" függvény használható.

Kiíratás

Eljárások készítése során szükség lehet üzenetek elhelyezésére a konzolon. A Scilab alapértelmezésként minden végrehajtott művelet eredményét kiírja. Amennyiben azt szeretnénk, hogy a számolás eredménye ne írodjon ki (például mellékszámítás egy eljárás során), akkor az utasítás után ";"-t kell tenni.
Egyszerű szöveget a `disp` utasítással tudunk kiiratni a konzolra.

```
disp('Egyszerű üzenet')
```

Formázott kiiratás

```
printf('A megoldás %1.14f\n',c)
```

A printf utasítás használatával lehet szöveges környezetben a változókat kiiratni. A fenti utasítás egy változót (c) helyettesíti be a "%1.14" helyére. A változóhelyettesítés lehet (a teljesség igénye nélkül)

- %d - decimálisan kerül ábrázolásra az adott változó
- %1.14f - tizedestört alakban kerül ábrázolásra a változó.

Változó értékadás

Változónak értéket adni az "="-lel lehet.

```
a=2
```

Az alábbi utasítással eljárások futása közben kérhetünk értéket a változónak:

```
a=input('Mi legyen "a" értéke? ')
```

Ekkor az eljárás futása felfüggesztődik az <ENTER> lenyomásáig. A kapott értéket az eljárás használhatja, mint "a" változó értékét.

Scilab programozás és a numerikus algoritmusok Scilabban
2. előadás
Változók, beépített konstansok, vektor-mátrixműveletek

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

1. Változók, konstansok
2. Vektorok
   - Létrehozása, módosítása
   - Műveletek
3. Mátrixok
   - Létrehozása, módosítása
   - Speciális mátrixok
   - Műveletek
   - Szöveges feladat

Változók

- Változónak értéket adni a következőképpen lehet:
  `a = 2`
- Az érték tetszőleges adattípusú lehet (egész, racionális, valós, komplex, vektor, mátrix, sztring, logikai vagy algebrai kifejezés).
- Kis- és nagybetűérzékeny
- Nem típusos változók, tetszőleges adattípust értékül adhatunk függetlenül attól, hogy a változó kapott-e már értéket korábban.
- Vigyázat, 'nem bolondbiztos'!

Beépített konstansok

- %e - természetes alapú logaritmus alapja
- %i - képzetes egység
- %pi - π
- %inf - végtelen
- %nan - 'not a number'
- %t - logikai igaz
- %f - logikai hamis

Vektorok létrehozása

A Scilab program megkülönbözteti a sor és oszlopvektorokat.
Sorvektor létrehozása elemek megadásával:

```
a=[2,4,6,-1]
```

Oszlopvektor létrehozása elemek megadásával:

```
b=[1;1;2;3;5]
```

Üres vektor megadása:

```
c=[]
```

Vektorok bővítése

Az "a" sorvektor bővítése:

```
a=[5,a,-2,4]
```

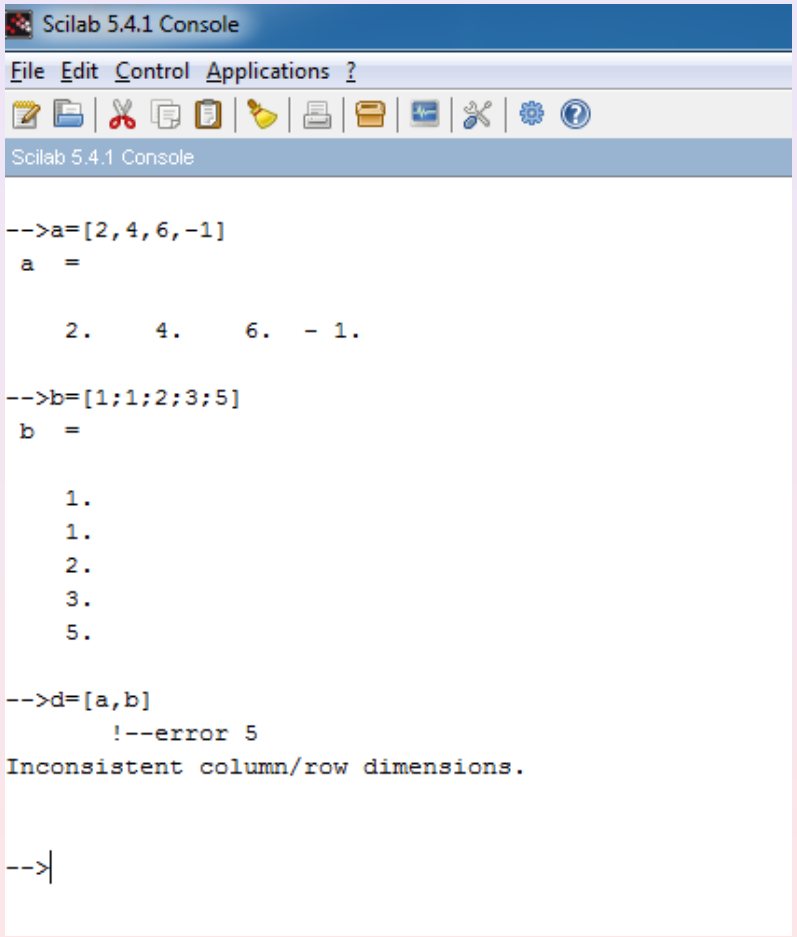
Többszörözni is lehet a vektorokat:

```
b=[b;b]
```

Sor és oszlopvektor összefűzése hibát eredményez:

```
d=[a,b]
```

Vektorok bővítése



Vektorok elemei

Vektorok elemeire a pozíciója segítségével tudunk hivatkozni. A számozás „1”-gyel kezdődik. A következő utasítás a vektor harmadik elemét adja eredményül:

```
b(3)
```

Megváltoztatni a vektor egyetlen elemének értékét a következőképpen lehet:

```
b(2)=-3
```

Vektorbővítés

Amennyiben nem létező vektorelemnek akarunk értéket adni, hiba helyett a program végrehajtja az értékadást, a hiányzó elemeket 0 értékkel tölti fel. Az alábbi utasítás eredménye egy 10 hosszú vektor, melynek az első hat értéke az eredeti érték, a $7-8$. helyen 0 érték áll, az utolsó helyen pedig a kapott érték.

```
b(10)=8
```

Részvektor képzése

Az alábbi utasítás a vektor első két eleméből képzett részvektort adja eredményül:

```
b(1:2)
```

A vektor elemeinek értéke nem változott meg.

Amennyiben változtatni szeretnénk, akkor értékadó utasítással tehetjük ezt meg (a „b” vektor oszlopvektor):

```
b(1:2)=[-6;8]
```

Elemek törlése

Elem törlése a vektorból értékadással történhet. Az üres vektort adjuk értékül az elemnek.

```
b(8)=[]
```

Természetesen részvektornak is adhatjuk az üres vektort értékül.

```
b(3:5)=[]
```

Speciális vektorok

Természetes számokból képzett vektort készíthetünk a következőképpen:

```
f=1:5
```

Számsorozatból is készíthetünk vektort. A következő példa a $[0;1]$ intervallumot osztja fel $0,01$ hosszú intervallumokra és készít vektort.

```
g=0:0.01:1
```

Vektorműveletek

Az algebrából megismert szabályokat a program megköveteli. Csak a megfelelő vektorokkal lehet elvégezni a műveleteket (legyen a sorvektor, b oszlopvektor).

- `a'` - transzponálás
- `a+b` - hiba
- `a*b` - megfelelő méretű vektorok esetén végrehajtja a szorzást, eredmény egy mátrix lesz
- `a.*b` - koordinátánkénti szorzás
- `a^2` - hiba
- `a.^2` - koordinátánkénti hatványozás
- `length(a)` - vektor hossza

Mátrixok létrehozása

Megadhatunk mátrixot az elemeit sorfolytonosan felsorolva:
`A=[1,2,3;6,5,4]`
Eredménye egy 2×3-as mátrix lesz:

$$\begin{pmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \end{pmatrix}$$

Mátrix létrehozás vektorokkal

Legyen a sorvektor, b oszlopvektor. Ezekkel az előre elkészített vektorokkal létrehozhatunk mátrixokat:

`A=[a;a;a]`, illetve `B=[b,b,b]`

Természetesen részvektorokból is képezhető mátrix:

`A=[a(1:3);a(2:4);a(3:5)]`, illetve

`B=[b(3:5),b(2:4),b(1:3)]`

Mátrix elemei, részmátrix

Legyen A egy 3×3-as mátrix. Ekkor az A hatodik eleme a második sor harmadik eleme lesz. Ez azt jelenti, hogy ebben az esetben `A(6)` ugyanaz az eleme lesz a mátrixnak, mint az `A(2,3)`.
Részmátrixot a vektorokhoz hasonlóan tudunk készíteni:
`A(1:2,2:3)` részmátrix az eredeti mátrix első két sorának második és harmadik eleméből képzett mátrix.
Amennyiben a mátrix valamely sorának (oszlopának) egészére van szükségünk, akkor a következő utasítást használhatjuk:

- `A(2,:)` - a mátrix második sora
- `A(:,1:2)` - a mátrix első két oszlopa

Mátrixbővítés

Matematikai értelemben mátrixot csak teljes sorral (oszloppal) lehet bővíteni. Amennyiben nem így bővítünk, a program 0-val tölti fel a hiányzó értékeket.
Legyen A egy 3×3-as mátrix. Hajtsuk végre egymás után a következő utasításokat:

- `A=[A;A(2,:)]` - teljes sorral bővítünk
- `A=[A,A(:,1:2)]` - két teljes oszloppal bővítünk
- `A(5,5)=3` - A mátrix 5. sor, 5. eleme az adott érték, az 5. sor többi eleme 0 lesz.

Törlés mátrixból

Mátrixból törölni csak teljes sort (oszlopot) tudunk. Minden más esetben hibát jelez a program.

- `A(2,:)=[]` - a mátrix második sorának törlése
- `A(:,1:2)=[]` - a mátrix első két oszlopának törlése

Egységmátrix

Az alábbi utasítás eredménye egy 3×3-as egységmátrix lesz:

```
eye(3,3)
```

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

Általánosan egy $n \times n$-es egységmátrix a következőképpen adható meg:

```
eye(n,n)
```

Diagonális mátrix

Az alábbi utasítás eredménye egy 3×3-as diagonális mátrix lesz:

```
diag([1,2,3])
```

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

Véletlen mátrix

A `rand()` utasítás a $[0; 1]$ intervallumról vett véletlen számokkal tölti fel a mátrixot. Paraméterként a mátrix méretét kell megadnunk.

```
rand(3,5)
```

Amennyiben egy $[a, b]$ intervallumról vett egész értékekkel szeretnénk a mátrixot feltölteni, a következő utasítást használhatjuk:

```
round((b-a)rand(3,5)+a)
```

Nullmátrix

Az alábbi utasítás végeredménye egy 3×4-as mátrix lesz, melynek minden elem csupa 0 érték:

```
zeros(3,4)
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Konstansmátrix

Konstansmátrixot beépített utasítással csupa egyesekből tudunk létrehozni:

```
ones(4,3)
```

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

Egyestől eltérő konstansmátrixot az egyesekből álló mátrix konstans szorzásával tudunk készíteni:

```
5*ones(4,3)
```

$$\begin{pmatrix} 5 & 5 & 5 \\ 5 & 5 & 5 \\ 5 & 5 & 5 \\ 5 & 5 & 5 \end{pmatrix}$$

Mátrixműveletek

Konstans értékkel való kommutatív műveletek:

- `A+3` - az A mátrix minden eleméhez hozzáad hármat.
- `A*3` - az A mátrix minden elemét megszorozza hárommal.

Mátrixműveletek

Konstans értékkel való nem kommutatív műveletek:

- `A-5` - az A mátrix minden eleméből kivon ötöt.
- `A/4` - az A mátrix minden elemét elosztja néggyel.
- `4/A` - az A mátrixszal megegyező méretű csupa négyesből álló mátrix minden elemét elosztja az A mátrix megfelelő elemével.
- `A\4` - a művelet végeredménye megegyezik az előző utasítás eredményével.

Mátrixtranszponálás

Az alábbi utasítás transzponálja az A mátrixot:
`A'`

$$A = \begin{pmatrix} 5 & 4 & 3 \\ 0 & -1 & -2 \\ 3 & 5 & 8 \\ 6 & -9 & 12 \end{pmatrix} \qquad A' = \begin{pmatrix} 5 & 0 & 3 & 6 \\ 4 & -1 & 5 & -9 \\ 3 & -2 & 8 & 1 \end{pmatrix}$$

Összeadás

Algebrai értelemben vett összeadás (azonos méretű mátrixok):

```
A+A
```

Megfelelő méretű részmátrixok összegzését is hasonlóképpen végezhetjük el. A következő példában a mátrix egyetlen sorát módosítjuk.

$$A = \begin{pmatrix} 5 & 4 & 3 \\ 0 & -1 & -2 \\ 3 & 5 & 8 \\ 6 & -9 & 12 \end{pmatrix} \qquad v = \begin{pmatrix} 5 & 0 & 2 \end{pmatrix}$$

```
A(2,:)=A(2,:)+v
```

$$A = \begin{pmatrix} 5 & 4 & 3 \\ 5 & -1 & 0 \\ 3 & 5 & 8 \\ 6 & -9 & 12 \end{pmatrix}$$

Mátrixszorzás

Algebrai értelemben a mátrixok szorzása csak megfelelő méretű ($m \times n$-es és $n \times k$) mátrixok között lehetséges.
A szorzás mátrixok körében nem kommutatív művelet! Legyen A egy 4×3, B egy 3×2 méretű mátrix:

$$A = \begin{pmatrix} 5 & 4 & 3 \\ 5 & -1 & 0 \\ 3 & 5 & 8 \\ 6 & -9 & 12 \end{pmatrix} \qquad B = \begin{pmatrix} 2 & 6 \\ 0 & -2 \\ 3 & -9 \end{pmatrix}$$

Ekkor az A×B utasítás eredménye a következő lesz, míg a B×A utasítás hibát ad eredményül.

$$\begin{pmatrix} 19 & -5 \\ 10 & 32 \\ 30 & -64 \\ 48 & -54 \end{pmatrix}$$

Koordinátánkénti szorzás

A Scilab lehetőséget ad a mátrix megfelelő elemeinek szorzására is. Ebben az esetben azonos méretű mátrixok szorzása lehetséges. A művelet kommutatív. Legyen A és B az alábbi két mátrix:

$$A = \begin{pmatrix} 5 & 4 & 3 \\ 5 & -1 & 0 \\ 3 & 5 & 8 \\ 6 & -9 & 12 \end{pmatrix} \qquad B = \begin{pmatrix} 2 & 6 & 4 \\ 0 & -2 & 7 \\ 3 & -9 & 0 \\ 1 & 5 & -4 \end{pmatrix}$$

Ekkor az A.$\times$B és a B.$\times$A utasítások eredménye is a következő lesz:

$$\begin{pmatrix} 10 & 24 & 12 \\ 0 & 2 & 0 \\ 9 & -45 & 0 \\ 6 & -45 & -48 \end{pmatrix}$$

Hatványozás

Legyen A négyzetes mátrix a következő:

$$A = \begin{pmatrix} 5 & 4 & 3 \\ 0 & -1 & -2 \\ 3 & 5 & 8 \end{pmatrix}$$

Ekkor az A mátrix harmadik hatványa a következőképpen számolható ki:

```
A^3
```

Az utasítás eredménye a következő lesz:

$$\begin{pmatrix} 263 & 260 & 288 \\ -72 & -85 & -112 \\ 384 & 424 & 527 \end{pmatrix}$$

Koordinátánkénti hatványozás

Lehetőségünk van a mátrix elemeinek hatványozására is. Ez a művelet tetszőleges mátrixokon elvégezhető. Legyen az A mátrix a következő:

$$A = \begin{pmatrix} 5 & 4 & 3 \\ 0 & -1 & -2 \\ 3 & 5 & 8 \\ 3 & -2 & 6 \end{pmatrix}$$

Ekkor az A mátrix 4. hatványaiból képzett mátrix a `A.^4` utasítással kapható meg:
Az utasítás eredménye a következő lesz:

$$\begin{pmatrix} 625 & 256 & 81 \\ 0 & 1 & 16 \\ 81 & 625 & 4096 \\ 81 & 16 & 1296 \end{pmatrix}$$

Invertálás

Mátrixok invertálására a Scilab több lehetőséget is biztosít. Az A négyzetes mátrix inverzét a következő utasítások segítségével számolhatjuk ki:

- `inv(A)`
- `A^-1`
- `1/A`

Megjegyezzük, hogy a `4/A` utasítás nem az inverz mátrix négyszeresét adja eredményül.

"Mátrix osztás"

Legyen A egy $n \times n$-es négyzetes mátrix, x és b pedig két $n \times 1$ méretű oszlopvektor. Ekkor az $A * x = b$ egyenletből x kifejezéséhez szükségünk van az A mátrixszal való "osztásra". Matematikai értelemben ez az A mátrix inverzével való balról szorzást jelenti. A Scilab azonban lehetőséget biztosít arra, hogy ezt az "osztást" elvégezhessük. Legyen A az alábbi mátrix, b pedig egy oszlopvektor:

$$A = \begin{pmatrix} 5 & 4 & 3 \\ 5 & -1 & 0 \\ 3 & 5 & 8 \end{pmatrix} \qquad b = \begin{pmatrix} 13 \\ 3 \\ 13 \end{pmatrix}$$

Ekkor az `x=A\b` utasítás eredménye az x oszlopvektor, amely az $A * x = b$ egyenlet megoldása.

"Mátrix osztás"

Legyen A és B az alábbi két mátrix:

$$A = \begin{pmatrix} 5 & 4 & 3 \\ 5 & -1 & 0 \\ 3 & 5 & 8 \end{pmatrix} \qquad B = \begin{pmatrix} 2 & 6 & 4 \\ 0 & -2 & 7 \\ 3 & -9 & 0 \\ 1 & 5 & -4 \end{pmatrix}$$

Ekkor a `C=A/B` utasítás eredménye az a C mátrix, amely az $A = C * B$ egyenlet megoldása.

"Mátrix osztás"

Scilab 5.4.1 Console

File Edit Control Applications ?

Scilab 5.4.1 Console

```
-->A
 A  =

    5.     4.     3.
    0.   - 1.   - 2.
    3.     5.     8.
    3.   - 2.     6.

-->B
 B  =

    2.     6.     4.
    0.   - 2.     7.
    3.   - 9.     0.
    1.     5.   - 4.

-->C=A/B
 C  =

    1.5108696  - 0.4347826    0.6594203    0.
  - 0.1195652  - 0.2173913    0.0797101    0.
    1.2391304    0.4347826    0.1739130    0.
    0.6630435    0.4782609    0.5579710    0.

-->C*B
 ans  =

    5.           4.     3.
  - 6.106D-16  - 1.   - 2.
    3.           5.     8.
    3.         - 2.     6.
```

Mátrix függvények

Mátrixokon az alábbi függvényeket használhatjuk a teljesség igénye nélkül.

- `[ertek, poz]=max(A)` - mátrix maximális elemének és poziciójának megkeresése.
- `[ertek, poz]=min(A)` - mátrix minimális elemének és poziciójának megkeresése.
- `[sor,oszlop]=size(A)` - mátrix méretének meghatározása.
- `d=det(A)` - mátrix determinánsának kiszámolása.
- `n=norm(A)` - mátrix kettes normájának kiszámítása.

Szöveges feladat

Tekintsük az alábbi szöveges feladatot:
Keressük azt a parabolát, amely átmegy az alábbi három ponton:

$$(-1;9), \qquad (1;5) \qquad (2;12)!$$

Megoldás:
Keressük a parabolát

$$p(x) = ax^2 + bx + c$$

alakban.

Átírása egyenletté

A megadott három pont kielégíti a parabola egyenletét, vagyis az alábbi három egyenletet írhatjuk fel:

$$a(-1)^2 + b(-1) + c = 9$$

$$a(1)^2 + b(1) + c = 5$$

$$a(2)^2 + b(2) + c = 12$$

Ebből a három egyenletből az alábbi egyenlet írható fel:

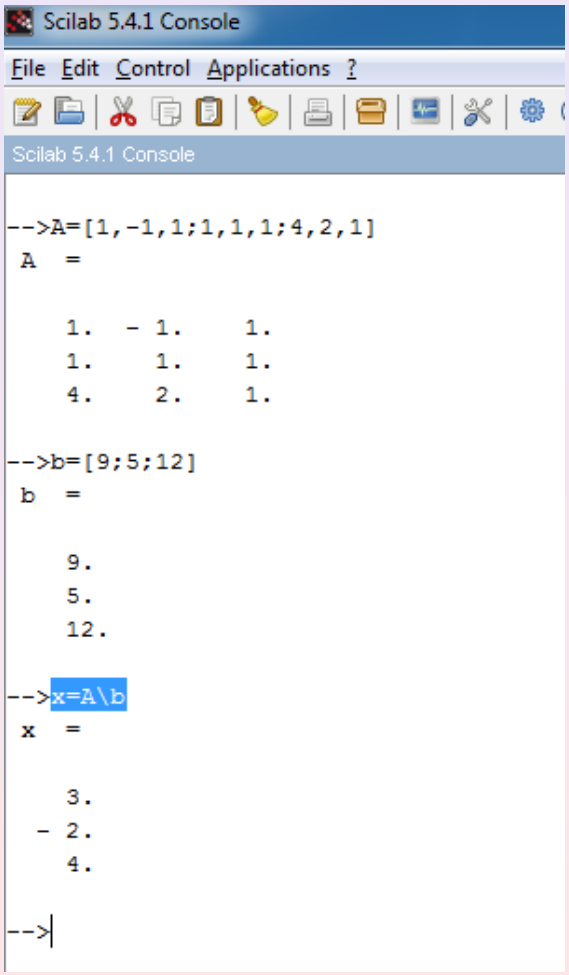
$$\begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \\ 4 & 2 & 1 \end{pmatrix} * \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 9 \\ 5 \\ 12 \end{pmatrix}$$

Megoldása

A feladat megoldását (a parabola együtthatóit) Scilabban az alábbi utasítások segítségével kapjuk meg:

```
A=[1,-1,1;1,1,1;4,2,1]
b=[9;5;12]
x=A\b
```

Megoldása


Scilab 5.4.1 Console
File Edit Control Applications ?
Scilab 5.4.1 Console
-->A=[1,-1,1;1,1,1;4,2,1]
A =
1. - 1. 1.
1. 1. 1.
4. 2. 1.
-->b=[9;5;12]
b =
9.
5.
12.
-->x=A\b
x =
3.
- 2.
4.
-->

Megoldása

A keresett parabola együtthatói :

$$a = 3, \qquad b = -2, \qquad c = 4.$$

A keresett parabola, ami átmegy a három ponton:

$$y = 3x^2 - 2x + 4$$

Scilab programozás és a numerikus algoritmusok Scilabban
3. előadás
Grafikai lehetőségek

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

1. Kétdimenziós ábrázolás
   - Grafikus ablak
   - Ábrázolási lehetőségek, beállítások
2. Háromdimenziós ábrák
3. Összefoglalás

A plot parancs: egyszerű x-y terek

plot parancs

A parancs általános formája a következő:
plot(x,y[xcap,ycap,caption])
plot(y[xcap,ycap,caption])

- x és y oszlopvektorok (egyforma hosszúak).
- xcap: az x tengely elnevezése.
- ycap: az y tengely elnevezése.
- 'caption': az ábra elnevezése.

Scilab grafikus ablak

Menüpontok:
File:

- **New figure:** Új grafikai ablak megnyitása.
- **Load:** Fájl beolvasása.
- **Save:** Mentés Scilab formátumban (scg).
- **Export:** Mentés különböző formátumban (png, jpg, gif).
- **Vectorial export:** Újabb formátumokba mentés (pdf, ps, stb.).
- **Copy to clipboard:** Átmásolja az ábrákat clipboardra.
- **Page setup:** Oldalbeállítások.
- **Print:** Alapértelmezett Windows nyomtatás.
- **Close:** Ablak bezárása.

Scilab grafikus ablak

Tools:

- **Show/Hide Toolbar:** megjeleníti/elrejti a Toolbart.
- **Zoom:** Magyítás bal egérgombot húzva a célterületen.
- **Original View:** Eredeti nézet.
- **2D/3D Rotation:** 2D/3D forgatás.

Scilab grafikus ablak

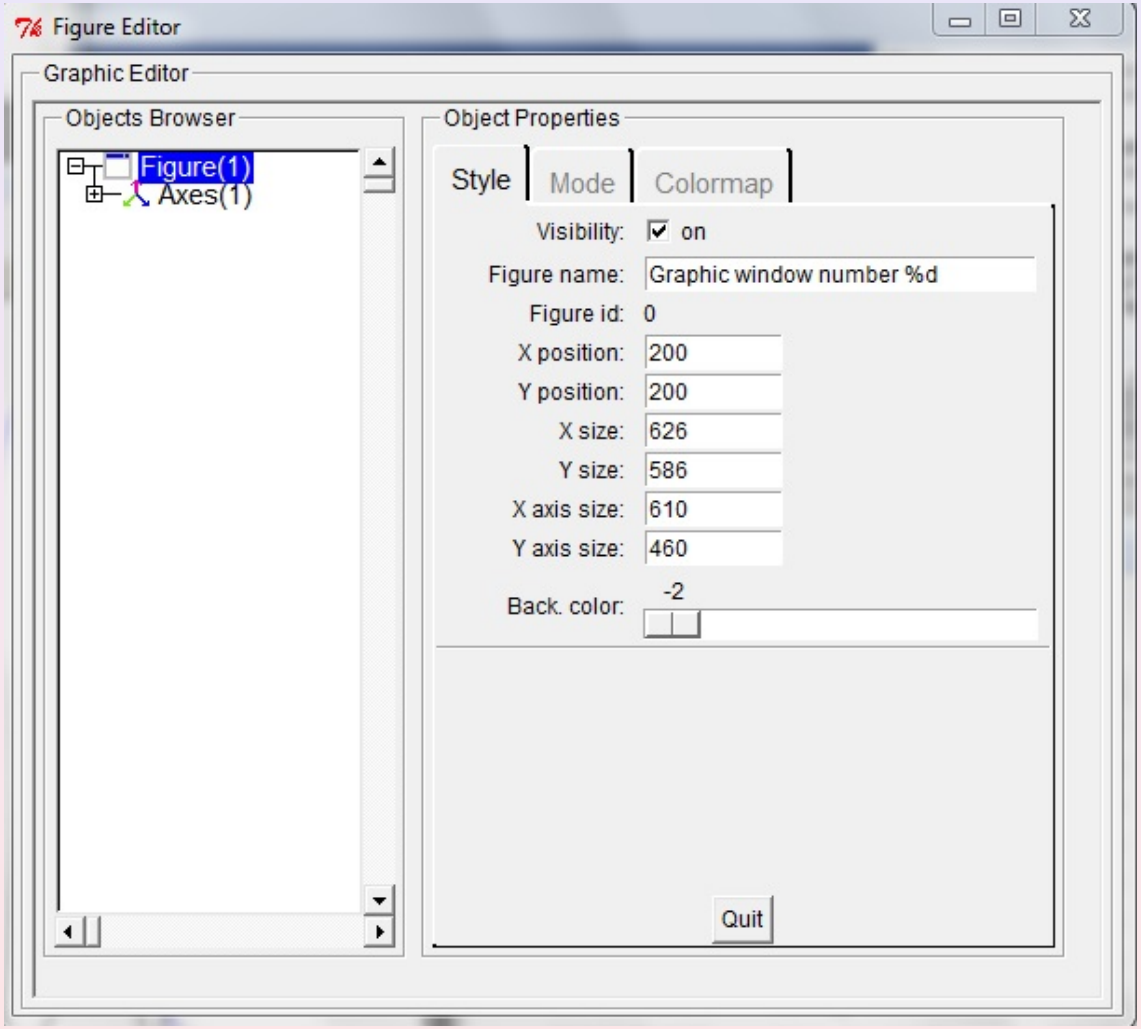
Edit:

- **Select as current figure:** Jelenlegi grafika kiválasztása.
- **Clear figure:** Grafika törlése.
- **Figure properties:** Grafika beállításai.
- **Axes properties:** Tengely beállítások.
- **Start entity picker:** A kiválasztott objektumon módosításokat enged.
- **Stop entity picker:** Az előbbi beállítás kikapcsolása.

Grafikai beállítások

Az edit-figure properties alatt módosításokat hajthatunk végre az általunk rajzoltatott grafikán. A Style fül alatt megadhatjuk a grafika nevét (Figure name), számát (Figure ID), pozícióját (X,Y pos), a tengelyek méreteit (X,Y axis size), valamint a háttér színét.

Grafikai beállítások



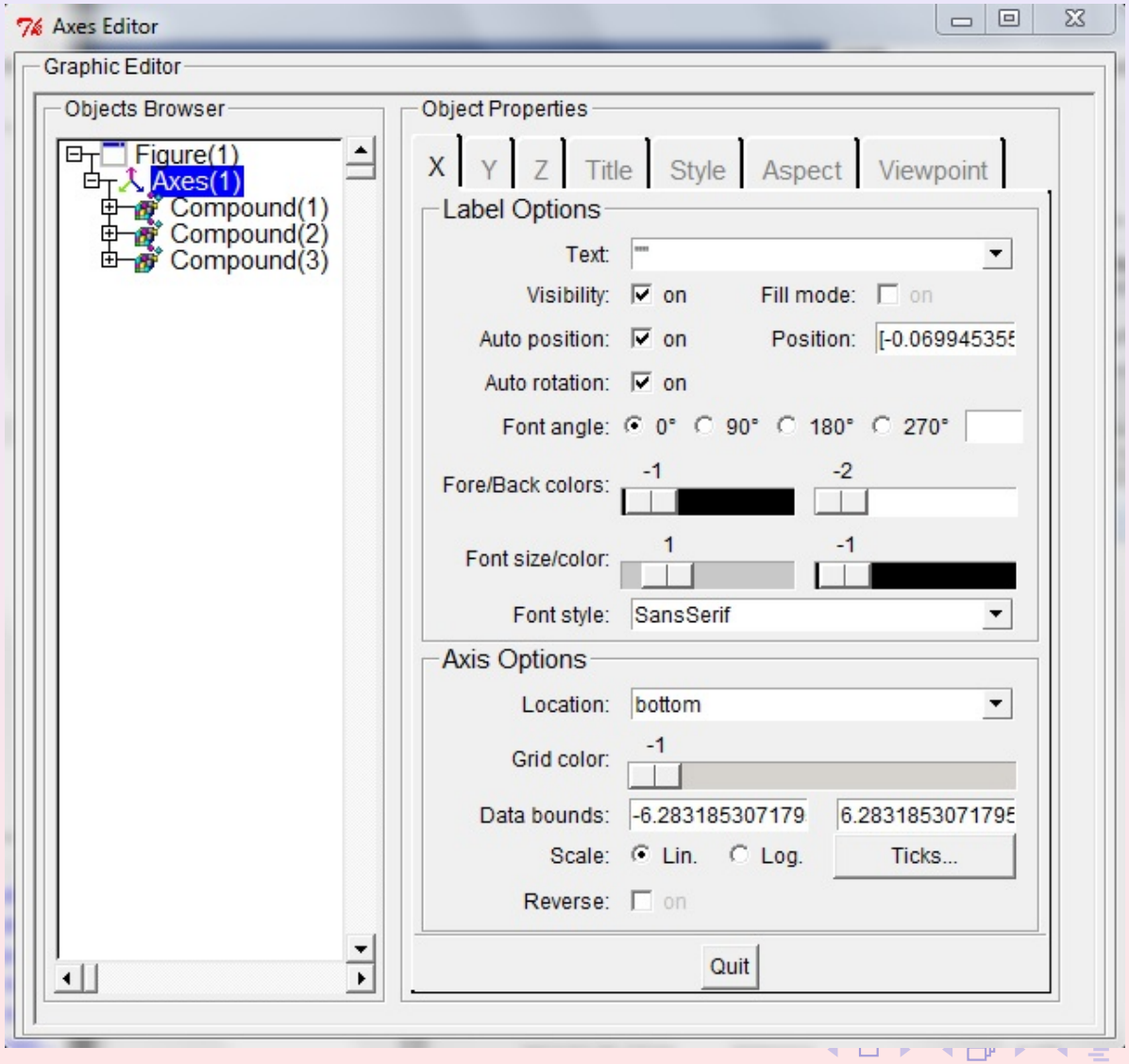
Grafikai beállítások

A Mode fül alatt bekapcsolható az automata méretezés, megadható a rajzolási mód és a forgatási stílus.
Az utolsó fül a Colormap, ahol a színeket állíthatjuk be.

Tengely beállítások

Ugyanitt megtaláljuk a tengelyek beállításait is. (Figure alatt Axes). Lehetőségünk van egyesével beállítani a tengelyeket, azok nevét, helyzetét, színét. Képesek vagyunk elforgatni a tengelyeket, pozícionálni őket kézileg vagy automatikusan. A tengelyek stílus beállításainál a vonalak stílusát, illetve színét állíthatjuk be. Aspect fül alatt margókat, rácsozást, Viewpoint fül alatt pedig a 2D, illetve 3D nézetet valamint a forgatást találjuk. Az Objects Browserben az Axes fület lenyitva találunk még 2 alopciót (Compound, Polyline), ahol további beállításokat végezhetünk.

Tengely beállítások



Hiba vonalak

Az **errbar** paranccsal lehet hozzáadni hiba vonalakat a függvényhez. Függvény hívása:
errbar (x,y,em,ep)
az x,y,em,ep 4 azonos méretű mátrix.

Az alábbi utasítássorozat a szinusz függvényt ábrázolja hibavonalakkal.

```
x=(-%pi:%pi/50:%pi); y=sin(x)
em= 0.05*rand(x); ep=0.1*rand(x);
plot(x,y)
errbar (x,y,em,ep)
```

Rács hozzáadása az ábrához

xgrid() parancs

Parancs rács hozzáadásához: **xgrid()**. Ha a függvény argumentumába egész számot írunk, az megváltoztatja a rácsvonalak színét.

Az alábbi utasítássorozat a szinusz függvény tábrázolja rácsvonalakkal.

```
x=(-%pi:%pi/50:%pi);y=sin(x)
plot(x,y)
xgrid(2)
```

Grafika igazítása más vonal parancsokkal

Az aktuális kép alakítására szolgáló általános grafikai parancsok:

- **xbasc():** A grafikai ablak kiürítése.
- **xclear(i):** Kiürít egy vagy több grafikus ablakot.
- **xdel(i):** Grafikai ablak törlése.
- **xselect():** Grafikai ablak előhozása.
- **xsave('fájlnév',i):** Elmenti az i grafikai ablakot.
- **xload('fájlnév',[i]):** Fájl betöltése.
- **winsid():** Visszaadja a számát a grafikai ablaknak.

Egy ábra globális paramétereinek megváltoztatása

Globális paraméterek: címkék betűinek mérete és típusa, színtérkép, hányas számú ablakot nyissuk meg, grafikai ablak pozíciója.
A tulajdonságok megváltoztatását **'xset'**-tel kezdjük.

- **xset("background",color):** A háttér beállítása.
- **xset("dashes",i):** Kötőjel beállítása.
- **xset("default"):** Alapbeállítások.
- **xset("foreground",color):** Az előtér színének beállítása.
- **xset("linemode",type):** A vonal rajzoló mód beállítása.

Egy ábra globális paramétereinek megváltoztatása

- **xset("mark",markid,marksize):** A jel és a méretének beállítása.
- **xset("pattern",value):** Sablon beállítása. Value egy egész szám.
- **xset("thickness",value):** Vonal vastagság beállítása.
- **xset("viewport",x,y):** Az ábra pozíciójának beállítása.
- **xset("wdim",width,height):** Szélesség és magasság beállítása.
- **xset("window",window number):** Az ablak ablakszámára állítása és az ablak elkészítése.

Grafikai ablak hátterének megváltoztatása

Háttér változtatása:

A grafikai ablak háttere alapvetően fehér, de a háttérszín megváltoztatható a **xset**('background', color)paranccsal, ahol a *color* egy nemnegatív szám.

0 black	1 black	2 dark blue
3 bright green	4 sky blue	5 bright red
6 purple	7 bright red	8 white
9 light blue	10 blue	11 dark blue
12 sky blue	13 dark green	14 dark green
15 bright green	16 dark green	17 dark greenish blue
18 greenish blue	19 dark red	20 dark red
21 red	22 dark purple	23 bright purple
24 dark reddish brown	25 dark reddis brown	26 reddish brown
27 dark orange	28 pink	29 pink
30 pink	31 pink	32 dark yellow

A plot2d parancs

plot2d parancs

2 dimenziós ábránál használható
Általános alakja:
plot2d(x,y[style,strf,leg,rect,nax])
plot2d(y);

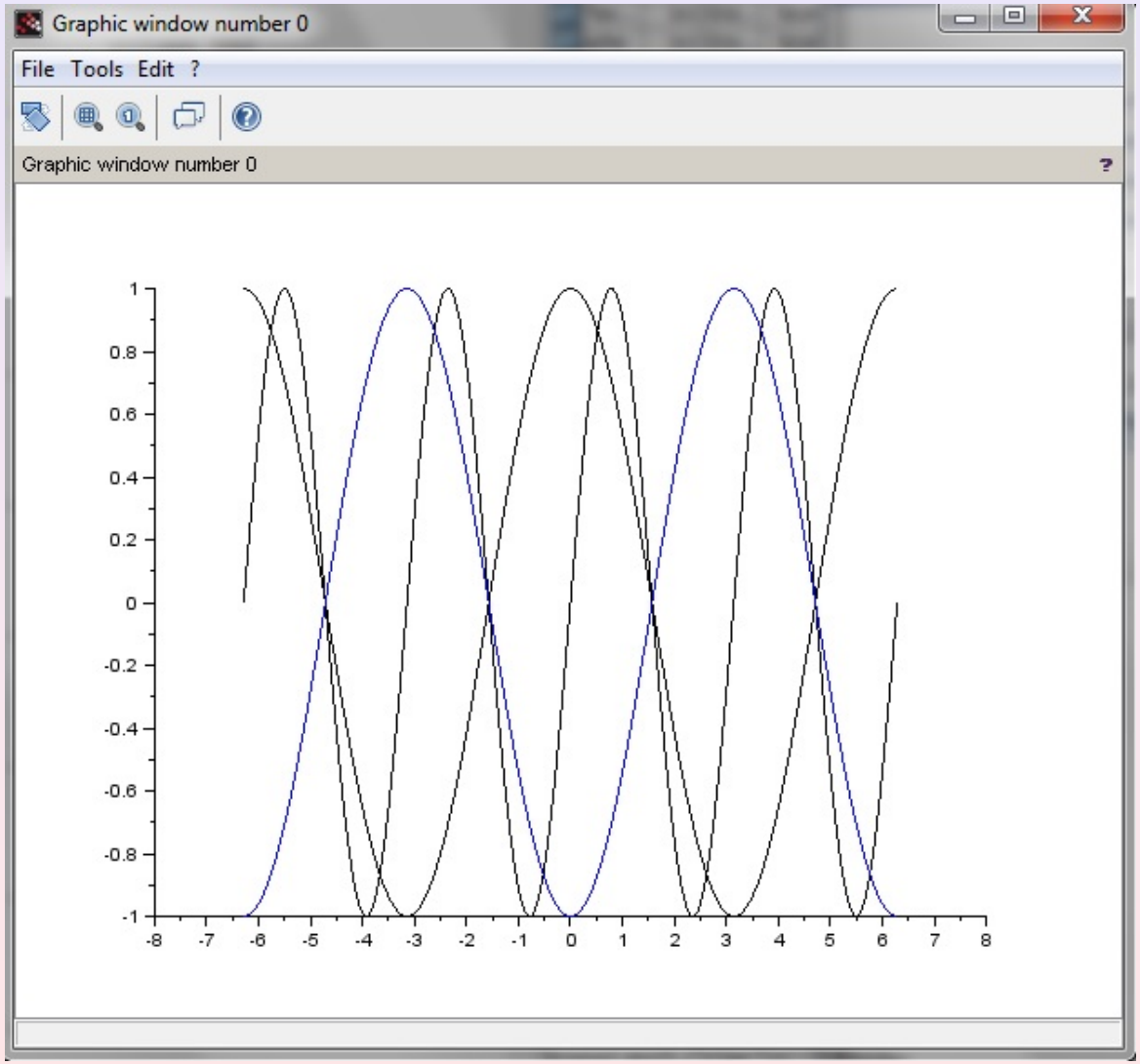
Egyszerű példa:

1. Hozzuk létre a következő vektorokat:

```
x=(-2*%pi:%pi/100:2*%pi)
y=sin(2*x)
z=cos(x)
w=-cos(x)
```

2. Függvények hívása:
plot2d(x,y) plot2d(x,z) plot2d(x,w)

Az előbbi példa ábrázolása



Rács definiálása

plotframe parancs

Általános alak:
plotframe(rect,tics,[arg_opt1, arg_opt2, arg_opt3])

- rect: [xmin,ymin,xmax,ymax] vektor, reprezentálja az x és y értékeket
- tics: [nx,mx,ny,my] vektor, az mx és nx: x tengely, az my és ny: y tengely
- arg_opt1, arg_opt2 és arg_opt3 opcionális argumentumok

A plotframe parancs a plot2d paranccsal együtt használatos

Rács definiálása

Példa:

```
x=[-0.3:0.8:27.3]; y=rand(x);
rect=[min(x), min(y), max(x), max(y)];
tics=[4,10,2,5];
plotframe(rect, tics, [f,f],['my plot with grids', 'x','y']
                          [0.5,0,0.5,0.5]);
plot2d(x,y,2,'000')
```

Más két-dimenziós plot parancsok

A plot és a plot2d parancsok folytonos vonalakat készítenek a rajzolandó görbékhez. Más típusú görbék: plot2d1, plot2d2, plot2d3, plot2d4.

- **plot2d1:** Lineáris görbék, de lehetséges logaritmikus skálákkal
- **plot2d2:** Konstans görbék
- **plot2d3:** Függőleges vonalak
- **plot2d4:** Nyíl stílus

Általános alak:

plot2di(str,x,y,[style,strf,leg,rect,max]) (ahol i=1,2,3,4)

Hisztogramok

A hisztogramok

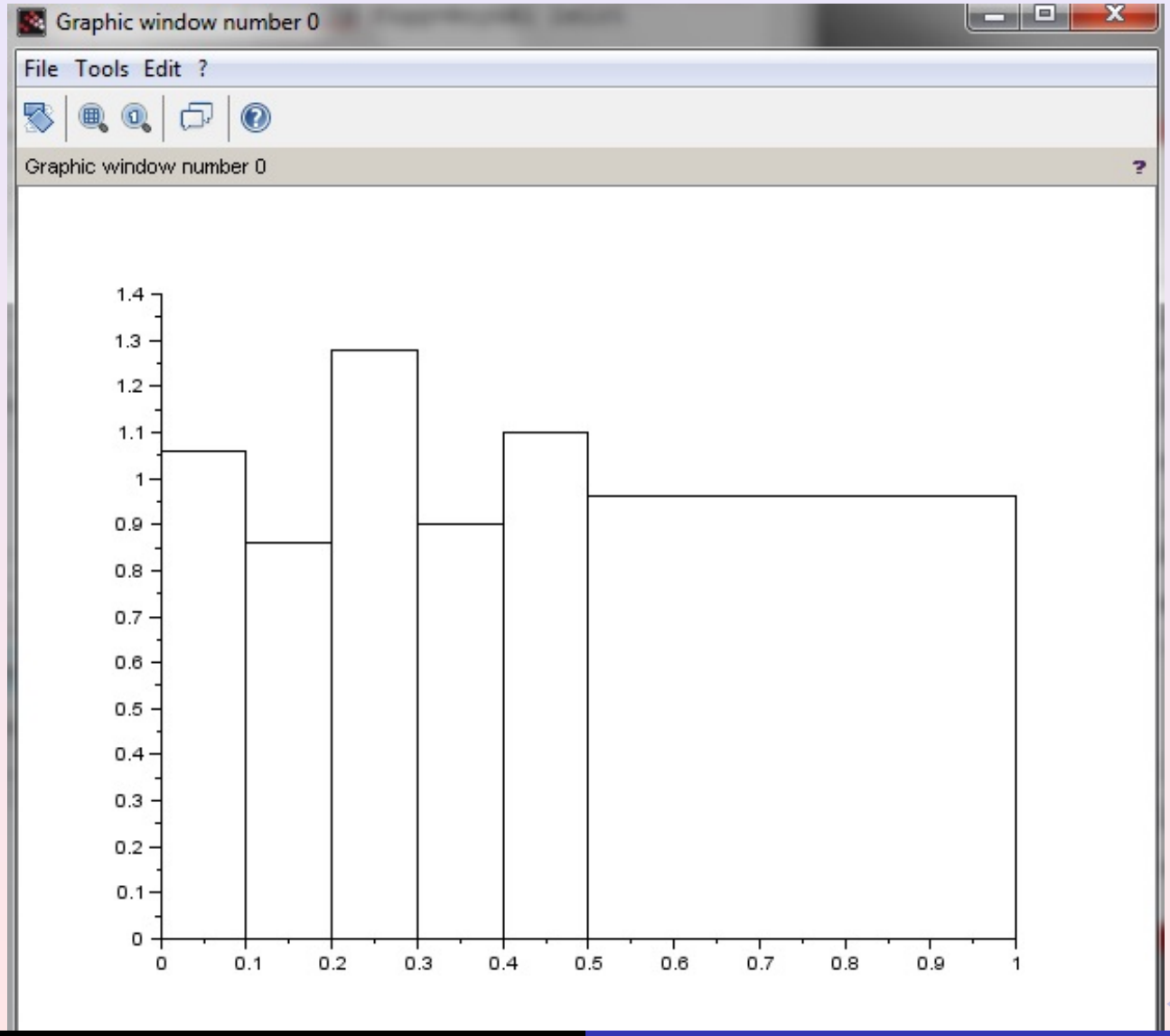
Téglalapok, egy adathalmaz gyakorisági osztályozását mutatja. Az összegyűjtött adat osztályozva van, a hisztogram oszlopok magassága reprezentálja az adatok mennyiségét egy bizonyos osztályban.
hitsplot(npoint,data,[style,strf,leg,rect,nax])

- *npoint* paraméter: egész érték, az osztályok száma.
- *data* paraméter: egy valós vektor, ami tartalmazza az adat elérhető gyakorisági osztályozását.
- *opcionális* paraméterek: style, strf, leg, rect és a nax követik a plot2d függvénynél leírt definíciókat.

Példa:

```
x=rand(1:500);
oszt=[0.,0.1,0.2,0.3,0.4,0.5,1.0];
histplot(oszt,x)
```

Az előbbi példa ábrázolva:



Alablakok készítése xsetech paranccsal

A grafikus ablakot feloszhatjuk részekre.

xsetech parancs

Az általános formája ennek a parancsnak:
xsetech(wrect[,frect,logflag])

- **wrect:** az alablak definiálása egy négyelemű vektorral.
- **frect:** az alablak rajzeterületének definiálása egy négyelemű vektorral.
- **logflag:** egy 2 karakter hosszú string a tengelyek skálázásra. Értéke lehet "n" (normál) vagy "l" (logaritmikus).

Grafikai beállítások módosítása

A következő parancsok arra használhatók, hogy módosítsuk az ábra tulajdonságait:

- **xgrid(style):** hozzáad egy rácsot a kétdimenziós képhez, a hívó a paraméter szín
- **xtitle(title,xlabel,ylabel,frame):** hozzáadja a címet és a tengely címkéket
- **titlepage(string):** elhelyezi a címoldalt az ábra közepén
- **xclear(x,y,w,h):** törli az (x,y) által megadott terület tartalmát a balfelső saroktól, szélessége=w, magassága=h
- **xstring(x,y,str,[angle,flag]):** kirajzolja az 'str' stringet, ami az (x,y) pontban kezdődik

Ábra elrejtése

A következő utasítással elrejthetjük az ábránkat: **drawlater()**
Az utasítás kiadása után az ábrák nem jelennek meg. Az elrejtett ábrák a
drawnow()
utasítás kiadása után jelennek meg ismét.

A színtérkép

Színtérkép

A színtérkép egy mx3-as mátrix. A színtérkép megváltoztatható az **xset**("colormap",cmap) paranccsal.

A színtérkép

Példa:

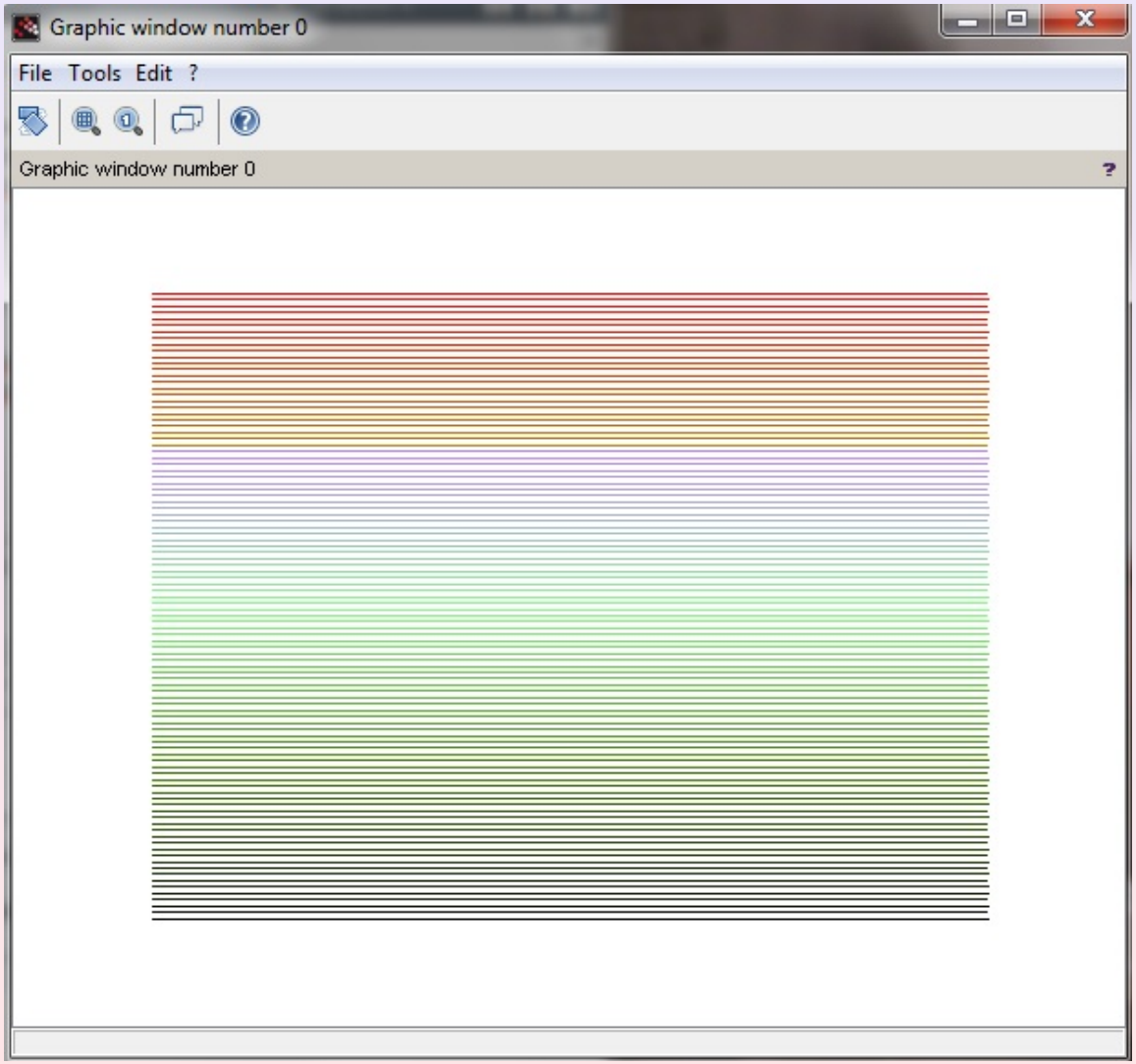
A következő parancs segítségével elkészítjük a saját színtérképünket 100 színből. Ezen parancsok futtatása előtt kattintsunk a **restart** menüpontra. Az első lépés, hogy elkészítsük az R (piros), a G (zöld), és a B (kék) vektort:

```
m=100; R=[(1:m)/m]
n2=m/2; G1=[(1:n2)/n2];
G2=[(n2+1:m)]; G3=(m-G2)/(m-n2); G=[G1, G3];
B=[zeros(1:m/4); G1; zeros(1:m/4)];
myColorMap1=[R',G',B']; xset('colormap', myColorMap1);
```

Kirajzolás:

```
x=(0:0.1:1)'; y=zeros(x);
for j=1:100, y= y+2;
plot2d(x,y,j,'010','',[0 0 1 200]), end
```

Az előző példa ábrázolása



Újabb példa a színtérképhez

Példa 2: A piros különböző árnyalatai:
(Zöld és kék 0-ra állítva)

- m=100; R=[(1:m)/m];
- G=[zeros(1:m)]; B=[zeros(1:m)];
- myColorMap2=[R',G',B']; xset('colormap',myColorMap2);
- xbasc();
- x=(0:0.1:1)'; y=zeros(x);
- for j=1:100, y=y+2; plot2d(x,y,j,'010',",[0 0 1 200]), end

A 3 dimenziós grafika

plot3d parancs

3 dimenziós grafika kirajzolása a következő 3 módon lehetséges:
plot3d (x,y,z,[theta,alpha,leg,flag,cbox])
plot3d (xf,yf,zf,[theta,alpha,leg,flag,cbox])
plot3d (xf,yf,list(zf,colors),[theta,alpha,leg,flag,cbox])

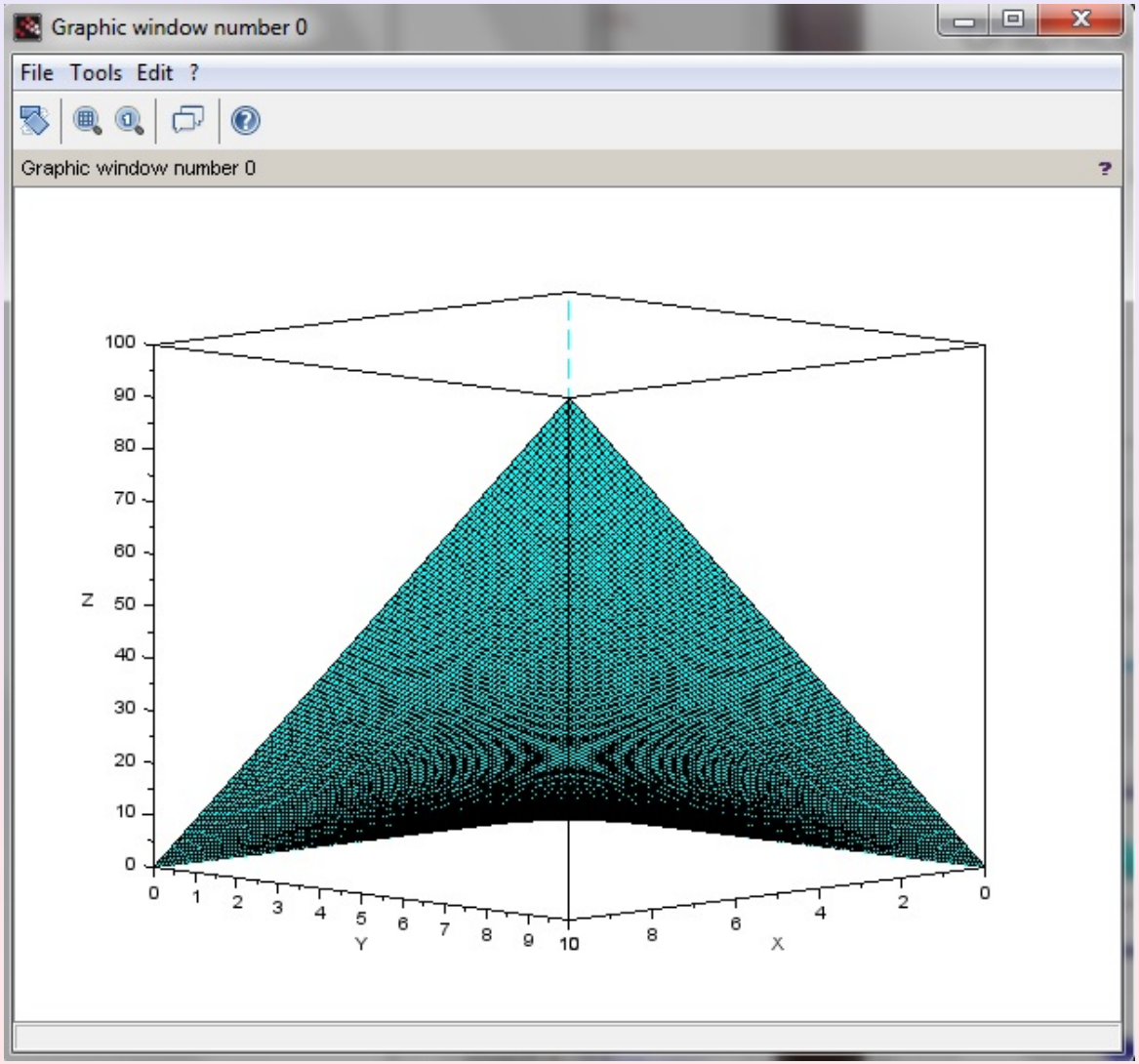
Példa plot3d-re és 3 dimenziós grafika forgatására

Példa:

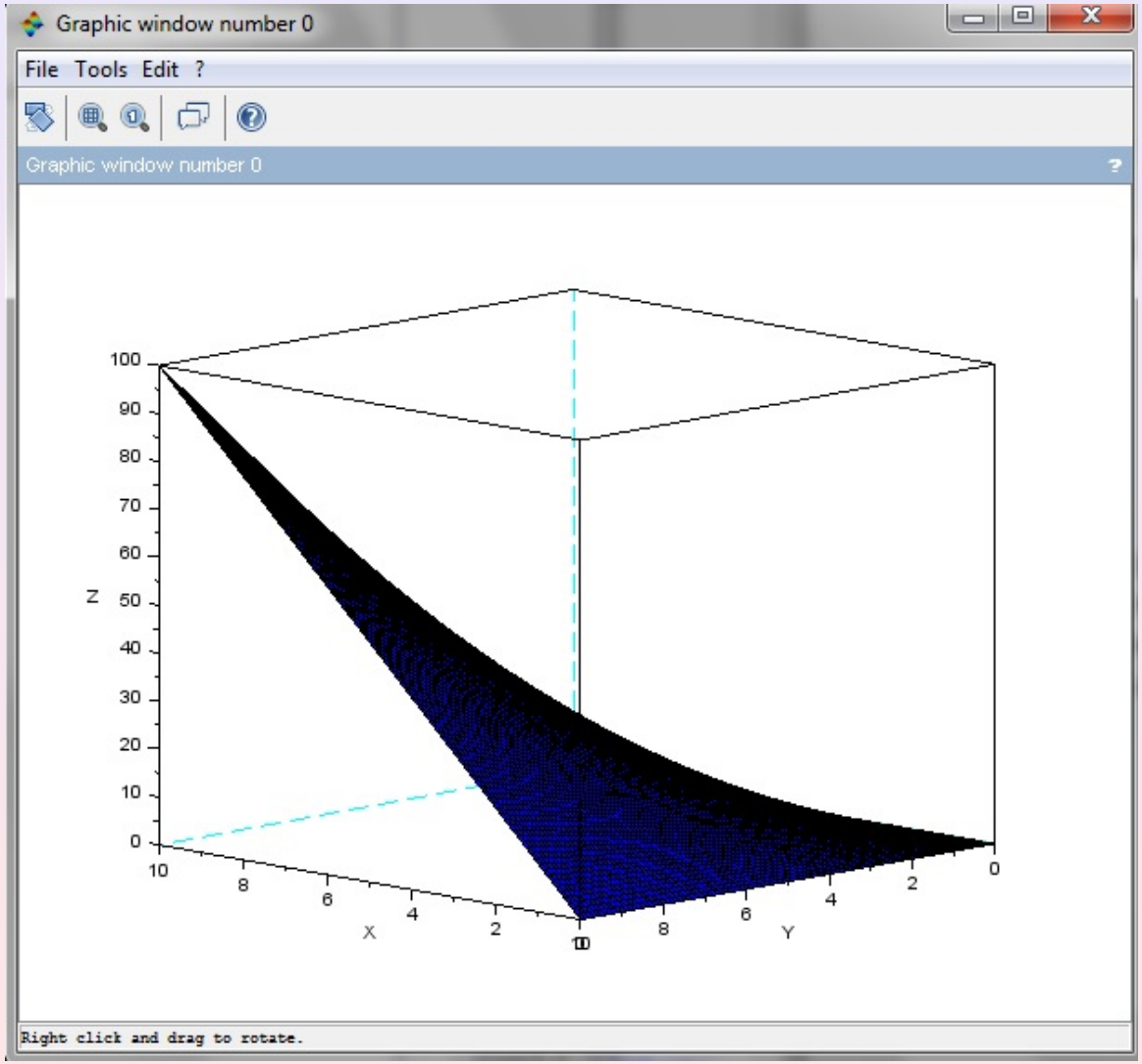
- x=(0:0.1:10)'; y=x; z=x*y'
- plot3d(x,y,z)

Ez a példa generál egy 3d-s grafikát, ami a tengelyekkel 45 fokos szöget zár be. Majd ezt a grafikát megforgatjuk a grafikus ablakban.

A példa ábrázolása



A példa ábrázolása



A megforgatott grafika

3 dimenziós hisztogramok

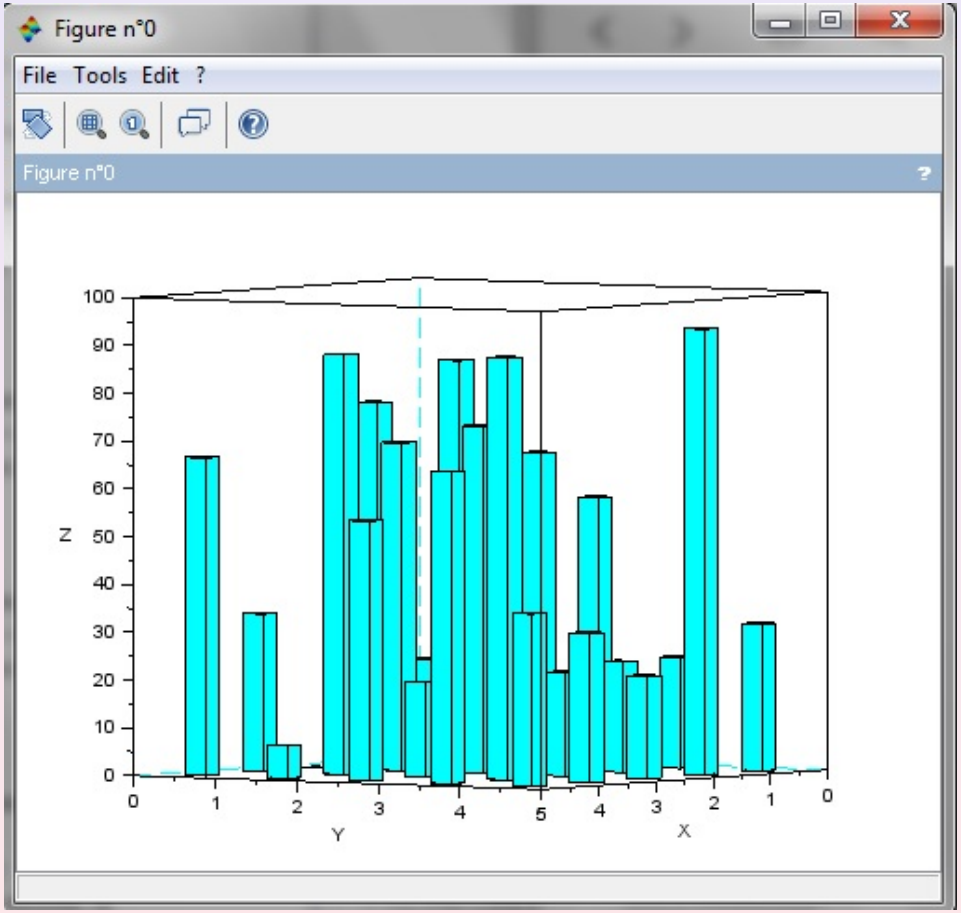
3d-s hisztogramok

3D-s hisztogram létrehozásának 2 különböző formája van:
hist3d (f,[theta,alpha,leg,flag,cbox])
hist3d (list(f,x,y),[theta,alpha,leg,flag,cbox])

Példa:

```
xset('default');
x=(-10:5:10); y=x;
[nx x]=size(x); [ny my]=size(y);
f=rand(mx,my)*100
xbasc();
hist3d(f)
```

A példa ábrázolása



Scilab grafikai funkciók áttekintése

2 dimenziós ábrázolás:

- **plot:** Eyszerű görbe rajzolása.
- **plot2d:** Görbe rajzolása.
- **plot2d1:** Görbe rajzolása logaritmikus tengelyekkel.
- **plot2d2:** Görbe rajzolása lépcsőzetesen.
- **plot2d3:** Görbe rajzolása függőleges sávokkal.
- **plot2d4:** Görbe rajzolása nyilakkal.
- **fplot2d:** Görbe által definiált függvény rajzolása.
- **champ:** 2D vektor mező.
- **champ1:** 2D vektor mező színes nyilakkal.

Scilab grafikai funkciók áttekintése

2 dimenziós ábrázolás:

- **grayplot:** Színezett 2D-s grafika a felületen.
- **fgrayplot:** Színezett 2D-s függvény által meghatározott grafika.
- **Sgrayplot:** Sima 2D-s színezett grafika a felületen.
- **Sfgrayplot:** Sima 2D-s színezett, függvény által meghatározott grafika a felületen.
- **xgrid:** Hozzáad egy rácsot a 2D-s grafikához.
- **errbar:** Függőleges hiba vonalakat ad hozzá a 2D-s grafikához.
- **histplot:** Rajzol egy hisztogramot.
- **Matplot:** Mátrix 2D-s színezett ábrázolása.

Scilab grafikai funkciók áttekintése:

3 dimenziós ábrázolás:

- **plot3d:** Rajzol egy felületet.
- **plot3d1:** Rajzol egy felületet szürke vagy színes szintekkel.
- **fplot3d:** Rajzol egy függvény által meghatározott felületet.
- **fplot3d1:** Rajzol egy függvény által meghatározott felületet szürke vagy színes szintekkel.
- **param3d:** Rajzol egy görbét.
- **param3d1:** Görbéket rajzol.
- **hist3d:** 3D-s hisztogram létrehozása.

Scilab programozás és a numerikus algoritmusok Scilabban
4. előadás
Programozási alapismeretek, vezérlési szerkezetek

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

1. Relációs és logikai operátorok
2. Elágazások
   - Feltételes elágazás
   - Esetszétválasztásos elágazás
3. Függvények, változók
   - Futtatás fájlból
   - Függvények
   - Globális változók
   - Függvény utasítások
4. Deguggolás, input, output műveletek
   - Debuggolás
   - Ki- és bementetek
   - Kiíratás, beolvasás
   - Műveletek fájlokkal
5. Stringműveletek
   - Összefűzés

Aritmetikai operátorok

- egyenlőség ==
- kisebb <
- kisebb-egyenlő <=
- nagyobb >
- nagyobb-egyenlő >=
- nem egyenlő $\sim =$ vagy <>

Logikai operátorok

- és &
- vagy |
- negált $\sim$

"If" elágazás

Az feltételes függvény bonyolultságától függően három esetet különböztetünk meg:

- `if feltétel then utasítások, end`
- `if feltétel then utasítások, else utasítások, end`
- `if feltétel then utasítások, elseif feltétel then utasítások, else utasítások, end`

"If" elágazás

Példák:

- `x = 10; y = 5; if x> 5 then disp(y), end`
  Eredménye: 5 lesz.
- `x = 3 ; y = 5; if x>5 then disp(y), else disp(x), end`
  Eredménye: 3 lesz.
- `x = 3; y = 5; z = 4; if x>5 then disp(x),`
  `elseif x>6 then disp(y), else disp(z), end`
  Eredménye: 4 lesz.

"Select-case" elágazás

Általános szintacisa:

```
select változó, case n1, utasítások,
                case n2, utasítások,
                .
                .
                case nm, utasítások,
                end
```

, ahol n1,n2,..., nm az egyes esetek

"Select-case" elágazás

Példa:

```
x = -1; select x, case 1, y = x+5,
                  case -1, y = sqrt(x), end
r = 7; select r, case 1, k=r,
                 case 2, k=r^2,
                 case 7, k=r^3, end
```

Eredmény:

```
y=i
k=343.
```

Parancsok futtatása fájlból

A Scilabban lehetőség van arra, hogy a programjainkat elmentsük egy fájlba, és azt később az alábbi parancs segítségével teljes egészében lefuttassuk. Alapértelmezésben a program a saját könyvtárában, azon belül pedig a bin mappában keresi a fájlokat, de ha az elérési utat is megadjuk, akkor ez módosulhat.

```
exec('program1.txt')
```

Parancsok futtatása fájlból

Legyen a program1.txt fájl tartalma a következő:

```
clear //változók törlése
x = list(10., -1., 3., 5., -7., 4., 2.);
suma = 0;
n = size(x);
for j = 1:n
suma = suma + x(j);
end
xbar = suma/n;
n
xbar
```

Futtassuk a következő parancsot Scilabban:

```
exec('feladatok\textbackslash program1.txt'
```

Függvény

A függvények olyan eljárások, melyeknek paraméterben megadhatunk 0, 1 vagy akár több paramétert is, kimenetként pedig szintén 0, 1 vagy több értéket kaphatunk. (A 0 kimeneti értékkel rendelkező függvényt speciális esetben alkalmazzuk.)

```
deff('[y1,...,yn] = fname(x1,...,xm)','utasítások'),
```

,ahol y1,...,yn kimeneti változók, és x1,...,xm bemeneti paraméterek.

Függvény

Példa:
deff('[z]=Euler(r,theta)','z=r*exp(%i*theta)')
Euler(1.0,-%pi/2)
Eredmény:

```
ans=6.123D-17 - i
```

Függvények tárolása fájlban

Lehetőség van függvényeket fájlba írni, és azokat egy paranccsal lefuttatni.
A fájl felépítése (pelda.txt):

```
Function [y1,...,yn] = fname(x1,...,xm)
utasítások
```

Futtatás:

```
exec('pelda.txt')
[z1,...,zn]=fname(v1,...,vm)
```

Függvények tárolása fájlban

Példa (fgv.txt):

```
function [x,y,z] = fgv(a,b,c)
x = a*b
y = a*c
z = a
```

Futtatás:

```
exec('fgv.txt')
[x1,y1,z1]=fgv(10.0, 5.0, 3.0)
```

Eredmény:

```
x=50.
y=30.
z=10.
```

Globális változók

Alapértelmezésben a Scilabban ha létrehozunk egy változót, az lokális hatókörrel fog rendelkezni. Ha például egy függvényen belül hozunk létre egy változót, akkor az csak addig fog létezni, amíg a függvény lefut. Ahhoz, hogy ezt megakadályozzuk, globális tartományba kell helyezni a változót, melyet a global parancs segítségével tehetünk. A global parancsot használva a függvények egy új használati módja jön létre, a változó inicializálás.

```
global('vált1',...,'váltn')
global vált1 ... váltn
```

Globális változók

Példa (inic.txt):

```
function inic()
    global a b
    a = 2
    b = 7
```

Futtatás:

```
exec('inic.txt')
inic()
```

Eredmény:

```
a=2.
b=7.
```

Speciális függvény utasítások

- argn - A függvény argumentumainak a számát adja.
- error - Hibaüzeneteke küldésére használható. A függvény futása megáll a használatakor.
- warning - Figyelmeztetés küldése használat közben. A függvény futása nem áll meg.
- pause - Szünetelteti a függvény futását.
- break - Ciklusok megszakítására használatos utasítás.
- return - Függvények visszatérési értéke adható meg a használatával.

Példafüggvény

```
function [z] = func1(x,y)
[out,in]=argn(0)
if x == 0 then
error('0-val nem lehet osztani!');
end,
slope = y/x;
pause,
z = sqrt(slope);
s = resume(slope);
```

Debuggolás

- A legegyszerűbb módja a debuggolásnak Scilab függvényekben a *pause* paranccsal van. Amikor a függvény végrehajtása eléri ezt a sort, a függvény végrehajtása megáll.
- A függvény folytatódik, ha beírjuk a *return* vagy a *resume* parancsot.
- A függvény végrehajtását megszakíthatjuk az *abort* paranccsal.

Változók fájlba mentése és visszaolvasása

```
save(fájlnév,változók);
load(fájlnév,változók);
```

Példa:

```
A = [1. 2. 3.; 3. 4. 5.];
b = 1:10;
save("DataAb.dat","A","b")
clear //töröljük az összes változót
load("DataAb.dat")
```

Formázatlan adat képernyőre és fájlba írása

```
print(cél,változók-stringek)
```

Cél lehet:

- 6 vagy %io(2) - képernyő
- 'fájlnév.kit' - fájl

Példa:

```
r = 1:2:25; A = rand(5,3);
print (6,A,r,' változók')
print('data1.txt',A,r,' változók')
```

Beolvasás billentyűzetről

```
x = read (forrás,sor,oszlop);
```

Források:

- 5 vagy %io(1) - billentyűzet
- 'fájlnév.kit' - fájl (részletesen a fájlkezelés pontban)

Példa:

```
x = read (5,1,1);
```

Fájlkezelés

```
[elnevezés [,hiba]]=file('open', fájl-név [,státusz]
                        [,hozzáférés [,méret]] [,forma])
```

Elemei:

- fájl-név - a fájl valós neve
- státusz - "new", "old", "unknown", "scratch"
- hozzáférés - "sequential", "direct"
- forma - "formatted", "unformatted"
- méret - rekord mérete bájtban (ha a hozzáférés="direct")
- elnevezés - így hivatkozunk később a fájlra
- hiba - megnyitási hiba esetén a megadott hibakódot írja ki

Műveletek megnyitott fájllal

`file(művelet,elnevezés)`

Műveletek:

- "open" - fájl megnyitása
- "close" - bezárás
- "rewind" - mutató a fájl elejére
- "backspace" - mutató az utolsó rekord elejére
- "last" - mutató az utolsó rekord végére

Fájlba írás

```
write(elnevezés,változók-stringek)
```

Példa:

```
x1 = 0:0.5:10
x2 = x1^2
B = [x1',x2']
m = file('open','fajl.txt','new')
write(m,B,'(2(f10.6,2x))')
file('close',m)
```

Fájlból olvasás

`read(elnevezés,sor,oszlop,[formátum])`

A sor és oszlop indexek. Ekkora méretű tömböt fog kiolvasni a művelet hatására. Ha -1-et adunk meg értéknek, akkor a maximális méretet fogja kiolvasni.
Példa:

```
file('rewind',u)
B = read(u,-1,3)
file('close',u)
```

Stringek összefűzése

```
string=string1+string2+...+stringn
```

Példa:

```
s1 = 'Az összefűzés '
s2 = 'eredménye '
s3 = 'ez lett.'
string = s1 + s2 + s3
```

Eredmény:

```
string=Az összefűzés eredménye ez lett.
```

String vektor összefűzése stringgel

```
strcat(strink-vektor,string)
```

Példa:

```
sv=string(1:5) //számok 1-től 5-ig (vektor)
s=strcat(sv,':') //s már nem vektor, hanem sima string
```

Eredmény:

```
sv=! 1   2   3   4   5   !
s=1:2:3:4:5
```

String hossza

```
length(string)
```

Példa:

```
s='Ez egy string'
length(s)
```

Eredmény:

```
ans=13.
```

Rész-string

`part(string,paraméter)`

Paraméterként megadhatunk 1 vagy több indexet is, illetve intervallumot is.

Rész-string index alapján

Példa:

```
x=part('abcd',1)
y=part('abcd',[1,3])
z=part('abcd',[1,2,4])
```

Eredmény:

```
x=a
y=ac
z=abd
```

Rész-string intervallum alapján

Példa:

```
s='megszentségteleníthetetlenségeskedéseitekért'
x=part(s,[4:11])
y=part(s,[4:2:11])
```

Eredmény:

```
x=szentség
y=seté //csak minden 2. karaktert tartalmazza
```

String rész-stringjének indexe

```
strindex(string,rész-string)
```

Példa:

```
s='megszentségteleníthetetlenségeskedéseitekért'
x=strindex(s,'hetet')
y=strindex(s,'het')
z=strindex(s,'lens')
```

Eredmény:

```
x=19.
y=19.
z=24.
```

Rész-string cseréje

```
strsubst(string,mit,mire)
```

Példa:

```
s='Ez egy string.'
x=strsubst(s,'egy','egy másik')
```

Eredmény:

```
x=Ez egy másik string.
```

Szám konvertálása stringgé

`string(paraméter)`

A paraméter lehet 1 vagy több szám vagy változó, illetve intervallum is. Amennyiben intervallumot adunk meg, akkor a végeredmény egy string vektor lesz.
Példa:

```
s='A 3 és 5 számok összege: '+string(3+5)
sv=string(1:5)
```

Eredmény:

```
s=A 3 és 5 számok összege: 8
sv=! 1 2 3 4 5 !
```

String konvertálása számmá

```
evstr(string vagy string vektor);
```

A kimenet egy szám vagy szám vektor lesz attól függően, mi volt a paraméter. Amennyiben az átadott stringben nincsen szám, akkor a végeredmény üres lesz.

String konvertálása számmá

Példa:

```
s=string(3141791)
sv=string(1:5)
n=evstr(s)
nv=evstr(sv)
```

Eredmény:

```
s=3141791
sv=! 1  2  3  4  5  !
n=3141791.
nv=1. 2. 3. 4. 5.
```

Kimenet feliratozása

A kimeneti értékeket stringgé alakítjuk, és kiegészítjük további sztringekkel értelmes szöveggé.
Példa:

```
d = [0.5:0.25:1.5]
[n m] = size(d)
for j = 1:m, string(j) + '. távolság = '
                       + string(d(j)), end
```

Eredmény:

```
ans = 1. távolság = 0.5
ans = 2. távolság = 0.75
ans = 3. távolság = 1
ans = 4. távolság = 1.25
ans = 5. távolság = 1.5
```

Disp() függvény használata

A disp() függvény segítségével az előző példát tökéletesíthetjük az alábbi módon.
Példa:

```
d = [0.5:0.25:1.5]
[n m] = size(d)
for j=1:m, disp(string(j) + '. távolság = ' + string(d(j)) )
```

Eredmény:

```
1. távolság = 0.5
2. távolság = 0.75
3. távolság = 1
4. távolság = 1.25
5. távolság = 1.5
```

Az ans változó

Minden olyan érték, amit egy függvény vagy művelet visszaad, de nem mentettünk el változóba, az az ans-ba kerül eltárolásra. Ezt az értéket tetszőleges számban visszakérhetjük, de csak addig, amíg nem írjuk felül.
Példa:

```
3+5
exp(ans)
```

Eredmény:

```
ans=5.
ans=148.41316
```

Scilab programozás és a numerikus algoritmusok Scilabban
5. előadás
Ciklusszervezés

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

1. Lépésszámlálós cilklus
   - Általános szintaxis
   - Mintafeladat
   - Faktoriális
   - Mátrixfeltöltés
2. Elöltesztelős ciklus
   - Általános szintaxis
   - Mintafeladat
   - Végtelen ciklus
3. Rekurzió
   - Faktoriális
   - N. Fibonacci szám

Ciklusok szervezése

Ciklusokat kétféle módon készíthetünk Scilabban:

- Lépésszámlálós - for ciklus
- Elöltesztelős - while ciklus

Lépésszámlálós ciklus

Kétféle módon szervezhetünk lépésszámlálós ciklust.
Általános szintaxis:

```
for index = kezd : növ : vég
    ..műveletek (utasítások)..
  end
```

Ahol

- index: ciklusváltozó
- kezd: ciklusváltozó kezdőértéke
- növ: ciklusváltozó növekedése
- vég: ciklusváltozó végértéke

Lépésszámlálós ciklus

Második megadási mód:

```
for index = kezd : vég
    ..műveletek (utasítások)..
  end
```

Ahol

- index: ciklusváltozó
- kezd: ciklusváltozó kezdőértéke
- vég: ciklusváltozó végértéke

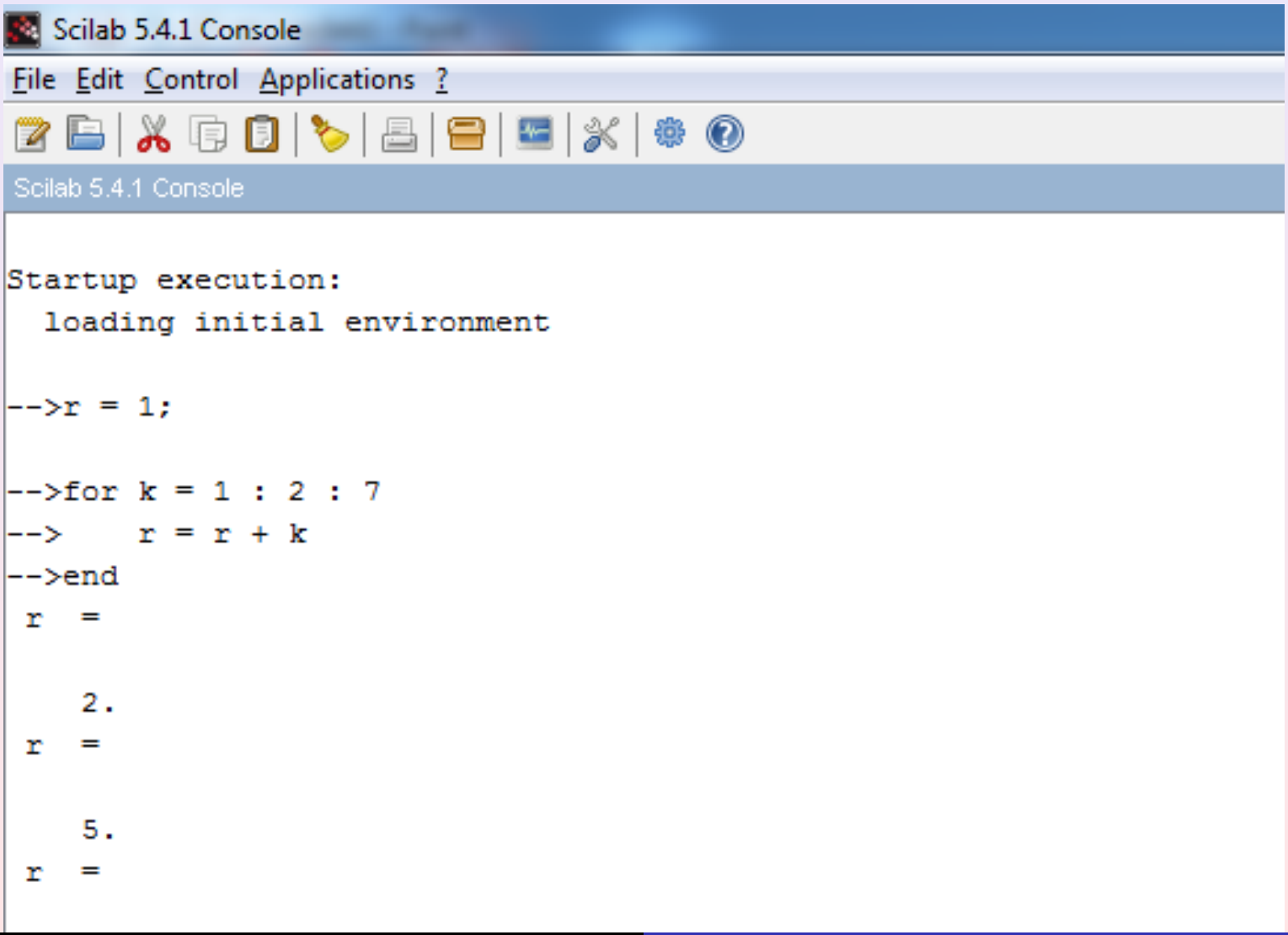
Ebben az esetben, azaz ha elhagyjuk a növekedést megadó változót, akkor azt a program alapértelmezetten 1-nek tekinti!

Mintafeladat

Gépeljük be a következő sorokat Scilabba, és az Enter leütése után figyeljük meg, hogyan működik a *for* ciklus.

```
r = 1;
for k = 1 : 2 : 7
    r = r + k
end
```

Mintafeladat

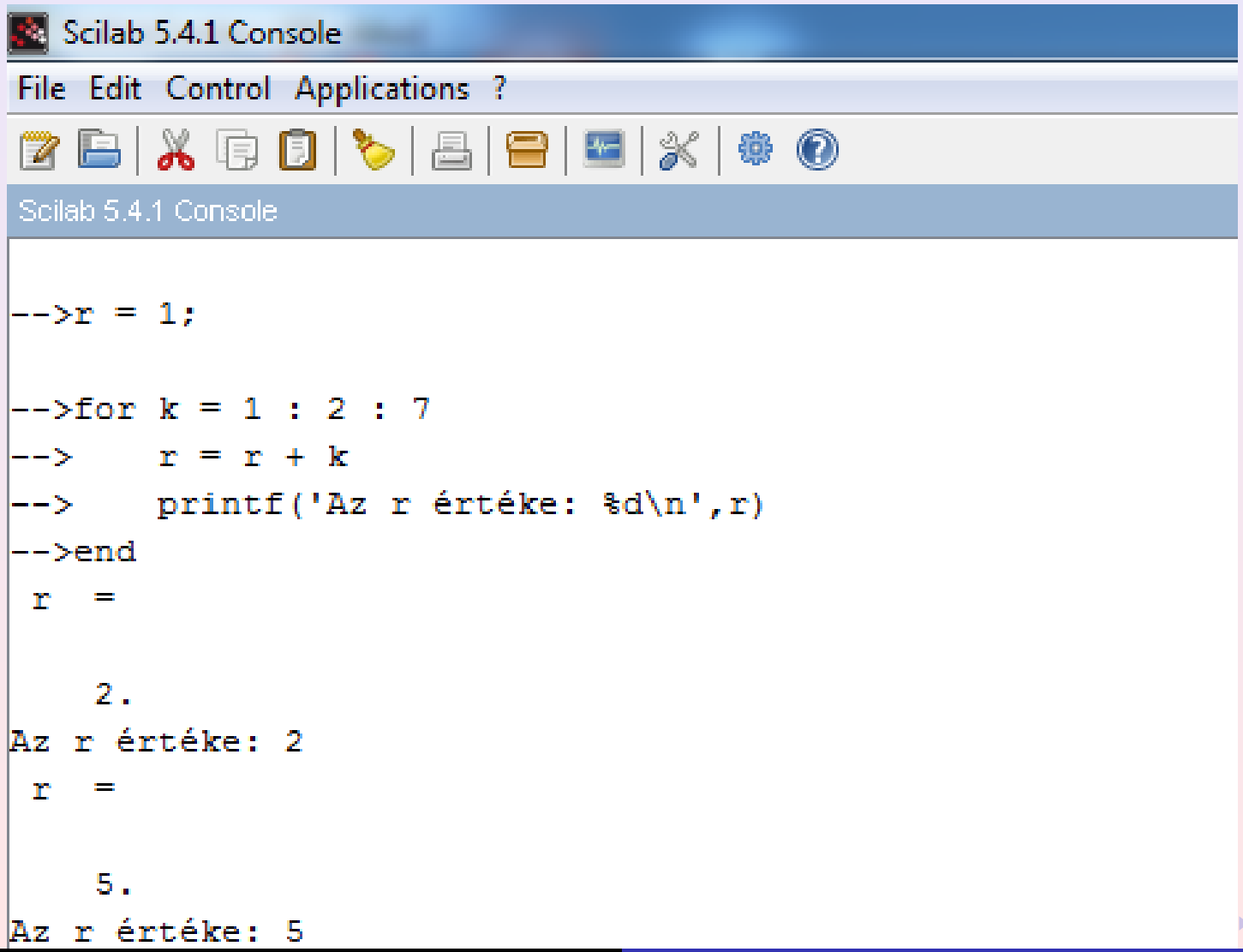


Mintafeladat

Figyeljük meg, hogy a program alapértelmezetten annak a változónak az értékét írja ki a képernyőre, mely a ciklus közbeni műveletek által módosul.
Amennyiben formázott szöveget szeretnénk megjeleníteni, használjuk a printf(' ') parancsot a ciklusban.

```
r = 1;
for k = 1 : 2 : 7
    r = r + k
    printf('Az r értéke: %d\n',r)
end
```

Mintafeladat

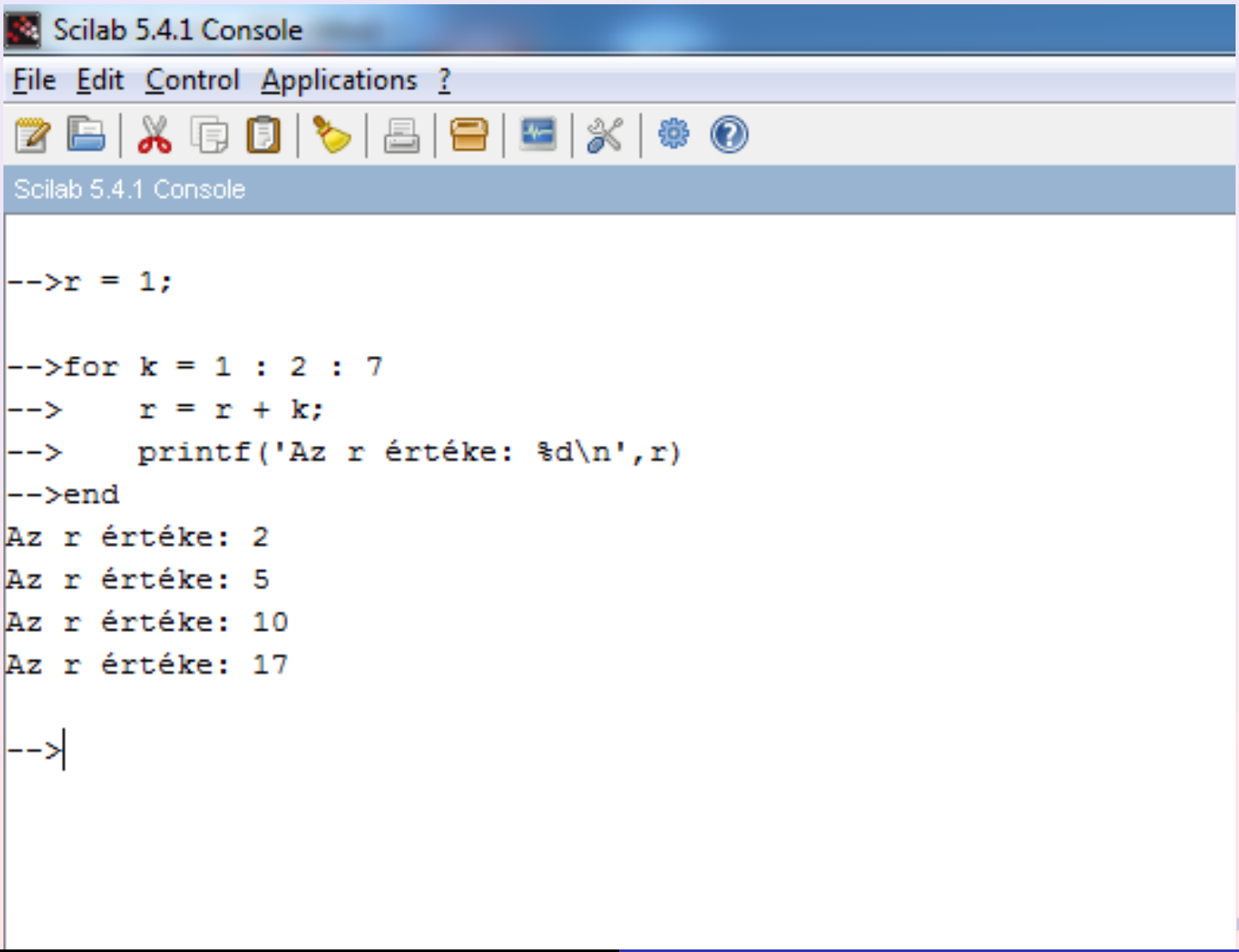


Mintafeladat

Amennyiben a mellékszámításokat nem szeretnénk megjeleníteni, ";" használatával az utasítások után megtehetjük:

```
r = 1;
for k = 1 : 2 : 7
    r = r + k;
    printf('Az r értéke: %d\n',r)
end
```

Mintafeladat



Mintafeladat

Amennyiben a feladatot paraméterként megkapott két érték között szeretnénk elkészíteni, a ciklust fájlban kell tárolnunk függvényként. Készítsük el külön fájlba az alábbi utasításokat:

```
function kettes(a,b)
for k = a : 2 : b
    a = a + k;
    printf('Az r értéke: %d\n',r)
end
endfunction
```

At eljárást a következőképpen futtathajuk:

```
kettes(1,7)
```

Mintafeladat

Scilab 5.4.1 Console

```
Startup execution:
  loading initial environment

-->exec('I:\_TaMOP\Numerikus_tananyag\Veglegesek\05\kettes.sci', -1)

-->kettes(1,7)
Az r értéke: 2
Az r értéke: 5
Az r értéke: 10
Az r értéke: 17

-->
```

Faktoriális számítása

Oldja meg az alábbi feladatot:
Készítsen eljárást, ami megadott paraméterig kiírja a számok faktoriálisát!

Faktoriális számítása

Egy lehetséges megoldás a következő:

```
function faktor(a)
f(1)=1
for i = 2 : a+1
    f(i)=f(i-1)*i
    printf('A(z) %d faktoriálisa: %d\n',i-1,f(i-1))
end
endfunction
```

At eljárást a következőképpen futtathatjuk:

```
faktor(5)
```

Faktoriális számítása

Scilab 5.4.1 Console

```
Startup execution:
  loading initial environment

-->exec('I:\_TaMOP\Numerikus_tananyag\Veglegesek\05\faktor.sci', -1)

-->faktor(5)
A(z) 1 faktoriálisa: 1
A(z) 2 faktoriálisa: 2
A(z) 3 faktoriálisa: 6
A(z) 4 faktoriálisa: 24
A(z) 5 faktoriálisa: 120

-->
```

Mátrixfeltöltés

Oldja meg az alábbi feladatot:
Készítsen eljárást, ami megadott méretű mátrixot készít, melynek elemeit egyesével kéri a felhasználótól! Az eljárás vágán irassa ki az elkészült mátixot!

Mátrixfeltöltés megvalósítása

Egy lehetséges megoldás a következő:

```
function mfeltolt(a,b)
for i = 1 : a
    for j = 1 : b
     A(i,j)=input('Kérem a következő elemet: ')
    end
end
disp(A)
endfunction
```

At eljárást a következőképpen futtathatjuk:

```
mfeltolt(2,2)
```

Mátrixfeltöltés futtatási eredmény

Scilab 5.4.1 Console

```
-->exec('I:\_TaMOP\Numerikus_tananyag\Veglegesek\05\mfeltolt.sce', -1)

-->mfeltolt(2,2)
Kérem a következő elemet: 2
Kérem a következő elemet: -3
Kérem a következő elemet: 4
Kérem a következő elemet: 2

    2.   - 3.
    4.     2.

-->
```

Elöltesztelős ciklus

Három lehetséges szintaxisa van az elöltesztelős ciklusnak. A három megadási mód ugyanúgy működik.

```
while feltétel utasítások1,...
[,else utasítások2], end
while feltétel do utasítások1,...
            [,else utasítások2], end
while feltétel then utasítások1,...
            [,else utasítások2], end
```

Ahol a *feltétel* valamilyen logikai kifejezés (ez lesz a leállási feltétel), a *utasítások1* a ciklus magja, *utasítások2* a ciklus befejezése utáni műveletek.
A *feltétel*ben szereplő összes változónak értékkel kell rendelkeznie a ciklus kezdetén!

Mintafeladat

Gépeljük be a következő sorokat Scilabba, és az Enter leütése után figyeljük meg a példákon, hogyan is működik a *while* ciklus.

```
s = 100;
while s > 50
disp(s^2)
 s = s - 5;
 end
```

Futási eredmény

Scilab 5.4.1 Console

```
-->s = 100;

-->while s > 50
-->disp(s^2)
--> s = s - 5;
--> end

    10000.

    9025.

    8100.

    7225.

    6400.

    5625.

    4900.

    4225.

    3600.

    3025.

-->
```

Faktoriális

Az előzőekben elkészítettük a faktoriális kiszámítását végző eljárást.
Alakítsuk át elöltesztelős cikluszervezési módon.
Emlékezetőül a lépésszámlálós megoldás:

```
function faktor(a)
f(1)=1
for i = 2 : a+1
    f(i)=f(i-1)*i
    printf('A(z) %d faktoriálisa: %d\n',i-1,f(i-1))
end
endfunction
```

Faktoriális

Egy lehetséges megoldás:

```
function faktor(a)
i=2;
f(1)=1
while i<=a+1 do
    f(i)=f(i-1)*i
    printf('A(z) %d faktoriálisa: %d\n',i-1,f(i-1))
    i=i+1;
end
endfunction
```

Futtatása ugyanúgy történik, mint a "for" ciklus esetén:

```
faktor(5)
```

Faktoriális

Figyeljük meg a következőket:

- Az "i" cilkusváltozó a ciklus előtt kapott kezdőértéket, hiszen az első futáskor már kiértékelődik.
- A ciklusváltozó nem növekszik automatikusan, a ciklus magjában kell növelnünk.

Faktoriális futtatási eredménye

Scilab 5.4.1 Console

```
Startup execution:
  loading initial environment

-->exec('I:\_TaMOP\Numerikus_tananyag\Veglegesek\05\faktor.sci', -1)

-->faktor(5)
A(z) 1 faktoriálisa: 1
A(z) 2 faktoriálisa: 2
A(z) 3 faktoriálisa: 6
A(z) 4 faktoriálisa: 24
A(z) 5 faktoriálisa: 120

-->
```

Végtelen ciklus

Tekintsük az alábbi kódot:

```
function vegtelen(a)
i=0;
while i<=a do
    printf('A(z) %d négyzete: %d\n',i,i^2)
end
endfunction
```

Tetszőleges paraméterrel futtatva a végtelen ciklust készítettünk. Leállítására a CTRL+X billenytűkombinációval van lehetőségünk és utána az abort utasítással kapjuk vissza a promptot. Menüsorban a Control menüben található abort menüpont szakítja meg az éppen aktuálisan futó programot.

Futtatási eredmény

```
A(z) 0 négyzete: 0
A(z) 0 négyzete: 0
A(z) 0 négyzete: 0
A(z) 0 négyzete: 0
A(z) 0 négyzete: 0
A(z) 0 négyzete: 0
A(z) 0 négyzete: 0
A(z) 0 négyzete: 0
A(z) 0 négyzete: 0

-1->abort

-->
```

Megoldási lehetőségek

A végtelen ciklus elkerülésére vezessünk be egy változót, ami csak a ciklus lépésszámlálására használunk. Amennyiben a lépésszám eléri a megadott paramétert, állítsa meg a ciklus futását.

```
function vegtelen(a)
i=0;
l=1
while i<=a do
    if l>a
        printf('Valami elromlott,
                    a ciklus futása leáll.')
break
    end
    printf('A(z) %d négyzete: %d\n',i,i^2)
    l=l+1;
end
endfunction
```

Futtatási eredmény

Scilab 5.4.1 Console

```
-->exec('I:\_TaMOP\Numerikus_tananyag\Veglegesek\05\végtelen.sci', -1)
Warning : redefining function: vegtelen                . Use funcprot(0) to avoid


-->vegtelen(4)
A(z) 0 négyzete: 0
A(z) 0 négyzete: 0
A(z) 0 négyzete: 0
A(z) 0 négyzete: 0
Valami elromlott, a ciklus futása leáll.
-->
```

Ciklus megszakítása

Ciklus futásának megszakítására két lehetőségünk van:

- break - Ciklus futását megszakítja, a ciklus után következő utasításokat végrehajtja.
- abort - A teljes eljárás futását megszakítja. Utána visszakapjuk a promptot.

Faktoriális számítása rekurzióval

Sok feladat oldható meg ciklusok szervezése helyett rekurzív módon. A faktoriális számítása az egyik legnépszerűbb rekurzív ódon számolható feladat.

- Leállási feltétel ("triviális eset"): $n = 0$.
- Rekurzív hívás: faktrec(n-1)*n
- Függvényhívás: faktrec(n)

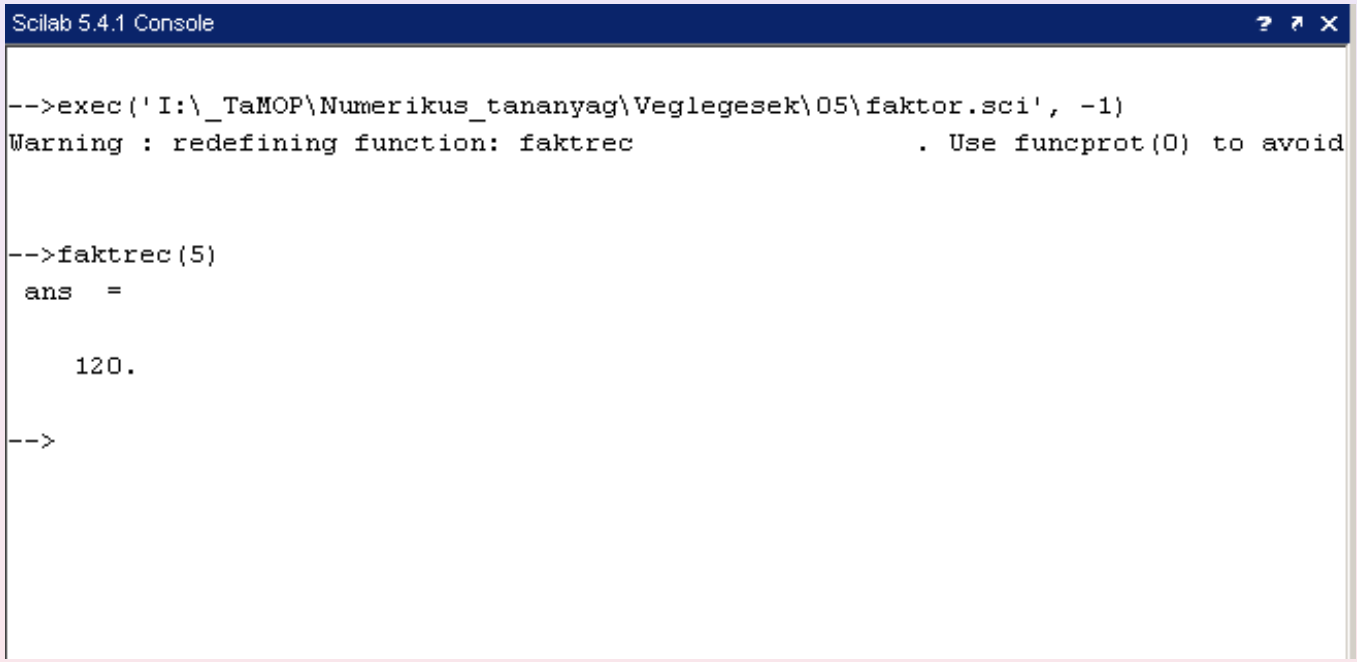
Ahol "n" pozitív egész szám.

Faktoriális számítása rekurzióval

Egy lehetséges magvalósítás Scilabban:

```
function y=faktrec(n)
      if  n==0 then
          y=1;
      else
          y=faktrec(n-1)*n;
    end
endfunction
```

Futtatási eredmény



N. Fibonacci szám számítása rekurzióval

Az n. Fibonnaci szám kiszámítása is megoldható rekurzív módon.

- Leállási feltétel ("triviális eset"): $n < 2$.
- Rekurzív hívás: fibonacci(n-1)+fibonnacci(n-2)
- Függvényhívás: fibonacci(n)

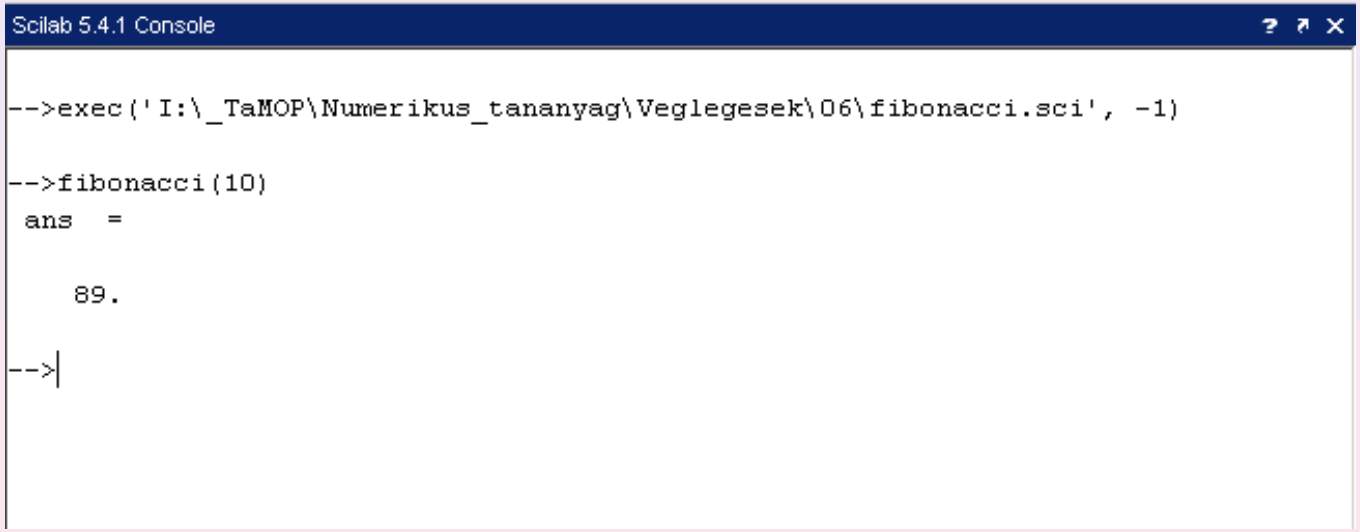
Ahol "n" pozitív egész szám. Megoldásunkban a leállási feltételt kettébontjuk. Az $n < 2$ feltétel egyenértékű a $n = 1$ vagy $n = 0$ feltételekkel.

N. Fibonacci szám számítása rekurzióval

Egy lehetséges magvalósítás Scilabban:

```
function y=fibonacci(n)
    select n
    case 0 then
        y=1
    case 1 then
        y=1
    else
        y=fibonacci(n-1)+fibonacci(n-2)
    end
endfunction
```

Futtatási eredmény



Scilab programozás és a numerikus algoritmusok Scilabban
6. előadás
Fixpont iteráció

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

1 Tételek

2 1. feladat

3 Scilab megvalósítás

4 2. feladat

5 3. feladat

6 Rekurzió

Fixpont iteráció - bevezetés

A p számot a g függvény fixpontjának nevezzük, ha

$$g(p) = p.$$

A numerikus analízisben szereplő sorozatokat gyakran ***rekurzív definíció***val, más néven ***iteráció***val adunk meg.
Ebben a szakaszban a leggyakoribb esettel, az egylépéses iterációval, ezen belül a ***fixpont iteráció***val foglalkozunk részletesebben.

Fixpont iteráció - bevezetés (folyt.)

Egy $p_{k+1} = h(p_k, p_{k-1}, ..., p_{k-m+1})$ $(k \geq m-1)$ rekurzív definícióval megadott iterációs módszert m-lépéses iterációnak nevezzük.
Egy m lépéses iterációs sorozatot m kezdeti érték, $p_0, p_1, ..., p_{m-1}$ határoz meg egyértelműen.

Fixpont tétel

Legyen $g : [a, b] \to [a, b]$ folytonos függvény, g differenciálható (a, b)-n, és tegyük fel hogy létezik olyan $0 \leq c < 1$ szám, hogy $|g'(x)| \leq c$ minden $x \in (a, b)$ -re.
Legyen $p_0 \in [a, b]$ tetszőleges, és $p_{k+1} = g(p_k) (k \geq 0)$.
Létezik (egyetlen) $[a; b]$-beli p fixpontja a g-nek és a p_k sorozat konvergál a függvény p fixpontjához, továbbá

$$|p_k - p| \leq c^k |p_0 - p|,$$

valamint

$$|p_k - p| \leq \frac{c^k}{1-c} |p_1 - p_0| .$$

Fixpont iterációval kapcsolatos további tétel

Legyen $g : [a, b] \to [a, b]$ (vagy $R \to R$) folytonos függvény, $p_0 \in [a, b]$ rögzített, és tekintsük a $p_{k+1} = g(p_k)$ fixpont iterációs sorozatot.
Ha p_k konvergens és $p_k \to p$, akkor $p = g(p)$.

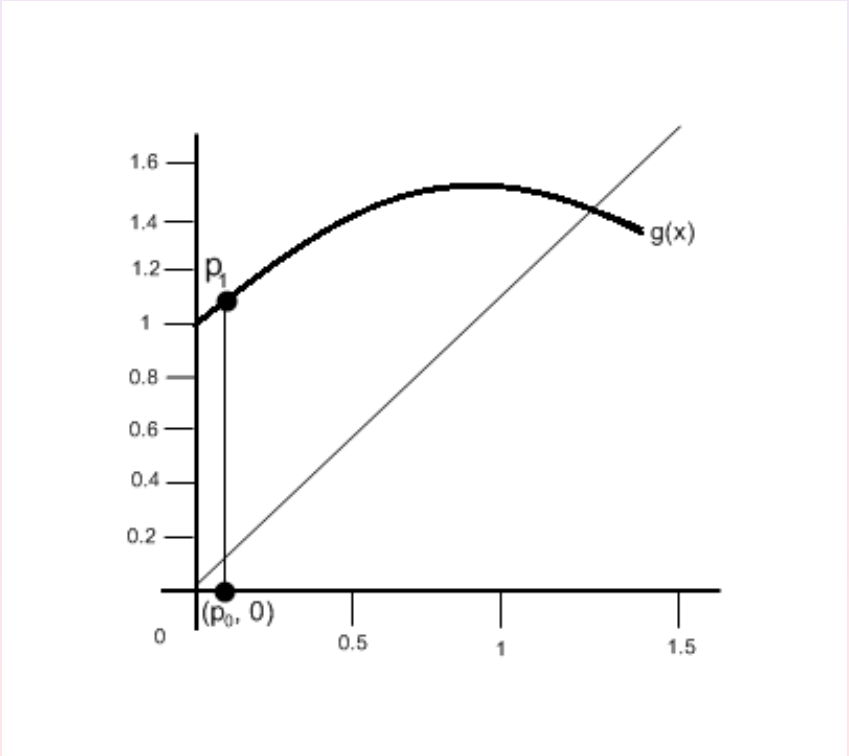
Példa fixpont iterációra

Tekintsük a $g(x) = -0.5x^2 + x + 1$ függvényt! Legyen az iteráció kezdőpontja $p_0 = 0,1$. Ekkor az első néhány tagja az iterációs sorozatnak a következő számítással kapható meg:

$$\begin{aligned} p_1 &= g(p_0) &&= 1,095 \\ p_2 &= g(p_1) &&= 1,4954875 \\ p_3 &= g(p_2) &&= 1,37724606 \\ p_4 &= g(p_3) &&= 1,4288427 \\ p_5 &= g(p_4) &&= 1,4080469685 \end{aligned}$$

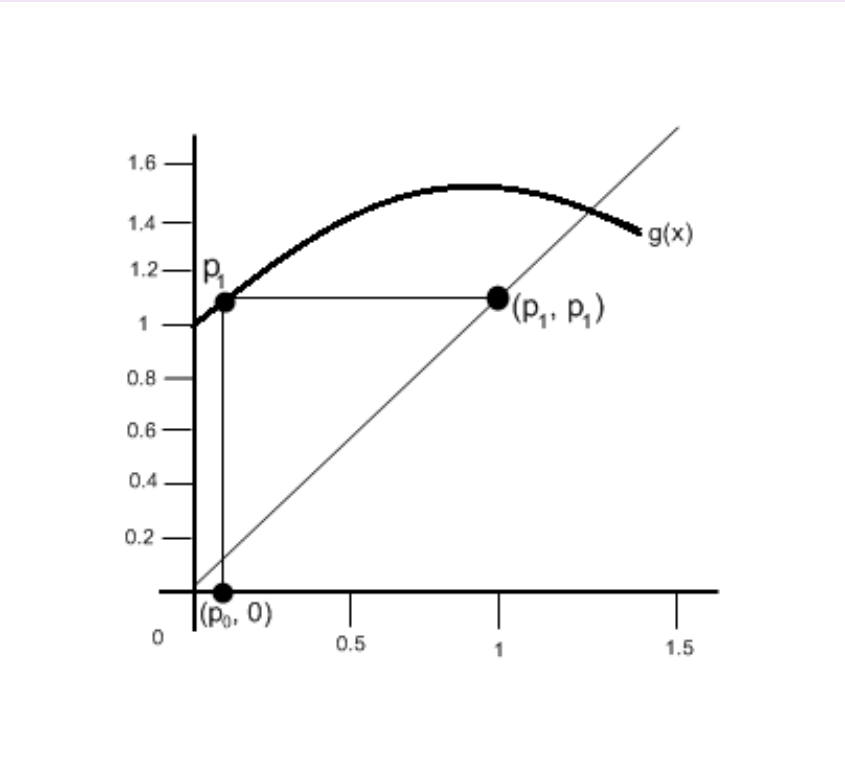
Példa fixpont iterációra (folyt.)

A kiindulási $(p_0, 0)$ pontból rajzolunk egy függőleges egyenest a g függvény grafikonjáig. A kimetszett pont y-koordinátája adja a sorozat p_1 elemét.



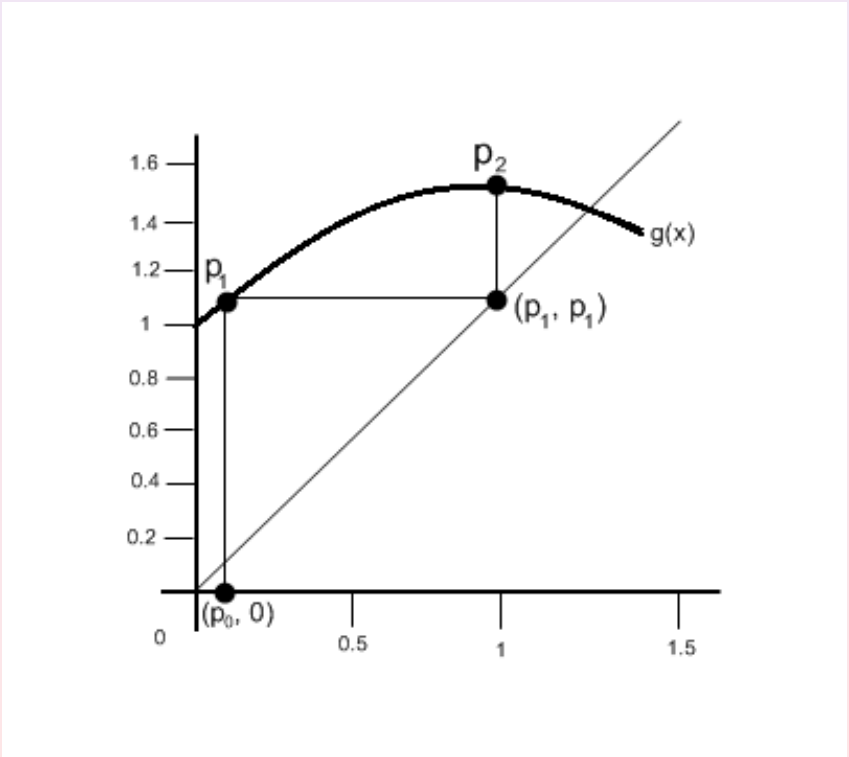
Példa fixpont iterációra (folyt.)

A (p_0, p_1) pontból egy vízszintes szakaszt rajzolunk az y = x egyenes (p_1, p_1) pontjáig.



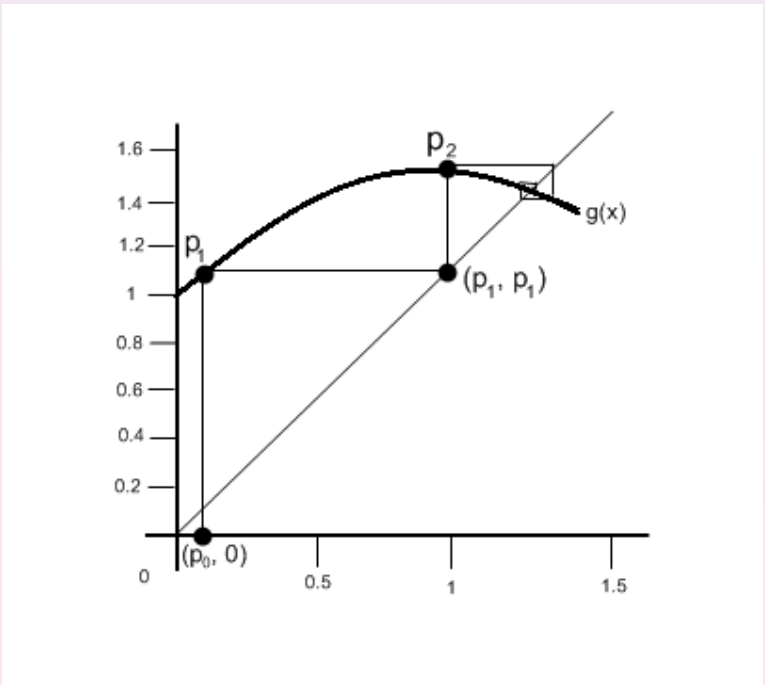
Példa fixpont iterációra (folyt.)

A sorozat $p_2 = g(p_1)$ elemét úgy kapjuk geometriailag, ha ebből a pontból egy függőleges szakasz mentén elmegyünk a g grafikonjáig, és a kimetszett pont y-koordinátája lesz p_2.



Példa fixpont iterációra (folyt.)

Ezt az eljárást folytatva kapjuk az ábrán látható törött vonalat. A töröttvonal ennél a példánál spirálisan ráhúzódik az $y = x$ egyenes és az $y = g(x)$ görbe egyik metszéspontjára. A metszéspont koordinátái $(\sqrt{2}, \sqrt{2})$.



Példa fixpont iterációra (folyt.)

Az előbbi példában azt tapasztaltuk, hogy a fixpont sorozat az y=x egyenes és az y = g(x) grafikon metszéspontjának x-koordinátájához konvergál. Ennek a pontnak az x-koordinátája (és persze az y-koordinátája is) a g(x) = x egyenletet teljesíti.

Fixpont iteráció Scilab megvalósítása

A fixpont iteráció három bejövő paramétert használ:

- g: a függvény, aminek a fixpontját keressük
- a: kezdőpont
- t: pontosság

Az algoritmus közelítő eljárás, célszerű while ciklusszervezési módot használni. Leállási feltételként az egymást követő lépések eltérésének abszolút értékét használhatjuk. Amennyiben ez az eltérés kisebb lesz, mint a megadott pontosság, az algoritmus megáll és az utoljára kiszámolt értéket adja visszatérési értéknek.
Ha az eljárás nem találja meg a fixpontot 100 iterációs lépés alatt, akkor leállítjuk a futását. Természetesen ez az érték megváltoztatható.

Fixpont algoritmus Scilab megvalósítása 2

Az algoritmus:

```
function fixpont(g,a,t)
 printf('A kezdőpont: %1.1f\n',a)
  l=1;
  while abs(a-g(a)) >= t & l<100
    a=g(a)
    printf('%d. lépésben a közelítő megoldás:
                                        %1.14f\n',l,a)
    l=l+1;
  end
printf('%d. lépésben a közelítő megoldás:
%1.14f\n',l,g(a))
endfunction
```

Kezdőpont keresés

Amennyiben olyan feladatot kell megoldani, amelyben a kezdőpont nincs megadva, próbálgatás helyett ábra segítségével dolgozzunk. Tekintsük a következő függvény:

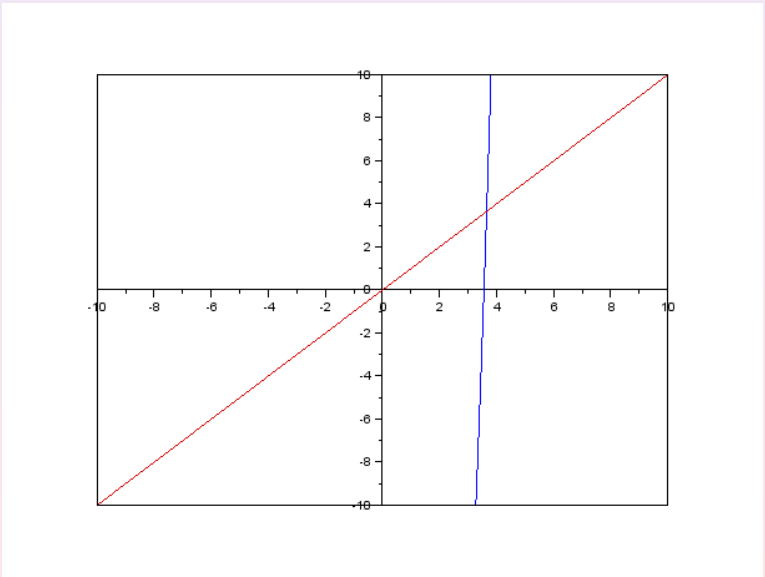
$$g(x) = x^3 - 45$$

Készítsük el a függvényt és az $y = x$ egyenest Scilabban és ábrázoljuk:

```
function y=fgv(x)
y=x^3-45
endfunction
function y=fgvx(x)
y=x
endfunction
plot(-10:0.01:10,fgv)
plot(-10:0.01:10,fgvx)
```

Kezdőpont keresés

Az ábrán látható metszéspont lesz a függvény fixpontja. Az eljárás kezdőpontját célszerű ezen pont közelében választani.



Numerikus példa

A példában azonban bármilyen közel is adjuk meg a kezdőpontot a reménybeli fixponthoz, az eljárás 7-10 lépésben divergál.

```
1. lépésben a közelítő megoldás: -18.00000000000000
2. lépésben a közelítő megoldás: -5877.00000000000000
3. lépésben a közelítő megoldás: -202986461178.00000000000000
4. lépésben a közelítő megoldás: -836375334767964240000000000000000.00000000000000
5. lépésben a közelítő megoldás: -585064369274328760000000000000000000000000000000000000000000000000000000000000000000000000000000
6. lépésben a közelítő megoldás: -200267718596661300000000000000000000000000000000000000000000000000000000000000000000000000000000
7. lépésben a közelítő megoldás: -Inf
```

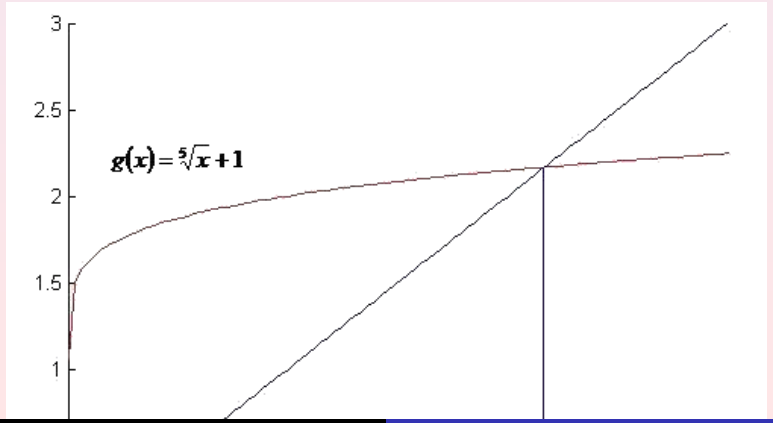
Ennek oka a Fixpont tétel segítségével adható meg. A $[3;4]$ intervallumon a függvény deriváltja ($g'(x) = 3x^2$ értéke 27 és 48 között van, ami nagyobb, mint 1.

Numerikus példa

Határozzuk meg az $x - \sqrt[5]{x} - 1 = 0$ egyenlet gyökét $\varepsilon = 10^{-6}$ hibakorláttal!

1. A gyököt tartalmazó intervallum meghatározása

Rendezzük át a fenti egyenletet az $x = \sqrt[5]{x} + 1$ fixpont iterációs alakra. Ábrázoljuk a $g(x) = \sqrt[5]{x} + 1$ és $y(x) = x$ függvényeket közös koordináta rendszerben. Az alábbi ábrából látható, hogy az eredeti egyenlet gyöke, azaz a g(x) függvény fixpontja benne van az $[1, 3]$ intervallumban.



Numerikus példa (folyt.)

2. konvergencia vizsgálata

Vizsgáljuk meg a konvergencia feltételeket az $[1;3]$ intervallumra nézve.

Mivel $|g'(x)| = |\frac{1}{5\sqrt[5]{x^4}}| \leq |\frac{1}{5\sqrt[5]{1^4}}| = \frac{1}{5} = c < 1$ és

$1 < g(1) \leq g(x) \leq g(3) < 3$ $x \in [1;3]$ esetén, ezért a módszer konvergens az adott intervallumon.

Numerikus példa (folyt.)

Készítsük el a függvényt és az $y = x$ egyenest Scilabban és ábrázoljuk:

```
function y=fgv(x)
y=nthroot(x,5)+1
endfunction
```

Futtassuk a fixpont iterációs eljárást az alábbi paraméterekkel:

```
fixpont(fgv,2,0.000001)
```

Numerikus példa (folyt.)

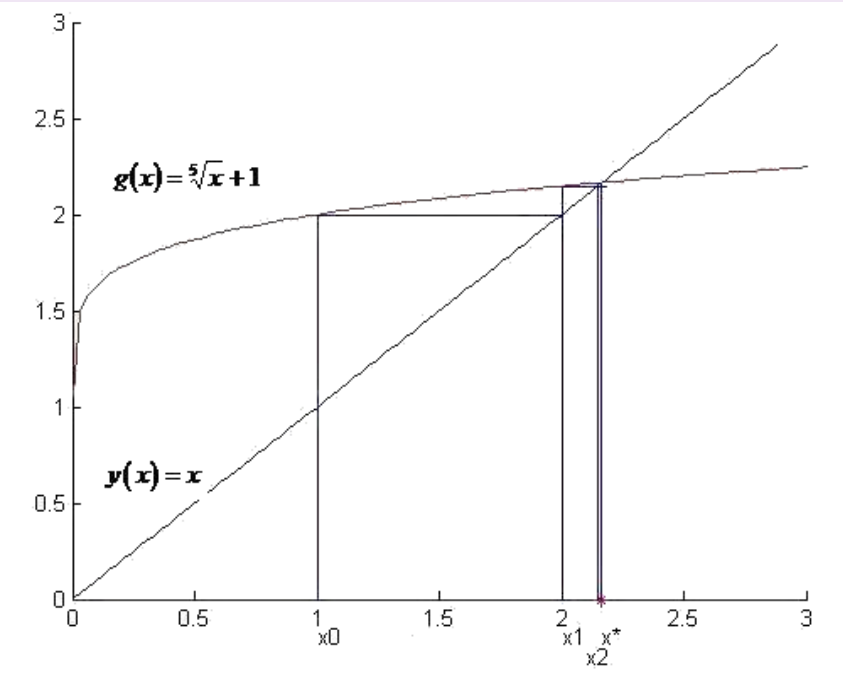
Az alábbi eredményt kapjuk:

```
A kezdőpont: 2.0
1. lépésben a közelítő megoldás: 2.14869835499703
2. lépésben a közelítő megoldás: 2.16529287293869
3. lépésben a közelítő megoldás: 2.16708726264560
4. lépésben a közelítő megoldás: 2.16728063284039
5. lépésben a közelítő megoldás: 2.16730146349464
6. lépésben a közelítő megoldás: 2.16730370737199
7. lépésben a közelítő megoldás: 2.16730394908136
```

Mivel az utolsó két lépés eltérése $2,42 * 10^{-7} < \varepsilon$, ezért $p_7 = 2,1673039$ a gyök adott hibakorlátnak megfelelő közelítő érték.

Numerikus példa (folyt.)

Az alábbi ábrán az első két közelítés meghatározása látható grafikusan:

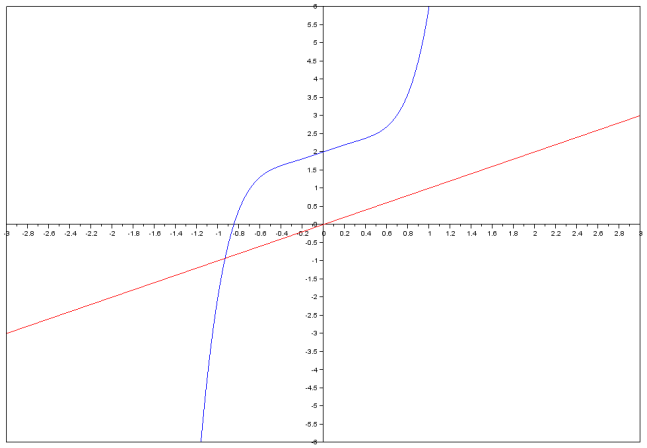


Feladat

Alakítsuk át a következő egyenletet fixpont egyenletté, majd fixpont iteráció segítségével adjuk meg az egyenlet közelítő megoldását 4 tizedesjegy pontossággal: $4x^5 - x^3 + 2 = 0$.

1. átalakítás:

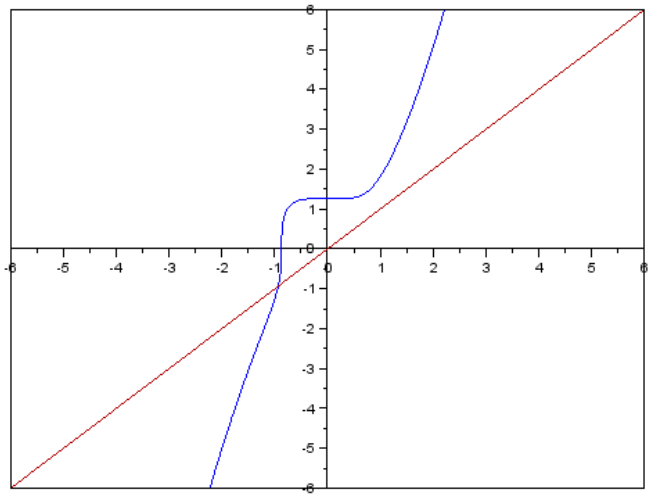
$$4x^5 - x^3 + x + 2 = x$$



Feladat (folyt.)

2. átalakítás:

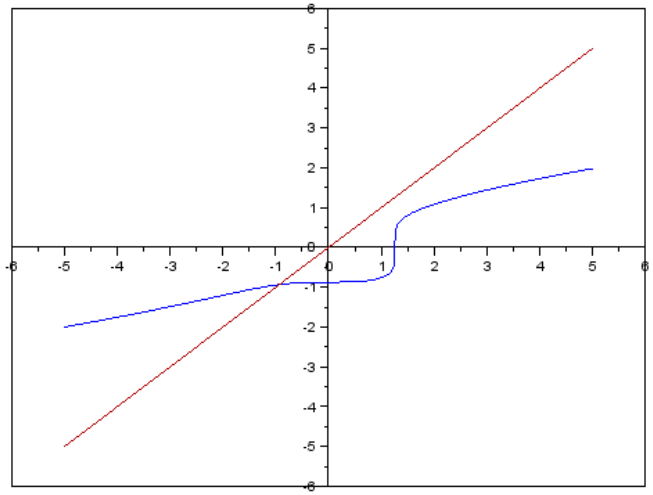
$$4x^5 + 2 = x^3$$
$$\sqrt[3]{4x^5 + 2} = x$$



Feladat (folyt.)

3. átalakítás:

$$-x^3 + 2 = -4x^5$$
$$\sqrt[5]{\frac{x^3-2}{4}} = x$$

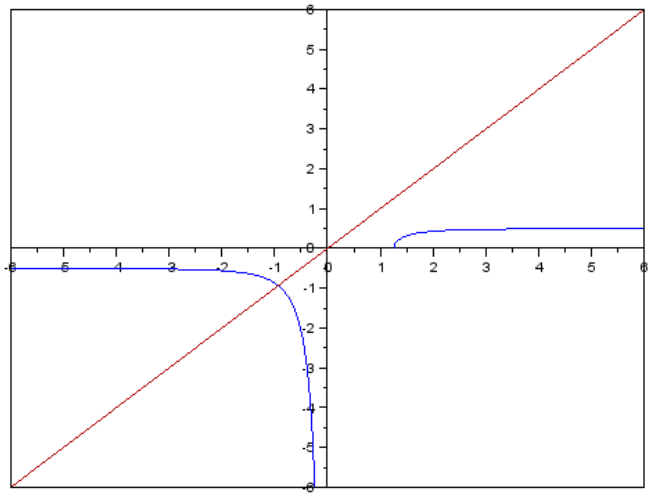


Feladat (folyt.)

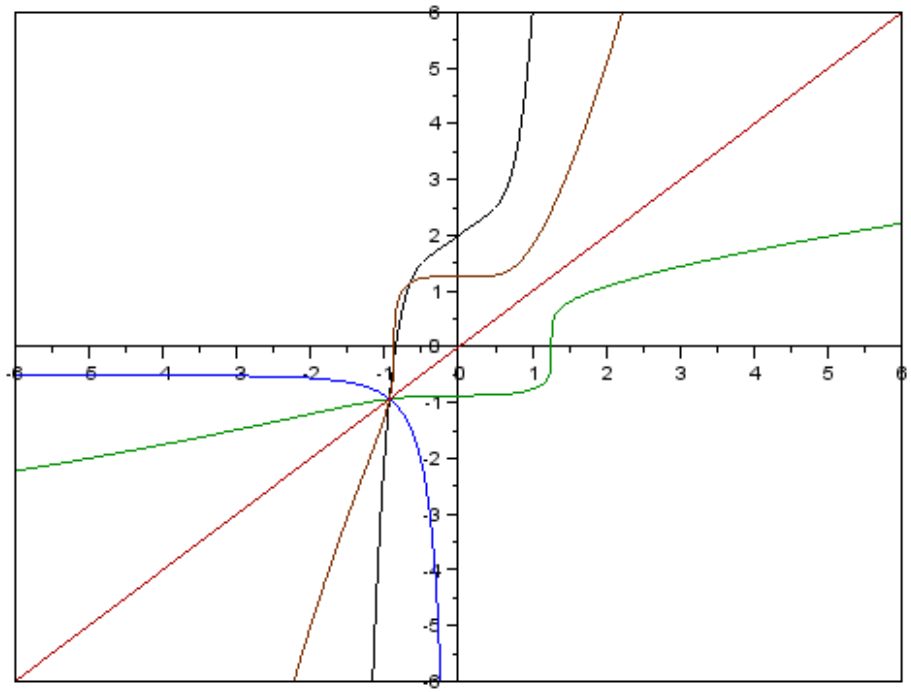
4. átalakítás:

$$x^3(4x^2 - 1) = -2$$

$$|x| = \sqrt{\frac{x^3 - 2}{4x^3}}$$



Feladat (folyt.)



Feladat

1. átalakítás : $g(x) = 4x^5 - x^3 + x + 2$

```
function y=fgv(x)
y=4*x^5-x^3+x+2
endfunction
```

```
A kezdőpont: -0.9
1. lépésben a közelítő megoldás: -0.53296000000000
2. lépésben a közelítő megoldás: 1.44642351868458
3. lépésben a közelítő megoldás: 25.74459665089742
4. lépésben a közelítő megoldás: 45219616.24235416900000
5. lépésben a közelítő megoldás: 7563004402204512800000000000000000000000.00000000000000
6. lépésben a közelítő megoldás: 9897642365379110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Feladat

2. átalakítás : $g(x) = \sqrt[3]{4x^5 + 2}$

```
function y=fgv(x)
y=nthroot(4*x^5+2,3)
endfunction
```

```
A kezdőpont: -0.9
1. lépésben a közelítő megoldás: -0.71266734552869
2. lépésben a közelítő megoldás: 1.08140924625260
3. lépésben a közelítő megoldás: 1.99295502895391
4. lépésben a közelítő megoldás: 5.03655129002919
5. lépésben a közelítő megoldás: 23.49260073102338
6. lépésben a közelítő megoldás: 305.89461853497556
7. lépésben a közelítő megoldás: 22044.81160110465900
8. lépésben a közelítő megoldás: 27512537.64555848800000
9. lépésben a közelítő megoldás: 3980195583414.40040000000000
10. lépésben a közelítő megoldás: 158681885733401470000.00000000000000
11. lépésben a közelítő megoldás: 342688017105221010000000000000000000.00000000000000
12. lépésben a közelítő megoldás: 266390169828095590000000000000000000000000000000000000000000000.00000000000000
13. lépésben a közelítő megoldás: 175072119003187280000000000000000000000000000000000000000000000000000000000000000000000000000000000
14. lépésben a közelítő megoldás: Inf
```

Feladat

3. átalakítás: $g(x) = \sqrt[5]{\frac{x^3-2}{4}}$

```
function y=fgv(x)
y=nthroot((x^3-2)/4,5)
endfunction
```

```
A kezdőpont: -0.9
1. lépésben a közelítő megoldás: -0.92637901568858
2. lépésben a közelítő megoldás: -0.93081700158494
3. lépésben a közelítő megoldás: -0.93158042220700
4. lépésben a közelítő megoldás: -0.93171222833380
5. lépésben a közelítő megoldás: -0.93173499925490
```

Feladat

4. átalakítás: $g(x) = -\sqrt{\frac{x^3 - 2}{4x^3}}$

```
function y=fgv(x)
y=-sqrt((x^3-2)/(4*x^3))
endfunction
```

```
92. lépésben a közelítő megoldás: -0.63408077676053
93. lépésben a közelítő megoldás: -1.48703222650837
94. lépésben a közelítő megoldás: -0.63408032301108
95. lépésben a közelítő megoldás: -1.48703364223389
96. lépésben a közelítő megoldás: -0.63407998054796
97. lépésben a közelítő megoldás: -1.48703471074120
98. lépésben a közelítő megoldás: -0.63407972207744
99. lépésben a közelítő megoldás: -1.48703551718709
100. lépésben a közelítő megoldás: -0.63407952699966
```

Feladat konvergenciavizsgálata

A négy átalakítás közül a harmadik találta meg a fixpontot. Az ok ismét a Fixpont-tételben keresendő: Vizsgáljuk meg az első két átalakítás deriváltjait a $[-1; -0,8]$ intervallumon:

$$g'(x) = 20x^4 - 3x^2 + 1 > g'(-0,8) = 11,692 > 1$$

$$g'(x) = \frac{20x^4}{3\sqrt[3]{(4x^5+2)^2}} > g'(-1) = 3,612083975 > 1$$

Mindkét esetben a derivált értéke a fixpont környezetében nagyobb, mint 1.

Feladat konvergenciavizsgálata

A harmadik átalakítás:

$$g'(x) = \frac{12x^2}{5\sqrt[5]{(8x^3-16)^4}} < g'(-1) = 0,1888175023 < 1$$

Ez az átalakítás találta meg a megoldást.

Feladat konvergenciavizsgálata

A negyedik átalakítás érdekesen viselkedik:

$$g'(x) = \frac{3}{4\sqrt{\frac{x^3-2}{x^3}x^4}}) < g'(-0,9) = 0,5908173249$$

Ez alapján azt várnánk, hogy konvergens lesz az átalakítás. Azonban nem találta meg a fixpontot az eljárás. Ennek oka, hogy az eljárás során két érték között ($-1,487$ és $-0,634$) ugrál az iteráció. Ezen két pont körül a második esetében a derivált függvény értéke nagyobb egynél.

Rekurzív megoldás

A fixpont iteráció lehetséges programozási megoldása a rekurzív mód.
A következő paramétereket használjuk:

- Leállási feltétel ("triviális eset"): $|a - f(a)| \leq t$.
- Leállási feltétel (divergencia elkerülése): $l < 100$.
- Függvényhívás: fixpontr(f,a,t,l)

A lépésszámot paraméterként kell átadnunk, mivel a rekurzió során a növekedését figyelembe kell vennünk.

Rekurzív megvalósítás

```
function fixpontr(f,a,t,l)
    if abs(a-f(a)) <= t & l<100
        printf('%d. lépésben a megfelelő pontosságú
                közelítő megoldás: %1.14f\n',l,f(a))
    else
        printf('%d. lépésben a közelítő megoldás:
                                    %1.14f\n',l,f(a))
        fixpontr(f,f(a),t,l+1)
    end
endfunction
```

Rekurzív megvalósítás futtatása

Ellenőrzésképpen futtassuk le az eljárást a harmadik feladat harmadik átalakítására.

```
function y=fgv(x)
y=nthroot((x^3-2)/4,5)
endfunction
```

A futtató utasítás:

```
fixpontr(fgv,-0.9,0.0001,1)
```

Rekurzív megvalósítás eredménye

```
1. lépésben a közelítő megoldás: -0.92637901568858
2. lépésben a közelítő megoldás: -0.93081700158494
3. lépésben a közelítő megoldás: -0.93158042220700
4. lépésben a közelítő megoldás: -0.93171222833380
5. lépésben a megfelelő pontosságú közelítő megoldás: -0.93173499925490
```

A futtatás ugyanazt az eredményt adta, mint a nem rekurzív megvalósítás.

Scilab programozás és a numerikus algoritmusok Scilabban
7. előadás
Intervallum felezés és húrmódszer

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

1. Intervallum felezés
   - Bevezetés
   - Mintafeladat
   - Specifikáció
   - Scilab megvalósítás
   - Mintafeladat Scilab megoldása
   - Scilab megvalósítás "for" ciklusszervezéssel
2. Húrmódszer
   - Bevezetés
   - Mintafeladat
   - Specifikáció
   - Scilab megvalósítás

Fogjunk oroszlánt a sivatagban!

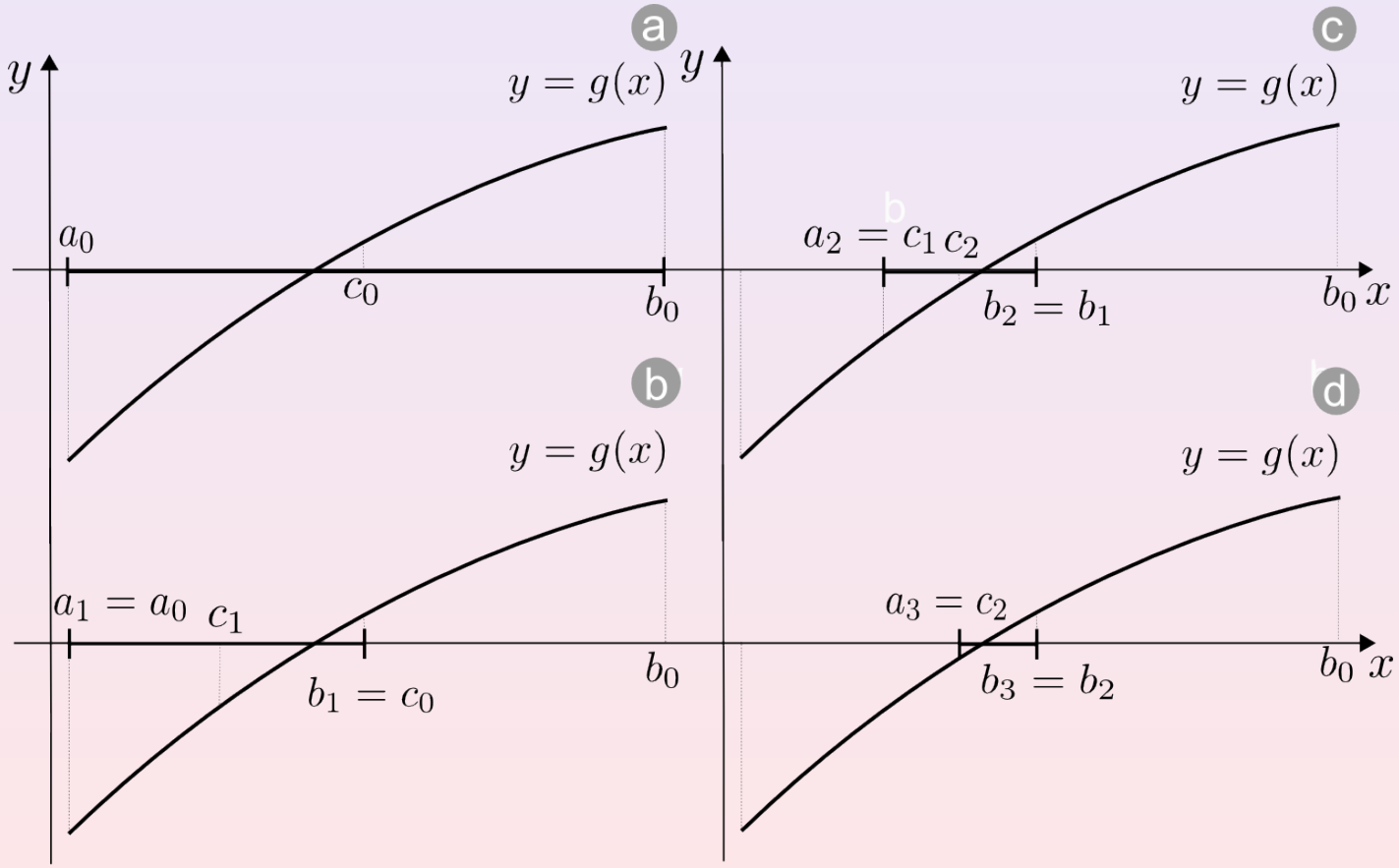
Egy matematikus oroszlánt akar fogni, ehhez rendelkezésére áll egy oroszlán-detektor. A módszer csak azt tudja megmondani, hogy a vizsgált területen van-e az oroszlán. Matematikusunk körbekeríti a területet, majd megfelezi azt. A módszert használva megállapítja, hogy melyik félterületben van az oroszlán. A másik féllel többet nem foglalkozik. A területet újra megfelezi. Ezt az eljárást addig folytatja amíg a terület elég kis részét kerítette körbe ahhoz, hogy elfogja az oroszlánt.

Intervallum felezés

Kicsit matematikusabban...

Adott egy folytonos $f(x)$ függvény, amiről tudjuk, hogy valahol $[a; b]$-ben pontosan egyszer metszi az x tengelyt. Az intervallumot minden lépésben megfelezzük és csak azzal a résszel foglalkozunk tovább, amelyikben a megoldásunk található. Mindezt a kívánt pontosság eléréséig folytatjuk.

Intervallum felezés



Intervallum felezés

Előnyei:

- Egyszerű gyökkereső algoritmus.
- Biztosan megtalálja megadott pontossággal a megoldást.
- Az algoritmus lépésszáma pontosan számolható.

Hátrányai:

- Viszonylag lassú.
- Előre ismerni kell az $[a; b]$ intervallumot.

Példafeladat

Keressük meg az $f(x) = x^3 - x - 2$ függvény gyökét az $[1; 2]$ intervallumon 2 tizedesjegy pontossággal!

Példafeladat

Az $f(x) = x^3 - x - 2$ függvény gyöke az $[1; 2]$ intervallumon 2 tizedesjegy pontossággal:

lépés	a	b	c	|b-a|
0	1	2	1,5	1
1	1,5	2	1,75	0,5
2	1,5	1,75	1,625	0,25
3	1,5	1,625	1,5625	0,125
4	1,5	1.5625	1,53125	0,0625
5	1,5	1.53125	1,515625	0,03125
6	1,515625	1,53125	1,5234375	0,015625

Követelmények

- Az f(x) függvény folytonos.
- A választott intervallum két végpontján az f(x) függvény helyettesítési értéke ellentétes előjelű.
- Tehát:

$$f(a) * f(b) < 0$$

Működés

- Tetszőleges ε pontossággal megkereshetjük a gyököt.
- Választunk egy $[a; b]$ intervallumot úgy, hogy a kezdeti feltételek teljesüljenek.
- Minden lépésben felezzük az $[a; b]$ intervallum hosszúságát.

Leállási feltétel vonatkozhat:

- intervallum hosszára
- felezőpontban felvett függvényértékre
- iterációszámra

Működés

- Az intervallum felezőpontja:

$$c_n = \frac{(b_n + a_n)}{2}$$

- Ha $f(a_n) * f(c_n) < 0$, akkor $b_{n+1} = c_n$
- Ha $f(b_n) * f(c_n) < 0$, akkor $a_{n+1} = c_n$
- Ha $f(a_n) * f(c_n) = 0$, gyököt találtunk
- A fenti lépéseket addig ismételjük, amíg $|f(c_n)| > \varepsilon$

Hiba

A közelítő megoldás hibája n lépés esetén:

$$\varepsilon_n <= \frac{|b_0 - a_0|}{2^n}$$

Számítógépes használat esetén számba kell venni még a lebegőpontos számábrázolásból adódó hibát is.

Az algoritmus Scilab-ban

```
function Intfel(f,a,b,t)
if f(a)*f(b)>0
   printf('Nem ellentétes előjelű az intervallum!')
   abort
end
if f(a)==0
   printf('Az intervallum első végpontja megoldás!')
   abort
end
if f(b)==0
   printf('Az intervallum második végpontja megoldás!')
   abort
end
l=1;
while abs(b-a)>t & l < 100
   c=(a+b)/2;
   if f(a)*f(c) < 0
      b=c;
   elseif f(b)*f(c) < 0
      a=c;
   else
      printf('%d. lépésben a pontos gyök: %1.14f\n',l,c)
      abort
   end
   printf('%d. lépésben a közelítő gyök: %1.14f\n',l,c)
   l=l+1;
end
endfunction
```

Az algoritmus Scilab-ban

```
function Intfel(f,a,b,t)
  [...]
endfunction
```

Létrehoztuk az *Intfel* nevű függvényt
Hívása 4 paraméterrel történik:

- f: A vizsgálni kívánt függvény
- a: Az intervallum eleje
- b: Az intervallum vége
- t: A kívánt pontosság. Például 2 tizedesjegy pontossághoz 10^{-2}, vagy 0.01

Az algoritmus Scilab-ban

```
if f(a)*f(b)>0
  printf('Nem ellentetes elojelu az intervallum!')
  abort
end
```

Ellenőrizzük a kezdeti feltétel meglétét. Ha ez nem teljesül, az algoritmus nem indul el.

Az algoritmus Scilab-ban

```
if f(a)==0
  printf('Az intervallum elso vegpontja megoldas!')
abort
end
if f(b)==0
  printf('Az intervallum masodik vegpontja megoldas!')
abort
end
```

Ha az intervallumunk kezdő, vagy végpontja megoldás, akkor az algoritmus nem indul el.

Az algoritmus Scilab-ban

```
l=1;
c=(a+b)/2;
while abs(b-a)>=t & l < 100
c=(a+b)/2;
[...]
l=l+1;
end
```

Az algoritmus Scilab-ban

- Megfelezzük az intervallumot
- Bevezettünk egy *i* változót, ami a lépéseket számolja
- Az algoritmus leáll, ha:
  - Az iterációk száma eléri a 100-at
  - Elértük a kívánt pontosságot, amit a $|b_n - a_n| < t$ leállási feltétel biztosít.

Az algoritmus Scilab-ban

```
if f(a)*f(c) < 0
b=c;
elseif f(b)*f(c) < 0
a=c;
else
printf('%d. lepesben a pontos megoldas: %1.14f\n',l,c)
abort
end
```

A feltételeknek megfelelően beállítjuk az új intervallum kezdő és végpontját. Ha pontos gyököt találtunk, megállítjuk a program futását.

Feladat

Keresse meg az $f(x) = x^3 - x - 2$ függvény gyökét az $[1; 2]$ intervallumon 10^{-5} pontossággal Scilab használatával az intervallum felezés módszerével!

Feladat

Az $f(x) = x^3 - x - 2$ függvény gyöke az $[1; 2]$ intervallumon 10^{-5} pontossággal:

```
function y=fgv(x)
y=x^3-x-2
endfunction

Intfel(fgv,1,2,10^-5)
1. lépésben a közelítõ gyök:  1.50000000000000
2. lépésben a közelítõ gyök:  1.75000000000000
3. lépésben a közelítõ gyök:  1.62500000000000
4. lépésben a közelítõ gyök:  1.56250000000000
5. lépésben a közelítõ gyök:  1.53125000000000
6. lépésben a közelítõ gyök:  1.51562500000000
7. lépésben a közelítõ gyök:  1.52343750000000
8. lépésben a közelítõ gyök:  1.51953125000000
9. lépésben a közelítõ gyök:  1.52148437500000
10. lépésben a közelítõ gyök:  1.52050781250000
11. lépésben a közelítõ gyök:  1.52099609375000
12. lépésben a közelítõ gyök:  1.52124023437500
13. lépésben a közelítõ gyök:  1.52136230468750
14. lépésben a közelítõ gyök:  1.52142333984375
15. lépésben a közelítõ gyök:  1.52139282226563
16. lépésben a közelítõ gyök:  1.52137756347656
17. lépésben a közelítõ gyök:  1.52138519287109
```

Feladat

A Scilab beépített *fsolve* függvénye a következő eredményt adja:

```
-->function y=fgv(x)
-->y=x^3-x-2
-->endfunction

-->fsolve(1,fgv)
 ans   =

    1.5213797
```

Az algoritmus lépésszáma

Az intervallum felezés módszere lépésszáma pontosan számolható. Mivel

$$\varepsilon \leq \frac{b-a}{2^n},$$

átrendezve az egyenlőtlenséget:

$$n \geq \frac{\ln(\frac{b-a}{\varepsilon})}{\ln(2)}.$$

Ezután

$$n = \left[\frac{\ln(\frac{b-a}{\varepsilon})}{\ln(2)}\right] + 1$$

iterációs lépés után a közelítő megoldásunkra tejlesül az ε pontosságú közelítés.

Az algoritmus Scilabban

```
n=ceil(log((b-a)/t)/log(2));
for i=1:n
c=(a+b)/2;
[...]
end
```

A ceil függvény felfelé kerekíti a kapott értéket.

Az algoritmus Scilab-ban

```
printf('%d. lepesben a pontos megoldas: %1.14f\n',i,c)
```

A "for" ciklusszervezési mód adott számszor hajtja végre az iterációs lépést, emiatt nem kell végtelen ciklustól tartanunk. Az "l" változó feleslegessé vált, kitöröljük az algoritmusból. A kiiratásnál volt szerepe ennek a változónak a lépésszám kiírásánál, ezt a szerepet most átveszi az "i" ciklusváltozó.

Feladat

Futtassuk le a már megoldott feladatra az újonnan elkészült programot.
Az $f(x) = x^3 - x - 2$ függvény gyöke az $[1; 2]$ intervallumon 10^{-5} pontossággal:

```
-->Intfel(fgv,1,2,10^-5)
1. lépésben a közelítõ gyök:  1.500000000000000
2. lépésben a közelítõ gyök:  1.750000000000000
3. lépésben a közelítõ gyök:  1.625000000000000
4. lépésben a közelítõ gyök:  1.562500000000000
5. lépésben a közelítõ gyök:  1.531250000000000
6. lépésben a közelítõ gyök:  1.515625000000000
7. lépésben a közelítõ gyök:  1.523437500000000
8. lépésben a közelítõ gyök:  1.519531250000000
9. lépésben a közelítõ gyök:  1.521484375000000
10. lépésben a közelítõ gyök:  1.520507812500000
11. lépésben a közelítõ gyök:  1.520996093750000
12. lépésben a közelítõ gyök:  1.521240234375000
13. lépésben a közelítõ gyök:  1.521362304687500
14. lépésben a közelítõ gyök:  1.521423339843750
15. lépésben a közelítõ gyök:  1.521392822265625
16. lépésben a közelítõ gyök:  1.521377563476563
17. lépésben a közelítõ gyök:  1.521385192871094
```

Húrmódszer

Bár az intervallum felezés módszere biztosan ad eredményt, ha a kezdeti feltételek teljesülnek, de hatékonysága nem a legjobb. A húrmódszer általában gyorsabban konvergál.

Működés

Adott f(x) függvény, ami folytonos $[a; b]$-ben, illetve egyszer metszi az x tengelyt. Az intervallum felezése helyett inkább az intervallum végein felvett f(a) és f(b) pontok közé húzott szakasz és az x tengely metszéspontját használjuk a közelítéshez. Ezzel általában gyorsabb konvergenciát érünk el.

Követelmények

A húrmódszer használatának alapfeltétele megegyezik a korábban már ismertetett intervallum felezésével:

- Az f(x) függvény folytonos.
- A választott intervallum két végpontján az f(x) függvény helyettesítési értéke ellentétes előjelű.
- Tehát:

$$f(a) * f(b) < 0$$

Példafeladat

Keressük meg az $f(x) = x^3 - x - 2$ függvény gyökét az $[1; 2]$ intervallumon 2 tizedesjegy pontossággal!

Példafeladat

Az $f(x) = x^3 - x - 2$ függvény gyöke az $[1; 2]$ intervallumon 2 tizedesjegy pontossággal:

lépés	a	b	c	f(c)
1	1	2	1,333333333	-0,962962963
2	1,333333333	2	1,462686567	-0,333338875
3	1,462686567	2	1,504019004	-0,101817977
4	1,504019004	2	1,516330565	-0,029894804
5	1,516330565	2	1,519918550	-0,008675066

Működés

- Választunk egy $[a; b]$ intervallumot úgy, hogy a kezdeti feltételek teljesüljenek.
- Minden lépésben kiszámoljuk az új húr és az x-tengely metszéspontját.
- Az új intervallumon végrehajtjuk az iterációt.

Működés

- Az egyenes egyenlete:
  $(y - f(a))(b - a) = (x - a)(f(b) - f(a))$.
- Innen $y - f(a) = \frac{(f(b)-f(a))*(x-a)}{b-a}$.
- Meghatározzuk, hogy hol metszi az x tengelyt az előbb felírt egyenes.
- A megoldás:

$$x = a - \frac{f(a) * (b - a)}{f(b) - f(a)}$$

.

- Minden lépésben a metszéspontot használjuk tovább.

Működés

- A húr és az x tengely metszéspontja:

$$c_{n+1} = a_n - \frac{f(a_n) * (b_n - a_n)}{f(b_n) - f(a_n)}$$

.

- Ha $f(a_n) * f(c_n) < 0$, akkor $b_{n+1} = c_n$.
- Ha $f(b_n) * f(c_n) < 0$, akkor $a_{n+1} = c_n$.
- Különben pontos gyököt találtunk.
- A fenti lépéseket addig ismételjük, amíg $|f(c)| > t$.

Az algoritmus Scilabban

```
function Hur(f,a,b,t)
    if f(a)*f(b)>0
        printf('Nem ellentétes előjelű az intervallum!')
        abort
    end
    if f(a)==0
        printf('Az intervallum első végpontja megoldás!')
        abort
    end
    if f(b)==0
        printf('Az intervallum második végpontja megoldás!')
        abort
    end
    l=1;
    c=a-f(a)*(a-b)/(f(a)-f(b));
    while abs(f(c))>=t & l < 100
        c=a-f(a)*(a-b)/(f(a)-f(b));
        if f(a)*f(c) < 0
            b=c;
        elseif f(b)*f(c) < 0
            a=c;
        else
            printf('%d. lépésben a pontos megoldás: %1.14f\n',l,c)
            abort
        end
        printf('%d. lépésben a közelítő megoldás: %1.14f\n',l,c)
        l=l+1;
    end
endfunction
```

Az algoritmus Scilabban

```
function Hur(f,a,b,t)
  [...]
endfunction
```

Létrehoztuk a *Hur* nevű függvényt
Hívása 4 paraméterrel történik:

- f: A vizsgálni kívánt függvény
- a: Az intervallum eleje
- b: Az intervallum vége
- t: A kívánt pontosság. Például 2 tizedesjegy pontossághoz 10^{-2}, vagy 0.01

Az algoritmus Scilabban

A húrmódszer kezdeti feltételei ugyanazok, mint az intervallum felezés módszerénél. Programozási szempontból két lényeges eltérés van a két algoritmus között:

- Az $[a; b]$ intervallum felosztása.
- A leállási feltétel. Az intervallum felezés módszerénél használt az intervallum hosszára vonatkozó feltétel itt nem használható (konvex függvények esetében előfordulhat, hogy a megoldást már megközelítettük, de az intervallum hossza nem teljesíti a feltételt).

Az algoritmus Scilabban

```
l=1;
 c=a-f(a)*(a-b)/(f(a)-f(b));
while abs(f(c))>=t & l < 100
c=a-f(a)*(a-b)/(f(a)-f(b));
[...]
l=l+1;
end
```

Az algoritmus Scilabban

- Kiszámoljuk az x tengely és a húr metszéspontját
- Bevezettünk egy *I* változót, ami a lépéseket számolja
- Az algoritmus leáll, ha:
  - Az iterációk száma eléri a 100-at
  - Elértük a kívánt pontosságot, ami az $|f(c_n)| < t$

Az algoritmus Scilabban

```
if f(a)*f(c) < 0
b=c;
elseif f(b)*f(c) < 0
a=c;
else
printf('%d. lepesben a pontos megoldas: %1.14f\n',l,c)
abort
end
```

A feltételeknek megfelelően beállítjuk az új intervallum kezdő és végpontját. Ha pontos gyököt találtunk, megállítjuk a program futását.

Feladat

Keresse meg az $f(x) = x^3 - x - 2$ függvény gyökét az $[1, 2]$ intervallumon 10^{-5} pontossággal Scilab használatával húrmódszer segítségével!

Feladat

```
-->Hur(fgv,1,2,10^-5)
1. lépésben a közelítő gyök: 1.3333333333333
2. lépésben a közelítő gyök: 1.46268656716418
3. lépésben a közelítő gyök: 1.50401900394995
4. lépésben a közelítő gyök: 1.51633056476026
5. lépésben a közelítő gyök: 1.51991855002336
6. lépésben a közelítő gyök: 1.52095748137193
7. lépésben a közelítő gyök: 1.52125774912629
8. lépésben a közelítő gyök: 1.52134448423152
9. lépésben a közelítő gyök: 1.52136953453760
10. lépésben a közelítő gyök: 1.52137676908718
11. lépésben a közelítő gyök: 1.52137885840389
```

Feladat

A Scilab beépített *fsolve* függvénye a következő eredményt adja:

```
-->function y=fgv(x)
-->y=x^3-x-2
-->endfunction

-->fsolve(1,fgv)
 ans   =

    1.5213797
```

Scilab programozás és a numerikus algoritmusok Scilabban
8. előadás
Érintő és szelőmódszer

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

1. Newton-módszer
   - Bevezetés
   - Mintafeladat
   - Scilab megvalósítás
   - Az m-edik gyök közelítése
2. Szelőmódszer
   - Bevezetés
   - Mintafeladat
   - Scilab megvalósítás

Bevezetés

Tekintsük a $f(x) = 0$ egyenlet megoldásakor a következő közelítést: Legyen p_0 kezdőpont tetszőleges eleme az f függvény értelmezési tartományának. Vegyük az f függvény p_0 körüli elsőrendű Taylor-polinomját.
Keressük meg ennek a polinomnak a gyökét.

Geometriai jelentés

Geometriailag ez azt jelenti, hogy a függvény p_0 pontjában húzott érintő egyenes és az x tengely metszéspontját vesszük. A metszéspontot az alábbi egyenlet megoldása adja:

$$0 = f(p_0) + f'(p_0)(x - p_0).$$

A megoldás:

$$x = p_0 - \frac{f(p_0)}{f'(p_0)}$$

feltéve, hogy $f'(p_0) \neq 0$.

Definíció

A kapott megoldást jelöljük p_1-gyel. Ezután ismételjük meg az eljárást. Így kapjuk az alábbi rekurzív sorozatot:

$$p_{k+1} = p_k - \frac{f(p_k)}{f'(p_k)}$$

A kapott iterációt **Newton-Ralphson**, röviden **Newton-módszer**nek, illetve **érintőmódszer**nek nevezik.

Konvergencia

Az érintő módszer fixpont iteráció a

$$g(x) = x - \frac{f(x)}{f'(x)}$$

iterációs függvénnyel.

Tétel (Hartung 2.23. tétel)
Legyen $f \in \mathrm{C}^2(a; b)$ és legyen $p \in (a; b)$ olyan pont, amelyre $f(p) = 0$ és $f'(p) \neq 0$. Ekkor az érintőmódszer lokálisan konvergál p-hez.

Mintafeladat

Számolja ki a következő függvény megoldásait érintőmódszerrel. Kezdőpontként először a $p_0 = 0$, másodszor a $p_0 = -1$ értékkel számoljon. A megoldást $|f(p_k)| < 0,01$ megállási kritérium szerint adja meg!

$$f(x) = x^2 + x - 12$$

Megoldás:
Az érintőmódszer használatához szükségünk van a függvény deriváltfüggvényére.

$$f'(x) = 2x + 1$$

Mintafeladat megoldása

A $p_0 = 0$ kezdőpontot használva az alábbi értékeket kapjuk:

lépés	p_k	$f(p_k)$	$f'(p_k)$	p_{k+1}
0	0	-12	1	12
1	12	144	25	6,24
2	6,24	33,1776	13,48	3,77875
3	3,77875	6,0577	8,5575	3,070868
4	3,070868	0,5011	7,1417	3,0007
5	3,0007	**0,0049**	7,0014	

Mintafeladat megoldása

A $p_0 = -1$ kezdőpontot használva az alábbi értékeket kapjuk:

lépés	p_k	$f(p_k)$	$f'(p_k)$	p_{k+1}
0	-1	-12	-1	-13
1	-13	144	-25	-7,24
2	-7,24	33,1776	-13,48	-4,77875
3	-4,77875	6,0577	-8,5575	-4,070868
4	-4,070868	0,5011	-7,1417	-4,0007
5	-4,0007	**0,0049**	-7,0014	

Mintafeladat megoldása

Az eljárás két különböző kezdőpontból kiindulva különböző eredményt adott. A másodfokú függvény két megoldása $x_0 = 3$ és $x_0 = -4$.
A kapott két közelítő érték a két megoldást közelítette meg kellő pontossággal.

Pszeudokód

Input: $f(x)$, $f'(x)$, a, t
while $f(a) \geq t$ **do**
$a \longleftarrow a - f(a)/f'(a)$
end while
Output: a

Scilab megvalósítás

```
function erinto(f,df,a,t)
if f(a)==0 & abs(f(a))<t
   printf('A kezdőpont megoldás!')
   abort
end
l=1;
while abs(f(a))>=t & l < 100
   if df(a) == 0
     printf('Nem található megoldás, az érintő
                  párhuzamos az x tengellyel!')
     abort
   end
   a=a-f(a)/df(a);
   printf('%d. lépés után a közelítő megoldás:
                                    %1.14f\n',l,a)
   l=l+1;
end
endfunction
```

Paraméterek

Az eljárás négy paramétert vár indításkor:

- f - a függvény, aminek a megoldását keressük
- df - az 'f' függvény deriváltja
- a - kezdőpont
- t - elvárt pontosság

Az algoritmus első része a megadott paraméterek vizsgálata. Két esetet vizsgál külön a program:

- A kezdőpont pontos, illetve elvárnál pontosabb megoldás
- Az érintő párhuzamos az x tengellyel.

Az algoritmus második része az iterácó. A leállási feltétel két részből áll:

- $|f(a)| < t$ - teljesül a leállási feltétel
- $l > 100$ - elértük a maximális lépésszámot, végtelen ciklus elkerülése miatt leáll az algoritmus

Mintafeladat megoldása Scilabban

Hozzuk létre a függvényt és a deriváltfüggvényt Scilabban.

```
function y=fgv(x)
    y=x^2+x-12;
endfunction

function y=dfgv(x)
    y=2*x+1;
endfunction
```

Mintafeladat megoldása

Futtatás $p_0 = 0$ kezdőértékkel:

Scilab 5.4.1 Console

```
-->erinto(fgv,dfgv,0,0.01)
1. lépés után a közelítő megoldás:  12.000000000000000
2. lépés után a közelítő megoldás:  6.240000000000000
3. lépés után a közelítő megoldás:  3.778753709198810
4. lépés után a közelítő megoldás:  3.070868456191710
5. lépés után a közelítő megoldás:  3.000703237621970
```

Mintafeladat megoldása

Futtatás $p_0 = -1$ kezdőértékkel:

Scilab 5.4.1 Console

```
-->erinto(fgv,dfgv,-1,0.01)
1. lépés után a közelítő megoldás: -13.000000000000000
2. lépés után a közelítő megoldás: -7.240000000000000
3. lépés után a közelítő megoldás: -4.778753709198810
4. lépés után a közelítő megoldás: -4.070868456191710
5. lépés után a közelítő megoldás: -4.000703237621970
```

$\sqrt{2}$ közelítése

Az érintőmódszer nagyon hatékonyan, gyorsan közelíti meg az $x^2 - 2 = 0$ egyenlet megoldását (általánosan is igaz a $x^2 - a = 0, a > 0$ egyenletre). Az iterációs képlet alapján a következő közelítést írhatjuk fel:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^2 - 2}{2x_n} = \frac{x_n^2 + 1}{2x_n} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$$

Bizonyítható, hogy tetszőleges $x_0 \neq 0$pontból indítva valamelyik gyökhöz tart ($x_0 < 0$ esetén $-\sqrt{2}$-höz, $x_0 > 0$ esetén $\sqrt{2}$-höz).

$\sqrt{2}$ közelítése

Számoljuk ki az iteráció első három lépését a $p_0 = 1,5$ kezdőpontból indítva.

x_i	Eltérés
$x_1 = \frac{1}{2}\left(\frac{3}{2} + \frac{3}{4}\right) = \frac{17}{12} \approx 1,41667$	$\leq 0,0025$
$x_2 = \frac{1}{2}\left(\frac{17}{12} + \frac{24}{17}\right) = \frac{577}{408} \approx 1,4142157$	$\leq 0,0000022$
$x_3 = \frac{1}{2}\left(\frac{577}{408} + \frac{816}{577}\right) = \frac{665857}{470832} \approx 1,414213562$	$\leq 1,65 * 10^{-12}$

Általános m-edik gyök keresése

Keressük az alábbi függvény gyökeit:

$$f(x) = x^m - a$$

Az iterációs képlet:

$$x_{n+1} = x_n - \frac{x_n^m - a}{m x_n^{m-1}} = \frac{1}{m}\left((m-1)x_n + \frac{a}{x_n^{m-1}}\right)$$

$\sqrt[3]{5}$ keresése Scilabban

A függvény, aminek a megoldását keressük:

$$f(x) = x^3 - 5$$

A függvény és a deriváltja Scilabban:

```
function y=fgv(x)
    y=x^3-5;
endfunction

function y=dfgv(x)
    y=3*x^2;
endfunction
```

$\sqrt[3]{5}$ keresése Scilabban

A $p_0 = 1$ kezdőpontból indítva az iterációt és $|f(p_k)| < 10^{-10}$ megállási kritériumot használva a következő futási eredményt kapjuk:

Scilab 5.4.1 Console

```
-->erinto(fgv,dfgv,2,10^-10)
1. lépés után a közelítő megoldás:  1.75000000000000
2. lépés után a közelítő megoldás:  1.71088435374150
3. lépés után a közelítő megoldás:  1.70997642891697
4. lépés után a közelítő megoldás:  1.70997594667683

-->
```

Érintő módszer átalakítása

A kiszámolt iterációs képlet segítségével átalakíthatjuk az érintőmódszert megvalósító eljárást speciálisan $\sqrt[m]{a}$ közelítő számítását végző algoritmussá.
Az eljárás négy paramétert kér indításkor:

- m - az m-edik gyök
- a - az érték, aminek az m-edik gyökét keressük
- p - kezdőpont
- ε - elvárt pontosság

```
function mgyok(m,a,p,t)
if p^m-a==0 & abs(p^m-a)<t
printf('A kezdőpont megoldás!')
abort
end
l=1;
while abs(p^m-a)>=t & l < 100
if m*p^(m-1) == 0
printf('Nem található megoldás, az érintő
                    párhuzamos az x tengellyel!')
abort
end
p=1/m*((m-1)*p+a/p^(m-1));
printf('%d. lépés után a közelítő megoldás:
                                %1.14f\n',l,p)
l=l+1;
end
endfunction
```

Definíció

Lefuttatva az $m = 3$, $a = 5$, $p = 1$ és $t = 10^{-10}$ paraméterekkel az eljárást ugyanazt az eredményt kaptuk, mint az érintő módszert futtatva.

Scilab 5.4.1 Console

```
-->mgyok(3,5,1,10^-10)
1. lépés után a közelítő megoldás: 2.33333333333333
2. lépés után a közelítő megoldás: 1.86167800453515
3. lépés után a közelítő megoldás: 1.72200188005861
4. lépés után a közelítő megoldás: 1.71005973660029
5. lépés után a közelítő megoldás: 1.70997595078219
6. lépés után a közelítő megoldás: 1.70997594667670

-->
```

Bevezetés

Az eddigi gyökkereső iterációs eljárások mindegyikének volt valamilyen speciális feltétele, ami miatt megkötéseket kellett tenni a használatuk során.
Az intervallum felező és a húrmódszer kezdeti intervallumára kellett feltételt tenni, az érintő módszer pedig a függvény deriváltját követelte meg.
A szelőmódszer mindegyiknél általánosabb eljárás, ahol sem a kezdeti intervallumra nem kell feltételt tennünk, sem a deriváltfüggvényre nincs szükség.

Definíció

Legyen p_0 és p_1 két tetszőleges, különböző pontja az f függvény értelmezési tartományának. Tekintsük a két pontban felvett függvényértéket összekötő szelő egyenest. Ennek az egyenesnek az egyenlete:

$$y = f(p_1) + \frac{f(p_1) - f(p_0)}{p_1 - p_0}(x - p_1)$$

Ez a szelő egyenes az x tengyelyt a következő pontban metszi:

$$x = p_1 - \frac{p_1 - p_0}{f(p_1) - f(p_0)} f(p_1)$$

Legyen ez a pont p_2.

Definíció

Az eljárás következő lépésében tekintsük a p_1, p_2 pontokhoz tartozó szelő egyenest. Ennek az egyenesnek a metszéspontját az x tengellyel legyen p_3. Általánosan az eljárás a következő iterációs egyenlettel adható meg:

$$p_{k+1} = p_k - \frac{p_k - p_{k-1}}{f(p_k) - f(p_{k-1})} f(p_k)$$

Konvergencia tételek

1. tétel (Hartung 2.26. tétel)
Legyen $f \in \mathrm{C}^2(a;b)$ és legyen $p \in (a;b)$ olyan, hogy $f(p) = 0$ és $f'(p) \neq 0$. Legyen p_k a szelőmódszerrel generált sorozat. Ekkor minden k-ra léteznek olyan $\xi_k \in \langle p_k; p_{k-1}; p \rangle$ és $\eta_k \in \langle p_k; p_{k-1} \rangle$ számok, hogy

$$p_{k+1} - p = \frac{1}{2} \frac{f''(\xi_k)}{f'(\eta_k)} (p_k - p)(p_{k-1} - p).$$

2. tétel (Hartung 2.27. tétel)
Legyen $f \in \mathrm{C}^2(a;b)$ és legyen $p \in (a;b)$ olyan, hogy $f(p) = 0$ és $f'(p) \neq 0$. Ekkor a szelőmódszer lokálisan konvergál p-hez.

Példa

Számolja ki a következő függvény megoldásait szelőmódszerrel. A két kezdőpont legyen a $p_0 = -5$, $p_1 = 5$ pontok. A megoldást $|f(p_k)| < 0,01$ megállási kritérium szerint adja meg!

$$f(x) = 3x - 2^x$$

Mintafeladat megoldása

A fenti paramétereket használva az alábbi értékeket kapjuk (p_0 és p_1 kezdőérték nem szerepel a táblázatban):

k	p_k	$f(p_k)$
2	-81,3492	-244,0476
3	11,465324	-2793,12965
4	-90,235	-270,7056
5	-101,14968	-303,449
6	$4 * 10^{-14}$	-0.99999
7	0,334435	-0,257578
8	0,450465	-0,015
9	0,45768	**-0,00028**

Pszeudokód

Input: $f(x)$, p_0, p_1, t
$p \longleftarrow b - f(b) * (b - a)/(f(b) - f(a))$
while $f(a) \geq t$ **do**
  $a \longleftarrow b$
  $b \longleftarrow p$
  $p \longleftarrow b - f(b) * (b - a)/(f(b) - f(a))$
end while
Output: p

Scilab megvalósítás

```
function szelo(f,a,b,t)
if f(a)==0
printf('Az első kezdőpont megoldás!')
abort
end
if f(b)==0
printf('A második kezdőpont megoldás!')
abort
end
l=1;
if f(a)-f(b) == 0
printf('Nem található megoldás, a szelő
                párhuzamos az $x$ tengellyel!')
abort
end
```

Scilab megvalósítás

```
p=b-f(b)*(b-a)/(f(b)-f(a));
while abs(f(p))>=t & l < 100
if f(a)-f(b) == 0
printf('Nem található megoldás, a szelő
              párhuzamos az $x$ tengellyel!')
abort
end
printf('%d. lépésben a közelítő megoldás:
                    %1.14f\n',l,p)
a=b;
b=p;
p=b-f(b)*(b-a)/(f(b)-f(a));
l=l+1;
end
endfunction
```

Paraméterek

Az eljárás négy paramétert vár indításkor:

- f - a függvény, aminek a megoldását keressük
- a - p_0 kezdőpont
- b - p_1 kezdőpont
- t - elvárt pontosság

Az algoritmus első része a megadott paraméterek vizsgálata. Három esetet vizsgál külön a program:

- A p_0 kezdőpont pontos, illetve elvártnál pontosabb megoldás
- A p_1 kezdőpont pontos, illetve elvártnál pontosabb megoldás
- Az szelő párhuzamos az x tengellyel.

Az algoritmus második része az iterácó. A leállási feltétel két részből áll:

- $|f(a)| < t$ - a p_k pontban a függvényérték elérte az elvárt pontosságot
- $l > 100$ - elértük a maximális lépésszámot, végtelen ciklus elkerülése miatt leáll az algoritmus

Mintafeladat megoldása Scilabban

Hozzuk létre a függvényt Scilabban.

```
function y=fgv(x)
    y=3*x-2^x;
endfunction
```

Mintafeladat megoldása

Futtatás $p_0 = -5$ és $p_1 = 5$ kezdőértékkel:

Scilab 5.4.1 Console
Scilab Preferences

```
-->szelo(fgv,-5,5,10^-2)
1. lépésben a közelítő megoldás: -81.34920634920636
2. lépésben a közelítő megoldás: 11.46532438478747
3. lépésben a közelítő megoldás: -90.23521508773747
4. lépésben a közelítő megoldás: -101.14968072283500
5. lépésben a közelítő megoldás: 0.00000000000004
6. lépésben a közelítő megoldás: 0.33443544736532
7. lépésben a közelítő megoldás: 0.45046516772272
```

Scilab programozás és a numerikus algoritmusok Scilabban
9. előadás
Gauss és Gauss-Jordan-elimináció, mátrix invertálás

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

Tartalomjegyzék

1. Bevezetés
2. Trianguláris egyenletrendszerek
   - Mintafeladat
   - Definíció
   - Scilab megvalósítás
3. Gauss-elimináció
   - Definíció
   - Mintafeladat
   - Scilab megvalósítás
4. Gauss-Jordan-elimináció
   - Definíció
   - Mintafeladat
   - Scilab megvalósítás
5. Mátrix invertálás
   - Definíció
   - Mintafeladat
   - Scilab megvalósítás

Bevezetés

Lineáris egyenletrendszerekkel számos területen találkozhatunk:

- mechanikában,
- geodéziában,
- villamosságtanban,
- ökológiai és
- gazdasági területeken, stb.

Bevezetés

Egy lineáris egyenletrendszer általános alakja az alábbi:

$$A\bar{x} = \bar{b},\ A = (a_{ij}) \in \mathbb{R}^{m\times n},\ \bar{x} \in \mathbb{R}^n,\ \bar{b} \in \mathbb{R}^m$$

amelyet kifejtve, az alábbi egyenletrendszer írható fel:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m \end{aligned}$$

Bevezetés

Feltételezés

A következőkben feltételezzük, hogy az egyenletrendszer A mátrixa négyzetes, azaz $m = n$. A későbbiekben csak ilyen tulajdonságú feladatokat vizsgálunk. Éppen ezért a vizsgált egyenletrendszer:

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

Trianguláris egyenletrendszerek

Példa

Oldjuk meg a következő egyenletrendszert:

$$\begin{array}{rcrcrcrcr} x_1 & + & x_2 & - & x_3 & + & x_4 & = & 8 \\ & - & 2x_2 & + & x_3 & - & 3x_4 & = & -13 \\ & & & & 3x_3 & - & 6x_4 & = & -21 \\ & & & & & & 4x_4 & = & 16 \end{array}$$

Trianguláris egyenletrendszerek

Megoldás:

- A negyedik egyenletet x_4-re megoldhatjuk: $x_4 = 4$.
- Ezt visszahelyettesítve a harmadik egyenletbe kapjuk $x_3 = (-21 + 6x_4)/3 = 1$, majd a
- második egyenletből $x_2 = -(-13 - x_3 + 3x_4)/2 = 1$.
- Végül az első egyenletből $x_1 = 8 - x_2 + x_3 - x_4 = 4$.

Trianguláris egyenletrendszerek

Az előző példa általánosítva, egy n dimenziós felülről trianguláris egyenletrendszer, $A\bar{x} = \bar{b}$, azaz

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ & & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ & & & & \ddots & & \vdots & & \vdots \\ & & & & & & a_{nn}x_n & = & b_n \end{array}$$

megoldásának módszerét, az ún. **visszahelyettesítés módszerét** a következő algoritmussal adhatjuk meg.

Pszeudokód

Input: $a_{ij}, (i = 1, \ldots, n, \;\; j = i, \ldots, n), \;\; b_i, (i = 1, \ldots, n)$
for $i = n$ to 1 **do**
$x_i \longleftarrow \left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j \right) / a_{ii}$
end for
Output: $x_1, x_2, \ldots, x_n$

Scilab kód

```
function triang(A,b)
sA=size(A);
sb=size(b);
if sA(1) ~= sA(2) | sA(1) ~= sb(1) | sb(2) ~= 1
    disp('Rossz dimenziók!')
    abort
end
n=sA(1);
for i=1:n
    if A(i,i)==0 & b(i) ==0
      disp('Nincs egyértelmű megoldás')
abort
elseif A(i,i)==0
  disp('Főátlóban 0 van, nincs megoldás!')
        abort
    end
end
```

Scilab kód

```
A=[A,b];
printf('0. lépés:\n');
disp(A)
for i=n:-1:1,
    x(i)=A(i,n+1);
    for j=i+1:n,
        x(i)=x(i)-A(i,j)*x(j);
    end
    x(i)=x(i)/A(i,i);
end
disp('A megoldás:')
disp(x)
endfunction
```

Gauss-elimináció

Tekintsük az alábbi lineáris egyenletrendszert:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ & & & & \ddots & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \dots & + & a_{nn}x_n & = & b_n \end{array}$$

Gauss-elimináció

A Gauss-elimináció a következő lépésekből áll:
Az egyenletrendszer együtthatóiból és az egyenletrendszer jobb oldalából készítsük el az ún. kibővített mátrixot.

$$\tilde{\mathbf{A}}^{(0)} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1,n} & a_{1,n+1} \\ a_{21} & a_{22} & \cdots & a_{2,n} & a_{2,n+1} \\ \vdots & \vdots & & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{n,n} & a_{n,n+1} \end{pmatrix},$$

ahol

$$a_{i,n+1} \equiv b_i; (i = 1; \cdots ; n).$$

Gauss-elimináció

Legyen $a_{11} \neq 0$. Ekkor kivonjuk az első egyenlet $l_{i1} := (a_{i1}/a_{11})$-szeresét az i-edik egyenletből, ahol $i = 2; 3; \cdots ; n$. Az A mátrix első sorának elemeit a_{ij}-vel, a kivonás által $i > 1$-re létrehozott új elemeket pedig $a_{ij}^{(1)}$-vel jelölve, a művelet az alábbi:

$$a_{ij}^{(1)} := a_{ij} - l_{i1} a_{1j}$$

Ekkor az első lépés után az egyenletrendszer az alábbi lesz:

$$\tilde{\mathbf{A}}^{(1)} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1,n} & a_{1,n+1} \\ 0 & a_{22}^{(1)} & \cdots & a_{2,n}^{(1)} & a_{2,n+1}^{(1)} \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{n,n}^{(1)} & a_{n,n+1}^{(1)} \end{pmatrix},$$

Gauss-elimináció

Általánosan a k-adik lépés után a következő mátrixot kapjuk:

$$\tilde{\mathbf{A}}^{(k)} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1,k} & a_{1,k+1} & \cdots & a_{1,n} & a_{1,n+1} \\ 0 & a_{22}^{(1)} & \cdots & a_{2,k}^{(1)} & a_{2,k+1}^{(1)} & \cdots & a_{2,n}^{(1)} & a_{2,n+1}^{(1)} \\ & & \ddots & & & & & \\ 0 & 0 & \cdots & a_{k,k}^{(k-1)} & a_{k,k+1}^{(k-1)} & \cdots & a_{k,n}^{(k-1)} & a_{k,n+1}^{(k-1)} \\ 0 & 0 & \cdots & 0 & a_{k+1,k+1}^{(k)} & \cdots & a_{k+1,n}^{(k)} & a_{k+1,n+1}^{(k)} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 & a_{n,k+1}^{(k)} & \cdots & a_{n,n}^{(k)} & a_{n,n+1}^{(k)} \end{pmatrix},$$

ahol

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - l_{ik} a_{kj}^{(k-1)}, \; l_{ik} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, \; i = k+1, \ldots, n, \; j = k+1, \ldots, n+1.$$

Példa

Oldja meg az alábbi egyenletrendszert.

$$\begin{aligned} x_1 + x_2 - x_3 + x_4 &= 8 \\ 2x_1 + x_2 - x_3 - x_4 &= 3 \\ x_1 - 2x_2 - 2x_3 + 3x_4 &= 12 \\ x_1 + x_2 - x_3 - 2x_4 &= 0 \end{aligned}$$

Példa

Oldja meg az alábbi egyenletrendszert.

$$\begin{aligned} x_1 + x_2 - x_3 + x_4 &= 8 \\ 2x_1 + x_2 - x_3 - x_4 &= 3 \\ x_1 - 2x_2 - 2x_3 + 3x_4 &= 12 \\ x_1 + x_2 - x_3 - 2x_4 &= 0 \end{aligned}$$

$$\left(\begin{array}{cccc|c} \underline{1} & 1 & -1 & 1 & 8 \\ 2 & 1 & -1 & -1 & 3 \\ 1 & -2 & -2 & 3 & 12 \\ 1 & 1 & -1 & -2 & 0 \end{array}\right)$$

Példa

Oldja meg az alábbi egyenletrendszert.

$$\begin{aligned} x_1 + x_2 - x_3 + x_4 &= 8 \\ 2x_1 + x_2 - x_3 - x_4 &= 3 \\ x_1 - 2x_2 - 2x_3 + 3x_4 &= 12 \\ x_1 + x_2 - x_3 - 2x_4 &= 0 \end{aligned}$$

$$\left(\begin{array}{cccc|c} \underline{1} & 1 & -1 & 1 & 8 \\ 2 & 1 & -1 & -1 & 3 \\ 1 & -2 & -2 & 3 & 12 \\ 1 & 1 & -1 & -2 & 0 \end{array}\right) \sim \left(\begin{array}{cccc|c} 1 & 1 & -1 & 1 & 8 \\ 0 & \underline{-1} & 1 & -3 & -13 \\ 0 & -3 & -1 & 2 & 4 \\ 0 & 0 & 0 & -3 & -8 \end{array}\right)$$

Példa

Oldja meg az alábbi egyenletrendszert.

$$\begin{aligned} x_1 + x_2 - x_3 + x_4 &= 8 \\ 2x_1 + x_2 - x_3 - x_4 &= 3 \\ x_1 - 2x_2 - 2x_3 + 3x_4 &= 12 \\ x_1 + x_2 - x_3 - 2x_4 &= 0 \end{aligned}$$

$$\left(\begin{array}{cccc|c} \underline{1} & 1 & -1 & 1 & 8 \\ 2 & 1 & -1 & -1 & 3 \\ 1 & -2 & -2 & 3 & 12 \\ 1 & 1 & -1 & -2 & 0 \end{array}\right) \sim \left(\begin{array}{cccc|c} 1 & 1 & -1 & 1 & 8 \\ 0 & \underline{-1} & 1 & -3 & -13 \\ 0 & -3 & -1 & 2 & 4 \\ 0 & 0 & 0 & -3 & -8 \end{array}\right) \sim$$

$$\left(\begin{array}{cccc|c} 1 & 1 & -1 & 1 & 8 \\ 0 & -1 & 1 & -3 & -13 \\ 0 & 0 & \underline{-4} & 11 & 43 \\ 0 & 0 & 0 & -3 & -8 \end{array}\right)$$

Példa

Oldja meg az alábbi egyenletrendszert.

$$\begin{aligned} x_1 &+ x_2 - x_3 + x_4 = 8 \\ 2x_1 &+ x_2 - x_3 - x_4 = 3 \\ x_1 &- 2x_2 - 2x_3 + 3x_4 = 12 \\ x_1 &+ x_2 - x_3 - 2x_4 = 0 \end{aligned}$$

$$\left(\begin{array}{cccc|c} \underline{1} & 1 & -1 & 1 & 8 \\ 2 & 1 & -1 & -1 & 3 \\ 1 & -2 & -2 & 3 & 12 \\ 1 & 1 & -1 & -2 & 0 \end{array}\right) \sim \left(\begin{array}{cccc|c} 1 & 1 & -1 & 1 & 8 \\ 0 & \underline{-1} & 1 & -3 & -13 \\ 0 & -3 & -1 & 2 & 4 \\ 0 & 0 & 0 & -3 & -8 \end{array}\right) \sim$$

$$\left(\begin{array}{cccc|c} 1 & 1 & -1 & 1 & 8 \\ 0 & -1 & 1 & -3 & -13 \\ 0 & 0 & \underline{-4} & 11 & 43 \\ 0 & 0 & 0 & -3 & -8 \end{array}\right) \sim \left(\begin{array}{cccc|c} 1 & 1 & -1 & 1 & 8 \\ 0 & -1 & 1 & -3 & -13 \\ 0 & 0 & -4 & 11 & 43 \\ 0 & 0 & 0 & -3 & -8 \end{array}\right)$$

Példa - megoldás

Mint korábban láthattuk, a kapott megoldásunk egy trianguláris egyenletrendszer, melynek a megoldása korábban bemutatásra került. Így a megoldásvektor a következő:

$$\bar{x} = \begin{pmatrix} 0,333 \\ 1,583 \\ -3,416 \\ 2,667 \end{pmatrix}$$

Pszeudokód

Input: a_{ij}, $(i = 1, \ldots, n, \quad j = 1, \ldots, n+1)$
for $k = 1$ to $n-1$ **do**
  for $i = k+1$ to n **do**
    $l_{ik} \longleftarrow a_{ik}/a_{kk}$
    for $j = k+1$ to $n+1$ **do**
      $a_{ij} \longleftarrow a_{ij} - l_{ik} * a_{kj}$
    end for
  end for
end for
for $i = n$ to 1 **do**
  $x_i \longleftarrow \left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j\right) / a_{ii}$
end for
Output: $x_1, x_2, \ldots, x_n$

Scilab kód

```
function gauss(A,b)

sA=size(A);
sb=size(b);
if sA(1) ~= sA(2) | sA(1) ~= sb(1) | sb(2) ~= 1
    disp('Rossz dimenziók!')
    abort
end
n=sA(1);
A=[A,b];
printf('0. lépés:\n');
disp(A)
```

Scilab kód

```
for k=1:n-1,
    for i=k+1:n,
        if A(k,k)==0,
            disp('Nem hajtható végre az eliminációs,
                                    főelem=0')
            abort
        end
        l=A(i,k)/A(k,k);
        for j=k:n+1,
            A(i,j)=A(i,j)-l*A(k,j);
        end
    end
```

Scilab kód

```
    printf('A(z) %d. eliminációs lépés:\n',k);
    disp(A)
end
for i=n:-1:1,
    x(i)=A(i,n+1);
    for j=i+1:n,
        x(i)=x(i)-A(i,j)*x(j);
    end
    x(i)=x(i)/A(i,i);
end
disp('A megoldás:')
disp(x)
endfunction
```

Példa

A mintafeladat megoldásához először hozzuk létre az együttható-mátrixot és az egyenletrendszer jobb oldalából képzett oszlopvektort. Scilabban.

```
A=[1,1,-1,1;2,1,-1,-1;1,-2,-2,3;1,1,-1,-2]
b=[8;3;12;0]
```

Ezután futtassuk le az eljárást.

```
gauss(A,b)
```

Példa

Lefuttatva a programot a következő futási eredményt kapjuk:

```
Scilab 5.4.1 Console

-->A=[1,1,-1,1;2,1,-1,-1;1,-2,-2,3;1,1,-1,-2];

-->b=[8;3;12;0];

-->gauss(A,b)
0. lépés:

    1.    1.  - 1.    1.    8.
    2.    1.  - 1.  - 1.    3.
    1.  - 2.  - 2.    3.    12.
    1.    1.  - 1.  - 2.    0.
A(z) 1. eliminációs lépés:

    1.    1.  - 1.    1.    8.
    0.  - 1.    1.  - 3.  - 13.
    0.  - 3.  - 1.    2.    4.
    0.    0.    0.  - 3.  - 8.
```

Példa

Lefuttatva a programot a következő végeredményt kapjuk:

Scilab 5.4.1 Console

```
A(z) 2. eliminációs lépés:

    1.     1.   - 1.     1.      8.
    0.   - 1.     1.   - 3.    - 13.
    0.     0.   - 4.     11.     43.
    0.     0.     0.   - 3.    - 8.
A(z) 3. eliminációs lépés:

    1.     1.   - 1.     1.      8.
    0.   - 1.     1.   - 3.    - 13.
    0.     0.   - 4.     11.     43.
    0.     0.     0.   - 3.    - 8.

 A megoldás:

    0.3333333
    1.5833333
  - 3.4166667
    2.6666667
```

Scilab kód változat

Az algoritmus iterációs része három egymásba ágyazott "for" ciklust. Ezek közül az egyik - a legbelső - elhagyható, amennyiben kihasználjuk, hogy a Scilab képes teljes sorokat, oszlopokat kezelni. Az alábbi kódrészlet

```
for j=k:n+1,
 A(i,j)=A(i,j)-l*A(k,j);
end
```

helyettesíthető az alábbi utasítással:

```
A(i,:)=A(i,:)-l*A(k,:);
```

Az elimináció módszere

A Gauss-Jordan-elimináció a Gauss-elimináció módosított változata. Ebben az eliminációban az együtthatómátrixot a Gauss elimináció lépéseivel nem trianguláris, hanem egységmátrixszá alakítjuk át, vagyis az (A, b) kibővített mátrixot $I, b^{(n-1)}$ alakúra hozzuk. A megoldás ekkor az utolsó oszlopban található $b^{(n-1)}$ vektor lesz.

A Gauss-elimináció végén a visszahelyettesítéses lépéssor leegyszerüsödik egy osztásra. Az elimináció végén a kibővített mátrixban az együtthatómátrix helyén egy diagonális mátrixot kaptunk. A visszahelyettesítés helyett elegendő az utolsó oszlopot soronként elosztani az adott sor főátlóbeli elemével.

Példa

Tekintsük a Gauss-eliminációnál vizsgált feladatot.

$$\begin{aligned} x_1 + x_2 - x_3 + x_4 &= 8 \\ 2x_1 + x_2 - x_3 - x_4 &= 3 \\ x_1 - 2x_2 - 2x_3 + 3x_4 &= 12 \\ x_1 + x_2 - x_3 - 2x_4 &= 0 \end{aligned}$$

Példa

A Gauss-Jordan-elimináció során kapott mátrixok:

$$\left(\begin{array}{cccc|c} \underline{1} & 1 & -1 & 1 & 8 \\ 2 & 1 & -1 & -1 & 3 \\ 1 & -2 & -2 & 3 & 12 \\ 1 & 1 & -1 & -2 & 0 \end{array}\right) \sim \left(\begin{array}{cccc|c} 1 & 1 & -1 & 1 & 8 \\ 0 & \underline{-1} & 1 & -3 & -13 \\ 0 & -3 & -1 & 2 & 4 \\ 0 & 0 & 0 & -3 & -8 \end{array}\right) \sim$$

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & -2 & -5 \\ 0 & -1 & 1 & -3 & -13 \\ 0 & 0 & \underline{-4} & 11 & 43 \\ 0 & 0 & 0 & -3 & -8 \end{array}\right) \sim \left(\begin{array}{cccc|c} 1 & 0 & 0 & -2 & -5 \\ 0 & -1 & 0 & -0,25 & -2,25 \\ 0 & 0 & -4 & 11 & 43 \\ 0 & 0 & 0 & -3 & -8 \end{array}\right)$$

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 0,333 \\ 0 & -1 & 0 & 0 & -1,583 \\ 0 & 0 & -4 & 0 & 13,667 \\ 0 & 0 & 0 & \underline{-3} & -8 \end{array}\right)$$

Példa - megoldás

A kapott mátrix utolsó oszlopát végigosztva a megfelelő főátlóbeli értékekkel a megoldásvektor a következő lesz:

$$\bar{x} = \begin{pmatrix} 0,333 \\ 1,583 \\ -3,416 \\ 2,667 \end{pmatrix}$$

Pszeudokód

Input: a_{ij}, $(i = 1, \ldots, n, \;\; j = 1, \ldots, n+1)$
for $k = 1$ to n **do**
  for $i = 1$ to n **do**
    if $i \neq k$ **then**
      $l_{ik} \longleftarrow a_{ik}/a_{kk}$
      for $j = k+1$ to $n+1$ **do**
        $a_{ij} \longleftarrow a_{ij} - l_{ik} * a_{kj}$
      end for
    end if
  end for
end for
for $i = 1$ to n **do**
  $x_i \longleftarrow a_{i,n+1}/a_{ii}$
end for
Output: $x_1, x_2, \ldots, x_n$

Scilab kód

```
function GaussJordan(A,b)

sA=size(A);
sb=size(b);
if sA(1) ~= sA(2) | sA(1) ~= sb(1) | sb(2) ~= 1
    disp('Rossz dimenziók!')
    abort
end
n=sA(1);
A=[A,b];
printf('0. lépés:\n');
disp(A)
```

Scilab kód

```
for i=1:n
  if A(i,i)== 0
    printf('Főátlóban 0 érték van, az eliminációleáll!')
    abort
  end
  for j=1:n
    if i~=j
      kiv=A(j,i)/A(i,i);
      A(j,:)=A(j,:)-kiv*A(i,:);
    end
  end
  printf('%d. lépésben az iterációs mátrix:\n',i)
  disp(A)
end
```

Scilab kód

```
for i=1:n
  A(i,n+1)=A(i,n+1)/A(i,i)
  A(i,i)=1
end
printf('A megoldásvektor:\n')
disp(A(:,n+1))
endfunction
```

Az algoritmus futtatása megegyezik a Gauss-elimináció használatával.

Mátrix invertálás Gauss-Jordan-eliminációval

Legyen az A mátrix nemszinguláris négyzetes mátrix.
Ekkor az $AX = I$ mátrix egyenletnek a megoldása az A^{-1}-gyel jelölt inverz mátrix lesz. A fenti mátrix egyenletet megoldhatjuk Gauss-Jordan-eliminációval.
Az algoritmusban a változás mindösszesen annyi lesz, hogy a b oszlopvektor helyett most egy $n \times n$ méretű egységmátrixszal bővítjük az eredeti mátrixot és erre a kibővített $n \times 2n$ méretű mátrixra hajtjuk végre az eliminációt.

Példa

Számoljuk ki az alábbi mátrix inverzét.

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

Példa

Az invertálás Gauss-Jordan-eliminációval lépései:

$$\begin{pmatrix} 1 & 1 & 2 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 & 0 \\ -1 & 1 & -1 & 0 & 0 & 1 \end{pmatrix} \sim \begin{pmatrix} 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & -2 & -2 & -1 & 1 & 0 \\ 0 & 2 & 1 & 1 & 0 & 1 \end{pmatrix} \sim$$

$$\begin{pmatrix} 1 & 0 & 1 & 0,5 & 0,5 & 0 \\ 0 & -2 & -2 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & 0 & 0,5 & 1,5 & 1 \\ 0 & -2 & 0 & -1 & -1 & -2 \\ 0 & 0 & -1 & 0 & 1 & 1 \end{pmatrix}$$

Példa - megoldás

A kapott inverzmátrixunk a következő mátrix lesz:

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

Scilab kód

Az eljárásunk a Gauss-Jordan-eliminációhoz képest a paraméterek számában változott (egy paraméter lesz, az invertálandó mátrix). Emiatt a feltételvizsgálat egyszerűsödött.

```
function minv(A)
s=size(A);
if s(1)~=s(2)
  printf('Invertálni négyzetes mátrixot lehet!')
  abort
end
n=s(1);
A=[A,eye(n,n)];
disp('0. lépés:')
disp(A)
```

Scilab kód

```
for i=1:n
  if A(i,i)== 0
    printf('Főátlóban 0 érték van, az eliminációleáll!')
    abort
  end
  for j=1:n
    if i~=j
      kiv=A(j,i)/A(i,i);
      A(j,:)=A(j,:)-kiv*A(i,:);
    end
  end
  printf('%d. lépésben az iterációs mátrix:\n',i)
  disp(A)
end
```

Scilab kód

Az eliminációt követő osztást most nem csak a főátlóbeli elemre és az utolsó oszlopra kell végrehajtani, hanem az egész sorra (pontosabban csak az $n+1$. oszloptól a $2n$. oszlopig, de az újabb "for" ciklus helyettesíthető teljes sor hozzáadásával)

```
for i=1:n
  A(i,:)=A(i,:)/A(i,i)
end
printf('Az inverz mátrix:\n')
disp(A(:,n+1:2*n))
endfunction
```

Futtatása:

```
A=[1,1,2;1,-1,0;-1,1,-1]
minv(A)
```

Scilab programozás és a numerikus algoritmusok Scilab-ban
10. előadás
Főelemkiválasztásos Gauss eliminációk

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

Tartalomjegyzék

1. Gauss-elimináció részleges főelemkiválasztással
   - Mintafeladat
   - Scilab megvalósítás
2. Gauss-elimináció teljes főelemkiválasztással
   - Mintafeladat
   - Scilab megoldás
3. Gauss-elimináció sorkiegyenlítéssel
   - Mintafeladat
   - Scilab megoldás
4. Összefoglalás

Részleges főelemkiválasztás

A Gauss elimináció nem találja meg a megoldást abban az esetben, amikor a főátlóban 0 értéket talál az eljárás során. Ennek a megoldására születtek meg a főelemkiválasztásos eliminációk.

Általánosan

Általánosan elmondható, hogy a Gauss elimináció k. lépésében meg kell keresni a k. oszlopban a főátlóban és az alatta álló elemek közül a legnagyobb abszolút értékűt, azaz legyen

$$|a_{lk}| = max\{|a_{ik}| : i = k, \ldots, n\}$$

.

Cseréljük fel a k. és l. sort, majd folytassuk az eliminációt.

Abban az esetben, amikor a kiválaszott elem 0 (minden elem 0 az oszlopban), akkor nincs szükség sorcserére és eliminációra sem. A

Példa

Oldjuk meg a következő egyenletrendszert:

$$\begin{aligned}
x_1 &+ x_2 &+ &+ 3x_4 &= 4\\
2x_1 &+ x_2 &- x_3 &+ x_4 &= 1\\
3x_1 &- x_2 &- x_3 &+ 2x_4 &= -3\\
-x_1 &+ 2x_2 &+ 3x_3 &- x_4 &= 4
\end{aligned}$$

Mintafeladat

$$\left(\begin{array}{cccc|c} 1 & 1 & 0 & 3 & 4 \\ 2 & 1 & -1 & 1 & 1 \\ 3 & -1 & -1 & 2 & -3 \\ -1 & 2 & 3 & -1 & 4 \end{array}\right) \sim \left(\begin{array}{cccc|c} \underline{3} & -1 & -1 & 2 & -3 \\ 2 & 1 & -1 & 1 & 1 \\ 1 & 1 & 0 & 3 & 4 \\ -1 & 2 & 3 & -1 & 4 \end{array}\right) \sim$$

$$\left(\begin{array}{cccc|c} 3 & -1 & -1 & 2 & -3 \\ 0 & \underline{5/3} & -1/3 & -1/3 & 3 \\ 0 & 4/3 & 1/3 & 7/3 & 5 \\ 0 & 5/3 & 8/3 & -1/3 & 3 \end{array}\right) \sim \left(\begin{array}{cccc|c} 3 & -1 & -1 & 2 & -3 \\ 0 & 5/3 & -1/3 & -1/3 & 3 \\ 0 & 0 & 3/5 & 13/5 & 13/5 \\ 0 & 0 & 3 & 0 & 0 \end{array}\right)$$

Mintafeladat

$$\left(\begin{array}{cccc|c} 3 & -1 & -1 & 2 & -3 \\ 0 & 5/3 & -1/3 & -1/3 & 3 \\ 0 & 0 & \underline{3} & 0 & 0 \\ 0 & 0 & 3/5 & 13/5 & 13/5 \end{array}\right) \sim \left(\begin{array}{cccc|c} 3 & -1 & -1 & 2 & -3 \\ 0 & 5/3 & -1/3 & -1/3 & 3 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 13/5 & 13/5 \end{array}\right)$$

A megoldást egy trianguláris egyenlet megoldásaként kapjuk.

$$\bar{x} = \begin{pmatrix} -1 \\ 2 \\ 0 \\ 1 \end{pmatrix}$$

Pszeudókód

Input: a_{ij}, $(i = 1, \ldots, n, \;\; j = 1, \ldots, n+1)$
for $k = 1$ to $n-1$ **do**
  $max \longleftarrow abs(a_{kk})$, $index \longleftarrow k$
  for $l = k+1$ to n **do**
    if $abs(a_{lk}) > max$ **then**
      $max \longleftarrow abs(a_{lk})$
      $index \longleftarrow l$
    end if
  end for
  $csere(A, k, index)$
  for $i = k+1$ to n **do**
    $l_{ik} \longleftarrow a_{ik}/a_{kk}$
    for $j = k+1$ to $n+1$ **do**
      $a_{ij} \longleftarrow a_{ij} - l_{ik} * a_{kj}$
    end for
  end for
end for

Pszeudokód

$x_n \longleftarrow b_n / a_{nn}$
for $i = n - 1$ to 1 **do**
  $x_i \longleftarrow \left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j \right) / a_{ii}$
end for
Output: $x_1, x_2, \ldots, x_n$

Megjegyzés: A csere eljárás az A mátrix k-adik és *index*-edik sorát cseréli fel.

Scilab kód

```
function rgauss(A,b)

sA=size(A);
sb=size(b);
if sA(1) ~= sA(2) | sA(1) ~= sb(1) | sb(2) ~= 1
    disp('Rossz dimenziók!')
    abort
end
n=sA(1);
A=[A,b];
printf('0. lépés:\n');
disp(A)
```

Scilab kód

```
for k=1:n-1,
    [maxe,maxindex]=max(abs(A(k:n,k)));
    if maxindex>1,      // kell sorcsere
        sorindex=maxindex+k-1;
        sor=A(sorindex,:);
        A(sorindex,:)=A(k,:);
        A(k,:)=sor;
        printf('A(z) %d. és %d. sor cseréje:\n'
                                        ,k,sorindex);
        disp(A)
    end
```

Maximális abszolútértékű elem kiválasztása történik a "max" függvény segítségével.

- maxe - A maximális abszolútérték
- maxindex - a keresett részoszlopbeli indexe
- sorindex - a mátrixbeli indexe a maximális elemnek

Scilab kód

```
    for i=k+1:n,
        if A(k,k)==0,
            disp('Nem hajtható végre az eliminációs,
                                                főelem=0')
            abort
        end
```

Ebben az esetben nincs egyértelmű megoldása az egyenletrendszernek, hiszen a főátló alatt csupa 0 érték van.

Scilab kód

```
            l=A(i,k)/A(k,k);
            for j=k+1:n+1,
                A(i,j)=A(i,j)-l*A(k,j);
            end
            //nullázás a kiiratás kedvéért:
            for j=1:k,
                A(i,j)=0;
            end
        end
        printf('A(z) %d. eliminációs lépés:\n',k);
        disp(A)
end
```

Scilab kód

```
x(n)=A(n,n+1)/A(n,n);
for i=n-1:-1:1,
    x(i)=A(i,n+1);
    for j=i+1:n,
        x(i)=x(i)-A(i,j)*x(j);
    end
    x(i)=x(i)/A(i,i);
end
disp('A megoldás:')
disp(x)
endfunction
```

Teljes főelemkiválasztás

A részleges főelemkiválasztás sokat javít a Az alkalmazott módszer neve: **teljes főelemkiválasztás**.

Alapötlet

A Gauss-elimináció k-adik lépése előtt keressük meg az első olyan l és m sor- és oszlopindexet, amelyre

$$|a_{lm}| = max\{|a_{ij}| : i = k, \ldots, n, j = k, \ldots, n\}$$

Ekkor cseréljük fel a k-adik és l-edik sort, valamint a k-adik és m-edik oszlopot. Jegyezzük fel az együtthatók cseréjét és folytassuk az eliminációs eljárást.

A módszer hátránya: sokkal több összehasonlítást kell elvégeznünk a megoldás során.

Példa

Oldjuk meg a következő egyenletrendszert teljes főelemkiválasztással:

$$\begin{array}{rcrcrcrcr} x_1 & + & x_2 & + & & + & 3x_4 & = & 4 \\ 2x_1 & + & x_2 & - & x_3 & + & x_4 & = & 1 \\ 3x_1 & - & x_2 & - & x_3 & + & 2x_4 & = & -3 \\ -x_1 & + & 2x_2 & + & 3x_3 & - & x_4 & = & 4 \end{array}$$

Példa

Oldjuk meg a következő egyenletrendszert teljes főelemkiválasztással:

$$\begin{aligned} x_1 + x_2 + + 3x_4 &= 4 \\ 2x_1 + x_2 - x_3 + x_4 &= 1 \\ 3x_1 - x_2 - x_3 + 2x_4 &= -3 \\ -x_1 + 2x_2 + 3x_3 - x_4 &= 4 \end{aligned}$$

$$\left(\begin{array}{cccc|c} x_1 & x_2 & x_3 & x_4 & b \\ \hline 1 & 1 & 0 & 3 & 4 \\ 2 & 1 & -1 & 1 & 1 \\ 3 & -1 & -1 & 2 & -3 \\ -1 & 2 & 3 & -1 & 4 \end{array}\right)$$

Példa

Oldjuk meg a következő egyenletrendszert teljes főelemkiválasztással:

$$\begin{array}{rcrcrcrcr} x_1 & + & x_2 & + & & + & 3x_4 & = & 4 \\ 2x_1 & + & x_2 & - & x_3 & + & x_4 & = & 1 \\ 3x_1 & - & x_2 & - & x_3 & + & 2x_4 & = & -3 \\ -x_1 & + & 2x_2 & + & 3x_3 & - & x_4 & = & 4 \end{array}$$

$$\left(\begin{array}{cccc|c} x_1 & x_2 & x_3 & x_4 & b \\ \hline 1 & 1 & 0 & 3 & 4 \\ 2 & 1 & -1 & 1 & 1 \\ 3 & -1 & -1 & 2 & -3 \\ -1 & 2 & 3 & -1 & 4 \end{array}\right) \sim \left(\begin{array}{cccc|c} x_4 & x_2 & x_3 & x_1 & b \\ \hline \underline{3} & 1 & 0 & 1 & 4 \\ 1 & 1 & -1 & 2 & 1 \\ 2 & -1 & -1 & 3 & -3 \\ -1 & 2 & 3 & -1 & 4 \end{array}\right)$$

Mintafeladat

$$\left(\begin{array}{rrrr|r} x_4 & x_2 & x_3 & x_1 & b \\ \hline 3 & 1 & 0 & 1 & 4 \\ 0 & 2/3 & -1 & 5/3 & -1/3 \\ 0 & -5/3 & -1 & 7/3 & -17/3 \\ 0 & 7/3 & 3 & -2/3 & 16/3 \end{array}\right)$$

Mintafeladat

$$\left(\begin{array}{rrrr|r} x_4 & x_2 & x_3 & x_1 & b \\ \hline 3 & 1 & 0 & 1 & 4 \\ 0 & 2/3 & -1 & 5/3 & -1/3 \\ 0 & -5/3 & -1 & 7/3 & -17/3 \\ 0 & 7/3 & 3 & -2/3 & 16/3 \end{array}\right) \sim \left(\begin{array}{rrrr|r} x_4 & x_2 & x_3 & x_1 & b \\ \hline 3 & 1 & 0 & 1 & 4 \\ 0 & 7/3 & 3 & -2/3 & 16/3 \\ 0 & -5/3 & -1 & 7/3 & -17/3 \\ 0 & 2/3 & -1 & 5/3 & -1/3 \end{array}\right)$$

Mintafeladat

$$\left(\begin{array}{rrrr|r} x_4 & x_2 & x_3 & x_1 & b \\ \hline 3 & 1 & 0 & 1 & 4 \\ 0 & 2/3 & -1 & 5/3 & -1/3 \\ 0 & -5/3 & -1 & 7/3 & -17/3 \\ 0 & 7/3 & 3 & -2/3 & 16/3 \end{array}\right) \sim \left(\begin{array}{rrrr|r} x_4 & x_2 & x_3 & x_1 & b \\ \hline 3 & 1 & 0 & 1 & 4 \\ 0 & 7/3 & 3 & -2/3 & 16/3 \\ 0 & -5/3 & -1 & 7/3 & -17/3 \\ 0 & 2/3 & -1 & 5/3 & -1/3 \end{array}\right)$$

$$\sim \left(\begin{array}{rrrr|r} x_4 & x_3 & x_2 & x_1 & b \\ \hline 3 & 0 & 1 & 1 & 4 \\ 0 & \underline{3} & 7/3 & -2/3 & 16/3 \\ 0 & -1 & -5/3 & 7/3 & -17/3 \\ 0 & -1 & 2/3 & 5/3 & -1/3 \end{array}\right)$$

Mintafeladat

$$\left(\begin{array}{cccc|c} x_4 & x_2 & x_3 & x_1 & b \\ \hline 3 & 1 & 0 & 1 & 4 \\ 0 & 2/3 & -1 & 5/3 & -1/3 \\ 0 & -5/3 & -1 & 7/3 & -17/3 \\ 0 & 7/3 & 3 & -2/3 & 16/3 \end{array}\right) \sim \left(\begin{array}{cccc|c} x_4 & x_2 & x_3 & x_1 & b \\ \hline 3 & 1 & 0 & 1 & 4 \\ 0 & 7/3 & 3 & -2/3 & 16/3 \\ 0 & -5/3 & -1 & 7/3 & -17/3 \\ 0 & 2/3 & -1 & 5/3 & -1/3 \end{array}\right)$$

$$\sim \left(\begin{array}{cccc|c} x_4 & x_3 & x_2 & x_1 & b \\ \hline 3 & 0 & 1 & 1 & 4 \\ 0 & \underline{3} & 7/3 & -2/3 & 16/3 \\ 0 & -1 & -5/3 & 7/3 & -17/3 \\ 0 & -1 & 2/3 & 5/3 & -1/3 \end{array}\right) \sim \left(\begin{array}{cccc|c} x_4 & x_3 & x_2 & x_1 & b \\ \hline 3 & 0 & 1 & 1 & 4 \\ 0 & 3 & 7/3 & -2/3 & 16/3 \\ 0 & 0 & -8/9 & 19/9 & -35/3 \\ 0 & 0 & 13/9 & 13/9 & 13/9 \end{array}\right)$$

Példa

$$\sim \left(\begin{array}{cccc|c} x_4 & x_3 & x_1 & x_2 & b \\ \hline 3 & 0 & 1 & 1 & 4 \\ 0 & 3 & -2/3 & 7/3 & 16/3 \\ 0 & 0 & \underline{19/9} & -8/9 & -35/9 \\ 0 & 0 & 13/9 & 13/9 & 13/9 \end{array}\right)$$

Példa

$$\sim \left(\begin{array}{cccc|c} x_4 & x_3 & x_1 & x_2 & b \\ \hline 3 & 0 & 1 & 1 & 4 \\ 0 & 3 & -2/3 & 7/3 & 16/3 \\ 0 & 0 & \underline{19/9} & -8/9 & -35/9 \\ 0 & 0 & 13/9 & 13/9 & 13/9 \end{array}\right) \sim \left(\begin{array}{cccc|c} x_4 & x_3 & x_1 & x_2 & b \\ \hline 3 & 0 & 1 & 1 & 4 \\ 0 & 3 & -2/3 & 7/3 & 16/3 \\ 0 & 0 & 19/9 & -8/9 & -35/9 \\ 0 & 0 & 0 & 39/9 & 78/9 \end{array}\right)$$

Példa

$$\sim \left(\begin{array}{cccc|c} x_4 & x_3 & x_1 & x_2 & b \\ \hline 3 & 0 & 1 & 1 & 4 \\ 0 & 3 & -2/3 & 7/3 & 16/3 \\ 0 & 0 & \underline{19/9} & -8/9 & -35/9 \\ 0 & 0 & 13/9 & 13/9 & 13/9 \end{array}\right) \sim \left(\begin{array}{cccc|c} x_4 & x_3 & x_1 & x_2 & b \\ \hline 3 & 0 & 1 & 1 & 4 \\ 0 & 3 & -2/3 & 7/3 & 16/3 \\ 0 & 0 & 19/9 & -8/9 & -35/9 \\ 0 & 0 & 0 & 39/9 & 78/9 \end{array}\right)$$

Így tehát az eredményvektor:

$$\bar{x} = \begin{pmatrix} -1 \\ 2 \\ 0 \\ 1 \end{pmatrix}$$

Pszeudokód

Input: a_{ij}, $(i = 1, \ldots, n, \;\; j = 1, \ldots, n+1)$
for $k = 1$ to $n-1$ **do**
  SORESOSZLOPCSERE(A,k)
  for $i = k+1$ to n **do**
    $l_{ik} \leftarrow a_{ik}/a_{kk}$
    for $j = k+1$ to $n+1$ **do**
      $a_{ij} \leftarrow a_{ij} - l_{ik} * a_{kj}$
    end for
  end for
end for
$x_n \longleftarrow b_n/a_{nn}$
for $i = n-1$ to 1 **do**
  $x_i \longleftarrow \left(b_i - \sum_{j=i+1}^{n} a_{ij}x_j\right)/a_{ii}$
end for
Output: $x_1, x_2, \ldots, x_n$

Pszeudokód

A SORESOSZLOPCSERE(A,k) eljárás pszeudókódja:

$max \leftarrow |a_{kk}|$
$l \leftarrow k$
$m \leftarrow k$
for $i = k$ to n **do**
  for $j = k$ to n **do**
    if $|a_{ij}| > max$ **then**
      $max \leftarrow |a_{ij}|$
      $l \leftarrow i$
      $m \leftarrow j$
    end if
  end for
end for
$Csere(A, k, l, m)$

Scilab kód

```
function tgauss(A,b)

sA=size(A);
sb=size(b);
if sA(1) ~= sA(2) | sA(1) ~= sb(1) | sb(2) ~= 1
    disp('Rossz dimenziók!')
    abort
end
n=sA(1);
A=[A,b];
oind=1:n;
printf('0. lépés:\n');
disp(A)
```

Scilab kód

```
for k=1:n-1,
    [maxe,maxindex]=max(abs(A(k:n,k:n)));
    sorindex=maxindex(1)+k-1;
    oszlopindex=maxindex(2)+k-1;
    if sorindex>k,      // kell sorcsere
        sor=A(sorindex,:);
        A(sorindex,:)=A(k,:);
        A(k,:)=sor;
    end
    if oszlopindex>k,      // kell oszlopcsere
        oszlop=A(:,oszlopindex);
        A(:,oszlopindex)=A(:,k);
        A(:,k)=oszlop;
        oi=oind(k);
        oind(k)=oind(oszlopindex);
        oind(oszlopindex)=oi;
    end
```

Scilab kód

```
if sorindex>k & oszlopindex==k,
    printf('A(z) %d. és %d. sor cseréje:\n'
                              ,k,sorindex);
elseif  sorindex==k & oszlopindex>k,
    printf('A(z) %d. és %d. oszlop cseréje:\n'
                               ,k,oszlopindex);
elseif  sorindex>k & oszlopindex>k,
    printf('A(z) %d. és %d. sor, %d. és %d. oszlop
      cseréje:\n', k,sorindex,k,oszlopindex);
end
if sorindex>k | oszlopindex>k,
    disp(A)
end
```

Scilab kód

```
    for i=k+1:n,
        if A(k,k)==0,
            disp('Nem hajtható végre az eliminációs,
főelem=0')
            abort
        end
        l=A(i,k)/A(k,k);
        for j=k+1:n+1,
            A(i,j)=A(i,j)-l*A(k,j);
        end
        //nullázás a kiiratás kedvéért:
        for j=1:k,
            A(i,j)=0;
        end
    end
    printf('A(z) %d. eliminációs lépés:\n',k);
    disp(A)
end
```

Scilab kód

```
x(n)=A(n,n+1)/A(n,n);
for i=n-1:-1:1,
    x(i)=A(i,n+1);
    for j=i+1:n,
        x(i)=x(i)-A(i,j)*x(j);
    end
    x(i)=x(i)/A(i,i);
end
disp('A változók sorrendje')
disp(oind')
disp('A megoldás')
disp(x)
endfunction
```

Sorkiegyenlítés

Előfordulhat, hogy az együtthatómátrix elemei között komoly nagyságrendbeli eltérés van. Ekkor jelentős kerekítési hibával számolhatunk. Ezt kiküszöbölendő alkalmazható az a stratégia, hogy az egyenleteket megszorozzuk egy számmal, így az együtthatómátrix elemei közel azonos nagyságrendűek lesznek. Ezt a technikát **sorkiegyenlítésnek** nevezzük.

Általánosan

Keresünk tehát olyan $d_1, \ldots, d_n \neq 0$ számokat, hogy a $\mathbf{B} \equiv \mathbf{DA}$ mátrix elemei közel azonos nagyságrendűek legyenek, ahol $\mathbf{D} = \mathbf{diag(d_1, \ldots, d_n)}$. Ekkor az $\mathbf{Ax} = \mathbf{b}$ egyenletrendszer helyett a $\mathbf{DAx} = \mathbf{Db}$ egyenletrendszert oldjuk meg numerikusan. Az egyik egyszerű stratégia szerint úgy választjuk $\mathbf{D}$-t, hogy $max\{|b_{ij}| : 1 \leq j \leq n\} \approx 1$ legyen minden $i = 1, \ldots, n$-re. Ezt elérhetjük a $d_i \equiv 1/s_i, s_i \equiv max\{|a_{ij}| : 1 \leq j \leq n\}$ választással. Ezzel az a probléma, hogy az osztások további kerekítési hibát vezetnek be a számolásba.

Sorkiegyenlítés

Megoldás

Ezt kiküszöbölendő a következőt tehetjük: legyen β a számábrázolás alapja a számítógépen, és legyen r_i a legkisebb egész, hogy $\beta^{r_i} \geq s_i$, és legyen $b_{ij} \equiv a_{ij}/\beta^{r_i} (i, j = 1, \ldots, n)$. Ekkor az osztásnál nem lesz kerekítési hiba, és $1/\beta < max_{1 \leq j \leq n} |b_{ij}| \leq 1$ teljesül minden $i = 1, \ldots, n$-re.

Tétel (Hartung, 2011)

Tegyük fel, hogy egy $\mathbf{A}$ együtthatómátrixon sorkiegyenlítést végeztünk olyan $\mathbf{D} = \mathbf{diag}(\mathbf{d_1}, \ldots, \mathbf{d_n})$ szorzótényezőkkel (pl. β hatványokkal), amelyek nem eredményeztek kerekítési hibát. Ekkor ha a $\mathbf{DA}$ mátrixon végzett (részleges vagy teljes) főelemkiválasztás ugyanazokat a sorcseréket (és oszlopcseréket) eredményezi, mint az $\mathbf{A}$ mátrixon, akkor az $\mathbf{Ax} = \mathbf{b}$ és $\mathbf{DAx} = \mathbf{Db}$ egyenletek numerikus megoldásai pontosan ugyanazok lesznek.

Példa

$$\left(\begin{array}{rrr|r} 3 & 2 & -1 & 7 \\ 5 & 3 & 2 & 4 \\ -1 & 1 & -3 & -1 \end{array}\right) \longrightarrow \bar{s} = \begin{pmatrix} 3 \\ 5 \\ 3 \end{pmatrix}$$

$$\left(\begin{array}{rrr|r} 3 & 2 & -1 & 7 \\ 0 & -1/3 & 11/3 & -23/3 \\ 0 & 5/3 & -10/3 & 4/3 \end{array}\right) \sim \left(\begin{array}{rrr|r} 3 & 2 & -1 & 7 \\ 0 & 5/3 & -10/3 & 4/3 \\ 0 & -1/3 & 11/3 & -23/3 \end{array}\right) \sim$$

$$\left(\begin{array}{rrr|r} 3 & 2 & -1 & 7 \\ 0 & 5/3 & -10/3 & 4/3 \\ 0 & 0 & 3 & -37/5 \end{array}\right)$$

A megoldás:

$$\bar{x} = \begin{pmatrix} 64/15 \\ -62/15 \\ -37/15 \end{pmatrix}$$

Pszeudokód

Input: a_{ij}, $(i = 1, \ldots, n, \;\; j = 1, \ldots, n+1)$
for $i = 1$ to n **do**
  $s_i = max_{1 \leq j \leq n} |a_{ij}|$
end for
for $k = 1$ to $n - 1$ **do**
  legyen l a legkisebb olyan index, amelyre
  $|a_{lk}|/s_l = max_{k \leq i \leq n} |a_{ik}|/s_i$
  cseréljük fel az **A** mátrix k-adik és l-edik sorát
  for $i = k + 1$ to n **do**
    $l_{ik} \longleftarrow a_{ik}/a_{kk}$
    for $j = k + 1$ to $n + 1$ **do**
      $a_{ij} \longleftarrow a_{ij} - l_{ik} * a_{kj}$
    end for
  end for
end for

Pszeudokód

$x_n \longleftarrow a_{n,n+1}/a_{nn}$
for $i = n-1$ to 1 **do**
  $x_i \longleftarrow \left(a_{i,n+1} - \sum_{j=i+1}^{n} a_{ij}x_j\right)/a_{ii}$
end for
Output: $x_1, x_2, \ldots, x_n$

Scilab kód

```
function gaussor(A,b)

sA=size(A);
sb=size(b);
if sA(1) ~= sA(2) | sA(1) ~= sb(1) | sb(2) ~= 1
    disp('Rossz dimenziók!')
    return
end
n=sA(1);

s=[];
for i=1:n
    s=[s max(abs(A(i,:)))];
end

A=[A,b];
printf('0. lépés:\n');
```

Scilab kód

```
disp(A)
for k=1:n-1,
    seged = abs(A(k:n,k));
    for i=1:n-k+1
       seged(i) = seged(i) / s(i);
    end
    [maxe,maxindex]=max(seged);
    if maxindex>1,
        sorindex=maxindex+k-1;
        sor=A(sorindex,:);
        A(sorindex,:)=A(k,:);
        A(k,:)=sor;
        printf('A(z) %d. és %d. sor cseréje:\n'
                                        ,k,sorindex);
        disp(A)
    end
```

Scilab kód

```
    for i=k+1:n,
        if A(k,k)==0,
            disp('Nem hajtható végre az eliminációs,
                                        főelem=0')
            return
        end
        l=A(i,k)/A(k,k);
        for j=k+1:n+1,
            A(i,j)=A(i,j)-l*A(k,j);
        end
        for j=1:k,
            A(i,j)=0;
        end
    end
    printf('A(z) %d. eliminációs lépés:\n',k);
    disp(A)
end
```

Scilab kód

```
x(n)=A(n,n+1)/A(n,n);
for i=n-1:-1:1,
    x(i)=A(i,n+1);
    for j=i+1:n,
        x(i)=x(i)-A(i,j)*x(j);
    end
    x(i)=x(i)/A(i,i);
end
disp('megoldás:')
disp(x)
endfunction
```

Összefoglalás

A fenti három algoritmus futtatása Scilabban ugyanúgy történik, mint az előző előadáson megismert Gauss-elimináció. Mindegyik program két paramétert kap,

- együtthatómátrix (A)
- megoldás oszlopvektor (b)

Scilab programozás és a numerikus algoritmusok Scilabban
11. előadás
Jacobi és Gauss-Seidel iteráció

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

1. Jakobi-iteráció
   - Mintafeladat
   - Általános alak
   - Tételek
   - Scilab megvalósítás
2. Gauss-Seidel iteráció
   - Mintafeladat
   - Általános alak
   - Tételek
   - Scilab megvalósítás

Mintafeladat

Tekintsük az

$$\begin{array}{rcrcrcl} 4x_1 & + & 2x_2 & - & x_3 & = & 9 \\ 5x_1 & - & 10x_2 & + & 2x_3 & = & 8 \\ -2x_1 & + & 3x_2 & - & 7x_3 & = & 3 \end{array}$$

egyenletrendszert! Fejezzük ki az egyenletből x_1-et, a másodikból x_2-t, a harmadikból pedig x_3-at:

$$\begin{aligned} x_1 &= (9 - 2x_2 + x_3)/4 \\ x_2 &= (-8 + 5x_1 + 2x_3)/10 \\ x_3 &= (-3 - 2x_1 + 3x_2)/7. \end{aligned}$$

Mintafeladat

Ez az egyenletrendszer egy lineáris háromdimenziós fixpont egyenlet, ezért definiáljuk a következő iterációs módszert $k = 0, 1, 2, \ldots$-re:

$$\begin{aligned} x_1^{(k+1)} &= (9 - 2x_2^{(k)} + x_3^{(k)})/4 \\ x_2^{(k+1)} &= (-8 + 5x_1^{(k)} + 2x_3^{(k)})/10 \\ x_3^{(k+1)} &= (-3 - 2x_1^{(k)} + 3x_2^{(k)})/7, \end{aligned}$$

ahol az $x_1^{(0)}$, $x_2^{(0)}$, $x_3^{(0)}$ kezdeti értékek tetszőlegesen megválaszthatók.

Mintafeladat

Az alábbi táblázat az $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 0$ kezdeti értékekből számolt értékeket 10^-6 pontossággal mutatja.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0.0000000	0.0000000	0.0000000
1	2.2500000	-0.8000000	-0.4285714
2	2.5428571	0.2392857	-1.4142857
3	1.7767857	0.1221173	-1.0525510
4	1.8925765	-0.1221173	-0.8554082
⋮	⋮	⋮	⋮
19	2.0000268	-0.0000010	-1.0000107
20	1.9999978	0.0000112	-1.0000081
21	1.9999924	-0.0000027	-0.9999946
22	2.0000027	-0.0000027	-0.9999990
23	2.0000016	0.0000016	-1.0000019

Mintafeladat

Megfigyelhetjük, hogy az iterációs sorozat konvergens és a határértéke $x_1 = 2$, $x_2 = 0$, $x_3 = -1$, ami a feladat megoldása. Az iteráció röviden

$$\mathrm{x}^{(k+1)} = \mathrm{T}\mathrm{x}^{(k)} + \mathrm{c}$$

alakban írható fel, ahol

$$\mathbf{T} = \begin{pmatrix} 0 & -2/4 & 1/4 \\ 5/10 & 0 & 2/10 \\ -2/7 & 3/7 & 0 \end{pmatrix} \text{ és } \mathrm{c} = \begin{pmatrix} 9/4 \\ -8/10 \\ -3/7 \end{pmatrix}$$

A fent leírt lineáris fixpont iteráció konvergens, ha a $\mathbf{T}$ mátrix valamely normája kisebb mint 1. Mivel $\| \mathbf{T} \|_\infty = \max\{3/4, 7/10, 5/7\} < 1$, ezért az eredeti egyenletből felírt iteráció konvergens.

A Jacobi-iteráció általánosan

Tekintsük az általános

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \dots & + & a_{nn}x_n & = & b_n \end{array}$$

egyenletet. Ha $a_{ii} \neq 0$ minden $i = 1; \dots; n$-re, akkor ezt az egyenletet átírhatjuk

$$x_i = -\sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{a_{ij}}{a_{ii}} x_j + \frac{b_i}{a_{ii}}, \quad i = 1, \dots, n$$

alakba, és definiálhatjuk, az ún. *Jacobi-iterációt* $k = 0; 1; 2, \dots;$ n-re:

$$x_i^{(k+1)} = -\sum_{j=1}^{n} \frac{a_{ij}}{a_{ii}} x_j^{(k)} + \frac{b_i}{a_{ii}}, \quad i = 1; \dots; n.$$

A Jacobi-iteráció általánosan

Ha $a_{ii} = 0$ valamely i-re, akkor megpróbálhatjuk sorcserékkel (és oszlopcserével) elérni, hogy $a_{ii} \neq 0$ legyen $i = 1; ...; n$-re. Vezessük be a következő jelölést: $\mathbf{A}=\mathbf{L}+\mathbf{D}+\mathbf{U}$, ahol

$$\mathbf{L}= \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ a_{21} & 0 & 0 & \dots & 0 \\ a_{31} & a_{32} & 0 & \dots & 0 \\ \vdots & \vdots & & \ddots & \\ a_{n1} & a_{n2} & \dots & a_{n,n-1} & 0 \end{pmatrix}$$

A Jacobi-iteráció általánosan

$$\mathbf{U}=\begin{pmatrix} 0 & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & 0 & a_{23} & \dots & a_{2n} \\ 0 & 0 & 0 & \dots & a_{3n} \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 0 & a_{n-1,n} \end{pmatrix}$$

és $\mathbf{D}$=diag$(a_{11}, a_{22}, ..., a_{nn})$. $\mathbf{L}$ és $\mathbf{U}$ alulról ill. felülről triangu-láris mátrixok (amelyeknek a fődiagonálisa is zéró). Ezzel a je-löléssel az $\mathbf{Ax}$=$\mathbf{b}$ egyenletrendszert a $\mathbf{Dx}$=-$(\mathbf{L}+\mathbf{U})\mathbf{x}+\mathbf{b}$ alakba ír-juk, majd beszorozzuk az egyenletet balról $\mathbf{D}^{-1}$-gyel. Ennélfogva a Jacobi-iteráció az adott képlettel definiálható, ahol $\mathbf{T} = \mathbf{T_j} \equiv -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})$, és $\mathbf{c} = \mathbf{D}^{-1}\mathbf{b}$.

Jacobi-iterációhoz tartozó tételek

1. tétel (Hartung, 2011)
A Jacobi-iteráció akkor és csak akkor konvergens tetszőleges $x^{(0)}$ kezdeti vektorra, ha $\mathbf{T_j}$ spektrálsugara, $\rho(\mathbf{T_j}) < 1$.

Következmény:
Ha $\| \mathbf{T_J} \| < 1$ valamely $\| \bullet \|$ mátrixnormában, akkor a Jacobi-iteráció konvergens bármely $\mathbf{x^{(0)}}$ kezdeti értékre.

2. tétel
Ha $\mathbf{A}$ diagonálisan domináns, akkor a Jacobi-iteráció konvergens bármely $\mathbf{x^{(0)}}$ kezdeti értékre.

Jacobi-iteráció pszeudo kód

Input: A, b,x,t,
$p \longleftarrow 2 * x + t$
while $|x - p| > t$ **do**
  $p \longleftarrow x$
  for $i = 1$ to n **do**
    $x_i \longleftarrow b_i$
    for $j = 1$ to n **do**
      if $j \neq i$ **then**
        $x_i \longleftarrow x_i - A_{i,j} * p_j$
      end if
    end for
    $x_i \longleftarrow x_i / A_{i,i}$
  end for
end while
Output: $x_1, x_2, \ldots, x_n$

Scilab megvalósítás

```
function jacobi_iteracio2(A,b,x,t)
sA=size(A);
sb=size(b);
if sA(1) ~= sA(2) | sA(1) ~= sb(1) | sb(2) ~= 1
      disp('Rossz dimenziók!')
      return
end
n=sA(1);
printf('0. lépés:');
disp(x)
l=1;
p=x+2*t;
```

Scilab kód folytatás

```
while abs(x-p)>t & l<100,
    p=x;
    for i=1:n,
        x(i)=b(i);
        for j=1:n,
            if j~=i,
                x(i)=x(i)-A(i,j)*p(j);
            end
        end
        x(i)=x(i)/A(i,i);
    end
    printf('%d. lépés:',l);
    disp(x)
    l=l+1;
end
endfunction
```

Scilab kód

Scilab környezetben, ez a kód a következőképp fog kinézni:

```
function jacobi$(A,b,x,t)
sA=size(A);
sb=size(b);
if sA(1) ~= sA(2) | sA(1) ~= sb(1) | sb(2) ~= 1
    disp('Rossz dimenziók!')
    return
end
n=sA(1);
fprintf('0. lépés:');
disp(x)
l=1;
p=x+2*tol;
while abs(x-p)>t & l<100,
  p=x;
  for i=1:n,
    x(i)=b(i);
    for j=1:n,
      if j~=i,
        x(i)=x(i)-A(i,j)*p(j);
      end
    end
```

Példa

Tekintsük a feladatot:
Adja meg az alábbi egyenletrendszer megoldását három tizedesjegy pontossággal!

$$\begin{array}{rcrcrcr} 4x_1 & + & 2x_2 & - & x_3 & = & 9 \\ 5x_1 & - & 10x_2 & + & 2x_3 & = & 8 \\ -2x_1 & + & 3x_2 & - & 7x_3 & = & 3 \end{array}$$

Scilab megoldás

Az eljárás betöltése után (Execute menü), meg kell adnunk a paramétereket, amikkel futtatni szeretnénk az eljárást:

```
A=[4,2,-1;5,-10,2;-2,3,-7]
b=[9;8;3]
k=[0;0;0]
jacobi(A,b,k,0.001)
```

Mint azt láthatjuk, az **A** mátrix az egyenlet bal oldalán található együtthatókat, a **b** vektor pedig az egyenlet jobb oldalán található konstansokat tartalmazza. A **k** egy kezdeti vektorunk, illetve az utolsó sor (**jacobi**) tartalmazza a futtatási parancsot, és a szükséges adatokat, köztük a leállási feltételét az algoritmusnak (két egymást követő iterációs pont különbsége), melyet a kódban **t** néven szerepeltetünk.

Változók módosítása

Scilabban a megadott változókat egyszerűen módosíthatjuk a program jobb felső sarkában található ablakban.

Variable Browser

	Name	Dimension	Type	Visibility
	k	3x1	Double	local
	b	3x1	Double	local
	A	3x3	Double	local

Feladat megoldása

A futtatás után a következő eredményt kapjuk::

Scilab Console

```
    2.0069858
    0.0442746
  - 1.0383991
7. lépés:
    1.9682629
  - 0.0041869
  - 0.9830211
8. lépés:
    2.0063382
  - 0.0124728
  - 0.9927267
9. lépés:
    2.0080547
    0.0046238
  - 1.0071564
10. lépés:
    1.995899
    0.0025961
  - 1.0003197
11. lépés:
    1.998622
  - 0.0021144
  - 0.9977157
12. lépés:
    2.0016283
  - 0.0002321
  - 1.0005125
13. lépés:
    1.9999879
    0.0007117
  - 1.0005647

-->
```

Példa

Használjuk újra a Jacobi-iterációknál tárgyalt egyenletet! Definiáljuk az

$$\begin{aligned} x_1^{(k+1)} &= (9 - 2x_2^{(k)} + x_3^{(k)})/4 \\ x_2^{(k+1)} &= (-8 + 5x_1^{(k+1)} + 2x_3^{(k)})/10 \\ x_3^{(k+1)} &= (-3 - 2x_1^{(k+1)} + 3x_2^{(k+1)})/7 \end{aligned}$$

iterációt! Az a különbség a két definíció között, hogy ennél a módszernél amikor egy x_i változónak már kiszámoltuk az új értékét a $k+1$-edik iterációban, akkor ezt az új értéket már felhasználhatjuk a következő változó számításához: x_1 $k+1$-edik értékét az első egyenlettel számoljuk, az x_2 új értékének számításához már az x_1 új értékét (ami várhatóan jobb közelítése a megoldásnak mint $x_1^{(k)}$) használjuk a második egyenletben $x_3^{(k)}$-val együtt, mivel annak még nem számoltunk új értéket.

Mintafeladat

A táblázatban található a módszernek az $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 0$ kezdeti értékhez tartozó numerikus eredménye 10^{-6} pontossággal.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0.0000000	0.0000000	0.0000000
1	2.2500000	0.3250000	-0.9321429
2	1.8544643	-0.0591964	-0.9837883
3	2.0336511	0.0200679	-1.0010141
4	1.9897125	-0.0053466	-0.9993521
5	2.0028353	0.0015472	-1.0001470
6	1.9991897	-0.0004346	-0.9999547
7	2.0002286	0.0001234	-1.0000124
8	1.9999352	-0.0000349	-0.9999964
9	2.0000183	0.0000099	-1.0000010
10	1.9999948	-0.0000028	-0.9999997
11	2.0000015	0.0000008	-1.0000001
12	1.9999996	-0.0000002	-1.0000000

Gauss-Seidel-iteráció

Láthatjuk, hogy ez az iterációs módszer gyorsabban konvergál ezen a feladaton, mint a Jacobi-iteráció. Az általános lineáris egyenletrendszer megoldására definiáljuk a Gauss-Seidel-iterációt $k = 0, 1, 2, ...$-re (ha $a_{ii} \neq 0$ minden $i = 1, ..., n$-re):

$$x_i^{(k+1)} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(k+1)} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^{(k)} + \frac{b_i}{a_{ii}}, \quad i = 1, ..., n$$

Ez az egyenlet átrendezhető a következő alakba:

$$\sum_{j=1}^{i} a_{ij} x_j^{(k+1)} = -\sum_{j=i+1}^{n} a_{ij} x_j^{(k)} + b_i, \quad i = 1, ..., n$$

Gauss-Seidel-iteráció

azaz mátrix jelöléssel

$$(\mathbf{D}+\mathbf{L})\mathbf{x}^{(k+1)} = -\mathbf{U}\mathbf{x}^{(k)} + \mathbf{b},$$

ahol **L**, **D**, **U** ugyanaz, mint a Jacobi-iterációnál. Látható, hogy a Gauss-Seidel-iteráció is felírható

$$x^{(k+1)} = \mathbf{T}\mathbf{x}^{k} + \mathbf{c}$$

alakban $\mathbf{T} = \mathbf{T}_G \equiv -(\mathbf{D}+\mathbf{L})^{-1}\mathbf{U}$ és $\mathbf{c} = (\mathbf{D}+\mathbf{L})^{-1}\mathbf{b}$ választással.

Gauss-Seidel-iterációhoz tartozó tételek

1. tétel (Hartung, 2011)
A Gauss-Seidel-iteráció akkor és csak akkor konvergens tetszőleges $x^{(0)}$ kezdővektorra, ha $\rho(\mathbf{T}_G) < 1$.

Következmény
Ha $\| \mathbf{T}_G \| < 1$ valamely $\| \bullet \|$ mátrixnormában, akkor a Gauss-Seidel-iteráció konvergens bármely $\mathbf{x}^{(0)}$ kezdeti vektorra.
Megmutatható, hogy a Jacobi-iterációhoz hasonlóan diagonálisan domináns mátrixokra a Gauss-Seidel-módszer is konvergens.

2. tétel (Hartung, 2011)
Ha $\mathbf{A}$ diagonálisan domináns, akkor a Gauss-Seidel-iteráció konvergens bármely $\mathbf{x}^{(0)}$ kezdeti vektorra.

Gauss-Seidel-iterációhoz tartozó tételek

Ebből az egyenlőtlenségből következik az is, hogy diagonálisan domináns mátrixok esetén a Gauss-Seidel-módszerre jobb hibabecslést tudunk adni, mint a Jacobi-iterációra, tehát várhatóan legalább olyan gyorsan konvergál, mint a Jacobi-iteráció. Az általános esetben az, hogy a Jacobi- vagy a Gauss-Seidel-iteráció konvergál-e gyorsabban, attól függ, hogy a $\rho(\mathbf{T}_J)$ vagy $\rho(\mathbf{T}_G)$ kisebb-e. Ennek eldöntésére, az $\mathbf{A}$ mátrix együtthatói ismeretében, nem ismert egyszerű feltétel. Egy speciális esetre vonatkozik a következő tétel.

Gauss-Seidel-iterációhoz tartozó tételek és következményeik VI.

3. tétel (Stein-Rosenberg) Tegyük fel, hogy $a_{ij} \leq 0$ ha $i \neq j$ és $a_{ii} > 0$ minden $i = 1, ..., n$-re. Ekkor a következő állítások közül pontosan egy teljesül:

- $0 \leq \rho(\mathbf{T}_G) < \rho(\mathbf{T}_J) < 1$
- $1 < \rho(\mathbf{T}_J) < \rho(\mathbf{T}_G)$
- $\rho(\mathbf{T}_J) = \rho(\mathbf{T}_G) = 0$
- $\rho(\mathbf{T}_J) = \rho(\mathbf{T}_G) = 1$

A tételből következik, hogy a feltételeknek eleget tevő együtthatómátrixú egyenletrendszerek esetében a Jacobi-iteráció pontosan akkor konvergens, amikor a Gauss-Seidel-iteráció, és a Gauss-Seidel-iteráció mindig gyorsabban konvergál. Általában viszont nem igaz, hogy ha a Gauss-Seidel-iteráció konvergens, akkor a Jacobi is az, vagy fordítva.

Pszeudokód

Input: A, b,x,t,
$p \longleftarrow 2 * x + t$
while $|x - p| > t$ **do**
  $p \longleftarrow x$
  for $i = 1$ to n **do**
    $x_i \longleftarrow b_i$
    for $j = 1$ to $i - 1$ **do**
      $x_i \longleftarrow A_{(i,j)} * x_j$
    end for
    for $j = i + 1$ to n **do**
      $x_i \longleftarrow A_{(i,j)} * p_j$
    end for
    $x_i \longleftarrow x_i / A_{i,i}$
  end for
end while
Output: $x_1, x_2, \ldots, x_n$

Scilab megvallósítás

A Gauss-Seidel-iteráció programkódja:

```
function g_s(A,b,x,t)
sA=size(A);
sb=size(b);
if sA(1) ~= sA(2) | sA(1) ~= sb(1) | sb(2) ~= 1
    disp('Rossz dimenziók!')
    abort
end
n=sA(1);
printf('0. lépés:');
disp(x)
l=1;
p=x+2*t;
```

Scilab megvalósítás

```
while abs(x-p)>t & l<100,
    p=x;
    for i=1:n,
        x(i)=b(i);
        for j=1:i-1,
            x(i)=x(i)-A(i,j)*x(j);
        end
        for j=i+1:n,
            x(i)=x(i)-A(i,j)*p(j);
        end
        x(i)=x(i)/A(i,i);
    end
    printf('%d. lépés:',l);
    disp(x)
    l=l+1;
end
endfunction
```

Gauss-Seidel-iteráció futtatása Scilabban

Az előző egyenletek adatait használva adjuk meg itt is a szükséges adatokat, majd futtassuk le az eljárást! A megoldást 10^{-3}pontossággal keressük

```
-->exec('C:\Users\user\Downloads\g_s.sci', -1)
0. lépés:
    0.
    0.
    0.
1. lépés:
    2.25
    0.325
  - 0.9321429
2. lépés:
    1.8544643
  - 0.0591964
  - 0.9837883
3. lépés:
    2.0336511
    0.0200679
  - 1.0010141
4. lépés:
    1.9897125
  - 0.0053466
  - 0.9993521
5. lépés:
    2.0028353
    0.0015472
  - 1.000147

-->
```

Gauss-Seidel-iteráció ellenőrzés

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0.0000000	0.0000000	0.0000000
1	2.2500000	0.3250000	-0.9321429
2	1.8544643	-0.0591964	-0.9837883
3	2.0336511	0.0200679	-1.0010141
4	1.9897125	-0.0053466	-0.9993521
5	2.0028353	0.0015472	-1.0001470

Scilab programozás és a numerikus algoritmusok Scilabban
12. előadás
Numerikus integrálás

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

1. Határozott integrál
2. Numerikus integrálási módszerek
3. Trapéz formula
   - Definíció
   - Mintafeladat
   - Scilab megvalósítás
4. Simpson formula
   - Definíció
   - Mintafeladat
   - Scilab megvalósítás
5. Két görbe közötti terület közelítő kiszámítása
6. Beépített függvények

Határozott integrál

Határozzuk meg egy f függvény grafikonja alatti területet, azaz egy kicsit pontosabban: határozzuk meg az $y = 0$, $x = a$, $x = b$ egyenletű egyenesek és a nemnegatív f függvény grafikonja által határolt síktartomány (görbevonalú trapéz) területét.
Megoldás: Ha f integrálható $[a, b]$-n, akkor $T = \int_a^b f(x)dx$.

Tétel

Newton-Leibniz formula, Ha F az f egy primitív függvénye, akkor $\int_a^b f(x)dx = [F(x)]_a^b = F(b) - F(a)$.

Határozott integrál fogalma

Példa

Határozzuk meg az $f(x) = x^2$ képlettel adott függvény grafikonja alatti területet, ha $a = 0$ és $b = 1$.

Megoldás:

$$T = \int_a^b f(x)dx = \int_0^1 x^2 dx = \left[\frac{x^3}{3}\right]_0^1 = \frac{1}{3}$$

Megjegyzés:
Ha az előző problémafelvetésben a f-re vonatkozó nemnegativitási feltételt elhagyjuk, akkor a határozzott integrál értéke előjeles területet ad, vagyis pl. $\int_0^{2\pi} \sin(x)dx = 0$, jóllehet a keletkező síktartomány területe nyilván nem 0, hanem 4.

Numerikus integrálási módszerek

Numerikus integrálás fogalma

A numerikus integrálás közelítő eljárás az integrál kiszámítására. A numerikus analízisben a numerikus integrálás számos algoritmust jelent, melyek segítségével egy határozott integrál közelítőleg kiszámítható. Több esetben is használatos eljárások, például, ha a függvény integrálható, de nincs primitív függvénye az elemi függvények körében.

A numerikus integrálás általában az jelenti, hogy egy integrál közelítő értékét számoljuk ki.

Numerikus integrálási módszerek

Az integranduszt véges pontokban határozzák meg, ezeket integrálási pontoknak is hívják, és ezen értékek súlyozott összegével közelítik az integrált. Az integrálási pontok és a súlyozás az alkalmazott módszertől és a kívánt pontosságtól függ.

Elemi trapézformula

Elemi trapézformula

$$\int_a^b f(x)\,\mathrm{d}x = \frac{h}{2}\left(f(a)+f(b)\right) - \frac{h^3}{12}f''(\xi), \qquad \xi \in (a,b),$$

ahol $h = b - a$.
A formula a nevét onnan kapta, hogy a kifejezés az f függvény grafikonjának a és b x-koordinátájú pontjához tartozó húr alatti területet (egy trapéz területe).
Akkor alkalmazható sikeresen, ha az intervallum hossza kicsi.
Nagyobb intervallumok esetén osszuk fel az intervallumot egyenlő részekre és a kapott részintervallumokra alkalmazzuk a szabályt.

Összetett trapézformula

Az $[a; b]$ intervallumot n egyenlő részre osztva minden részintervallumra alkalmazva az elemi trapézformulát kapjuk az összetett formulát.

Összetett trapézformula

$$\int_a^b f(x)\,\mathrm{d}x = \frac{h}{2}\left(f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right) - \frac{(b-a)h^2}{12} f''(\xi),$$

ahol $i = 0; 1; \ldots, n$, $h = \dfrac{b-a}{n}$, $x_i = a + ih$.

Mintafeladat

Számolja ki az alábbi integrál közelítő értékét trapézformulával, az intervallumot 4 részre osztva.

$$\int_0^{\pi} \sin(x)\,\mathrm{d}x$$

Elsőként a részintervallumok hosszát határozzuk meg:

$$h = \frac{b-a}{n} = \frac{\pi}{4}$$

Az osztópontok a következők lesznek:

$$x_0 = 0, \qquad x_1 = \frac{\pi}{4}, \qquad x_2 = \frac{\pi}{2}, \qquad x_3 = \frac{3\pi}{4}, \qquad x_4 = \pi$$

Mintafeladat

Az osztópontokhoz tartozó függvényértékek a következők:

$$f(x_0) = 0, \qquad f(x_1) = \frac{\sqrt{2}}{2}, \qquad f(x_2) = 1$$

$$f(x_3) = \frac{\sqrt{2}}{2}, \qquad f(x_4) = 0$$

Behelyettesítve a formulába:

$$\int_0^{\pi} \sin(x)\,\mathrm{d}x = \frac{\pi}{8}\left(f(x_0) + 2 * (f(x_1) + f(x_2) + f(x_3)) + f(x_4)\right) =$$

$$\frac{\pi}{8}\left(2\sqrt{2} + 2\right) = 1,896118898$$

Mintafeladat

A pontos megoldás:

$$\int_0^{\pi} \sin(x)\,\mathrm{d}x = [-\cos(x)]_0^{\pi} = -\cos(\pi) - (-\cos(0)) = 2$$

A trapéz formulával kapott megoldásunk:

$$\int_0^{\pi} \sin(x)\,\mathrm{d}x = \frac{\pi}{8}\left(2\sqrt{2}+2\right) = 1,896118898$$

Már négy részre osztva az intervallumot is majdnem $0,01$ pontosságú megoldást kaptunk.

Trapézformula Scilabban

```
function trapez(f,a,b,n)
h=(b-a)/n;
for i=1:n+1
x(i)=a+h*(i-1);
end
mo=f(a)+f(b);
for i=2:n
mo=mo+2*f(x(i));
end
mo=h/2*mo
printf('A közelítő megoldás %d részre osztva az
                        intervallumot: %1.14f\n',n,mo)
endfunction
```

Trapézformula Scilabban

Paraméter lista

- f: függvény neve
- a: intervallum kezdete
- b: intervallum vége
- n: részintervallumok száma

Trapézformula Scilabban

A program működése:

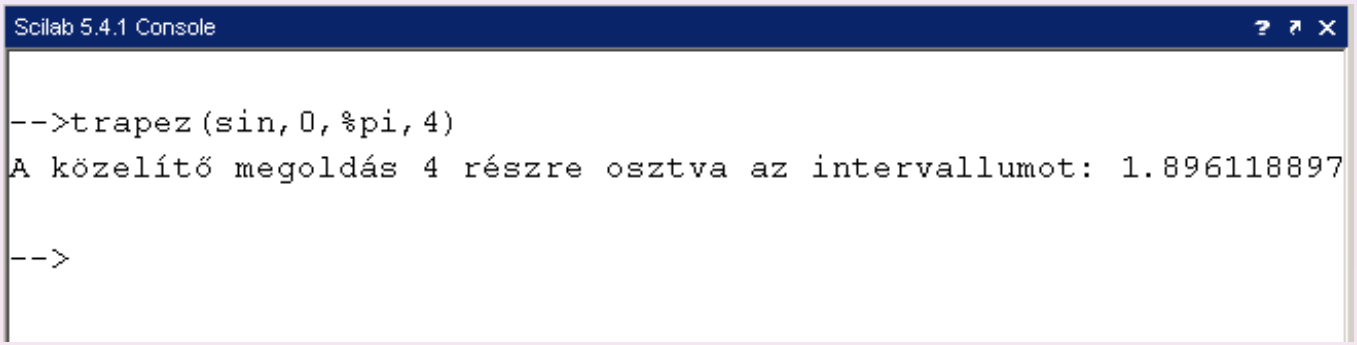
- A függvény először kiszámolja a részintervallumok hosszát
- Az osztópontokat egy **x** vektorba tárolja ($n+1$ elemű).
- A formulában megadott módon behelyettesít a formulába
- Végül kiírja a kapott közelítő megoldást.

Trapézformula Scilabban

Az eljárás műveletigényét csökkenthetjük a következő módon:

```
function trapez(f,a,b,n)
h=(b-a)/n;
for i=1:n+1
x(i)=a+h*(i-1);
end
mo=(f(a)+f(b))/2;
for i=2:n
mo=mo+f(x(i));
end
mo=h *mo
printf('A közelítő megoldás %d részre osztva az
                        intervallumot: %1.14f\n',n,mo)
endfunction
```

Mintafeladat megoldása Scilabban



Természetesen nem elemi függvény integráljának számításakor a függvényt létre kell hoznunk, mint Scilab függvényt.

Simpson-formula

Elemi Simpson-formula

$$\int_a^b f(x)\,\mathrm{d}x = \frac{h}{3}\left(f(a) + 4f(x_0 + h) + f(b)\right) - \frac{h^5}{90} f^{(4)}(\eta), \qquad \eta \in (a, b),$$

ahol $h = \dfrac{b-a}{2}$.

Látható. hogy az elemi Simpson-formula már tartalmaz egy intervallum osztást. Amikor az elemi szabályt alkalmazzuk többször (felosztjuk az intervallumot), pontosan kétszer annyi osztópontunk lesz, mint ahányszor végrehajtjuk a felosztást.

Összetett Simpson-formula

Az $[a; b]$ intervallumot n egyenlő részre osztva minden részintervallumra alkalmazva az elemi Simpson-formulát kapjuk az összetett formulát. Az elemi formulában szereplő osztás miatt valójában $2n$ részre osztjuk fel az intervallumot.

Összetett trapézformula

$$\int_a^b f(x)\,\mathrm{d}x = \frac{h}{3}\left(f(x_0) + 4\sum_{i=1}^{n} f(x_{2i-1}) + 2\sum_{i=1}^{n-1} f(x_{2i}) + f(x_n)\right) - \frac{(b-a)h^4}{180} f^{(4)}(\eta), \qquad \eta \in$$

ahol $i = 0; 1; \ldots; 2n$, $h = \dfrac{b-a}{2n}$, $x_i = a + ih$.

Mintafeladat

Tekintsük a trapézformulánál számolt integrált.
Számolja ki az alábbi integrál közelítő értékét kétszer használva az egyszerű Simpson-formulát.

$$\int_0^{\pi} \sin(x)\,\mathrm{d}x$$

Elsőként a részintervallumok hosszát határozzuk meg:

$$h = \frac{b-a}{2n} = \frac{\pi}{4}$$

Az osztópontok a következők lesznek:

$$x_0 = 0, \qquad x_1 = \frac{\pi}{4}, \qquad x_2 = \frac{\pi}{2}, \qquad x_3 = \frac{3\pi}{4}, \qquad x_4 = \pi$$

Mintafeladat

Az osztópontokhoz tartozó függvényértékek a következők:

$$f(x_0) = 0, \qquad f(x_1) = \frac{\sqrt{2}}{2}, \qquad f(x_2) = 1$$

$$f(x_3) = \frac{\sqrt{2}}{2}, \qquad f(x_4) = 0$$

Behelyettesítve a formulába:

$$\int_0^{\pi} \sin(x)\,\mathrm{d}x = \frac{\pi}{12}\left(f(x_0) + 4(f(x_1) + f(x_3)) + 2f(x_2) + f(x_4)\right) =$$

$$\frac{\pi}{12}\left(4\sqrt{2} + 2\right) = 2,004559755$$

Mintafeladat

A pontos megoldás:

$$\int_0^{\pi} \sin(x)\,\mathrm{d}x = [-\cos(x)]_0^{\pi} = -\cos(\pi) - (-\cos(0)) = 2$$

A Simpson-formulával kapott megoldásunk:

$$\int_0^{\pi} \sin(x)\,\mathrm{d}x = \frac{\pi}{8}\left(2\sqrt{2}+2\right) = 2,004559755$$

Már négy részre osztva az intervallumot is $0,01$ pontosságú megoldást kaptunk.

Simpson-formula Scilabban

```
function Simpson(f,a,b,n)
if pmodulo(n,2)~=0
printf('A negyedik paraméternek párosnak kell lennie!')
abort
end
h=(b-a)/n;
for i=1:n+1
x(i)=a+h*(i-1);
end
mo=f(a)+f(b);
mo1=0;
for i=1:n/2
mo1=mo1+f(x(2*i));
end
```

Simpson-formula Scilabban

```
mo2=0;
for i=1:(n/2-1)
mo2=mo2+f(x(2*i+1));
end
mo=h/3*(mo+4*mo1+2*mo2)
printf('A közelítő megoldás %d részre osztva az
                        intervallumot: %1.14f\n',n,mo)
endfunction
```

Simpson-formula Scilabban

Paraméter lista

- f: Függvény neve.
- a: Intervallum kezdete.
- b: Intervallum vége.
- n: Részintervallumok száma, mely ebben a programban csak páros lehet (ezek a belső részintervallumokat jelentik). Pontosan a kétszer annyi, mint ahányszor az elemi Simpson-formulát alkalmazzuk.

Fontos megjegyezni, hogy a formulában a páratlan sorszámú osztópontokhoz tartozó függvényértékeket négyszeres, a párosadikakat kétszeres súllyal számoltuk. A Scilab eljárásban ez pont fordítva történik. Ennek oka, hogy az osztópontokat számoló vektor elemeinek számozása eggyel kezdődik, vagyis a vektor első eleme lesz x_0.

Simpson-formula Scilabban

```
if pmodulo(n,2)~=0
printf('A negyedik paraméternek párosnak kell lennie!')
abort
end
```

A fenti feltétel azt vizsgálja, hogy páros sok részre osztjuk-e az intervallumot. Amennyiben nem, akkor kilép az eljárásból. Másfajta megoldás lehet, ha futtatás közben módosítható az intervallumok száma, figyelmeztetve egyben arra, hogy páros értéket adjon meg. Ekkor az "abort" utasítás helyett írjuk a következőt:

```
n=input('Adja meg, hány páros sok részre
                    osszam az intervallumot: ')
```

Simpson-formula Scilabban

Amennyiben szeretnénk, hogy az x vektor a valóságnak megfelelően tartalmazza az osztópontokat (x első eleme valóban x_1 legyen), akkor a programot elég jelentősen át kell alakítani. Ekkor a két végpontot nem vesszük fel a vektor elemei közé, vagyis csak a belső osztópontokat tartalamzza majd a programbeli x változó.

```
function Simpson(f,a,b,n)
if pmodulo(n,2)~=0
printf('A negyedik paraméternek párosnak kell lennie!')
abort
end
h=(b-a)/n;
for i=1:n-1
x(i)=a+h*(i);
end
mo=f(a)+f(b);
```

Mintafeladat megoldása Scilabban

```
Scilab 5.4.1 Console

-->Simpson(sin,0,%pi,4)
A közelítő megoldás 4 részre osztva az intervallumot: 2.004559754

-->
```

Természetesen nem elemi függvény integráljának számításakor a függvényt létre kell hoznunk, mint Scilab függvényt.

Függvények közti terület meghatározása

Számolja ki az alábbi két függvény grafikonja által közrezárt korlátos síkidom közelítő területét:

$$f(x) = x^2$$

$$g(x) = 2x$$

Függvények közti terület meghatározása

Először hozzuk létre a két függvényt.

```
function y=f(x)
y=x^2
endfunction
```

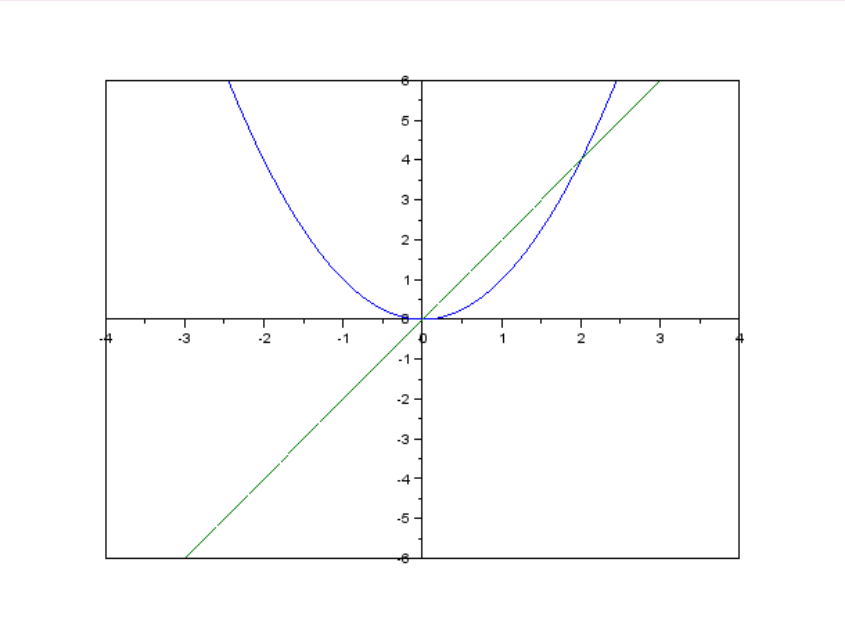
Illetve

```
function y=g(x)
y=2*x
endfunction
```

Függvények közti terület meghatározása

Első lépésben ábrázoljuk a függvényeket, hogy megtudjuk, mely függvény a körülhatárolt terület felső korlátja.
Ezt a következő módon tehetjük meg:

```
->x=[-5:0.1:5]
->plot(x,f(x),x,g(x))
```



Függvények közti terület meghatározása

Ezek után a két függvényt összevonjuk.

```
function y=fgv(x)
y=2*x-x^2
endfunction
```

Függvények közti terület meghatározása

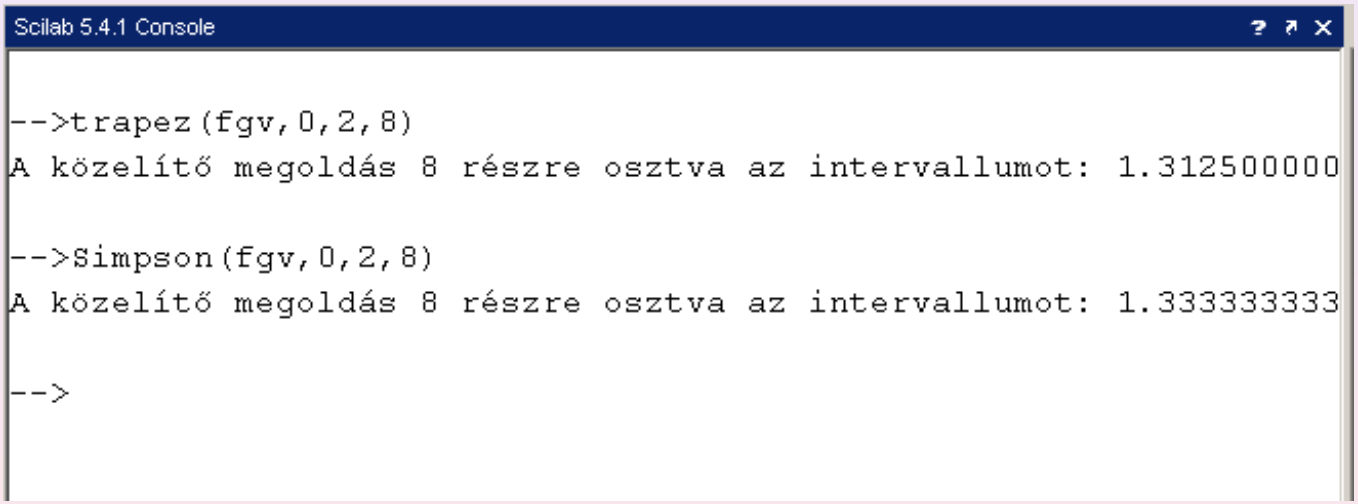
A két metszéspont meghatározáshoz használhatjuk a gyökkereső algoritmusok valamelyikét. A húrmódszer az alábbi paraméterezéssel minkét végpontot 16 lépésben megtalálja:

```
Hur(fgv,-2,1,0.0001)
Hur(fgv,1,4,0.0001)
```

A két metszéspont a 0 és a 2 pontok lettek.

Függvények közti terület meghatározása

Lefuttatva mindkét módszerrel az eredmény a következő lett:



A feladat jellegéből adódóan a Simpson-formula pontos megoldást adott.

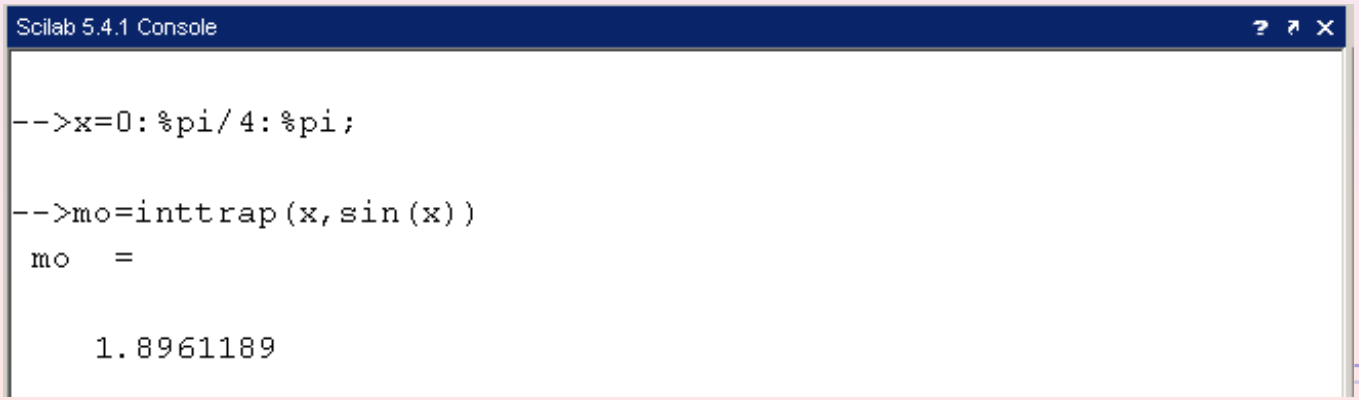
Beépített trapézformula

A Scilab program rendelkezik beépített utasítással, ami az összetett trapézformula alapján számolja ki a közelítő integrál értékét.
A következőképpen tudjuk használni:

```
inttrap([x,] s)
```

ahol

- x: az osztópontokat tartalmazó vektor
- s: a függvény, aminek az integrálját keressük



Beépített Sipmson-formula

A Scilab program rendelkezik beépített utasítással, ami az összetett Simpson-formula alapján számolja ki a közelítő integrál értékét.
A következőképpen tudjuk használni:

```
intsplin([x,] s)
```

ahol

- x: az osztópontokat tartalmazó vektor
- s: a függvény, aminek az integrálját keressük

Integrálás a Scilab környezetben

A Scilab rendelkezik saját, beépített integráló függvénnyel, melyet az

–>integrate

paranccsal érünk el.

Bővebben erről a beépített funkcióról a

–>help integrate

parancs kiadása után olvashatunk.

Paraméter lista

- expr: az integrandusz
- v: az integrate parancs többváltozós függvényeket is képes integrálni, így meg kell adnunk, mely változó szerint akarunk integrálni
- x_0: intervallum alsó határa
- x_1: intervallum felső határa
- atol: abszolút hiba. A Simpson és Trapéz formulákkal ellentétben, itt nem a részintervallumok számát adjuk meg, hanem az abszolút hiba nagyságát, tizedestörtben
- rtol: (opcionális) relatív hiba

Scilab programozás és a numerikus algoritmusok Scilabban
13. előadás
Interpolációk

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

1. Interpolációs feladat
2. Lineáris interpoláció
3. Lagrange interpoláció
   - Definíciók
   - Scilab megoldás
4. Lagrange interpoláció Newton-féle alakja
   - Osztott differenciák
   - Scilab megvalósítás
5. Hermite-interpoláció
   - Definíció
   - Scilab megvalósítás
6. Spline interpoláció

Interpoláció, Interpolációs feladat

Adottak $x_0, x_1, ..., x_n \in [a, b]$ páronként különböző osztópontok, és a hozzá tartozó $y_0, y_1, ..., y_n$ függvényértékek.
Keresünk olyan g függvényt, amely *interpolálja* a megadott pontokat, azaz teljesíti a

$$g(x_i) = y_i, \qquad i = 0, 1..., n$$

egyenleteket. Az interpolációs feladat geometriai jelentése az, hogy olyan g függvényt keresünk, amely valamely megadott tulajdonságokkal rendelkezik, és a grafikonja átmegy az (x_i, y_i) pontokon.

Lineáris interpoláció

Az egyik legegyszerűbb megoldás, hogy az ismeretlen (y) értékét az adott ponthoz (x) balról és jobbról legközelebb eső ismert osztópontokból és a hozzájuk tartozó értékekből számoljuk ki, lineárisan:

$$y = y_0 + \frac{(x - x_0)(y_1 - y_0)}{x_1 - x_0}$$

Az n darab ismert értékhez illesztett általános függvény, szakaszokra bontva:

$$y^* = y_i + \frac{(x^* - x_i)(y_{i+1} - y_i)}{x_{i+1} - x_i}; \text{ ahol } i \in \{0, n-1\} \subset \mathbb{N},$$

$$\text{valamint } x_i < x_{i+1} \text{ és } x_i <= x^* < x_{i+1}$$

Lineáris interpoláció

Lineáris interpolációt számolhatunk, illetve ábrázolhatunk az alábbi Scilab utasítással:

Függvényhívás

```
[y]=interpln(xyd,x)
```

Paraméterek:

- xyd - 2×n mátrix (a pontok x,y koordinátái)
- x - vektor (x-koordináták a számoláshoz, ábrázoláshoz)

Az y változó tartalmazza majd az x értékekhez tartozó függvényértékeket.

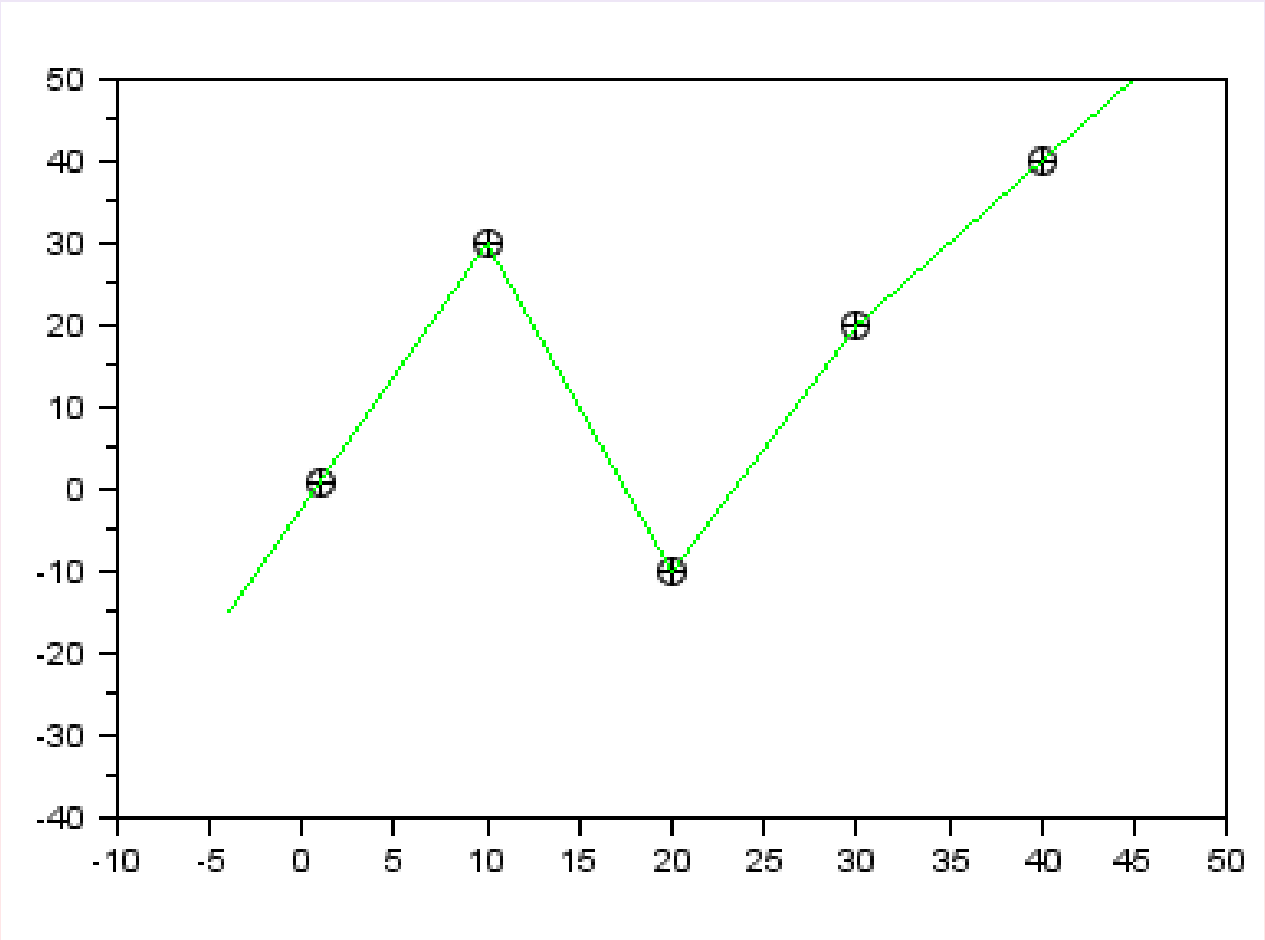
Lineáris interpoláció, Scilab beépített függvény

Példa

```
x=[1  10  20  30  40];
y=[1  30  -10  20  40];
plot2d(x',y',[-3],"011"," ",[-10,-40,50,50]);
yi=interpln([x;y],-4:45);}
plot2d((-4:45)',yi',[3],"000");
```

Lineáris interpoláció, Scilab beépített függvény

Az előző dián látott utasítássorozat eredménye az alábbi függvény.



Lineáris interpoláció, Scilab beépített függvény

n dimenziós lineáris interpoláció

[yp]=linear_interpn(xp1,xp2,...,xpn, x1, ..., xn, v [,out_mode]

Paraméterek:

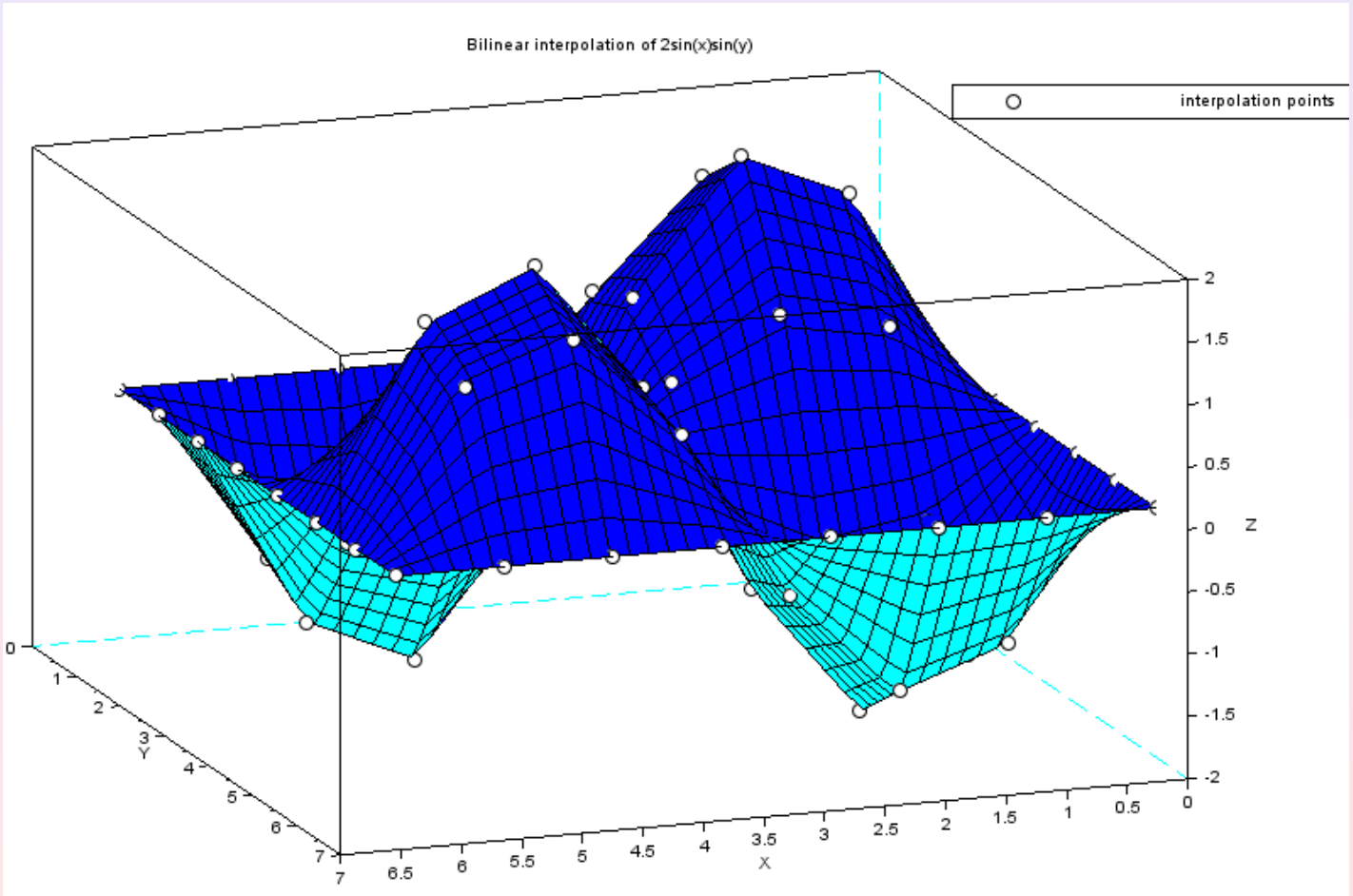
- xp1, xp2, . . . , xpn - megegyező méretű vektorok vagy mátrixok
- x1,. . . ,xn - sorozatvektor, az interpolációs háló számolásához
- v - vektor(ha n=1) / mátrix(ha n=2) vagy hipermátrix
- out_mode(opcionális) - ábrázolás módja

Lineáris interpoláció

Az alábbi példa mutatja a függvény működését

```
x = linspace(0,2*\%pi,8); y = x; //vektor készítés
z = 2*sin(x')*sin(y); // a függvény
xx = linspace(0,2*\%pi, 40);// hálóhoz vektor
[xp,yp] = ndgrid(xx,xx);// háló készítése
zp = linear_interpn(xp,yp, x, y, z);
                //interpolációs pontok
plot3d(xx, xx, zp, flag=[]2 6 4[]) // 3D ábra
[xg,yg] = ndgrid(x,x);
param3d1(xg,yg, list(z,-9*ones(1,n)), flag=[0 0]
show_window()
```

Lineáris interpoláció, Az előző példa eredménye



Lagrange interpoláció

Lagrange-féle interpolációs feladat

Keresünk egy olyan L_n legfeljebb n-edfokú polinomot, amelyre

$$L_n(x_i) = y_i, \qquad i = 0, 1, ..., n$$

Tétel

A Lagrange-féle interpolációs feladatnak létezik egyértelmű megoldása, amely az

$$L_n(x) = \sum_{k=0}^{n} y_k \frac{(x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0)(x_k - x_1) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)}$$

alakban adható meg.

Lagrange interpoláció

Tekintsük a következő alappontokat és a hozzá tartozó függvényértékeket:

x_i	-1	1	2	3
y_i	-2	0	-2	2

Az adatokhoz tartozó Lagrange-féle interpolációs polinom:

$$L_3(x) = -2\frac{(x-1)(x-2)(x-3)}{(-1-1)(-1-2)(-1-3)} - 2\frac{(x+1)(x-1)(x-3)}{(2+1)(2-1)(2-3)} +$$
$$+ 2\frac{(x+1)(x-1)(x-2)}{(3+1)(3-1)(3-2)} = x^3 - 3x^2 + 2$$

Lagrange interpoláció

A Lagrange interpoláció során a polinom együtthatóit egy másik módon is kiszámolhatjuk.
Keressük a Lagrange interpolációs polinomot a következő alakban:

$$L_n(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_n x^n$$

A polinom definíciójából adódóan az alábbi egyenleteknek teljesülniük kell:

$$\begin{aligned} c_0 + c_1 x_0 + c_2 x_0^2 + \ldots + c_n x_0^n &= y_0 \\ c_0 + c_1 x_1 + c_2 x_1^2 + \ldots + c_n x_1^n &= y_1 \\ &\vdots \\ c_0 + c_1 x_n + c_2 x_n^2 + \ldots + c_n x_n^n &= y_n \end{aligned}$$

Vandermonde mátrix

Az előző egyenletrendszert mátrix alakban felírva az alábbi egyenlethez jutunk:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Az interpolációs polinom együtthatóit ennek az egyenletnek a megoldásával kapjuk. A megoldáshoz használhatjuk bármelyik eliminációs algoritmust. A Scilab eljárás megoldásában beépített utasítást alkalmazunk.

Lagrange interpoláció

```
function LagrIntV(x,y)
if length(x)~=length(y)
printf('Rossz vektorokat adott meg!')
abort
end
n=length(x);
for i=1:n
for j=1:n
V(i,j)=x(i)^(j-1);
end
end
if det(V)==0
disp('Két azonos x értéket adott meg!')
abort
```

Lagrange interpoláció

```
else
disp('A Vandermonde mátrix:')
disp(V)
disp('A Vandermonde mátrix determinánsa:')
disp(det(V))
a=V\y;
disp('A polinom együtthatói:')
disp(a)
end
endfunction
```

Lagrange interpoláció

Oldjuk meg a fejezet elején megadott - és megoldott - feladatot.

Scilab 5.4.1 Console

```
-->x=[-1;1;2;3];

-->y=[-2;0;-2;2];

-->LagrIntV(x,y)

 A Vandermonde mátrix:

    1.   - 1.     1.   - 1.
    1.     1.     1.     1.
    1.     2.     4.     8.
    1.     3.     9.     27.

 A Vandermonde mátrix determinánsa:

    48.

 A polinom együtthatói:

    2.
    0.
  - 3.
```

Lagrange interpoláció

Az eljárás végén megkaptuk a Lagrange interpolációs polinom együtthatóit. A kapott megoldás az együtthatókat $c_0, c_1, \ldots c_n$ sorrendben adja meg. A keresett polinom a következő lesz:

$$L_3(x) = 2 + 0x + (-3)x^2 + x^3$$

Osztott differenciák

Adott egy $f : [a, b] \to \mathbb{R}$ függvény és $x_i \in [a, b] (i = 0, ..., n)$ páronként különböző alappontok. Az f függvény x_0 pontbeli *nulladrendű osztott differenciáján* az $f[x_i] \equiv f(x_i)$ számot értjük.

Az f függvény x_0, x_1 pontjaira felírt *elsőrendű osztott differencia*:

$$f[x_0, x_1] \equiv \frac{f[x_1] - f[x_0]}{x_1 - x_0}$$

általánosan, az f függvény $x_0, x_1, ..., x_n$ pontjaira felírt *n-edrendű osztott differencia*:

$$f[x_0, x_1, ..., x_n] \equiv \frac{f[x_1, x_2, ..., x_n] - f[x_0, x_1, ..., x_{n-1}]}{x_n - x_0}$$

Newton-polinom

A Lagrange-féle interpolációs polinomot megadhatjuk osztott differenciák segítségével a következő képlettel is:

$$L_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots$$
$$+ f[x_0, x_1, ..., x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1})$$

A fenti formulával definiált alakot nevezzük a Lagrange-féle interpolációs polinom *Newton-féle alakjának*.

Példa

Legyenek adottak az alábbi pontok:

$$(0;1),\ (2;5),\ (4;17)$$

Osztott differenciák segítségével adjuk meg a Lagrenge-féle interpolációs polinom Newton-féle alakját!

$x_0 = 0 \quad f[x_0] = 1$

$x_1 = 2 \quad f[x_1] = 5 \quad f[x_0, x_1] = \frac{5-1}{2-0} = 2$

$x_1 = 4 \quad f[x_2] = 17 \quad f[x_1, x_2] = \frac{17-5}{4-2} = 6 \quad f[x_0, x_1, x_2] = \frac{6-2}{4-0} = 1$

Ebből következik:

$$\begin{aligned} L_2(x) &= f[x_0] + f[x_0, x_1]x + f[x_0, x_1, x_2]x(x-2) \\ &= 1 + 2x + x(x-2) \\ &= 1 + x^2 \end{aligned}$$

Newton-féle alak pszeudokód

$(n+1)$ = alappontok száma, (x_i) = alappontok, (y_i) = fgv. értékek

Input: $x_i, (i = 0, \ldots, n, \quad y_i, (i = 0, \ldots, n)$
for $i = 0$ to n **do**
  $a_i \longleftarrow y_i$
end for
for $j = 1$ to n **do**
  for $i = n$ to j **do**
    $a_i \longleftarrow (a_i - a_{i-1}/(x_i - x_{i-j})$
  end for
end for
Output: $a_0, a_1, \ldots, a_n$

Newton-féle alak pszeudokód

A Newton-polinom kiértékelése
$(n+1)$ = alappontok száma, (x_i) = alappontok, (y_i) = fgv. értékek

Input: x_i, $(i = 0, \ldots, n)$, y_i, $(i = 0, \ldots, n)$, x
$y \longleftarrow a_n$
for $i = n-1$ to 0 **do**
$y \longleftarrow y(x - x_i) + a_i$
end for
Output: y

Interpoláció Newton-polinommal

```
function a=NewtIntp(x,y)
if length(x)~=length(y)
printf('Rossz vektorokat adott meg!')
abort
end
n=length(x);
for i = 1 : n
a(i) = y(i)
end
for j=2:n
for i=n:-1:j
a(i)=(a(i)-a(i-1))/(x(i)-x(i-j+1));
end
end
disp('A Newton-polinom együtthatói: ')
disp(a)
endfunction
```

Newton-polinom kiértékelése

```
function NewtIntk(x,a,t)
n=length(x);
sa=length(a);
if sa ~= n
    printf('Rossz dimenziójú input adatok!')
abort
end
y=a(n);
for i=n-1:-1:1
    y=y*(t-x(i))+a(i);
end
printf('A(z) %d helyen vett érték: %1.14f\n',t,y)
endfunction
```

Interpoláció Newton-polinommal

Az előző függvény használata

Scilab 5.4.1 Console

```
-->x=[0;2;4];

-->y=[1;5;17];

-->NewtIntp(x,y)

 A Newton-polinom együtthatói:

    1.
    2.
    1.

-->
```

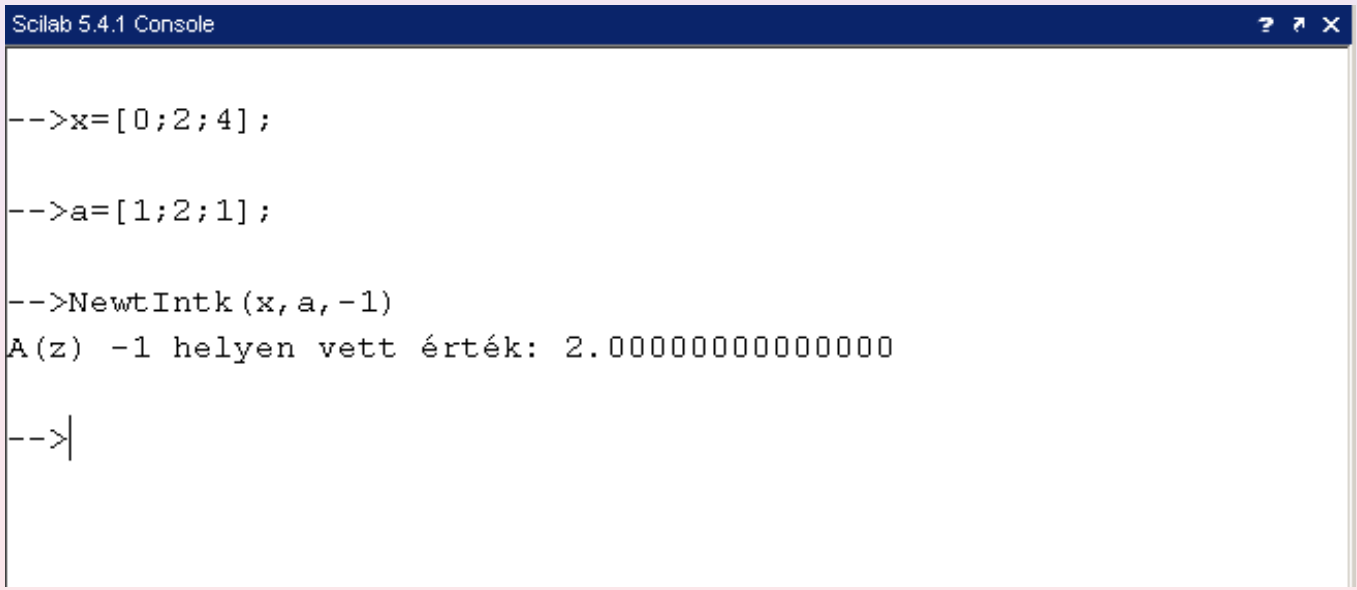
Lagrange interpoláció

Az eljárás végén megkaptuk a Lagrange interpolációs polinom Newton-féle alakjának együtthatóit. A keresett polinom a következő lesz:

$$L_2(x) = 1 + 2x + x(x-2)$$

Newton-polinom kiértékelése

Az előző feladatban megkapott polinom kiértékelése az $x = 1$ helyen a következőképpen történik.



Hermite-interpoláció

A Hermite-féle interpolációs feladatban az $y_i = f(x_i)$ függvényértékek interpolálációján túl, az $y_i' \equiv f'(x_i)$ deriválté értékeket is szeretnénk interpolálni. Keresünk tehát egy olyan $g(x) = c_0 + c_1 x + \cdots + c_m x^m$ polinomot, amelyre

$$g(x_i) = y_i, \qquad g'(x_i) = y_i', \qquad i = 0, 1..., n$$

teljesül. A keresett polinom grafikonja a *megadott irányokban* megy át az adott (x_i, y_i) pontokon, azaz az érintőjének iránytangense megegyezik az y_i' értékekkel.

Hermite-interpoláció

A feladat megoldásához magasabbrendű, speciális osztott differenciákra van szükség. Legyen:

$$f[x_i, x_i] = y_i'$$

Ezek után a rekurzív definíció:

$$f[x_0, x_0, x_1, x_1, \ldots, x_n, x_n] = \frac{f[x_0, x_1, x_1, \ldots, x_n, x_n] - f[x_0, x_0, x_1, x_1, \ldots, x_n]}{x_n - x_0}$$

Hermite-interpoláció

Példa

x_i	-1	1	2
y_i	-1	1	29
y_i'	-5	7	61

-1	**-1**					
-1	-1	**-5**				
1	1	1	**3**			
1	1	7	3	**0**		
2	29	28	21	6	**2**	
2	29	61	33	12	2	**0**

$$H_5(x) = \textbf{-1-5}(x+1)+ \\ +\textbf{3}(x+1)^2+\textbf{2}(x+1)^2(x-1)^2 \\ = 2x^4 - x^2 + x - 1$$

Hermite-interpoláció

```
function Hermint(x,y,yd)
n=length(x);
foszt=zeros(2*n);
for i=1:n
  xx(2*i-1)=x(i);
  xx(2*i)=x(i);
  foszt(2*i-1,1)=y(i);
  foszt(2*i,1)=y(i);
  foszt(2*i,2)=yd(i);
end
for i=2:n
  foszt(2*i-1,2)=(foszt(2*i-1,1)-foszt(2*i-2,1))/
                 (xx(2*i-1)-xx(2*i-2));
end
```

Hermite-interpoláció

```
for j=3:2*n
  for i=j:2*n
    foszt(i,j)=(foszt(i,j-1)-foszt(i-1,j-1))/
                              (xx(i)-xx(i-j+1));
  end
end
printf('A megoldás:')
disp(foszt)
endfunction
```

Hermite-interpoláció

Az Hermite-interpolácó feladat megoldása Scilab segítségével.

Scilab 5.4.1 Console

```
-->x=[-1,1,2];

-->y=[-1,1,29];

-->yd=[-5,7,61];

-->Hermint(x,y,yd)
A megoldás:
  - 1.      0.      0.      0.      0.      0.
  - 1.    - 5.      0.      0.      0.      0.
    1.      1.      3.      0.      0.      0.
    1.      7.      3.      0.      0.      0.
    29.     28.     21.     6.      2.      0.
    29.     61.     33.     12.     2.      0.

-->
```

Spline interpoláció

Legyen $a = x_0 < x_1 < ... < x_n = b$ az $[a, b]$ intervallumnak egy felosztása. Az $S : [a, b] \to \mathbb{R}$ folytonos függvényt az x_i osztópontokhoz tartozó *k-adrendű spline* függvénynek nevezzük, ha $S \in C^{k-1}(a, b)$, és S megszorítása minden $[x_i, x_{i+1}]$ intervallumra egy k-adrendű polinom.

Az első-, másod- illetve harmadrendű **spline függvényeket** *lineáris, kvadratikus, ill. kubikus spline függvényeknek* nevezzük.

Spline interpoláció

A Scilab két beépített függvény segítségével képes kubikus spline interpolációt számolni:

Függvényhívás

[d]=splin(x, y, [,spline_type [, der]])

Paraméterek:

- x - szigorúan növekvő vektor (legalább 2 hosszú)
- y - vektor, hasonló x-hez
- spline_type(opcionális) - kívánt spline típusa
- der(opcionális) - 2 hosszú vektor, a végpontok deriváltjai

Spline interpoláció

Kétváltozós függvények interpolálására használható a következő függvény:

Scilab függvényhívás

C=splin2d(x, y, z, [,spline_type])

Paraméterek:

- x,y - szigorúan növekvő sorvektorok (legalább 2 komponens)
- z - nx*ny méretűmátrix, ahol nx = x hossza, és ny = y hossza
- spline_type(opcionális) - kívánt kubikus spline típusa

Scilab programozás és a numerikus algoritmusok Scilabban
14. előadás
Szélsőértékszámítás

Dr. Mihálykó Csaba - Pozsgai Tamás

2014. május 4.

Tartalomjegyzék

1. Szélsőértékszámítás
2. Aranymetszés
   - Definíció
   - Scilab megvalósítás
3. Szimplex módszer
   - Definíció
   - Scilab megvalósítás
4. Nelder-Mead-módszer
5. Gradiens módszer
   - Definíció
   - Scilab megvalósítás

Általános feladat

Egy- és többváltozós függvények lokális szélsőértékei keresésével foglalkozunk.
Elengendő csak minimumkeresési módszereket vizsgálni, mivel egy $f(x)$ függvény ott veszi fel a maximumát, ahol a $-f(x)$ függvény a minimumát, így a maximum keresés mindig visszavezethető minimum keresésre.

Szélsőértékszámítás

Algoritmusoknak **három** nagy csoportja közül az első kettővel foglalkozunk:

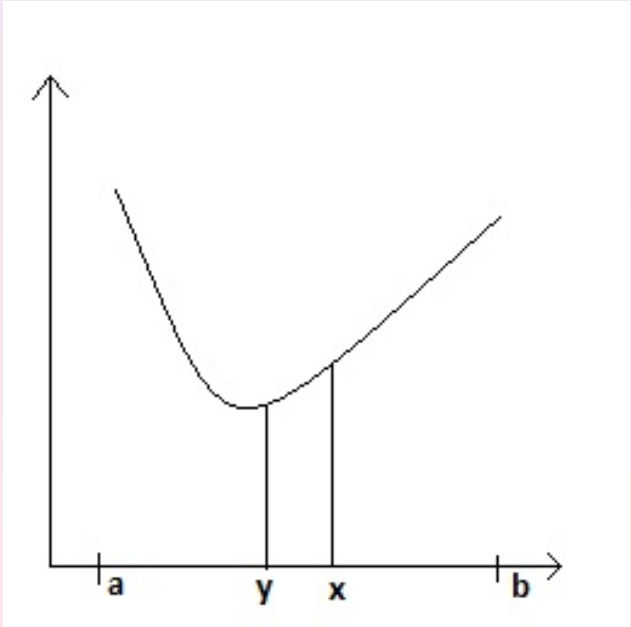
- deriváltat nem használó (aranymetszés, szimplex és a Nelder-Mead-módszer)
- csak első deriváltat használó (gradiens módszer)
- első és második deriváltat is igénylő módszerekkel (Newton-módszer)

Aranymetszés szerinti keresés módszere

Unimodális függvény:
Legyen $f : [a, b] \to \mathbb{R}$ folytonos, és feltesszük, hogy f **unimodális**, azaz f-nek egyértelmű lokális minimuma van $[a, b]$ intervallumon.

Aranymetszés szerinti keresés módszere

Az aranymetszés szerinti keresés módszerénél, az intervallumfelezés módszeréhez hasonlóan, egyre szűkebb intervallumokra határoljuk be a függvény minimumhelyét:
Legyen $a < y < x < b$. Ha $f(x) > f(y)$, akkor $p \in [a, x]$, ellenkező esetben $p \in [y, b]$ teljesül.



Aranymetszés szerinti keresés módszere

Az x és y pontokat úgy választjuk, hogy az $[a, x]$ és $[y, b]$ intervallum hossza azonos legyen: $x - a = b - y = r(b - a)$ valamely $0 < r < 1$-re.
Ekkor

$$x = a + r(b - a)$$

$$y = a + (1 - r)(b - a)$$

alakú.

Aranymetszés szerinti keresés módszere

Az $x > y$ feltételből kapjuk, hogy $0,5 < r < 1$ kell legyen.
Jelölje $[a', b']$ a következő intervallumot. Válasszuk az új osztópontokat, x'-t és y'-t az előző számítási mód szerint, és $f(x')$ és $f(y')$ összehasonlításával határozzuk meg a következő intervallumot.
Az aranymetszés szerinti keresés módszere úgy választja meg r-t, hogy az új x', y' osztópontok közül az egyik egyezzen meg egy előző osztóponttal, azért hogy az első kivételével minden lépésben csak egy új függvényértéket kelljen kiértékelni.
Ekkor

$$r^2 + r - 1 = 0$$

Aranymetszés szerinti keresés módszere

Az előző egyenlet pozitív megoldása $r = (\sqrt{5} - 1)/2 \approx 0.61834$
Ez az aranymetszés arányossági tényezője: r kielégíti az

$$\frac{r}{1-r} = \frac{1}{r}$$

egyenletet.

Aranymetszés szerinti keresés pszeudokód 1.

Input: $f(x)$, $[a, b]$, ε
$r = (\sqrt{5} - 1)/2$
$x \leftarrow a + r(b - a)$
$y \leftarrow a + (1 - r)(b - a)$
$fx \leftarrow f(x)$
$fy \leftarrow f(y)$

Aranymetszés szerinti keresés pszeudokód 2.

while $(b-a) > \varepsilon$ **do**
  if $fx > fy$ **then**
    $b \longleftarrow x$
    $x \longleftarrow y$
    $fx \longleftarrow fy$
    $y \longleftarrow a + (1-r)(b-a)$
    $fy \longleftarrow f(y)$
  else
    $a \longleftarrow y$
    $y \longleftarrow x$
    $fy \longleftarrow fx$
    $x \longleftarrow a + r(b-a)$
    $fx \longleftarrow f(x)$
  end if
end while
Output: $((a+b)/2)$

Scilab forráskód

```
function aranymetszes(f,a,b,t)
r=(sqrt(5.)-1)/2;
x=a+r*(b-a);
y=a+(1-r)*(b-a);
fx=f(x);
fy=f(y);
l=0;
while b-a>t,
    if fx>fy,
        b=x;
        x=y;
        fx=fy;
        y=a+(1-r)*(b-a);
        fy=f(y);
```

Aranymetszés szerinti keresés módszere

```
    else
        a=y;
        y=x;
        fy=fx;
        x=a+r*(b-a);
        fx=f(x);
    end
    l=l+1;
    printf('%d. lépésben az új intervallum:
                        [%1.6f, %1.6f]\n',l,a,b)
end
x=(a+b)/2;
y=f(x);
printf('Minimumhely = %1.10f\n',x);
printf('Minimális függvényérték = %1.10f\n',y);
printf('Lépésszám = %d\n',l);
endfunction
```

Példa

Keresse meg az alábbi függvény minimumhelyét $0,001$ pontossággal. Kezdeti intervallumként a $[0;4]$ intervallumot használja.

$$f(x) = x\left(1 - \frac{3x}{10}\right)$$

A megoldáshoz először hozzuk létre a függvényt Scilabban.

```
function y=fgv(x)
  y=-x*(1-0.3*x);
endfunction
```

Példa

Lefuttatva a programot a következő megoldást kapjuk:

Scilab 5.4.1 Console

```
-->aranymetszes(fgv,0,4,0.001)
1. lépésben az új intervallum: [0.000000, 2.472136]
2. lépésben az új intervallum: [0.944272, 2.472136]
3. lépésben az új intervallum: [0.944272, 1.888544]
4. lépésben az új intervallum: [1.304952, 1.888544]
5. lépésben az új intervallum: [1.527864, 1.888544]
6. lépésben az új intervallum: [1.527864, 1.750776]
7. lépésben az új intervallum: [1.613009, 1.750776]
8. lépésben az új intervallum: [1.613009, 1.698154]
9. lépésben az új intervallum: [1.645531, 1.698154]
10. lépésben az új intervallum: [1.645531, 1.678054]
11. lépésben az új intervallum: [1.657954, 1.678054]
12. lépésben az új intervallum: [1.657954, 1.670376]
13. lépésben az új intervallum: [1.662699, 1.670376]
14. lépésben az új intervallum: [1.665631, 1.670376]
15. lépésben az új intervallum: [1.665631, 1.668564]
16. lépésben az új intervallum: [1.665631, 1.667444]
17. lépésben az új intervallum: [1.666324, 1.667444]
18. lépésben az új intervallum: [1.666324, 1.667016]
Minimumhely = 1.6666698816
Minimális függvényérték = -0.8333333333
Lépésszám = 18
```

Szimplex módszer

Egy n-dimenziós **szimplexen** olyan $n+1$ darab n-dimenziós vektor konvex burkát, azaz az

$$\{\alpha_0 \mathbf{x}^{(0)} + ... + \alpha_n \mathbf{x}^{(n)} : 0 \leq \alpha_i \leq 1,\ \alpha_0 + ... + \alpha_n \leq 1\}$$

halmazt értjük, ahol az $\mathbf{x}_1 - \mathbf{x}_0, \mathbf{x}_2 - \mathbf{x}_0, ..., \mathbf{x}_n - \mathbf{x}_0$ vektorok lineárisan függetlenek. Ekkor az $\mathbf{x}^{(0)}, ..., \mathbf{x}^{(n)}$ vektorokat a **szimplex csúcspontjainak** hívjuk.

Szimplex módszer

- egydimenziós szimplexek a **szakaszok**
- kétdimenziós szimplexek a **háromszögek**
- háromdimenziós szimplexek pedig a **tetraéderek**

A szimplex módszert n-változós függvények minimumhely keresésére használjuk. Vegyünk fel kiindulásként egy n-dimenziós szimplexet. Keressük meg, hogy melyik a "legrosszabb" csúcspont, azaz melyik csúcspontban veszi fel at f függvény a legnagyobb értéket.

Szimplex módszer

Legyen ez például az $\mathbf{x}^{(j)}$ pont. Ekkor a szimplex legrosszabb pontját tükrözzük az $\mathbf{x}^{(j)}$ ponttal szemben fekvő oldal középpontjára, azaz a többi csomópont

$$\mathbf{x}_c \equiv \frac{1}{n} \sum_{\substack{i=0 \\ i \neq j}}^{n} \mathbf{x}^{(i)}$$

súlypontjára. A tükrözött pont koordinátáit az

$$\mathbf{x}_r = 2\mathbf{x}_c - \mathbf{x}^{(j)}$$

képlettel számíthatjuk ki.

Szimplex módszer

Ha $f(\mathbf{x}_r)$ nem kisebb, mint az előző lépésbeli legnagyobb függvényérték, $f(\mathbf{x}^{(j)})$, akkor a tükrözést nem fogadjuk el, hanem ahelyett a legjobb csúcspontból fele akkorára zsugorítjuk a szimplexet:

Legyen $\mathbf{x}^{(k)}$ a legjobb csúcspontja a szimplexnek, azaz ebben a legkisebb a függvényérték. Ekkor a többi csúcspontot az

$$\mathbf{x}^{(i)} \leftarrow \mathbf{x}^{(k)} + \frac{1}{2}(\mathbf{x}^{(i)} - \mathbf{x}^{(k)})$$

i = 0;1;. . .;k - 1; k + 1;. . .;n

képlettel számoljuk újra. Ezután a kapott (vagy tükrözött vagy zsugorított) szimplexszel megismételjük az eljárást.

Szimplex módszer

Az eljárás megvalósításánál a következő paraméterezést használjuk:

- fgv - a függvény, aminek a minimumhelyét keressük
- A - a kezdőpontokbók képzett mátrix
- m - lépésszám

Visszatérési értéke az eljárásnak a keresett minimumhely, illetve a minimum érték lesz.

Szimplex módszer

```
function szimplexmodszer(f,A,m)
sA=size(A);
n=sA(2);
if sA(1) ~= n+1,
    printf('A kezdeti szimplex (n+1),
        az x n dimenziós kell legyen!');
    abort
end
printf('0. lépés:\n');
disp(A)
for k=1:m,
    for i=1:n+1,
      fv(i)=f(A(i,:));
    end
    [fvs,ind]=gsort(fv,'r','i');
    A=A(ind,:);
    xc=A(1,:);
```

Szimplex módszer

```
    for i=2:n,
        xc=xc+A(i,:);
    end
    xc=xc/n;
    xr=2*xc-A(n+1,:);
    if f(xr)<f(A(n+1,:)),
        A(n+1,:)=xr;
    else
        for i=2:n+1,
            A(i,:)=A(1,:)+0.5*(A(i,:)-A(1,:));
        end
    end
    printf('%d. lépés:\n',k);
    disp(A)
end
```

Szimplex módszer

```
minx=A(1,:);
for i=2:n+1,
    minx=minx+A(i,:);
end
minx=minx/(n+1);
miny=f(minx);
printf('Minimális függvényérték = %1.10f\n',miny);
printf('Minimumhely = ');
disp(minx)
endfunction
```

Példa

Keresse meg az alábbi függvény minimumhelyét. A $[-1;6]$, $[0;6]$, $[-0,5;5]$ kezdeti háromszöget használja.

$$f(x,y) = 2\left(\frac{x^2}{2} - y^2\right) + (x-1)^2$$

A megoldáshoz először hozzuk létre a függvényt Scilabban.

```
function y=fgv2(x)
y=2*(0.5*x(1)^2-x(2))^2+(x(1)-1)^2;
endfunction
```

Példa

Lefuttatva a programot a következő futási eredményt kapjuk:

Scilab 5.4.1 Console

```
-->A=[-1,6;0,6;-0.5,5];

-->szimplexmodszer(fgv2,A,30)
0. lépés:

  - 1.      6.
    0.      6.
  - 0.5     5.
1. lépés:

  - 0.5     5.
  - 1.      6.
  - 1.5     5.
2. lépés:

  - 1.5     5.
  - 0.5     5.
  - 1.      4.
```

Példa

Lefuttatva a programot a következő végeredményt kapjuk:

```
29. lépés:

    0.9375      0.375
    0.875       0.375
    0.90625     0.4375
30. lépés:

    0.90625     0.4375
    0.9375      0.375
    0.96875     0.4375
Minimális függvényérték = 0.0049446954
Minimumhely =
    0.9375     0.4166667
```

Nelder-Mead-módszer

A szimplex módszernek egy módosított változata a Nelder-Mead-módszer. Ennél a módszernél a szimplexet tükrözzük, illetve megnyújtjuk vagy zsugorítjuk aszerint, hogy milyen értéket vesz fel a függvény a csúcspontokban. Feltesszük, hogy minden egyes lépésben a csúcspontokat úgy indexezzük, hogy a függvényértékek növekvő sorrendben legyenek, azaz

$$f(\mathbf{x}^{(0)}) \leq f(\mathbf{x}^{(1)}) \leq \ldots \leq f(\mathbf{x}^{(n)}).$$

Ekkor $\mathbf{x}^{(n)}$ a legrosszabb csúcspont, ezt tükrözzük a szemben fekvő oldal

$$x_c = \frac{1}{n} \sum_{i=0}^{n-1} x^{(i)}$$

súlypontjára.

Nelder-Mead-módszer

Legyen a tükrözött pont $\mathbf{x}_r = 2\mathbf{x}_c - \mathbf{x}^{(n)}$. Vizsgáljuk meg, hogy milyen értéket vesz fel az f függvény $\mathbf{x}_r$-ben. **Három** esetet különböztetünk meg:

1. $f\left(\mathbf{x}^{(0)}\right) < f(\mathbf{x}_r) < f\left(\mathbf{x}^{(n-1)}\right)$
2. $f\left(\mathbf{x}_r\right) \leq f\left(\mathbf{x}^{(0)}\right)$, azaz $\mathbf{x}_r$ lenne az új legjobb pont,
3. $f\left(\mathbf{x}_r\right) \geq f\left(\mathbf{x}^{(n-1)}\right)$, azaz $\mathbf{x}_r$ lenne az új legrosszabb pont.

Nelder-Mead-módszer

Az 1. eset:

$\mathbf{x}^{(n)}$-t $\mathbf{x}_r$-re kicseréljük (elfogadtuk a tükrözést), és folytatjuk az iterációt.

A 2. eset:

Először megpróbáljuk az $\mathbf{x}_r$ irányban megnyújtani egy kicsit a szimplexet, hátha még jobb pontot kapunk. Legyen

$$\mathbf{x}_e \equiv \mathbf{x}_c + \alpha(\mathbf{x}_r - \mathbf{x}_c)$$

ahol, $\alpha > 1$ egy rögzített szám (egy paraméter a módszerben). Ha ekkor $f(\mathbf{x}_e) < f(\mathbf{x}^{(0)})$ teljesül, akkor a megnyújtást sikeresnek ítéljük, és $\mathbf{x}^{(n)}$-t $\mathbf{x}_e$-re cseréljük ki. Ellenkező esetben viszont $\mathbf{x}^{(n)}$-t $\mathbf{x}_r$-re cseréljük ki, azaz tükrözünk, de nem nyújtjuk meg a szimplexet.

Nelder-Mead-módszer

A 3. eset:

Azt gondoljuk, hogy túl messze tükröztük $\mathbf{x}^{(n)}$-t, így megpróbáljuk zsugorítani a szimplexet. Legyen

$$\mathbf{x}_z \equiv \begin{cases} \mathbf{x}_c - \beta(\mathbf{x}_r - x_c) & haf(\mathbf{x}^{(n)}) < f(\mathbf{x}_r), \\ \mathbf{x}_c + \beta(\mathbf{x}_r - x_c) & haf(\mathbf{x}^{(n)}) \geq f(\mathbf{x}_r) \end{cases}$$

ahol $0 < \beta < 1$ egy újabb paraméter.
Ha $f(\mathbf{x}_z) < \min\{f(\mathbf{x}^{(n)}), f(\mathbf{x}_r)\}$, akkor $\mathbf{x}^{(n)}$-t $\mathbf{x}_z$-vel cseréljük fel.
Ellenkező esetben viszont a szimplexet a legjobb pontjából, $\mathbf{x}^{(0)}$-ból a felére zsugorítjuk össze:

$$\mathbf{x}^{(i)} \leftarrow \mathbf{x}^{(0)} + \tfrac{1}{2}(\mathbf{x}^{(i)} - \mathbf{x}^{(0)}), \quad i = 1; \ldots; n$$

Gradiens módszer

Tekintsünk egy $f : \mathbb{R}^n \to \mathbb{R}$ függvényt

Analízisből ismert tétel szerint egy p pontban az f függvény a $-f1(\mathbf{p})$ irányban csökken lokálisan a leggyorsabban.

Tétel:

Legyen $f \in C^1$. Ekkor a

$$\lim_{t \to 0+} \frac{f(\mathbf{p} + t\mathbf{u}) - f(\mathbf{p})}{t} \qquad \|\mathbf{u}\| = 1$$

iránymenti deriváltak minimuma az $\mathbf{u} = -f(\mathbf{p}/\left\|f'(\mathbf{p})\right\|_2$ irányban van.

Gradiens módszer

A gradiens módszer szerint egy $\mathbf{p}^{(0)}$ kezdeti pontból a negatív gradiensvektor irányában kell elmozdulni. Szokás az előbbiek miatt ezt a **legmeredekedebb lejtő módszerének** is nevezni.
A módszer általános képlete ezért:

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} - \alpha_k f'(\mathbf{p}^{(k)})$$

ahol α_k a lépésközt meghatározó szorzótényező.

Gradiens módszer

Legyen h rögzített, és használjuk az

$$\alpha_k = h / \left\| f'(\mathbf{p}^{(k)}) \right\|_2$$

számot.
Ekkor az egyes pontok közötti távolság konstans h lesz.
Természetesen ekkor a minimumhelyet h-nál pontosabban nem tudjuk megadni. Leállási feltételként az ismétlésszámot használjuk.

Optimális gradiens módszer

Egy másik változatban úgy választjuk meg a lépésközt, hogy

$$\phi_k(\alpha_k) = \min_{t \in \mathbb{R}} \phi_k(t)$$

legyen, ahol

$$\phi_k(t) \equiv f\left(\mathbf{p}^{(k)} - tf'(\mathbf{p}^{(k)})\right).$$

Optimális gradiens módszer

Ekkor minden egyes lépésben a gradiensvektor által meghatározott egyenes mentén egy egyváltozós függvényt kell minimalizálni. Ez utóbbi módon választott lépésközt használó gradiens módszert **optimális gradiens módszernek** hívjuk.
Az optimális gradiens módszernél a gradiensvektorral párhuzamos egyenes mentén egy olyan pontig lépünk, ahol az egyenes érint egy szintvonalat.
Abból a pontból pedig a pontbeli gradiensvektorral párhuzamosan lépünk tovább.
Ebből következik, hogy az optimális gradiens módszernél az egymás utáni lépések irányai merőlegesek egymásra.

Gradiens módszer

Az ismertetett algoritmus az állandó lépésközű gradiens módszer.
Az eljárás megvalósításánál a következő paraméterezést használjuk:

- fgv - a függvény, aminek a minimumhelyét keressük
- dfgv - az fgv függvény deriváltja
- x - a kezdőpont
- h - lépésköz
- m - lépésszám

Visszatérési értéke az eljárásnak a keresett minimumhely, illetve a minimum érték lesz.

Gradiens módszer

```
function gradiensmodszer(f,df,x,h,m)
sx=size(x);
if sx(2)~=1
    printf('A kezdeti vektor nem oszlopvektor!')
    abort
end
tmp=df(x);
stmp=size(tmp);
if stmp(2)~=1,
    printf('A gradiensfüggvény nem oszlopvektort
                                    ad vissza!')
    abort
end
printf('0. lépés: %1.10f, %1.10f\n',x(1),x(2));
```

Gradiens módszer

```
for k=1:m,
    u=df(x);
    u=u/norm(u);
    x=x-h*u;
    printf('%d. lépés: %1.10f, %1.10f\n ',k,x(1),x(2))
end
minx=x;
miny=f(minx);
printf('Függvényérték = %1.10f\n',miny);
endfunction
```

Példa

Keresse meg az alábbi függvény minimumhelyét.
A $[-1; 1,25]$ kezdőpontot használja. A lépésköz $h = 0,1$ hosszú legyen.
Az algoritmus 40 lépés után álljon le.

$$f(x,y) = 2\left(\frac{x^2}{2} - y^2\right) + (x-1)^2$$

Példa

A megoldáshoz először hozzuk létre a függvényt Scilabban.

```
function y=fgv2(x)
y=2*(0.5*x(1)^2-x(2))^2+(x(1)-1)^2;
endfunction
```

Hozzuk létre a függvény deriváltfüggvényét is.

```
function y=dfgv2(x)
y(1)=4*(0.5*x(1)^2-x(2))*x(1)+2*(x(1)-1);
y(2)=-4*(0.5*x(1)^2-x(2));
endfunction
```

Példa

Lefuttatva a programot a következő futási eredményt kapjuk:

Scilab 5.4.1 Console

```
-->gradiensmodszer1(fgv2,dfgv2,[-1; 1.25],0.1,40);
0. lépés: -1.0000000000, 1.2500000000
1. lépés: -0.9683772234, 1.1551316702
 2. lépés: -0.9261555756, 1.0644821668
 3. lépés: -0.8754001137, 0.9783202084
 4. lépés: -0.8180456749, 0.8964027600
 5. lépés: -0.7555762311, 0.8183158303
 6. lépés: -0.6890221662, 0.7436796698
 7. lépés: -0.6190668512, 0.6722216129
 8. lépés: -0.5461550870, 0.6037827742
 9. lépés: -0.4705745142, 0.5383031819
 10. lépés: -0.3925105200, 0.4758050793
 11. lépés: -0.3120815062, 0.4163813994
 12. lépés: -0.2293607595, 0.3601917378
 13. lépés: -0.1443893711, 0.3074673304
 14. lépés: -0.0571832300, 0.2585276843
 15. lépés: 0.0322635342, 0.2138144159
 16. lépés: 0.1239757019, 0.1739536796
 17. lépés: 0.2179881935, 0.1398706717
 18. lépés: 0.3143120266, 0.1130058403
 19. lépés: 0.4128098470, 0.0957379688
 20. lépés: 0.5127474278, 0.0922052737
```

Példa

Lefuttatva a programot a következő végeredményt kapjuk:

```
21. lépés:  0.6112402062,  0.1095018809
22. lépés:  0.6997643013,  0.1560151540
23. lépés:  0.7701321641,  0.2270669935
24. lépés:  0.8363618101,  0.3019911817
25. lépés:  0.9022885249,  0.3771820607
26. lépés:  0.9613950083,  0.4578444659
27. lépés:  1.0576129899,  0.4850859788
28. lépés:  0.9753667590,  0.5419679400
29. lépés:  1.0511395740,  0.4767109059
30. lépés:  0.9699887992,  0.5351450717
31. lépés:  1.0468605335,  0.4711862063
32. lépés:  0.9664615980,  0.5306505752
33. lépés:  1.0440541552,  0.4675681286
34. lépés:  0.9641603451,  0.5277094548
35. lépés:  1.0422233951,  0.4652101730
36. lépés:  0.9626642598,  0.5257935369
37. lépés:  1.0410333198,  0.4636783984
38. lépés:  0.9616939231,  0.5245492464
39. lépés:  1.0402615257,  0.4626854297
40. lépés:  0.9610655564,  0.5237427671
Függvényérték =  0.0091838817
```