MÉRNÖKI PROGRAMOZÁS

OKTATÁSI SEGÉDLET

Írta:
DR. SZABÓ NORBERT PÉTER
Egyetemi tanársegéd

Miskolci Egyetem
Geofizikai Tanszék
Miskolc
2006.

TARTALOMJEGYZÉK

1. STRUKTURÁLT PROGRAMOZÁS MATLAB FEJLESZTŐI RENDSZER ALKALMAZÁSÁVAL

A MATLAB rendszer adatstruktúrájának alaptípusa *N*M* méretű mátrix (*N*: mátrix sorainak, *M*: mátrix oszlopainak a száma), melynek elemeit dupla pontossággal ábrázolt komplex számok képezik. Valós szám esetén a tárolás 8 byte-on történik (ha nincs képzetes része, akkor azt nem ábrázolja), komplex számoknál értelemszerűen 16 byte-on. Ezzel az adatszerkezettel számos műszaki és természettudományos probléma gyors implementációja válik lehetővé. A rendszert többek között e tulajdonság miatt is a gyártó MATHWORKS[1] cég „The language of technical computing"-nak nevezi.

A rendszer alapvetően két felhasználói felületből áll. Az egyik a COMMAND WINDOW (parancsablak), ahol a programokat futtathatjuk, számításokat végezhetünk és kommunikálhatunk a rendszerrel. A másik egy szövegszerkesztő (EDITOR), ahol a programkódot írhatjuk. Ezen kívül rendelkezik egy GUI (Graphical Unit Interface) felülettel, ahol objektum-orientált alkalmazásokat is létrehozhatunk. Rendelkezik továbbá a rendszer egy TOOLBOX könyvtárral, mely sokféle matematikai függvényt és algoritmust kínál speciális problémák megoldására. Néhány példa a MATLAB gazdag eszköztáraiból:

Communications Toolbox,
Control System Toolbox,
Database Toolbox,
Filter Design Toolbox,
Financial Derivatives Toolbox,
Genetic Algorithm Toolbox,
Instrument Control Toolbox,
Mapping Toolbox,
Model Predictive Control Toolbox,
Optimization Toolbox,
Statistics Toolbox,
System Identification Toolbox,
Virtual Reality Toolbox stb.

1.1 A MATLAB ALAPVETŐ PARANCSAI

A rendszer indítása után a COMMAND WINDOW jelentkezik be, ahol a >> (promt) után gépelhetjük a parancsokat. Elsőként a munkakönyvtárba való belépésről kell gondoskodnunk, mely a felhasználó által bármely periférián létrehozott könyvtár lehet. Ez alapesetben a MATLAB/WORK nevű könyvtár (ld. 1. ábra). Az aktuális könyvtár lekérdezése a **pwd** paranccsal történik, melynek neve az **ans** (answer) változóban tárolódik (ennek tartalma mindig az aktuális változó: komplex szám vagy mátrix, valamint string):

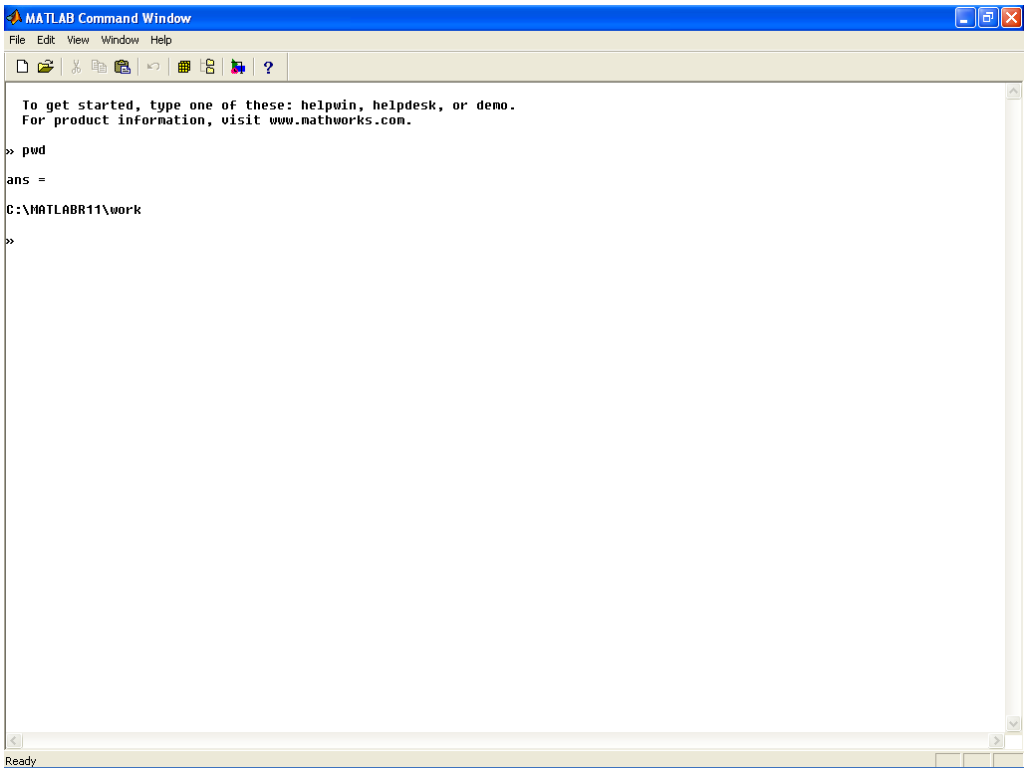
```
>> pwd
ans =
C:\MATLAB7\work.
```

[1]MATHWORKS cég internet címe: http://www.mathworks.com. Hasznos lehet továbbá az ingyenes filecserélő rendszer (File Exchange Centre), ahol számos MATLAB programot tölthetünk le és futathatunk (továbbfejleszthetünk) saját rendszerünk alatt. Ennek címe: http://www.mathworks.com/matlabcentral/fileexchange/loadCategory.do

A leggyakrabban alkalmazott könyvtárműveletek DOS-os hagyományokon alapulnak. A **cd** parancs könyvtárváltást, a **md** új könyvtár létrehozását jelenti. A következő négy parancs sorrendben: egy szinttel feljebb lép; az aktuális meghajtó könyvtárrendszerének gyökerébe lép; az aktuális helyen egy új *nev* nevű könyvtárat hoz létre; végül egy általunk megadott helyen hozza létre a *nev* nevű könyvtárat:

```
>> cd ..
>> cd \
>> !md nev
>> !md d:\nev
```

Az aktuális könyvtár tartalma az **ls** vagy **dir** parancsokkal kérhető le. Alapvető tulajdonsága a rendszernek, hogy a kis és nagybetűket külön változóként kezeli, ezért legyünk körültekintőek a könyvtárakra, valamint a változók azonosítóira történő hivatkozásoknál.



1. ábra: A MATLAB parancsablaka és munkakönyvtára

Segítséget a **help** paranccsal kérhetünk, ekkor megjelenik a „HELP topic” felirat után a leírásokat tartalmazó összes alkönyvtár listája, melyben a parancsok jelentései után tallózhatunk. Pl. az alap matematikai függvények könyvtára a help-ben: „matlab\elfun - Elementary math functions”-ként jelenik meg, ekkor a kulcsszavak részletes listáját így kérhetünk

```
>> help elfun
```

valamint egy abban található (jelen esetben a szinusz-hiperbolikusz) függvény leírását így kérhetjük

```
>> help sinh.
```

Egyszerűbb keresést a MATLAB főmenüjének HELP menüpontja alatt valósíthatunk meg, ahol HTML formátumú leírások, dokumentációk, példák és demok találhatók. Ha nincs ismeretünk a keresett függvény nevéről, akkor a **lookfor** parancs után megadhatunk olyan szavakat, amelyek kapcsolatban lehetnek a keresett funkcióval. Ennek hatására a rendszer az összes olyan függvényt felsorolja, melynek rövid leírásában szerepel az általunk megadott karaktersorozat. Pl., ha numerikus differenciálást szeretnénk végezni, de nem tudjuk a megfelelő függvény kulcsszavát, akkor az alábbiak szerint végezzünk keresést

```
>> lookfor differen
SETDIFF Set difference.
DIFF Difference and approximate derivative.
POLYDER Differentiate polynomial.
DDE23 Solve delay differential equations (DDEs) with constant delays.
DEVAL Evaluate the solution of a differential equation problem.
ODE113 Solve non-stiff differential equations, variable order method.
…
```

A kapott listából ekkor kiválaszthatjuk a nekünk megfelelőt, és a help paranccsal lekérdezhetjük a függvény részletes leírását. (A listázás és a programok futtatása megszakítható az univerzális **Ctrl+C** paranccsal.)

1.2 VÁLTOZÓK MEGADÁSA

A MATLAB hatékonyságát annak köszönheti, hogy alapvető adattípusa a mátrix, mellyel gyors számítások végezhetők. A mátrix megadása egyszerű értékadással történik, melyhez deklaráció nem szükséges. Ennek legegyszerűbb esete a skalár megadása, mely bármely valós vagy komplex szám lehet, pl.:

```
>> k=2-3.5*i
k =
  2.0000 - 3.5000i.
```

E speciális mátrix mérete a **size** nevű paranccsal kérhető le, ahol a sorok és az oszlopok száma az *ans* (1×2-es mátrix) változóban tárolódik:

```
>> size(k)
ans =
    1     1.
```

Megjegyezzük, hogy ha egy parancs után pontosvesszőt teszünk, akkor az utasítás végrehajtásának eredményét nem írja ki a rendszer. Viszont abban az esetben, amikor üres karakter vagy vessző van az utasítás után, az eredmény megjelenik a képernyőn. Vannak speciális számok a rendszerben, melyek számára bizonyos stringek lefoglalásra kerültek, ilyen pl. a π (*pi*) és a gépi epsilon (*eps*), ezeket célszerű kihagyni változók megadásakor. A MATLAB használja a *NaN* (Not a Number) és az *inf* értékeket, a nem számot ill. a végtelent eredményező műveletek kimenetelének megjelenítése céljából. Változók tartalmát a **clear** paranccsal, valamint az összes változót a **clear all** paranccsal törölhetjük a MATLAB munkaterületéről. A képenyő törlést (ami nem jelenti a változók törlését) a **clc** paranccsal végezhetjük.

A következő speciális mátrix a vektor, melynek elemei valós és komplex számok egyaránt lehetnek. Az alábbi példa a *vo* oszlopvektor egész számokkal történő megadását mutatja be:

```
>> vo=[-3;4;6;7;10]
vo =
   -3
    4
    6
    7
   10,
```

melynek a mérete

```
>> size(vo)
ans =
     5    1.
```

A *vs* sorvektor megadása a fentiektől csak kismértékben tér el, pl.:

```
>> vs=[1,2,-3,4,-5]
vs =
    1    2   -3    4   -5
>> size(vs)
ans =
    1    5.
```

A vektorok elemeire való hivatkozás a komponens sorszámának megadásával történik mind sor-, mind pedig oszlopvektorok esetén, pl. írassuk ki a *vo* vektor 4. elemét:

```
» vo(4)
ans =
     7.
```

A mátrix megadása többféleképpen történhet. A fentiekhez hasonlóan egész számokból álló mátrixot az elemek megfelelő felsorolásával adhatjuk meg, pl. az 5 sorral és 4 oszloppal rendelkező *M* mátrix esetén:

```
>> M=[11 22 33 43;54 64 76 85;91 10 11 12;13 14 15 16;17 18 19 20]
M =
   11   22   33   43
   54   64   76   85
   91   10   11   12
   13   14   15   16
   17   18   19   20,
```

melynek mérete

```
>> size(M)
ans =
     5     4.
```

A mátrix elemeire való hivatkozás a sor és az oszlopindexek megadásával történik, pl. az *M* mátrix 3. sorának 4. oszlopbeli eleme a következő módon kérdezhető le:

```
>> M(3,4)
ans =
    12.
```

A MATLAB indítása óta a felhasználó által megadott változókról a **whos** paranccsal gyors lista készíthető, mely tartalmazza a változó nevét, méretét, memóriafoglalását (egyenként és együtt) és a szám típusát. Az ebben a fejezetben eddig felhasznált változók listája:

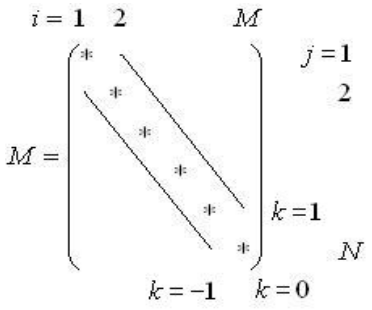
```
>> whos
  Name        Size            Bytes  Class
  M           5x4               160  double array
  ans         1x1                 8  double array
  k           1x1                16  double array (complex)
  vo          5x1                40  double array
  vs          1x5                40  double array
Grand total is 32 elements using 264 bytes.
```

Az aktuális idő (év, hónap, nap, óra, perc, másodperc) a **clock** paranccsal kérdezhető le, melynél az *ans* változó normálalakú 1×6-méretű mátrix.

1.3 MÁTRIXOK ÉS MÁTRIXMŰVELETEK

A számítási algoritmusok számára számos beépített mátrixot hívhatunk meg, mely gyorsítja a programírás folyamatát. Ilyen pl. az egységmátrix (**eye**), egyesmátrix (**ones**), nullamátrix (**zeros**), Hilbert-mátrix (**hilb**) és inverz Hilbert-mátrix (**invhilb**), melyek argumentuma a mátrix sorainak és oszlopainak számát tartalmazzák.

Algebrai számításokban gyakori a mátrix diagonálisainak felhasználása. Az elemek gyors lekérdezése céljából egy külön beépített függvényt használhatunk fel. Az 2. ábrán egy tetszőleges *M* mátrix k=0 főátlóját a csillaggal jelölt elemek képezik. Emellett léteznek mellékátlók is, pl. a k=-1. mellékátló a főátló alatt, a k=1. pedig a főátló felett közvetlen elhelyezkedő elemekből áll (ld. az ábrán fekete vonallal jelölt helyeken). Ez alapján a k-adik diagonálist azok az elemek alkotják, amelyre teljesül a $k = i - j$ összefüggés, ahol *j* a mátrixelem sor-, *i* pedig az oszlopindexe.



2. ábra. Mátrix fő- és mellékátlói

Definiáljunk a **rand** véletlenszám generátor segítségével egy 5×5 elemű véletlen mátrixot, melynek elemei egyenletes valószínűséggel kerüljenek kiválasztásra a [0,1] intervallumból:

```
>> M=rand(5,5)
M =
    0.9501    0.7621    0.6154    0.4057    0.0579
    0.2311    0.4565    0.7919    0.9355    0.3529
    0.6068    0.0185    0.9218    0.9169    0.8132
    0.4860    0.8214    0.7382    0.4103    0.0099
    0.8913    0.4447    0.1763    0.8936    0.1389.
```

Ebben az esetben az *M* mátrix fődiagonálisának elemeit a *d* oszlopvektorban a **diag** paranccsal hívhatjuk:

```
>> d=diag(M)
d =
    0.9501
    0.4565
    0.9218
    0.4103
    0.1389.
```

A mellékátlók elemeinek megadásához a *k* szám is szükséges (ld. 1. ábra), pl. a k=1-hez tartozó elemek a következőképpen hívhatók:

```
>> d1=diag(M,1)
d1 =
    0.7621
    0.7919
    0.9169
    0.0099.
```

Ha szeretnénk megnövelni egy mátrix méretét és azt új elemekkel feltölteni, akkor azt hivatkozással értékadó utasítás keretében kell megtenni. Pl. az *M* mátrixot bővítsük úgy ki, hogy egy új sora és egy új oszlopa legyen, és azok utolsó eleme legyen 1:

```
>> M(6,6)=1
M =
    0.9501    0.7621    0.6154    0.4057    0.0579         0
    0.2311    0.4565    0.7919    0.9355    0.3529         0
    0.6068    0.0185    0.9218    0.9169    0.8132         0
    0.4860    0.8214    0.7382    0.4103    0.0099         0
    0.8913    0.4447    0.1763    0.8936    0.1389         0
         0         0         0         0         0    1.0000.
```

Látható, hogy ebben az esetben arra kell vigyázni, hogy az új elem pozícióját kivéve a bővítmény teljes oszlopát és sorát a rendszer hivatkozás hiánya esetén zérus elemekkel tölti fel. A mátrix bővítése során alkalmazhatjuk az egyszerű megadást is, pl. az 1.2 fejezetben bevezetett *vs* sor-, és *vo* oszlopvektort egészítsük ki két-két új elemmel:

```
>> vs=[vs,-7,2]
vs =
     1     2    -3     4    -5    -7     2
>> vo=[vo;-1;1]
vo =
    -3
     4
     6
     7
    10
    -1
     1.
```

Ezek után tekintsük az alapvető mátrix műveleteket. Mielőtt ezeket részleteznénk, idézzük fel a mátrixalgebra egy fontos szabályát, miszerint két mátrix akkor és csak akkor szorozható egymással össze, ha a bal oldali mátrix oszlopainak száma megegyezik a jobb oldali mátrix sorainak a számával

$$\underline{\underline{A}}_{(N\times M)} \cdot \underline{\underline{B}}_{(M\times L)} = \underline{\underline{C}}_{(N\times L)} .$$

Ezen összefüggés figyelembevételével az 1. táblázatban találhatók a MATLAB rendszerben végrehajtható leggyakrabban alkalmazott mátrixműveletek, ahol *A* és *B* (a fenti szabálynak megfelelő) méretű mátrixok, és *a* pedig egy tetszőleges skalár. A táblázatban foglalt műveletek többsége természetesen vektorokra és skalárra is egyaránt alkalmazhatók.

1. táblázat. Mátrixműveletek

MATLAB művelet	***Jelentés***
>>A+B	Összeadása
>>A-B	Kivonás
>>A*B	Mátrixszorzás
>>a*A	Mátrix skalárral való szorzása
>>A^a	Négyzetes mátrix hatványozása
>>A.*B	Elemenkénti szorzás
>>A./B	Elemenkénti osztás
>>A.^B	Elemenkénti hatványozás
>>A’	Mátrix transzponáltja
>>inv(A)	Mátrix inverze
>>sqrt(A)	Mátrix elemeinek négyzetgyöke
>>abs(A)	Mátrix abszolút értéke
>>tril(A,k)	Alsóháromszög mátrix (k-adik főátlóval bezárólag)
>>triu(A,k)	Felsőháromszög mátrix (k-adik főátlóval bezárólag)

Az eddigi ismeretek alapján, egy példán keresztül egy gyors megoldást szeretnénk ismertetni inhomogén lineáris egyenletrendszerek megoldására. Legyen $\underline{\underline{A}}\vec{x} = \vec{b}$ egyenletrendszer ismeretlen vektora *x*, ahol *A* négyzetes együtthatómátrix és *b* valós elemű véletlen oszlopvektor. Képezzük az *A* 5×5-ös mátrix elemeit a [-2,5] intervallumból

egyenletes valószínűséggel generált egész számokból. A véletlen valós számok általános kerekítésére alkalmazzuk a **round** függvényt. Pl.:

```
>> A=round(7*rand(5,5)-2)
A =
     5     3     2     1    -2
     0     1     4     5     0
     2    -2     4     4     4
     1     4     3     1    -2
     4     1    -1     4    -1

>> b=rand(5,1)
b =
     0.2028
     0.1987
     0.6038
     0.2722
     0.1988.
```

Ekkor az egyenletrendszer gyors megoldása:

```
>> x=A\b
x =
     0.0254
     0.2061
    -0.0349
     0.0264
     0.2497.
```

A megoldás pontossága könnyen ellenőrizhető, ha képezzük a b_0 vektort és annak eltérését (különbségét) az eredeti *b* vektortól. Az alábbiakban látható, hogy a két vektor 15 tizedesjegyet figyelembe véve teljesen megegyezik:

```
>> b0=A*x
b0 =
     0.2028
     0.1987
     0.6038
     0.2722
     0.1988

>> b-b0
ans =
     1.0e-015 *
     0.0278
     0.0555
     0.2220
    -0.0555
     0.0278.
```

1.4 BEÉPÍTETT VEKTOR ÉS MÁTRIXJELLEMZŐ FÜGGVÉNYEK

Vektorok és mátrixok speciális tulajdonságainak gyors számítására a MATLAB saját erőforrásokkal rendelkezik. E függvények a COMMAND WINDOW-ból közvetlenül hívhatók, valamint a tárolt programokba szabadon beépíthetők. E fejezetben e speciális vektor és mátrixjellemző függvényeket vesszük sorra.

Mátrixok és vektorok legnagyobb elemének kiválasztására a **max** függvény szolgál. Vektorok esetén az eredmény egyértelmű, mátrixok esetén egy olyan sorvektort kapunk, mely a mátrix oszlopainak maximális elemeit tartalmazza. Pl.

```
>> M=10*rand(5,5)
M =
   7.3491   1.9112   4.6077   0.0558   9.8299
   6.8732   4.2245   4.5735   2.9741   5.5267
   3.4611   8.5598   4.5069   0.4916   4.0007
   1.6603   4.9025   4.1222   6.9318   1.9879
   1.5561   8.1593   9.0161   6.5011   6.2520

>> max(M)
ans =
   7.3491   8.5598   9.0161   6.9318   9.8299.
```

Tehát a maximális elem megadásához kétszer kell hívnunk a *max* függvényt:

```
>> max(max(M))
ans =
   9.8299.
```

A fentiekhez hasonlóan működik a **min** függvény, mellyel a mátrix legkisebb elemét kereshetjük meg. Oldjuk meg azt a feladatot, hogy megkeressük a mátrix minimális elemének sor- és oszlopindexét. Ehhez bevezetjük az ún. **colon (:)** operátort, mellyel a mátrix valamely sorához vagy oszlopához tartozó összes elemre egyidejűleg (tömören) hivatkozhatunk (ld. 1.7 fejezet). A fenti feladatot két lépésben oldhatjuk meg. Először megkeressük a mátrix minimális elemét és annak oszlopindexét

```
>> [e,i]=min(min(M))
e =
   0.0558
i =
    4.
```

Majd ezután a colon operátor felhasználásával az adott oszlopban megvizsgáljuk az összes sort és a végén kiválasztjuk a minimális elemhez tartozó sorindexet:

```
>> [e,j]=min(M(:,i))
e =
   0.0558
j =
    1.
```

Az eredmény könnyen ellenőrizhető:

```
>> M(j,i)
ans =
   0.0558.
```

Vektor elemeinek az összegét a **sum** függvény adja vissza. Mátrix esetén a függvény annak oszlopaiban szereplő elemeinek az értékét adja össze, és sorvektor formájában szolgáltatja. A szummázás ismételt alkalmazásával tehát megkapjuk a mátrix elemeinek az összegét. A **prod** függvény hasonlóan működik, mely a mátrixelemek szorzatával tér vissza. Vektorok hosszát (komponenseinek a száma) a **length** függvénnyel kérdezhetjük le, ennek eredménye egy skalár, mely mátrix esetén a nagyobb index (sor vagy oszlop) irányában számítja az elemek számát. Pl.:

```
>>v=rand(5,1), A=rand(7,3)
v =
   0.3603
   0.5485
   0.2618
   0.5973
   0.0493

A =
   0.5711   0.8030   0.5134
   0.7009   0.0839   0.7327
   0.9623   0.9455   0.4222
   0.7505   0.9159   0.9614

>> length(v)
ans =
    5

>> length(A)
ans =
    4,
```

mivel az *A* mátrix sorainak a száma nagyobb oszlopainak számánál. További vektor és mátrixjellemző mennyiségeket a 2. táblázat tartalmaz, ahol *v* egy tetszőleges sor- vagy oszlopvektor és *A* pedig tetszőleges méretű mátrix.

2. táblázat. Vektor és mátrixjellemző mennyiségek

MATLAB művelet	***Jelentés***
>>norm(v,p)	Vektor Lp-normája
>>norm(v)	Vektor L_2-normája
>>norm(v,inf)	Vektor L_∞-normája
>>cond(A)	Mátrix kondíciószáma
>>norm(A)	Mátrix normája
>>det(A)	Mátrix determinánsa
>>rank(A)	Mátrix rangja

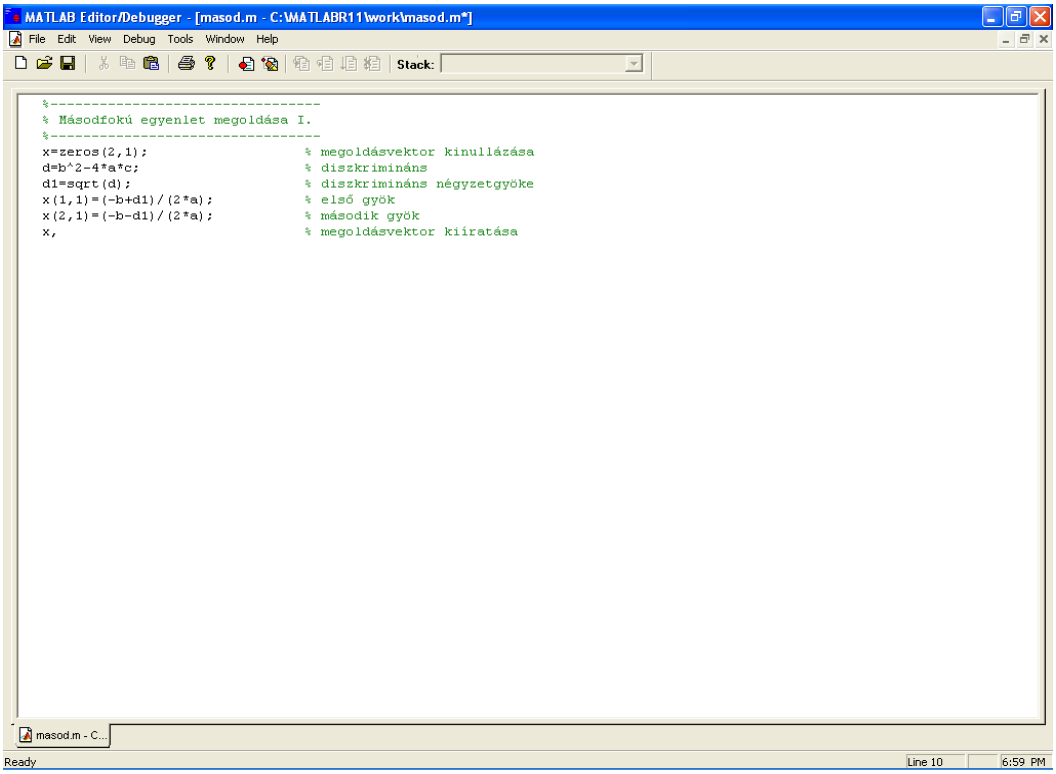
1.5 TÁROLT PROGRAMOK LÉTREHOZÁSA

A MATLAB rendszer alatt kétféle **.m* kiterjesztésű programfájl készíthető. Az első az ún. *script*, amely az egymásután végrehajtandó utasítások sorozatát tartalmazza. A másik csoportot az ún. *function* fájlok képezik, melyek képesek bemenő paramétereket fogadni és a rajtuk elvégzett műveletek eredményét kimenő paraméterek formájában szolgáltatni. Ez utóbbiak dinamikusan felhasználhatók más script fájlba ágyazva, valamint egymásból is közvetlenül hívhatók.

Egy egyszerű példán keresztül szemléltetjük a két fájltípus szerkezete közti különbséget. Legyen a feladat

$$ax^2 + bx + c = 0$$

másodfokú egyenlet megoldása, ahol *a,b,c* tetszőleges komplex konstansok. Az input értékeket kezdetben csak a COMMAND WINDOW-ban adjuk meg. Első esetben egyetlen script fájlt hozzunk létre a fenti feladat megoldása céljából. Ehhez behívjuk a MATLAB menűrendszerében lévő FILE menü, NEW almenüjében az M-FILE parancsot. Ekkor egy EDIT-ablak nyílik meg (ld. 3. ábra).



3. ábra: A MATLAB editor ablaka

A program listája egyszerű, kinullázzuk az *x* vektort az egyenlet gyökei számára, majd értelmezzük a diszkriminánst, és a megfelelő megoldóképletet alkalmazzuk. Megjegyezzük, hogy a továbbiakban a promt nélkül bevezetett sorok az EDITOR-ban szerkesztett program listáját jelenti majd, a promt-al jelöltek (>>) továbbra is a COMAND WINDOW parancsait jelölik. A programok listájában feltüntetett **%** jel után megjegyzést (comment) tehetünk, amelyet a MATLAB a program futása során nem vesz figyelembe.

```
%----------------------------------------
% Másodfokú egyenlet megoldása I.
%----------------------------------------
x=zeros(2,1);              % megoldásvektor kinullázása
d=b^2-4*a*c;               % diszkrimináns
d1=sqrt(d);                % diszkrimináns négyzetgyöke
x(1,1)=(-b+d1)/(2*a);      % első gyök
x(2,1)=(-b-d1)/(2*a);      % második gyök
x,                         % megoldásvektor kiíratása.
```

A fájl mentése a FILE menüben a SAVE AS .. paranccsal történik. A tárolt programot ezután a COMMAND WINDOW-ból indíthatjuk (ne felejtsük a script-et tartalmazó aktuális könyvtárba való belépést). Adjuk meg az *a,b,c* együtthatók értékeit, majd hívjuk a programot a script fájl nevének begépelésével. Ennek eredményeként megkapjuk a másodfokú egyenlet megoldásait. Egy példa a program futtatására:

```
>>a=5+i;
>>b=2.3-2*i;
>>c=1;
>> masod
x =
  -0.1313 - 0.2299i
  -0.2341 + 0.7029i.
```

Második esetben saját fejlesztésű függvénnyel oldjuk meg a feladatot. Ekkor a függvény bemenő paraméterei az *a,b,c* együtthatók, kimenő paramétere a megoldást tartalmazó oszlopvektor lesz. A függvényt minden esetben be kell vezetni a *function* szócskával és a paraméterlista elemeit is meghatározott sorrenben kell megadni. További megszorítást az jelent, hogy a függvény nevének meg kell egyeznie a programlistát tároló **.m* kiterjesztésű fájl nevével. Ez alapján a feladat megoldása:

```
%----------------------------------------
% Másodfokú egyenlet megoldása II.
%----------------------------------------
function [x]=masod2(a,b,c)
    x=zeros(2,1);
    d=b^2-4*a*c;
    d1=sqrt(d);
    x(1,1)=(-b+d1)/(2*a);
    x(2,1)=(-b-d1)/(2*a);
return.
```

A függvény hívása a COMMAND WINDOW-ból történik, ahol meg kell adni a függvény nevét, ill. a kimenő (*x*) és a bemenő paramétereket (*a,b,c* számszerű értékeit):

```
>> [x]=masod2(a,b,c)
x =
  -0.1313 - 0.2299i
  -0.2341 + 0.7029i.
```

A fenti program kódjában már megjelentek bizonyos aritmetikai kifejezések. Ezek alkalmazásánál fontos tudnunk a műveletek végrehajtási sorrendjére vonatkozó *precedencia szabály*-okat:

1. Két különböző precedenciájú operátor esetén először a magasabb precedenciájú operátor által előírt művelet kerül végrehajtásra (ld. 3. táblázat).
2. Azonos precedenciájú operátorok a felírásuk sorrendjében (balról jobbra) kerülnek kiértékelésre.
3. A zárójel megváltoztatja a műveletek kiértékelésének sorrendjét. Először a zárójelben megadott műveleteket hajtjuk végre.

3. táblázat. Aritmetikai operátorok precedenciája

MATLAB jelölés	***Precedencia szint***
^	1.
+, - (előjel)	2.
, ., /, ./, \	3.
+, -	4.
:	5.

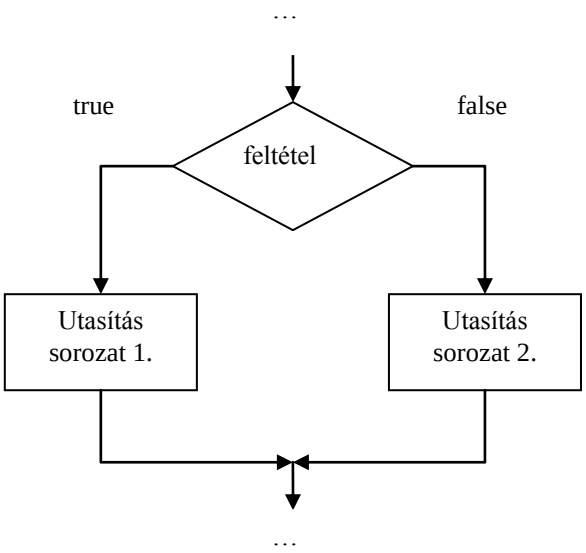
1.6 FELTÉTELES UTASÍTÁSOK

A feltételes utasítások olyan programelágazást tesznek lehetővé, melyek bizonyos logikai kifejezések teljesüléséhez, ill. nem teljesüléséhez köthetők. A feltételeket előíró logikai kifejezésekben szereplő operátorokat a 4. táblázatban találhatjuk meg. Ha a feltétel teljesül, akkor eredménye igaz (true = 1), ellenkező esetben hamis (false = 0) lesz.

4. táblázat. Relációs és logikai operátorok

MATLAB jelölés	***Jelentés***
= =	Egyenlő
<	Kisebb
>	Nagyobb
<=	Kisebb egyenlő
>=	Nagyobb egyenlő
~=	Nem egyenlő
&	Logikai „és”
|	Logikai „vagy”
~	Negáció

A feltételes utasítást az **if…else…end** szerkezet alkalmazásával valósíthatjuk meg, mely két-, vagy többirányú elágazást tesz lehetővé. A többirányú programelágazást *if* utasítások egymásba ágyazásával, vagy egyszerűen az ún. *case* utasítás alkalmazásával tehetjük meg. A kétirányú elágazást biztosító *if* utasítás folyamatábráját és MATLAB nyelvű implementációs kódját a 4. ábrán láthatjuk. Az *if* szerkezetben természetesen nemcsak logikai kifejezés állhat, hanem olyan változó is, melynek 0 vagy 1 értékéhez köthetjük az utasítások végrehajtását.




```
if feltétel (logikai kifejezés)
   utasítás_1.a;
   utasítás_1.b;
   ...
   utasítás_1.N;
else
   utasítás_2.a;
   utasítás_2.b;
   ...
   utasítás_2.M;
end
```

4. ábra. Az if-utasítás folyamatábrája és MATLAB sémája

Tekintsük azt az egyszerű példát, amikor egy megadott vektor elemszámát vizsgáljuk. Pl., ha a vektor dimenziójának a száma három, akkor irassuk ki a képernyőre a „3 dimenziós” szöveget, ellenkező esetben azt, hogy „nem 3 dimenziós”:

```
>> vs1=[2 3 4]; vs2= [2;5;14;3;9];
>> if length(vs1)==3 disp('3 dimenziós'); else disp('nem 3 dimenziós'); end
3 dimenziós
>> if length(vs2)==3 disp('3 dimenziós'); else disp('nem 3 dimenziós'); end
nem 3 dimenziós.
```

Látható, hogy ezt az egyszerű feladatot a parancsablakon keresztül is megoldhatjuk és a feltételes utasítást egy sorban, tömören megadhatjuk (természetesen a formai szabályok szigorú betartása mellett).

Az *if* utasítás alkalmas többirányú programelágazás alkalmazására is. Három feltétel esetén még nem szükséges a *case* utasítást használnunk, ekkor az **if…elseif...else…end** szerkezet hatékonyan alkalmazható. Ennek sémája a következő:

```
if feltétel_1
    utasítás_1.a;
    ...
    utasítás_1.N;
elseif feltétel_2
    utasítás_2.a;
    ...
    utasítás_2.M;
else
    utasítás_3.a;
    ...
    utasítás_3.W;
end.
```

Példaként tekintsük az 1.5 fejezetben kitűzött feladatot, melyet egészítsünk ki annyival, hogy a másodfokú egyenlet megoldását a valós számok körére korlátozzuk. Abban az esetben, amikor a diszkrimináns kisebb, mint zérus, akkor csak komplex megoldás van, azaz az egyenletnek a valós számok halmazán nincs megoldása. Ekkor írassuk ki a képernyőre, hogy nincs megoldás, és a két megoldást tartalmazó változónak adjuk sztring (karaktervváltozó) értéket: „komplex szám”. A feladat megoldása függvény segítségével a következő:

```
%-----------------------------------------
% Másodfokú egyenlet megoldása III.
%-----------------------------------------
function [x1,x2]=masod3(a,b,c)
    d=b^2-4*a*c;
    d1=sqrt(d);
    if d<0
            disp('Nincs valós gyök');
                x1='komplex szám';
                x2=x1;
            elseif d>0
                x1=(-b+d1)/(2*a);
                x2=(-b-d1)/(2*a);
            else
                x1=(-b+d1)/(2*a);
                x2=x1;
            end
return.
```

A programot a feladatnak megfelelően hívjuk meg két különböző esetben, ahol az eredményt először valós, azután pedig komplex számok képezik:

```
>> [x1,x2]=masod3(2,2,-4)
x1 =
     1
x2 =
    -2

>> [x1,x2]=masod3(2,2,4)
Nincs valós gyök
x1 =
    komplex szám
x2 =
    komplex szám.
```

Végül tekintsük azt az esetet, amikor a feltételes utasításban az elágazások száma háromnál több. Ekkor a **case** utasítást alkalmazhatjuk. Az utasítást a **switch…end** szerkezetben használjuk. Itt is az egyes feltételek külön utasítássorozatokat vezethetnek be, melyek opcionális végrehajtása után folytatódik a program futása. Tegyük fel, hogy egy numerikus algoritmusokból álló programrendszert írunk, melynek elemei egy-egy algoritmust magában foglaló különálló script fájl. Készítsünk egy keretprogramot, melyben kiválaszthatjuk, melyik script-et szeretnénk elsőként futtatni. A főprogram neve legyen *start.m*, a script-eké pedig: *program1.m, program2.m, program3.m* és *program4.m*. Kérjük be

a választott algoritmust futtató program sorszámát, ha ez a szám nagyobb, mint a scriptek száma lépjünk ki a programból. Az értékadás billentyüzetről való végrehajtásáról az **input** függvény gondoskodik, mely egy megadott változóhoz rendeli az általunk megadott számot. A beolvasás során a programnak kommunikálnia kell a felhasználóval, ill. fel kell kínálnia a választható lehetőségeket. Az **fprint** utasítással a képernyőre írhatunk ki üzeneteket megadott formátumban. A kiíratást végző függvény argumentumának első paramétere a kiírás helyére vonatkozik, mely jelen esetben a displaynek megfelelő 1-es szám. Az fprintf utasítás alkalmazásával kapcsolatban még később a fájlműveleteknél bővebben kitérünk. A feladat megoldása case utasítással a következő:

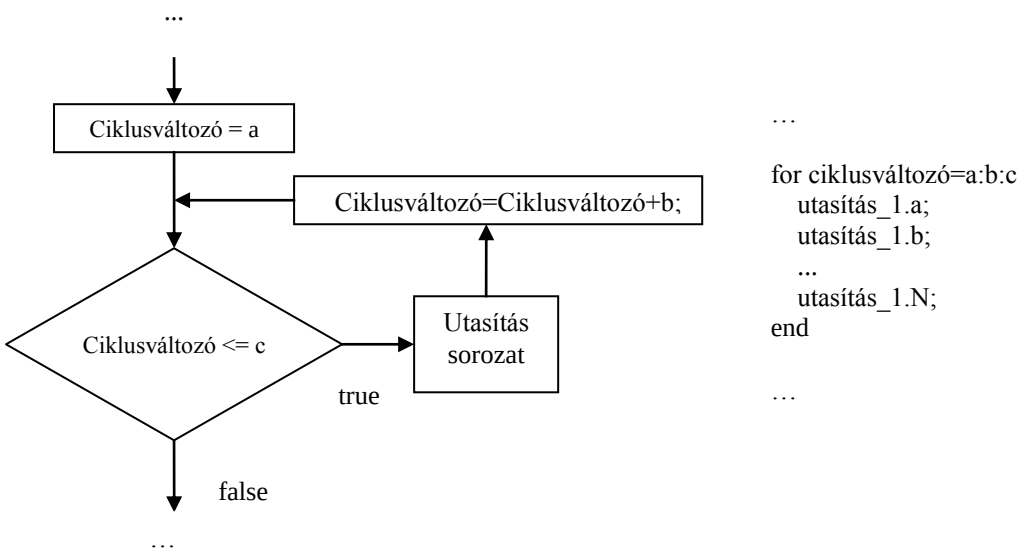
```
%-------------------------------------------------------
% Főprogram numerikus algoritmusok futtatására
%-------------------------------------------------------
clc;
clear all;
fprintf(1,' 1. PROGRAM 1 FUTTATÁSA \n');
fprintf(1,'\n');
fprintf(1,' 2. PROGRAM 2 FUTTATÁSA \n');
fprintf(1,'\n');
fprintf(1,' 3. PROGRAM 3 FUTTATÁSA \n');
fprintf(1,'\n');
fprintf(1,' 4. PROGRAM 4 FUTTATÁSA \n');
fprintf(1,'\n');
valaszt=input(' Adja meg a választott program sorszámát = ');
switch valaszt;
   case 1,
      program1;
   case 2,
      program2;
   case 3,
      program3;
   case 4,
      program4;
   otherwise
      fprintf(1,'\n');
      disp(' Kilépés.');
 end.
```

1.7 CIKLUS UTASÍTÁSOK

Azonos utasítások egymást követő többszöri végrehajtására ciklust szervezhetünk, mellyel jelentősen redukálhatjuk a programírás idejét és a programlista hosszát. A ciklusok ismétlésének a számát kétféleképpen szabályozhatjuk. Első esetben egy ciklusváltozó felhasználásával előre megadjuk a ciklus ismétlődésének a számát. A másik megoldás szerint bizonyos logikai vagy egyéb feltételhez kötjük a ciklus ismétlődését vagy annak befejezését.

A rögzített ismétlésszámon alapuló ciklusok írására a **for…end** szerkezet alkalmas. Ennek keretében a ciklusváltozónak kezdeti értéket (a), végértéket (c), és a növekményét (b) rögzítjük. Megadásánál nem feltétel, hogy a végérték nagyobb legyen a kezdeti értéknél,

mivel az inkrementum lehet negatív, valamint nem egész (valós) szám is. A ciklusváltozó lehet vektor is, melynek az az előnye, hogy a ciklus futása során a változó csak megadott értékeket vehet fel. Mivel a MATLAB alkalmas mátrixok kezelésére, így a ciklusutasítások egymásba ágyazásával hatékonyan kezelhetők bonyolultabb mátrixműveletek is. A for ciklus folyamatábráját és MATLAB nyelvű sémáját az 5. ábrán láthatjuk.



5. ábra. A for-ciklus folyamatábrája és MATLAB sémája

A *for*-ciklus gyakorlására írjunk egy script programot *hilbert.m* néven, mely a tetszőleges méretű Hilbert-mátrix elemeit számítja ki. A Hilbert-mátrix definíciója

$$H_{ij} = \frac{1}{i + j - 1}, \tag{1.7.1}$$

ahol *i* a sor-, *j* az oszlopindexet jelöli. Az (1.7.1) mátrix négyzetes, melynek dimenzióját kérjük be billentyűzetről. Végül az eredményt hasonlítsuk össze a MATLAB rendszerben definiált Hilbert-mátrix-al, úgy hogy annak elemei és a programunk által számított mátrix elemeinek különbségét és annak mátrix-normáját képezzük (ld. 1.3 fejezet). A program listája:

```
%-----------------------------------
%  Hilbert mátrix számítása
%-----------------------------------
n=input('Mátrix dimenziója =');
H=zeros(n,n);
for i=1:n
    for j=1:n
            H(i,j)=1/(i+j-1);
    end
end
H,
Elteres=norm(hilb(n)-H),
```

A futási eredmény n=7 esetén:

```
>> hilbert
Mátrix dimenziója =7
H =
   1.0000   0.5000   0.3333   0.2500   0.2000   0.1667   0.1429
   0.5000   0.3333   0.2500   0.2000   0.1667   0.1429   0.1250
   0.3333   0.2500   0.2000   0.1667   0.1429   0.1250   0.1111
   0.2500   0.2000   0.1667   0.1429   0.1250   0.1111   0.1000
   0.2000   0.1667   0.1429   0.1250   0.1111   0.1000   0.0909
   0.1667   0.1429   0.1250   0.1111   0.1000   0.0909   0.0833
   0.1429   0.1250   0.1111   0.1000   0.0909   0.0833   0.0769
Elteres =
   0.
```

A *for*-ciklus MATLAB-ban még tömörebben kifejezhető a *colon* operátor segítségével, mellyel az 1.4 fejezetben ismerkedtünk meg (mátrix elemeire való hivatkozás esetén). Ezúttal a fenti operátort a ciklusutasítás egyszerűsítésére használjuk fel. Példaként hozzuk létre a 10 elemű *v1* sorvektort, melynek elemei megegyeznek az indexek értékével:

```
>> for q=1:1:10 v1(q)=q; end; v1
v1 =
    1    2    3    4    5    6    7    8    9   10.
```

Ugyanerre a megoldásra vezet a *colon* operátorral *v2* vektor képzése:

```
>> v2=[1:1:10],
v2 =
    1    2    3    4    5    6    7    8    9   10.
```

Látható, hogy az utóbbi esetben nincs szükség csak a kezdeti-, a végérték és a lépésköz megadására. A colon használatával igen gyorsan kiírathatók függvények helyettesítési értékei. Pl. értékeljük ki az $f(x)=e^{-x}\sin(x)$ függvényt ($0\le x\le\pi$) $\Delta x=\pi/15$ lépésközzel, majd a független és a függő változó értékeit helyezzük el az *A* mátrixba:

```
>> x=[0:pi/15:pi]';y=exp(-x).*sin(x);A=[x,y],
A =
    0         0
    0.2094    0.1686
    0.4189    0.2675
    0.6283    0.3136
    0.8378    0.3215
    1.0472    0.3039
    1.2566    0.2707
    1.4661    0.2296
    1.6755    0.1862
    1.8850    0.1444
    2.0944    0.1066
    2.3038    0.0742
    2.5133    0.0476
    2.7227    0.0267
    2.9322    0.0111
    3.1416    0.0000.
```

A *colon* operátor további alkalmazási területei lehetnek a részvektor, részmátrix képzések, és a mátrix elemeinek felcserélése. Pl. készítsünk el egy *A* mátrix partíciót, mely az *M* második és harmadik sorát, valamint első, második és harmadik oszlopát tartja meg:

```
>> M=rand(5,5),A=M(2:3,1:3),
M =
    0.9501    0.7621    0.6154    0.4057    0.0579
    0.2311    0.4565    0.7919    0.9355    0.3529
    0.6068    0.0185    0.9218    0.9169    0.8132
    0.4860    0.8214    0.7382    0.4103    0.0099
    0.8913    0.4447    0.1763    0.8936    0.1389

A =
    0.2311    0.4565    0.7919
    0.6068    0.0185    0.9218.
```

Abban az esetben, amikor a részmátrixnak a partíció bizonyos egymást nem követő elemeit kell tartalmaznia, élhetünk a mátrix indexeire való vektorikus hivatkozással. Pl. az M(2,1), M(2,3), M(3,1), M(3,3)-ból képzett *A* részmátrix:

```
>> A=M([2,3],[1,3])
A =
    0.2311    0.7919
    0.6068    0.9218.
```

A mátrix bizonyos elemeinek felcserélése számos lineáris egyenletrendszer megoldásnál előfordul. Pl. cseréljük fel a fent látható *M* mátrix első és második sorát a colon használatával, majd az újonnan kapott mátrix harmadik és negyedik oszlopát (*M* neve ne változzon meg):

```
>> M([1,2],:)=M([2,1],:)

M =

   0.2311    0.4565    0.7919    0.9355    0.3529
   0.9501    0.7621    0.6154    0.4057    0.0579
   0.6068    0.0185    0.9218    0.9169    0.8132
   0.4860    0.8214    0.7382    0.4103    0.0099
   0.8913    0.4447    0.1763    0.8936    0.1389

>> M(:,[3,4])=M(:,[4,3])

M =
   0.2311    0.4565    0.9355    0.7919    0.3529
   0.9501    0.7621    0.4057    0.6154    0.0579
   0.6068    0.0185    0.9169    0.9218    0.8132
   0.4860    0.8214    0.4103    0.7382    0.0099
   0.8913    0.4447    0.8936    0.1763    0.1389.
```

Végül a *colon* operátor és a *for*-ciklus együttes alkalmazására nézzünk egy utolsó példát. Oldjuk meg az $\underline{\underline{A}}\vec{x} = \lambda\vec{x}$ sajátérték feladatot. A MATLAB-ban az **eig** függvény értelmezése szerint az *x* sajátvektorokat egy mátrix oszlopaiban, és az ennek megfelelő λ sajátértékeket egy másik mátrix főátlójában kapjuk. Írjunk *sajatertek.m* néven programot a sajátértékek és a sajátvektorok gyors számítására. Definiáljunk egy skaláris mérőszámot a számítási pontosság jellemezésére, mely az $\vec{e}^{(i)} = \underline{\underline{A}}\vec{x}^{(i)} - \lambda^{(i)}\vec{x}^{(i)},\ i = 1...length(x)$ eltérésvektorok Euklideszi normájának az összegét adja meg. A script fájl a következő:

```
%-------------------------------------
%  Sajátérték feladat megoldása
%-------------------------------------
n=input('Mátrix dimenziója =');
A=rand(n,n);
[x,Lambda]=eig(A);
Hiba=0;
for i=1:n
    elteresv=zeros(n,1);
    elteresv=(A*x(:,i))-(Lambda(i,i)*x(:,i))
    Hiba=Hiba+norm(elteresv);
end
A,
x,
Lambda=diag(Lambda),
Hiba,
```

A futási eredmény:

```
>> sajatertek
Mátrix dimenziója =5

A =
   0.8892    0.5944    0.9805    0.6390    0.4831
   0.8660    0.3311    0.7918    0.6690    0.6081
   0.2542    0.6586    0.1526    0.7721    0.1760
   0.5695    0.8636    0.8330    0.3798    0.0020
   0.1593    0.5676    0.1919    0.4416    0.7902

x =
   0.5705    0.0441   -0.6839   -0.4407    0.4078
   0.5134   -0.1289   -0.2756   -0.3798    0.2314
   0.3362    0.2189    0.5980    0.7912   -0.8150
   0.4392    0.4215    0.3139   -0.1066    0.3081
   0.3240   -0.8694    0.0146    0.1554   -0.1452

Lambda =
   2.7681         0.6039        -0.0322        -0.3746        -0.4222

Hiba =
   3.5139e-015.
```

A rögzítetlen ismétlésszámon alapuló ciklusok végrehajtására a **while…end** szerkezet alkalmas, melynek szerkezete a MATLAB-ban a következő:

```
while feltétel
    utasítás_a;
    ...
    utasítás_N;
end.
```

A fenti ciklus addig ismétlődik, míg a feltétel eredménye igaz (azaz amíg a feltétel eredménye hamis logikai értéket fel nem vesz). Ennek szemléltetésére nézzük a következő példát. Tegyük fel azt a kérdést, hogy az x=2 (radián) helyen hány tagot kell a koszinusz függvény Mc-Laurin sorában összegezni ahhoz, hogy az a koszinusz függvény valódi értékét $\varepsilon=10^{-10}$ pontossággal közelítse. A pontosság jellemzésére használjuk a két érték abszolút eltérését. Induljunk ki a koszinusz függvény Mc-Laurin sorának képletéből:

$$\cos(x)=\sum_{k=1}^{K}(-1)^k\frac{x^{2k}}{(2k)!}, \qquad (1.7.2)$$

ahol a feladat szerint *K* a kérdés. A faktoriális függvény alkalmazása céljából használjunk gamma-függvényt, melyre igaz, hogy $\Gamma(z+1)=z!$, így z=2k helyen az (1.7.2) sorfejtésben szereplő nevező könnyen átírható. Ezzel a program:

```
%-----------------------------------
%   Mc-Laurin sor alkalmazása
%-----------------------------------
x=2;
y=cos(2);                   % valódi érték
%
%   Inicializálás:
%
k=0;                        % tagok száma
d=1;                % abszolút eltérés
yk=0;                       % közelítő érték
%
%   Számítás:
%
while d>10^(-10)
    yk=yk+((-1)^k*(x^(2*k)))/gamma(2*k+1);
    d=abs(y-yk);
    k=k+1;
end
Hiba=d,
Tagok szama=k,
```

Vegyük észre, hogy a *while*-ciklusban nincs ciklusváltozó, ezért megadásáról nekünk kell külön gondoskodnunk. A ciklusban az utasítássorozat végrehajtása után az aktuális ciklusváltozó értékét 1-el meg kell növelnünk (jelen esetben a *k* értékét). Ha ezt nem tesszük meg, a program ugyanazt a számítási sorozatot ismétli, amely könnyen végtelen ciklushoz vezet.

1.8 LINEÁRIS ÉS NEMLINEÁRIS EGYENLET(RENDSZER)EK MEGOLDÁSA

Az eddigi ismeretek alapján nagyobb programok írásába is belekezdhetünk. A MATLAB numerikus eszköztára gazdag, azonban a mérnöki gyakorlat során előfordulnak olyan speciális problémák is, melyek megoldása megbízhatóbb eredményt ad, ha saját készítésű algoritmussal dolgozunk. A nagyméretű lineáris egyenletrendszerek megoldásánál sok esetben numerikus problémába ütközhetünk (rosszul kondícionáltság, szinguláris együtthatómátrix), és az adott problémától függ, hogy melyik módszer alkalmazása vezet eredményre. Ilyenkor tanácsos kihasználni a MATLAB beépített függvényeit (kondíciószám, rang stb.), melyek diagnosztikai paraméterekként segíthetik előrevetíteni a problémákat.

Ebben a részben talán a legegyszerűbb *lineáris egyenletrendszer* megoldó algoritmust, a részleges főelemkiválasztás nélküli Gauss-Jordan eljárás egy lehetséges MATLAB implementációját mutatjuk be. A módszer elvi alapjait röviden összefoglaljuk. Az $\underline{\underline{A}}\vec{x} = \vec{b}$ inhomogén lineáris egyenlet i=1,…,m egyenletből és j=1,..,n ismeretlenből áll. Ha m=n, akkor a feladat egyértelműen megoldható algebrai eszközökkel, ellenkező esetben alul-, vagy túlhatározott rendszerrel kell számolnunk, melynek nincs algebrai megoldása (ilyenkor is létezik megoldás, mellyel a matematika optimalizáció elmélete foglalkozik). A fenti egyenlet m=n estén részletesen kiírva:

$$\left.\begin{array}{l} a_{11}x_1 + \ldots + a_{1j}x_j + \ldots + a_{1n}x_n = b_1 \\ \qquad\qquad \vdots \\ a_{i1}x_1 + \ldots + a_{ij}x_j + \ldots + a_{in}x_n = b_i \\ \qquad\qquad \vdots \\ a_{n1}x_1 + \ldots + a_{nj}x_j + \ldots + a_{nn}x_n = b_n \end{array}\right\}. \qquad (1.8.1)$$

A Gauss-módszer két fázisból áll. Először a (1.8.1) egyenletrendszert felsőháromszög alakúra hozzuk (eliminációs fázis), majd ezután a felsőháromszög mátrixú egyenletrendszert megoldjuk (visszahelyettesítő fázis). Az eliminációs fázis első lépésében arra kell törekednünk, hogy az $a_{11}x_1$ tag alatti elemeket kinullázzuk. Ennek feltétele: $a_{i1} - \gamma a_{11} = 0 \rightarrow \gamma = a_{i1}/a_{11}$ (ha i≠1). Ez alapján az *i*-edik sor elemeiből kivonjuk az első sor gamma-szorosát. A kapott ekvivalens egyenletrendszer a következő, ahol a hullámos jellel megkülönböztetett együtthatók azt jelzik, hogy azok (1.8.1) felülírt értékei:

$$\left.\begin{array}{l} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ \qquad \tilde{a}_{22}x_2 + \ldots + \tilde{a}_{2n}x_n = \tilde{b}_2 \\ \qquad\qquad \vdots \\ \qquad \tilde{a}_{n2}x_2 + \ldots + \tilde{a}_{nn}x_n = \tilde{b}_n \end{array}\right\}.$$

Könnyen belátható, hogy ha az *i*-edik sorból kivonjuk a *k*-adik sor gamma-szorosát ($a_{i,k} - \gamma a_{k,k} = 0 \rightarrow \gamma = a_{i,k}/a_{k,k}$), akkor a következő egyenletrendszerre jutunk

$$\left(a_{i,k+1} - \gamma a_{k,k+1}\right)x_{k+1} + \ldots + \left(a_{i,n} - \gamma a_{k,n}\right)x_n = b_i - \gamma b_k \text{, ahol } \left\{\begin{array}{l} i = k+1,\ldots,n \\ k = 1,\ldots,n-1 \end{array}\right\}. \qquad (1.8.2)$$

Ezzel az eliminációs fázis utolsó lépésében (i=n) kapott egyenletrendszer a következő:

$$\left.\begin{aligned} a_{11}x_1 + \ldots + a_{1h}x_h + \ldots + a_{1n}x_n &= b_1 \\ &\vdots \\ \tilde{a}_{hh}x_h + \ldots + \tilde{a}_{hn}x_n &= \tilde{b}_h \\ &\vdots \\ \tilde{a}_{nn}x_n &= \tilde{b}_n \end{aligned}\right\}.$$

Ha $a_{nn}\neq 0$ és $a_{11}\neq 0$, akkor az egyenletrendszer pivotálás nélkül megoldható a visszahelyettesítő fázisban az alábbi algoritmussal:

$$x_h = \begin{cases} \dfrac{\tilde{b}_n}{\tilde{a}_{nn}} & , h = n \\ \dfrac{1}{\tilde{a}_{hh}}\left(\tilde{b}_h - \sum_{q=h+1}^{n}\tilde{a}_{hq}x_q\right) & , h = n-1,\ldots,1 \end{cases}. \qquad (1.8.3)$$

A fenti módszer alkalmazására nézzünk egy számítási példát. Elsőként a programírásnál létre kell hoznunk (1.8.1) egyenletrendszert, ez legyen a következő:

$$\left.\begin{aligned} x_1 + 2x_2 + 5x_3 - x_4 &= 4 \\ 2x_1 + 3x_2 + 8x_3 - 3x_4 &= 7 \\ x_1 + 2x_2 + 6x_3 - 5x_4 &= 8 \\ x_1 + 2x_2 - 2x_3 + x_4 &= 5 \end{aligned}\right\},$$

melyre ezután alkalmazhatjuk az (1.8.2) és (1.8.3) képleteket, majd végül kiíratjuk az egyenletrendszer megoldását tartalmazó *x* vektort és az elemi műveletek számát. Ez utóbbit a **flops** utasítással kapjuk, mely hasznos mérőszám lehet a numerikus algoritmus gazdaságos és hatékony működése szempontjából (memóriaigény, egyszerűbb struktúra stb.). Az 1 flop egyenlő azzal a számítási munkával, mely 1 aritmetikai alapművelet elvégzéséhez kell. Az egyenletrendszert megoldó program teljes listája:

```
%------------------------------------------------------------------
% Lineáris egyenletrendszer megoldása Gauss-módszerrel
%------------------------------------------------------------------
clc;
clear;
%
% Együttható mátrix és jobb oldali vektor megadása
%
A=[1,2,5,-1;2,3,8,-3;1,2,6,-5;1,2,-2,1];
b=[4,7,8,5];
disp('A=');
disp(A);
disp('b=');
disp(b');
```

```
%
% Eliminációs fázis
%
[n,n]=size(A);
for k=1:n-1                              % felsőháromszög mátrix létrehozása
    for i=k+1:n
            gamma=A(i,k)/A(k,k);
            A(i,k+1:n)=A(i,k+1:n)-gamma*A(k,k+1:n);
            b(i)=b(i)-gamma*b(k);
    end
end
for i=1:n                                % alsóháromszög zérus elemekkel feltöltése
    for j=1:n
            if j<i A(i,j)=0; end
    end
end
disp('Transzformált A-mátrix és b-vektor:');
disp('A=');
disp(A);
disp('b=');
disp(b');
%
% Visszahelyettesítő fázis
%
x=zeros(n,1);
x(n)=b(n)/A(n,n);                        % megoldás h=n esetben
for h=n-1:-1:1
    ax=0;
    for q=h+1:n
            ax1=A(h,q)*x(q);
            ax=ax+ax1;
    end
    x(h)=(b(h)-ax)/A(h,h);               % megoldás h=n-1,...,1 esetekben
end
disp('Megoldás:');
disp('x=');disp(x);
```

A futási eredmény:

```
A=
   1    2    5   -1
   2    3    8   -3
   1    2    6   -5
   1    2   -2    1

b=
   4
   7
   8
   5
```

Transzformált A-mátrix és b-vektor:

```
A=
    1    2    5   -1
    0   -1   -2   -1
    0    0    1   -4
    0    0    0  -26

b=
    4
   -1
    4
   29
```

Megoldás:

```
x=
  -0.8846
   3.0385
  -0.4615
  -1.1154.
```

Az f(x)=0 alakban megadott *nemlineáris egyenlet*-ek megoldására az **fzero** függvény szolgál. Ehhez egy function-ben kell megadnunk a megoldandó egyenletet, mely a független változón kívül további paramétereket is tartalmazhat (itt arra kell figyelni, hogy a paraméterlistában a többi változót is fel kell tüntetni). Az fzero függvény hívásakor meg kell jelölni a function nevét, melyben az egyenletet definíciója található, továbbá a numerikus algoritmushoz egy kiindulási pontot az x-tengelyen, és egy opcionális vektort ([] esetén alapbeállítás), majd legvégül sorrendben a function paraméterlistáján található további függvény paraméter-eket. Pl. oldjuk meg a $3x^3-x^2-20x+20=0$ egyenletet. Ehhez létre kell hoznunk az *fequ.m* fájl-t az *x* független változóval és *p* paraméterrel:

```
function [f]=fequ(x,p)
    f=3*x*x*x-x*x-20*x+p;
return
```

Az fzero hívása esetén információt kapunk a zérushelyről (*x*) és a hozzá tartozó függvényértékről (*f*), a megoldás sikerességéről (1-true, 0-false), és a rendszer által automatikusan alkalmazott keresési algoritmusról:

```
>> [x,f,exit,iter]=fzero('fequ',[0],[],20)
x =
   1.1736
f =
    0
exit =
    1
iter =
   intervaliterations: 12
   iterations: 6
   funcCount: 31
```

A megoldás ellenőrzését elvégezhetjük a **feval** függvény (function) kiértékelkő utasítással (megjegyezzük, hogy függvény függvényben történő hívása esetén is ezt az utasítást kell alkalmazni). A feval utasítással itt a parancsablakban közvetlenül értékelhetjük ki a zérushelyen az f(x) függvény értékét (az fequ paramétereinek aktuális értékével), amelynek természetesen y=0 körüli értéknek kell lennie:

```
>> y=feval('fequ',[1.1736],20)
y =
 -8.9912e-006.
```

Polinomok zérushelyének keresése esetén létezik egyszerűbb megoldás a fentinél. Ezen egyenletek általános alakja a következő (*a*-együtthatók):

$$a_0 + \sum_{i=1}^{n} a_i x^i = 0.$$

A **roots** függvény alkalmas a zérushely gyors kiszámítására, melynek argumentumában a független változó együtthatóit kell egy sorvektorban szerepeltetni, ahol a változó legmagasabb kitevőjű hatványához tartozó együttható szerepel először. Pl., a $-10.3x^4-5x^3+2x^2+2.1x-4.4=0$ egyenlet megoldása a roots használatával:

```
>> x=roots([-10.3,5,2,2.1,-4.4])
x =
  -0.5038 + 0.5964i
  -0.5038  - 0.5964i
   0.7466 + 0.3788i
   0.7466  - 0.3788i.
```

Nemlineáris egyenletrendszer-ek megoldása hasonló, itt is egy függvényben kell definiálnunk az egyenletrendszert és annak paramétereit. Legyen a kérdéses egyenletrendszer, ahol „p” egy valós konstans.

$$\left.\begin{aligned} x_5 - x_1^2 &= -3 \\ x_1 + x_2 &= p \\ e^{x_3+p} - x_1 &= -0.1 \\ x_3 + x_4 &= 0.3 \\ x_1 x_3 - x_2 x_4 x_5 &= 0 \end{aligned}\right\}.$$

Az ennek megfelelő *fequ2.m* function, melynek bemenő paraméterei az *x* vektor és a *p* paraméter:

```
function [f]=fequ2(x,p)
    f(1)=x(5)-x(1)*x(1)+3;
    f(2)=x(1)+x(2)-p;
    f(3)=exp(x(3)+p)-x(1)+0.1;
    f(4)=x(3)+x(4)-0.3;
    f(5)=x(1)*x(3)-x(2)*x(4)*x(5);
return
```

Általánosan az F(x)=0 ($x \in R^n$) egyenletrendszert a legkisebb négyzetek módszerén alapuló illesztéssel oldja meg a MATLAB. A megoldást az **fsolve** függvénnyel kaphatjuk meg, mely az fzero függvény többváltozós megfelelője. A fentiekhez hasonlóan a F(x) függvény értelmezési tartományának egy pontjából indul ki az eljárás, és az fzero-nál ismertetett információkon túl megadja a függvény-rendszerre számított Jacobi-mátrixot:

```
>> [x,F,exit,iter,jacobi]=fsolve('fequ2',[1;1;1;1;0],[],2)
x =
   2.5637
  -0.5637
  -1.0983
   1.3983
   3.5724
F =
  -0.4885
   0.0888
  -0.1416
   0.0500
  -0.1332
exit =
     1
iter =
      iterations: 8
      funcCount: 71
      stepsize: 1.0000
      algorithm: 'medium-scale: Gauss-Newton, line-search'
jacobi =
  -5.1273         0         0         0    1.0000
   1.0000    1.0000         0         0         0
  -1.0000         0    2.4637         0         0
         0         0    1.0000    1.0000         0
  -1.0983   -4.9955    2.5637    2.0137    0.7882.
```

1.9 NUMERIKUS DIFFERENCIÁLÁS ÉS INTEGRÁLÁS

A *numerikus deriválás* számára léteznek a MATLAB rendszerben beépített függvények, de mivel egyszerű formuláról van szó, a következőkben saját programot írunk. Az első és a második deriváltat a Taylor-sorfejtésből vezethetjük le, ahol $O(h^2)$ pontosság mellett a következő képletek adódnak:

$$\frac{df(x)}{dx} \cong \frac{f(x+h)-f(x-h)}{2h}, \qquad \frac{d^2f(x)}{dx^2} \cong \frac{f(x+h)-2f(x)+f(x-h)}{h^2}, \tag{1.9.1}$$

ahol *h* elegendően kicsiny pozitív szám. A deriválás alkalmazására egy mélyfúrási geofizikai probléma megoldását tűzzük ki célul.

A mélyfúrási geofizikai mérésekből a fúrással harántolt rétegek bizonyos petrofizikai tulajdonságait tudjuk meghatározni, mint a porozitás, víz- és szénhidrogén-telítettség, agyagtartalom, kőzetösszetétel, permeabilitás stb. A petrofizikai paramétereket a geofizikai adatok inverziójával határozzuk meg, mivel ezek közvetlenül nem mérhető mennyiségek, azonban a karotázs szondákkal mért fizikai paraméterek információt hordoznak róluk. A geofizikai inverz feladat megoldása során a petrofizikai modellen ún. szonda-válaszegyenletek segítségével elvi adatokat számítunk, melyeket az inverziós eljárásban a mért adatokkal összehasonlítva (iteratív eljárásban) becslést végzünk a petrofizikai paraméterek tényleges értékére vonatkozóan. Az inverziós eljárást célszerű, ha ún. érzékenységi-vizsgálat előzi meg, mely arról tájékoztat, hogy méréseinket egy-egy petrofizikai paraméter mennyire befolyásolja egy adott paraméter-tartományban. Az érzékenységi függvény definíció szerint az adatoknak a petrofizikai paraméterek szerinti első deriváltja:

$$\psi_{ij} = \frac{\partial d_i}{\partial p_j}\frac{p_j}{d_i},$$

ahol d_i a mért mélyfúrási geofizikai adatrendszer *i*-edik adata, p_j pedig a petrofizikai modell *j*-edik paramétere. A fenti mennyiség számításához (1.9.1) szerinti numerikus deriváltat fogjuk felhasználni, az adatokat pedig az elméleti-szonda válaszfüggvénnyekkel közelítjuk. Legyen a feladatunk a következő. Számítsuk ki a mikrolaterolog szondával mért *RS* fajlagos ellenállás adatok agyagtartalomra (*VSH*) vonatkozó érzékenységi függvényét a ($0 \le VSH \le 1.0$) tartományban az alábbi válaszfüggvény (Indonéziai-egyenlet) segítségével

$$\frac{1}{RS^{1/2}} = \left\{ \frac{VSH^{(1-VSH/2)}}{RSH^{1/2}} + \frac{POR^{BM/2}}{(BA \cdot RMF)^{1/2}} \right\} SX0^{BN/2},$$

ahol az 5. táblázat szerint definiált függvénykonstansok ismert mennyiségek. A fenti függvény maximuma arról tájékoztat, hogy a fajlagos ellenállás-szonda mely agyagtartalomnál a legérzékenyebb a kőzetben lévő agyag jelenlétére.

5. táblázat. Függvénykonstansok

Függvénykonstans	**Érték**	**Jelentés**
RSH	2.5 Ωm	Agyag fajlagos ellenállása
POR	0.1	Porozitás
BA	1	Tortuozitási tényező
RMF	2 Ωm	Iszapfiltrátum fajlagos ellenállás
BM	2	Cementációs tényező
SX0	1.0	Kiöblített zóna fajlagos ellenállása
BN	2	Szaturációs tényező

A script fájlban futtatása után az érzékenységi függvény alapján adjuk meg, mely értéknél lesz az ellenállásmérés az agyagtartalomra a legérzékenyebb (a megoldás VSH=0.35, azaz 35% agyagtartalomnál). Ezzel a program listája a következő:

```
%-------------------------------------------------
% RS szelvény agyagtartalom érzékenysége
%-------------------------------------------------
clc;
```

```
clear;
rsh = 2.5;
rmf =  2;
n  =  2;
m  =  2;
a  =  1;
sx0 =  1;
por =  .1;
h=0.01;                          % h (1.9.1)-ből
pszi=zeros(101,1);               % érzékenységi függvényértékek vektora
vsh=(0:0.01:1)';                 % értelmezési tartomány
for i=1:length(vsh)
    vsh1=vsh(i)+h;
    vshtag=vsh1^(1-(vsh1/2));
    rs1=(vshtag/sqrt(rsh)+(por^(m/2))/(sqrt(a*rmf)))*(sx0^(n/2));
    rs1=1/(rs1*rs1);
    vsh2=vsh1-h;
    vshtag2=vsh2^(1-(vsh2/2));
    rs2=(vshtag2/sqrt(rsh)+(por^(m/2))/(sqrt(a*rmf)))*(sx0^(n/2));
    rs2=1/(rs2*rs2);
    pszi(i)=(((rs2-rs1)/(2*h))*(vsh1/rs1));
end
[e,j]=max(pszi);
max_erzekenyseg=vsh(j),
```

A MATLAB egyváltozós függvények *határozott integrál*-jának (görbe alatti terület) számítására az összetett Simpson-formulát ajánlja. A Simpson-formula *n* számú részintervallum esetén ($a=x_1< x_2<\ldots< x_{n+1}=b$)

$$\int_a^b f(x)dx \cong \sum_{i=1}^{n} \frac{x_{i+1}-x_i}{6}\left(f(x_i) + 4f\left(\frac{x_i+x_{i+1}}{2}\right) + f(x_{i+1})\right). \qquad (1.9.2)$$

A fenti formulát alkalmazhatjuk a **quad** függvény hívásával, melynek argumentumában meg kell adni az integrandusz definícióját tartalmazó function nevét, az integrálás alsó és felső határát, és az előírt pontosságot. A MATLAB 5.3 verzióval bezárólag lehetőségünk van az integráláshoz felvett pontok grafikus ábrázolására is, melyek száma az előírt hibahatártól függ. Ennek az ábrának a nevét egy string megadásával rögzíthetjük a paraméterlista utolsó elemeként. A paraméterlista további elemeket is tartalmazhat, melyek az integranduszt definiáló function paraméterlistájában esetlegesen szereplő további változók hivatkozására szolgálnak. Példaként integráljuk a szinusz-függvényt a $[0,\pi]$ intervallumban 10^{-5} pontosság mellett:

```
>> quad('sin',0,pi,1e-5)
ans =
   2.0000.
```

Megjegyezzük, hogy a számok kiíratása alapértelmezés szerint négy tizedesig történik. Ha kíváncsiak vagyunk pl. a további 15 tizedesjegyre, akkor a **format long** utasítással output

formátumot válthatunk. Az alapbeállításra a **format short** paranccsal térhetünk vissza. A formátumok részletes listája a HELP-ben megtalálható.

A kettős határozott integrálok (felület alatti térfogat) számítása is egyszerűen végrehajtható a MATLAB **dblquad** beépített függvényével. Egy tetszőleges kétváltozós függvény határozott integráljának formulája:

$$V = \int_{y_{min}}^{y_{max}} \left\{ \int_{x_{min}}^{x_{max}} f(x,y)dx \right\} dy . \qquad (1.9.3)$$

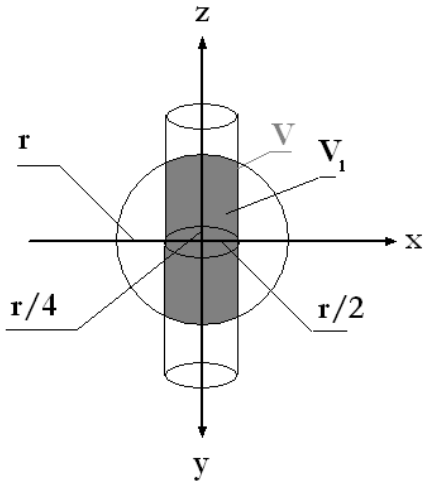
A számítás két lépésben történik, először a belső integrált értékeljük ki, melynek eredménye olyan függvény, amely csak az *y* változótól függ. Ez azt jelenti, hogy a felület egymással párhuzamos *xz* síkokra vonatkozó metszetgörbéit diszkrét *y* értékeknél *x* szerint integráljuk (az *y* pontok száma a Simpson-formulában megadott pontosságtól függ, ez 1.9.2-ben az *n* szám). Ezen y=konstans értékekhez hozzárendeljük az f(x,y=konstans) metszetgörbék alatti területek értékeit, majd a kapott függvényt *y* szerint integráljuk. Pl. számítsuk ki az $f(x,y) = e^{(x+y^2)}$ felület és az *xy*-sík közötti térfogatot azon a tartományon, ahol x_{min}=-2, x_{max}=1, y_{min}=-1, y_{max}=1. E feladatot a MATLAB-ban kétféleképpen is megoldhatjuk: saját program írásával, vagy beépített függvény alkalmazásával. Először tekintsük az első megoldást. Ehhez szükség van az f(x,y) függvényt tartalmazó function-re (*fxy.m*), továbbá egy újabb függvényre mely a belső (*x* szerinti) integrálást végzi (*intdx.m*), végül egy script fájlra mely a külső (*y* szerinti) határozott integrált értékeli ki (*doubleint.m*):

```
%--------------------
%   fxy.m
%--------------------
function f=fxy(x,y)
    f=exp(x+y*y);
return

%--------------------
%   intdx.m
%--------------------
function z=intdx(y,fxy,xmin,xmax)
    n=length(y);
    for i=1:n
            z(i)=quad(fxy,xmin,xmax,[],[],y(i));
    end
return

%--------------------
%   doubleint.m
%--------------------
clc;
clear;
xmin=-2;
xmax=1;
ymin=-1;
ymax=1;
V=quad('intdx',ymin,ymax,[],[],'fxy',xmin,xmax),
```

Ugyanez a feladat a beépített dblquad függvénnyel azonos eredményre vezet. Itt a függvény argumentumában az f(x,y) függvényt tartalmazó function nevét és az integrálási határokat kell megadni (x_{min}, x_{max}, y_{min}, y_{max} sorrendben):

```
» V=dblquad('fxy',-2,1,-1,1),
V =
   7.5560.
```

A gyakorlatban sokszor előfordul, hogy a határozott integrálokat változó határok között kell kiszámítani. Tekintsük a következő feladatot. Legyen adva egy *r* sugarú gömb, valamint egy a=r/2 és b=r/4 féltengelyű ellipszis alapú henger. A kérdés az, hogy a 6. ábra szerinti elrendezésben mekkora térfogatot (*V*) metsz ki a henger a gömbből.



6. ábra. A gömbből az ellipszis alapú henger által kimetszett térfogat

Először tisztázzuk az integrálási tartományt. Nyilvánvaló, hogy a szimmetria miatt a kérdéses *V* térfogat megegyezik a V_1 térfogat kétszeresével. Az *y* szerinti integrálást –b-től b-ig, az *x* változó szerinti integrálást pedig y=0 esetben –a-tól a-ig, y≠0 esetén az ellipszis görbéjének megfelelő két pontja között kell elvégezni. Ez utóbbi esetben ki kell fejezni az integrálási tartomány x(y) alakú egyenletét, mely ellipszis esetén a következő:

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1, \quad x=\pm\sqrt{a^2\left(1-\frac{y^2}{b^2}\right)}. \tag{1.9.3}$$

Figyelembe véve (1.9.3) integrálási határokat, ill. azt, hogy az integranduszt a gömb egyenletéből ($r^2=x^2+y^2+z^2$) kifejezhetjük, az (1.9.3) alakú határozott integrál adja meg a kérdéses térfogatot:

$$V=2\int_{-a}^{a}\int_{-\sqrt{a^2\left(1-\frac{y^2}{b^2}\right)}}^{\sqrt{a^2\left(1-\frac{y^2}{b^2}\right)}}\sqrt{r^2-x^2-y^2}\,dxdy\ .$$

A fenti térfogat számítására írjunk MATLAB programot, mely az r sugár értékét a felhasználótól kéri be. A megoldás hasonló a doubleint.m programhoz, azzal a különbséggel, hogy az *x*-hez tartozó változó integrálási határokról két külön függvényben kell

gondoskodnunk. Az *y* értékhez tartozó also határt az *also.m*, a felso határt a *felso.m* fájlban számítsuk. Az f(x,y,r) függvényt tartalmazó function neve legyen *fxyr.m*, az *x* szerinti integrálást *intdx.m* végezze el, majd az *y* szerinti integrált *doubleint2.m* értékelje ki:

```
%--------------------
%   fxyr.m
%--------------------
function f=fxyr(x,y,r)
    f=sqrt((r*r)-(x.*x)-(y*y));
return

%--------------------
%   also.m
%--------------------
function x=also(y,a,b)
    x=-sqrt((a^2)*(1-(y^2/b^2)));
return

%--------------------
%   felso.m
%--------------------
function x=felso(y,a,b)
    x=sqrt((a^2)*(1-(y^2/b^2)));
return

%--------------------
%   intdx.m
%--------------------
function z=intdx(y,fxyr,also,felso,a,b,r)
n=length(y);
for i=1:n
    tol=feval(also,y(i),a,b);
    ig=feval(felso,y(i),a,b);
    z(i)=quad(fxyr,tol,ig,[],[],y(i),r);
end
return

%--------------------
%   doubleint2.m
%--------------------
clc;
clear;
r=input('r=');
a=r/2;
b=r/4;
V=2*quad('intdx',-b,b,[],[],'fxyr','also','felso',a,b,r),
```

A hármas (térfogat felett értelmezett) határozott integrálok számítása a MATLAB-ban csak a 7-es verzió felett megoldott. A **triplequad** beépített függvény paraméterei csupán a harmadik változó alsó és felső határainak értékeivel bővül ill. az argumentum első paramétere a @-al

bevezetett function név, amely az integranduszt tartalmazza. A geofizikában elsősorban az előremodellezési feladatok megoldása során gyakori, hogy valamely fizikai mennyiségnek az (x,y,z) térkoordinátákkal leírt tartomány feletti integráljára vagyunk kíváncsiak (pl. elektromágneses térkomponensek térfogati integrálja stb.). E feladatot hasonlóan a kettős integrál kezeléséhez a quad függvény többszöri alkalmazásával oldhatjuk meg. A határok természetesen nemcsak konstansok lehetnek, hanem függvények is, melyekkel igen bonyolult tértartományok is körülhatárolhatók. Példaként írjunk saját programot a következő térfogati integrál számítására

$$I = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \sqrt{x^2+y^2+z^2}\, e^{\sqrt{x^4+y^2+z^2}}\, dx\,dy\,dz.$$

Legyen az integrandusz az u(x,y,z) függvény, melyet a *fxyz.m* function-ben definiáljunk. Először az *x* szerint integrálást végezzük el (*intdx*), majd a kapott $u^*(y,z)$ függvényt *y* szerint (*intdy*), végül *z* szerint integráljuk (*tripleint.m*). A programba globális változókat szükséges a *z* szerint integrálás során deklarálni, melyek értékeit a scriptben és a függvényekben is szabadon el kell érnünk. E változókat a **global** utasítás után feltüntetve minden fájlban deklarálni kell, ahol használatuk szükséges. Ezzel a fenti feladat megoldása:

```
%---------------------
%     fxyz.m
%---------------------
function u=fxyz(x,y,z0)
global z0
global i
    u=sqrt(x.^2+y.^2+z0(i).^2).*exp(sqrt(x.^4+y.^2+z0(i).^2));
return

%--------------------
%     intdx.m
%--------------------
function z=intdx(y,fxyz,xmin,xmax)
    n=length(y);
    for i=1:n
            z(i)=quadl(fxyz,xmin,xmax,[],[],y(i));
    end
return
%--------------------
%     intdy.m
%--------------------
function w=intdy(z,fxyz,xmin,xmax,ymin,ymax)
global z0
global i
    n=length(z);
    z0=z;

    for i=1:n
            w(i)=quad('intdx',ymin,ymax,[],[],fxyz,xmin,xmax);
    end
return
```

```
%--------------------
%    tripleint.m
%--------------------
clc;
clear;
xmin=-1;
xmax=1;
ymin=xmin;
ymax=xmax;
zmin=xmin;
zmax=xmax;
I=quad('intdy',zmin,zmax,[],[],'fxyz',xmin,xmax,ymin,ymax),
```

A feladat megoldása a triplequad függvény alkalmazásával a következő:

```
%--------------------
%     fxyz2.m
%--------------------
function u=fxyz2(x,y,z)
    u=(sqrt(x.^2+y.^2+z.^2).*exp(sqrt(x.^4+y.^2+z.^2)));
return

>> I=triplequad(@fxyz2,-1,1,-1,1,-1,1),
I =
  20.9102.
```

1.10 A MATLAB GRAFIKUS LEHETŐSÉGEI

A geoinformatikai adatok grafikus megjenítése a MATLAB-ban objektum-orientált formában történik. A két- és háromdimenziós függvények adatainak ábrázolása során grafikus felhasználói felülettel rendelkező ún. *Figure* objektumot hozunk létre, melynek grafikus elemeit (pl. függvény, tengelyek, vonaltípus, szöveg stb.) programozhatjuk, ill. egyes tulajdonságait (pl. 3 dimenziós forgatás, magyarázó felirat beszúrása stb.) programozás nélkül egy kezelői felületen is szerkeszthetjük. A teljes grafikus objektumot **.fig* kiterjesztéssel el is menthetjük, és a MATLAB újraindítása után újra előhívhatjuk a COMMAND WINDOW-ban a FILE menü OPEN parancsával, és igény szerint módosíthatjuk. A Figure főmenüjében az EDIT menü alatt a COPY FIGURE paranccsal az ábrát a vágólapra helyezhetjük, majd azt windows metafile vagy bitmap formátumban dokumentumba illeszthetjük.

A következőkben az alapvető két- és háromdimenziós grafikus ábrázolással ismerkedünk meg. Egyváltozós függvények ábrázolására a **plot** utasítás alkalmas. E függvény első paramétere a független változó-, második paramétere a függő változó értékeit (diszkrét adatait) tartalmazó vektor. E két vektor dimenziójának meg kell egyezni, azonban az nem feltétel, hogy mindkét vektor egyformán oszlop vagy sorvektor legyen. A plot utasítás további paraméterei a vonaltípus és szín megadását szolgálja (ld. HELP). Egy ábrán több függvényt is ábrázolhatunk ezt a **hold** utasítás bekapcsolásával (hold on) érhetjük el (a kikapcsolás hold off-al történik). A tengelyértékek módosítását az **axis** utasítással érhetjük el, melynek argumentumában egy sorvektor szerepel. Ez a vektor tartalmazza az ábrázolni kívánt minimális és maximális abszcissza, valamint a minimális és maximális ordináta értékeket. Ha

rácshálót kívánunk fektetni a koordináta-rendszerre, azt a **grid** paranccsal tehetjük meg (bekapcsolása grid on, kikapcsolása grid off). Az ábrára feliratokat is illeszthetünk. A **title** utasítás címet ad az ábrának, az **xlabel** és az **ylabel** utasítások pedig a tengelyeket címezik meg. Ezen kívül a **text** utasítás az ábra tetszőleges koordinátájú pontjába szöveget szúr be. A grafikus ábra minden eleme egy-egy objektumot képvisel. Az objektum tulajdonságait a **set** paranccsal módosíthatjuk (a módosításhoz definiálnunk kell az adott objektum azonosítóját).

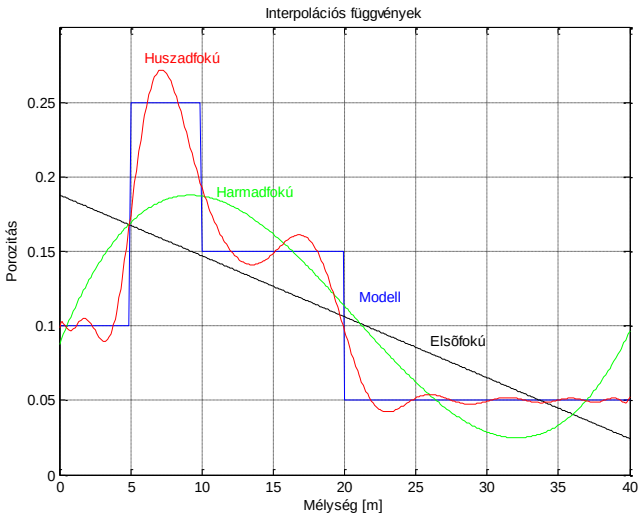
A *kétdimenziós ábrázolás*-t lehetővé tevő plot utasítás alkalmazására példaként illesszünk egy adatsorra polinomokat, majd ábrázoljuk azokat. Az interpolációt egy négyréteges kőzetfizikai modellen végezzünk el, ahol a független változó a mélység *(z)*, a függő változó pedig a rétegek porozitása *(Φ)*:

$$\Phi = \begin{cases} 0.1\ , 0 \le z \le 5\text{m} \\ 0.25, 5 < z \le 10\text{m} \\ 0.15, 10 < z \le 20\text{m} \\ 0.05, 20 < z \le 40\text{m}. \end{cases}$$

A porozitás eloszlást Δz=0.1 m lépésközzel adjuk meg, az interpolációra alkalmazzunk elsőfokú, harmadfokú, majd huszadfokú polinomot. A polinom interpoláció számítására beépített MATLAB függvényekkel is dolgozhatunk. A **polyfit** függvény megadja a polinom együtthatókat, melynek első paramétere a független változó értékeit-, második paramétere a függő változó értékeit tartalmazó vektor, továbbá utolsó paramétere az illesztett polinom fokszáma. A függvény a polinom együtthatóit tartalmazó vektorral tér vissza, ahol a vektor első eleme a legnagyobb fokszámhoz tartozó együtthatót jelenti. Az együtthatóvektor birtokában a **polyval** függvénnyel értékelhetjük ki az interpolációs polinomot a független változó függvényében. A fenti feladat megoldása a 7. ábrán látható, és a programlista a következő:

```
%----------------------------
% Polinom interpoláció
%----------------------------
clc; clear;
%
% Interpoláció
%
z=[0:0.1:40]';
por=zeros(0:0.1:40,1);
por(1:50,1)=0.1;
por(51:100,1)=0.25;
por(101:200,1)=0.15;
por(201:401,1)=0.05;
eh_1=polyfit(z,por,1);
eh_2=polyfit(z,por,3);
eh_3=polyfit(z,por,20);
poli_1=polyval(eh_1,z);
poli_2=polyval(eh_2,z);
poli_3=polyval(eh_3,z);
%
% Ábrázolás
```

```
%
plot(z,por);
tengely=[0,40,0,0.3];
axis(tengely);
hold on;
plot(z,poli_1,'k');
hold on;
plot(z,poli_2,'g');
hold on;
plot(z,poli_3,'r');
grid on;
title('Interpolációs függvények');
xlabel('Mélység [m]');
ylabel('Porozitás');
text_1=text(21,0.12,'Modell');
text_2=text(26,0.09,'Elsőfokú');
text_3=text(11,0.19,'Harmadfokú');
text_4=text(6,0.28,'Huszadfokú');
set(text_1,'Color','Blue');
set(text_2,'Color','Black');
set(text_3,'Color','Green');
set(text_4,'Color','Red');
```

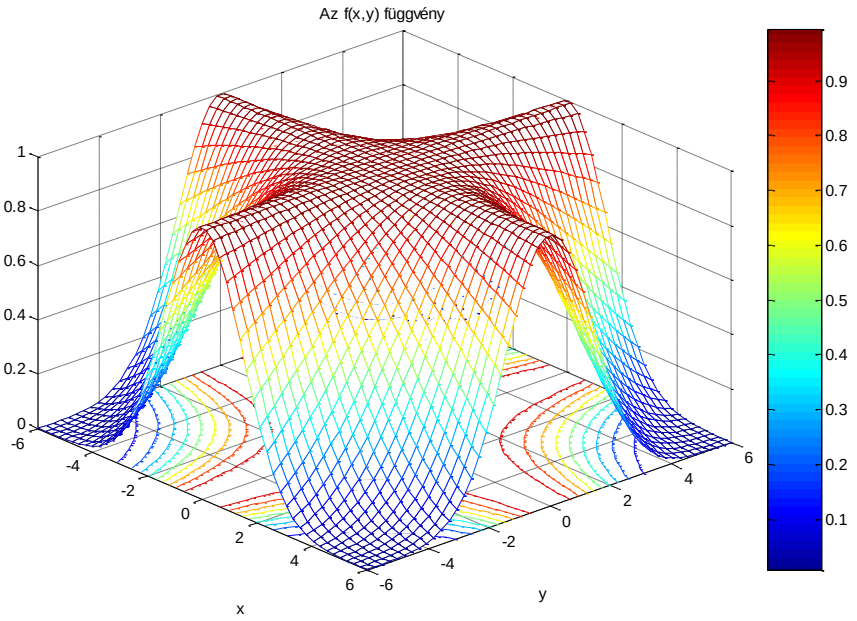


7. ábra. Példa a plot utasítás alkalmazására

A *háromdimenziós ábrázolás* sokféleképpen megoldható a MATLAB rendszerben. Különböző koordináta-rendszerekben (Descartes, polár, komplex) hagyományos (felület és kontúrtérkép) és speciális ábrázolások lehetségesek (pl. vektortérkép, térgörbék, 4D animáció). Ebben a részben néhány a geoszakemberek számára fontos grafikával ismerkedünk meg. Elsőként lássuk a hálógrafikon szerkesztést. Ehhez elsőként egy rácshálót kell létrehoznunk, melynek pontjait egy mátrixban tároljuk. A mátrixelemekhez, mint a rácsháló pontjaihoz kell hozzárendelni a háromdimenziós felületet definiáló kétváltozós függvény helyettesítési értékeit. A rácsháló létrehozásához a koordináta-tengelyeket nem szükséges egyenközűen felosztani, azonban az osztások számának mindkét (független változót tartalmazó) tengely esetén meg kell egyezni. Az egyenközű osztást a **linspace** függvénnyel

gyorsan elvégezhetjük, melynek paraméterei a felosztandó intervallum minimális és maximális értéke, valamint az osztások száma az adott intervallumon. A rácshálót a **meshgrid** függvény automatikusan generálja. A hálógrafikont végül a **mesh** vagy a **meshc** paranccsal ábrázolhatjuk (a meshc a független változók által megadott síkon izovonalakkal egészíti ki a grafikus objektumot). További lehetőség, hogy a felület elforgatható a koordináta-tengelyek körül. A **view** parancs adott nézőpontból engedi láttatni az ábrát. Ennek első paramétere a z-tengely körüli elforgatás szöge (óramutató járásával ellentétes értelemben), második paramétere az xy-síktól számított emelkedési szög (mindkét mennyiség fokban értendő). Megjegyezzük, hogy a grafikus szerkesztőablakon a forgatás interaktív módon is elvégezhető, továbbá egyéb objektumok is beszúrhatók az ábrába, mint pl. a színes értékskála (**color bar**), jelmagyarázat (**legend**) stb. Példaként ábrázoljuk az $f(x,y)=e^{-(xy)^2/150}$ függvényt a [-6,6] intervallumon (az intervallumot egyenletesen osszuk fel 50 részre). A z-tengely körüli elforgatás szöge legyen 48°, az emelkedési szög pedig 35°. Adjunk színskálát az ábrához.

```
%-------------------------------
%  Hálógrafikon ábrázolása
%-------------------------------
x=linspace(-6,6,50);
y=x;
[X,Y]=meshgrid(x,y);
f=exp(-(1/150)*(X.*Y).^2);
meshc(x,y,f);
title('Az f(x,y) függvény');
xlabel('x');
ylabel('y');
view(-48,35);
```



8. ábra. Példa a meshc utasítás alkalmazására

A fentiek ismeretében most már összetett grafikus feladatokat is meg tudunk oldani. Gyakran szükség lehet pl. több ábra egy grafikus felületen való ábrázolására. A **subplot** utasítással n×m db ábrát helyezhetünk el egy időben a felületen. A parancsszó első két

paramétere az n-et és az *m*-et rögzíti, a harmadik paraméter pedig az aktuális ábra sorszámát jelenti. A feladat legyen a következő. Adott az $F(\vec{x})$ vektorfüggvény két egyenlettel:

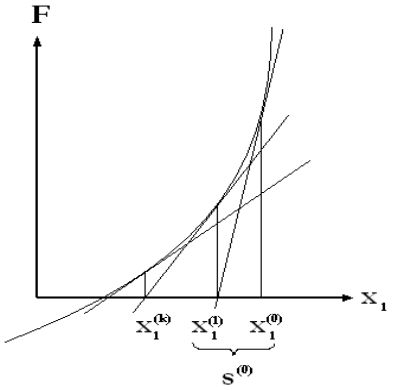
$$\left.\begin{aligned} f_1(x_1,x_2) &= x_2 - 2x_1^2 \\ f_2(x_1,x_2) &= x_2 - 2 - e^{x_1} \end{aligned}\right\},$$

ábrázoljuk külön grafikus ablakba a két felületet, majd határozzuk meg Newton-módszerrel, hogy az $F(\vec{x})=0$ síkban mely (x,y) koordinátájú pontban metszi egymást a két görbe. Ábrázoljuk a metszetgörbéket, valamint az iterációs lépések megoldásait (csillagokkal jelölve) az (x1-x2) koordináta-rendszerben, végül adjunk jelmagyarázatot és színskálát az ábrákhoz és tüntessük fel az ábrán a megoldás koordinátáit. A feladat megoldásához fejtsük Taylor-sorba az $F(\vec{x})$ függvényt $\vec{x}^{(0)} = \{x_1^{(0)}, x_2^{(0)}\}$ kiindulási vektor körül (lineáris taggal bezárólag):

$$F(\vec{x}) \cong F(\vec{x}^{(0)}) + \left.\frac{\partial F}{\partial x}\right|_{\vec{x}^{(0)}} (\vec{x} - \vec{x}^{(0)}), \qquad (1.10.1)$$

ahol $\underline{\underline{J}}^{(0)} = \left.\frac{\partial F}{\partial x}\right|_{\vec{x}^{(0)}}$ a Jacobi-mátrix. Az $F(\vec{x})$ függvény zérushelyét iteratív Newton-módszerrel keressük meg. A 9. ábrán látható, hogy a keresés az érintő bevonásával történik, így az (1.10.1) összefüggés a *k+1*-edik közelítésben $\vec{x}^{(k+1)} = \vec{x}^{(k)} + \vec{s}^{(k)}$, mellyel a zérushely:

$$F(\vec{x}^{(k+1)}) \cong F(\vec{x}^{(k)}) + \underline{\underline{J}}^{(k)} \vec{s}^{(k)} = 0 .$$



9. ábra. Newton módszer egyváltozós esetben

A fenti feladat algoritmusa tehát:

1. $\vec{s}^{(k)} = -\underline{\underline{J}}^{(k)} \vec{F}$ meghatározása,
2. $\vec{x}^{(k+1)} = \vec{x}^{(k)} + \vec{s}^{(k)}$ felülírása,
3. $\|\vec{s}\|_\infty < \varepsilon$ és k<kmax stop-kritérium vizsgálata.

A feladat egy lehetséges megoldása: a vektorfüggvényt a *felulet.m* function-ben definiáljuk. A számításhoz szükség van a Jacobi-mátrixra, melynek elemeit úgy számítjuk, hogy a két

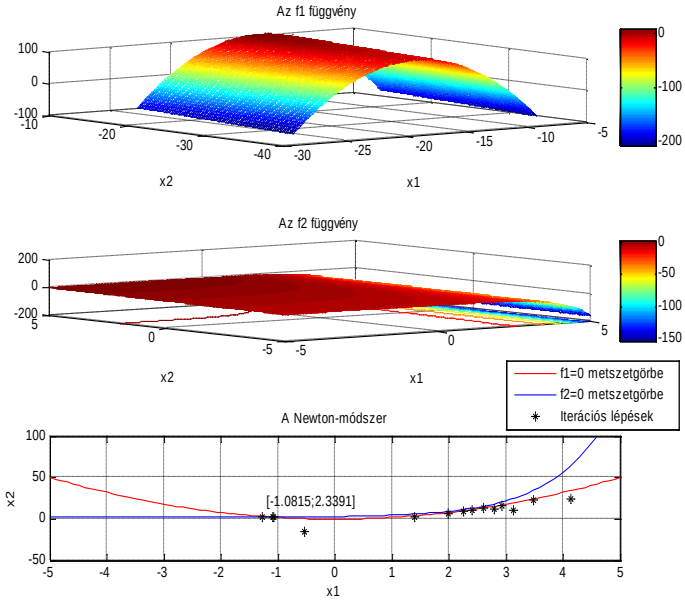
függvényt minden egyes változó szerint külön differenciáljuk. Tekintve, hogy a függvények egyszerűek, ezért a deriválást analitikusan is elvégezhetjük, melynek eredményét a *jacobi.m* function-ben tároljuk. A lineáris egyenlet megoldását a *newton.m* function-ben tároljuk, melyben egy while-ciklus gondoskodik a stop-kritérium teljesüléséről. A newton.m függvény hívását és a grafikák elkészítését az *abrazol.m* script végzi el. A koordináta-értékeket a text utasítással írathatjuk ki, melyhez a megoldás vektor elemeit a **num2str** paranccsal külön-külön karakter típussá transzformáljuk, majd az **strcat** utasítással összefűzzük őket. A jelmagyarázatot és a színskálákat a Figure objektumhoz interaktív módon adjuk hozzá. A feladat megoldása a 10. ábrán látható, a program listája pedig a következő:

```
%------------------
%   felulet.m
%------------------
function F=felulet(x)
    n=length(x); f=zeros(n,1);
    f(1)=-2*x(1)^2+x(2);
    f(2)=-2-exp(x(1))+x(2);
    F=[f(1);f(2)];
return

%------------------
%   jacobi.m
%------------------
function J=jacobi(x)

    J(1,1)=-4*x(1);
    J(1,2)=1;
    J(2,1)=-exp(x(1));
    J(2,2)=1;
return

%------------------
%   newton.m
%------------------
function [x,iteracio,x1,x2]=newton(felulet,x0,jacobi,epsilon,itermax)
    norma=inf; iteracio=0;
    while norma>=epsilon & iteracio<=itermax
            J=feval(jacobi,x0);
            F=feval(felulet,x0);
            s=J\(-F);
            x0=x0+s;
            x1(iteracio+1,1)=x0(1);
            x2(iteracio+1,1)=x0(2);
            norma=norm(s,inf);
            iteracio=iteracio+1;
    end
    disp('A megoldás és az iteráció:');
    x=x0;
return
```

```
%------------------
%   abrazol.m
%------------------
clc;clear;
%
% Szamítás
%
[x,iteracio,x1_iter,x2_iter]=newton('felulet',[1;1],'jacobi',1e-10,20);
x,
iteracio,
%
% Grafika
%
subplot(3,1,1);
x1=linspace(-10,10,100);
x2=x1;
[X1,X2]=meshgrid(x1,x2);
f1=X2-2*(X1.^2);
meshc(x1,x2,f1);
axis([-30,-5,-40,-10,-100,100]);
xlabel('x1');
ylabel('x2');
title('Az f1 függvény');
subplot(3,1,2);
x1=linspace(-5,5,100);
x2=x1;
[X1,X2]=meshgrid(x1,x2);
f2=X2-2-exp(X1);
meshc(x1,x2,f2);
xlabel('x1');
ylabel('x2');
title('Az f2 függvény');
subplot(3,1,3);
x1=linspace(-5,5,100);
x2=2*x1.^2;
plot(x1,x2,'r');
hold on;
x2=2+exp(x1);
plot(x1,x2,'b');
axis([-5,5,-50,100]);
hold on;
grid on;
plot(x1_iter,x2_iter,'k*');
title('A Newton-módszer');
xlabel('x1');
ylabel('x2');
koord1=num2str(x(1));
koord2=num2str(x(2));
koord=strcat('[',koord1,';',koord2,']');
text(-1.2,20,koord);
```

A feladat megoldása a parancsablakból való futtatás után:

A megoldás és az iteráció:
x =
-1.0815
2.3391
iteracio =
17



10. ábra. Newton módszer megoldása

Oldjuk meg a következő összetett grafikus feladatot, mely az optimalizáció elmélet egy modern fejezetére, az abszolút (globális) szélsőérték kereső eljárásokra épül. Legyen E(x,y) az optimalizálandó célfüggvény, melynek független változói egy modellt képeznek. Célunk az (x,y) modellparaméterek megadása azon a helyen, ahol a célfüggvény értéke minimális. A globális szélsőérték-keresés célja az, hogy az E függvény sok lehetséges lokális minimuma közül kiválassza az egyetlen abszolút minimumot. Ezt a feladatot többféle módszerrel is megoldhatjuk. Az egyik legmodernebb eljárás az ún. Simulated Annealing eljárás, mely esetében kimutatható, hogy az optimumhoz történő konvergencia során az *i*-edik modellre számított $E^{(i)}=E(x^{(i)},y^{(i)})$ célfüggvény *P* valószínűsége a Gibbs-féle eloszláshoz tart:

$$P(E^{(i)}) = \frac{\exp\left(-\frac{E^{(i)}}{T}\right)}{\sum_{j=1}^{M}\exp\left(-\frac{E^{(j)}}{T}\right)}, \qquad (1.10.2)$$

ahol *M* a modellek száma, *T* pedig az általánosított hőmérséklet. A hőmérséklet az optimalizációs eljárás lényges vezérlőparamétere, mely a modell elfogadását szabályozza. A feladat legyen a következő. Mutassuk meg grafikusan, hogy *T* mely értékeinél adja meg (1.10.2) valószínűség-sűrűségfüggvény egyértelműen a célfüggvény globális minimumát. Legyen adott egy f(x,y) kétváltozós függvény a következőképpen:

$$f(x,y)=\operatorname{sgn}\left(\frac{\sin x}{x}\right)\left(\left|\frac{\sin x}{x}\right|\right)^{\frac{1}{4}}\operatorname{sgn}\left(\frac{\sin y}{y}\right)\left(\left|\frac{\sin y}{y}\right|\right)^{\frac{1}{4}},$$

melynek számos lokális maximuma, viszont az x=0 és y=0 pontban egyetlen globális maximuma van. Képezzük a minimumkeresés számára a következő alkalmas célfüggvényt f(x,y) abszolút maximumának megtalálása érdekében:

$$E(x,y)=(1-f(x,y))^2 \to \min .$$

Ábrázoljuk egy grafikus ablakban az f(x,y), E(x,y) függvényeket, valamint az (1.10.2) valószínűség-sűrűségfüggvényt T=[10, 1, 0.1, 0.01[értékek mellett. Az (1.10.2) függvény számítására hozzunk létre egy külön function-t *gibbs.m* néven, ugyanis azt négyszer kell hívnunk majd a program során. A főprogram neve legyen *global.m*:

```
%----------------------
%    gibbs.m
%----------------------
function s=gibbs(T)
s=0;
for i=1:10
    for j=1:10
            s0=0;
            E0=0;
            t1=sin(i)./i;
            t2=sin(j)./j;
            f=sign(t1).*abs(t1).^(1/4).*sign(t2).*abs(t2).^(1/4);
            E0=(1-f).^2;
            s0=exp(-E0/T);
            s=s+s0;
    end
end
return

%------------------------
%    global.m
%------------------------
clc;
clear;
x=linspace(-10,10,100);
y=x;
[X,Y]=meshgrid(x,y);
t1=sin(X)./X;
t2=sin(Y)./Y;
f=sign(t1).*abs(t1).^(1/4).*sign(t2).*abs(t2).^(1/4);
subplot(3,2,1);
mesh(x,y,f);
title('f(x,y)');
E=(1-f).^2;
subplot(3,2,2);
```

```
mesh(x,y,E);
title('E(x,y)');
T=[10,1,0.1,0.01];
s=zeros(length(T),1);
P=zeros(length(s),1);
for i=1:length(T);
    s(i)=gibbs(T(i));
end
subplot(3,2,3);
p1=exp(-E/T(1))/s(1);
mesh(x,y,p1);
title('Gibbs pdf T=10');
subplot(3,2,4);
p2=exp(-E/T(2))/s(2);
mesh(x,y,p2);
axis([-10,10,-10,10,0,0.4]);
title('Gibbs pdf T=1');
subplot(3,2,5);
p3=exp(-E/T(3))/s(3);
mesh(x,y,p3);
title('Gibbs pdf T=0.1');
subplot(3,2,6);
p4=exp(-E/T(4))/s(4);
mesh(x,y,p4);
title('Gibbs pdf T=0.01');
```

A 11. ábra eredményéből látható, hogy minél kisebb a *T* értéke, annál határozottabban rajzolódik ki a globális szélsőérték helye. A globális optimumkeresés szempontjából ez szükséges, ugyanis amíg nagy a hőmérséklet, a modellek elfogadási valószínűsége is nagyobb. Mivel a globális optimalizációs módszerek véletlen keresést hajtanak végre a modelltérben, ezért az eljárás elején így sok modellt kipróbálhatunk. Amint később a rendszer a *T* csökkentése mellett a globális optimum felé halad, egyre kisebb lesz az elfogadási valószínűség is. Ez biztosítja, hogy ne történjen véletlen nagy ugrás a paramétertérben, és a divergencia helyett a globális szélsőértékhely felé konvergáljon az eljárás.

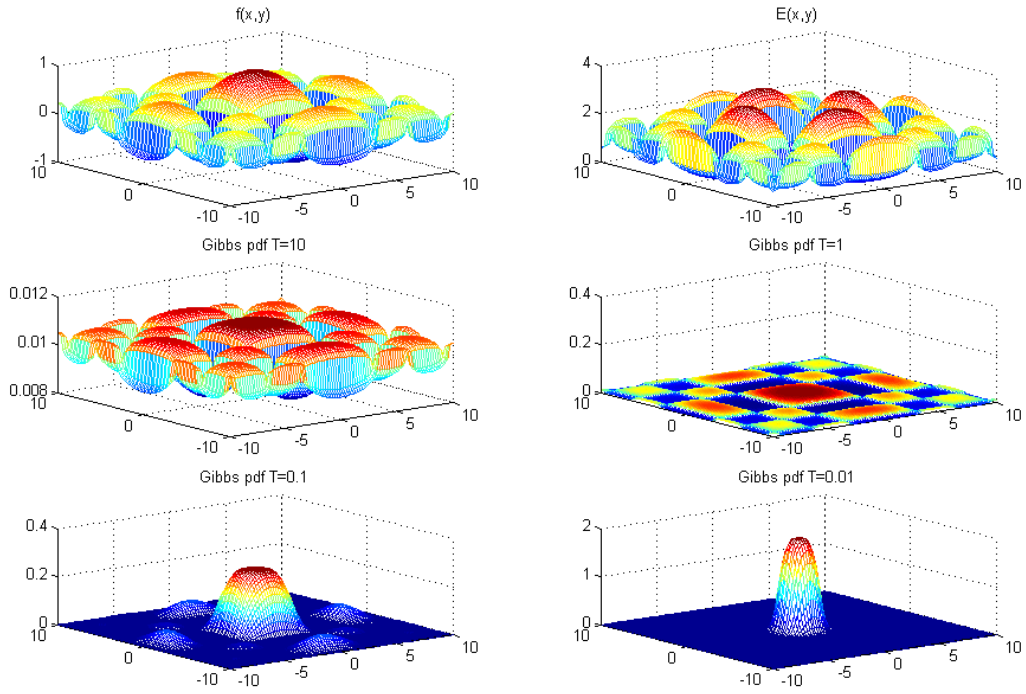
1.11 FÁJLMŰVELETEK

Az input/output utasítások fájlból a MATLAB rendszerbe adatokat olvasnak be, ill. íratnak a fájlba ki. A fájlban az adatokat szövegesen (ASCII kód) vagy binárisan tárolhatjuk. Ennek megfelelően különböző alacsony és magasabb szintű fájlművelettel operálhatunk. Az *alacsonyabb szintű input/output utasítások* körébe tartozik pl. a naplózás is, mely a **diary** utasítással fájlba írja a parancsablak tartalmát. További alacsony szintű művelet a változók mentésre szolgáló **save** utasítás, mely egy **.mat* kiterjesztésű fájlba binárisan menti a változókat. A változók betöltése a **load** utasítással történik a **.mat* fájl nevének megadásával.

A magasabb szintű fájlkezelő műveletek egyik csoportját a *bináris fájlművelet*-ek képezik. A bináris fájl létrehozását az fopen utasítással végezzük. Pl. generáljunk egy *bin1.dat* nevű fájlt, melyet bináris adattárolásra kívánunk használni:

```
» [fn1,uzenet]=fopen('bin1.dat','w')
fn1 =
     3
uzenet =
     ''
```

A fenti parancsban a fájl neve mellett *w*-vel megjelöltük, hogy írásra (write) kívánjuk használni a fájlt. Abban az esetben, ha a fájl már létezik, akkor a parancs törli annak tartalmát. Az *fn1* változó a megnyitott fájl logikai azonosítója, mely egy pozitív egész szám, ha sikeres volt a megnyitás, egyébként -1. Az *uzenet* változó sikeres megnyitás esetén üres karakterlánccal tér vissza, ellenkező esetben hibaüzenetet ad.



11. ábra. A Gibbs-eloszlás vizsgálata

Generáljuk az 5×5 elemű véletlen *K* mátrixot és annak elemeit írjuk ki binárisan a már korábban létrehozott *bin1.dat* nevű fájlba. A fájlba írást az **fwrite** utasítással végezhetjük el:

```
» adatsz=fwrite(fn1,K,'double',0)
adatsz =
      25.
```

Az *adatsz* paraméter a kiírt adatok számával tér vissza, melyek dupla pontossággal (64 biten) tárolódnak a fájlban. Az utolsó paraméter az fwrite parancs listájában megadja azt a bájt-számot, melyet ki szeretnénk hagyni az egyes adatok kiírása előtt. A fájl lezárása **fclose** utasítással történik, melynek egyetlen paramétere a fájl azonosítója (pl. *fn1*). Ezek után használjuk fel a fent létrehozott fájlt az adatbeolvasás bemutatására. A bináris fájlt hasonlóan az előzőekhez az fopen paranccsal olvasásra (read) nyitjuk meg, majd annak tartalmát egy új mátrixban tároljuk

```
» fn2=fopen('bin1.dat','r')
fn2 =
    3.
```

Az olvasásra való megnyitás az *r* paraméterrel történik, ekkor a fájlmutató a fájl elején áll. Megjegyezzük, hogy lehetőség van toldásra is megnyitni fáljokat, ekkor a mutató a fájl végén áll. Az adatok beolvasását az **fread** paranccsal végezzük, melynek paraméter listájában a fájl azonosítója után meg kell adnunk a beolvasott adatokat tartalmazó mátrix méretét is, azaz

```
» [K2,adatsz2]=fread(fn2,[5,5],'double',0)
K2 =
   0.9501   0.7621   0.6154   0.4057   0.0579
   0.2311   0.4565   0.7919   0.9355   0.3529
   0.6068   0.0185   0.9218   0.9169   0.8132
   0.4860   0.8214   0.7382   0.4103   0.0099
   0.8913   0.4447   0.1763   0.8936   0.1389
adatsz2 =
   25.
```

További utasításokat is meg kell említeni bináris fájlok kezelésével kapcsolatban. A fájlmutató pozícionálása sok esetben előfordul. A mutató aktuális pozícióját az **ftell** utasítással kérdezhetjük le. Példánkban 25 adat beolvasása után a mutató a fájl végén áll, valamint egy adatot 8 bájton tárol, így az eredmény

```
» ftell(fn2)
ans =
  200.
```

A mutatót a fájl elejére az **frewind** paranccsal pozícionálhatjuk, valamint annak tetszőleges helyzetbe való beállítását az **fseek** utasítással tehetjük meg. Ez utóbbi esetén azt meg kell adnunk, hogy az aktuális pozíciótól hány bájt-tal kell elmozdítanunk a mutatót. Itt megadhatjuk, hogy az aktuális pozíciótól (*cof*), ill. a fájl elejétől (*bof*) vagy a végétől (*eof*) kívánjuk a műveletet végrehajtani. Pl., ha az „előrecsévélt” *bin.dat* fájl-lal dolgozunk, akkor 3 adattal a fájl végének irányába a következőképpen ugorhatunk át

```
» siker=fseek(fn2,24,'cof')
siker =
    0
» ftell(fn2)
ans =
   24.
```

A fenti művelet output paramétere a művelet végrehajtásának sikerességét jelzi. Ha már nem kívánunk további fájlműveletet alkalmazni, a fájlt az fclose paranccsal be kell zárni.

A magasabb szintű fájlműveletek másik csoportját a szöveges fájlkezelő utasítások képezik. Ennek a tárolásmódnak megvan az az előnye, hogy az adatok vizuálisan is áttekinthetők, valamint közvetlenül felhasználhatók más geoinformatikai szoftverek által is (pl. Golden software Surfer, Grapher stb.). Az ASCII fájl-ok megnyitását az **fprintf** segítségével, valamint beolvasásukat az **fscanf** utasítással végezzük. Az előbbi parancs

paraméterlistájában megjelennek az adatformátumot szabályzó ún. formátum-specifikációk. Ezek tartalmazzák a legfontosabb konverziós utasításokat, melyek az adat típusát adják meg (pl. %f - lebegőpontos, %d - egész, %s - karaktertípus stb.), melyek a valós típusoknál kiegészülnek a mezőszélességet és a tizedespont utáni tizedesek számát megadó paraméterekkel (ez utóbbi megadása az adatbeolvasásnál szükségtelen). Megjegyezzük, hogy az adatok oszlop-folytonosan tárolónak, így mátrixok szöveges beolvasásánál egy transzponálás szükséges a helyes indexű mátrix elemek visszaállítása érdekében. Példaként írassuk ki lebegőpontos formátumban az *adat1.dat* ASCII típusú fájlba az 1.7 fejezetben kiértékelt $f(x)=e^{-x}\sin(x)$ függvény ($0\le x\le\pi$, $dx=\pi/15$) értékeit. Az első sor karakteresen rögzítse a mátrix elemeinek számát, a második az x, f(x), és az 1-f(x) értékek oszlopát. Az adatokat tartalmazó mátrix elemeinek kiíratását követően zárjuk be a fájlt, majd újra olvassuk be az adatokat. Hasonlítsuk össze az eredeti értékeket tartalmazó mátrix és a beolvasott mátrix tartalmát. A program a következő:

```
%-------------------------
%     ASCII I/O
%-------------------------
clc; clear;
x=[0:pi/15:pi]';
y=exp(-x).*sin(x);
u=1-y;
A=[x,y,u];
index=size(A);
f1=fopen('adat1.dat','w');
matrix=num2str(index(1)*index(2));
fprintf(f1,'A mátrix elemszáma: ');
fprintf(f1,matrix);
fprintf(f1,'\n');
fprintf(f1,'x      f(x)      1-f(x)');
fprintf(f1,'\n');
for i=1:index(1)
    for j=1:index(2)
            fprintf(f1,'%8.6f ',A(i,j));                    % adatok kiíratása fájlba
            if j==index(2) fprintf(f1,'\n'); end
    end
end
fclose(f1);
f2=fopen('adat1.dat','r');
s1=fscanf(f2,'%s',3);
elemsz=fscanf(f2,'%d',3);
s2=fscanf(f2,'%s',3);
B=zeros(index(1),index(2));
B=fscanf(f2,'%f',[index(2),index(1)]);                      % adatok beolvasása fájlból
A, B=B', A-B,
fclose(f2);
```

Az *adat1.dat* ASCII fájl kilistázható a MATLAB parancsablakán keresztül is a **type** parancs alkalmazásával (továbbá valamennyi **.m* kiterjesztésű program listája is):

» type adat1.dat

A mátrix elemszáma: 48

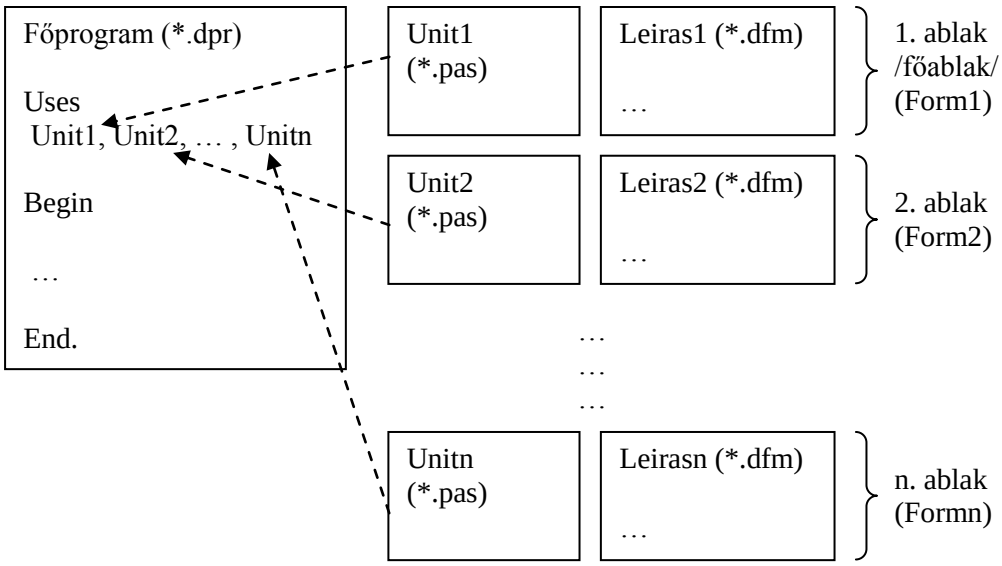
x	f(x)	1-f(x)
0.000000	0.000000	1.000000
0.209440	0.168624	0.831376
0.418879	0.267545	0.732455
0.628319	0.313576	0.686424
0.837758	0.321544	0.678456
1.047198	0.303905	0.696095
1.256637	0.270680	0.729320
1.466077	0.229565	0.770435
1.675516	0.186186	0.813814
1.884956	0.144404	0.855596
2.094395	0.106646	0.893354
2.303835	0.074222	0.925778
2.513274	0.047612	0.952388
2.722714	0.026721	0.973279
2.932153	0.011078	0.988922
3.141593	0.000000	1.000000.

2. OBJEKTUM-ORIENTÁLT PROGRAMOZÁS DELPHI FEJLESZTŐI RENDSZER ALKALMAZÁSÁVAL

A DELPHI 7 grafikus fejlesztői környezet széles körű programozási feladatok ellátására alkalmas. Elsősorban WINDOWS operációs rendszer alatt futó grafikus felhasználói felülettel rendelkező alkalmazások készítésére használható fel, azonban konzol alkalmazások készítésére is alkalmas. Programozási nyelve a Turbo Pascal továbbfejlesztett, objektum-orientált változatából kialakított Delphi nyelv. Itt érdemes megjegyezni, hogy a MATLAB rendszerrel (és a C-nyelvvel) ellentétben ez a programozási nyelv a kis és nagybetűk között nem tesz különbséget.

A DELPHI fejlesztői környezet és a nyelv alapvetően az objektum-orientált szoftverfejlesztést támogatja, melyben megmaradt a Turbo Pascal nyelvből átörökölt modulok használata, melyek segítségével hatékonyan bonthatjuk részekre a teljes alkalmazást. A szoftverfejlesztés hatékonyságának növelése érdekében a rendszer lehetővé teszi az általunk vagy már mások által előzőleg létrehozott szoftver komponensek felhasználását a saját alkalmazásaink elkészítése során. A DELPHI környezet maga is számos grafikus és nem grafikus komponenst kínál a fejlesztő számára. A komponenseket egymással kapcsolatban álló nagyszámú objektum építheti fel. Az objektumok tulajdonságait egy példán keresztül szemléltethetjük. Pl., ha egy grafikus ablakot objektumként kezelünk, akkor az objektum egyik alapvető jellemzője az ún. *adatmező,* melynek adatai az ablak megjelenését méretét, pozícióját, színét, típusát stb. definiálják. A másik csoportot az ún. *metódus*-ok képezik, melyek az adatmező adatain hajtanak végre műveleteket. Az adatmező elemeit tehát közvetlenül nem, csak metódusokon keresztül tudjuk elérni. A metódusok bizonyos csoportját az eseménykezelő eljárások képezik. Ezek megszabják azt, hogy az objektum külső és/vagy belső eseményekre hogyan reagáljon (pl. egérrel való kattintásra vonatkozó esemény stb.).

Az objektum-orientált Delphi alkalmazások felépítése általában bonyolultabb, mint a strukturált programoké. A 12. ábrán látható, hogy egy főprogramhoz különféle *modul(unit)*-ok tartozhatnak.



12. ábra. A DELPHI alkalmazások általános felépítése

A Delhi alkalmazáson belül alapvetően háromféle *fájltípus*-t különböztetünk meg:

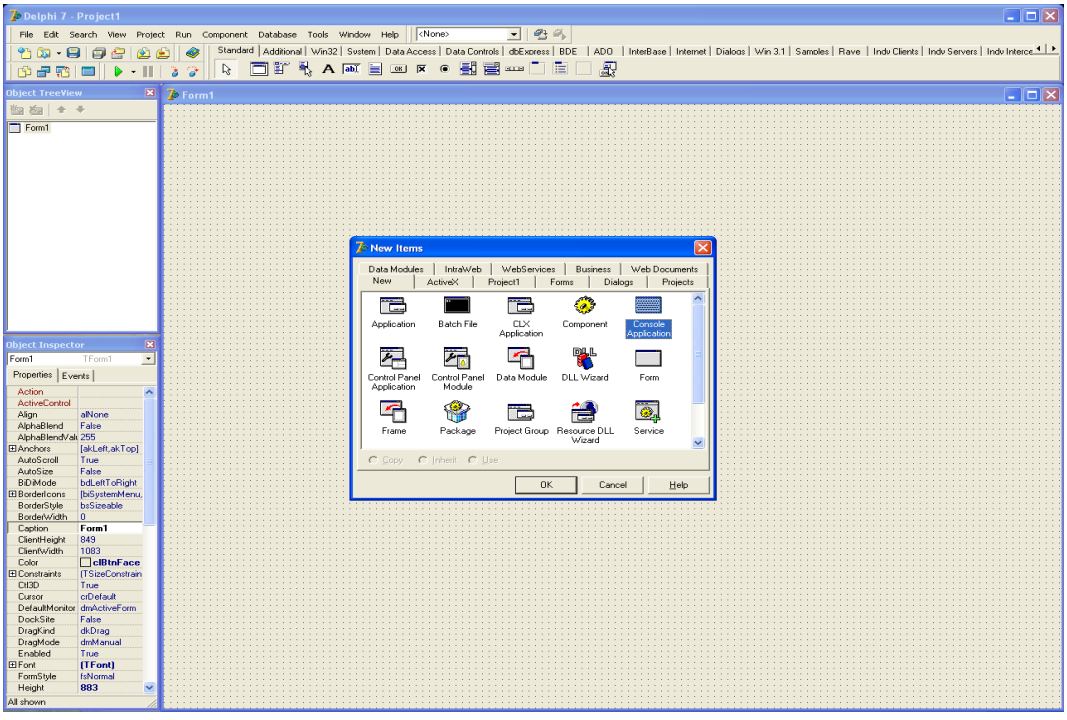
1. DPR: Projektállomány, mely a teljes programot tartalmazza. Ebből fordítjuk azonos névvel a *.exe (executable) futtatható fájl-t, mely a program DELPHI rendszerből való indítása során automatikusan generálódik.

2. DFM: Szöveges fájl, mely az ablakok paramétereinek a beállítását tartalmazza. A tervezés során ezt a fájlt szintén a rendszer generálja.

3. PAS: Ezek a tényleges Delphi nyelvű programok (Unit állományok), melyet a felhasználó ír. Az objektumok viselkedését (metódusok) írják le.

2.1 KONZOL ALKALMAZÁSOK FEJLESZTÉSE A DELPHI RENDSZERBEN

Konzol alkalmazás alatt a WINDOWS operációs rendszer MS-DOS parancsablakában futó grafikus felhasználói felülettel nem rendelkező programokat értjük. A fájlok szerkezete eltér a később tárgyalt WINDOWS-os alkalmazásokétól (ld. 2.2 fejezet). Új konzol-alkalmazást a DELPHI menürendszere alól hozhatunk létre a FILE, NEW, OTHER … menük és a CONSOLE APPLICATION ikon kiválasztásával. Ennek ablakát és a DELPHI 7 fejlesztői rendszer környezetét a 13. ábrán láthatjuk.



12. ábra. A konzol alkalmazás létrehozása

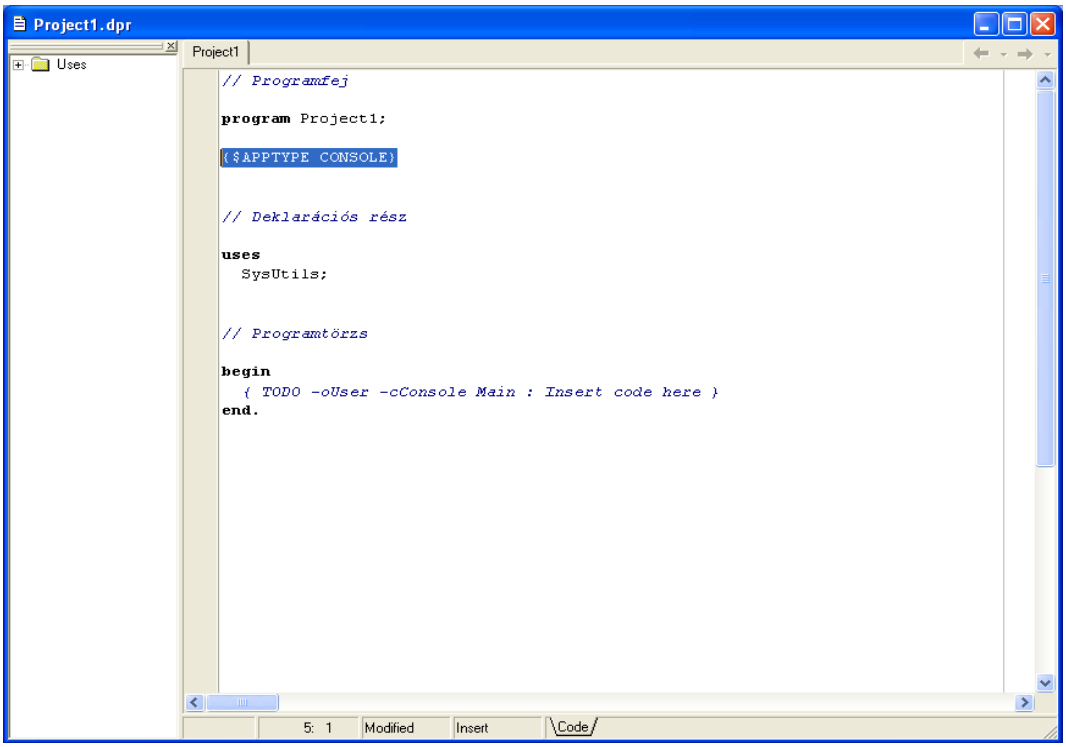
2.1.1 A konzol program szerkezete

A konzol program kiterjesztése **.dpr*, mely szerkezetileg három fő részből áll: programfejből, deklarációs részből és a program törzséből. Egy ilyen mintafájl látható a 13. ábrán (a // jel comment-et jelöl az adott sorra vonatkozóan). A *programfej* tartalmazza a

program nevét, melyet a **program** parancsszó vezet be. Ezen kívül a program egészére vonatkozó (globális hatású) fordítási direktívákat is megadja:

{$APPTYPE CONSOLE},

mely azt jelenti, hogy szöveges konzol-alkalmazást kívánunk létrehozni, és az ennek megfelelő fordítási szabályok lépnek érvénybe (a programlistából való törlése hibához vezet).



13. ábra. A konzol program szerkezeti felépítése

A *deklarációs rész*-ben fel kell sorolni a konzol program által használt erőforrásokat, változókat, függvényeket és eljárásokat. Elsőként a **uses** parancsszó után kell felsorolni a hívni kívánt modulokat, melyek a DELPHI rendszerbe előre beépített vagy általunk írt ún. Unit-ok lehetnek. Ezzel a program hozzáférését biztosítjuk a felsorolt erőforrásokhoz. Ugyancsak ebben a részben történik a címkék (**label**), konstansok (**const**), típus (**type**), változók (**var**) deklarációja. Az eljárások és függvények jelentésével később foglalkozunk (ld. 2.1.4 fejezet), azonban ezek szerkezetét a deklaráció miatt itt adjuk meg elsőként. Saját eljárás készítésénél a név, a paraméterlista elemei és a lokális változók megadása szükséges, pl.:

Procedure Név(Paraméterlista)
Var
// Lokális változók
Begin
// Utasítások;
End;

Saját függvény megadása hasonló módon történik, azzal a különbséggel, hogy a függvény neve és a paraméterlista után meg kell adni a kimenő változó típusát, valamint az értékét is:

```
Function Név(Paraméterlista):Típus;
    Var
        // Lokális változók
    Begin
        // Utasítások;
        // Név:=Utasítás;
    End;
```

Végül a *program törzs*-ében kell kifejtenünk az algoritmusunkat. Itt történik az eljárások, függvények és a változók hívása, különböző értékadó utasítások, ciklusutasítások, feltételes utasítások és egyéb műveletek elvégzése az adott programozási feladattól függően.

2.1.2 Változók deklarálása, alapvető műveletek és függvények

A DELPHI-ben (a MATLAB-tól eltérően) minden egyes változót külön-külön deklarálnunk kell. Leggyakrabban egész, valós, string és logikai típusú változókat kell használni, melyeket a **var** parancs vezet be (változódeklarációs rész). A 6. táblázat a leggyakoribb típusokat tartalmazza. E mellett a már MATLAB-nál is tanult fontosabb operátorok a DELPHI-ben is érvényesek. Ezek közül az aritmetikai operátorok precedenciája a 3. táblázatban, valamint a legfontosabb logikai és relációs operátorok a 4. táblázatban találhatók.

6. táblázat. Leggyakrabban alkalmazott deklarációk

Típus		**Értékkészlet**	**Helyfoglalás**
Egész	Integer	-2147483648, …, 2147483647	Előjeles 4 Byte
	Int64	-2^{63}, …, $2^{63}-1$	Előjeles 8 Byte
Valós	Real	$5\cdot10^{-324}$, … , $1.7\cdot10^{308}$	8 Byte
	Extended	$3.4\cdot10^{-4932}$, … , $1.1\cdot10^{4932}$	10 Byte
Karakter	String	Karakter ABC	2 GByte
Logikai	Boolean	False, True	1 Byte
	Longbool	False, True	4 Byte

A DELPHI **System** nevű modulja számos matematikai alapfüggvényt tartalmaz, mely a programozásban közvetlenül felhasználható. Ezeket a 7. táblázatban foglaltuk össze.

7. táblázat. A System modul matematikai függvényei

Függvénynév	**Jelentés**	**Paraméter típusa**	**Függvényérték típusa**
Abs	Abszolút érték	Egész, valós	Egész, valós
Sqr	Négyzet	Egész, valós	Egész, valós
Sqrt	Négyzetgyök	Egész, valós	Valós
Arctan	Arkusz-tangens	Egész, valós	Valós
Cos	Koszinusz	Egész, valós	Valós
Sin	Szinusz	Egész, valós	Valós
Exp	Exponenciális	Egész, valós	Valós
Ln	Természetes alapú logaritmus	Egész, valós	Valós

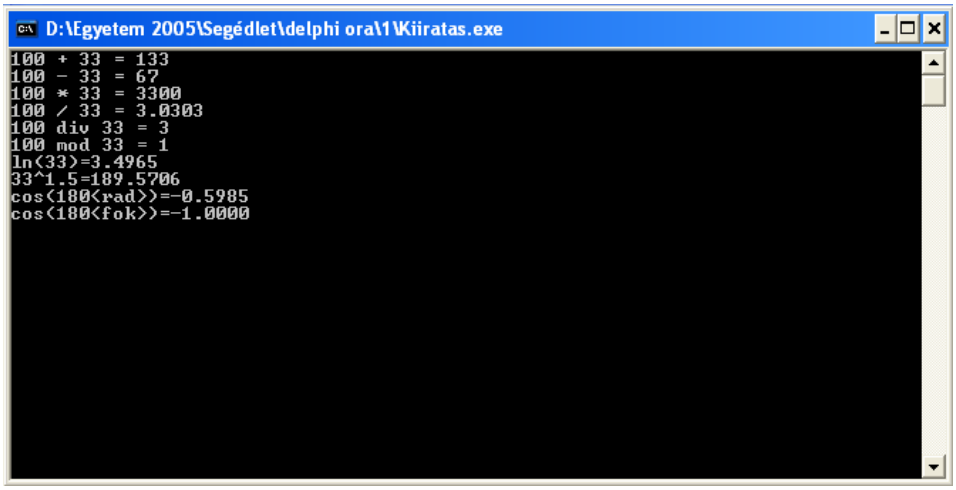
A fenti táblázatban szereplő függvényeken kívül természetesen további matematikai függvények is rendelkezésre állnak, melyeket a **Math** unitban találhatunk meg. E beépített

modult a konzol program deklarációs részében a *uses* szó után kell szerepeltetnünk, mint újabb erőforrást. A System-modul kerekítő (**Round)** és egészrészképző (**Int)** függvénye (legnagyobb egész szám, amely nem nagyobb az adott számnál) mellett itt megtalálhatjuk a felfelé kerekítés (**Ceil**) és a lefelé kerekítés (**Floor**) függvényét. Valós számok egész kitevőjű hatványozására alkalmas az **Intpower**, míg valós kitevő esetén a **Power** függvény használható. Gyakori feladatot jelent a változók típusának konvertálása. Pl. szögek esetén fokban megadott számokat radiánná a **DegToRad** konvertál (fordított esetben a **RadToDeg** függvénnyel dolgozhatunk). Ugyancsak gyakori a valós számok karakter típussá való konvertálása (**FloatToStr**), valamint a karaktersztring valós számmá alakítása (**StrToFloat**). A 7. táblázat kiegészítéseként megemlítünk néhány új matematikai függvényt, melyet szintén a Math unit tartalmaz. Az arkusz-koszinusz (**ArcCos**), arkusz-szinusz (**ArcSin**) és a kotangens (**Cotan**) függvények független változójának értékét radiánban kell megadnunk. Tetszőleges egész alapú logaritmust a **LogN** függvénnyel számíthatunk. A hiperbolikus függvények és inverzeik is felhasználhatók: **Sinh**, **Cosh**, **Tanh** és **ArcSinh**, **ArcCosh**, **ArcTanh**. E mellett egyváltozós polinomok számítása is lehetséges. A **Poly** függvény első paramétere a független változó értéke, míg a második paraméter egy tömb (vektor), mely a polinom együtthatóit tartalmazza. Ha már az adattömböket megemlítettük, elmondjuk, hogy kezelésüket szintén a Math unit függvényei teszik kényelmesebbé. A tömb minimális és maximális elemének megkeresését a **MinValue** és **MaxValue** függvények alkalmazásával tehetjük meg. Tömbökkel számításokat is végezhetünk, pl. a **Sum** függvény a tömb elemeinek az összegével, a **SumOfSquares** az elemek négyzetösszegével tér vissza. A tömbök elemei alapján statisztikai számításokat is végezhetünk. A **Mean** függvény az adatok átlagértékét, a **Variance** a szórásnégyzetet, valamint a **StdDev** a tapasztalati szórás értékét szolgáltatja. Végül a **Rand** függvény [0,1]-ban véletlen valós számot generál. Itt meg kell jegyeznünk, hogy ez az eljárás álvéletlen számgenerátorként működik, azonban ha alkalmazzuk a **Randomize** eljárást, akkor elindíthatjuk a véletlen számok véletlenszerű generálását is. A Math unit ezt kiegészíti a **RandG** függvénnyel, mely adott várható értékű és szórású Gauss-eloszlásból származó véletlen számmal tér vissza.

A fentiek alkalmazásaként készítsük el első konzol programunkat. A program hajtson végre egész és valós számok felhasználásval matematikai alapműveleteket, majd az eredményeket írja ki a konzol képernyőre. A kiíratást a **Write** és a **Writeln** függvényekkel hajthatjuk végre, melyek között az a különbség, hogy a writeln utasításnál a kiírandó szöveg automatikusan egy enter-rel zár. A futtatási eredmény a 14. ábrán látható, a *kiiratas.dpr* program listája pedig a következő:

```
program Kiiratas;

{$APPTYPE CONSOLE

uses

    SysUtils, Math;                          // felhasznált unitok

var

    a, b, c : integer;
    ered1,ered2,ered3,ered5,ered6 : integer;
    ered4,ered7,ered8,ered9,ered10: real;
```

```
begin
    a:=100; b:=33; c:=180;                  // értékadás (:= jelentése egyenlő)
    ered1 := a+b;                           // összeadás
    ered2 := a-b;                           // kivonás
    ered3 := a*b;                           // szorzás
    ered4 := a/b;                           // osztás
    ered5 := a div b;                       // maradékképzés (egész)
    ered6 := a mod b;                       // egész osztás  (egész)
    ered7 := ln(b);                         // logaritmus naturalis
    ered8 := power(b,1.5);                  // hatványozás
    ered9 := cos(c);                        // koszinusz
    ered10 := cos(DegtoRad(c));             // típus konvertálás
    writeln(a,' + ', b ,' = ',ered1);       // kiiratás, sorvége jellel
    writeln(a,' - ', b ,' = ',ered2);
    writeln(a,' * ', b ,' = ',ered3);
    writeln(a,' / ', b ,' = ',ered4:5:4);   // mezőszélesség, tizedesjegy
    writeln(a,' div ', b ,' = ',ered5);
    writeln(a,' mod ', b ,' = ',ered6);

    writeln('ln(',b,')=',ered7:6:4);
    writeln(b,'^1.5=',ered8:6:4);
    writeln('cos(',c,'<rad>)=',ered9:6:4);
    writeln('cos(',c,'<fok>)=',ered10:6:4);
    readln;                                 // enter leütéséig vár a program

end.
```



13. ábra. A kiiratas.dpr program futási eredménye

A programot a FILE menü SAVE AS .. parancsával, tetszőleges néven a kívánt könyvtárba elmenthetjük (**.dpr* kiterjesztéssel). Természetesen az első futtatás után a **.dpr* fájl nevének megfelelő **.exe* fájl-t is automatikusan generáljuk, mely önálló, a fejlesztői rendszeren kívül is működő alkalmazást jelent. A program futtatása előtt érdemes egy gyors tesztet végezni a **Ctrl+F9** billentyű-kombinációval, mely az esetleges hibákat ismeri fel és jelzi ki számunkra. Ha hibátlan a program, akkor az F9 billentyűvel, vagy a RUN menü azonos nevű parancsával futtathatjuk a programot. A program megszakítását a **Ctrl+C**

billentyű-kombinációval érhetjük el. Megjegyezzük, hogy a DELPHI rendszer HELP menüje hatékony eszköz lehet a programozás során. Ha a programlistában egy adott kulcsszó elé helyezzük a kurzort és megnyomjuk az F1 billentyűt, akkor az adott függvény leírása automatikusan megjelenik, mely részletes tájékoztatást nyújt a programfejlesztő számára.

2.1.3 Feltételes és ciklusutasítások

Az **if…else...end** szerkezetű feltételes utasítással az 1.6 fejezetben ismerkedtünk meg. Az ott elmondottak a DELPHI nyelv esetén is érvényesek. A 4. ábrán az *if* utasítás folyamatábráját és programsémáját láthatjuk. Az utóbbi a DELPHI-ben annyival módosul, hogy a *begin… end* részt a *then* szócska előzi meg. A **for** ciklusutasítás is hasonlóan programozható, mint MATLAB rendszerben. Használata az 1.7 fejezetben található. Az 5. ábrán bemutatott séma DELPHI-ben egyetlen *do* szócskával bővül. Mindkét utasítás végén az **end** után pontosvesszőt kell tenni. A fenti két feltételes és ciklusutasítás együttes alkalmazására tekintsük az alábbi példát. Írjunk konzol-alkalmazást, mely három valós szám beolvasása (**Readln**) után a számokat egy egydimenziós tömbbe (**array**[1..elemszám**]**) helyezi, majd a tömb elemeit növekvő sorrendbe rendezi, majd az új tömb tartalmát kiíratja a képernyőre. A megoldás:

```
program ciklusok;

{$APPTYPE CONSOLE}

uses

    SysUtils;

var

    a, b, c, seged : real;              // valós elemű számok beolvasáshoz
    szamok : array[1..3] of real;       // 3 elemű 1 dimenziós tömb
    i,j : integer;                      // for ciklusváltozói

begin

    write('1. szam : '); readln(a);     // kiíratás sorvége karakter nélkül és
    write('2. szam : '); readln(b);     // beolvasás sorvége karakterrel
    write('3. szam : '); readln(c);
    if b < a then
            begin
                    seged := a;         // két változó felcserélése segédváltozóval
                    a := b;             // ha a>b kicseréljük a változók tartalmát
                    b := seged;
            end;
    if c < a then
            begin
                    seged := a;
                    a := c;
                    c := seged;
```

```
            end;
    if c < b then
            begin
                    seged := b;
                    b := c;
                    c := seged;
            end;
    for j:=1 to 3 do                        // a,b,c beolvasása tömbbe
            begin
                    if j = 1 then
                            szamok[j]:=a
                    else if j < 3  then
                            szamok[j]:=b
                    else
                            szamok[j]:=c;
            end;
    writeln('A szamok novekvo sorrendben: ');
    for i:= 1 to 3 do writeln(floattostr(szamok[i]));
    readln;
end.
```

Az 1.7 fejezetben már megismerkedtünk a **while** ciklussal, melyben az utasítások addig ismétlődnek, amíg a kilépési feltétel igaz (true) értékű. A ciklusutasítás a DELPHI-ben is ugyanígy működik:

```
While feltétel do
    Begin
            // Utasítások;
    End;
```

Azonban ennek ellentéte is megtalálható, nevezetesen a **repeat…until** ciklus formájában, mely addig ismétlődik, míg a feltétel hamis (false) értékű. A ciklusból tehát a feltétel teljesülése esetén lépünk ki. Ennek kódja a következő:

```
Repeat
            // Utasítások;
Until feltétel;
```

Természetesen mind a *while*, mind pedig a *repeat* ciklusnál a ciklusváltozó inkrementálásáról külön kell gondoskodni. A repeat utasítás alkalmazására írjunk konzol-alkalmazást, mely egy előre megadott jelszó (pl. *geofizika*) lekérdezésével enged belépni egy alkalmazásba. A feladat egyszerű, ui. mindaddig, ameddig az általunk megadott jelszó hibás, újra kérjük be a jelszót. Ennek DELPHI kódja a következő:

```
program jelszo;

{$APPTYPE CONSOLE}

uses        SysUtils;
const       pw = 'geofizika';                   // string tipusú konstans deklarációja
```

```
var

    pwd: string[9];                                  // string tipusú változó

begin
    repeat
            write('Kerem a kodszot: ');
            readln(pwd);
            if pwd = pw then
                    begin
                            writeln('');
                            writeln('Belephet!');
                    end
            else
                    begin
                            writeln('A kodszo hibas! Probalja ujra!');
                    end;
    until pwd=pw;
    readln;
end.
```

A fenti feltételes utasítások mellett szintén alkalmazhatjuk a **case** utasítást, mely több sokirányú elágazást valósít meg adott feltételek teljesülése mellett (ld. 1.6 fejezet):

```
Case változó of
    1: // utasítás1;
    2: // utasítás2;
    …
    n: // utasításn;
    else     // utasítás(n+1);
End;
```

A feltételes utasítások közös alkalmazására tekintsük a következő példát. Határozzuk meg egy billentyűzetről bevitt egész számról, hogy prímszám-e (1-en és önmagán kívül más egész számmal maradék nélkül nem osztható). Az eredményt írassuk ki a képernyőre, és addig kínáljuk fel a folytatás lehetőségét, míg ki nem szeretnénk lépni a programból. Egy lehetséges megoldás a következő:

```
program prim;

{$APPTYPE CONSOLE}

uses

    SysUtils;

var

    szam ,i, j: integer;
```

```
    valasz: string;
begin
     repeat                                          // ismételjük amíg 'n'-et nem ütünk
          i := 2;                                    // egyel osztható minden egész
          j := 0;                                    // segédváltozó
          writeln('szam: '); readln(szam);           // egész szám bekérése
          while (i < szam ) do
               begin
                    if szam mod i=0 then             // ha a maradék nulla
                         begin
                              j:=1; break;           // kilép a while ciklusból
                         end;
                         i:=i+1;                     // egyel nagyobb egészt vizsgálunk
               end;
          case j of
               1: writeln('Nem primszam.');
               0: writeln('Primszam.');
          end;
          writeln('');
          writeln('Folytatja ?');
          writeln('y = igen     n = nem');
          readln(valasz);
          writeln('');
      until valasz = 'n';
end.
```

2.1.4 Eljárások és függvények alkalmazása

A 2.1.1 fejezetben bevezetett *eljárás*-ok (procedure) és *függvény*-ek (function) megírása a konzol program deklarációs részében történik. A két típus között az alapvető különbség az, hogy a függvény valamilyen értékkel (output) tér vissza, míg az eljárásnak csak input paraméterei vannak. A hívásuk a program törzsében a következő formában történik:

```
Procedure Név(Paraméterlista)
    Begin
          …
    End;
```

Hívás: Név(Aktuális paraméterek);

```
Function Név(Paraméterlista):Típus;
    Begin
          …
    End;
```

Hívás: Változó:=Név(Aktuális paraméterek);

Meg kell jegyeznünk, hogy függvény függvényt is és eljárást is hívhat (és fordítva), mely értelemszerűen a konzol program deklarációs részében tehető meg. A fentieknek megfelelően

a függvényeket általában értékadó utasításokban és matematikai számításokban alkalmazzuk, míg az eljárások gyakran az objektumok metódusait tartalmazzák.

Példaként végezzünk számításokat egy maximálisan 20 elemű valós tömbön. A *TombOlvas* nevű eljárás kérje be a tömb elemszámát, majd ennek megfelelően az elemeket is. Ezután az *Osszegszamitas* nevű függvény számítsa ki a tömb elemeinek az összegét, és az *AtlagSzamitas* nevű eljárás adja meg ugyanezen elemek számtani átlagát. A fenti eljárásokat és függvényeket a program törzsében hívjuk meg, majd az eredményeket írassuk ki a képernyőre. A feladatmegoldás során több változót is ugyanolyan típusú tömbnek kell deklarálnunk. Ebben az esetben célszerű alkalmazni az ún. típus-deklarációt, mely saját típusú (**type**) változó bevezetését teszi lehetővé. Megjegyezzük, hogy a program futtatása során a tizedesvesszőt konzol alkalmazások esetén a „.” karakter jelöli. A megoldás a fentiek figyelembevételével:

```
program Tomb;

{$APPTYPE CONSOLE}

// Deklarációs rész ------------------------------------------------------------------------------

uses        SysUtils;
type        tombtipus = array[1..20] of real;          // típusdeklaráció
var                                                    // globális változók

            db: integer;                               // tömb elemeinek száma
            x: tombtipus;                              // 20 elemű valós tömb
            AdatokOsszege, AdatokAtlaga : real;        // eredménynek

procedure TombOlvas(var w: tombtipus; var m: integer);
    var
            i : integer;                                        // lokális változó
    begin
            write('A tomb adatainak szama: ');
            readln(m);
            for i:=1 to m do
                    begin
                            write(i:2,'. adat: ');
                            readln(w[i]);                       // beolvasás billentyűzetről
                    end;
            end;

function OsszegSzamitas(w: tombtipus; m:integer): real;        // var w,m-nél már szerepelt
    var
            i : integer;
            sum : real;
    begin
            sum := 0;
            for i:=1 to m do sum:=sum+w[i];                     // összeg számítás
            OsszegSzamitas := sum;                              // kimenő paraméter
    end;
```

```
procedure AtlagSzamitas(w : tombtipus; m : integer; var atlag: real);
    begin
            atlag := OsszegSzamitas(w,m);
            atlag := atlag/m;
    end;

// Programtörzs ----------------------------------------------------------------------------------

begin
    TombOlvas(x,db);                                        // hívások akt. változókkal
    AdatokOsszege := OsszegSzamitas(x,db);
    AtlagSzamitas(x,db,AdatokAtlaga);
    writeln;
    writeln('A tomb adatainak osszege: ',Floattostr(AdatokOsszege));
    writeln('A tomb adatainak atlaga : ',Floattostr(AdatokAtlaga));
    readln;
end.
```

2.1.5 Fájlműveletek

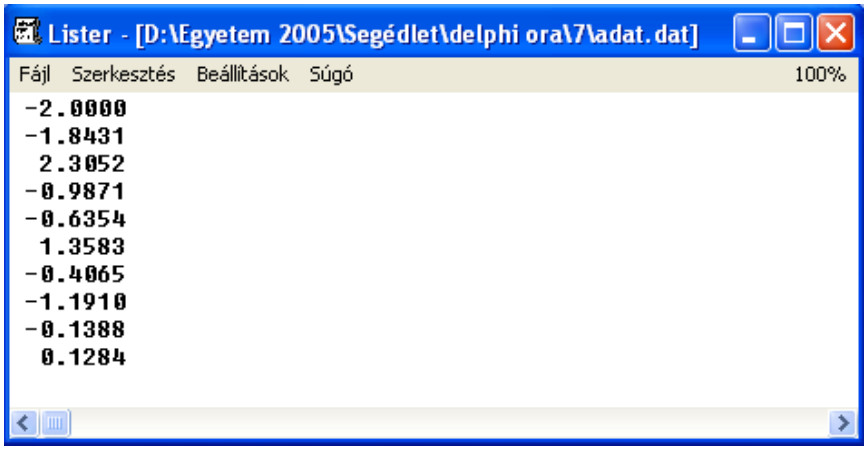
Fájlművelet alatt a számítógép háttértárain tárolt adatok írását vagy olvasását értjük (ld. 1.11 fejezet). Az Delphi nyelv is alkalmas állományokkal végzett műveletek végrehajtására. A geofizikai gyakorlatban a *szöveges (text) típusú állományok* kezelésére van alapvetően szükség, ezért ebben a fejezetben ASCII kódolt adatfájlok feldolgozásával foglalkozunk.

A szöveges fájlok kezelése alapvetően a következő négy lépcsőből áll. Először bizonyos előkészületeket kell tennünk az állomány egyértelmű eléréséhez. Ehhez a fájlt deklarálnunk kell, mely a program deklarációs részében a „var” szócska után ún. fájlváltozó megadásával történik. Szöveges fájl esetén a deklaráció a **TextFile** azonosítót kapja (típus deklarációnál a **file of** típusnév szerkezet használható). A fájlazonosító inicializálása az **AssignFile** paranccsal történik, melynél meg kell adni az azonosítót és a fájl nevét. Ezek után az állomány létrehozása, vagy a meglévő fájl megnyitása következik. Ekkor két eset lehetséges: egyrészt írásra (**Rewrite**), másrészt olvasásra (**Reset**) is megnyithatjuk az állományt. A megnyitás után az adatok feldolgozása történik, mely végül az állomány lezárásával (**CloseFile**) fejeződik be.

A fentiek tanulmányozása céljából végezzük el az alábbi programozási feladatot. Hozzunk létre egy 10 elemű tömböt, melyben valós, [-2,3] intervallumbeli véletlen számok szerepeljenek. Ezután írassuk ki a tömb tartalmát az *adat1.dat* szövegfájlba (ld. 14. ábra), majd zárjuk azt be. Ezután nyissuk meg az adatfájlt és olvassuk be az adatokat, majd számítsuk ki az adatok átlagértékét. A kiíratás után ismét zárjuk be az állományt.

```
program TextF1;

{$APPTYPE CONSOLE}

uses

SysUtils;
```

```
var

    a, s, atlag: real;                  // adatok statisztikájához
    db, i : integer;                    // adatszám
    f,g : TextFile;                     // fájlazonosító
    m : array[1..10] of real;           // adattömb

begin

    randomize;                          // véletlen generátor bekapcsolása
    for i:=1 to 10 do m[i]:=5*random-2;

    AssignFile(g,'adat.dat');           // fájlazonosító inicializálása
    Rewrite(g);                         // fájl létrehozása írásra
    for i:=1 to 10 do writeln(g,m[i]:7:4);  // fájlba kiír (egy sor + enter)
    CloseFile(g);                       // fájlbezárása
    AssignFile(f,'adat.dat');
    Reset(f);                           // fájl megnyitása olvasásra
    s:= 0;  db := 0;
    writeln('A beolvasott adatok: ');
    while not eof(f) do                 // fájl végéig ismétel
        begin
            db := db+1;
            readln(f,a);                // beolvasás fájlból  (egy sor + enter)
            writeln(a:9:4);             // képernyőre kiírja a beolvasott adatot
            s := s+a;                   // beolvasott adatok összege
        end;
    writeln;
    atlag := s/db;                      // beolvasott adatok átlaga
    CloseFile (f);
    writeln('Az adatok szama: ',db);
    writeln('Az adatok osszege: ',s:7:4);
    writeln('A beolvasott adatok atlaga: ',atlag:7:4);
    readln;

end.
```



14. ábra. Az adat.dat szövegfájl aktuális tartalma

2.1.6 A Unitok használata konzol programokban

A *Unit*-ok konzol programokban betöltött szerepével a 2. fejezetben már foglalkoztunk. A moduláris programozás jelentősen megkönnyíti a programfejlesztést. Sok esetben előfordul, hogy egy adott algoritmust többször is hívnunk kell a program során. Pl., ha egy geofizikai inverz problémát akarunk implementálni, akkor tudnunk kell, hogy az elvi geofizikai adatok számítása (direkt feladat) minden iterációs lépésben újra megtörténik. Ekkor célszerű az előremodellezés algoritmusát egy külön modulba (unitba) megírni, majd azt a főprogramban elegendő a megfelelő helyen újra meghívni. Később látni fogjuk, hogy az objektum-orientált programozás is hatékonyan kihasználja a moduláris programozás előnyeit.

Elsőként ismerkedjünk meg a unit-ok felépítésével. A **.pas* kiterjesztésű unit három részből áll: modulfej, interface és az implementation rész. A *modulfej* a globális hatású fordítási direktívát tartalmazza és a unit nevét. Az *interface* az a kapcsolódási felület, mely által a unit más programegységekből is elérhetővé válik. A deklarációkat szintén az interface rész tartalmazza, valamint a unitban később kifejtett eljárások és függvények deklarációját. Az *implementation* más néven definíciós rész tartalmazza az utasításokat, függvényeket és eljárásokat, valamint az elején a *uses* kulcsszó után azokat a modulokat, amelyeket csak a definíciós rész számára kívánunk felhasználni. A fenti szerkezetű unitok és a főprogram együtt egy alkalmazást képez, melyet a DELPHI-ben *project*-nek nevezünk. A unit használat bemutatására legyen az a feladat, hogy írjuk át a 2.1.4 fejezetben megadott programot úgy, hogy a *Project1.dpr* nevű főprogram (konzol-alkalmazás) hívja a *tomb.pas* nevű unitot, mely a tömb elemein végzett műveleteket tartalmazza. A programozás során a konzol programhoz új unitot a menüsoron belül a FILE, NEW, UNIT parancsokkal illeszthetünk. Ekkor a program (projekt) a **.dpr* kiterjesztésű főprogram mellett egy unittal egészül ki. A teljes program mentését a menü SAVE PROJECT AS .. paranccsal kell végezni. A fenti feladat programkódja:

A, Főprogram:

```
program Project1;

{$APPTYPE CONSOLE}

uses

    SysUtils, Tomb in 'Tomb.pas';                    // unitok neve, elérés

var

    db: integer;
    x: tombtipus;                                    // unitban deklarált típus
    AdatokOsszege, AdatokAtlaga : real;

begin

    TombOlvas(x,db);                                 // Tomb.pas beli eljárás hívása
    AdatokOsszege := OsszegSzamitas(x,db);           // Tomb.pas beli function hívása
    AtlagSzamitas(x,db,AdatokAtlaga);
    writeln; writeln('A tomb adatainak osszege: ',AdatokOsszege:6:2);
```

```
    writeln('A tomb adatainak atlaga : ',AdatokAtlaga:6:2);
    readln;

end.
```

B, Unit:

```
unit Tomb;

interface

    type
            tombtipus = array[1..10] of real;
            procedure TombOlvas(var w: tombtipus; var m: integer);
            function OsszegSzamitas(w: tombtipus; m:integer): real;
            procedure AtlagSzamitas(var w : tombtipus; var m : integer; var atlag: real);

implementation

    procedure TombOlvas(var w: tombtipus; var m: integer);

            var
                    i : integer;

            begin
                    write('A tomb adatainak szama: '); readln(m);
                    for i:=1 to m do
                            begin
                                    write(i:2,'. adat: ');
                                    readln(w[i]);
                            end;
            end;

    function OsszegSzamitas(w: tombtipus; m:integer): real;
            var
                     i : integer;
                    sum : real;
            begin
                    sum := 0;
                    for i:=1 to m do
                            sum:=sum+w[i];
                            OsszegSzamitas := sum;
                    end;

    procedure AtlagSzamitas(var w : tombtipus; var m : integer;var atlag: real);
            begin
                    atlag := OsszegSzamitas(w,m);
                    atlag := atlag/m;
            end;
end.
```

2.1.7 Objektumok használata konzol programokban

A 2. fejezetben bemutattuk az objektum-orientált programozás fő tulajdonságait. A WINDOWS-os alkalmazásoknál igen előnyös ez a technika, azonban konzol programok (mint grafikus felülettel nem rendelkező alkalmazások) esetén is használhatunk objektumokat. A Delphi nyelvben a **class** (osztály) felhasználói típusú dinamikus változókat nevezzük objektumoknak. Az adott változó egy *objektum példány*-t képvisel, mely a többivel együtt a **TObject** nevű ún. *ősosztály*-ból származik. Az objektumot típusként deklaráljuk, ahol elsőként az ún. adatmezőt, majd a metódusokat (tagfüggvények) kell megadnunk. Az objektum metódusainak (eljárások, függvények) kidolgozásánál a procedure vagy a function szócska után a metódus neve elé ponttal kell bevezetni az objektumtípus nevét. Az objektum példányaira mutatókkal hivatkozhatunk, melyeket szokásos módon a var változódeklarációs részben hozhatjuk létre. A programtörzsben először az objektum egy példányát a **Create** metódussal hozzuk létre, a program végeztével pedig a **Free** metódus segítségével szabadítjuk azt fel (dinamikus változó).

Az előző fejezetben tárgyaltuk a unitok alkalmazási lehetőségeit. E programszerkezetet szeretnénk most kiterjeszteni az objektumok bevonásával. Példaként hozzuk létre a *unit1.pas* modulban a *Teglalap* nevű osztályt, melynek tagfüggvényei inicializáljanak egy téglalap oldalait, majd számítsák ki a téglalap kerületét. A unitot az *objektum.dpr* főprogram hívja meg és egy objektumpéldányt hozzunk létre a számítás céljából. Az eredmények kiíratása után szabadítsuk fel az objektum e másolatát. A program a következő:

A, Főprogram:

```
program objektum;

    {$APPTYPE CONSOLE}

uses

    SysUtils, Unit1 in 'Unit1.pas';

var

    t : Teglalap;                       // unit1-ben definiált objektum típusú változó

begin

    t := Teglalap.Create;               // létrehozza az objektumpéldányt
    t.Init(4,5);                        // inicializáló metódus hívása
    t.Szamol;                           // kerületszámítást végző metódus hívása
    writeln('Kerulet: ',t.Megad:6:2);   // kiíratást végző metódus hívása
    t.Free;                             // törli (felszabadítja) az objektumpéldányt
    readln;

end.
```

B, Unit:

```
unit Unit1;

interface

type

    Teglalap = class                        // objektum neve
    // Adatmező:
    Private                                 // külvilág részére nem érhető el

            a,b : real;                     // téglalap oldalai
            eredmeny : real;                // téglalap kerülete

    Public                                  // külvilág számára elérhető
    // Metódusok:
            procedure Init(a0:real;b0:real);  // eljárás – kezdeti értékek megadása
            procedure Szamol;               // eljárás – kerület számítás
            function Megad:real;            // függvény – eredmény megadása

end;
implementation                              // tagfüggvények definiálása

    procedure Teglalap.Init(a0:real;b0:real);
            begin
                    a := a0;
                    b := b0;
                    eredmeny := 0;
            end;

    procedure Teglalap.Szamol;
            begin
                    eredmeny := 2*(a+b);
            end;

    function Teglalap.Megad:real;
            begin
                    Megad := eredmeny;
            end;
end.
```

2.2 WINDOWS ALKALMAZÁSOK FEJLESZTÉSE A DELPHI RENDSZERBEN

A DELPHI fejlesztői rendszer erőssége a WINDOWS operációs rendszer alatt futó *grafikus felhasználói felülettel rendelkező alkalmazás*-ok létrehozása. A WINDOWS-os alkalmazások készítéséhez a **TApplication** osztály metódusai kínálnak lehetőséget, melyek az alkalmazások létrehozását, futtatását és leállítását teszik lehetővé, valamint megteremtik az alkalmazás és a WINDOWS rendszer közötti kapcsolatot. A metódusok lehetővé teszik az

objektumok eseményeinek a kezelését, azaz megszabják, hogy a program hogyan reagáljon külső és belső hatásokra (üzenetekre). Új WINDOWS-os alkalmazás létrehozása a menüsor FILE, NEW, APPLICATION parancsaival hozható létre.

A WINDOWS-os alkalmazás könyvtára szerkezetileg a metódusokat tartalmazó unitokból *(*.pas)*, az objektumok (ablakok) tulajdonságait tartalmazó **.dfm* és **.res* kiterjesztésű (erőforrások csatlakozására vonatkozó információkat tartalmazó) fájlokból, valamint egy főprogramból áll (ld. 2. fejezet). A főprogramot gyakran *projektállomány*-nak nevezzük. Ennek szerkezete a következő:

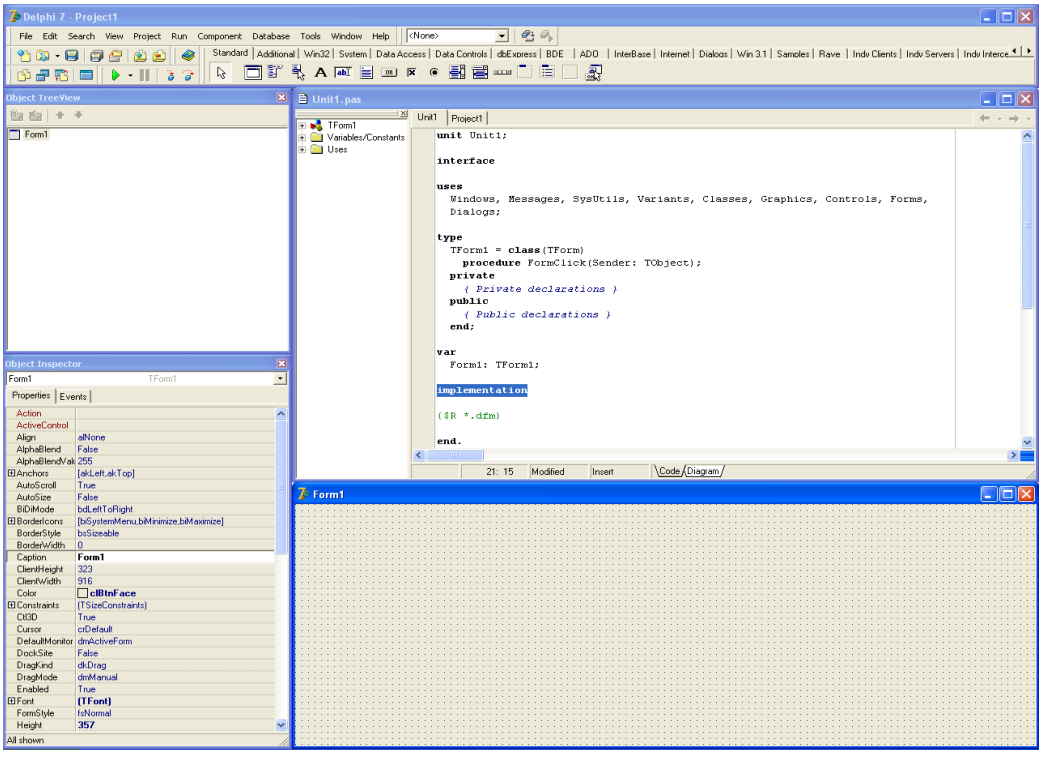
```
program Project1;
uses
    Forms,
    Unit1 in 'Unit1.pas' {Form1};
{$R *.res}
begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    Application.Run;
end.
```

A projektállomány kódja a VIEW menü UNITS .. parancsával, a projekt neve alapján hívható a be a fejlesztői környezetbe. E rövid program nagyon lényeges, ugyanis ez aktivizálja a WINDOWS-os alkalmazás létrehozásához szükséges definíciókat. Mint tudjuk, az alkalmazáshoz általában tartozik egy főablak (Form1), melyhez további ablakok kapcsolódhatnak. Az ablakok tulajdonságait a **TForm** osztály írja le, melynek deklarációját és metódusait a **Forms** modul tartalmazza. Ezt, mint erőforrást szerepeltetjük a „uses" után a deklarációs részben, valamint kezdetben a főablak (Form1) metódusait tartalmazó unitot (Unit1). A DELPHI alkalmazás globális változója az **Application**, melynek őse a már említett TApplication osztály. Tehát az Application változó egy objektumpéldány, mely az alkalmazás tulajdonságait és metódusait hordozza magában. A projektállományban a programtörzsben metódusokat aktivizálunk, mely az Application objektum alapbeállításait (**Initialize**), a főablak létrehozását (**CreateForm**), és az üzenetek kezelését (**Run**) teszi lehetővé. Látható, hogy a CreateForm metódus egy eseménykezelő eljárás (procedure), mely a létrehozandó főablak osztályát (TForm1) és annak referenciaváltozóját (Form1) hívja meg. A TForm1-et a unit1.pas modulban típusdeklarációval hozzuk létre, mint a TForm osztály egy objektumát. Ebből látható, hogy a DELPHI minden egyes ablakhoz külön unitot rendel. Az ablakkezelő metódusokat tartalmazó modulok a **.pas* kiterjesztésű unit állományok (az ablakok tulajdonságait a **.dfm* állomány tartalmazza).

A DELPHI alkalmazások fejlesztése a programírás során automatizált eszközökkel is támogatott. A 15. ábrán a fejlesztői környezet látható, melynek bal alsó részén található az OBJECT INSPECTOR nevű ablak. Ebben kiválaszthatjuk a program keresett objektumát (pl. ablak) és annak tulajdonságait (PROPERTIES) és metódusait (EVENTS) tetszés szerint szerkeszthetjük. Az események kiválasztása során a megfelelő unit programlistájában megjelenik a metódust definiáló eljárás, melynek kódját ezután tovább írhatjuk. A program írásánál egy további hasznos eszköz is adott. Ha beírjuk egy adott objektum azonosítóját a programba listájába és utána egy pontot teszünk, a DELPHI automatikusan felkínálja az összes lehetséges metódust, amely az objektumra alkalmazható. A listából gyorsan

kiválaszthatjuk a nekünk megfelelő eljárást (*Procedure*), tulajdonságot (*Property*) vagy függvényt (*Function*).

Első példaprogramunk a főablak kattintási eseményeit mutatja be. A Form1 fejlécén legyen az „Alapállapot” felirat, majd ha az objektum felületén egyszer kattintunk, írassuk ki ugyanoda azt, hogy „Ön egyszer kattintott”. Ugyanezt hajtsuk végre dupla kattintás esetén is, valamint egy „Kilépés” gomb szakítsa meg a program futását. Először nyissunk meg egy új alkalmazást. A fejléc nevének beírása az OBJECT INSPECTOR-on belül a CAPTION tulajdonságnál tehető meg. Ezután a kattintási események beállítása következik. Az OBJECT INSPECTOR EVENTS mezőjében keressük meg az ONCLICK eseményt. Az e melletti üres mezőre duplán kattintva a Unit1.pas programlistában automatikusan beépül a **FormClick** nevű eseménykezelő eljárás. Ez azt jelenti, hogy a program futása alatt, ha bármikor az ablak felületére kattintunk, abban a pillanatban meghívjuk a FormClick eljárást. Az, hogy mi történik ilyenkor, azt az eljárásban megírt utasítások fogják meghatározni. Itt pl. az a feladat, hogy írassuk ki az „Ön egyszer kattintott” c. szöveget. A szöveg kiíratásának a helye a fejléc, amely a CAPTION-nek a FormClick eljáráson belüli módosításával érhető el. A kettős kattintás hatására meghívott esemény ugyanígy elvégezhető az OBJECT INSPECTOR-on belül az ONDBLCLICK esemény kiválasztásával, mely a **FormDblClick** eljárást építi be a programba. A programból való kilépéshez építsünk be egy **Button** (nyomógomb) vezérlőt. Ezt az eszköztár STANDARD vezérlőcsoport (komponens) elemei között találjuk meg. Húzzuk a gombot az egérrel a Form1 felületére, ez automatikusan a Button1 nevet kapja (és az OBJECT INSPECTOR-ban a Form1-től elkülönülve megjelenik). A billentyű tulajdonság (PROPERTIES) mezőjében a vezérlő nevét állítsuk be (CAPTION - „Kilépés”), majd az események (EVENTS) közül válasszuk ki az ONCLICK eseményt, amely a **Button1Click** eljárást hívja meg. A teljes alkalmazás bezárására vonatkozó parancsot a **Terminate** nevű eljárással adhatjuk ki, mely az Application objektum-változóra alkalmazható. A *Project1.dpr* projektállomány és a hozzá tartozó *Unit1.pas* modul listája a következő:



15. ábra. A DELPHI 7 fejlesztői környezete és a Unit állomány szerkezete

A, Projektállomány:

```
program Project1;

uses

    Forms,
    Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin

    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    Application.Run;

end.
```

B, Unit:

```
unit Unit1;

interface

uses

    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls;                                  // felhasznált erőforrások

type

    TForm1 = class(TForm)
    Button1: TButton;                                   // TButton objektumtípus

    procedure FormCreate(Sender: TObject);              // eseménykezelő eljárások
    procedure FormClick(Sender: TObject);
    procedure FormDblClick(Sender: TObject);
    procedure Button1Click(Sender: TObject);

    private
            { Private declarations }
    public
            { Public declarations }
    end;

var

    Form1: TForm1;
implementation
```

```
{$R *.dfm}

    procedure TForm1.FormCreate(Sender: TObject);
            begin
                    Caption:='Szabó Norbert Péter';
            end;

    procedure TForm1.FormClick(Sender: TObject);
            begin
                    Caption:='Ön egyszer kattintott !';
            end;

    procedure TForm1.FormDblClick(Sender: TObject);
                    begin
                    Caption:='Ön duplán kattintott !';
            end;

    procedure TForm1.Button1Click(Sender: TObject);
            begin
                    Application.Terminate;
            end;
end.
```

2.2.1 Számítási feladatok objektum-orientált programokban

Az előző fejezetben a TButton osztályból származtatott Button vezérlővel már megismerkedtünk, mely egy gomb megnyomásával aktivál valamilyen eseménykezelő eljárást. Számítási feladatok megoldása esetén szükség lehet további vezérlőkre is az ablak felületén. Ilyen pl. a Tlabel típusú **Label** (*címke*), mely szöveg megjelenítésére, valamint az **Edit** (*szövegmező*), mely egysoros szövegszerkesztési feladatokra, valamint adatok bevitelére és információ kijelzésére (pl. számítási eredmények megtekintésére) szolgál.

A fentiek alkalmazására nézzük az alábbi geofizikai példát. A 16. ábra szerinti elrendezésben írjunk alkalmazást, mely a megadott porozitás (Φ), tortuozitási együttható (a) és cementációs kitevő (m) értékek alapján kiszámítja az ellenállás formációfaktor (F) értékét. A képlet az Archie-formula alapján:

$$F = \frac{a}{\Phi^m}.$$

A fejléc neve legyen „Formáció tényező számítása”, valamint a számítás elvégzését a „Számítás” nevű gomb aktivizálja. Az alkalmazást újabb számítások elvégzése céljából a „Kilépés” gomb megnyomásáig használhatjuk. A programírást kezdjük az alkalmazás felületének kialakításával, ennek megfelelően 7db címkét (Label1, …,Label7), 4db szövegmezőt (Edit1,…, Edit4), valamint 2db nyomógombot (Button1, Button2) helyezzünk a tervezőablakra (ld. 16. ábra). A címkék szövege a CAPTION-ben adhatók meg. A szövegmező **Text** tulajdonsága teszi lehetővé az adatok bevitelét és az eredmények megjelenítését. A programban az Editx.Text szerkezet alkalmas a mező pillanatnyi tartalmának lekérdezésére és az értékadásra. Ha azt szeretnénk elérni, hogy a program

indításakor az Edit mezők üresek legyenek, töröljük ki default értéküket az OBJECT INSPECTOR PROPERTIES/TEXT részében. A számítási feladatot ezután a „Számítás” nyomógomb OnClick eseményre hajtja végre, amennyiben a számítási algoritmust a ButtonClick nevű eljárás tartalmazza. Itt a hatványozás miatt ne felejtsük el a Math unit-ot is deklarálni. Megjegyezzük, hogy futtatásnál a konzol-alkalmazástól eltérően a tizedesvesszőt itt nem ponttal, hanem vesszővel kell megadnunk. Ezen kívül az objektumok tulajdonságait (méret, pozíció, betűtípus, színek, feliratok stb.) a **.dfm* fájlban találjuk.

16. ábra. A DELPHI program futása közben

A program listája:

A, Projektállomány:

```
program Project1;

uses

    Forms,
    Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    Application.Run;
end.
```

B, Unit:

```
unit Unit1;

interface
uses
```

```
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, Math;

type

    TForm1 = class(TForm)
            Label1: TLabel;
            Button1: TButton;
            Label2: TLabel;
            Label3: TLabel;
            Button2: TButton;
            Edit1: TEdit;
            Edit2: TEdit;
            Edit3: TEdit;
            Edit4: TEdit;
            Label4: TLabel;
            Label5: TLabel;
            Label6: TLabel;
            Label7: TLabel;

    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);

    private
            { Private declarations }
    public
            { Public declarations }
  end;

var

    Form1: TForm1;

implementation

{$R *.dfm}

    procedure TForm1.Button1Click(Sender: TObject);
            begin
                    Application.Terminate;
            end;

    procedure TForm1.Button2Click(Sender: TObject);

            var por,m,a,seg,f : real;

            begin
                    if (Edit1.Text <> '') and (Edit2.Text <> '') then
                            begin
                                    por:=StrToFloat(Edit1.Text);
```

```
                                m:=StrToFloat(Edit2.Text);
                                a:=StrToFloat(Edit3.Text);
                                seg:=power(por,-m);
                                f:=a/seg;
                                Edit4.Text:=FloatToStr(f);
                        end;
        end;
end.
```

C, Dfm file:

```
object Form1: TForm1

    Left = 212
    Top = 120
    BorderStyle = bsDialog
    Caption = ' Form'#225'ci'#243' t'#233'nyez'#337' sz'#225'm'#237't'#225'sa'
    ClientHeight = 330
    ClientWidth = 531
    Color = clBtnFace
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -11
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    OldCreateOrder = False
    PixelsPerInch = 96
    TextHeight = 13

    object Label1: TLabel
        Left = 40
        Top = 40
        Width = 43
        Height = 13
        Caption = 'Porozit'#225's'
    end
    object Label2: TLabel
        Left = 40
        Top = 80
        Width = 101
        Height = 13
        Caption = 'Cement'#225'ci'#243's t'#233'nyez'#337
    end
    object Label3: TLabel
        Left = 40
        Top = 120
        Width = 94
        Height = 13
        Caption = 'Tortuozit'#225'si t'#233'nyez'#337
    end
```

```
object Label4: TLabel
        Left = 144
        Top = 192
        Width = 83
        Height = 13
        Caption = 'Form'#225'ci'#243' t'#233'nyez'#337
end
object Label5: TLabel
        Left = 336
        Top = 40
        Width = 89
        Height = 13
        Caption = 'Tartom'#225'nya: 0 -1.0'
        Color = clCaptionText
        ParentColor = False
end
object Label6: TLabel
        Left = 336
        Top = 80
        Width = 101
        Height = 13
        Caption = 'Tartom'#225'nya: 1.0 - 3.0'
        Color = clCaptionText
        ParentColor = False
end
object Label7: TLabel
        Left = 336
        Top = 120
        Width = 101
        Height = 13
        Caption = 'Tartom'#225'nya: 1.0 - 2.0'
        Color = clCaptionText
        ParentColor = False
end
object Button1: TButton
        Left = 312
        Top = 256
        Width = 75
        Height = 25
        Caption = 'Kil'#233'p'#233's'
        TabOrder = 0
        OnClick = Button1Click
end
object Button2: TButton
        Left = 152
        Top = 256
        Width = 75
        Height = 25
        Caption = 'Sz'#225'm'#237't'#225's'
        TabOrder = 1
```

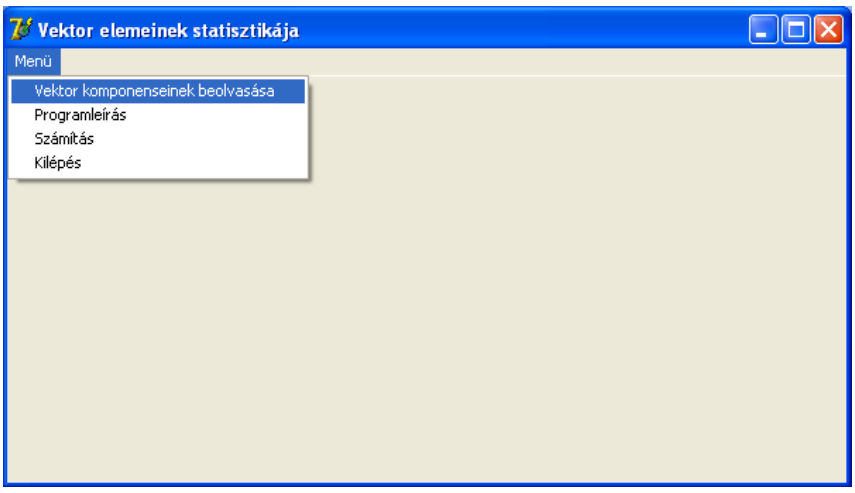
```
            OnClick = Button2Click
        end
        object Edit1: TEdit
            Left = 176
            Top = 40
            Width = 121
            Height = 21
            TabOrder = 2
        end
        object Edit2: TEdit
            Left = 176
            Top = 80
            Width = 121
            Height = 21
            TabOrder = 3
        end
        object Edit3: TEdit
            Left = 176
            Top = 120
            Width = 121
            Height = 21
            TabOrder = 4
        end
        object Edit4: TEdit
            Left = 288
            Top = 192
            Width = 121
            Height = 21
            TabOrder = 5
        end
    end
```

2.2.2 Menüvezérelt DELPHI alkalmazások

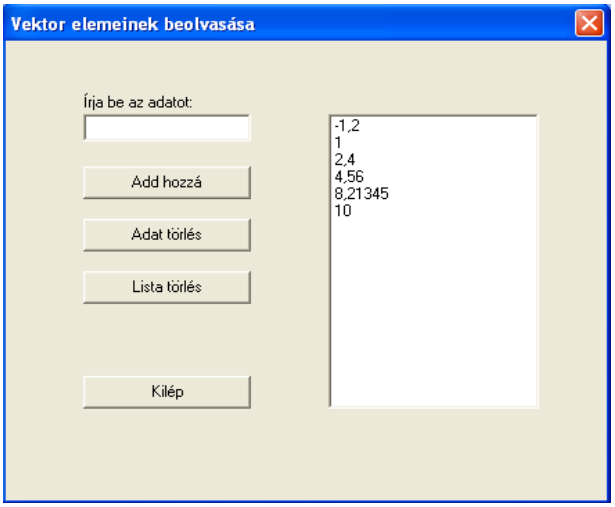
Az eddigiekben bemutatott alkalmazások egyetlen ablakot használtak csupán a programozási feladat megoldására. Azonban az összetettebb WINDOWS alatt futó programok gyakran sokablakosak, valamint többféle funkciót ellátó *menüvezérelt felhasználói felület*-tel rendelkeznek. A menü egyes elemei valamilyen részfeladatot látnak el, melyek kiválasztási sorrendeje a felhasználótól függ. A *menüsor* a főablak címsora alatt helyezkedik el, mely fő*menü*-ből, ill. abból nyíló további tetszőleges számú *almenü*-ből áll. Az események az adott menüpontra való kattintással aktivizálhatók. A menü létrehozása a STANDARD vezérlőcsoportban a MAINMENU kiválasztásával és a tervezőablakra való húzásával lehetséges.

A menü alkalmazását egy összetett WINDOWS alkalmazáson keresztül mutatjuk be. Készítsünk statisztikát egy tetszőleges valós elemekből felépülő tömb adatairől (elemek összege, átlagérték, minimális elem, maximális elem, Euklideszi-norma). A 17. ábrának megfelelően a főmenü valamely almenüjének kiválasztásával újabb ablak (Form2, Form3, Form4) nyíljon meg az adott részfeladat végrehajtása céljából. A vektor elemeit

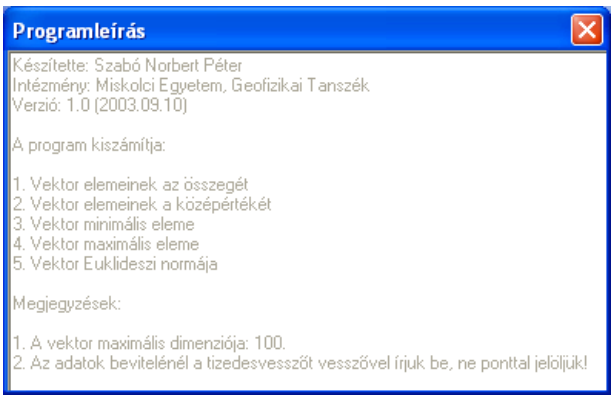
billentyűzetről olvassuk be, e mellett legyen lehetőség az adatlista szerkesztésére (ld. 18. ábra). Kis programleírást is adjunk külön ablakban az alkalmazásról (ld. 19. ábra). Az eredményeket külön ablakban Edit szövegmezőkben jelenítsük meg.



17. ábra. A DELPHI alkalmazás menürendszere (Form1)

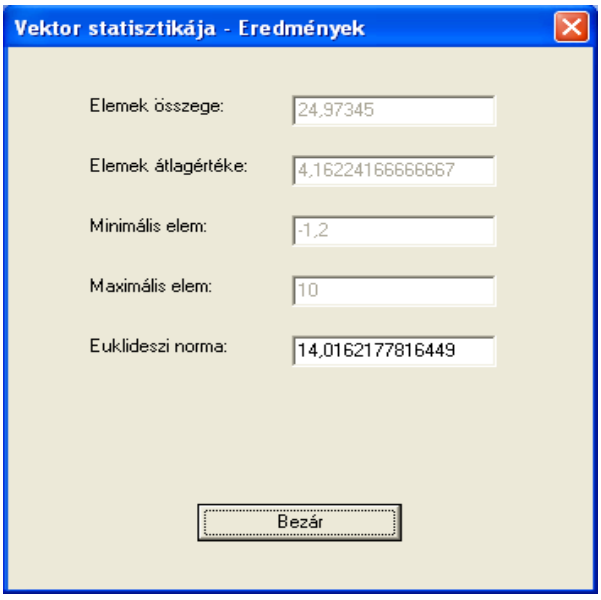


18. ábra. Az adatbeviteli és szerkesztési felület (Form2)



19. ábra. A programleírás ablaka (Form3)

A programozási feladat megoldásának első lépése a Form1 menürendszerének a kialakítása. Az egyes almenük címét az OBJECT INSPECTOR-on belül a CAPTION tulajdonságnál adhatjuk meg. Minden almenühöz új ablak létrehozása szükséges. Először külön-külön létrehozzuk az ablakokat a fejlesztői környezet menüjéből a FILE, NEW, FORM utasításokkal, majd az alkalmazás adott almenüjéhez OnClick eseményként hozzárendeljük a megfelelő ablakot. Ezt a főablakhoz tartozó *Unit1.pas* modul eseménykezelő eljárásaiban a **Showmodal** függvény segítségével tehetjük meg. Megjegyzésként megemlítjük, hogy a vezérlők mérete az ablakok méretének növelésével nem változik. Esztétikai okból, ezért ha az ablak fejlécén csak a bezárás gombot kívánjuk szerepeltetni (méretváltoztatást nem engedjük meg) az OBJECT INSPECTOR-ban a PROPERTIES/BORDERSTYLE tulajdonságot BSDIALOG típusra kell változtatnunk. A *Unit2.pas*-ban (Form2) egy új elemet láthatunk, ez a **ListBox** (lista) vezérlő. Ebben stringeket és adatokat kezelhetünk. A **TListbox** tulajdonsága, hogy az **Items** objektumot használja fel a lista elemein végzett műveletek kezelésére (hozzáadás, törlés, beszúrás stb.). Fontos tudnunk, hogy az Items **Count** tulajdonsága tartalmazza a lista elemeinek a számát, melyben az első elem a 0. sorszámot kapja (tehát ugyanúgy, mint a C++ nyelvben az i-edik adat mindig az [i-1]. indexxel hívható a listából). További új elem a Form3 ablakban (*Unit3.pas*) megjelenő **Memo** (többsoros szövegszerkesztő) vezérlő, melybe kívánt hosszúságú szöveget írhatunk. A program leírását tartalmazó szöveget a PROGRAM LIST EDITOR segítségével vihetjük be, melyet az OBJECT INSPECTOR/PROPERTIES/LINES tulajdonsága melletti TSTRINGS feliratra való kattintással hívhatunk be. Ha a szöveg elkészült, szintén az objektum tulajdonságainál a kurzort eltüntethetjük a CONSTRAINTS tulajdonság False értékre állításával. A **Close** mindig az adott ablak bezárását eredményezi, míg a Terminate eljárás az egész alkalmazást megszakítja. A Form2 bezárásakor a lista adatait átadjuk a *Modul.pas* nevű unit-ban szereplő *x* nevű tömb változónak (természetesen a unit implementációs részében azokat az erőforrásokat meg kell nevezni, amire a modul támaszkodik). Ez a modul végzi a tömb elemeinek felhasználásával a matematikai számításokat. A *Vektosszeg* nevű függvény az elemek összegét, a *Vektatlag* eljárás azok számtani átlagát, a *MinMax* eljárás a maximális és minimális elemet, és a *Norma* nevű függvény pedig az Euklideszi-norma (elemek négyzetösszegéből vont végyzetgyök) értékét számítja ki (ld. 20. ábra). A projektet felépítő fájlok (a **.dfm* fájl nélkül) a következők:



20. ábra. Az eredmények megjelenítése (Form4)

A, Projektállomány:

```
program stat;

uses
    Forms,
    Unit1 in 'Unit1.pas' {Form1},
    Unit2 in 'Unit2.pas' {Form2},
    Unit3 in 'Unit3.pas' {Form3},
    Unit4 in 'Unit4.pas' {Form4};
{$R *.res}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.CreateForm(TForm3, Form3);
  Application.CreateForm(TForm4, Form4);
  Application.Run;
end.
```

B, Unit (Form1):

```
unit Unit1;

interface

uses

    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, Menus;

type

    TForm1 = class(TForm)
            MainMenu1: TMainMenu;
            MEn1: TMenuItem;                                    // menüelem típusa
            Vektorkomponenseinekbeolvassa1: TMenuItem;
            Programlers1: TMenuItem;
            Szmts1: TMenuItem;
            Kilps1: TMenuItem;

    procedure Kilps1Click(Sender: TObject);
    procedure Vektorkomponenseinekbeolvassa1Click(Sender: TObject);
    procedure Programlers1Click(Sender: TObject);
    procedure Szmts1Click(Sender: TObject);

    private
    public

    end;
```

```
var

    Form1: TForm1;

implementation

uses Unit2, Unit3, Modul, Unit4;                    // felhasznált erőforrások

{$R *.dfm}

    procedure TForm1.Kilps1Click(Sender: TObject);
            begin
                    Application.Terminate;
            end;

    procedure TForm1.Vektorkomponenseinekbeolvassa1Click(Sender: TObject);
            begin
                    Form2.Showmodal;
            end;
    procedure TForm1.Programlers1Click(Sender: TObject);
            begin
                    Form3.Showmodal;
            end;

    procedure TForm1.Szmts1Click(Sender: TObject);
            begin
                    osszeg:= Vektosszeg(x,db);              // eljárások hívása (Modul)
                    Form4.Edit1.Text := FloatToStr(osszeg);
                    Vektatlag (x,db,atlag);
                    Form4.Edit2.Text := FloatToStr(atlag);
                    MinMax(x,db,max,min);
                    Form4.Edit3.Text := FloatToStr(min);
                    Form4.Edit4.Text := FloatToStr(max);
                    normert:=Norma(x,db);
                    Form4.Edit5.Text := FloatToStr(normert);
                    Form4.Showmodal;
            end;
end.
```

C, Unit (Form2):

```
unit Unit2;

interface

uses

    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls;
```

```
type
    TForm2 = class(TForm)
            Edit1: TEdit;
            Label1: TLabel;
            Button1: TButton;
            Button2: TButton;
            Button3: TButton;
            Button4: TButton;
            Adatok: TListBox;

    procedure Button4Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);

    private
    public
end;


var

    Form2: TForm2;

implementation

{$R *.dfm}

uses Modul, Unit1;

    procedure TForm2.Button4Click(Sender: TObject);
            var
                    i:integer;
            begin
                    for i:=0 to Adatok.Items.Count-1 do
                            begin
                                    x[i+1]:=StrToFloat(Adatok.Items[i]);
                            end;
                    db:=Adatok.Items.Count;
                    Close;
            end;

    procedure TForm2.Button3Click(Sender: TObject);
            begin
                    Adatok.Clear;
            end;

    procedure TForm2.Button1Click(Sender: TObject);
            begin
                    Adatok.Items.Add(Edit1.Text);
```

```
                Edit1.Clear;
                Edit1.SetFocus;                         // kurzort Edit1-re állítja
        end;

    procedure TForm2.Button2Click(Sender: TObject);
        begin
                Adatok.SetFocus;
                if Adatok.ItemIndex > -1 then           // vannak adatok a listában
                        begin
                                Adatok.Items.Delete(Adatok.ItemIndex);
                                showmessage('Törölve');     // üzenetablak
                                Edit1.Setfocus;
                        end;
        end;
end.
```

D, Unit (Form3):

```
unit Unit3;

interface

uses

    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls;

type

    TForm3 = class(TForm)
        Memo1: TMemo;
        private
        { Private declarations }
        public
        { Public declarations }
end;

var

    Form3: TForm3;

implementation

{$R *.dfm}

end.
```

E, Unit (Form4):

```
unit Unit4;
```

```
interface

uses

    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls;

type

    TForm4 = class(TForm)
            Label1: TLabel;
            Edit1: TEdit;
            Label2: TLabel;
            Edit2: TEdit;
            Label3: TLabel;
            Edit3: TEdit;
            Label4: TLabel;
            Edit4: TEdit;
            Button1: TButton;
            Label5: TLabel;
            Edit5: TEdit;

    procedure Button1Click(Sender: TObject);

    private
            { Private declarations }
    public
            { Public declarations }
end;

var

    Form4: TForm4;

implementation

{$R *.dfm}

    procedure TForm4.Button1Click(Sender: TObject);
            begin
                    Close;
            end;
end.
```

F, Unit (Számítások):

```
unit Modul;

interface
```

```
const n = 100;

type        vektor = array[1..100] of real;

var

     x: vektor;
    db: integer;
    max,min,osszeg,atlag,normert: real;
    procedure Vektatlag (a:vektor; m:integer; var atl: real);
    function VektOsszeg(a:vektor; m: integer): real;
    procedure MinMax(a:vektor; m:integer; var  maxertek, minertek: real);
    function Norma(a:vektor;m:integer):real;

implementation

    function Vektosszeg (a:vektor; m:integer):real;
            var
                    i:integer;
            begin
                    result := 0;
                    for i:=1 to m do
                    result:= result+a[i];                          // visszatérési érték
            end;

    procedure Vektatlag (a:vektor; m:integer; var atl: real);
            begin
                    atl := Vektosszeg(a,m)/m;                      // másik függvényt hív
            end;

    procedure MinMax(a:vektor; m:integer; var  maxertek, minertek: real);
            var
                    i:integer;
            begin
                    maxertek := a[1];
                    minertek := a[1];
                    for i:=2 to m do
                            begin
                                    if a[i] > maxertek then maxertek := a[i];
                                    if a[i] < minertek then minertek := a[i];
                    end;
            end;

    function Norma(a:vektor;m:integer):real;
            var
                    svalt:real;                                    // lokális segédváltozó
                    j:integer;                                     // lokális ciklusváltozó
            begin
                    svalt:=0;
```

```
                for j:=1 to m do
                        begin
                                svalt := svalt + (sqr(a[j]));
                        end;
                result:=sqrt(svalt);
        end;
begin
    {nincs más utasítás!}
end.
```

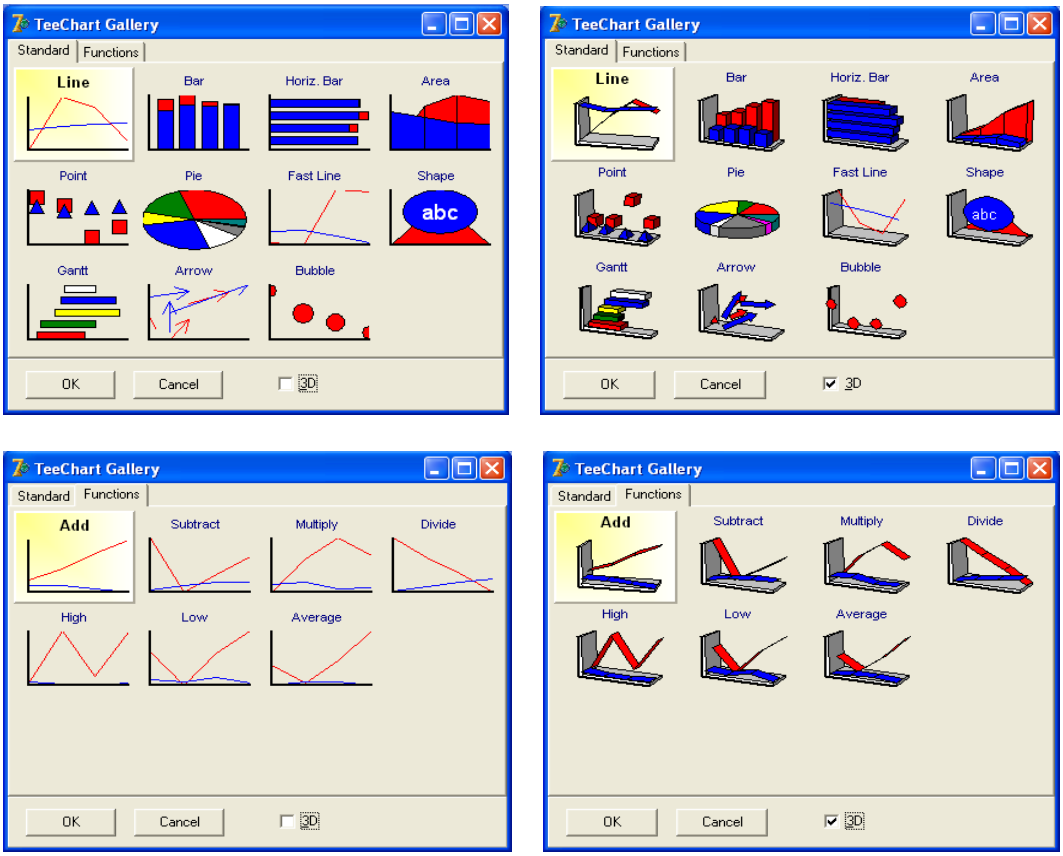
2.2.3 Grafikus felhasználói felülettel rendelkező DELPHI alkalmazások

A DELPHI rendszer a *grafikus megjelenítés*-t komponenseken keresztül támogatja. Ezek a geometriai alakzatok, szövegek, bitképek, képsorozatok stb. képként való megjelenítését teszik lehetővé (e mellett hagyományos rajzvásznat is használhatunk kézi rajzolásra). Egyik legfontosabb lehetőség a speciális grafikon megjelenítő komponens, a **Chart** használata. Ezt az ADDITIONAL palettalapon találhatjuk meg a kompolnensek között. E komponens sokféle adatmegjelenítési lehetőséget kínál. A grafikon tervezősablonra másolását követően a felbukkanó EDITINGCHART menü segítségével beállíthatjuk a diagram típusát (ld. 21. ábra) a CHART, SERIES, ADD parancsokkal, mely azonnal megjelenik a Form ablakban. Ahány diagrammot szeretnénk elhelyezni a Form-ra, annyi Chart vezérlőre van szükségünk. A Chart egyik tulajdonsága a **SeriesList**, mely az adatsorok (**Series**) listáját tartalmazza. A lista egy eleme egy adatsort tartalmaz, melyet az ADD gombal adhatunk a listához. Az adatsor futásidőben különböző metódusokkal kezelhető. Pl. függvényábrázolás esetén az **AddXY** függvény új adatpárt illeszt az adatsorba (törlés, egyéb adathozzáadási mód is lehetséges). A diagram-szerkesztés menüjében beállítható továbbá a zoom, képernyő-görgetési tulajdonságok (GENERAL), tengelyek tulajdonságai (AXIS), fejléc szerkeszthető (TITLES), jelmagyarázat szúrható be (LEGEND), a háttérszín kombináció változtatható (PENEL), a 3-D ábrázolási mód szerkeszthető (3D) stb.

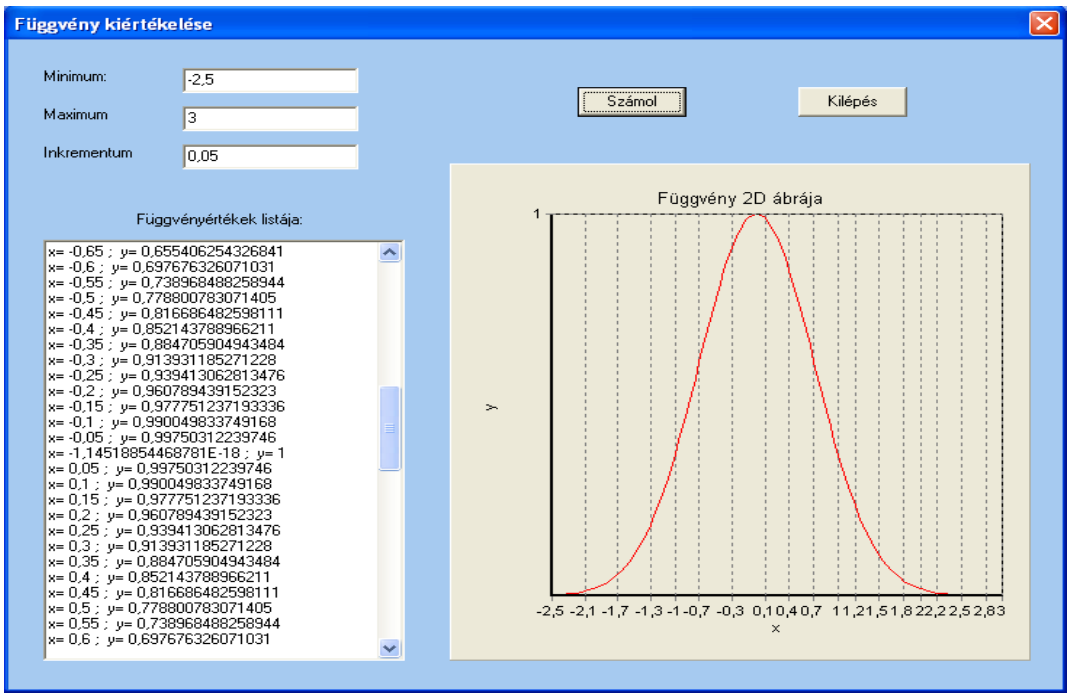
A geoinformatikai alkalmazások számára a legfontosabb grafikus feladatok a függvénykapcsolatok ábrázolásával kapcsolatosak. Írjunk DELPHI alkalmazást az

$$f(x) = e^{-x^2}$$

haranggörbe ábrázolására. Input adatként Edit mezőben adjuk meg a független változó minimális és maximális értékét, valamint a mintavételi közt. Az eredményt numerikusan Memo ablakban adjuk meg, valamint a függvény görbéjét Chart objektumban grafikusan is ábrázolja a program (ld. 22. ábra). A feladat projektállománya legyen *fgv.dpr*, és a felhasznált modul pedig a *unit1.pas*. (A Form és az objektumok tulajdonságait a *unit1.dfm* fájlban találhatjuk meg részletezve). A program tervezésénél először helyezzünk el a Form-on 3 db címkét (Label vezérlő) és szövegmezőt (Edit vezérlő), 2 db gombot (Button vezérlő), egy Memo szövegmezőt és a Chart objektumot. Az ablak tulajdonságait az OBJECT INSPECTOR-on belül megváltoztathatjuk (pl. háttérszín). Ezután definiáljuk az f(x) függvényt egy function-ben, majd írjunk eljárást, mely a „Számol” gomb megnyomásának hatására kiszámítja a függvény értékeit, majd azokat egyenként hozzáadja a Series lista (tömb) első eleméhez (nulladik indexű adatsor). A TMemo típusú *Lista* változó **Lines** tulajdonsága az adatokat karakteresen tartalmazza. Végül jelenítsük meg a képernyőn az eredményeket.



21. ábra. Diagram típusok a Chart menüben



22. ábra. Függvényábrázoló program futás közben

A program listája a következő:

A, Projektállomány:

```
program fgv;

uses
        Forms,
        Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
        Application.Initialize;
        Application.CreateForm(TForm1, Form1);
        Application.Run;
end.
```

B, Unit:

```
unit Unit1;

interface

uses

        Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
        Dialogs, StdCtrls, ExtCtrls, TeeProcs, TeEngine, Chart, Series;

type

        TForm1 = class(TForm)
                Label1: TLabel;
                Label2: TLabel;
                Label3: TLabel;
                Edit1: TEdit;
                Edit2: TEdit;
                Edit3: TEdit;
                Button1: TButton;
                Button2: TButton;
                Chart1: TChart;
                Label4: TLabel;
                Series1: TLineSeries;
                Lista: TMemo;

        procedure Button2Click(Sender: TObject);
        procedure Button1Click(Sender: TObject);

        private
                { Private declarations }
```

```
    public
        { Public declarations }
    end;

var
    Form1: TForm1;

implementation

{$R *.dfm}

    procedure TForm1.Button2Click(Sender: TObject);
        begin
            Application.Terminate;
        end;

    function fgv(var x:extended):extended;
        begin
            fgv:=exp(-sqr(x));
        end;

    procedure TForm1.Button1Click(Sender: TObject);
        var
            i:extended;
        begin
            Lista.Clear;                                    // Memo mező törlése
            i:=strtofloat(Edit1.Text);                      // minimális érték
            chart1.Series[0].Clear;

            while i < strtofloat(Edit2.Text) do
                begin

                   chart1.Series[0].AddXY(i,fgv(i),floattostr(i),clred);
                   Lista.Lines.Add('x= '+floattostr(i)+' ;  y= '+ floattostr(fgv(i)));
                   i:=i+strtofloat(Edit3.Text);

                end;
            Lista.Lines.Add('x= '+floattostr(i)+' ;  y= '+ floattostr(fgv(i)));

        end;
end.
```

C, Dfm file:

```
object Form1: TForm1

    Left = 203
    Top = 115
    BorderStyle = bsDialog
    Caption = 'F'#252'ggv'#233'ny ki'#233'rt'#233'kel'#233'se'
```

```
ClientHeight = 545
ClientWidth = 726
Color = clSkyBlue
Font.Charset = DEFAULT_CHARSET
Font.Color = clWindowText
Font.Height = -11
Font.Name = 'MS Sans Serif'
Font.Style = []
OldCreateOrder = False
PixelsPerInch = 96
TextHeight = 13
object Label1: TLabel
	Left = 24
	Top = 24
	Width = 44
	Height = 13
	Caption = 'Minimum:'
end
object Label2: TLabel
	Left = 24
	Top = 56
	Width = 44
	Height = 13
	Caption = 'Maximum'
end
object Label3: TLabel
	Left = 24
	Top = 88
	Width = 61
	Height = 13
	Caption = 'Inkrementum'
end
object TLabel
	Left = 120
	Top = 152
	Width = 3
	Height = 13
end
object Label4: TLabel
	Left = 88
	Top = 144
	Width = 115
	Height = 13
	Caption = 'F'#252'ggv'#233'ny'#233'rt'#233'kek list'#225'ja:'
	Color = clSkyBlue
	ParentColor = False
end
object Edit1: TEdit
	Left = 120
	Top = 24
```

```
        Width = 121
        Height = 21
        TabOrder = 0
end
object Edit2: TEdit
        Left = 120
        Top = 56
        Width = 121
        Height = 21
        TabOrder = 1
end
object Edit3: TEdit
        Left = 120
        Top = 88
        Width = 121
        Height = 21
        TabOrder = 2
end
object Button1: TButton
        Left = 392
        Top = 40
        Width = 75
        Height = 25
        Caption = 'Sz'#225'mol'
        TabOrder = 3
        OnClick = Button1Click
end
object Button2: TButton
        Left = 544
        Top = 40
        Width = 75
        Height = 25
        Caption = 'Kil'#233'p'#233's'
        TabOrder = 4
        OnClick = Button2Click
end
object Chart1: TChart
        Left = 304
        Top = 104
        Width = 400
        Height = 417
        AllowPanning = pmNone
        BackWall.Brush.Color = clWhite
        BackWall.Brush.Style = bsClear
        LeftWall.Color = clRed
        MarginBottom = 5
        MarginLeft = 5
        MarginRight = 5
        MarginTop = 5
        Title.Font.Charset = DEFAULT_CHARSET
```

```
            Title.Font.Color = clBlack
            Title.Font.Height = -13
            Title.Font.Name = 'Arial'
            Title.Font.Style = []
            Title.Text.Strings = ('F'#252'ggv'#233'ny 2D '#225'br'#225'ja')
            BottomAxis.MinorTickCount = 0
            BottomAxis.MinorTickLength = 0
            BottomAxis.RoundFirstLabel = False
            BottomAxis.Title.Caption = 'x'
            LeftAxis.ExactDateTime = False
            LeftAxis.Increment = 1.000000000000000000
            LeftAxis.MinorTickCount = 0
            LeftAxis.MinorTickLength = 0
            LeftAxis.RoundFirstLabel = False
            LeftAxis.Title.Caption = 'y '
            Legend.Visible = False
            View3D = False
            View3DOptions.Zoom = 93
            TabOrder = 5

            object Series1: TLineSeries
                Marks.ArrowLength = 8
                Marks.Visible = False
                SeriesColor = clRed
                Title = 'Series0'
                Pointer.InflateMargins = True
                Pointer.Style = psRectangle
                Pointer.Visible = False
                XValues.DateTime = False
                XValues.Name = 'X'
                XValues.Multiplier = 1.000000000000000000
                XValues.Order = loAscending
                YValues.DateTime = False
                YValues.Name = 'Y'
                YValues.Multiplier = 1.000000000000000000
                YValues.Order = loNone
            end
        end
        object Lista: TMemo
            Left = 24
            Top = 168
            Width = 249
            Height = 353
            ScrollBars = ssVertical
            TabOrder = 6
        end
end
```